UNIVERSITY of NIŠ
FACULTY OF CIVIL ENGINEERING AND ARCHITECTURE

NUMERICAL METHODS
In Computational Engineering

G.V. MILOVANOVIĆ, sci. advisor
&
Đ. R. ĐORĐEVIĆ, lecturer

financed by
Austrian
Development Cooperation

wusaustria

Niš, 2007

UNIVERSITY of NIŠ
FACULTY OF CIVIL ENGINEERING AND ARCHITECTURE

NUMERICAL METHODS in Computational Engineering

G.V. MILOVANOVIĆ, sci. advisor
(http://gauss.elfak.ni.ac.yu/)

Đ. R. ĐORĐEVIĆ, lecturer
(http://www.gaf.ni.ac.yu/cdp/lecturer.htm)

Niš, 2007.

Prof. Gradimir V. Milovanović, University Niš
Prof. Đorđe R. Đorđević, University Niš

Numerical Methods in Computational Engineering

ISBN 978 86-80295-81-7

The publishing of this script is part of the project CDP+ 53/2006 financed by Austrian Cooperation through WUS Austria

This copy is not for sale

Ova skripta je objavljena u okviru projekta WUS Austria CDP+ 53/2006 finansiranog od strane Austrian Cooperation

Besplatan primerak

Printed by Grafika Petković, Niš, 2007 (100 copies)

Contents

Preface

A course in Numerical Methods in Computational Engineering, oriented to engineering education, originates at first from the course in numerical analysis for graduate students of Faculty of Civil Engineering and Architecture of Niš (GAF), and then from course Numerical Methods held in English language at Faculty of Civil Engineering in Belgrade in the frame of project DYNET (Dynamical Network) in common of Faculty of Civil Engineering of University of Bochum, Faculty of Civil Engineering and Architecture of University of Niš, Faculty of Civil Engineering of University Belgrade, and IZIIS (Institute for Earthquake Engineering and Seismology) of University Skopje. The subject Numerical Analysis was held in the first semester of postgraduate studies at GAF by Prof. G.V. Milovanović for years. In continuation, following Bologna process, the new structured subject entitled Numerical Analysis is to be introduced to PhD students at GAF. In addition, having in mind that course in numerical analysis become accepted as an important ingredient in the undergraduate education in engineering and technology, it was with its main topics involved in undergraduate subject Informatics II at GAF Niš (As a collateral case, in Appendix A.4. – in electronic form – are given numerical methods in Informatics, what could be interesting for students of this orientation).

The backbone of this script are famous books of G.V. Milovanović, Numerical Analysis, Part I, II, and III, Naučna knjiga, Beograd, 1988 (Serbian). In addition, the book Programming Numerical Methods in Fortran, by G.V. Milovanović and Dj. R. Djordjević, University of Niš, 1981 (Serbian), with its engineering-oriented text and codes, was rather used.

As previously noted, this textbook is supporting undergraduate studies, master and doctoral study at GAF, and international master study in the frame of DYNET project. Presentation on GAF site would enable distance learning technique and on-line consultations with lecturer. By up-to-day engineering oriented applications the supporting of life long education of civil engineers will be enabled.

This script will be available on the site of GAF (`http://www.gaf.ni.ac.yu`*) under* `International Projects` *and can be reached by chapters using address* `http://www.gaf.ni.ac.yu/cdp/subject_syllabus.htm`*. Each chapter concludes with a basic bibliography and suggested further reading. Tutorial exercises in form of selected assignments are also presented on the site of GAF. Some hints for solutions are given in the same files.*

Devoted primarily to students of Civil Engineering (undergraduate and graduate - master & PhD), this textbook is dedicated also to industry and research purposes.

Authors

LECTURES

LESSON I

1. Mathematics and Computer Science

1.1 Calculus

The principal topics in calculus are the real and complex number systems, the concept of limits and convergence, and the properties of functions.

Convergence of a sequence of numbers x_i is defined as follows:

The sequence x_i converges to the limit x^* if, given any tolerance $\varepsilon > 0$, there is an index $N = N(\varepsilon)$ so that for all $i \geq N$ we have $|x_i - x^*| \leq \varepsilon$. The notation for this is

$$\lim_{i\to\infty} x_i = x^*.$$

Convergence is also a principal topics of numerical computation, but with a different emphasis. In calculus one studies limits and convergence with analytic tools; one tries to obtain the limit or to show that convergence takes place. In computations, one has the same problem but little or no theoretical knowledge about the sequence. One is frequently reduced to using empirical intuitive tests for convergence; often the principal task is to actually estimate the value of the tolerance for a given x.

The study of functions in calculus revolves about continuity, derivatives, and integrals. A function $f(x)$ is continuous if

$$\lim_{x_i\to x^*} f(x_i) = f(x^*)$$

holds for all x^* and all ways for the x_i to converge to x^*. We list six theorems from calculus which are useful for estimating values that appear in numerical computation.

Theorem 1 (Mean value theorem for continuous functions). *Let $f(x)$ be continuous on the interval $[a, b]$. Consider points XHI and XLOW in $[a, b]$ and a value y so that $f(XLOW) \leq y \leq f(XHI)$.Then there is a point ρ in $[a, b]$ so that*

$$f(\rho) = y.$$

Theorem 2 (Mean value theorem for sums). *Let $f(x)$ be continuous on the interval $[a, b]$, let $x_1, x_2, \ldots, x_n$ be points in $[a, b]$ and let $w_1, w_2, \ldots, w_n$ be positive numbers. Then there is a point ρ in $[a, b]$ so that*

$$\sum_{i=1}^{n} w_i(x) f(x_i) = f(\rho) \sum_{i=1}^{n} w_i.$$

Theorem 3 (Mean value theorem for integrals). *Let $f(x)$ be continuous on the interval $[a, b]$ and let $w(x)$ be a nonnegative function $[w(x) \geq 0]$ on $[a, b]$. Then there is a point ρ in $[a, b]$ so that*

$$\int_a^b w(x) f(x) dx = f(\rho) \int_a^b w(x) dx.$$

Theorems 2 and 3 show the analogy that exists between sums and integrals. This fact derives from the definition of the integral as

$$\int_a^b f(x)dx = \lim_{\max|x_{i+1}-x_i|\to 0} \sum_i f(x_i)(x_{i+1}-x_i),$$

where the points x_i with $x_i < x_{i+1}$ are a partition of $[a,b]$. This analogy shows up for many numerical methods where one variation applies to sums and another applies to integrals. Theorem 2 is proved from Theorem 1, and then Theorem 3 is proved by a similar method. The assumption that $w(x) \geq 0$ ($w_i > 0$) may be replaced by $w(x) \leq 0$ ($w_i < 0$) in these theorems; it is essential that $w(x)$ be on one sign shown by the example $w(x) = f(x) = x$ and $[a,b] = [-1,1]$.

Theorem 4 (Continuous functions assume max/min values). *Let $f(x)$ be continuous on the interval $[a,b]$ with $|a|, |b| \leq \infty$. Then there are points XHI and XLOW in $[a,b]$ so that for all x in $[a,b]$*

$$f(XHI) \leq f(x) \leq f(XLOW).$$

The **derivative** *of $f(x)$ is defined by*

$$\frac{df}{dx} = f'(x) = \lim_{h\to 0} \frac{f(x+h)-f(x)}{h}.$$

As an illustration of the difference between theory and practice, the quantity $[f(x+h)-f(x)]/h$ can be replaced by $f[(x+h)-f(x-h)]/(2h)$ with no change in the theory but with dramatic improvement in the rate of convergence; that is, much more accurate estimates of $f'(x)$ are obtained for a given value of h. The $k-th$ derivative is the derivative of the $(k-1)th$ derivative; they are denoted by d^kf/dx^k or $f''(x), f'''(x), f^{(4)}(x), f^{(5)}(x), \ldots$

Theorem 5 (Mean value theorem for derivatives). *Let $f(x)$ be continuous and differentiable in $[a,b]$, with $|a|, |b| < \infty$. Then there is a point ρ in $[a,b]$ so that*

$$\frac{f(b)-f(a)}{b-a} = f'(\rho)$$

$$f(x) = f(c) + f'(\rho)(x-c)$$

The special case of Theorem 5 with $f(a) = f(b) = 0$ is known as Rolle's theorem. It states that if $f(a) = f(b) = 0$, then there is a point ρ between a and b so that $f'(\rho) = 0$. This is derived from Theorem 5 by multiplying through by $b-a$, renaming a, b as x, c, and then applying the first form to the smaller interval $[x,c]$ or $[c,x]$, depending on the relation between x and c.

A very important tool in numerical analysis is the extension of the second part of Theorem 5 to use higher derivatives.

Theorem 6 (Tailor series with remainder). *Let $f(x)$ have $n+1$ continuous derivatives in $[a,b]$.*

Given points x and c in $[a,b]$ we have

$$f(x) = f(c) + f'(c)(x-c) + f''(c)\frac{(x-c)^2}{2!} + f'''\frac{(x-c)^3}{3!} + \cdots + f^{(n)}(c)\frac{(x-c)^n}{n!}$$

$$+R_{n+1}(x).$$

where R_{n+1} has either one of the following forms (ρ is a point between x and c):

$$R_{n+1}(x) = f^{(n+1)}(\rho)\frac{(x-c)^{n+1}}{(n+1)!}$$

$$R_{n+1}(x) = \frac{1}{n!}\int_c^x (x-t)^n f^{(n+1)}(t)dt$$

If a function f depends on several variables, one can differentiate it with respect to one variable, say x, while keeping all the rest fixed. This is a **partial derivative** of f and it is denoted by $\partial f/\partial x$ or f_x. Higher order and mixed derivatives are defined by successive differentiation. Taylor's series for functions of several variables is a direct extension of the formula in Theorem 6, although the number of terms in it grows rapidly. For two variables it is

$$f(x,y) = f(c,d) + f_x(x-c) + f_y(y-d) + \frac{1}{2}[f_{xx}(x-c)^2 + 2f_{xy}(x-c)(y-d)$$

$$+f_{yy}(y-d)^2] + \cdots,$$

where all the partial derivatives are evaluated at the point (c,d).

Theorem 7 (Chain rule for derivatives). *Let $f(x,y,\ldots,z)$ have continuous first partial derivatives with respect to all its variables. Let $x = x(t), y = y(t), \ldots, z = z(t)$ be continuous differentiable functions of t. Then*

$$g(t) = f(x(t), y(t), \ldots, z(t))$$

is continuously differentiable and

$$g'(t) = f_x x'(t) + f_y y'(t) + \cdots + f_z z'(t).$$

Finally, we state

Theorem 8 (Fundamental theorem of algebra). *Let $p(x)$ be a polynomial of degree $n \geq 1$, that is,*

$$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n,$$

where the a_i are real or complex numbers and $a_n \neq 0$. Then, there is a complex number ρ so that $p(\rho) = 0$.

1.2. Number representation

Numbers are represented in number systems. Any number of bases can be employed as the base of a number system, for example, the base 10 (decimal), 8 (octal), 12 (duodecimal), 16 (hexadecimal), or the base 2, (binary) system. The base 10, i.e. decimal system is the most common system used in human communication. In spite of not being optimal (optimal would be theoretical system with base e, base of natural logarithm, or technical system with base 3, trinary system), digital computers use, due to electronic technology, system with base 2, or binary system. In a digital computer, a binary number consists of a number of binary bits. The number of binary bits in a binary number determines the precision with which the binary number represents a decimal number. The most common size of binary number is a 32-bit number (we say, the machine word is 32 bits long, what defines the "32-bits word computer"), what can represent approximately 7 digits of a decimal number. Some computer have 64 bits binary numbers, i.e. 64 bits machine word length, which can represent 13 to 14 decimal digits. For many engineering and scientific calculations, 32 bit arithmetic is good enough. But, for many other applications, 64 bit arithmetic is required. Higher precision (i.e. 64 bit, or even 128 bit) can be reached by software means, using Double precision or Quad precision, respectively. Of course, such software enhancement must be payed by even 10 times execution times of single precision calculation.

As already told, computers store numbers not with infinite precision but rather in some approximation that can be packed into a fixed number of bits (binary digits) or bytes (groups of 8 bits). Almost all computers allow the programmer a choice among several different such representations or data types. Data types can differ in the number of bits utilized (the word-length), but also in the more fundamental respect of whether the stored number is represented in fixed-point (also called integer) or floating-point (also called real) format. A number in integer representation is exact. Arithmetic between numbers in integer representation is also exact, with the provisos that

(a) the answer is not outside the range of (usually signed) integers that can be represented, and
(b) division is interpreted as producing an integer result, throwing away any integer remainder.

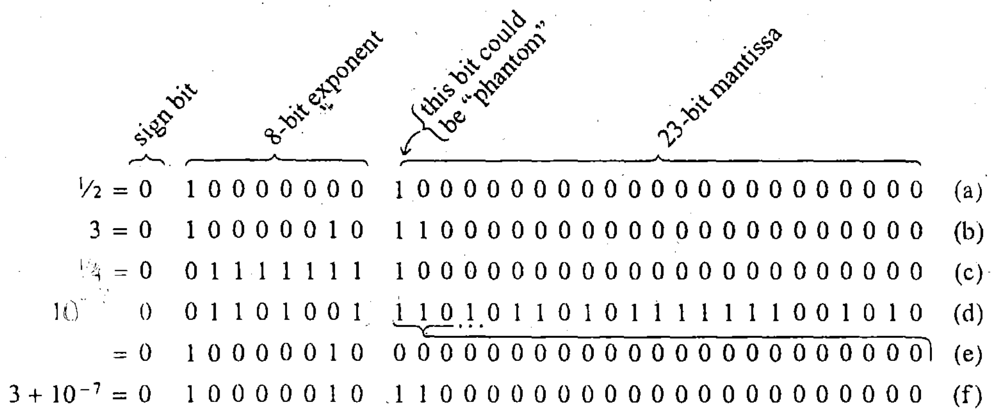

sign bit
8-bit exponent
this bit could be "phantom"
23-bit mantissa
½ = 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 (a)
3 = 0 1 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 (b)
¼ = 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 (c)
10⁻⁷ = 0 0 1 1 0 1 0 0 1 1 1 0 1 0 1 1 0 1 0 1 1 1 1 1 1 1 0 0 1 0 1 0 (d)
= 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 (e)
3 + 10⁻⁷ = 0 1 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 (f)

Figure 1.2.1.

In Fig. 1.2.1. are given floating point representations of numbers in a typical 32-bit (4-byte) format, with the following examples:

(a) The number 1/2 (note the bias in the exponent);
(b) the number 3;
(c) the number 1/4;
(d) the number 10^{-7}, represented to machine accuracy;
(e) the same number 10^{-7}, but shifted so as to have the same exponent as the number 3; with this shifting, all significance is lost and 10^{-7} becomes zero; shifting to a common exponent must occur before two numbers can be added;
(f) sum of the numbers $3 + 10^{-7}$, which equals 3 to machine accuracy. Even though 10^{-7} can be represented accurately by itself, it cannot accurately be added to a much larger number.

In floating-point representation, a number is represented internally by a sign bit s (interpreted as plus or minus), an exact integer exponent e, and an exact positive integer mantissa M. Taken together these represent the number

$$s \times M \times B^{e-E} \tag{1.2.1}$$

where B is the base of the representation (usually $B = 2$, but sometimes $B = 16$), and E is the bias of the exponent, a fixed integer constant for any given machine and representation.

Several floating-point bit patterns can represent the same number. If $B = 2$, for example, a mantissa with leading (high-order) zero bits can be left-shifted, i.e., multiplied by a power of 2, if the exponent is decreased by a compensating amount.

Bit patterns that are "as left-shifted as they can be" are termed normalized. Most computers always produce normalized results, since these do not waste any bits of the mantissa and thus allow a greater accuracy of the representation. Since the high-order bit of a properly normalized mantissa (when $B = 2$) is always one, some computers do not store this bit at all, giving one extra bit of significance. Arithmetic among numbers in floating-point representation is not exact, even if the operands happen to be exactly represented (i.e., have exact values in the form of equation (1.2.1). For example, two floating numbers are added by first right-shifting (dividing by two) the mantissa of the smaller (in magnitude) one, simultaneously increasing its exponent, until the two operands have the same exponent. Low-order (least significant) bits of the smaller operand are lost by this shifting. If the two operands differ too greatly in magnitude, then the smaller operand is effectively replaced by zero, since it is right-shifted to oblivion. The smallest (in magnitude) floating-point number which, when added to the floating-point number 1.0, produces a floating-point result different from 1.0 is termed the machine accuracy m. A typical computer with $B = 2$ and a 32-bit word-length has m around 3×10^{-8}. Generally speaking, the machine accuracy m is the fractional accuracy to which floating-point numbers are represented, corresponding to a change of one in the least significant bit of the mantissa.

1.3. Error, accuracy, and stability

Except for integers and some fractions, all binary representations of decimal numbers are approximations, owing to the finite word length of binary numbers.Thus, some loss of precision in the binary representation of decimal number is unavoidable. Result of arithmetic operation among binary numbers is typically a longer binary number which cannot be represented with the number of available bits of the digital computer. Thus, the results are rounded off in the last available binary bit. This rounding-off is called round-off error. Well, pretty much any arithmetic operation among floating numbers should be thought of as introducing an additional fractional error of at least ε_m, called roundoff error. It is important to understand that ε_m is not the smallest floating-point number that can be represented on a machine. That number depends on how many bits there are in the exponent, while ε_m depends on how many bits there are in the mantissa. Roundoff errors accumulate with increasing amounts of calculation. If, in the course of obtaining a calculated value, one performs n such arithmetic operations, he might be satisfied with a total roundoff error on the order of $\sqrt{n}\varepsilon_m$, if the roundoff errors come in randomly up or down. (The square root comes from a random-walk.) However, this estimate can be very badly off the mark for two reasons:

(i) It very frequently happens that the regularities of calculation, or the peculiarities of computer, cause the roundoff errors to accumulate preferentially in one direction. In this case the total will be of order $n\varepsilon_m$.

(ii) Some especially unfavorable occurrences can vastly increase the roundoff error of single operations. Generally these can be traced to the subtraction of two very nearly equal numbers, giving a result whose only significant bits are those (few) low-order ones in which the operands differed. You might think that such a "coincidental" subtraction is unlikely to occur, what is not always true. Some mathematical expressions magnify its probability of occurrence tremendously. For example, in the familiar formula for the solution of a quadratic equation,

$$x = \frac{-b + \sqrt{b^2 - 4ac}}{2a}, \tag{1.3.1}$$

when $ac << b^2$ the addition becomes critical and round-off could ruin the calculation (see section 1.6.).

Roundoff error is a characteristic of computer hardware. There is another, different, kind of error that is a characteristic of the program or algorithm used, independent of the hardware on which the program is executed. Many numerical algorithms compute "discrete" approximations to some desired "continuous" quantity. For example, an integral is evaluated numerically by computing a function at a discrete set of points, rather than at "every" point. Or, a function may be evaluated by summing a finite number of leading terms in its infinite series, rather than all infinity terms. In cases like this, there is an adjustable parameter, e.g., the number of points or of terms, such that the "true" answer is obtained only when that parameter goes to infinity. Any practical calculation is done with a finite, but sufficiently large, choice of that parameter.

The discrepancy between the true answer and the answer obtained in a practical calculation is called the truncation error. Truncation error would persist even on a hypothetical, "perfect" computer that had an infinitely accurate representation and no roundoff error. As a general rule there is not much that a programmer can do about roundoff error, other than to choose algorithms that do not magnify it unnecessarily. Truncation error, on the other hand, is entirely under the programmers control. In fact, it is only a slight exaggeration to say that clever minimization of truncation error is practically the entire content of the field of numerical analysis.

Most of the time, truncation error and roundoff error do not strongly interact with one another. A calculation can be imagined as having, first, the truncation error that it would have if run on an infinite-precision computer, and in addition, the roundoff error associated with the number of operations performed.

Some computations are very sensitive to round-off and others are not. In some problems sensitivity to round-off can be eliminated by changing the formula or method. This is always possible; there are many problems which are inherently sensitive to round-off and any other uncertainties. Thus we must distinguish between sensitivity of *methods* and sensitivity inherent in *problems*.

The word *stability* appears during numerical computations and refers to continuous dependence of a solution on the *data* of the problem or *method*. If one says that a method is *numerically unstable*, one means that the round-off effects are grossly magnified by the method. Stability also has precise technical meaning (not always the same) in different areas as well as in this general one.

Solving differential equations usually leads to difference equations, like

$$x_{i+2} = -(13/6)x_{i+1} + (5/2)x_i.$$

Here, the sequence $x_1, x_2, \ldots$ is defined, and for given initial conditions x_1 and x_2 of differential equation, we get the initial conditions for difference equation. For example, $x_1 = 30$, $x_2 = 25$. Computing in succession for $4, 8, 16, 32, 64$ decimal digits gives the results that can be compared with the exact one, $x_i = 36/(5/6)^i$. (Compute in `Mathematica`, using $N[x[I+2], k]$, where $k = 4, 8, 16, 32, 64$ number of decimal digits).

i	4	8	16	True value
1	30.00	30.00	30.00	30.00
2	25.00	25.00	25.00	25.00
3	20.83	20.8333	20.8333	20.8333
4	17.36	17.3611	17.3611	17.3611
5	14.46	14.4676	14.4676	14.4676
6	12.07	12.0563	12.0563	12.0563
7	10.00	10.0470	10.0469	10.0469
8	8.518	8.3724	8.3724	8.3724
9	6.541	6.9773	6.9770	6.9770
10	7.121	5.8133	5.8142	5.8142
11	.925	4.8478	4.8452	4.8452
12	15.790	4.0296	4.0376	4.0376
13	-31.920	3.3888	3.3647	3.3647
14	108.700	2.7318	2.8039	2.8039
16	954.600	1.2978	1.9472	1.9472
18	8576.000	-4.4918	1.3522	1.3522
20	77170.000	-51.6565	.9390	.9390
22	6.9×10^5	-472.7080	.6521	.6521
25	-1.8×10^7	12781.1000	.3776	.3774
28	5.0×10^8	-345079.0000	.2134	.2184
30	4.5×10^9	-3.1×10^6	.1071	.1517

35	-1.1×10^{12}	7.5×10^{8}	10.8822	.0609
40	-1.1×10^{14}	-1.8×10^{11}	−2629.5300	.0245
50	1.5×10^{19}	-1.0×10^{16}	-1.5×10^{8}	.0039
75	1.3×10^{31}	9.2×10^{27}	1.3×10^{20}	.00

This difference equation is unstable and one can see that the computation quickly "blows up". One nice thing about unstable computation is that they usually produce huge, nonsense numbers that one is not tempted to accept as correct. However, imagine that one wanted only 30 terms of the x_i and was using the computer with 16 decimal digits. How would one know that the last term is in error by 50 percent ?

The word *condition* is used to describe the sensitivity of problems to uncertainty. Imagine the solution of a problem being obtained by evaluation a function $f(x)$. Then, if x is changed a little to $x + \delta x$, the value $f(x)$ also changes. The relative *condition number* of this change is

$$\frac{|f(x+\delta x) - f(x)|}{|f(x)|} / |\frac{\delta x}{x}|,$$

or

$$\frac{f(x+\delta x) - f(x)}{\delta x} \times \frac{x}{f(x)},$$

and, for δx very small, condition number c is

$$c \sim \frac{x f'(x)}{f(x)}.$$

This number estimates how much an uncertainty in the data x of a problem is magnified in its solution $f(x)$. If this number is large, then the problem is said to be *ill-conditioned* or poorly conditioned.

The given formula is for the simplest case of a function of a single variable; it is not easy to obtain such formulas for more complex problems that depend on many variables of different types. We can see three different ways that a problem can have a large condition number:

1. *$f'(x)$ may be large while x and $f(x)$ are not*;

If we evaluate $1 + \sqrt{|x - 1|}$ for x very close to 1, then x and $f(x)$ are nearly 1, but $f'(x)$ is large and the computed value is highly sensitive to change in x.

2. *$f(x)$ may be small while x and $f'(x)$ are not*;

The Taylor's series for $\sin x$ near π or e^{-x} with x large exhibit this form of ill conditioning.

3. *x may be large while $f'(x)$ and $f(x)$ are not*;

The evaluation of $\sin x$ for x near 1000000π is poorly conditioned.

One can also say that computation is ill-conditioned and this is the same as saying it is numerically unstable. The condition number gives more information than just saying something is numerically unstable. It is rarely possible to obtain accurate values for condition numbers but one rarely needs much accuracy; an order of magnitude is often enough to know.

Note that is almost impossible for a method to be numerically stable for an ill-conditioned problem.

Example 1.3.1. An **ill-conditioned line intersection problem** consists in computing the point of intersection P of two nearly parallel lines. It is clear that a minor change in one line changes the point of intersection to $(P + \delta P)$ which is far from P. A mathematical model of this problem is obtained by introducing a coordinate system and writing equations

$$y = a_1 x + b_1$$
$$y = a_2 x + b_2$$

what leads to solving a system of equations

$$a_1x - y = -b_1$$
$$a_2x - y = -b_2$$

with the a_1 and a_2 nearly equal since the lines are nearly parallel. This numerical problem is unstable or ill-conditioned, as it reflects the ill-conditioning of the original problem.

A mathematical model is obtained by introducing a **coordinate system**. Any two vectors will do for a basis, and if we chose to use the unusual basis

$$\mathbf{b_1} = (0.5703958095, 0.8213701274)$$
$$\mathbf{b_2} = (0.5703955766, 0.8213701274)$$

then every vector $\mathbf{x}$ can be expressed as

$$\mathbf{x} = x\mathbf{b_1} + y\mathbf{b_2}$$

so that the equations of the two lines in this coordinate system are

$$y = -0.0000000513 + 0.9999998843x$$
$$y = -0.0000045753 + 1.000001596x$$

with the point of intersection P with coordinates $(-0.8903429339, 0.8903427796)$. Note that mathematical model is very ill-conditioned; a change of 0.0000017117 in the data makes the two lines parallel, with no solution.

The poor choice of a basis in the given example made the problem poorly conditioned. In more complex problems it is not so easy to see that a poor choice has been made. In fact, a poor choice is sometimes the most natural thing to do. For example, in problems involving the polynomials, one naturally takes vectors based on $1, x, x^2, \ldots, x^n$ as a basis, but these are terribly ill-conditioned even for n moderate in size.

Example 1.3.2. System of equations (input information)

$$2x + 6y = 8$$
$$2x + 6.0001y = 8.0001$$

have a solutions (output information) $x = 1$, $y = 1$. If the coefficients of second equation slightly change, i.e. if one takes the equation

$$2x + 5.99999y = 8.00002,$$

the solutions are $x = 10$, $y = -2$. This is typical round-off error.

Errors in methods occur usually because in numerical mathematics the problem to be solved is replaced by another one, closed to original, which is easier to solve.

Example 1.3.3. Integral $\int_a^b f(x)dx$ can be approximately calculated, for example, by replacing the function f by some polynomial P on segment $[a, b]$, which is in some sense close to given function. However, for approximative calculation it is possible to use the sum

$$\sum_{i=1}^{n} f(x_i)\Delta x_i.$$

In both cases the method error occurs.

In some sense, the round-off error are also method errors. Sum of all errors makes the total error.

Sometimes, however, an otherwise attractive method can be unstable. This means that any roundoff error that becomes "mixed into" the calculation at an early stage is successively magnified until it comes to swamp the true answer. An unstable method would be useful on a hypothetical, perfect computer; but in this imperfect world it is necessary for us to require that algorithms be stable or if unstable that we use them with great caution. Here is a simple, if somewhat artificial, example of an unstable algorithm (see [4], p.20).

Example 1.3.4. Suppose that it is desired to calculate all integer powers of the so-called "Golden Mean," the number given by

$$\Phi \equiv \frac{\sqrt{5}-1}{2} \approx 0.61803398 \tag{1.3.2}$$

Powers of Φ^n satisfy simple recurrence relation

$$\Phi^{n+1} = \Phi^{n-1} - \Phi^n . \tag{1.3.3}$$

Well, knowing the first two values $\Phi^0 = 1$ and $\Phi^1 = 0.61803398$, we can apply (1.3.3) by subtraction, rather than a slower multiplication by Φ, at each stage. Unfortunately, the recurrence (1.3.3) also has another solution, namely the value $-\frac{1}{2}(\sqrt{5}+1)$. Since the recurrence is linear, and since this undesired solution has magnitude greater than unity, any small admixture of it introduced by roundoff errors will grow exponentially. On a typical machine with 32-bit word-length, (1.3.3) starts to give completely wrong answers by about $n = 16$, at which point Φ^n is down to only 10^{-4}. Thus, the recurrence (1.3.3) is unstable, and cannot be used for the purpose stated.

On the end of this section, it remains the question: How to estimate errors and uncertainty ?

One almost newer knows the error in a computed result unless one already knows the true solution, and so one must settle for estimates of the error. There are three basic approaches to error estimates. The first is *forward error analysis*, when one uses the theory of the numerical method plus information about the uncertainty in the problem and attempts to predict the error in the computed result. The information one might use includes

- the size of round-off,
- the measurement errors in problem data,
- the truncation errors in obtaining the numerical model from the mathematical model,
- the differences between the mathematical model and the original physical model.

The second approach is *backward error analysis*, where one takes a computed solution and sees how close it comes to solving the original problem. The backward error is often called the the *residual* in equations. This approach requires that the problems involve satisfying some conditions (such as an equation) which can be tested with a trial solution. This prevents it from being applicable to all numerical computations, e.g. numerically estimating the value of π or the value of an integral.

The third approach is *experimental error analysis*, where one experiments with changing the computations, the method, or the data to see the effect they have on the results. If one truly wants certainty about the accuracy of a computed value, then one should give the problem to two (or even more) different groups and ask to solve it. The groups are not allowed to talk together, preventing a wrong idea from being passing around.

The relationship between these three approaches could be illustrated graphically, as given in the following figure.

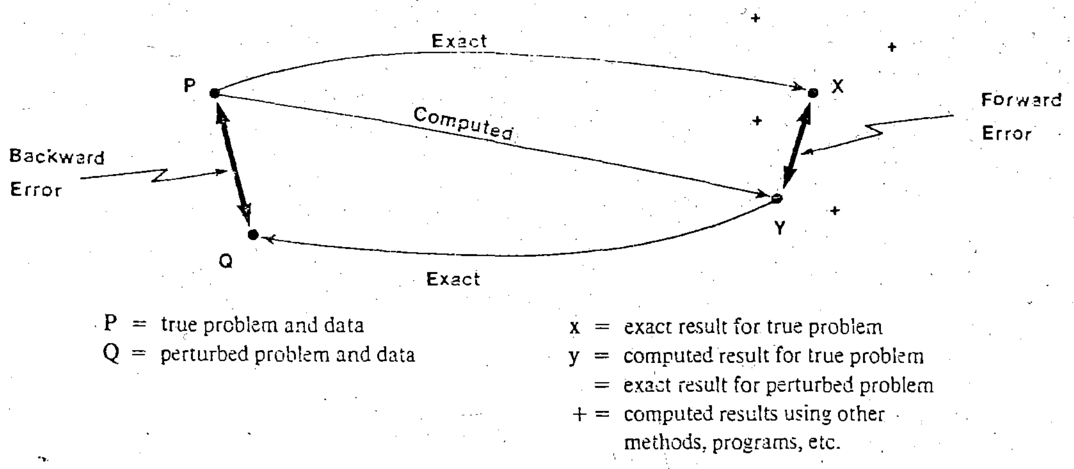

Exact
P
X
Computed
Forward
Error
Backward
Error
Y
Q
Exact
P = true problem and data
Q = perturbed problem and data
x = exact result for true problem
y = computed result for true problem
= exact result for perturbed problem
+ = computed results using other methods, programs, etc.

Figure 1.3.1

1.4. Programming

There are several areas of knowledge about programming that are needed for scientific computation. These include knowledge about:

- The programming language (FORTRAN, Pascal, C, Java, Mathematica (MatCAD, Matlab).
- The computer system in which the language runs
- Program debugging and verifying the correctness of results
- Computation organization and expressing them clearly.

Debugging programs is an art as well as a science, and it must be learned through practice. There are several effective tactics to use, like:
- Intermediate output
- Consultations about program with experienced user
- Use compiler and debugging tools.

Some abilities of compilers:
- Cross-reference tables
- Tracing
- Subscript checking
- Language standards checking.

Some hints:
- Use lots of comments
- Use meaningful names for variables
- Make the types of variables obvious
- Use simple logical control structures
- Use program packages and systems (Mathematica, Matlab) wherever possible
- Use structured programming
- Use (if possible) OOP technics for technical problems.

1.5. Numerical software

There are several journals that publish individual computer programs:

- ACM Transactions on Mathematical Software (IMSL, International Mathematical Scientific Library)

- Applied Statistics
- BIT
- The Computer Journal
- Numerische Mathematik

The ACM Algorithms series contains more than thousand items and is available as the **Collected Algorithms of the Association for Computing Machinery**.

Three general libraries of programs for numerical computations are widely available:

IMSL	International Mathematical Scientific Library
NAG	Numerical Algorithms Group, Oxford University
SSP	Scientific Subroutine Package, IBM Corporation

There are a substantial number of important, specialized software packages. Most of the packages listed below are available from IMSL, Inc.

MP	Multiple Precision Arithmetic Package
BLAS	Basic Linear Algebra Subroutines
DEPACK	Differential Equation Package
DSS	Differential System Simulator
EISPACK	Matrix Eigensystems Routines
FISHPACK	Routines for the Helmholtz Problem in Two or Three Dimensions
FUNPACK	Special Function Subroutines
ITPACK	Iterative Methods
LINPACK	Linear Algebra Package
PPPACK	Piecewise Polynomial and Spline Routines
ROSEPACK	Robust Statistics Package
ELLPACK	Elliptic Partial Differential Equations
SPSS	Statistical Package for the Social Sciences.

User interface to the IMSL library:

PROTRAN	John R. Rice, Purdue University

1.6. Case study: Errors, round-off, and stability

Example 1.6.1. Solve quadratic formula

$$ax^2 + bx + c = 0$$

with $5, 10, 15, \ldots 100$ decimal digits using `FORTRAN` and `Mathematica` code. Take $a = 1$, $c = 2, b = 5.2123(10)105.2123$. Use the following two codes:

```
DIS=SQRT(B*B-4.*A*C) DIS=SQRT(B*B-4.*A*C)
X1=(-B+DIS)/(2*A)    IF(B.LT.0) THEN
X2=(-B-DIS)/(2*A)    X1=(-B+DIS)/(2*A)
                     ELSE
                     X1=(-B-DIS)/(2*A)
                     ENDIF
                     X2=C/X1
```

Compare the obtained results.

There are two important lessons to be learned from example 1.6.1.:

1. *Round-off error can completely ruin a short, simple computation.*
2. *A simple change in the method might eliminate adverse round-off effects.*

Example 1.6.2. Calculation of π.

Using five following algorithms, calculate π in order to illustrate the various effects of round-off on somewhat different computations.

Algorithm 1.6.2.1. Infinite alternate series

$$\pi = 4(1 - 1/3 + 1/5 - 1/7 + 1/9 - \cdots)$$

Algorithm 1.6.2.2. Taylor's series of $\arcsin(1/2) = \pi/6$

$$\pi = 6(0.5 + \frac{(0.5)^2}{2 \times 3} + \frac{1 \times 3(0.5)^4}{2 \times 4 \times 5} + \frac{1 \times 3 \times 5(0.5)^6}{2 \times 4 \times 6 \times 7} + \cdots)$$

Algorithm 1.6.2.3. Archimedes' method. Place $4, 8, 16, \ldots, 2^n$ triangles inside a circle. The area od each triangle is $1/2\sin(\theta)$. The values of $\sin(\theta)$ are computed by the half angle formula

$$\sin(\theta) = \sqrt{[1 - \cos(2\theta)]/2}$$

and

$$\cos(\theta) = \sqrt{1 - \sin^2\theta}.$$

The calculation is initialized by $\sin(\pi/4) = \cos(\pi/4) = 1/\sqrt{2}$. As the number of triangles grows, they fill up the circle and their total area approaches π. (Archimed carried a similar procedure by hand with 96 triangles and obtained

$$3.1409\ldots = 3\frac{1137}{8069} < \pi < 3\frac{1335}{9347} = 3.1428\ldots)$$

Algorithm 1.6.2.4. Instead of inscribing triangles in a circle, we inscribe trapezoids in a quarter circle. As a number of trapezoids increases, the sum of their areas approaches $\pi/4$.

Algorithm 1.6.2.5. Monte Carlo integration.

Monte Carlo integration for $\int_0^2 \frac{2}{1+x}\,dx$ is proceeded by choosing a pair (x, y) at random with x, y in $[0, 2]$, and comparing y with $2/(1 + x)$. If $y \leq 2/(1 + x)$ then the point (x, y) is under the curve $y = 2/(1 + x)$ and variable SUM is increased by 1. After `M` pairs, the integral is estimated by the fraction `SUM/M` of points that are under the curve. Use this algorithm to estimate the area of the quarter circle.

Bibliography (Cited references and further reading)

[1] Milovanović, G.V., *Numerical Analysis I*. Naučna knjiga, Beograd, 1988 (Serbian).

[2] Milovanović, G.V. and Djordjević, Dj.R., *Programiranje numeričkih metoda na FORTRAN jeziku*. Institut za dokumentaciju zaštite na radu "Edvard Kardelj", Niš, 1981 (Serbian).

[3] Hoffman, J.D., *Numerical Methods for Engineers and Scientists*. Taylor & Francis, Boca Raton-London-New York-Singapore, 2001.

[4] Press, W.H., Flannery, B.P., Teukolsky, S.A., and Vetterling, W.T., *Numerical Recepies - The Art of Scientific Computing*. Cambridge University Press, 1989.

[5] Rice, J.R., *Numerical Methods, Software, and Analysis*. McGraw-Hill, New York, 1983.

[6] Abramowitz, M., and Stegun, I.A., *Handbook of Mathematical Functions*. National Bureau of Standards, Applied Mathematics Series, Washington, 1964 (reprinted 1968 by Dover Publications, New York).

[7] Hildebrand, F.B., *Introduction to Numerical Analysis*. Mc.Graw-Hill, New York, 1974.

[8] Stoer, J., and Bulirsch, R., *Introduction to Numerical Analysis.* Springer-Verlag, New York, 1980.

[9] Kahaner, D., Moler, C., and Nash, S., *Numerical Methods and Software.*, Prentice Hall, Englewood Cliffs, 1989.

[10] Johnson, L.W., and Riess, R.D., *Numerical Analysis.* 2nd ed., Addison- Wesley, Reading, 1982.

[11] Wilkinson, J.H., *Rounding Errors in Algebraic Processes.* Prentice-Hall, Englewood Cliffs, 1964.

[12] Milovanović, G.V. and Kovačević, M.A., *Zbirka rešenih zadataka iz numeričke analize.* Naučna knjiga, Beograd, 1985. (Serbian).

[13] Forsythe, G.E., Malcolm, M.A., and Moler, C.B., *Computer Methods for Mathematical Computations.* Englewood Cliffs, Prentice-Hall, NJ, 1977.

[14] *IMSL Math/Library Users Manual* , IMSL Inc., 2500 City West Boulevard, Houston TX 77042.

[15] *NAG Fortran Library*, Numerical Algorithms Group, 256 Banbury Road, Oxford OX27DE, U.K.

LECTURES

LESSON II

2. Linear Systems of Algebraic Equations: Direct Methods

2.1. ELEMENTS OF MATRIX CALCULUS

2.1.1. LR factorization of quadratic matrix

During solution of systems of linear equation there is often case to present a quadratic matrix in a form of product of two triangular matrices. This section is devoted to this problem.

Theorem 2.1.1.1. *If all determinants of form*

$$\Delta_k = \begin{vmatrix} a_{11} & \cdots & a_{1k} \\ \vdots & & \\ a_{k1} & \cdots & a_{kk} \end{vmatrix} \qquad (k = 1, \ldots, n-1)$$

are different from zero, the matrix $\mathbf{A} = [a_{ij}]_{n\times n}$ *can be written in form*

(2.1.1.1) $$\mathbf{A} = \mathbf{LR},$$

where **L** *lower, and* **R** *upper triangular matrix.*

Triangular matrices **L** and **R** of order n are of following forms:

(2.1.1.2) $$\mathbf{L} = [l_{ij}]_{n\times n} \quad (l_{ij} = 0 \Leftarrow i < j),$$

(2.1.1.3) $$\mathbf{R} = [r_{ij}]_{n\times n} \quad (r_{ij} = 0 \Leftarrow i > j).$$

Decomposition (2.1.1.1), known as **LR** factorization (decomposition), is not unique, having in mind the equality

$$\mathbf{LR} = (c\mathbf{L})(\frac{1}{c}\mathbf{R}) \quad (\forall c \neq 0).$$

Nevertheless, if diagonal elements of matrix **R** (or **L**) take fixed values, not one being equal to zero, the decomposition is unique. In regards to (2.1.1.2) and (2.1.1.3), and having in mind

$$a_{ij} = \sum_{k=1}^{max(i,j)} l_{ik} r_{kj} \quad (i, j = 1, \ldots, n)$$

the elements of matrices **L** and **R** can be easy determined by recursive procedure, giving in advance the values for elements $r_{ii}(\neq 0)$ or $l_{ii}(\neq 0)$ $(i = 1, \ldots, n)$. For example, if given numbers $r_{ii}(\neq 0)$ $(i = 1, \ldots, n)$, it holds

$$l_{11} = \frac{a_{11}}{r_{11}}$$

$$\left\{\begin{aligned} r_{1i} &= \frac{a_{1i}}{l_{11}} \\ l_{i1} &= \frac{a_{i1}}{r_{11}} \end{aligned}\right\} \quad (i = 2, \ldots, n);$$

$$\left\{\begin{array}{l} l_{ii} = \dfrac{1}{r_{ii}}\left(a_{ii} - \displaystyle\sum_{k=1}^{i-1} l_{ik} r_{ki}\right) \\ \left\{\begin{aligned} r_{ij} &= \frac{1}{l_{ii}}\left(a_{ij} - \sum_{k=1}^{i-1} l_{ik} r_{kj}\right) \\ l_{ji} &= \frac{1}{r_{ii}}\left(a_{ji} - \sum_{k=1}^{i-1} l_{jk} r_{ki}\right) \end{aligned}\right\} \quad (j = i+1, \ldots, n); \end{array}\right\} \quad (i = 2, \ldots, n).$$

In similar way can be defined recursive procedure for determination of matrix elements of matrices **L** and **R**, if the numbers $l_{ii}(\neq 0)$ $(i = 1, \ldots, n)$ are given in advance. In practical applications one usually takes $r_{ii} = 1$ $(i = 1, \ldots, n)$ or $l_{ii} = 1 (i = 1, \ldots, n)$.

Very frequent case in application is of multi-diagonal matrices, i.e. matrices with elements different from zero on the main diagonal and around the main diagonal. For example, if $a_{ij} \neq 0$ for $|i - j| \leq 1$ and $a_{ij} = 0$ for $|i - j| > 1$, the matrix is tri-diagonal. The elements of such a matrix are usually written as vectors $(a_2, \ldots, a_n), (b_1, \ldots, b_n), (c_1, \ldots, c_{n-1})$, i.e.

$$\text{(2.1.1.4)} \qquad \mathbf{A} = \begin{bmatrix} b_1 & c_1 & 0 & \ldots & 0 & 0 \\ a_2 & b_2 & c_2 & & 0 & 0 \\ 0 & a_3 & b_3 & & 0 & 0 \\ \vdots & & & & & \\ 0 & 0 & 0 & & a_n & b_n \end{bmatrix}.$$

If $a_{ij} \neq 0$ $(|i - j| \leq 2)$ and $a_{ij} = 0$ $(|i - j| > 2)$, we have a case of five-diagonal matrix. Let us now suppose that tri-diagonal matrix (2.1.1.4) fulfills the conditions of Theorem 2.1.1.1. For decomposition of such a matrix it is enough to suppose that

$$\mathbf{L} = \begin{bmatrix} \beta_1 & 0 & 0 & \ldots & 0 & 0 \\ \alpha_2 & \beta_2 & 0 & & 0 & 0 \\ 0 & \alpha_3 & \beta_3 & & 0 & 0 \\ \vdots & & & & & \\ 0 & 0 & 0 & & \alpha_n & \beta_n \end{bmatrix} \quad (\beta_1 \beta_2 \ldots \beta_n \neq 0)$$

and

$$\mathbf{R} = \begin{bmatrix} 1 & \gamma_1 & 0 & \ldots & 0 & 0 \\ 0 & 1 & \gamma_2 & & 0 & 0 \\ 0 & 0 & 1 & & 0 & 0 \\ \vdots & & & & & \\ 0 & 0 & 0 & & 0 & 1 \end{bmatrix}.$$

By comparing corresponding elements of matrix A and matrix

$$\mathbf{LR} = \begin{bmatrix} \beta_1 & \beta_1\gamma_1 & 0 & \dots & 0 & 0 \\ \alpha_2 & \alpha_2\gamma_1+\beta_2 & \beta_2\gamma_2 & & 0 & 0 \\ 0 & \alpha_3 & \alpha_3\gamma_2+\beta_3 & & 0 & 0 \\ \vdots & & & & & \\ 0 & 0 & 0 & & \alpha_n & \alpha_n\gamma_{n-1}+\beta_n \end{bmatrix},$$

we get the following recursive formulas for determination of elements $\alpha_i, \beta_i, \gamma_i$:

$$\begin{aligned} & \beta_1 = b_1, \qquad \gamma_1 = \frac{c_1}{\beta_1}, \\ \alpha_i = a_i, \quad & \beta_i = b_i - \alpha_i\gamma_{i-1}, \quad \gamma_i = \frac{c_i}{\beta_i} \quad (i = 2, \dots, n-1), \\ \alpha_n = a_n, \quad & \beta_n = b_n - \alpha_n\gamma_{n-1}. \end{aligned}$$

2.1.2. Matrix eigenvectors and eigenvalues

Definition 2.1.2.1. *Let* **A** *complex quadratic matrix of order n. Every vector $\vec{x} \in C^n$, different from zero-vector is named eigenvector of matrix* **A** *if there exists scalar $\lambda \in C$ such that*

$$\mathbf{A}\vec{x} = \lambda\vec{x}. \tag{2.1.2.1}$$

Scalar λ is then named the corresponding eigenvalue.

Considering that (2.1.2.1) can be written in form

$$(\mathbf{A} - \lambda\mathbf{I})\vec{x} = \vec{0},$$

one can conclude that equation (2.1.2.1) has non-trivial solutions (in $\vec{x}$) if and only if $\det(\mathbf{A} - \lambda\mathbf{I}) = 0$.

2.2. DIRECT METHODS IN LINEAR ALGEBRA

2.2.1. Introduction

Numerical problems in linear algebra can be classified in several groups:

1. Solution of system of linear algebraic equations

$$\mathbf{A}\vec{x} = \vec{b},$$

where **A** regular matrix, calculation of determinant of matrix **A**, and matrix **A** inversion;
2. Solution of arbitrary system of linear equations using least-square method;
3. Determination of eigenvalues and eigenvectors of given quadratic matrix;
4. Solution of problems in linear programming.

For solution of these problems, a number of methods is developed. They can be separated in two classes, as follows.

The first class contains so-called direct methods, known sometimes as exact methods. The basic characteristic of those methods is that after final number of transformations (steps) one gets the result. Presuming all operations being performed exact, the gained result would be absolutely exact. Of course, because the performed computations are performed with rounding intermediate results, the final result is of limited exactness.

The second class is made of iterative methods, obtaining the result after infinite number of steps. As initial values for iterative methods are usually used the results obtained by some direct method.

Note that at solution of systems with big number of equation, used for solution of partial differential equations, the iterative methods are usually used.

2.2.2. Gauss elimination with pivoting

Consider the system of linear algebraic equations

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n, \end{aligned} \tag{2.2.2.1}$$

or, in matrix form

$$\mathbf{A}\vec{x} = \vec{b}, \tag{2.2.2.2}$$

where

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & & & \\ a_{n1} & a_{n2} & \ldots & a_{nn} \end{bmatrix}, \quad \vec{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}, \quad \vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}.$$

Suppose that system of equation (2.2.2.2) has an unique solution. It is very known that solutions of system (2.2.2.1), i.e. (2.2.2.2), can be expressed using Crammer's rules

$$x_i = \frac{\det \mathbf{A}_i}{\det \mathbf{A}} \quad (i = 1, 2, \cdots n),$$

where matrix $\mathbf{A_i}$ is obtained from matrix $\mathbf{A}$ by replacing i-th column by vector $\vec{b}$. Nevertheless, these formulas are inappropriate for practical calculations because for calculation of $n+1$ determinants one needs a big number of calculations. Namely, if we would like to calculate the value of determinant of n-th degree by developing of determinant through rows or columns, it would be necessary to proceed $S_n = n! - 1$ additions and $M_n \cong n!(e-1)$ multiplications ($n > 4$), what gives the total number of calculations $P_n = M_n + S_n \cong n!e$. Supposing that one operation demands $100\mu s$ (what is the case with fast computers), the total time for calculation of value of determinant of order thirty ($n = 30$) would take approximately $2.3 \cdot 10^{20}$ years. Generally speaking, such one procedure is practically unusable for determinants of order $n > 5$. One of the most suitable direct methods for solution of system of linear equations is Gauss method of elimination. This method is based on reduction of system (2.2.2.2), using equivalent transformations, to the triangular system

$$\mathbf{R}\vec{x} = \vec{c}, \tag{2.2.2.3}$$

where

$$\mathbf{R} = \begin{bmatrix} r_{11} & r_{12} & \ldots & r_{1n} \\ & r_{22} & \ldots & r_{2n} \\ & & \ddots & \\ & \cdot & & r_{nn} \end{bmatrix}, \quad \vec{c} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix}.$$

System (2.2.2.3) is solved successively starting from the last equation. Namely,

$$x_n = \frac{c_n}{r_{nn}},$$
$$x_i = \frac{1}{r_{ii}}(c_i - \sum_{k=i+1}^{n} r_{ik}x_k) \quad (i = n-1, \ldots, 1).$$

Note that coefficients $r_{ii} \neq 0$, because of assumption that system (2.2.2.2), i.e. (2.2.2.3) has an unique solution.

We will show now how system (2.2.2.1) can be reduced to equivalent system with triangular matrix.

Supposing $a_{11} \neq 0$, let us compute first the factors

$$m_{i1} = \frac{a_{i1}}{a_{11}} \quad (i = 2, \ldots, n),$$

and then, by multiplication first equation in system (2.2.2.1) by m and subtracting from i-th equation, one gets the system of $n-1$ equations

$$\begin{aligned} a_{22}^{(2)}x_2 + \ldots + a_{2n}^{(2)}x_n &= b_2^{(2)} \\ &\vdots \\ a_{n2}^{(2)}x_2 + \ldots + a_{nn}^{(2)}x_n &= b_n^{(2)} \end{aligned} \tag{2.2.2.4}$$

where

$$a_{ij}^{(2)} = a_{ij} - m_{i1}a_{1j}, \quad b_i^{(2)} = b_i - m_{i1}b_1 \quad (i, j = 2, \ldots, n).$$

Assuming $a_{22} \neq 0$, and applying the same procedure to (2.2.2.4), with

$$m_{i2} = \frac{a_{i2}}{a_{22}} \quad (i = 3, \ldots, n),$$

one gets the system of $n-2$ equations

$$\begin{aligned} a_{33}^{(3)}x_3 + \ldots + a_{3n}^{(3)}x_n &= b_3^{(3)} \\ &\vdots \\ a_{n3}^{(3)}x_3 + \ldots + a_{nn}^{(3)}x_n &= b_n^{(3)} \end{aligned}$$

where

$$a_{ij}^{(3)} = a_{ij}^{(2)} - m_{i2}a_{2j}^{(2)}, \quad b_i^{(3)} = b_i^{(2)} - m_{i2}b_2^{(2)} \quad (i, j = 3, \ldots, n).$$

Continuing this procedure, after $n-1$ steps, one gets the equation

$$a_{nn}^{(n)}x_n = b_n^{(n)}.$$

From the obtained systems, taking the first equations, one gets the system of equations

$$\begin{aligned} a_{11}^{(1)}x_1 + a_{12}^{(1)}x_2 + a_{13}^{(1)}x_3 + \cdots + a_{1n}^{(1)}x_n &= b_1^{(1)} \\ a_{22}^{(2)}x_2 + a_{23}^{(2)}x_3 + \cdots + a_{2n}^{(2)}x_n &= b_2^{(2)} \\ a_{33}^{(3)}x_3 + \cdots + a_{3n}^{(3)}x_n &= b_3^{(3)} \\ &\vdots \\ a_{nn}^{(n)}x_n &= b_n^{(n)}, \end{aligned}$$

where we put $a_{ij}^{(1)} = a_{ij}$, $b_i^{(1)} = b_i$.

The presented triangular reduction, or as often called Gauss elimination, is actually determination of coefficients

$$\begin{aligned} m_{ik} &= \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}, \\ a_{ij}^{(k+1)} &= a_{ij}^{(k)} - m_{ik}a_{kj}^{(k)}, \\ b_i^{(k+1)} &= b_i^{(k)} - m_{ik}b_k^{(k)} \quad (i, j = k+1, \ldots, n) \end{aligned}$$

for $k = 1, 2, \ldots, n-1$. Note that the elements of matrix $\mathbf{R}$ and vector $\vec{c}$ are given as

$$r_{ij} = a_{ij}^{(i)}, \; c_i = b_i^{(i)} \quad (i = 1, \ldots, n; j = i, \ldots, n).$$

In order the presented reduction to exists, it is necessary to obtain the condition $a_{kk}^{(k)} \neq 0$. Elements $a_{kk}^{(k)}$ are known as main elements (pivotal elements or pivot). Assuming matrix $\mathbf{A}$ of system (2.2.2.2) being regular, the conditions $a_{kk}^{(k)} \neq 0$ are to be obtained by permutation of equations in system.

Moreover, from the point of view of accuracy of results, it is necessary to use so known strategy of choice of pivotal elements. Modification of Gauss elimination method in this sense is called Gauss method with choice of pivotal element. In accordance to this method, for pivotal element in k-th elimination step one takes the element $a_{rk}^{(k)}$, for which holds

$$|a_{rk}^{(k)}| = \max_{k \le i \le n} |a_{ik}^{(k)}|,$$

with permutation of k-th and r-th row.

If one obtains in addition to permutation of equations the permutation of unknowns, it is the best way to take for pivotal element in the k-th elimination step the element $a_{rk}^{(k)}$, for which it holds

$$|a_{rs}^{(k)}| = \max_{k \le i, j \le n} |a_{ij}^{(k)}|$$

with permutation of k-th and r-th row (equations) and k-th and s-th column (unknowns). Such method is called the method with total choice of pivotal element.

One can show (see [1], pp. 233-234) that total number of calculations by applying Gauss method is

$$N(n) = \frac{1}{6}(4n^3 + 9n^2 - 7n).$$

For n big enough, one gets $N(n) \cong 2n^3/3$. It was long time opinion that Gauss method is optimal regarding number of computations. Nowadays, V. Strassen, by involving iterative algorithm for multiplying and inverse of matrices, gave a new method for solution of system of linear equations, by which the number of computations is of order $n^{\log_2 7}$. Strassen method is thus better than Gauss method $\log_2 7 < 3$.

Triangular reduction obtains simple computation of system determinant. Namely, it holds

$$\det \mathbf{A} = a_{11}^{(1)} a_{22}^{(2)} \ldots a_{nn}^{(n)}.$$

When used Gauss method with choice of pivotal element, one should take care about number of permutations of rows (and columns by using method of total choice of pivotal element), what influences the sign of determinant. This way of determinant calculation is high efficient. For example, for calculation of determinant of order $n = 30$, one needs $0.18s$, presuming that one arithmetic operation takes $10\mu s$.

2.2.3. Matrix inversion using Gauss method

Let $\mathbf{A} = [a_{ij}]_{n\times n}$ be regular matrix and let

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & x_{2n} \\ \vdots & & & \\ x_{n1} & x_{n2} & \dots & x_{nn} \end{bmatrix} = [\,\vec{x}_1 \quad \vec{x}_2 \quad \dots \quad \vec{x}_n\,]$$

be its inverse matrix. Vectors $\vec{x}_1, \vec{x}_2, \dots, \vec{x}_n$ are first, second,..., n-th column of matrix $\mathbf{X}$. Let us now define vectors $\vec{e}_1, \vec{e}_2, \dots \vec{e}_n$ as

$$\vec{e}_1 = [1\ 0 \dots 0]^T,\ \vec{e}_2 = [0\ 1\ 0 \dots 0]^T,\ ,\dots, \vec{e}_n = [0\ 0 \dots 1]^T.$$

Regarding to equality

$$\mathbf{AX} = [\mathbf{A}\vec{x}_1\ \mathbf{A}\vec{x}_2\ \dots \mathbf{A}\vec{x}_n] = \mathbf{I} = [\vec{e}_1\ \vec{e}_2 \dots \vec{e}_n],$$

the problem of determination of inverse matrix can reduce to solving of n systems of linear equations

(2.2.3.1) $$\mathbf{A}\vec{x}_i = \vec{e}_i, \quad (i = 1, \dots, n).$$

For solving of system (2.2.3.1) it is convenient to use Gauss method, taking in account that matrix $\mathbf{A}$ appears as a matrix of all systems, so that its triangular reduction shell be done once only. By this procedure all the transformations necessary for triangular reduction of matrix $\mathbf{A}$ should be applied to the unit matrix $\mathbf{I} = [\vec{e}_1\vec{e}_2 \dots \vec{e}_n]$ too. In this way matrix $\mathbf{A}$ transforms to triangular matrix $\mathbf{R}$, and matrix $\mathbf{I}$ to matrix $\mathbf{C} = [\vec{c}_1\vec{c}_2 \dots \vec{c}_n]$. Finally, triangular systems of form

$$\mathbf{R}\vec{x}_i = \vec{c}_i \quad (i = 1, \dots, n)$$

should be solved.

2.2.4. Factorization methods

Factorization methods for solving of system of linear equations are based on factorization of matrix of system to product of two matrices in such form that enables reduction of system to two systems of equations which can be simple successive solved. In this section we will show up at the methods based on **LR** matrix factorization (see Section 2.1.1.).

Given the system of equations

(2.2.4.1) $$\mathbf{A}\vec{x} = \vec{b},$$

with quadratic matrix $\mathbf{A}$, which all main diagonal minors are zero different. Then, based on Theorem 2.1.1.1, it exists factorization matrix $\mathbf{A} = \mathbf{LR}$, where $\mathbf{L}$ lower and $\mathbf{R}$ upper triangular matrix. The factorization is unique defined, if, for example, one adopts unit diagonal of matrix $\mathbf{L}$. In this case, system (2.2.4.1), i.e. system $\mathbf{LR}\vec{x} = \vec{b}$ can be presented in equivalent form

(2.2.4.2) $$\mathbf{L}\vec{y} = \vec{b}, \quad \mathbf{R}\vec{x} = \vec{y}.$$

Based on previous, for solving of system of equations (2.2.4.1), the following method can be formulated:

1. Put $l_{ii} = 1 \quad (i = 1, \dots, n)$;

2. Determine other elements of matrix $\mathbf{L} = [l_{ij}]_{n\times n}$ and matrix $\mathbf{R} = [r_{ij}]_{n\times n}$ (see Section 2.1.1);
3. Solve first system of equations in (2.2.4.2);
4. Solve second system of equations in (2.2.4.2).

   Steps 3. and 4. are simple to be performed. Namely, let

$$\vec{b} = [b_1\, b_2 \dots b_n]^T, \quad \vec{y} = [y_1\, y_2 \dots y_n]^T, \quad \vec{x} = [x_1\, x_2 \dots x_n]^T.$$

Then

$$y_1 = b_1, \; y_i = b_i - \sum_{k=1}^{i-1} l_{ik} y_k \quad (i = 2, \dots, n)$$

and

$$x_n = \frac{y_n}{r_{nn}}, \quad x_i = \frac{1}{r_{ii}}(y_i - \sum_{k=i+1}^{n} r_{ik} x_k) \quad (i = n-1, \dots, 1).$$

The method presented is known in bibliography as method of Cholesky. In the case when matrix $\mathbf{A}$ is normal, i.e. symmetric and positive definite, the Cholesky method can be simplified. Namely, in this case one can take that $\mathbf{L} = \mathbf{R}^T$. . Thus, the factorization of matrix $\mathbf{A}$ in form $\mathbf{A} = \mathbf{R}^T\mathbf{R}$ should be performed. Based on formulas from Section 2.1.1 for elements of matrix $\mathbf{R}$ it holds:

$$\begin{aligned} r_{11} &= \sqrt{a_{11}} \\ r_{1j} &= \frac{a_{1j}}{r_{11}} \qquad (j = 2, \dots, n), \\ r_{ii} &= \sqrt{a_{ii} - \sum_{k=1}^{i-1} r_{ki}^2} \\ & \qquad\qquad\qquad\qquad\qquad\qquad (i = 2, \dots, n). \\ r_{ij} &= \frac{1}{r_{ii}}(a_{ij} - \sum_{k=1}^{i-1} r_{ki} r_{kj}) \quad (j = i+1, \dots, n). \end{aligned}$$

In this case the systems (2.2.4.2) become

$$\mathbf{R}^T\vec{y} = \vec{b}, \quad \mathbf{R}\vec{x} = \vec{y}.$$

Remark 2.2.4.1. *The determinant of normal matrix can be calculated by method of square root as*

$$\det \mathbf{A} = (r_{11}\, r_{22} \dots r_{nn})^2.$$

Factorization methods are specially convenient for solving of systems of linear equations where matrix of systems does not change, but only free vector $\vec{b}$. Such systems are very frequent in engineering.

Now it will be shown that Gauss method of elimination can be interpreted as **LR** factorization of matrix $\mathbf{A}$. Take matrix $\mathbf{A}$ such that during the elimination process permutation of rows and columns should not be performed. Denote the starting system as $\mathbf{A}^{(1)}\vec{x} = \vec{b}^{(1)}$. Gauss elimination procedure gives $n-1$ equivalent systems $\mathbf{A}^{(2)}\vec{x} = \vec{b}^{(2)} \dots \mathbf{A}^{(n)}\vec{x} = \vec{b}^{(n)}$, where matrix $\mathbf{A}^{(k)}$ is of form

$$\mathbf{A}^{(k)} = \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \dots & a_{1k}^{(1)} & \dots & a_{1n}^{(1)} \\ & a_{22}^{(2)} & & a_{2k}^{(2)} & & a_{2n}^{(2)} \\ & & \ddots & \vdots & & \\ & & & a_{kk}^{(k)} & & a_{kn}^{(k)} \\ & & & \vdots & & \\ & & & a_{nk}^{(k)} & & a_{nn}^{(k)} \end{bmatrix}.$$

Let us analyze modification of elements $a_{ij}(=a_{ij}^{(1)})$ during the process of triangular reduction. Because, for $k=1,2,\ldots,n-1$,

$$a_{ij}^{(k+1)}=a_{ij}^{(k)}-m_{ik}a_{kj}^{(k)}\quad(i,j=k+1,\ldots,n),$$

and

$$a_{i1}^{(k+1)}=a_{i2}^{(k+1)}=\ldots=a_{ik}^{(k+1)}=0\quad(i=k+1,\ldots,n),$$

by summation we get

$$a_{ij}=a_{ij}^{(1)}=a_{ij}^{(i)}+\sum_{k=1}^{i-1}m_{ik}a_{kj}^{(k)}\quad(i\le j)$$

and

$$a_{ij}=a_{ij}^{(1)}=0+\sum_{k=1}^{i}m_{ik}a_{kj}^{(k)}\quad(i>j).$$

By defining $m_{ii}=1$ $(i=1,\ldots,n)$, the last two equalities can be given in form

(2.2.4.3) $$a_{ij}=\sum_{k=1}^{p}m_{ik}a_{kj}^{(k)}\quad(i,j=1,\ldots,n),$$

where $p=\min(i,j)$. Equality (2.2.4.3) is pointing out that Gauss elimination procedure gives **LR** factorization of matrix **A**, where

$$\mathbf{L}=\begin{bmatrix}1&&&\\m_{21}&1&&\\&&\ddots&\\m_{n1}&m_{n2}&\cdots&1\end{bmatrix},\quad\mathbf{R}=\begin{bmatrix}r_{11}&r_{12}&\cdots&r_{1n}\\&r_{22}&\cdots&r_{2n}\\&&\ddots&\\&&&r_{nn}\end{bmatrix}.$$

and $r_{kj}=a_{kj}^{(k)}$. During program realization of Gauss method in order to obtain **LR** factorization of matrix **A** , it is not necessary to use new memory space for matrix **L**, but it is convenient to load factors m_{ik} in the place of matrix **A** coefficients which are annulled in process of triangular reduction. In this way, after completed triangular reduction, in the memory space of matrix **A** will be memorized matrices **L** and **R**, according to following scheme:

$$\mathbf{A}\Rightarrow\mathbf{LR}$$

Consider that diagonal elements of matrix **L**, all equal to unit, should not be memorized.

Cholesky method, based on **LR** factorization, is used when matrix **A** fulfils conditions of Theorem 2.1.1.1. Nevertheless, usability of this method can be broaden to other systems with regular matrix, taking in account permutation of equations in system. For factorization is used Gauss elimination method with pivoting. There will be $\mathbf{LR}=\mathbf{A}'$, where matrix $\mathbf{A}'$ is obtained from matrix **A** by finite number of row interchange. This means that in elimination process set of indices of pivot elements $I=(p_1,\ldots,p_{n-1})$, where p_k is number of row from which the main element is taken in k-th elimination step, should be memorized. By solving of system $\mathbf{A}\vec{x}=\vec{b}$, after accomplishing a process of factorization, according to set of indices I, coordinates of vector $\vec{b}$ should be permuted. In this way the transformed vector $\vec{b}'$ is obtained, so that solving of given system reduces to successive solving of triangular systems

(2.2.4.4) $$\mathbf{L}\vec{y}=\vec{b}',\quad\mathbf{R}\vec{x}=\vec{y}.$$

2.2.5. Program realization

This section is devoted to software realization of methods previously exposed in this chapter. For successful following of material in this subchapter it is necessary knowledge exposed in all previous subchapters of this chapter. In presented subprograms the matrices are treated as vectors.

Program 2.2.5.1. Subprogram for matrix transpose MTRN is of form:

```
      SUBROUTINE MTRN (A, B, N, M)
C
C TRANSPONTOVANJE MATRICE A
C
      DIMENSION A(1), B(1)
      IC=0
      DO 5 I=1, N
      IJ=I-N
      DO 5 J=1, M
      IJ=IJ+N
      IC=IC+1
    5 B(IC)=A(IJ)
      RETURN
      END
```

Parameters in the list of subprogram parameters have the following meaning:

A - input matrix of type $N \times M$, treated as vector of length NM (taken in form column by column);

B - output matrix of type $M \times N$ ($\mathbf{B} = \mathbf{A}^T$). Matrix is treated in the same way as matrix $\mathbf{A}$.

Program 2.2.5.2. Subprogram for multiplication of matrices A (of dimension $N \times M$) and B (of dimension $M \times L$) is of form

```
      SUBROUTINE MMAT (A, B, C, N, M, L)
C
C MATRICA A TIPA N*M
C MATRICA B TIPA M*L
C MATRICA C TIPA N*L
C MNOZENJE MATRICA C=A*B
C
      DIMENSION A(1), B (1), C (1)
      IC=0
      I2=-M
      DO 5 J=1,L
      I2=I2+M
      DO 5 I=1, N
      IC=IC+1
      IA=I-N
      IB=I2
      C(IC)=0.
      DO 5 K=1, M
      IA=IA+N
      IB=IB+1
    5 C(IC)=C(IC) + A(IA)*B(IB)
      RETURN
      END
```

Output matrix C ($\mathbf{C} = \mathbf{A} \times \mathbf{B}$) is of dimension $N \times L$.

Program 2.2.5.3. Let us write a program for computing matrix $\mathbf{B}^T\mathbf{A}$, by using previously given subprograms, for given matrices $\mathbf{A}$ and $\mathbf{B}$. Let matrix $\mathbf{A}$ be of type $N \times M$, and matrix $\mathbf{B}$ of type $N \times K$ (with maximal number of matrix elements for both matrices 100).

This program has the following form:

```
      DIMENSION A(100), B(100), C(100)
      OPEN(8,FILE='MTMM.IN')
      OPEN(5,FILE='MTMM.OUT')
      READ (8,10) N,M,K
   10 FORMAT (3I2)
      NM=N*M
      NK=K*M
      KM=K*M
      READ (8,20) (A(I), I=1, NM), (B(I), I=1, NK)
   20 FORMAT(16F5.0)
      CALL MTRN( B, C, N, K)
      CALL MMAT (C, A, B, K, N, M)
      WRITE (5,30) ((B(J), J=I, KM, K), I=1, K)
   30 FORMAT (5X, 'MATRIX C=B(TR)* A'// (2X,4F6.1))
      CLOSE(8)
      CLOSE(5)
      STOP
      END
```

Test of program, being proceeded with matrices

$$\mathbf{A} = \begin{bmatrix} -1 & 3 & 0 & 2 \\ 1 & 4 & 1 & 5 \\ 0 & 1 & -2 & 0 \\ -2 & 3 & 1 & 3 \end{bmatrix}, \text{ and } \mathbf{B} = \begin{bmatrix} 1 & -3 & 0 \\ 0 & 4 & -6 \\ 2 & -1 & 2 \\ -1 & 5 & 1 \end{bmatrix},$$

gave the following result:

```
     MATRIX C=B(TR)* A
     1.0   2.0  -5.0  -1.0
    -3.0  21.0  11.0  29.0
    -8.0 -19.0  -9.0 -27.0
```

Program 2.2.5.4. Method of Cholesky for solving of system of linear equations (see subchapter 2.2.4) can be realized in the following way:

```
C=====================================================
C         CHOLESKY  METHOD
C=====================================================
      DIMENSION A(10,10), B(10)
      OPEN(8,FILE='CHOLESKY.IN')
      OPEN(5,FILE='CHOLESKY.OUT')
   33 READ(8,100)N
  100 FORMAT(I2)
      IF(N)11,22,11
   11 READ(8,101)(B(I),I=1,N)
  101 FORMAT(8F10.4)
```

```
C    READ IN THE UPPER MATRIX TRIANGLE OF A
      READ(8,101)((A(I,J),J=1,N),I=1,N)
      WRITE(5,102)N
  102 FORMAT(/ 5X,'MATRIX DIMENSION =',I3//
     1 5X,'MATRICA  A',
     2 <(N-1)*12+3>X,'VEKTOR B'/)
      WRITE(5,103)((A(I,J),J=1,N),B(I),I=1,N)
  103 FORMAT(1X,<N>F12.7,F13.7)
C    FACTORIZATION OF MATRIX  A TO THE FORM  A=L*R
      DO 10 I=2,N
   10 A(1,I)=A(1,I)/A(1,1)
      DO 25 I=2,N
      I1=I-1
      S=A(I,I)
      DO 20 K=1,I1
   20 S=S-A(I,K)*A(K,I)
      A(I,I)=S
      IF(I.EQ.N) GO TO 40
      IJ=I+1
      DO 25 J=IJ,N
      S=A(I,J)
      T=A(J,I)
      DO 30 K=1,I1
      S=S-A(I,K)*A(K,J)
   30 T=T-A(J,K)*A(K,I)
      A(I,J)=S/A(I,I)
   25 A(J,I)=T
   40 WRITE(5,107)
  107 FORMAT(//5X,'MATRIX L'/)
      DO 111 I=1,N
  111 WRITE(5,103)(A(I,J),J=1,I)
      WRITE(5,108)
  108 FORMAT(//5X,'MATRIX R'/)
      N1=N-1
      DO 222 I=1,N1
      II=I+1
      M=N-I
  222 WRITE(5,99) (A(I,J),J=II,N)
      WRITE(5,99)
   99 FORMAT(<12*I-8>X,'1.0000000',<M>F12.7)
C     OBTAINING THE VECTOR OF SOLUTIONS
      B(1)=B(1)/A(1,1)
      DO 55 I=2,N
      I1=I-1
      DO 45 K=1,I1
   45 B(I)=B(I)-A(I,K)*B(K)
   55 B(I)=B(I)/A(I,I)
      DO 50 J=1,N1
      I=N-J
      I1=I+1
      DO 50 K=I1,N
   50 B(I)=B(I)-A(I,K)*B(K)
      WRITE(5,109)
  109 FORMAT(//13X,'VEKTOR OF SOLUTIONS'/)
      WRITE(5,104)(B(I),I=1,N)
  104 FORMAT(12X,F12.7)
      GO TO 33
   22 CLOSE(5)
      CLOSE(8)
      STOP
      END
```

For factorization of matrix $\mathbf{A}(=\mathbf{LR})$ we take in upper triangular matrix $\mathbf{R}$ unit diagonal, i.e. $r_{ii} = 1$ $(i = 1,\ldots,n)$. Program is organized in this way so that matrix $\mathbf{A}$ transforms to matrix $\mathbf{A}_1$, which lower triangle (including main diagonal) is equal to matrix $\mathbf{L}$, and strict upper triangle to matrix $\mathbf{R}$. Note that diagonal elements in matrix $\mathbf{R}$ are not memorized, but only formally printed, using statement `FORMAT`. Note also that in Section 2.2.4. the unit diagonal has been adopted into matrix $\mathbf{L}$.

By applying this program to the applicable system of equations, the following results are obtained:

```
 MATRIX DIMENSION =  4
 MATRICA  A                                                     VEKTOR B
1.0000000   4.0000000   1.0000000   3.0000000    9.0000000
 .0000000  -1.0000000   2.0000000  -1.0000000     .0000000
3.0000000  14.0000000   4.0000000   1.0000000   22.0000000
1.0000000   2.0000000   2.0000000   9.0000000   14.0000000
 MATRIX L
1.0000000
 .0000000  -1.0000000
3.0000000   2.0000000   5.0000000
1.0000000  -2.0000000  -3.0000000   2.0000000
 MATRIX R
1.0000000   4.0000000   1.0000000   3.0000000
       1.0000000  -2.0000000   1.0000000
                   1.0000000  -2.0000000
                   1.0000000
   VEKTOR OF SOLUTIONS
     1.0000000
     1.0000000
     1.0000000
     1.0000000
```

Program 2.2.5.5. In similar way can be realized square root method for solution of system of linear equations with symmetric, positive definite matrix. In this case it is enough to read in only main diagonal elements of matrix $\mathbf{A}$, and, for example, elements from upper triangle.

The program and output listing for given system of equations are given in the following text. Note that from the point of view of memory usage it is convenient to treat matrix $\mathbf{A}$ as a vector. Nevertheless, due to easier understanding, we did not follow this convenience on this place.

Program is organized in this way so that, in addition to solution of system of equation, the determinant of system is also obtained. In output listing the lower triangle of symmetric matrix is omitted.

```
$DEBUG
C==============================================
C  SOLUTION OF SYSTEM OF LINEAR EQUATIONS
C        BY SQARE ROOT METHOD
C==============================================
      DIMENSION A(10,10),B(10)
      OPEN(8,FILE='SQR.IN')
      OPEN(5,FILE='SQR.OUT')
    3 READ(8,100)N
  100 FORMAT(I2)
      IF(N) 1,2,1
```

```
C READ IN VECTOR B
    1 READ(8,101) (B(I),I=1,N)
  101 FORMAT(8F10.4)
C READ IN UPPER TRIANGULAR PART OF MATRIX  A
      READ(8,101)((A(I,J),J=I,N),I=1,N)
      WRITE(5,102)
  102 FORMAT(////5X,'MATRIX OF SYSTEM'/)
      WRITE(5,99)((A(I,J),J=I,N),I=1,N)
   99 FORMAT(<12*I-11>X,<N-I+1>F12.7)
      WRITE(5,105)
  105 FORMAT(//5X,'VECTOR OF FREE MEMBERS'/)
      WRITE(5,133)(B(I),I=1,N)
  133 FORMAT(1X,10F12.7)
C  OBTAINING OF ELEMENTS OF UPPER TRIANGULAR MATRIX
      A(1,1)=SQRT(A(1,1))
      DO 11 J=2,N
   11 A(1,J)=A(1,J)/A(1,1)
      DO 12 I=2,N
      S=0.
      IM1=I-1
      DO 13 K=1,IM1
   13 S=S+A(K,I)*A(K,I)
      A(I,I)=SQRT(A(I,I)-S)
      IF(I-N) 29,12,29
   29 IP1=I+1
      DO 14 J=IP1,N
      S=0.
      DO 15 K=1,IM1
   15 S=S+A(K,I)*A(K,J)
   14 A(I,J)=(A(I,J)-S)/A(I,I)
   12 CONTINUE
C  CALCULATION OF DETERMINANT
      DET=1.
      DO 60 I=1,N
   60 DET=DET*A(I,I)
      DET=DET*DET
C SOLUTION OF SYSTEM  L*Y=B
      B(1)=B(1)/A(1,1)
      DO 7 I=2,N
      IM1=I-1
      S=0.
      DO 8 K=1,IM1
    8 S=S+A(K,I)*B(K)
      P=1./A(I,I)
    7 B(I)=P*(B(I)-S)
C
C SOLUTION OF SYSTEM  R*X=Y
C MEMORIZING OF RESULTS INTO VECTOR  B
C
      B(N)=B(N)/A(N,N)
      NM1=N-1
      DO 30 II=1,NM1
      JJ=N-II
      S=0.
      JJP1=JJ+1
      DO 50 K=JJP1,N
   50 S=S+A(JJ,K)*B(K)
   30 B(JJ)=(B(JJ)-S)/A(JJ,JJ)
C
```

```
C PRINTING OF RESULTS
C
      WRITE (5,201)
  201 FORMAT(//5X,'MATRIX  R'/)
      Pause 1
C     DO 222 I=1,N
  222 WRITE(5,199)((A(I,J),J=I,N),I=1,N)
  199 FORMAT(<12*I-11>X,<N-I+1>F12.7)
      WRITE(5,208) DET
  208 FORMAT(//5X,'SYSTEM DETERMINANT  D=',F11.7/)
      WRITE(5,109)
  109 FORMAT(//5X,'SYSTEM SOLUTION '/)
      WRITE(5,133)(B(I),I=1,N)
      GO TO 3
    2 CLOSE(5)
      CLOSE(8)
      STOP
      END
```

```
     MATRIX OF SYSTEM
                              3.0000000
                               .0000000
                              1.0000000
                              2.0000000
                              1.0000000
                              1.0000000
     VECTOR OF FREE MEMBERS
    4.0000000    3.0000000    3.0000000
     MATRIX  R
                              1.7320510
                               .0000000
                               .5773503
                              1.4142140
                               .7071068
                               .4082483
     SYSTEM DETERMINANT  D=   1.0000000
     SYSTEM SOLUTION
     .9999999     .9999998    1.0000000
```

Program 2.2.5.6. Method of factorization for solution of systems of linear equations based on Gauss elimination with choice of pivotal element (see Sections 2.2.2 and 2.2.4) can be programmable realized using the following subprograms:

```
      SUBROUTINE LRFAK(A,N,IP,DET,KB)
      DIMENSION A(1),IP(1)
      KB=0
      N1=N-1
      INV=0
      DO 45 K=1,N1
      IGE=(K-1)*N+K
C
C  FINDING THE PIVOTAL ELEMENT IN  K-TH
C  ELIMINATION STEP
C
      GE=A(IGE)
      I1=IGE+1
      I2=K*N
      IMAX=IGE
      DO 20 I=I1,I2
      IF(ABS(A(I))-ABS(GE)) 20,20,10
 10   GE=A(I)
      IMAX=I
 20   CONTINUE
      IF(GE)25,15,25
 15   KB=1
C
C  MATRIX OF SYSTEM IS SINGULAR
C
      RETURN
 25   IP(K)=IMAX-N*(K-1)
      IF(IP(K)-K) 30,40,30
 30   I=K
      IK=IP(K)
C
C  ROW PERMUTATION
C
      DO 35 J=1,N
      S=A(I)
      A(I)=A(IK)
      A(IK)=S
      I=I+N
 35   IK=IK+N
      INV=INV+1
C
C  K-TH ELIMINATION STEP
C
 40   DO 45 I=I1,I2
      A(I)=A(I)/GE
      IA=I
      IC=IGE
      K1=K+1
      DO 45 J=K1,N
      IA=IA+N
      IC=IC+N
 45   A(IA)=A(IA)-A(I)*A(IC)
C
C  CALCULATION OF DETERMINANT
C
      DET=1.
      DO 50 I=1,N
      IND=I+(I-1)*N
 50   DET=DET*A(IND)
      IF(INV-INV/2*2) 55,55,60
 60   DET=-DET
 55   RETURN
      END
C
C
      SUBROUTINE RSTS(A,N,IP,B)
      DIMENSION A(1),IP(1),B(1)
C
C  SUCCESSIVE SOLUTION OF TRIANGULAR SYSTEMS
C
      N1=N-1
C  VECTOR B PERMUTATION
      DO 10 I=1,N1
      I1=IP(I)
      IF(I1-I) 5,10,5
  5   S=B(I)
      B(I)=B(I1)
      B(I1)=S
 10   CONTINUE
C  SOLUTION OF LOWER TRIANGULAR SYSTEM
      DO 15 K=2,N
      IA=-N+K
      K1=K-1
      DO 15 I=1,K1
      IA=IA+N
 15   B(K)=B(K)-A(IA)*B(I)
C  SOLUTION OF UPPER TRIANGULAR SYSTEM
      NN=N*N
      B(N)=B(N)/A(NN)
      DO 25 KK=1,N1
      K=N-KK
      IA=NN-KK
      I=N+1
      DO 20 J=1,KK
      I=I-1
      B(K)=B(K)-A(IA)*B(I)
 20   IA=IA-N
 25   B(K)=B(K)/A(IA)
      RETURN
      END
```

Parameters in subprogram list of LRFAK are of following meaning:

A - Input matrix of order N stored columnnwise (column by column). After N-1 elimination steps matrix A transforms to matrix which contains triangular matrices L and R (see section 2.2.4);

N - order of matrix **A**;

IP - vector of length N-1, which is formed during elimination procedure and contains indices of pivot elements (see section 2.2.4);

DET - output variable containing determinant of matrix of system A, as product of elements on diagonal of matrix R, with accuracy up to sign. This value are corrected by sign on the end of procedure, having in mind number of row permutations during elimination process;

KB - control number with value KB=0 if factorization is correctly performed, and KB=1 if matrix of system is singular. In the last case, *LR* factorization does not exist.

Subroutine RSTS solves successively systems of equations (2.2.4.4). Parameters in list of subroutine parameters are of following meaning:

A - matrix obtained in subroutine LRFAK;

N - order of matrix A;

IP - vector obtained in subroutine LRFAK;

B - vector of free members in system to be solved. This vector transforms to vector of system solutions.

Main program is written in such way that, at first, given matrix A is factorized by means of subroutine LRFAK, and then is possible to solve system of equations $\mathbf{A}\vec{x} = \vec{b}$ for arbitrary number of vectors $\vec{b}$, by calling subroutine RSTS.

Main program and output listing are of form:

```
      DIMENSION A(100),B(10),IP(9)
      OPEN(8,FILE='FACTOR.IN')
      OPEN(5,FILE='FACTOR.OUT')
      READ(8,5)N
```

```
    5 FORMAT(I2)
      NN=N*N
      READ(8,10)(A(I),I=1,NN)
   10 FORMAT(16F5.0)
      WRITE(5,34)
   34 FORMAT(1H1,5X,'MATRICA A'/)
      DO 12 I=1,N
   12 WRITE(5,15)(A(J),J=I,NN,N)
   15 FORMAT(10F10.5)
      CALL LRFAK(A,N,IP,DET,KB)
      IF(KB) 20,25,20
   20 WRITE(5,30)
   30 FORMAT(1H0,'MATRICA JE SINGULARNA'//)
      GO TO 70
   25 WRITE(5,35)
   35 FORMAT(1H0,5X,'FAKTORIZOVANA MATRICA'/)
      DO 55 I=1,N
   55 WRITE(5,15)(A(J),J=I,NN,N)
      WRITE(5,75)DET
   75 FORMAT(/5X,'DETERMINANTA MATRICE A='F10.6/)
   50 READ(8,10,END=70) (B(I),I=1,N)
      WRITE(5,40)(B(I),I=1,N)
   40 FORMAT(/5X,'VEKTOR B'//(10F10.5))
      CALL RSTS(A,N,IP,B)
      WRITE(5,45) (B(I),I=1,N)
   45 FORMAT(/5X,'RESENJE'//(10F10.5))
      GO TO 50
   70 CLOSE(5)
      CLOSE(8)
      STOP
      END
```

```
1     MATRICA A
   3.00000   1.00000   6.00000
   2.00000   1.00000   3.00000
   1.00000   1.00000   1.00000
0     FAKTORIZOVANA MATRICA
   3.00000   1.00000   6.00000
    .33333    .66667  -1.00000
    .66667    .50000   -.50000
     DETERMINANTA MATRICE A=  1.000000
     VEKTOR B
   2.00000   7.00000   4.00000
     RESENJE
  18.99999  -7.00000  -8.00000
     VEKTOR B
   1.00000   1.00000   1.00000
     RESENJE
    .00000   1.00000    .00000
```

Program 2.2.5.7. Using subroutine LRFAK and RSTS, having in mind section 2.2.3, it is easy to write program for matrix inversion. The corresponding program and output result (for matrix from previous example) have the following forms:

```
C=================================================
C   INVERZIJA MATRICE
C=================================================
      DIMENSION A(100), B(10), IP(9),AINV(100)
      open(8,file='invert.in')
      open(5,file='invert.out')
      READ(8,5) N
    5 FORMAT(I2)
      NN=N*N
      READ(8,10)(A(I),I=1,NN)
   10 FORMAT(16F5.0)
      WRITE(5,34)
   34 FORMAT(1H1, 5X, 'MATRICA A'/)
      DO 12 I=1,N
   12 WRITE(5,15) (A(J),J=I,NN,N)
   15 FORMAT(10F10.5)
      CALL LRFAK(A,N,IP,DET,KB)
      IF(KB) 20,25,20
   20 WRITE(5,30)
   30 FORMAT(1H0,'MATRICA A JE SINGULARNA'//)
      GO TO 70
   25 DO 45 I=1,N
      DO 40 J=1,N
   40 B(J)=0.
      B(I)=1.
      CALL RSTS(A,N,IP,B)
      IN=(I-1)*N
      DO 45 J=1,N
      IND=IN+J
   45 AINV(IND)=B(J)
      WRITE(5,50)
   50 FORMAT(1H0,5X,'INVERZNA MATRICA'/)
      DO 55 I=1,N
   55 WRITE(5,15)(AINV(J),J=I,NN,N)
   70 CLOSE(5)
      CLOSE(8)
      STOP
      END
```

```
1     MATRICA A
   3.00000   1.00000   6.00000
   2.00000   1.00000   3.00000
   1.00000   1.00000   1.00000
0     INVERZNA MATRICA
  -2.00000   5.00000  -3.00000
   1.00000  -3.00000   3.00000
   1.00000  -2.00000   1.00000
```

Bibliography

[1] Milovanović, G.V., *Numerical Analysis I*, Naučna knjiga, Beograd, 1988 (Serbian).

[2] Milovanović, G.V. and Djordjević, Dj.R., *Programiranje numeričkih metoda na FORTRAN jeziku.* Institut za dokumentaciju zaštite na radu "Edvard Kardelj", Niš, 1981 (Serbian).

(The full list of references and further reading is given on the end of Chapter 3.)

LECTURES

LESSON III

3. Linear Systems of Algebraic Equations: Iterative Methods

3.1. Introduction

Consider system of linear equations

(3.1.1)
$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n, \end{aligned}$$

which can be written in matrix form

(3.1.2)
$$\mathbf{A}\vec{x} = \vec{b},$$

where

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & & & \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix}, \quad \vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad \vec{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}.$$

In this chapter we always suppose that system (3.1.1), i.e. (3.1.2) has an unique solution.

Iterative methods for solving systems (3.1.2) have as goal determination of solution $\vec{x}$ with accuracy given in advance. Namely, starting with arbitrary vector $\vec{x}^{(0)}$ $(= [x_1^{(0)} \ldots x_n^{(0)}]^T)$ by iterative methods one defines the series $\vec{x}^{(k)}$ $(= [x_1^{(k)} \ldots x_n^{(k)}]^T)$ such that

$$\lim_{k\to+\infty} \vec{x}^{(k)} = \vec{x}.$$

3.2. Simple iteration method

One of the most simplest methods for solving system of linear equations is method of simple iteration. For application of this method it is necessary to transform previously system (3.1.2) to the following equivalent form

(3.2.1)
$$\vec{x} = \mathbf{B}\vec{x} + \vec{\beta}.$$

Then, the method of simple iteration is given as

$$\vec{x}^{(k)} = \mathbf{B}\vec{x}^{(k-1)} + \vec{\beta} \quad (k = 1, 2, \ldots). \tag{3.2.2}$$

Starting from arbitrary vector $\vec{x}^{(0)}$ and using (3.2.2) one generates series $\{\vec{x}^{(k)}\}$, which under some conditions converges to solution of given system.

If

$$\mathbf{B} = \begin{bmatrix} b_{11} & b_{12} & \ldots & b_{1n} \\ b_{21} & b_{22} & \ldots & b_{2n} \\ \vdots & & & \\ b_{n1} & b_{n2} & \ldots & b_{nn} \end{bmatrix}, \text{ and } \vec{\beta} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \end{bmatrix},$$

iterative method (3.2.2) can be written in scalar form

$$\begin{aligned} x_1^{(k)} &= b_{11}x_1^{(k-1)} + \ldots + b_{1n}x_n^{(k-1)} + \beta_1, \\ x_2^{(k)} &= b_{21}x_1^{(k-1)} + \ldots + b_{2n}x_n^{(k-1)} + \beta_2, \\ &\vdots \\ x_n^{(k)} &= b_{n1}x_1^{(k-1)} + \ldots + b_{nn}x_n^{(k-1)} + \beta_n, \end{aligned}$$

where $k = 1, 2, \ldots$.

One can prove (see [1]) that iterative process (3.2.2) converges if all eigenvalues of matrix **B** are by modulus less than one. Taking in account that determination of eigenvalues of matrix is rather complicated, in practical applications of method of simple iteration only sufficient convergence conditions are examined. Namely, for matrix **B** several norms can be defined, as for example,

$$\begin{aligned} ||\mathbf{B}||_1 &= (\sum_{ij} b_{ij}^2)^{1/2}, \\ ||\mathbf{B}||_2 &= \max_i \sum_{j=1}^{n} |b_{ij}|, \\ ||\mathbf{B}||_3 &= \max_j \sum_{i=1}^{n} |b_{ij}|. \end{aligned} \tag{3.2.3}$$

It is not difficult to prove that iterative process (3.2.2) converges if $||B|| < 1$, for arbitrary initial vector $\vec{x}^{(0)}$.

3.3. Gauss-Seidel method

Gauss-Seidel method is constructed by modification of simple iterative method. As we have seen, at simple iteration method, the value of i-th component $x_i^{(k)}$ of vector $\vec{x}^{(k)}$ is obtained from values $x_1^{(k-1)}, \ldots, x_n^{(k-1)}$, i.e.

$$x_i^{(k)} = \sum_{j=1}^{n} b_{ij}x_j^{(k-1)} + \beta_i \quad (i = 1, \ldots, n;\ k = 1, 2, \ldots).$$

This method can be modified in this way so that for computation of $x_i^{(k)}$ are used all previously computed values $x_1^{(k)}, \ldots, x_{i-1}^{(k)}, x_i^{(k-1)}, \ldots, x_n^{(k-1)}$ and the rest will be part of vector, obtained in previous iteration, i.e

$$(3.3.1) \qquad x_i^{(k)} = \sum_{j=1}^{i-1} b_{ij}x_j^{(k)} + \sum_{j=1}^{n} b_{ij}x_j^{(k-1)} + \vec{\beta}_i \quad (i = 1, \ldots, n;\ k = 1, 2, \ldots).$$

Noted modification of simple iterative method is known as Gauss-Seidel method. The iterative process (3.3.1) can be written in matrix form too. Namely, let

$$\mathbf{B} = \mathbf{B}_1 + \mathbf{B}_2,$$

where

$$\mathbf{B}_1 = \begin{bmatrix} 0 & 0 & \ldots & 0 & 0 \\ b_{21} & 0 & & 0 & 0 \\ \vdots & & & & \\ b_{n1} & b_{n2} & & b_{n,n-1} & 0 \end{bmatrix}, \quad \mathbf{B}_2 = \begin{bmatrix} b_{11} & b_{12} & \ldots & b_{1n} \\ 0 & b_{22} & & b_{2n} \\ \vdots & & & \\ 0 & 0 & & b_{nn} \end{bmatrix}.$$

Then (3.3.1) becomes

$$(3.3.2) \qquad \vec{x}^{(k)} = \mathbf{B}_1\vec{x}^{(k)} + \mathbf{B}_2\vec{x}^{(k-1)} + \vec{\beta} \quad (k = 1, 2, \ldots).$$

Theorem 3.3.1. *For arbitrary vector $\vec{x}^{(0)}$, iterative process (3.3.2) converges then and only then if all roots of equation*

$$\det[\mathbf{B}_2 - (\mathbf{I} - \mathbf{B}_1)\lambda] = \begin{bmatrix} b_{11} - \lambda & b_{12} & \ldots & b_{1n} \\ b_{21}\lambda & b_{22} - \lambda & & b_{2n} \\ \vdots & & & \\ b_{n1}\lambda & b_{n2}\lambda & & b_{nn} - \lambda \end{bmatrix} = 0$$

are by modulus less than one.

3.4. Program realization

Program 3.4.1. Let's write a program for solving a system of linear equations of form $\vec{x} = \mathbf{B}\vec{x}+\vec{\beta}$, by simple iteration method. Because this method converges when norm of matrix $\mathbf{B}$ is less than one, for examination of this condition we will write a subroutine `NORMA`, in which, depending on k, are computed norms $(k = 1, 2, 3)$ in accordance with formula (3.2.3). Parameters in list of parameters are of following meaning:
A - matrix stored as vector, which norm is to be calculated;
N - order of matrix;
K - number which defines norm (K=1,2,3);
ANOR - corresponding norm of matrix A.

```
      SUBROUTINE NORMA(A,N,K,ANOR)
      DIMENSION A(1)
      NU=N*N
      ANOR=0
      GO TO (10, 20,40),K
   10 DO 15 I=1,NU
```

```
15  ANOR=ANOR+A(I)**2
    ANOR=SQRT(ANOR)
    RETURN
20  DO 25 I=1,N
    L=-N
    S=0.
    DO 30 J=1,N
    L=L+N
    IA=L+I
30  S=S+ABS(A(IA))
    IF(ANOR-S) 35,25,25
35  ANOR=S
25  CONTINUE
    RETURN
40  L=-N
    DO 50 J=1,N
    S=0.
    L=L+N
    DO 45 I=1,N
    LI=L+I
45  S=S+ABS(A(LI))
    IF(ANOR-S) 55,50,50
55  ANOR=S
50  CONTINUE
    RETURN
    END
```

Main program is organized in this way that before iteration process begins, the convergence is examined. Namely, if at least one norm satisfies $\|\mathbf{B}\|_k < 1$ $(k = 1, 2, 3)$, iterative process proceeds, and if not, the message that convergence conditions are not fulfilled is printed and program terminates.

For multiplying matrix **B** by vector $\vec{x}^{(k+1)}$ we use subroutine **MMAT**, which is given in 2.2.5.2. As initial vector we take $\vec{x}^{(0)}$.

As criteria for termination of iterative process we adopted

$$|x_i^{(k)} - x_i^{(k-1)}| \le \varepsilon \quad (i = 1, \ldots, n).$$

On output we print the last iteration which fulfills above given criteria.

```
    DIMENSION B(100), BETA(10), X(10), X1(10)
    OPEN(8,FILE='ITER.IN')
    OPEN(5,FILE='ITER.OUT')
    READ(8,5) N, EPS
  5 FORMAT(I2,E5.0)
    NN=N*N
    READ(8,10)(B(I),I=1,NN),(BETA(I),I=1,N)
 10 FORMAT(16F5.1)
    WRITE(5,13)
 13 FORMAT(1H1,5X,'MATRICA B', 24X,'VEKTOR BETA')
    DO 15 I=1,N
 15 WRITE(5,20) (B(J),J=I,NN,N),BETA(I)
 20 FORMAT(/2X,4F8.1,5X,F8.1)
    DO 30 K=1,3
    CALL NORMA(B,N,K,ANOR)
    IF(ANOR-1.) 25,30,30
 30 CONTINUE
    WRITE(5,35)
 35 FORMAT(5X,'USLOVI ZA KONVERGENCIJU'
   1' NISU ZADOVOLJENI')
    GO TO 75
 25 ITER=0
    DO 40 I=1,N
 40 X(I)=BETA(I)
 62 ITER=ITER+1
    CALL MMAT(B,X,X1,N,N,1)
    DO 45 I=1,N
 45 X1(I)=X1(I)+BETA(I)
    DO 55 I=1,N
    IF(ABS(X1(I)-X(I))-EPS)55,55,60
 55 CONTINUE
    WRITE(5,42)ITER
 42 FORMAT(/3X,I3,'.ITERACIJA'/)
    WRITE(5,50)(I,X1(I),I=1,N)
 50 FORMAT(3X,4(1X,'X(',I2,')=',F9.5))
    GO TO 75
 60 DO 65 I=1,N
 65 X(I)=X1(I)
    GO TO 62
 75 CLOSE(8)
    CLOSE(5)
    STOP
    END
```

Taking accuracy $\varepsilon = 10^{-5}$, for one concrete system of equation of fourth degree (see output listing) we get the solution in fourteenth iteration.

```
   MATRICA B                                  VEKTOR BETA
   -.1       .4       .1       .1                 .7
    .4      -.1       .1       .1                 .7
    .1       .1      -.2       .2                1.2
    .1       .1       .2      -.2               -1.6
  14.ITERACIJA
 X( 1)=  1.00000 X( 2)=  1.00000 X( 3)=  1.00000
 X( 4)= -1.00000
```

Program 3.4.2. Write a code for obtaining a matrix $\mathbf{S} = e^{\mathbf{A}}$ where $\mathbf{A}$ is given square matrix of order n, by using formula

$$e^{\mathbf{A}} = \sum_{k=0}^{+\infty} \frac{1}{k!}\mathbf{A}^k. \tag{3.4.2.1}$$

Let S_k be k-th partial sum of series (3.4.2.1), and U_k its general member. Then the equalities

$$U_k = \frac{1}{k}U_{k-1}\mathbf{A}, \quad S_k = S_{k-1} + U_k \quad (k = 1, 2, \ldots) \tag{3.4.2.2}$$

hold, whereby $\mathbf{U}_0 = \mathbf{S}_0 = \mathbf{I}$ (unity matrix of order n). By using equality (3.4.2.2) one can write a program for summation of series (3.4.2.1), where we usually take as criteria for termination of summation the case when norm of matrix is lesser than in advance given small positive number ε. In our case we will take norm $\|.\|_2$ (see formula (3.2.3)) and $\varepsilon = 10^{-5}$.

By using subroutine MMAT for matrices multiplication and subroutine NORMA for calculation of matrix norm, we have written the following program for obtaining the matrix $e^{\mathbf{A}}$

```
C=====================================================
C   ODREDJIVANJE MATRICE EXP(A)
C=====================================================
      DIMENSION A(100), S(100), U(100), P(100)
      OPEN(8,FILE='EXPA.IN')
      OPEN(5,FILE='EXPA.OUT')
      READ(8,10) N,EPS
   10 FORMAT(I2,E5.0)
      NN=N*N
      READ(8,15)(A(I),I=1,NN)
   15 FORMAT(16F5.0)
C   FORMIRANJE JEDINICNE MATRICE
      DO 20 I=1,NN
      S(I)=0.
   20 U(I)=0.
      N1=N+1
      DO 25 I=1,NN,N1
      S(I)=1.
  25  U(I)=1.
C SUMIRANJE MATRICNOG REDA
      K=0
  30  K=K+1
      CALL MMAT(U,A,P,N,N,N)
      B=1./K
      DO 35 I=1,NN
      U(I)=B*P(I)
   35 S(I)=S(I)+U(I)
C ISPITIVANJE USLOVA ZA PREKID SUMIRANJA
      CALL NORMA(U,N,2,ANOR)
      IF(ANOR.GT.EPS)GO TO 30
      WRITE(5,40) ((A(I),I=J,NN,N),J=1,N)
   40 FORMAT(2X,<5*N-9>X,'M A T R I C A       A'
     1 //(<N>F10.5))
      WRITE(5,45)((S(I),I=J,NN,N),J=1,N)
   45 FORMAT(//<5*n-9>X,'M A T R I C A    EXP(A)'
     1 //(<N>F10.5))
      CLOSE(8)
      CLOSE(5)
      END
```

This program has been tested on the example

$$\mathbf{A} = \begin{bmatrix} 4 & 3 & -3 \\ 2 & 3 & -2 \\ 4 & 4 & -3 \end{bmatrix},$$

for which can be obtained analytically

(3.4.2.3) $$\mathbf{A} = \begin{bmatrix} 3e-2 & 3e-3 & -3e+3 \\ 2e-2 & 2e-1 & -2e+2 \\ 4e-4 & 4e-4 & -4e+5 \end{bmatrix}.$$

Output listing is of form

```
          M A T R I C A       A
   4.00000
   3.00000
  -3.00000
   2.00000
   3.00000
  -2.00000
   4.00000
   4.00000
  -3.00000
         M A T R I C A   EXP(A)
  16.73060
  14.01232
 -14.01232
   9.34155
  12.05983
  -9.34155
  18.68310
  18.68310
 -15.96482
```

By using (3.4.2.3) it is not hard to prove that all figures in obtained results are exact.

It is suggested to readers to write a code for previous problem using program Mathematica.

3.5. Packages for systems of linear algebraic equations

For many computer-early years ago (late sixties and early seventies of previous century) the most popular program (at least at Niš University), linear equations solver, was SIMQ from SSP (Scientific Subroutine Package) by IBM corporation.

Nowadays, in many cases you will have no alternative but to use sophisticated black-box program packages. Several good ones are available. LINPACK was developed at Argonne National Laboratories and deserves particular mention because it is published, documented, and available for free use. A successor to LINPACK, LAPACK, is becoming available. Packages available commercially include those in the IMSL and NAG libraries. One should keep in mind that the sophisticated packages are designed with very large linear systems in mind. They therefore go to great effort to minimize not only the number of operations, but also the required storage. Routines for the various tasks are usually provided in several versions, corresponding to several possible simplifications in the form of the input coefficient matrix: symmetric, triangular, banded, positive definite, etc. If one has a large matrix in one of these forms, he should certainly take advantage of the increased efficiency provided by these different routines, and not just use the form provided for general matrices. There is also a great watershed dividing routines that are direct (i.e., execute in a predictable number of operations) from routines that are iterative (i.e., attempt to converge to the desired answer in however many steps are necessary). Iterative methods become preferable when the battle against loss of significance is in danger of being lost, either due to large n or because the problem is close to singular. Very interesting techniques are those the borderline between direct and iterative methods, namely the iterative improvement of a solution that has been obtained by direct methods.

Many commercial software packages contain solvers for systems of linear algebraic equations. Some of the more prominent packages are Matlab and Mathcad. The spreadsheet Excel can also be used to solve systems of equations. More sophisticated packages, such as Mathematica, Macsyma, and Maple also contain linear equations solvers.

The book *Numerical Recipes* [4] contains several subroutines for solving systems of linear algebraic equations. Some algorithms, from which some are codded, are given in book *Numerical Methods for Engineers and Scientists* [3] (see Chapter 1).

Some general guidelines for selecting a method for solving systems of linear algebraic equations are given as follows.

- Direct elimination methods are preferred for small systems ($n \leq 50$ to 100) and systems with few zeros (nonsparse systems). Gauss elimination is the method of choice.
- For tridiagonal systems, the Thomas algorithm is the method of choice ([3], Chapter 1).
- LU factorization methods are the methods of choice when more than one vectors $\vec{b}$ must be considered.
- For large systems that are not diagonally dominant, the round-off errors can be large.
- Iterative methods are preferred for large, sparse matrices that are diagonally dominant. The SOR (Successive-Over-Relaxation) method is recommended. Numerical experimentation to find the optimum over-relaxation factor ω is usually worthwhile if the system of equations is to be solved for many vectors $\vec{b}$.

Bibliography (Cited references and further reading)

[1] Milovanović, G.V., *Numerical Analysis I*, Naučna knjiga, Beograd, 1988 (Serbian).

[2] Milovanović, G.V. and Djordjević, Dj.R., *Programiranje numeričkih metoda na FORTRAN jeziku.* Institut za dokumentaciju zaštite na radu "Edvard Kardelj", Niš, 1981 (Serbian).

[3] Hoffman, J.D., *Numerical Methods for Engineers and Scientists.* Taylor & Francis, Boca Raton-London-New York-Singapore, 2001.

[4] Press, W.H., Flannery, B.P., Teukolsky, S.A., and Vetterling, W.T., *Numerical Recepies - The Art of Scientific Computing.* Cambridge University Press, 1989.

[5] Albrecht, J., *Fehlerabschätzungen bei Relaxationsverfahren zur numerischen Auflösung linearer Gleichungsysteme.* Numer. Math. 3(1961), 188-201.

[6] Apostolatos, N. und Kulisch, U., *Über die Konvergenz des Relaxionsvrfahrens bei nicht-negativen und diagonaldominanten Matrizen.* Computing 2(1967), 17-24.

[7] Golub, G.H. & Varga, R.S., *Chebyshev semi-iterative methods, successive overrelaxation iterative methods and second order Richardson iterative methods.* Numer. Math. 3(1961), 147-168.

[8] Soutwell, R.V., *Relaxation Methods in Theoretical Physics.* 2 vols. Oxford Univ. Press. Fair Lawn., New Jersey, 1956.

[9] Varga, R.S., *Matrix Iterative Analysis.* Prentice Hall, Englewood Cliffs, New Jersey, 1962.

[10] Young, D.M., *Iterative Solution of Large Systems.* Academic Press, New York, 1971.

[11] Ralston,A., *A First Course in Numerical Analysis.* McGraw-Hill, New York, 1965.

[12] Hildebrand, F.B., *Introduction to Numerical Analysis.* Mc.Graw-Hill, New York, 1974.

[13] Acton, F.S., *Numerical Methods That Work* (corrected edition). Mathematical Association of America, Washington, D.C., 1990.

[14] Abramowitz, M., and Stegun, I.A., *Handbook of Mathematical Functions.* National Bureau of Standards, Applied Mathematics Series, Washington, 1964 (reprinted 1968 by Dover Publications, New York).

[15] Rice, J.R., *Numerical Methods, Software, and Analysis.* McGraw-Hill, New York, 1983.

[16] Forsythe, G.E., Malcolm, M.A., and Moler, C.B., *Computer Methods for Mathematical Computations.* Englewood Cliffs, Prentice-Hall, NJ, 1977.

[17] Forsythe, G.E., *Solving linear algebraic equations can be interesting.* Bull. Amer. Math. Soc. 59(1953), 299-329.

[18] Forsythe & Moler, C.B., *Computer Solution of Linear Algebraic Systems.* Prentice-Hall. Englewood Cliffs, NJ, 1967.

[19] Kahaner, D., Moler, C., and Nash, S., 1989, *Numerical Methods and Software.* Englewood Cliffs, Prentice Hall, NJ, 1989.

[20] Hamming, R.W., *Numerical Methods for Engineers and Scientists.* Dover, New York, 1962 (reprinted 1986).

[21] Ferziger, J.H., *Numerical Methods for Engineering Applications.* Stanford University, John Willey & Sons, Inc., New York, 1998

[22] Pearson, C.E., *Numerical Methods in Engineering and Science.* University of Washington, Van Nostrand Reinhold Company, New York, 1986.

[23] Stephenson, G. and Radmore, P.M., *Advanced Mathematical Methods for Engineering and Science Students.* Imperial College London, University College, London Cambridge Univ. Press, 1999

[24] Milovanović, G.V. and Kovačević, M.A., *Zbirka rešenih zadataka iz numeričke analize.* Naučna knjiga, Beograd, 1985. (Serbian).

[25] *IMSL Math/Library Users Manual* , IMSL Inc., 2500 City West Boulevard, Houston TX 77042

[26] *NAG Fortran Library*, Numerical Algorithms Group, 256 Banbury Road, Oxford OX27DE, U.K.

[27] Dongarra, J.J., et al., *LINPACK Users Guide.* S.I.A.M., Philadelphia, 1979.

[28] Golub, G.H., and Van Loan,C.F., *Matrix Computations, 2nd ed.* (Baltimore: Johns Hopkins University Press), 1989.

[29] Gill, P.E., Murray, W., and Wright, M.H., *Numerical Linear Algebra and Optimization, vol. 1.* Addison-Wesley, Redwood City, CA, 1991.

[30] Stoer, J., and Bulirsch, R., *Introduction to Numerical Analysis.* Springer-Verlag, New York, 1980.

[31] Coleman, T.F.,and Van Loan,C., *Handbook for Matrix Computations.* S.I.A.M., Philadelphia, 1988.

[32] Wilkinson, J.H., and Reinsch, C. , *Linear Algebra, vol. II of Handbook for Automatic Computation.* Springer-Verlag, New York, 1971.

[33] Westlake, J.R. *A Handbook of Numerical Matrix Inversion and Solution of Linear Equations.* Wiley, New York, 1968.

[34] Johnson, L.W., and Riess, R.D., *Numerical Analysis, 2nd ed.* Addison- Wesley, Reading, MA, 1982.

[35] Ralston, A., and Rabinowitz, P., *A First Course in Numerical Analysis, 2nd ed.* McGraw-Hill, New York, 1978.

[36] Isaacson, E., and Keller, H.B., *Analysis of Numerical Methods.* Wiley, New York, 1966.

[37] Horn,R.A., and Johnson, C.R., *Matrix Analysis.* Cambridge University Press, Cambridge, 1985.

LECTURES

LESSON IV

4. Eigenvalue Problems

4.1. Introduction

An $n \times n$ matrix $\mathbf{A}$ is said to have an *eigenvector* $\vec{\mathbf{x}}$ and corresponding *eigenvalue* λ if

$$\mathbf{A} \cdot \vec{\mathbf{x}} = \lambda \vec{\mathbf{x}} \tag{4.1.1}$$

Obviously any multiple of an eigenvector $\vec{\mathbf{x}}$ will also be an eigenvector, but we won't consider such multiples as being distinct eigenvectors. (The zero vector is not considered to be an eigenvector at all). Evidently (4.1.1) can hold only if

$$\det |\mathbf{A} - \lambda \mathbf{I}| = \mathbf{0}, \tag{4.1.2}$$

which, if expanded out, is an nth degree polynomial in λ whose roots are the eigenvalues. This proves that there are always n (not necessarily distinct) eigenvalues. Equal eigenvalues coming from multiple roots are called *degenerate*. Root searching in the characteristic equation (4.1.2) is usually a very poor computational method for finding eigenvalues (see [2], pp. 449-453).

The above two equations also prove that every one of the n eigenvalues has a (not necessarily distinct) corresponding eigenvector: If λ is set to an eigenvalue, then the matrix $\mathbf{A} - \lambda \mathbf{I}$ is singular, and we know that every singular matrix has at least one nonzero vector in its null-space (consider singular value decomposition).

If you add $\tau \vec{x}$ to both sides of (4.1.1), you will easily see that the eigenvalues of any matrix can be changed or shifted by an additive constant τ by adding to the matrix that constant times the identity matrix. The eigenvectors are unchanged by this shift. Shifting, as we will see, is an inportant part of many algorithms for computing eigenvalues. We see also that there is no special significance to a zero eigenvalue. Any eigenvalue can be shifted to zero, or any zero eigenvalue can be shifted away from zero.

Definitions

A matrix is called symmetric if it is equal to its transpose,

$$\mathbf{A} = \mathbf{A}^T \quad \text{or} \quad a_{ij} = a_{ji}. \tag{4.1.3}$$

It is called Hermitian or self-adjoint if it equals to the complex-conjugate of its transpose (its Hermitian conjugate, denoted by "†")

$$\mathbf{A} = \mathbf{A}^\dagger \quad \text{or} \quad a_{ij} = a_{ji}^*. \tag{4.1.4}$$

It is termed *orthogonal* if its transpose equals its inverse

$$\mathbf{A}^T \cdot \mathbf{A} = \mathbf{A} \cdot \mathbf{A}^T = I \tag{4.1.5}$$

and *unitary* if its Hermitian conjugate equals its inverse. Finally, a matrix is called *normal* if it *commutes* with its Hermitian conjugate,

$$\mathbf{A} \cdot \mathbf{A}^\dagger = \mathbf{A}^\dagger \cdot \mathbf{A}. \tag{4.1.6}$$

For real matrices, Hermitian means the same as symmetric, unitary means the same as orthogonal, and both of these distinct classes are normal.

The reason that "Hermitian" is an important concept has to do with eigenvalues. The eigenvalues of a Hermitian matrix are all real. In particular, the eigenvalues of a real symmetric matrix are all real. Contrariwise, the eigenvalues of a real nonsymmetric matrix may include real values, but may also include pairs of conjugate values; and the eigenvalues of a complex matrix that is not Hermitian will in general be complex.

The reason that "normal" is an important concept has to do with the eigenvectors. The eigenvectors of a normal matrix with non-degenerate (i.e., distinct) eigenvalues are complete and orthogonal, spanning the n-dimensional vector space. For a normal matrix with degenerate eigenvalues, we have the additional freedom of replacing the eigenvectors corresponding to a degenerate eigenvalue by linear combinations of themselves. Using this freedom, we can always perform Gramm-Schmidt orthogonalization and find a set of eigenvectors that are complete and orthogonal, just as in the non-degenerate case. The matrix whose columns are an orthonormal set of eigenvectors is evidently unitary. A special case is that the matrix of eigenvectors of a real symmetric matrix is orthogonal, since the eigenvectors of that matrix are all real.

When a matrix is not normal, as typified by any random, nonsymmetric, real matrix, then in general we cannot find any orthonormal set of eigenvectors, nor even any pairs of eigenvectors that are orthogonal (except perhaps by rare chance). While the n non-orthonormal eigenvectors will "usually" span the n-dimensional vector space, they do not always do so; that is, the eigenvectors are not always complete. Such a matrix is said to be defective.

Left and Right Eigenvectors

While the eigenvectors of a non-normal matrix are not particularly orthogonal among themselves, they do have an orthogonality relation with a different set of vectors, which we must now define. Up to now our eigenvectors have been column vectors that are multiplied to the right of a matrix $\mathbf{A}$, as in (4.1.1). These, more explicitly, are termed *right eigenvectors*. We could also, however, try to find row vectors, which multiply $\mathbf{A}$ to the left and satisfy

$$\vec{x} \cdot \mathbf{A} = \lambda \vec{x}. \tag{4.1.7}$$

These are called *left eigenvectors*. By taking the transpose of (4.1.7), one can see that every left eigenvector is the transpose of a right eigenvector of the transpose of $\mathbf{A}$. Now by comparing to (4.1.2), and using the fact that the determinant of a matrix equals the determinant of its transpose, we also see that the left and right eigenvalues of $\mathbf{A}$ are identical.

If the matrix $\mathbf{A}$ is symmetric, then the left and right eigenvectors are just transposes of each other, that is, have the same numerical values as components. Likewise, if the matrix is self-adjoint, the left and right eigenvectors are Hermitian conjugates of each other. For the general non-normal case, however, we have the following calculation: Let $\mathbf{X}_R$ be the matrix formed by columns from the right eigenvectors, and $\mathbf{X}_L$ be the matrix formed by rows from the left eigenvectors. Then (4.1.1) and (4.1.7) can be rewritten as

$$\mathbf{A} \cdot \mathbf{X}_R = \mathbf{X}_R \cdot \mathrm{diag}(\lambda_1 \ldots \lambda_n); \qquad \mathbf{X}_L \cdot \mathbf{A} = \mathrm{diag}(\lambda_1 \ldots \lambda_n) \cdot \mathbf{X}_L. \tag{4.1.8}$$

Multiplying the first of these equations on the left by $\mathbf{X}_L$, the second on the right by $\mathbf{X}_R$, and subtracting the two, gives

$$(\mathbf{X}_L \cdot \mathbf{X}_R) \cdot \mathrm{diag}(\lambda_1 \dots \lambda_n) = \mathrm{diag}(\lambda_1 \dots \lambda_n) \cdot (\mathbf{X}_L \cdot \mathbf{X}_R). \tag{4.1.9}$$

This says that the matrix of dot products of the left and right eigenvectors commutes with the diagonal matrix of eigenvalues. But the only matrices that commute with a diagonal matrix of distinct elements are themselves diagonal. Thus, if the eigenvalues are non-degenerate, each left eigenvector is orthogonal to all right eigenvectors except its corresponding one, and vice versa. By choice of normalization, the dot products of corresponding left and right eigenvectors can always be made unity for any matrix with non-degenerate eigenvalues.

If some eigenvalues are degenerate, then either the left or the right eigenvectors corresponding to a degenerate eigenvalue must be linearly combined among themselves to achieve orthogonality with the right or left ones, respectively. This can always be done by a procedure akin to Gram-Schmidt orthogonalization. The normalization can then be adjusted to give unity for the nonzero dot products between corresponding left and right eigenvectors. If the dot product of corresponding left and right eigenvectors is zero at this stage, then you have a case where the eigenvectors are incomplete. Note that incomplete eigenvectors can occur only where there are degenerate eigenvalues, but do not always occur in such cases (in fact, never occur for the class of "normal" matrices).

In both the degenerate and non-degenerate cases, the final normalization to unity of all nonzero dot products produces the result: The matrix whose rows are left eigenvectors is the inverse matrix of the matrix whose columns are right eigenvectors, *if the inverse exists.*

Diagonalization of a Matrix

Multiplying the first equation in (4.1.8) by $\mathbf{X}_L$, and using the fact that $\mathbf{X}_L$ and $\mathbf{X}_R$ are matrix inverses, we get

$$\mathbf{X}_R^{-1} \cdot \mathbf{A} \cdot \mathbf{X}_R = \mathrm{diag}(\lambda_1 \dots \lambda_n). \tag{4.1.10}$$

This is a particular case of a *similarity transform* of the matrix $\mathbf{A}$,

$$\mathbf{A} \to \mathbf{Z}^{-1} \cdot \mathbf{A} \cdot \mathbf{Z} \tag{4.1.11}$$

for some transformation matrix $\mathbf{Z}$. Similarity transformations play a crucial role in the computation of eigenvalues, because they leave the eigenvalues of a matrix unchanged. This is easily seen from

$$\begin{aligned}\det |\mathbf{Z}^{-1} \cdot \mathbf{A} \cdot \mathbf{Z} - \lambda \mathbf{I}| &= \det |\mathbf{Z}^{-1} \cdot (\mathbf{A} - \lambda \mathbf{I}) \cdot \mathbf{Z}| \\ &= \det |\mathbf{Z}| \ \det |\mathbf{A} - \lambda \mathbf{I}| \ \det |\mathbf{Z}^{-1}| \\ &= \det |\mathbf{A} - \lambda \mathbf{I}|\end{aligned} \tag{4.1.12}$$

Equation (4.1.10) shows that any matrix with complete eigenvectors (which includes all normal matrices and "most" random non-normal ones) can be diagonalized by a similarity transformation, that the columns of the transformation matrix that effects the diagonalization are the right eigenvectors, and that the rows of its inverse are the left eigenvectors.

For real, symmetric matrices, the eigenvectors are real and orthonormal, so the transformation matrix is orthogonal. The similarity transformation is then also an *orthogonal transformation* of the form

$$\mathbf{A} \to \mathbf{Z}^T \cdot \mathbf{A} \cdot \mathbf{Z} \tag{4.1.13}$$

While real nonsymmetric matrices can be diagonalized in their usual case of complete eigenvectors, the transformation matrix is not necessarily real. It turns out, however, that a real similarity transformation can "almost" do the job. It can reduce the matrix down to a form with little two-by-two blocks along the diagonal, all other elements zero. Each two-by-two block corresponds to a complex-conjugate pair of complex eigenvalues.

The "grand strategy" of virtually all modern eigensystem routines is to nudge the matrix $\mathbf{A}$ towards diagonal form by a sequence of similarity transformations,

$$\begin{aligned} \mathbf{A} &\rightarrow \mathbf{P}_1^{-1} \cdot \mathbf{A} \cdot \mathbf{P}_1 \rightarrow \mathbf{P}_2^{-1} \cdot \mathbf{P}_1^{-1} \cdot \mathbf{A} \cdot \mathbf{P}_1 \cdot \mathbf{P}_2 \\ &\rightarrow \mathbf{P}_3^{-1} \cdot \mathbf{P}_2^{-1} \cdot \mathbf{P}_1^{-1} \cdot \mathbf{A} \cdot \mathbf{P}_1 \cdot \mathbf{P}_2 \cdot \mathbf{P}_3 \rightarrow \quad \text{etc.} \end{aligned} \tag{4.1.14}$$

If we get all the way to diagonal form, then the eigenvectors are the columns of the accumulated transformation

$$\mathbf{X}_R = \mathbf{P}_1 \cdot \mathbf{P}_2 \cdot \mathbf{P}_3 \cdot \ldots \tag{4.1.15}$$

Sometimes we do not want to go all the way to diagonal form. For example, if we are interested only in eigenvalues, not eigenvectors, it is enough to transform the matrix $\mathbf{A}$ to be triangular, with all elements below (or above) the diagonal zero. In this case the diagonal elements are already the eigenvalues, as you can see by mentally evaluating (4.1.2) using expansion by minors.

There are two rather different sets of techniques for implementing the strategy (4.1.14). It turns out that they work rather well in combination, so most modern eigensystem routines use both. The first set of techniques constructs individual $\mathbf{P}_i$'s as explicit "atomic" transformations designed to perform specific tasks, for example zeroing a particular off-diagonal element (Jacobi transformation), or a whole particular row or column (Householder transformation, elimination method). In general, a finite sequence of these simple transformations cannot completely diagonalize a matrix. There are then two choices: either use the finite sequence of transformations to go most of the way (e.g., to some special form like *tridiagonal* or *Hessenberg*) and follow up with the second set of techniques about to be mentioned; or else iterate the finite sequence of simple transformations over and over until the deviation of the matrix from diagonal is negligibly small. This latter approach is conceptually simplest. However, for n greater than ~ 10, it is computationally inefficient by a roughly constant factor ~ 5.

The second set of techniques, called *factorization methods*, is more subtle. Suppose that the matrix $\mathbf{A}$ can be factored into a left factor $\mathbf{F}_L$ and a right factor $\mathbf{F}_R$. Then

$$\mathbf{A} = \mathbf{F}_L \cdot \mathbf{F}_R \quad \text{or equivalently} \quad \mathbf{F}_L^{-1} \cdot \mathbf{A} = \mathbf{F}_R \tag{4.1.16}$$

If we now multiply back together the factors in the reverse order, and use the second equation in (4.1.16) we get

$$\mathbf{F}_R \cdot \mathbf{F}_L = \mathbf{F}_L^{-1} \cdot \mathbf{A} \cdot \mathbf{F}_L \tag{4.1.17}$$

which we recognize as having effected a similarity transformation on $\mathbf{A}$ with the transformation matrix being $\mathbf{F}_L$. The **QR** method which exploits this idea will be explained later.

Factorization methods also do not converge exactly in a finite number of transformations. But the better ones do converge rapidly and reliably, and, when following an appropriate initial reduction by simple similarity transformations, they are the methods of choice. The presented considerations are very important for those dealing with dynamics of construction and seismic engineering, especial in the phase of modelling and dynamic response computation.

Definitions and theorems regarding eigenvalue problem

For further considerations we need some theorems and definitions, as follows (see [1], pp. 211-213).

Definition 4.1.1. *Let $\mathbf{A} = [a_{ij}]$ be complex square matrix of order n. Every vector $\vec{f} \in \mathbf{C}^n$, which is different from zero-vector, is called eigenvector of matrix $\mathbf{A}$ if exists scalar $\lambda \in \mathbf{C}$ such that holds (4.1.1). Scalar λ in (4.1.1) is called corresponding eigenvalue. Having in mind that (4.1.1) can be presented in the form*

$$(\mathbf{A} - \lambda\mathbf{I})\vec{x} = \vec{0},$$

we conclude that equation (4.1.1) has non-trivial solutions (in $\vec{x}$) then and only then if holds (4.1.2).

Definition 4.1.2. *If $\mathbf{A}$ is square matrix, then polynomial $\lambda \to P(\lambda) = \det(\mathbf{A} - \lambda\mathbf{I})$ is called characteristic polynomial, and corresponding equation $P(\lambda) = 0$ its characteristic equation.*

Let $\mathbf{A} = [a_{ij}]_{n\times n}$. The characteristic polynomial can be expressed in the form

$$P(\lambda) = \begin{vmatrix} a_{11}-\lambda & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22}-\lambda & & a_{2n} \\ \vdots & & & \\ a_{n1} & a_{n2} & & a_{nn}-\lambda \end{vmatrix}$$

or

$$P(\lambda) = (-1)^n(\lambda^n - p_1\lambda^{n-1} + p_2\lambda^{n-2} - \cdots + (-1)^{n-1}p_{n-1}\lambda + (-1)^n p_n),$$

where p_k is sum of all principal minors of order k of determinant of matrix $\mathbf{A}$, i.e.

$$p_k = \sum_{1\le i_1<i_2<\cdots<i_k\le n} \det(\mathbf{A}^{i_1 i_2\cdots i_k}_{i_1 i_2\cdots i_k}).$$

Note that

$$p_1 = \sum_{i=1}^{n} a_{ii} = \text{tr}\ \mathbf{A} \quad \text{and} \quad p_n = \det(\mathbf{A}).$$

Often, in place of characteristic polynomial P is used so known normed characteristic polynomial H, defined by

$$H(\lambda) = (-1)^n P(\lambda) = \lambda^n - p_1\lambda^{n-1} + p_2\lambda^{n-2} - \cdots + (-1)^n p_n.$$

Eigenvalues of matrix $\mathbf{A}$ (i.e. zeros of polynomial P) $\lambda_i (i = 1, \dots, n)$ will be denoted as $\lambda_i(\mathbf{A})$.

Definition 4.1.3. *The set of all eigenvalues of square matrix $\mathbf{A}$ is called spectrum of that matrix and denoted with $Sp(\mathbf{A})$.*

Definition 4.1.4. *Spectral radius $\rho(\mathbf{A})$ of square matrix $\mathbf{A}$ is number*

$$\rho(\mathbf{A}) = \max_i |\lambda_i(\mathbf{A})|.$$

Theorem 4.1.1. *Every matrix is, in matrix sense, null of its characteristic polynomial.*

This theorem is known as Cayley-Hamilton theorem.

Theorem 4.1.2. *Let $\lambda_1, \dots, \lambda_n$ be eigenvalues of matrix $A = [a_{ij}]$ of order n and $x \to Q(x)$ scalar polynomial of degree m. Then*

$$Q(\lambda_1), \dots, Q(\lambda_n)$$

are eigenvalues of matrix $\mathbf{Q}(\mathbf{A})$.

Theorem 4.1.3. *Let $\lambda_1, \dots, \lambda_n$ be eigenvalues of regular matrix $\mathbf{A}$ of order n. Then*

$$\lambda_1^{-1}, \dots, \lambda_n^{-1}$$

are eigenvalues of matrix $\mathbf{A}^{-1}$.

Theorem 4.1.4. *Eigenvalues of triangular matrix are equal to diagonal elements.*

The following theorem gives recursive procedure for obtaining characteristic polynomial of tridiagonal matrix.

Theorem 4.1.5. *Let*

$$\mathbf{A}_k = \begin{bmatrix} b_1 & c_1 & 0 & \dots & 0 \\ a_2 & b_2 & c_2 & & 0 \\ \vdots & & & & \\ 0 & 0 & 0 & & b_k \end{bmatrix} \quad \textit{and} \quad H_k(\lambda) = (-1)^k \det(\mathbf{A}_k - \lambda \mathbf{I}).$$

Normed characteristic polynomial $\lambda \to H(\lambda)$ $(= H_n(\lambda))$ of matrix $\mathbf{A}(= \mathbf{A}_n)$ is to be obtained by recursive procedure

$$H_k(\lambda) = (\lambda - b_k)H_{k-1}(\lambda) - a_{k-1}c_{k-1}H_{k-2}(\lambda) \quad (k = 2, \dots, n),$$

where $H_0(\lambda) = 1$ and $H_1(\lambda) = \lambda - b_1$.

Definition 4.1.5. *For matrix $\mathbf{B}$ one says to be similar to matrix $\mathbf{A}$ if there exists at least one regular matrix $\mathbf{C}$ such that*

$$\mathbf{B} = \mathbf{C}^{-1}\mathbf{A}\mathbf{C}.$$

Theorem 4.1.6. *Similar matrices have identical characteristic polynomials, and therewith identical eigenvalues.*

4.2. Localization of Eigenvalues

A lot of problems reduce to eigenvalue problem. Here we will give some results regarding localization of eigenvalues in complex space (see [1], pp. 290-292).

Theorem 4.2.1. *(Gershgorin). Let $A = [a_{ij}]_{n \times n}$ square matrix of order n and C_i $(i = 1, \dots n)$ discs in complex space with centers in a_{ii} and radiuses $r_i = \sum\limits_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}|$, i.e.*

$$C_i = \{z \mid |z - a_{ii}| \leq r_i\} \quad (i = 1, \dots, n).$$

If we denote with C union of these discs, then all eigenvalues of matrix $\mathbf{A}$ are in C.

Remark 4.2.1. *Regarding fact that matrix $\mathbf{A}^T$ has same eigenvalues as matrix $\mathbf{A}$, on the basis of previous theorem one can conclude that all eigenvalues of matrix $\mathbf{A}$ are located in the union of D discs*

$$D_j = \{z \mid |z - a_{jj}| \leq s_j\} \quad (j = 1, \dots, n),$$

where $s_j = \sum\limits_{\substack{i=1 \\ i \neq j}}^{n} |a_{ij}|$.

Based on previous one concludes that all eigenvalues of matrix $\mathbf{A}$ lie in the cut of sets C and D.

Theorem 4.2.2. *If m discs from Theorem 4.2.1. form connected area which is isolated from other discs, then exact m eigenvalues of matrix A are located in this area.*

The proof of this theorem could be found in extraordinary monograph of Wilkinson [7].

Example 4.2.1.

Take matrix

$$\mathbf{A} = \begin{bmatrix} 1 & 0.1 & -0.1 \\ 0 & 2 & 0.4 \\ -0.2 & 0 & 3 \end{bmatrix}.$$

Based on theorem 4.2.1 eigenvalues are located in discs

$$C_1 = \{z \mid |z-1| \le 0.2\}, \quad C_2 = \{z \mid |z-2| \le 0.4\}, \quad C_3 = \{z \mid |z-3| \le 0.2\}.$$

Note that, based on remark 4.2.1., it follows that discs D_1, D_2, D_3 have radiuses $0.2, 0.1, 0.5$, respectively. By the way, the exact values of eigenvalues, given with seven figures, are $\lambda_1 = 0.9861505$, $\lambda_2 = 2.0078436, \lambda_3 = 3.0060058$, and normed characteristic polynomial is

$$H(\lambda) = \lambda^3 - 6\lambda^2 + 10.98\lambda - 5.952.$$

Theorem on localization of eigenvalues has theoretical and practical importance (for example, for determining initial values at iterative methods, for analysis at perturbation problems, etc).

For determining eigenvalues there are a lot of methods, whereby some of them enable finding of all eigenvalues, and others only some of them, for example, dominating ones, i.e. with maximum modulus. Some of methods perform only determination of coefficients of characteristic polynomial, so that some of methods for solution of algebraic equations have to be used (see Chapter 5). Such approach is not recommended, being in most cases numerically unstable, i.e. ill-conditioned. Namely, because the coefficients of characteristic polynomials are, in general, subjects to round-off error, due to ill-conditioning of characteristic polynomials, the big errors in eigenvalues occur.

4.3. Methods for dominant eigenvalues

Very often, in many applications (i.e. in dynamic of constructions), one needs only maximal (by module) eigenvalue and corresponding eigenvector.

Let $\lambda_1, \ldots, \lambda_n$ be eigenvalues and $\vec{x}_1, \ldots, \vec{x}_n$ corresponding eigenvectors of matrix $\mathbf{A} = [a_{ij}]_{n\times n}$. If

$$|\lambda_1| = \ldots = |\lambda_r| > |\lambda_{r+1}| \ge \cdots \ge |\lambda_n|$$

we say that $\lambda_1, \ldots, \lambda_r$ are dominant eigenvalues of matrix $\mathbf{A}$. In this section we will consider a method for determination of dominant eigenvalue and corresponding eigenvector, as well as some modifications of this method. We suppose that eigenvectors are linearly independent, forming a basis in $\mathcal{R}^n$. Therefore, the arbitrary non-zero vector $\vec{v}_0$ can be expressed as

(4.3.1)
$$\vec{v}_0 = \sum_{i=1}^{n} \alpha_i \vec{x}_i,$$

where α_i some scalars. Define now the iterative process

$$\vec{v}_k = \mathbf{A}\vec{v}_{k-1} \quad (k = 1, 2, \ldots).$$

Then

$$\vec{v}_k = \mathbf{A}\vec{v}_{k-1} = \mathbf{A}^2\vec{v}_{k-2} = \ldots = \mathbf{A}^k\vec{v}_0 = \sum_{i=1}^{n} \alpha_i \mathbf{A}^k \vec{x}_i,$$

or, regarding to (4.3.1) and assertion of Theorem 4.1.2,

(4.3.2)
$$\vec{v}_k = \sum_{i=1}^{n} \alpha_i \lambda_i^k \vec{x}_i.$$

The special interesting case here is when one dominant eigenvalue λ_1 ($r = 1$) exists. Assuming $\alpha_1 \neq 0$, on the basis of (4.3.2) we have

$$\vec{v}_k = \alpha_1\lambda_1^k(\vec{x}_1 + \sum_{i=2}^{n} \frac{\alpha_i}{\alpha_1}(\frac{\lambda_i}{\lambda_1})^k\vec{x}_i) = \alpha_1\lambda_1^k(\vec{x}_1 + \vec{\epsilon}_k),$$

where $\vec{\epsilon}_k \to 0$, when $k \to +\infty$.

Introduce now notation $(\vec{y})_i$ for i-th coordinate of some vector $\vec{y}$. Then i-th coordinate of vector $\vec{v}_k$ is

$$(\vec{v}_k)_i = \alpha_1\lambda_1^k((\vec{x}_1)_i + (\vec{\epsilon}_k)_i).$$

Because of

$$\vec{v}_{k+1} = \alpha_1\lambda^{k+1}(\vec{x}_1 + \vec{\epsilon}_{k+1}),$$

based on previous, for every i ($1 \leq i \leq n$) we have

$$\frac{(\vec{v}_{k+1})_i}{(\vec{v}_k)_i} = \lambda_1\frac{(\vec{x}_1)_i + (\vec{\epsilon}_{k+1})_i}{(\vec{x}_1)_i + (\vec{\epsilon}_k)_i} \to \lambda_1 \quad (k \to +\infty).$$

Based on this fact, the method for determination of dominant eigenvalue λ_1, known as power method, can be formulated. Vector $\vec{v}_k$ is thereby an approximation of non-normed eigenvector which corresponds to dominant eigenvalue*. By practical realization of this method the norming of eigenvector is performed, i.e. of vector $\vec{v}_k$ after every iteration step. Norm-setting is performed by dividing vector $\vec{v}_k$ by its coordinate with maximal module. So, power method can be expressed by

$$\vec{z}_k = \mathbf{A}\vec{v}_{k-1}, \ \vec{v}_k = \vec{z}_k/\gamma_k,$$

where γ_k is coordinate of vector $\vec{z}_k$ with maximal module, i.e., $\gamma_k = (\vec{z}_k)_i$ and $|(\vec{z}_k)_i| = ||\vec{z}_k||$. Note that $\gamma_k \to \lambda_1$ and $\vec{v}_k \to \dfrac{\vec{x}_1}{||\vec{x}_1||_\infty}$, when $k \to +\infty$.

Speed of convergence of this method depends on ratio $|\lambda_1/\lambda_2|$. Namely, it holds

$$|\lambda_1 - \gamma_k| = \mathbf{O}(|\frac{\lambda_2}{\lambda_1}|^k). \tag{4.3.3}$$

Note that by deriving of this method we suppose that $\alpha_1 \neq 0$, meaning that method converges if λ_1 is dominant eigenvalue and if initial vector $\vec{v}_0$ has a component with same direction as eigenvector $\vec{x}_1$. On behavior of this method without those assumptions one can find in the monograph of Wilkinson [7, p. 570] and Parlett and Poole [11]. Practically, due to round-off errors in iterative process, the condition $\alpha_1 \neq 0$ will be satisfied after few steps, although starting assumption for vector $\vec{v}_0$ not being fulfilled.

Example 4.3.1.

Let

$$\mathbf{A} = \begin{bmatrix} -261 & 209 & -49 \\ -530 & 422 & -98 \\ -800 & 631 & -144 \end{bmatrix},$$

with eigenvalues $\lambda_1 = 10, \lambda_2 = 4, \lambda_3 = 3$.

By taking for initial vector $\vec{v}_0 = [0 \ \ 0 \ \ -1]^T$, by power method we get the results given in Table 4.3.1.

* If $\vec{x}$ eigenvector, then $c\vec{x}$ ($c \neq 0$) is also eigenvector corresponding to the same eigenvalue.

Table 4.3.1

k	γ_k	$(\vec{v}_k)_1$	$(\vec{v}_k)_2$	$(\vec{v}_k)_3$
1	144.0000	0.340278	0.680556	1.
2	13.2083	0.334911	0.669821	1.
3	10.7287	0.333774	0.667549	1.
4	10.2038	0.333463	0.666926	1.
5	10.0599	0.333372	0.666744	1.
6	10.0179	0.333345	0.666690	1.
7	10.0054	0.333337	0.666674	1.
8	10.0016	0.333334	0.666669	1.
9	10.0005	0.333334	0.666667	1.
10	10.0001	0.333333	0.666667	1.
11	10.0000	0.333333	0.666667	1.

Because of linear convergence of the power method, for convergence acceleration the Aitken δ^2 method can be used. A simple method for convergence acceleration is given in [1], pp. 303-305.

4.4. Methods for subdominant eigenvalues

Suppose that eigenvalues of matrix **A** are ordered in a way

$$|\lambda_1| > |\lambda_2| > \cdots > \lambda_n.$$

In this section the methods for determination of subdominant eigenvalues, i.e. $\lambda_2, \lambda_3, \ldots, \lambda_m$ $(m < n)$ will be considered. The three methods will be explained.

1. Method of orthogonalization. Suppose, at first, that matrix **A** is symmetric, and that eigenvector $\vec{x}_1$ which corresponds to dominant eigenvalue λ_1 ($|\lambda_1| > |\lambda_i|$, $i = 2, \ldots, n$) has been determined by, for example, power method. Starting with arbitrary vector $\vec{z}$, let us form vector $\vec{v}_0$ which is orthogonal to vector $\vec{x}_1$. So we have (see Gram-Schmidt's method of orthogonalization)

$$\vec{v}_0 = \vec{z} - \frac{(\vec{z}, \vec{x}_1)}{(\vec{x}_1, \vec{x}_1)}\vec{x}_1. \tag{4.4.1}$$

Because of $(\vec{v}_0, \vec{x}_1) = 0$, from theoretical point of view, the series $\vec{v}_k = \mathbf{A}\vec{v}_{k-1}$ $(k = 1, 2, \ldots)$ in the power method could be used for determination of λ_2 and corresponding eigenvector $\vec{x}_2$. Nevertheless, regardless of fact that $\vec{v}_0$ does not have the component in direction of eigenvector $\vec{x}_1$, power method would, because of round-off errors, after some number of iterations, converge toward eigenvector $\vec{x}_1$. This fact was mentioned in the previous section.

It is possible to eliminate this influence of round-off errors using so known periodical "purification" of vector $\vec{v}_0$ from component in direction of $\vec{x}_1$. That means that, after, say, r steps, we compute $\vec{v}_0$ using $\vec{v}_r$ in place of $\vec{z}$ in (4.4.1), i.e. by means

$$\vec{v}_0 = \vec{v}_r - \frac{(\vec{v}_r, \vec{x}_1)}{(\vec{x}_1, \vec{x}_1)}\vec{x}_1.$$

In this way, if the period of "purification" is small enough so that it cannot happen significant accumulation of round-off error, by power method can be determined eigenvalue λ_2 and eigenvector $\vec{x}_2$.

By continuation of this procedure we can further determine λ_3 and $\vec{x}_3$.

Generally, if we determine $\lambda_1, \ldots, \lambda_\nu$ and corresponding vectors $\vec{x}_1, \ldots, \vec{x}_\nu$ $(\nu < m)$ it is possible to determine $\lambda_{\nu+1}$ and $\vec{x}_{\nu+1}$ using power method, by forming vector $\vec{v}_0$ orthogonal to $\vec{x}_1, \ldots, \vec{x}_\nu$. So, starting from arbitrary vector $\vec{z}$, we have

$$\vec{v}_0 = \vec{z} - \sum_{i=1}^{\nu} \frac{(\vec{z}, \vec{x}_i)}{(\vec{x}_i, \vec{x}_i)} \vec{x}_i, \tag{4.4.2}$$

meaning that vector $\vec{v}_0$ has components only in direction of residual eigenvectors, i.e.

$$\vec{v}_0 = \alpha_{\nu+1}\vec{x}_{\nu+1} + \ldots + \alpha_n \vec{x}_n.$$

Power method applied to $\vec{v}_0$ gives $\vec{x}_{\nu+1}$ and $\lambda_{\nu+1}$ in absence of round-off errors. Being not the case, it is necessary frequent "purification" of vector $\vec{v}_k$ from components in direction $\vec{x}_1, \ldots, \vec{x}_n$. In other words, after r steps, one should determine again $\vec{v}_0$ using (4.4.2), by using $\vec{v}_r$ in place of $\vec{z}$.

Also, in the case when matrix $\mathbf{A}$ is not symmetric, but has complete system of eigenvectors, the given orthogonalizing procedure can be applied.

2. Inverse iteration method. This method is applied to general matrix $\mathbf{A}$ and is based on solution of system of equations

$$(\mathbf{A} - p\mathbf{I})\vec{v}_k = \vec{v}_{k-1}, \tag{4.4.3}$$

where p is constant, and $\vec{v}_0$ arbitrary vector. System (4.4.3) is usually to be solved by Gauss method of elimination or Cholesky method by factorization of matrix $\mathbf{B} = \mathbf{A} - p\mathbf{I}$. Note that the method of inverse iteration is equivalent to the power method applied to $\mathbf{B}$. Therefore, by applying method of inverse iteration the dominant eigenvalue of matrix $\mathbf{B}$ is obtained, i.e. $\mu_\nu = 1/(\lambda_\nu - p)$ for which it holds

$$\min_j |\lambda_j - p| = |\lambda_\nu - p|.$$

Eigenvalue λ_ν is closest eigenvalue of matrix $\mathbf{A}$ to the number p. Eigenvector obtained thereby is the same one for matrices $\mathbf{B}$ and $\mathbf{A}$.

By convenient choice of parameter p all eigenvalues of matrix $\mathbf{A}$ can be, in principle, obtained.

Similar to power method, here is also suitable to perform norming of vector $\vec{v}_k$ so that we have

$$\mathbf{B}\vec{z}_k = \vec{v}_{k-1}, \qquad \vec{v}_k = \vec{z}_k/\gamma_k, \tag{4.4.4}$$

where γ_k is coordinate of vector $\vec{z}_k$ with greatest module.

Example 4.4.1.

Using method of inverse iteration for matrix

$$\mathbf{A} = \begin{bmatrix} 4 & 1 & 4 \\ 1 & 10 & 1 \\ 4 & 1 & 10 \end{bmatrix},$$

we will determine eigenvalue closest to number $p = 9$, as well as corresponding eigenvector.

Using factorization by Gauss method with pivoting for matrix $\mathbf{B} = \mathbf{A} - 9\mathbf{I}$, we get

$$\mathbf{LR} = \mathbf{PB},$$

where

$$\mathbf{L} = \begin{bmatrix} 1 & 0 & 0 \\ -4/5 & 1 & 0 \\ -1/5 & 2/3 & 1 \end{bmatrix}, \quad \mathbf{R} = \begin{bmatrix} -5 & 1 & 4 \\ 0 & 9/5 & 21/5 \\ 0 & 0 & -1 \end{bmatrix}$$

and permutation matrix $\mathbf{P}$ defined by index series $I = (1, 3)$.

The method of inverse iteration (4.4.4) can now be expressed in the form

$$\mathbf{L}\vec{y}_k = p\vec{v}_{k-1}, \quad \mathbf{R}\vec{z}_k = \vec{y}_k, \quad \vec{v}_k = \vec{z}_k/\gamma_k,$$

by which application the results given in Table 4.4.1 are obtained.

Table 4.4.1

k	$(\vec{v}_k)_1$	$(\vec{v}_k)_2$	$(\vec{v}_k)_3$	β_k
1	0.	1.	−1.	6.
2	−0.2	1.	−0.5	9.3
3	−0.17241	1.	−0.48276	9.34483
4	−0.17200	1.	−0.48000	9.34800
5	−0.17185	1.	−0.47980	9.34835
6	−0.17184	1.	−0.47977	9.34838

For initial vector we took $\vec{v}_0 = [1 \;\; 0 \;\; 0]^T$. In the last column of table is given the quantity $\beta_k = p + 1/\gamma_k$, which gives approximation for corresponding eigenvalue λ. One can see that this eigenvalue has approximate value 9.34838.

3. Deflation methods. The methods of this kind are composed from construction of sequence of matrices $\mathbf{A}_n$ $(= \mathbf{A})$, $\mathbf{A}_{n-1}, \ldots, \mathbf{A}_1$, which order is equal to index and thereby

$$\mathrm{Sp}\,(\mathbf{A}_n) \supset \mathrm{Sp}\,(\mathbf{A}_{n-1}) \supset \cdots \supset \mathrm{Sp}\,(\mathbf{A}_1),$$

where $\mathrm{Sp}\,(\mathbf{A}_k)$ denotes spectrum of matrix $\mathbf{A}_k$.

We will describe now a special and important case of deflation method, when matrix $\mathbf{A}$ is Hermitian.

Let $\vec{x} = [x_1 \;\; x_2 \ldots x_n]^T$ be eigenvector of matrix $\mathbf{A}$ corresponding to eigenvalue λ and normed

$$(\vec{x}, \vec{x}) = \vec{x} * \vec{x} = \|\vec{x}\|_E^2 = 1$$

with first coordinate x_1 being nonnegative.

Have a look over matrix

$$\mathbf{P} = \mathbf{I} - 2\vec{w}\vec{w}^*, \tag{4.4.5}$$

where the vector $\vec{w} = [w_1 \;\; w_2 \;\ldots w_n]^T$ is defined by first vector $\vec{e}_1 = [1 \; 0 \;\ldots\; 0]^T$ from natural basis of space $\mathcal{R}^n$ in the following way:

$$\vec{w} * \vec{w} = \|\vec{x}\|_E^2 = 1, \quad w_1 \geq 0, \tag{4.4.6}$$

$$\mathbf{P}\vec{e}_1 = \vec{x}. \tag{4.4.7}$$

The matrix $\mathbf{P}$ is of form

$$\mathbf{P} = \begin{bmatrix} 1 - 2w_1\bar{w}_1 & -2w_1\bar{w}_2 & \ldots & -2w_1\bar{w}_n \\ -2w_2\bar{w}_1 & 1 - 2w_2\bar{w}_2 & & -2w_2\bar{w}_n \\ \vdots & & & \\ -2w_n\bar{w}_1 & -2w_n\bar{w}_2 & & 1 - 2w_n\bar{w}_n \end{bmatrix}.$$

Note that $\mathbf{P}^* = \mathbf{P}$, what means that matrix $\mathbf{P}$ is Hermitian, too. Moreover, regarding to (4.4.6), by direct multiplication we see that

$$\mathbf{P}^*\mathbf{P} = \mathbf{P}^2 = \mathbf{I},$$

and conclude that matrix $\mathbf{P}$ is unitary.

Based on (4.4.7) we find coordinates of vector w. So, from $1 - 2w_1\bar{w}_1 = x_1$ and $-2w_k\bar{w}_1 = x_k$ $(k = 2, \ldots, n)$ it follows

$$w_1 = \sqrt{\frac{1 - x_1}{2}} \quad \text{and} \quad w_k = -\frac{x_k}{2w_1} \qquad (k = 2, \ldots, n).$$

Note that $\bar{w}_1 = w_1 > 0$.

Now, based on (4.4.7) and $\mathbf{A}\vec{x} = \lambda\vec{x}$ we find that $\mathbf{AP}\vec{e}_1 = \mathbf{P}\vec{e}_1$, wherefrom we conclude that $\mathbf{P}^*\mathbf{AP}\vec{e}_1 = \lambda\vec{e}_1$; i.e. $\vec{e}_1$ is eigenvector of matrix $\mathbf{B} = \mathbf{P}^*\mathbf{AP} = \mathbf{PAP}$. Note, also, that first column in matrix $\mathbf{B}$ is just vector $\lambda\vec{e}_1$, i.e.

$$\mathbf{B} = \begin{bmatrix} \lambda & b_{12} & b_{13} & \cdots & b_{1n} \\ 0 & b_{22} & b_{23} & & b_{2n} \\ 0 & b_{32} & b_{33} & & b_{3n} \\ \vdots & & & & \\ 0 & b_{n2} & b_{n3} & & b_{nn} \end{bmatrix} = \begin{bmatrix} \lambda & \vec{b}^T_{n-1} \\ \vec{0}_{n-1} & \mathbf{A}_{n-1} \end{bmatrix},$$

where with $\mathbf{A}_{n-1}$ we denoted matrix of order $n-1$ which matches with enclosed block. $\vec{0}_{n-1}$ is zero-vector of order $n-1$, and, finally, $\vec{b}^T_{n-1} = [b_{12}\ b_{13} \cdots b_{1n}]^T$.

Regarding the fact that matrix $\mathbf{B}$ is similar (we say also unitary similar) to matrix $\mathbf{A}$, we conclude that

$$\mathrm{Sp}(\mathbf{A}_{n-1}) = \mathrm{Sp}(\mathbf{A}_n)\backslash(\lambda) \qquad (\mathbf{A}_n = \mathbf{A}).$$

In order to get matrix $\mathbf{A}_{n-2}$ we are proceeding in a similar way. In place of matrix $\mathbf{P}$ we use matrix

$$\mathbf{P}_1 = \begin{bmatrix} 1 & \vec{0}^T_{n-1} \\ \vec{0}_{n-1} & \mathbf{Q} \end{bmatrix},$$

where matrix $\mathbf{Q}$ is of order $n-1$ and of form (4.4.5), satisfying the conditions (4.4.6) and (4.4.7) regarding eigenvector $\vec{y}$ and eigenvalue ν of matrix $\mathbf{A}_{n-1}$. Because of $\mathbf{P}_1^{-1} = \mathbf{P}_1^* = \mathbf{P}_1$ we conclude that matrix $\mathbf{P}_1$ is unitary, too.

Now matrix $\mathbf{C} = \mathbf{P}_1\mathbf{BP}_1 = \mathbf{P}_1\mathbf{PAPP}_1$ has a form

$$\mathbf{C} = \begin{bmatrix} \lambda & c_{12} & c_{13} & \cdots & c_{1n} \\ 0 & \mu & c_{23} & & c_{2n} \\ 0 & 0 & c_{33} & & c_{3n} \\ \vdots & & & & \\ 0 & 0 & c_{n3} & & c_{nn} \end{bmatrix} = \begin{bmatrix} \lambda & c_{12} & c_{13} & \cdots & c_{1n} \\ 0 & \mu & c_{23} & & c_{2n} \\ 0 & 0 & & & \\ \vdots & & & \mathbf{A}_{n-2} & \\ 0 & 0 & & & \end{bmatrix},$$

where matrix $\mathbf{A}_{n-2}$ is of order $n-2$. By continuing this procedure we get upper triangular matrix which is unitary similar to initial matrix $\mathbf{A}$. Having in mind that matrix $\mathbf{A}$ is Hermitian, we conclude that it is unitary similar to diagonal matrix.

The presented procedure demands, before of every step, determination of one eigenvalue and corresponding eigenvector, what can be done by some of previously presented methods. Thus, before the first step, one has to determine eigenvalue λ and eigenvector

$\vec{x}$ of matrix $\mathbf{A}$, before the second step eigenvalue μ and eigenvector $\vec{y}$ of matrix $\mathbf{A}_{n-1}$, and so on.

It is clear that eigenvalues of matrix $\mathbf{A}$ are diagonal elements of obtained triangular matrix, i.e. $\lambda_1 = \lambda$, $\lambda_2 = \mu$, etc. It remains the question what is with eigenvectors of matrix $\mathbf{A}$? It is clear that $\vec{x}_1 = \vec{x}$. We will show how, based on obtained results, the second eigenvector of matrix $\mathbf{A}$ can be found.

If the coordinates of eigenvector $\vec{y}$ are $y_2, \ldots, y_n$, in order to find, at first, eigenvector $\vec{y}\,'$ of matrix $\mathbf{B}$, put $\vec{y}\,' = [y_1\ y_2\ \ldots y_n]^T$ and try to determine y_1.

Because of

$$\mathbf{B}\vec{y}\,' = \begin{bmatrix} \lambda & \vec{b}_{n-1}^T \\ \vec{0}_{n-1} & \mathbf{A}_{n-1} \end{bmatrix} \cdot \begin{bmatrix} y_1 \\ \vec{y} \end{bmatrix} = \begin{bmatrix} \lambda y_1 + \vec{b}_{n-1}^T \vec{y} \\ \mathbf{A}_{n-1}\vec{y} \end{bmatrix},$$

i.e.

$$\mathbf{B}\vec{y}\,' = \begin{bmatrix} \lambda y_1 + \vec{b}_{n-1}^T \vec{y} \\ \mu \vec{y} \end{bmatrix},$$

it follows

$$y_1 + \vec{b}_{n-1}^T \vec{y} = y_1.$$

If $\lambda \neq \mu$, by virtue of previous equality we get

$$y_1 = \frac{1}{\lambda - \mu} \vec{b}_{n-1}^T \vec{y} = \frac{1}{\lambda - \mu}(b_{12}y_2 + \cdots + b_{1n}y_n).$$

Now simply find the eigenvector $\vec{x}_2$ of matrix $\mathbf{A}$, corresponding to eigenvalue $\lambda_2 \neq \mu$. Indeed, because of $\mathbf{PAP}\vec{y}\,' = \mu\vec{y}\,'$, i.e. $\mathbf{A}(\mathbf{P}\vec{y}\,') = \mu(\mathbf{P}\vec{y}\,')$ we conclude that $\vec{x}_2 = \mathbf{P}\vec{y}\,'$.

In a similar way the other eigenvectors can be determined.

4.5. Eigenvalue problem for symmetric tridiagonal matrices

Let $\mathbf{A}$ be real symmetric tridiagonal matrix of order n which non-zero elements will be denoted as

$$a_{ii} = b_i \quad (i = 1, \ldots, n),$$
$$a_{i,i-1} = a_{i-1,i} = c_i \quad (i = 2, \ldots, n).$$

With $p_k(\lambda)$ denote main minor of order k of matrix $\mathbf{A} - \lambda\mathbf{I}$, i.e.

$$p_k(\lambda) = \begin{vmatrix} b_1 - \lambda & c_2 & & & & \\ c_2 & b_2 - \lambda & c_3 & & \mathbf{0} & \\ & \ddots & \ddots & \ddots & & \\ & \mathbf{0} & & c_{k-1} & b_{k-1} - \lambda & c_k \\ & & & & c_k & b_k - \lambda \end{vmatrix}$$

and define $p_0(\lambda) = 1$. Note that $p_1(\lambda) = b_1 - \lambda$.

By developing of determinant $p_k(\lambda)$ up to elements of last row we get

$$p_k(\lambda) = (b_k - \lambda)p_{k-1}(\lambda) - c_k^2 p_{k-2}(\lambda).$$

The value of characteristic polynomial of matrix $\mathbf{A}$ can be simple evaluated, based on previous, using three-term recurrence relation

$$\begin{aligned} &p_k(\lambda) = (b_k - \lambda)p_{k-1}(\lambda) - c_k^2 p_{k-2}(\lambda) \quad (k = 2, \ldots, n), \\ &p_0(\lambda) = 1, \quad p_1(\lambda) = b_1 - \lambda. \end{aligned} \tag{4.5.1}$$

A simple method for determination of eigenvalues of symmetric tridiagonal matrices is based on usage of recurrent relation (4.5.1), method of interval bisection, and statement of the following theorem, which is simple to prove:

Theorem 4.5.1. *(Givens) Let all elements $c_k \neq 0$ of symmetric tridiagonal matrix* **A** *of order n. Then it holds:*

(1) Zeros of every polynomial p_k $(k = 2, \ldots, n)$ are real, different, and separated by zeros of polynomial p_{k-1};
(2) If $p_n(\lambda) \neq 0$, number of eigenvalues of matrix **A** less than λ is equal to number of sign change $s(\lambda)$ in the series

$$p_0(\lambda),\ p_1(\lambda),\ \ldots, p_n(\lambda). \tag{4.5.2}$$

If some $p_k(\lambda) = 0$, then on this place in series (4.5.2) can be taken arbitrary sign, regarding to $p_{k-1}(\lambda)p_{k+1}(\lambda) < 0$.

Remark that in theorem there exists condition $c_k \neq 0$ for every $k = 2, \ldots, n$. If, for example, for some $k = m$, $c_m = 0$, then problem simplifies, because it splits in two problems of lower order (m and $n - m$). Namely, matrix **A** becomes

$$\mathbf{A} == \begin{bmatrix} \mathbf{A}' & \mathbf{0} \\ \mathbf{0} & \mathbf{A}'' \end{bmatrix},$$

where $\mathbf{A}'$ and $\mathbf{A}''$ are tridiagonal symmetric matrices of order m and $n-m$, respectively, and in this case is

$$\det(\mathbf{A} - \lambda\mathbf{I}) = \det(\mathbf{A}' - \lambda\mathbf{I}) \det(\mathbf{A}'' - \lambda\mathbf{I}).$$

Using multiple values for λ it is possible by systematic application of Theorem 4.5.1 to determine disjunct intervals in which lie eigenvalues of matrix **A**. Thus, if we find that

$$s(\lambda_1) = s_1 \quad \text{and} \quad s(\lambda_2) = s_2 = s_1 + 1 \quad (\lambda_1 < \lambda_2),$$

based on Theorem 4.5.1 we have that in interval (λ_1, λ_2) lies one only eigenvalue of matrix **A**. Then for its determination the simple method of halving of interval (bisection method) can be used, by contraction of this starting interval up to desired exactness.

For determination of intervals in which lie eigenvalues it can be used also theorem of Gershgorin, so that those intervals are

$$\begin{aligned} &[b_1 - |c_2|,\ b_1 + |c_2|], \\ &[b_i - |c_i| - |c_{i+1}|,\ b_i + |c_i| + |c_{i+1}|], \quad (i = 2, \ldots, n-1), \\ &[b_n - |c_n|,\ b_n + |c_n|]. \end{aligned}$$

Unfortunately, these intervals are not disjunct, and in general case contain not only one eigenvalue of matrix **A**.

Example 4.5.1.

For given matrix

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & & \\ 1 & 3 & 2 & \\ & 2 & 5 & 3 \\ & & 3 & 7 \end{bmatrix}.$$

we have

$$p_0(\lambda) = 1,\ p_1(\lambda) = 1 - \lambda,\ p_2(\lambda) = (3 - \lambda)p_1(\lambda) - p_0(\lambda),$$
$$p_3(\lambda) = (5 - \lambda)p_2(\lambda) - 4p_1(\lambda),\ p_4(\lambda) = (7 - \lambda)p_3(\lambda) - 9p_2(\lambda).$$

Let $\lambda = 0$. Then we have $p_0(0) = 1$, $p_1(0) = 1$, $p_2(0) = 2$, $p_3(0) = 6$, $p_4(0) = 24$. Thus, in the series (4.5.2) are $+++++$, what means that there is no sign change, i.e.

$s(0) = 0$. According to Theorem 4.5.1, matrix **A** does not have negative eigenvalues, i.e. it is positive-definite.

Taking in sequence for λ values $1, 2, 4, 5, 7, 9, 10$ we get the results given in Table 4.5.1.

Table 4.5.1

λ	$p_0(\lambda)$	$p_1(\lambda)$	$p_2(\lambda)$	$p_3(\lambda)$	$p_4(\lambda)$	$s(\lambda)$
1	1	0	−1	−4	−15	1
2	1	−1	−2	−2	8	2
4	1	−3	2	14	24	2
5	1	−4	7	16	−31	3
7	1	−6	23	−22	−207	3
9	1	−8	47	−156	−111	3
10	1	−9	62	−274	264	4

Based on values of $s(\lambda)$ we conclude that in each interval $(0,1), (1,2), (4,5), (9,10)$ is located one eigenvalue of matrix **A**. These eigenvalues with six figures are

$$\lambda_1 \cong 0.322548, \ \lambda_1 \cong 1.745761, \ \lambda_1 \cong 4.536620, \ \lambda_1 \cong 9.395071.$$

Note that these are zeroes of Laguerre polynomial L_4.

4.6. LR and QR algorithms

This section is devoted to so known factorization methods. First such method for solution of problem of eigenvalues for arbitrary matrix **A** was described by H. Rutishauser ([14]) in the year 1958, which called it **LR** transformation. Method consists in construction of series of matrices $\{\mathbf{A}_k\}_{k\in N}$, starting from $\mathbf{A}_1 = \mathbf{A}$, in the following way: Matrix **A** factorizes to lower triangular matrix $\mathbf{L}_k$ with unit diagonal and upper triangular matrix $\mathbf{R}_k$, i.e.

(4.6.1)
$$\mathbf{A}_k = \mathbf{L}_k\mathbf{R}_k,$$

and then the following member is determined by multiplication of obtained factors in opposite sequence, i.e.

$$\mathbf{A}_{k+1} = \mathbf{R}_k\mathbf{L}_k.$$

Note that matrices $\mathbf{A}_{k+1}$ and $\mathbf{A}_k$ are similar, because they are connected with transformation of similarity

(4.6.2)
$$\mathbf{A}_{k+1} = \mathbf{L}_k^{-1}\mathbf{A}_k\mathbf{L}_k.$$

Factorization of (4.6.1) can be performed by Gauss method of elimination.

If we put

$$\mathbf{L}^{(k)} = \mathbf{L}_1 \ldots \mathbf{L}_k \quad \text{and} \quad \mathbf{R}^{(k)} = \mathbf{R}_k \ldots \mathbf{R}_1,$$

based on (4.6.2) we have

$$\mathbf{L}^{(k)}\mathbf{A}_{k+1} = \mathbf{A}\mathbf{L}^{(k)},$$

wherefrom it follows

$$\mathbf{L}^{(\mathbf{k})}\mathbf{R}^{(\mathbf{k})} = \mathbf{L}^{(k-1)}\mathbf{A}_k\mathbf{R}^{(k-1)} = \mathbf{A}\mathbf{L}^{(k-1)}\mathbf{R}^{(k-1)}.$$

By iterating the last equality, we get

$$\mathbf{L}^{(k)}\mathbf{R}^{(k)} = \mathbf{A}^2\mathbf{L}^{(k-2)}\mathbf{R}^{(k-2)} = \ldots = \mathbf{A}^k,$$

what means that $\mathbf{L^{(k)}R^{(k)}}$ is factorization of matrix $\mathbf{A}^k$. Using this facts, Rutishauser (see, also [7]) showed that under certain conditions series of matrices $\{\mathbf{A}_k\}$ converges towards some upper triangular matrix, which elements on the main diagonal give eigenvalues of matrix **A**. Usually, **LR** method is applied to matrices previously reduced to upper Hessenberg form ($a_{ij} = 0$ for $i \geq j+2$). If, by means of some method, general matrix reduced to lower Hessenberg form we apply **LR** method to transposed matrix, which has the same eigenvalues. All matrices in series $\{\mathbf{A}_k\}$ have Hessenberg form. Acceleration of convergence of series $\{\mathbf{A}_k\}$ can be done by convenient shifting p_k, so that, in place of $\mathbf{A}_k$ we factorize $\mathbf{B}_k = \mathbf{A}_k - p_k\mathbf{I} = \mathbf{L}_k\mathbf{R}_k$, whereby $\mathbf{A}_{k+1} = p_k\mathbf{I} + \mathbf{R}_k\mathbf{L}_k$.

Unfortunately, **LR** algorithm has several disadvantages (see monograph of Wilkinson [7]). For example, factorization does not exist for every matrix. One better factorization method was developed by J.G.F. Francis ([15]) and V.N. Kublanovskaya ([16]), where matrix **L** is replaced with unitary matrix **Q**. So one gets **QR** algorithm defined by

$$\mathbf{A}_k = \mathbf{Q}_k\mathbf{R}_k, \quad \mathbf{A}_{k+1} = \mathbf{R}_k\mathbf{Q}_k \quad (k = 1, 2, \ldots), \tag{4.6.3}$$

starting from $\mathbf{A}_1 = \mathbf{A}$. Note that $\mathbf{A}_{k+1} = \mathbf{Q}_k^*\mathbf{A}_k\mathbf{Q}_k$.

If we put

$$\mathbf{Q}^{(k)} = \mathbf{Q}_1 \ldots \mathbf{Q}_k \quad \text{and} \quad \mathbf{R}^{(k)} = \mathbf{R}_k \ldots \mathbf{R}_1, \tag{4.6.4}$$

similar as **LR** method, we find

$$\mathbf{Q}^{(k)}\mathbf{A}_{k+1} = \mathbf{A}\mathbf{Q}^{(k)} \quad \text{and} \quad \mathbf{Q}^{(k)}\mathbf{R}^{(k)} = \mathbf{A}^k. \tag{4.6.5}$$

Theorem 4.6.1. *If matrix* **A** *regular, then exists decomposition* $\mathbf{A} = \mathbf{QR}$, *where* **Q** *is unitary, and* **R** *upper triangular matrix. Moreover, if diagonal elements of matrix* **R** *are positive, decomposition is unique.*

QR factorization (4.6.3) can be performed by using unitary matrices of form $\mathbf{I} - 2\vec{w}\vec{w}^*$. So, in order to transform $\mathbf{A}_k$ to $\mathbf{R}_k$, i.e. reduction of columns to $\mathbf{A}_k$, we have

$$(\mathbf{I} - 2\vec{w}_{n-1}\vec{w}_{n-1}^*) \ldots (\mathbf{I} - 2\vec{w}_1\vec{w}_1^*)\mathbf{A}_k = \mathbf{R}_k. \tag{4.6.6}$$

The matrix $\mathbf{Q}_k$ is then

$$\mathbf{Q}_k = (\mathbf{I} - 2\vec{w}_1\vec{w}_1^*) \ldots (\mathbf{I} - 2\vec{w}_{n-1}\vec{w}_{n-1}^*). \tag{4.6.7}$$

QR algorithm is efficient if initial matrix has (upper) Hessenberg form. Then, previously mentioned unitary matrices reduce to two-dimensional rotations. All matrices $\mathbf{A}_k$ are of Hessenberg form. Thus, eigenvalue problem for general matrix is most convenient to be solved in two steps. At first, reduce matrix to Hessenberg form, and then apply the **QR** algorithm.

In special case, when initial matrix is tridiagonal, matrices $\mathbf{A}_k$ in **QR** algorithm are also tridiagonal. In that case, using conveniently chosen shift p_k, **QR** algorithm becomes very efficient for solving eigenvalue problem of tridiagonal matrices.

Similar to **QR** algorithm, it is developed **QL** algorithm ([18]), where **L** is lower triangular matrix, and **Q** unitary matrix. Also, it has been developed so known implicit **QL** algorithm ([19]).

4.7. Software eigenpackages

Some general guidelines for solving eigenproblems are summarized below [22].

- When only the largest and (or) the smallest eigenvalue of a matrix is required, the power method can be employed.
- Although it is rather inefficient, the power method can be used to solve for intermediate eigenvalues.

- The direct method is not a good method for solving linear eigenproblems. However, it can be used for solving nonlinear eigenproblems.
- For serious eigenproblems, the QR method is recommended.
- Eigenvectors corresponding to a known eigenvalue can be determined by one application of the shifted inverse power method.

Almost all software routines in use nowadays trace their ancestry back to routines published in Wilkinson and Reinsch's boock *Handbook for Automatic Computation, Vol. II, Linear Algebra* [13]. A public-domain implementation of the *Handbook* routines in `FORTRAN` is the `EISPACK` set of programs [3]. The routines presented in majority of most frequently used software packages are translations of either the *Handbook* or `EISPACK` routines, so understanding these will take a lot of the way towards understanding those canonical packages.

`IMSL` [4] and `NAG` [5] each provide proprietary implementations in `FORTRAN` of what are essentially the *Handbook* routines.

Many commercial software packages contain eigenproblem solvers. Some of the more prominent packages are `Matlab` and `Mathcad`. More sophisticated packages, such as `Mathematica`, `Macsyma`, and `Maple` also contain eigenproblem solvers. The book *Numerical Recepies* [2] contains subroutines and advice for solving eigenproblems.

A good "eigenpackage" will provide separate routines, or separate paths through sequences of routines, for the following desired calculations

- all eigenvalues and no eigenvectors
- all eigenvalues and some corresponding eigenvectors
- all eigenvalues and all corresponding eigenvectors.

The purpose of these distinctions is to save compute time and storage; it is wasteful to calculate eigenvectors that you don't need. Often one is interested only in the eigenvectors corresponding to the largest few eigenvalues, or largest few in the magnitude, or few that are negative. The method usually used to calculate "some" eigenvectors is typically more efficient than calculating all eigenvectors if you desire fewer than about a quarter of the eigenvectors.

A good eigenpackage also provides separate paths for each of the above calculations for each of the following special forms of the matrix:

- real, symmetric, tridiagonal
- real, symmetric, banded (only a small number of sub- and super-diagonals are nonzero)
- real, symmetric
- real, nonsymmetric
- complex, Hermitian
- complex, non-Hermitian.

Again, the purpose of these distinctions is to save time and storage by using the least general routine that will serve in any particular application.

Good routines for the following paths are available:

- all eigenvalues and eigenvectors of a real, symmetric, tridiagonal matrix
- all eigenvalues and eigenvectors of a real, symmetric, matrix
- all eigenvalues and eigenvectors of a complex, Hermitian matrix
- all eigenvalues and no eigenvectors of a real, nonsymmetric matrix.

4.8. Generalized and nonlinear eigenvalue problems

Many eigenpackages also deal with the so-called *generalized eigenproblem* [6],

$$\mathbf{A} \cdot \vec{\mathbf{x}} = \lambda \mathbf{B} \cdot \vec{\mathbf{x}} \tag{4.8.1}$$

where $\mathbf{A}$ and $\mathbf{B}$ are both matrices. Most such problems, where $\mathbf{B}$ is nonsingular, can be handled by the equivalent

$$(\mathbf{B}^{-1} \cdot \mathbf{A}) \cdot \vec{x} = \lambda \vec{x} \tag{4.8.2}$$

Often $\mathbf{A}$ and $\mathbf{B}$ are symmetric and $\mathbf{B}$ is positive definite. The matrix $\mathbf{B}^{-1} \cdot \mathbf{A}$ in (4.8.2) is not symmetric, but we can recover a symmetric eigenvalue problem by using the Cholesky decomposition $\mathbf{B} = \mathbf{L} \cdot \mathbf{L}^T$. Multiplying equation (4.8.1) by $\mathbf{L}^{-1}$ we get

$$\mathbf{C} \cdot (\mathbf{L}^T \cdot \vec{x}) = \lambda(\mathbf{L}^T \cdot \vec{x}) \tag{4.8.3}$$

where

$$\mathbf{C} = \mathbf{L}^{-1} \cdot \mathbf{A} \cdot (\mathbf{L}^{-1})^T \tag{4.8.4}$$

The matrix $\mathbf{C}$ is symmetric and its eigenvalues are the same as those of the original problem (4.8.1); its eigenfunctions are $\mathbf{L}^T \cdot \vec{x}$. The efficient way to form $\mathbf{C}$ is first to solve the equation

$$\mathbf{Y} \cdot \mathbf{L}^T = \mathbf{A} \tag{4.8.5}$$

for the lower triangle of the matrix $\mathbf{Y}$. Then solve

$$\mathbf{L} \cdot \mathbf{C} = \mathbf{Y} \tag{4.8.6}$$

for the lower triangle of the symmetric matrix $\mathbf{C}$.

Another generalization of the standard eigenvalue problem is to problems nonlinear in the eigenvalue λ, for example,

$$(\mathbf{A}\lambda^2 + \mathbf{B}\lambda + \mathbf{C}) \cdot \vec{x} = 0 \tag{4.8.7}$$

This can be turned into a linear problem by introducing an additional unknown eigenvector $\vec{y}$ and solving the $2n \times 2n$ eigensystem,

$$\begin{bmatrix} 0 & \mathbf{I} \\ -\mathbf{A}^{-1} \cdot \mathbf{C} & -\mathbf{A}^{-1} \cdot \mathbf{B} \end{bmatrix} \begin{bmatrix} \vec{x} \\ \vec{y} \end{bmatrix} = \lambda \begin{bmatrix} \vec{x} \\ \vec{y} \end{bmatrix}.$$

This technique generalizes to higher-order polynomials in λ. A polynomial of degree m produces a linear $mn \times mn$ eigensystem, as given in [7].

Bibliography (Cited references and further reading)

[1] Milovanović , G.V., *Numerical Analysis I*, Naučna knjiga, Beograd, 1988 (Serbian).

[2] Press, W.H., Flannery, B.P., Teukolsky, S.A., and Vetterling, W.T., *Numerical Recepies - The Art of Scientific Computing*. Cambridge University Press, 1989.

[3] Smith, B.T., et al., *Matrix Eigensystem Routines - EISPACK Guide, 2nd ed., vol 6 of Lecture Notes in Computer Science*. Springer, New York, 1976.

[4] *IMSL Math/Library Users Manual* , IMSL Inc., 2500 City West Boulevard, Houston TX 77042

[5] *NAG Fortran Library*, Numerical Algorithms Group, 256 Banbury Road, Oxford OX27DE, U.K., Chapter F02.

[6] Golub, G.H., and Van Loan, C.F., *Matrix Computation*, Johns Hopkins University Press, Baltimore 1989.

[7] Wilkinson, *The Algebraic Eigenvalue Problem*, Oxford University Press, New York, 1965.

[8] Acton, F.S., *Numerical Methods that Work*, corrected edition, Mathematical Association of America, Chapter 13, Washington, 1970.

[9] Horn, R.A., and Johnson, C.R., *Matrix Analysis*, Cambridge University Press, Cambridge, 1985.

[10] Milovanović, G.V. and Djordjević, Dj.R., *Programiranje numeričkih metoda na FORTRAN jeziku.* Institut za dokumentaciju zaštite na radu "Edvard Kardelj", Niš, 1981 (Serbian).

[11] Parlett, B.N. and Poole, W.G., *A geometric theory for the QR, LU, and power iterations.* SIAM J. Numer. Anal. 10(1973), 389-412.

[12] Bart, W., Martin, R.S., Wilkonson, J.H. *Calculation of the eigenvalues of a symmetric tridiagonal matrix by the bisection method.* Numer Math. 9(1967), 386-393.

[13] Wilkinson, J.H. & Reisch, C., *Handbook for Automatic Computation. Vol. II Linear Algebra.* Springer Verlag, Berlin-Heidelberg-New York, 1971.

[14] Rutishauser, H., *Solution of eigenvalue problem with the LR-transformation.* Appl. Math. Ser. Nat. Bur. Stand. 49(1958), 47-81.

[15] Francis, J.G.F., *The QR transformation - a unitary analogue to the LR transformation.* Comput. J. 4(1961/62), 265-271, 332-345.

[16] Kublanovskaya, V.N., *O nekotoryh algorifmah dlja rešenija polnoĭ problemy sobstvennyh značeniĭ.* Ž. Vyčisl. Mat. i Mat. Fiz. 1(1961), 555-570.

[17] Golub, G.H. & Welsch, J.H., *Calculation of Gauss quadrature rules.* Math. Comp. 23(1969), 221-230.

[18] Bowdler, H., Martin, R.S., Reinsch, C. Wilkinson, J.H., *The QR and QL algorithms for symmetric matrices.* Numer. Math. 11(1968), 293-306.

[19] Dubrulle, A., Martin, R.S., Wilkinson, J.H., *The implicit QR algorithm.* Numer. Math. 12(1968), 377-383.

[20] Stoer, J., and Bulirsch, R., *Introduction to Numerical Analysis*, Springer, New York, 1980.

[21] Mitrinović, D.S. and Djoković, D.Ž., *Polinomi i matrice.* ICS, Beograd, 1975 (Serbian).

[22] Hoffman, J.D., *Numerical Methods for Engineers and Scientists.* Taylor & Francis, Boca Raton-London-New York-Singapore, 2001.

Faculty of Civil Engineering
Belgrade
Master Study
COMPUTATIONAL ENGINEERING

Faculty of Civil Engineering and Architecture
Niš
Doctoral Study

LECTURES

LESSON V

5. Nonlinear Equations and Systems

5.1. Nonlinear Equations

5.1.0. Introduction

We consider that most basic of tasks, solving equations numerically. While most equations are born with both a right-hand side and a left-hand side, one traditionally moves all terms to the left, leaving

(5.1.0.1) $$f(x) = 0$$

whose solution or solutions are desired. When there is only one independent variable, the problem is one-dimensional, namely to find the root or roots of a function. Figure 5.1.0.1 illustrates the problem graphically.

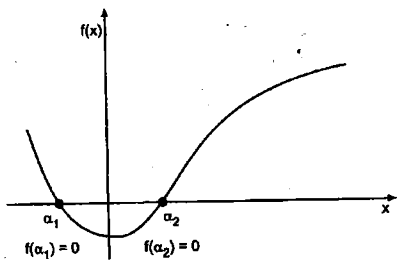


Figure 5.1.0.1

With more than one independent variable, more than one equation can be satisfied simultaneously. You likely once learned the implicit function theorem which (in this context) gives us the hope of satisfying n equations in n unknowns simultaneously. Note that we have only hope, not certainty. A nonlinear set of equations may have no (real) solutions at all. Contrariwise, it may have more than one solution. The implicit function theorem tells us that generically the solutions will be distinct, pointlike, and separated from each other. But, because of nongeneric, i.e., degenerate, case, one can get a continuous family of solutions. In vector notation, we want to find one or more n-dimensional solution vectors $\vec{x}$ such that

(5.1.0.2) $$\vec{f}(\vec{x}) = \vec{0}$$

where $\vec{f}$ is the n-dimensional vector-valued function whose components are the individual equations to be satisfied simultaneously. Simultaneous solution of equations in n dimensions is much more difficult than finding roots in the one-dimensional case. The principal difference between one and many dimensions is that, in one dimension, it is possible to bracket or "trap" a root between bracketing values, and then find it out directly. In multidimensions, you can never be sure that the root is there at all until you have found it. Except in linear problems, root finding invariably proceeds by iteration, and this is equally true in one or in many dimensions. Starting from some approximate trial solution, a useful algorithm will improve the solution until some predetermined convergence criterion is satisfied. For smoothly varying functions, good algorithms will always converge, provided that the initial guess is good enough. Indeed one can even determine in advance the rate of convergence of most algorithms. It cannot be overemphasized, however, how crucially success depends on having a good first guess for the solution, especially for multidimensional problems. This crucial beginning usually depends on analysis rather than numerics. Carefully crafted initial estimates reward you not only with reduced computational effort, but also with understanding and increased self-esteem. Hammings motto, "the purpose of computing is insight, not numbers," is particularly apt in the area of finding roots. One should repeat this motto aloud whenever program converges, with ten-digit accuracy, to the wrong root of a problem, or whenever it fails to converge because there is actually no root, or because there is a root but initial estimate was not sufficiently close to it. For one-dimensional root finding, it is possible to give some straightforward answers: You should try to get some idea of what your function looks like before trying to find its roots. If you need to mass-produce roots for many different functions, then you should at least know what some typical members of the ensemble look like. Next, you should always bracket a root, that is, know that the function changes sign in an identified interval, before trying to converge to the roots value. Finally, one should never let iteration method get outside of the best bracketing bounds obtained at any stage. We can see that some pedagogically important algorithms, such as secant method or Newton-Raphson, can violate this last constraint, and are thus not recommended unless certain fixups are implemented. Multiple roots, or very close roots, are a real problem, especially if the multiplicity is an even number. In that case, there may be no readily apparent sign change in the function, so the notion of bracketing a root and maintaining the bracket becomes difficult. We nevertheless insist on bracketing a root, even if it takes the minimum-searching techniques to determine whether a tantalizing dip in the function really does cross zero or not. As usual, we want to discourage the reader from using routines as black boxes without understanding them.

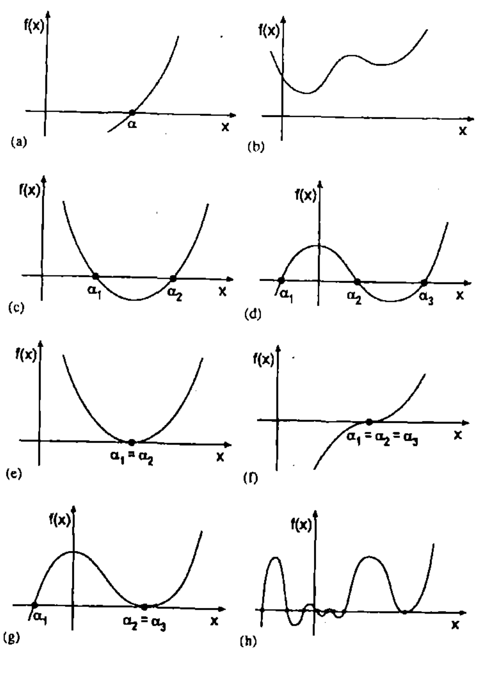


Figure 5.1.0.2

Nonlinear equations can behave in various ways in the vicinity of a root. Algebraic and transcendental equations may have distinct (i.e. simple) real roots, repeated (i.e.

multiple) real roots, or complex roots. Polynomials may have real or complex roots. If the polynomial coefficients are all real, complex roots occur in conjugate pairs. If the polynomial coefficients are complex, single complex roots can occur. Figure 5.1.0.2 illustrates several distinct types of behavior of nonlinear equations in the vicinity of a root. (a) illustrates the case of a single real root, called simple root. (b) illustrates a case where no real roots exist. Complex roots may exist in such a case. Two and three simple roots are showed on (c) and (d), respectively. Two and three multiple roots are illustrated on (e) and (f), respectively. A case with one simple root and two multiple roots is given in (g), and in (h) is illustrated the general case with any number of simple and multiple roots.

There are two distinct phases in finding the roots of nonlinear equation (see [2], pp. 130-135):
(1) Bounding the solution, and
(2) Refining the solution.

In general, nonlinear equations can behave in many different ways in the vicinity of a root.

(1) Bounding the solution

Bounding the solution involves finding a rough estimate of the solution that can be used as the initial approximation, or the starting point, in a systematic procedure that refines the solution to a specified tolerance in an efficient manner. If possible, it is desirable to bracket the root between two points at which the value of the nonlinear function has opposite signs. The bounding procedures can be:

1. Drafting the function,
2. Incremental search,
3. Previous experience or similar problem,
4. Solution of a simplified approximate model.

Drafting the function involves plotting the nonlinear function over the range of interest. Spreadsheets generally have graphing capabilities, as does `Mathematica`, `Matlab` and `Mathcad`. The resolution of the plots is generally not precise enough for accurate result. However, they are accurate enough to bound the solution. The plot of the nonlinear function displays the behavior of nonlinear equation and gives view of scope of problem.

An incremental search is conducted by starting at one end of the region of interest and evaluating the nonlinear function with small increments across the region. When the value of the function changes the sign, it is assumed that a root lies in that interval. Two end points of the interval containing the root can be used as initial guesses for a refining method (second phase of solution). If multiple roots are suspected, one has to check for sigh changes in the derivative of the function between the ends of the interval.

(2) Refining the solution

Refining the solution involves determining the solution to a specified tolerance by an efficient procedure. The basic methods for refining the solution are:

2.1 Trial and error,
2.2 Closed domain methods (bracketing method),
2.3 Open domain methods.

Trial and error methods simply presume (guess) the root, $x = \alpha$, evaluate $f(\alpha)$, and compare to zero. If $f(\alpha)$ is close enough to zero, quit, if not guess another α and continue until $f(\alpha)$ is close enough to zero.

Closed domain (bracketing methods) are methods that start with two values of x which bracket the root, $x = \alpha$, and systematically reduce the interval, keeping root inside of brackets (inside of interval). Two most popular methods of that kind are:

2.2.1 Interval halving (bisection),
2.2.2 False position (Regula Falsi).

Bracketing methods are robust and reliable, since root is always inside of closed interval, but can be slow to convergence.

Open domain methods do not restrict the root to remain trapped in a closed interval. Therefore, there are not as robust as bracketing methods and can diverge. But, they use information about the nonlinear function itself to come closer with estimation of the root. Thus, they are much more efficient than bracketing methods.

Some general hints for root finding

Nonlinear equations can behave in various ways in the vicinity of a root. Algebraic and transcendental equations may have simple real roots, multiple real roots, or complex roots. Polynomials may have real or complex roots. If the polynomial coefficients are all real, complex root occur in conjugate pairs. If the polynomial coefficients are complex, single complex roots can occur.

There are numerous methods for finding the roots of a nonlinear equation. Some general philosophy of root finding is given below.

1. Bounding method should bracket a root, if possible.
2. Good initial approximations are extremely important.
3. Closed domain methods are more robust than open domain methods because they keep the root in a closed interval.
4. Open domain methods, when converge, in the general case converge faster than closed domain methods.
5. For smoothly varying functions, most algorithms will always converge if the initial approximation is close enough. The rate of convergence of most algorithms can be determined in advance.
6. Many problems in engineering and science are well behaved and straightforward. In such cases, a straightforward open domain method, such as Newton's method, or the secant method, can be applied without worrying about special cases and strange behavior. If problems arise during the solution, then the peculiarities of the nonlinear equation and the choice of solution method can be reevaluated.
7. When a problem is to be solved only once or a few times, then the efficiency of method is not of major concern. However, when a problem is to be solved many times, efficiency is of major concern.
8. Polynomials can be solved by any of the methods for solving nonlinear equations. However, the special techniques applicable to polynomials should be considered.
9. If a nonlinear equation has complex roots, that has to be anticipated when choosing a method.
10. Time for problem analysis versus computer time has to be considered during method selection.
11. Generalizations about root-finding methods are generally not possible.

The root-finding algorithms should contain the following features:

1. An upper limit on the number of iterations.
2. If the method uses the derivative $f'(x)$, it should be monitored to ensure that it does not approach zero.
3. A convergence test for the change in the magnitude of the solution, $|x_{i+1} - x_i|$, or the magnitude of the nonlinear function, $|x_{i+1}|$, has to be included.
4. When convergence is indicated, the final root estimate should be inserted into the nonlinear function $f(x)$ to guarantee that $f(x) = 0$ within the desired tolerance.

5.1.1. Newton's method

Newton's or often called Newton-Raphson method is basic method for determination of isolated zeros of nonlinear equations.

Let isolated unique simple root $x = a$ of equation (5.1.0.1) exist on segment $[\alpha, \beta]$ and let $f \in C[\alpha, \beta]$. Then, using Taylor development, we get

$$f(a) = f(x_0) + f'(x_0)(a - x_0) + \mathbf{O}\big((a - x_0)^2\big), \tag{5.1.1.1}$$

where $\xi = x_0 + \theta(a - x_0)$ $(0 < \theta < 1)$. Having in mind that $f(a) = 0$, by neglecting last member on the right-hand side of (5.1.1.1), we get

$$a \cong x_0 - \frac{f(x_0)}{f'(x_0)}.$$

If we denote left-hand side of last approximative equation with x_1, we get

(5.1.1.2) $$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}.$$

Here x_1 represents the abscissa of intersection of tangent on the curve $y = f(x)$ in the point $(x_0, f(x_0))$ with x-axis (see Figure 5.1.1.1).

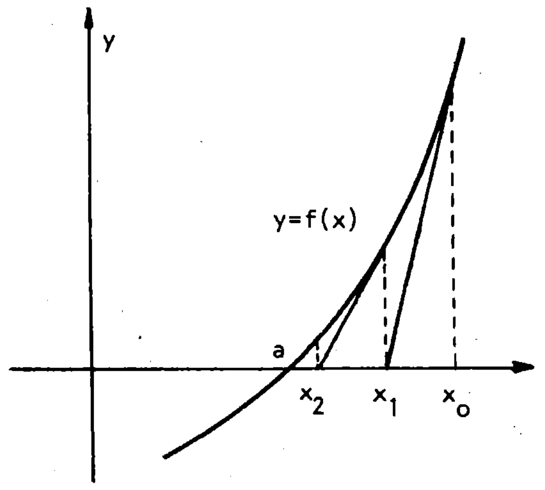


Figure 5.1.1.1

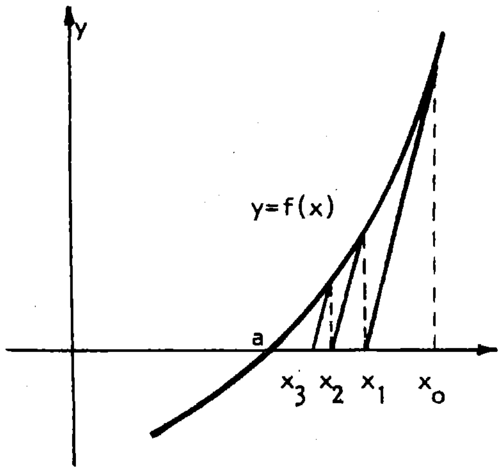


Figure 5.1.1.2

The equality (5.1.1.2) suggests the construction of iterative formula

(5.1.1.3) $$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \quad (k = 0, 1, \ldots).$$

known as Newton's method or tangent method.

We can examine the convergence of iterative process (5.1.1.3) by introducing the additional assumption for function f, namely, assume that $f \in C^2[\alpha, \beta]$. Because the iterative function ϕ is at Newton's method given as

$$\phi(x) = x - \frac{f(x)}{f'(x)},$$

by differentiation we get

(5.1.1.4) $$\phi'(x) = 1 - \frac{f'(x^2) - f(x)f''(x)}{f'(x)^2} = \frac{f(x)f''(x)}{f'(x)^2}.$$

Note that $\phi(a) = a$ and $\phi'(a) = 0$. Being, based on accepted assumptions for f, function ϕ' continuous on $[\alpha, \beta]$, and $\phi'(a) = 0$, there exists a neighborhood of point $x = a$, denoted as $U(a)$ where it holds

(5.1.1.5) $$|\phi'(x)| = \left|\frac{f(x)f''(x)}{f'(x)^2}\right| \leq q < 1.$$

Theorem 5.1.1.1. *If $x_0 \in U(a)$, series $\{x_k\}$ generated using (5.1.1.3) converges to point $x = a$, whereby*

(5.1.1.6) $$\lim_{k \to +\infty} \frac{x_{k+1} - a}{(x_k - a)^2} = \frac{f''(a)}{2f'(a)}.$$

(see [1], pp. 340-341).

Example 5.1.1.1.

Find the solution of equation

$$f(x) = x - \cos x = 0$$

on segment $[0, \pi/2]$ using Newton's method

$$x_{k+1} = x_k - \frac{x_k - \cos x_k}{1 + \sin x_k} = \frac{x_k \sin x_k + \cos x_k}{1 + \sin x_k} \quad (k = 0, 1, \ldots).$$

Note that $f'(x) = 1 + \sin x > 0 (\forall x \in [0, \pi/2])$. Starting with $x_0 = 1$, we get the results given in Table 5.1.1.

•

Table 5.1.1

k	x_k
0	1.000000
1	0.750364
2	0.739133
3	0.739085
4	0.739085

The last two iterations give solution of equation in consideration with six exact figures.

Example 5.1.1.2.

By applying the Newton's method on solution of equation $f(x) = x^n - a = 0$ $(a > 0, n > 1)$ we obtain the iterative formula for determination of n-th root of positive number a

$$x_{k+1} = x_k - \frac{x_k^n - a}{n x_k^{n-1}} = \frac{1}{n}\left\{(n-1)x_k + \frac{a}{x_k^{n-1}}\right\} \quad (k = 0, 1, \ldots).$$

A special case of this formula, for $n = 2$ gives as a result square root.

At application of Newton method it is often problem how to chose initial value of x_0 in order series $\{x_k\}_{k\in\mathcal{N}}$ to be monotonous. One answer to this question was given by Fourier. Namely, if f'' does not change a sign on $[\alpha, \beta]$ and if x_0 is chosen in such a way that $f(x_0)f''(x_0) > 0$, the series $\{x_k\}_{k\in\mathcal{N}}$ will be monotonous. This statement follows from (5.1.1.4).

Based on Theorem 5.1.1.1 we conclude that Newton's method applied to determination of simple root $x = a$ has square convergence if $f''(a) \neq 0$. In this case factor of convergence (asymptotic constant of error) is

$$C_2 = \left|\frac{f''(a)}{2f'(a)}\right|.$$

The case $f''(a)$ is specially to be analyzed. Namely, if we suppose that $f \in C^3[\alpha, \beta]$ one can prove that

$$\lim_{k\to+\infty} \frac{x_{k+1} - a}{(x_k - a)^3} = \frac{f'''(a)}{3f'(a)}.$$

Example 5.1.1.3.

Consider the equation

$$f(x) = x^3 - 3x^2 + 4x - 2 = 0.$$

Because of $f(0) = -2$ and $f(1.5) = 0.625$ we conclude that on segment $[0, 1.5]$ this equation has a root. On the other hand, $f'(x) = 3x^2 - 6x + 4 = 3(x-1)^2 + 1 > 0$, what means that the root is simple, enabling application of Newton's method. Starting with $x_0 = 1.5$, we get the results in Table 5.1.2.

Table 5.1.2

k	x_k
0	1.5000000
1	1.1428571
2	1.0054944
3	1.0000003

The exact value for root is $a = 1$, because $f(x) = (x-1)^3 + (x-1)$.

In order to reduce number of calculations, it is often used the following modification of Newton method

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_0)} \quad (k = 0, 1, \ldots).$$

Geometrically, x_{k+1} represents abscissa of intersection of x-axes and straight line passing through point $(x_k, f(x_k))$ and being parallel to tangent of curve $y = f(x)$ in the point $(x_0, f(x_0))$ (see Figure 5.1.1.2).

Iterative function of such modified Newton's method is

$$\phi_1(x) = x - \frac{f(x)}{f'(x_0)}.$$

Because of $\phi_1(a) = a$ and $\phi_1'(a) = 1 - \dfrac{f'(a)}{f'(x_0)}$, we conclude that method has order of convergence one, i.e. it holds

$$x_{k+1} - a \sim \big(1 - \frac{f'(a)}{f'(x_0)}\big)(x_k - a) \quad (k \to +\infty),$$

whereby the condition

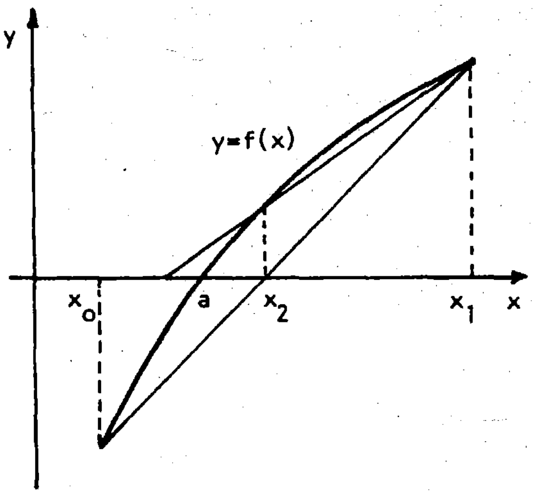
$$\left|1 - \frac{f'(x)}{f'(x_0)}\right| \le q < 1,$$

is analogous to condition (5.1.1.5)

By approximation of first derivative $f'(x_k)$ in Newton's method with divided difference $\dfrac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}$ one gets secant method

$$(5.1.1.7) \qquad x_{k+1} = x_k - \frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})} f(x_k) \quad (k = 1, 2, \ldots),$$

which belongs to open domains methods (two steps method). For starting of iterative process (5.1.1.7) two initial values x_0 and x_1 are needed. Geometrical interpretation of secant method is given in Figure 5.1.3.1.



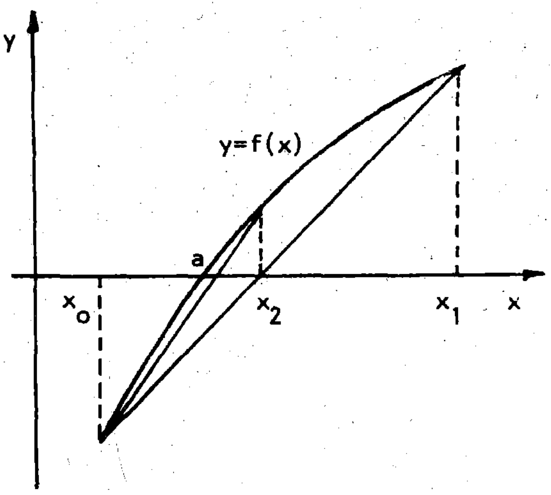


Figure 5.1.3.1 Figure 5.1.3.2

The secant method can be modified in such a way that

$$x_{k+1} = x_k - \frac{x_k - x_0}{f(x_k) - f(x_0)} f(x_k) \quad (k = 1, 2, \ldots). \tag{5.1.1.8}$$

This method is often called **regula falsi** or false position method. Differently from secant method, where is enough to take $x_1 \neq x_0$, at this method one needs to take x_1 and x_0 on different sides of root $x = a$. Geometric interpretation of false position method is given in Figure 5.1.3.2.

5.1.2. Bisection method

Let on segment α, β exist isolated simple root $x = a$ of equation

$$f(x) = 0, \tag{5.1.2.1}$$

where $f \in C[\alpha, \beta]$. Method of interval bisection for solution of equation (5.1.2.1) consists in construction of series' of intervals $\{(x_k, y_k)\}_{k \in N}$ such that

$$y_{k+1} - x_{k+1} = \frac{1}{2}(y_k - x_k), \quad (k = 1, 2, \ldots)$$

having thereby $\lim\limits_{k \to +\infty} x_k = \lim\limits_{k \to +\infty} y_k = a$. The noted process of construction of intervals is interrupted when, for example, interval length becomes lesser than in advance given small positive number ε. This method can be described with four steps:

I. $k := 0,\ x_1 = \alpha,\ y_1 = \beta$;

II. $k := k + 1, z_k := \frac{1}{2}(x_k + y_k)$;

III. If

$$f(z_k)f(x_k) \begin{cases} < 0 \text{ take} & x_{k+1} := x_k,\ y_{k+1} := z_k, \\ > 0 \text{ take} & x_{k+1} := z_k,\ y_{k+1} := y_k, \\ = 0 \text{ take} & a := z_k; \text{ end of calculation} \end{cases}$$

IV. If

$$|y_{k+1} - x_{k+1}| \begin{cases} \geq \varepsilon & \text{go to II}, \\ < \varepsilon & z_{k+1} := \frac{1}{2}(x_{k+1} + y_{k+1}) \\ & \text{end of calculation.} \end{cases}$$

Note that error estimation for approximation of z_{k+1} is

$$|z_{k+1} - a| \leq \frac{1}{2^{k+1}}(\beta - \alpha).$$

5.1.3. Program realization

In this section we present some simple programs for solution of nonlinear equations. It is recommended to reader to write a short code for the same programs in `Mathematica`, and, if having pleasure, in `Pascal`, i.e. `Delphi`.

Example 5.1.3.1.

Write a program for solving equation

$$f(x) = 1 - x^{-a} + (1 - x)^{-a} = 0 \quad (a = 0.5\ (0.1)\ 2.8),$$

using Newton's method, with accuracy $\varepsilon = 10^{-5}$. For initial approximation take $x_0 = 0.5$. On output print value of parameter a, root x and corresponding value of $f(x)$. Criterion for interrupting iterative process is accuracy ε. Namely, we consider the root to be found with accuracy ε if $f(x)$ changes the sign in interval $(x_n - \varepsilon, x_n + \varepsilon)$. The program and output list are of form:

```
C=========================================================
C     SOLVING EQUATION
C     1 - X**(-A) + (1-X)**(-A) = 0
C     BY NEWTON'S METHOD
C=========================================================
      FUNK(X,A)=  1 - X**(-A) + (1-X)**(-A)
      PRIZ(X,A)= A*X**(-A-1) + A*(1-X)**(-A-1)
      OPEN(6,File='NEWT1.out')
      WRITE(6,10)
   10 FORMAT(10X,'A', 10X, 'X', 12X, 'F(X)'/)
      EPS=1.E-5
      DO 11 I=5,28
      A=I*0.1
      X0=0.5
    6 X=X0-FUNK(X0,A)/PRIZ(X0,A)
      IF(FUNK(X+EPS,A)*FUNK(X-EPS,A).LT.0.) GO TO 7
      X0=X
      GO TO 6
    7 Y=FUNK(X,A)
      WRITE(6,20)A,X,Y
   20 FORMAT(9X, F3.1, 5X, F9.6, 5X, F9.6)
   11 CONTINUE
      STOP
      END
```

and the output list of results is

```
          A          X             F(X)
         .5       .219949       -.000014
         .6       .267609        .000000
         .7       .305916       -.000026
         .8       .336722       -.000003
         .9       .361641        .000000
        1.0       .381966        .000000
        1.1       .398689        .000000
        1.2       .412563        .000000
        1.3       .424159       -.000084
```

```
 1.4     .433933     -.000044
 1.5     .442217     -.000024
 1.6     .449281     -.000013
 1.7     .455337     -.000007
 1.8     .460554     -.000005
 1.9     .465068     -.000002
 2.0     .468990     -.000001
 2.1     .472410     -.000001
 2.2     .475402      .000000
 2.3     .478029     -.000001
 2.4     .480340      .000000
 2.5     .482380     -.000001
 2.6     .484184      .000000
 2.7     .485784     -.000001
 2.8     .487205      .000000
```

Example 5.1.3.2.

If the equation $f(x) = 0$ has a root $x = a$ in interval α, β, where $f(\alpha)\, f(\beta) < 0$, write a program for finding a root $x = a$ with accuracy ε, using bisection. Use double precision arithmetic. For program testing use the following example:

$$f(x) = e^x - 2(x-1)^2, \ (\alpha, \beta) = (-0.5, 1.0), \ \varepsilon = 10^{-12}.$$

The program code is

```
C=========================================================
C    SOLVING NONLINEAR EQUATION
C    BY BISECTION
C=========================================================
      DOUBLE PRECISION X,Y,Z,F,FZ,EPS
      F(X)=DEXP(X)-2.*(X-1.)**2
      OPEN(8,File='Bisect.in')
      OPEN(6,File='Bisect.out')
C  PRINTING OF TABLE HEADING
      WRITE(6,9)
    9 FORMAT(2X,'K', 2X,'(',8X,'X(K)',8X,',',8X,'Y(K)',
     18X,')',5X,'F(Z(K))'/ )
C  READ IN THE INITIAL INTERVAL   (ALPHA, BETA)
      READ(8,10)ALFA, BETA
   10 FORMAT(2F5.0)
      EPS=1.D-12
      K=-1
      X=ALFA
      Y=BETA
    5 K=K+1
      Z=0.5*(X+Y)
      FZ=F(Z)
      IF(K/5*5-K.LT.0)GO TO 25
      WRITE(6,20)K,X,Y,FZ
   20 FORMAT(1X,I2,2X,'(', D20.13,',',D20.13,')',2X,D12.5)
   25 IF(FZ*F(X))1,2,3
    1 Y=Z
      GO TO 4
    2 IF(K/5*5-K.EQ.0) GO TO 6
      GO TO 7
    3 X=Z
    4  IF(DABS(Y-X).GE.EPS) GO TO 5
      Z=0.5*(X+Y)
      K=K+1
      FZ=F(Z)
    7 WRITE(6,20) K,X,Y,FZ
    6 WRITE(6,30) Z,EPS
   30 FORMAT(/5X,'A = ', D20.13,' (WITH EXACTNESS EPS = ',
     1 D7.1,')')
      STOP
      END
```

and the output list of results is

```
  K  (        X(K)          ,         Y(K)          )      F(Z(K))
  0  ( -.5000000000000D+00,  .1000000000000D+01)     .15903D+00
  5  (  .2031250000000D+00,  .2500000000000D+00)     .57870D-01
 10  (  .2119140625000D+00,  .2133789062500D+00)    -.29038D-02
 15  (  .2132873535156D+00,  .2133331298828D+00)     .70475D-05
 20  (  .2133073806763D+00,  .2133088111877D+00)    -.23607D-05
 25  (  .2133086323738D+00,  .2133086770773D+00)     .89352D-07
 30  (  .2133086337708D+00,  .2133086351678D+00)     .53733D-09
 35  (  .2133086343383D+00,  .2133086343820D+00)     .58801D-10
 40  (  .2133086343465D+00,  .2133086343479D+00)     .19761D-11
 41  (  .2133086343465D+00,  .2133086343472D+00)     .48077D-12
     A =    .2133086343468D+00 (WITH EXACTNESS EPS =   .1D-11)
```

Example 5.1.3.3.

Write a program for solving nonlinear equation $f(x) = 0$ by regula-falsi method

$$x_i = \frac{x_0 f(x_{i-1}) - x_{i-1} f(x_0)}{f(x_{i-1} - f(x_0)} \quad (i = 2, 3, \ldots).$$

Iterative process interrupt when the condition $f(x_i - \varepsilon)f(x_i + \varepsilon) \leq 0$ is fulfilled. For program testing use the following example:

$$f(x) = e^x - 2(x-1)^2,\ x_0 = -0.5, x_1 = 1.0,\ \varepsilon = 10^{-5}.$$

The program code and output list are of form:

```
C==========================================================
C     SOLVING NONLINEAR EQUATION
C     EXP(X) - 2*(X-1)**2 = 0
C     BY REGULA-FALSI METHOD
C==========================================================
      F(X)=EXP(X)-2.*(X-1.)**2
      OPEN(6,File='Reg-Fal.out')
      WRITE(6,10)
   10 FORMAT(9X,'I', 10X,'Xi',14X,'F(Xi)' / )
      X0= -0.5
      X1=1
      I=2
    3 X=(X0*F(X1)-X1*F(X0))/(F(X1)-F(X0))
      Y=F(X)
      WRITE(6,20)I,X,Y
   20 FORMAT(8X,I2,5X,E14.7,2X, E14.7)
      IF(F(X-1.E-5)*F(X+1.E-5)) 1,1,2
    2 X1=X
      I=I+1
      GO TO 3
    1 STOP
      END
```

and the output list of results is

```
I          Xi               F(Xi)
2       .3833067E+00     .7065066E+00
3       .2476403E+00     .1489089E+00
4       .2200995E+00     .2971124E-01
5       .2146460E+00     .5861370E-02
6       .2135718E+00     .1153714E-02
7       .2133604E+00     .2269801E-03
8       .2133188E+00     .4465835E-04
9       .2133106E+00     .8795895E-05
```

Example 5.1.3.4.

For polynomial of form $P(x) = a_1z^3 + a_2z^2 + a_3z + a_4$ $(a_1 \neq 0)$ write a program for determination of zeros using following algorithm:

1^0 One root find using Newton's method (see previous examples) with accuracy 10^{-7} $(|x_{n+1} - x_n| < 10^{-7})$;

2^0 With, in this way obtained zero z_1 evaluate coefficients of polynomial $Q(z) = P(z)/(z - z_1)$;

3^0 Solve the square equation $Q(z) = 0$ by standard formula.

For calculation of polynomial value use subroutine of type FUNCTION. Algorithm steps 1^0 and 2^0 realize in subprogram of type SUBROUTINE. The program should solve arbitrary number of equations. On output print coefficients of polynomials P and Q and roots of equation $P(z) = 0$.

For program testing use the following polynomials. $P(z) = 3z^3 - 7z^2 + 8z - 2$ and $P(z) = z^3 - 5z^2 - z + 5$.

For calculation of polynomial value, subprogram of type FUNCTION, named PL using Horner's scheme, is written. Arguments in parameter list have the following meaning:

Z - value of argument;
A - polynomial coefficients;
N - degree of polynomial.

This subprogram obtains polynomials $P(z)$ and $P'(z)$.

```
      FUNCTION PL(Z,A,N)
      DIMENSION A(1)
      PL=A(1)
      DO 10 I=1,N
   10 PL=PL*Z+A(I+1)
      RETURN
      END
C
C
      SUBROUTINE KJ(A,B,C,X1,Y1,X2,Y2)
      D=B*B-4*A*C
      IF(D) 25,10,20
   10 X1=-B/2./A
      X2=X1
   15 Y1=0.
      Y2=0.
      RETURN
   20 X1=(-B+SQRT(D))/2./A
      X2=(-B-SQRT(D))/2./A
      GO TO 15
   25 X1=-B/2./A
      X2=X1
      Y1=SQRT(-D)/2./A
      Y2=-Y1
      RETURN
      END
C
C
      SUBROUTINE NEWT(A,B,N,Z1)
      DIMENSION A(1), B(1)
C  EVALUATION OF COEFFICIENTS P'(Z)
      DO 5 I=1,3
 5    B(I)=A(I)*(4-I)
C  EVALUATION OF REAL ROOT  Z(1)
      Z0=0.
 10   Z1=Z0-PL(Z0,A,N)/PL(Z0,B,N-1)
      IF(ABS(Z1-Z0)-1.E-7) 20,15,15
 15   Z0=Z1
      GO TO 10
C  EVALUATION OF COEFFICIENTS Q(Z)
 20   B(1)= A(1)
      DO 25 I=2,3
 25   B(I)=A(I)+B(I-1)*Z1
      RETURN
      END
```

For solving of square equation $Q(z) = az^2 + bz + c = 0$ we formed subprogram KJ. Arguments in parameter list of subprogram are of following meaning:

A, B, C - coefficients of equation;
X1, Y1 - real and imaginary part of the first root of equation;
X2, Y2 - real and imaginary part of the second root of equation.

For algorithm steps 1^0 and 2^0 the subroutine NEWT is written, with following arguments:

A - coefficients of polynomial P;
B - coefficients of polynomial P' and Q;
N - degree of polynomial P ($N = 3$);
Z1 - real root of equation $P(z) = 0$ obtained by Newton's method.

Main program and output list of results are of following form:

```
C=====================================================
C    SOLVING NONLINEAR EQUATION
C    OF DEGREE THREE
C=====================================================
     DIMENSION A(4), B(3), ZR(3), ZI(3)
     OPEN(6,File='POL.OUT')
     OPEN(8,File='POL.IN')
  5  READ(8,10,END=99)(A(I),I=1,4)
 10  FORMAT(4F10.0)
     IF(A(1)) 15,99,15
 15  CALL NEWT(A,B,3,Z1)
     ZR(1)=Z1
     ZI(1)=0.
     WRITE(6,20)  (I,A(I), I=1,4)
 20  FORMAT(/ 22X, 'COEFFICIENTS OF POLYNOMIAL P(Z)'// 5X,
    *4('A(',I1,')=',F8.5,3X)//)
     WRITE(6,25) (I, B(I),I=1,3)
 25  FORMAT(/23X,'COEFFICIENTS OF POLYNOMIAL Q(Z)'//5X,
    *3('B(',I1,')=',F8.5,3X)// )
     WRITE(6,30)
 30  FORMAT(/23X, '  ZEROS OF POLYNOMIAL P(Z)'//27X,
    *'REAL',8X,'IMAG'/  )
     CALL KJ(B(1),B(2),B(3),ZR(2),ZI(2),ZR(3),ZI(3))
     WRITE(6,35) (I,ZR(I),ZI(I),I=1,3)
 35  FORMAT(/18X,'Z(',I1, ')=', 2F12.7)
     GO TO 5
 99   STOP
      END
                        COEFFICIENTS OF POLYNOMIAL P(Z)
  A(1)= 3.00000    A(2)=-7.00000   A(3)= 8.00000   A(4)=-2.00000
                         COEFFICIENTS OF POLYNOMIAL Q(Z)
  B(1)= 3.00000    B(2)=-6.00000   B(3)= 6.00000
                          ZEROS OF POLYNOMIAL P(Z)
                            REAL         IMAG
                 Z(1)=    .3333333     .0000000
                 Z(2)=   1.0000000    1.0000000
                 Z(3)=   1.0000000   -1.0000000
                        COEFFICIENTS OF POLYNOMIAL P(Z)
  A(1)= 1.00000    A(2)=-5.00000   A(3)=-1.00000   A(4)= 5.00000
                         COEFFICIENTS OF POLYNOMIAL Q(Z)
  B(1)= 1.00000    B(2)=  .00000   B(3)=-1.00000
                          ZEROS OF POLYNOMIAL P(Z)
                            REAL         IMAG
                 Z(1)=   5.0000000     .0000000
                 Z(2)=   1.0000000     .0000000
                 Z(3)=  -1.0000000     .0000000
```

Example 5.1.3.5.

Write a program for evaluation of coefficients of polynomial of form

$$P(z) = C_{n+1}z^n + C_n z^{n-1} + \ldots + C_2 z + C_1 \quad (C_{n+1} = 1)$$

if all zeros $z_k = x_k + i\,y_k$ $(k = 1, \ldots, n)$ are known.
Let

$$P_k(z) \overset{\text{def}}{=} \prod_{i=1}^{k}(z - z_i) = C_{k+1}^{(k)}z^k + C_k^{(k)}z^{k-1} + \ldots + c_2^{(k)}z + C_1^{(k)}.$$

Then for polynomial $P(z)$ it holds $P(z) = P_n(z)$, i.e. $C_i = C_i^{(n)}$ $(i = 1, \ldots, n+1)$. Because of

$$P_k(z) = (z - z_k)P_{k-1}(z)$$

the following recurrence relations hold

$$\begin{aligned} C_1^{(k)} &= -z_k C_1^{(k-1)}, \\ C_i^{(k)} &= C_{i-1}^{(k-1)} - z_k C_i^{(k-1)} \qquad (i = 2, \ldots, k), \\ C_{k+1}^{(k)} &= C_k^{(k-1)}. \end{aligned}$$

starting with $C_1^{(1)} = -z_1$, $C_2^{(1)} = 1$.
Based on previous, the subroutine VIETE is written, with following arguments:
Z - vector of zeros of length N;
N - degree of polynomial;
C - polynomial coefficients;
KB- flag (KB = 0 for correctly given polynomial, KB = 1 for polynomial with degree less than one).
Program routines are realized in complex arithmetic. Main program, subroutine, and output list with results are given as follows.

```
C=====================================================
C     EVALUATION OF POLYNOMIAL COEFFICIENTS
```

```
C    FROM GIVEN POLYNOMIAL ROOTS
C===========================================================
      COMPLEX Z(10), C(11)
      OPEN(6,File='VIETE.OUT')
      OPEN(8,File='VIETE.IN')
C  READ-IN POLYNOMIAL ZEROS
  2   READ(8,10,END=99) N, (Z(I),I=1,N)
 10   FORMAT(I2/(10F8.2))
C  SUBPROGRAM VIETE
      CALL VIETE(Z,N,C,KB)
      IF(KB.EQ.0) GO TO 1
      WRITE(6,20)
 20   FORMAT(/5X,'DATA ERROR:'
     *' POLYNOMIAL DEGREE LESS THAN 1'//)
      GO TO 2
C PRINTING ZEROS AND COEFFICIENTS OF POLYNOMIAL
  1   WRITE(6,25) (i,Z(I), I=1,N)
 25   FORMAT(/16X,'POLYNOMIAL ZEROS'//23X,'REAL',5X,
     *'IMAG'//(10X,'Z(',I2,')=',4X,F10.7,1X,F10.7) )
      N=N+1
      WRITE(6,30) (I,C(I), I=1,N)
 30   FORMAT(/16X,'POLYNOMIAL COEFFICIENTS'//23X,'REAL',5X,
     *'IMAG'//(10X,'C(',I2,')=',4X,F10.7,1X,F10.7) )
      GO TO 2
 99   STOP
      END
C
      SUBROUTINE VIETE(Z,N,C,KB)
      COMPLEX Z(1),C(1),A,B
      IF(N.GE.1) GO TO 5
      KB=1
      RETURN
  5   KB=0
      C(1)=-Z(1)
      C(2)=1.
      IF(N.GE.2) GO TO 20
      RETURN
 20   DO 15 K=2,N
      A=C(1)
      C(1)=-Z(K)*C(1)
      DO 10 I=2,K
      B=C(I)
      C(I)=A-Z(K)*B
 10   A=B
 15   C(K+1)=A
      RETURN
      END
```

```
                POLYNOMIAL ZEROS
                       REAL     IMAG
          Z( 1)=     1.0000000  1.0000000
          Z( 2)=     1.0000000  1.0000000
          Z( 3)=     1.0000000 -1.0000000
                POLYNOMIAL COEFFICIENTS
                       REAL     IMAG
          C( 1)=    -2.0000000 -2.0000000
          C( 2)=     4.0000000  2.0000000
          C( 3)=    -3.0000000 -1.0000000
          C( 4)=     1.0000000   .0000000
```

Example 5.1.3.6.

Write a program for evaluation of complex root of transcendental equation $f(z) = 0$ using Newton's method

$$z_{n+1} = z_n - \frac{f(z_n)}{f'(z_n)} \quad (f'(z_n) \neq 0),$$

where $z_n = x_n + i\,y_n \quad (n = 0, 1, \ldots)$. Iterative process is to be interrupted when the conditions

$$|x_{n+1} - x_n| \leq \varepsilon, \qquad |y_{n+1} - y_n| \leq \varepsilon,$$

where ε is in advance given accuracy, are contemporary fulfilled.

Program is organized in the following way:

1^0 Functions $\text{Re}(f(z)), \text{Im}(f(z)), \text{Re}(f'(z), \text{Im}(f'(z)))$ are given in subroutine of type FUNCTION;

2^0 In subprogram of type SUBROUTINE is calculated one iterative step of Newton's method;

3^0 Main program reads in initial values x_0, y_0, ε, calls subroutine for iteration, checks the termination condition and prints result.

The example taken is $f(z) = e^x - 0.2z + 1, z_0 = 1 + \pi\,i, \varepsilon = 10^{-6}$. Two separate programs are written, in real and in complex arithmetics.

By separation of real and imaginary part in Newton's formula, one gets

$$\begin{aligned}
x_{n+1} &= x_n - \frac{1}{\Delta}(\text{Re}(f(z_n))\text{Re}(f'(z_n)) + \text{Im}(f(z_n))\text{Im}(f'(z_n)))\\
y_{n+1} &= y_n - \frac{1}{\Delta}(\text{Im}(f(z_n))\text{Re}(f'(z_n)) - \text{Re}(f(z_n))\text{Im}(f'(z_n))),
\end{aligned}$$

where $\Delta = |f'(z_n)|^2 = |\text{Re}(f'(z_n))|^2 + |\text{Im}(f'(z_n))|^2$.

Being, in our case, $f(z) = e^z - 0.2z + 1$ and $f'(z) = e^z - 0.2$, we have $\text{Re}(f(z)) = e^z \cos y - 0.2x + 1$, $\text{Im}(f(z)) = e^x \sin y - 0.2y$, $\text{Re}(f'(z)) = e^x \cos y - 0.2$, $\text{Im}(f'(z) = e^x \sin y)$, what is given by function subroutine EF.

Program routines are realized in real and complex arithmetic. Main program, subroutine, and output list with results are given as follows.

```
C=======================================================
C    EVALUATION OF COMPLEX ROOT OF TRANSCENDENT
C    EQUATION F(Z)=0 BY NEWTON'S METHOD
C    USING REAL ARITHMETIC
C=======================================================
      OPEN(6,File='NEWT-TRR.OUT')
      OPEN(8,File='NEWT-TRR.IN')
      READ(8,5) XO, YO, EPS
  5   FORMAT(2F10.0,E5.0)
      WRITE(6,10)
 10   FORMAT(//10X, 'NEWTON''S METHOD FOR SOLVING TRANSCEN'
     *'DENT EQUATION'//18X,'F(Z)=EXP(Z) - 0.2*Z + 1 = 0'
     *//5X,'ITER.No.',4x,'REAL(Z)',5X,'IMAG(Z)',4X,
     *'REAL(F(Z))',2X,'IMAG(F(Z))'/)
      ITER=0
      KBR=1
 15   A=EF(XO,YO,1)
      B=EF(XO,YO,2)
      WRITE(6,20) ITER,XO,YO,A,B
 20   FORMAT(5X,I4,2X,2F13.7,2F12.6)
      GO TO (22,50),KBR
 22   ITER=ITER + 1
```

```
      XS=X0
      YS=Y0
      CALL TRANS(X0,Y0,A,B,R)
      IF(R) 25,25,35
 25   WRITE (6,30)
 30   FORMAT(//5X,'FIRST DERIVATIVE OF FUNCTION = 0')
      GO TO 50
 35   IF(ABS(X0-XS)-EPS) 40,40,15
 40   IF(ABS(YS-Y0)-EPS) 45,45,15
 45   KBR=2
      GO TO 15
 50   WRITE(6,55)EPS
 55   FORMAT(/5X,'SPECIFIED ACCURACY OF CALCULATION '
     *'EPSYLON = ',E7.1)
      STOP
      END
C
C
      FUNCTION EF(X,Y,I)
      GO TO(10,20,30,40),I
 10   EF=EXP(X)*COS(Y)-0.2*X+1
      RETURN
 20   EF=EXP(X)*SIN(Y)-0.2*Y
      RETURN
 30   EF=EXP(X)*COS(Y)-0.2
      RETURN
 40   EF=EXP(X)*SIN(Y)
      RETURN
      END
C
C
      SUBROUTINE TRANS(X0,Y0,A,B,R)
      C=EF(X0,Y0,3)
      D=EF(X0,Y0,4)
      R=C*C+D*D
      IF(R)  5,10,5
  5   X0=X0-(A*C-B*D)/R
      Y0=Y0-(B*C-A*D)/R
 10   RETURN
      END
```

```
        NEWTON'S METHOD FOR SOLVING TRANSCENDENT EQUATION
                F(Z)=EXP(Z) - 0.2*Z + 1 = 0
 ITER.No.    REAL(Z)     IMAG(Z)    REAL(F(Z))   IMAG(F(Z))
    0       1.0000000   3.1415920   -1.918282    -.628317
    1        .3426673   2.9262880    -.444708    -.284296
    2        .0372190   2.7002840     .054076    -.096737
    3        .0497327   2.6425620     .067235    -.025535
    4        .0911207   2.6459620     .018186    -.008234
    5        .1015006   2.6458960     .006090    -.002721
    6        .1049549   2.6459040     .002026    -.000909
    7        .1060995   2.6459040     .000678    -.000304
    8        .1064820   2.6459040     .000227    -.000102
    9        .1066103   2.6459040     .000076    -.000034
   10        .1066533   2.6459040     .000026    -.000011
   11        .1066677   2.6459040     .000009    -.000004
   12        .1066726   2.6459040     .000003    -.000001
   13        .1066742   2.6459040     .000001     .000000
   14        .1066747   2.6459040     .000000     .000000
```

```
  SPECIFIED ACCURACY OF CALCULATION EPSYLON =  .1E-05
```

The given subprograms and main program can be rather simple realized in complex arithmetic, what is, with results, given in continuation.

```
C=========================================================
C    EVALUATION OF COMPLEX ROOT OF TRANSCENDENT
C    EQUATION F(Z)=0 BY NEWTON'S METHOD
C    USING COMPLEX ARITHMETIC
C=========================================================
      COMPLEX Z,Z0,F,Y,A
      OPEN(6,File='NEWT-TRC.OUT')
      OPEN(8,File='NEWT-TRC.IN')
      READ(8,10) Z0
 10   FORMAT(2E14.7)
      EPS=1.E-6
      WRITE(6,20)
 20   FORMAT(//10X, 'NEWTON''S METHOD FOR SOLVING TRANSCEN'
     *'DENT EQUATION'//18X,'F(Z)=EXP(Z) - 0.2*Z + 1 = 0'
     *//5X,'ITER.No.',4x,'REAL(Z)',5X,'IMAG(Z)',4X,
     *'REAL(F(Z))',2X,'IMAG(F(Z))'/)
      ITER=0
      Y=F(Z0,1)
 13   WRITE(6,30)ITER,Z0,Y
 30   FORMAT(5X,I4,2X,2F13.7,2F12.6)
      Y=F(Z0,2)
      B=CABS(Y)
      IF(B.EQ.0.) GO TO 99
      CALL NEW(Z,Z0)
      ITER=ITER+1
      Y=F(Z,1)
      A=Z-Z0
      IF(ABS(REAL(A)).GT.EPS) GO TO 95
      IF(ABS(AIMAG(A)).LE.EPS) GO TO 98
 95   Z0=Z
      GO TO 13
 99   WRITE(6,40)
 40   FORMAT(//5X,'FIRST DERIVATIVE OF FUNCTION = 0')
      GO TO 97
 98   WRITE(6,30) ITER,Z,Y
 97   WRITE(6,55) EPS
 55   FORMAT(/5X,'SPECIFIED ACCURACY OF CALCULATION '
     *'EPSYLON = ',E7.1)
      STOP
      END
C
C
      COMPLEX FUNCTION F(Z,I)
      COMPLEX Z
      GO TO(1,2)I
 1    F=CEXP(Z) - 0.2*Z + 1
      RETURN
 2    F=CEXP(Z) - 0.2
      RETURN
      END
C
C
      SUBROUTINE NEW(Z,Z0)
      COMPLEX Z,Z0,F
      Z=Z0 - F(Z0,1)/F(Z0,2)
      RETURN
      END
```

```
        NEWTON'S METHOD FOR SOLVING TRANSCENDENT EQUATION
                  F(Z)=EXP(Z) - 0.2*Z + 1 = 0
   ITER.No.     REAL(Z)       IMAG(Z)     REAL(F(Z))  IMAG(F(Z))
      0       1.0000000     3.1415920    -1.918282    -.628317
      1        .3426675     2.9262880     -.444709    -.284296
      2        .1036775     2.7002840     -.023705    -.066273
      3        .1054019     2.6458710      .001517    -.000634
      4        .1066756     2.6459040     -.000001     .000000
      5        .1066750     2.6459040      .000000     .000000
   SPECIFIED ACCURACY OF CALCULATION EPSYLON =  .1E-05
```

5.2. Systems of nonlinear equations

5.2.1. Newton-Kantorowitch (Raphson) method

Consider system of nonlinear equations

$$f_i(x_1,\ldots,x_n)=0 \quad (i=1,\ldots,n). \tag{5.2.1.1}$$

By taking $\vec{x}=[x_1\ldots x_n]^T$, $\theta=[0\ldots 0]^T$, where θ is null-vector, we can write

$$\vec{f}(\vec{x})=\begin{bmatrix} f_1(x_1,\ldots,x_n)\\ \vdots \\ f_n(x_1,\ldots,x_n)\end{bmatrix}=\theta. \tag{5.2.1.2}$$

Basic iterative method for solving equations (5.2.1.2) is method of Newton-Kantorowich, which generalizes classical Newton's method. Fundamental results regarding existence and uniqueness of solutions of eq. (5.2.1.2) and convergence of the method are given by L.V. Kantorowich (see [22]).

Let $\vec{a}=[a_1\ \ldots a_n]^T$ be exact solution of this system. Using Taylor development for functions which appear in (5.2.1.1), we get

$$\begin{aligned} f_i(a_1,\ldots,a_n)&=f_i(x_1^{(k)},\ldots,x_n^{(k)})+\frac{\partial f_i}{\partial x_1}(a_1-x_1^{(k)})+\ldots\\ &+\frac{\partial f_i}{\partial x_n}(a_n-x_n^{(k)})+r_i^{(k)} \quad (i=1,\ldots,n),\end{aligned}$$

where partial derivatives on the right-hand side of given equations are calculated in point $\vec{x}^{(k)}$. $r_i^{(k)}$ represents corresponding remainder term in Taylor's formula.

Because of $f_i(a_1,\ldots,a_n)=0$ $(i=1,\ldots,n)$, previous system of equations can be represented in matrix form

$$\vec{0}=\vec{f}(\vec{x}^{(k)})+W(\vec{x}^{(k)})(\vec{a}-(\vec{x}^{(k)})+\vec{r}^{(k)},$$

and

$$W(\vec{x})=\begin{bmatrix}\frac{\partial f_1}{\partial x_1} & \ldots & \frac{\partial f_1}{\partial x_n}\\ \vdots & & \\ \frac{\partial f_n}{\partial x_1} & \ldots & \frac{\partial f_n}{\partial x_n}\end{bmatrix},$$

where $\vec{r}^{(k)} = [r_1^{(k)} \ldots r_n^{(k)}]^T$. If Jacobian matrix for $\vec{f}$ is regular, then we have

$$\vec{a} = \vec{x}^{(k)} - W^{-1}(\vec{x}^{(k)})\vec{f}(\vec{x}^{(k)}) - W^{-1}(\vec{x}^{(k)})\vec{r}^{(k)}.$$

By neglecting the very last member on the right-hand size, in place of of vector $\vec{a}$ we get its new approximation, denoted with $\vec{x}^{(k+1)}$. In this way, one gets

$$(5.2.1.3) \qquad \vec{x}^{(k+1)} = \vec{x}^{(k)} - W^{-1}(\vec{x}^{(k)})\vec{f}(\vec{x}^{(k)}) \quad (k = 0, 1, \ldots),$$

where $\vec{x}^{(k)} = [x_1^{(k)} \ \ldots \ x_n^{(k)}]^T$. This method is often called Newton-Raphson method.

Method (5.2.1.3) can be modified in the sense that inverse matrix of $W(\vec{x})$ is not evaluated at every step, but only at first. Thus,

$$(5.2.1.4) \qquad \vec{x}^{(k+1)} = \vec{x}^{(k)} - W^{-1}(\vec{x}^{(0)})\vec{f}(\vec{x}^{(k)}) \quad (k = 0, 1, \ldots).$$

In [1, pp. 384-386] the Newton-Kantorowich method is illustrated with system of nonlinear equation in two unknowns. It is suggested to reader to write a program code in `Mathematica` and `Fortran`.

Example 5.2.1.1. *Solve the system of nonlinear equation*

$$f_1(x_1, x_2) = 9x_1^2x_2 + 4x_2^2 - 36 = 0$$
$$f_2(x_1, x_2) = 16x_2^2 - x_1^4 + x_2 + 1 = 0,$$

which has a solution in first quadrant $(x_1, x_2 > 0)$.

Using graphic presentation of implicit functions f_1 and f_2 in first quadrant, one can see that solution $\vec{a}$ is located in the neighborhood of point $(2, 1)$, so that we take for initial vector $\vec{x}^{(0)} = [2 \ 1]^T$, i.e. $x_1^{(0)} = 2$ and $x_2^{(0)} = 1$.

By partial derivation of f_1 and f_2 one gets the Jacobian

$$W(x) = \begin{bmatrix} 18x_1x_2 & 9x_1^2 + 8x_2 \\ -4x_1^3 & 32x_2 + 1 \end{bmatrix},$$

and its inverse

$$W^{-1}(\vec{x}) = \frac{1}{\Delta(\vec{x})}\begin{bmatrix} 32x_2 + 1 & -(9x_1^2 + 8x_2) \\ 4x_1^3 & 18x_1x_2 \end{bmatrix},$$

where

$$\Delta(\vec{x}) = 18x_1x_2(32x_2 + 1) + 4x_1^3(9x_1^2 + 8x_2).$$

By putting $f_i^{(k)} \equiv f_i(x_1^{(k)}, x_2^{(k)})$ and $\Delta_k \equiv \Delta(x^{(k)})$ $(i = 1, 2; \ k = 0, 1 \ldots)$ in the scalar form of Newton-Kantorowich formula (5.2.1.3), we get the iteration formula

$$x_1^{(k+1)} = x_1^{(k)} - \frac{1}{\Delta_k}\{(32x_2^{(k)} + 1)f_1^{(k)} - (9x_1^{(k)^2} + 8x_2^{(k)})f_2^{(k)}\},$$

$$x_2^{(k+1)} = x_2^{(k)} - \frac{1}{\Delta_k}\{4x_1^{(k)^3}f_1^{(k)} + 18x_1^{(k)}x_2^{(k)}f_2^{(k)}\}.$$

The appropriate `Fortran` code for solving given nonlinear equation is

```
Double precision x1,x2,x10,x11,x20,x21,f1,f2,Delta,EPS
F1(x1,x2)=9*x1**2*x2 + 4*x2**2-36
F2(x1,x2)=16*x2**2 - x1**4 + x2 + 1
```

```
      Delta(x1,x2)=18*x1*x2*(32*x2+1)+4*x1**3*(9*x1**2+8*x2)
      Open(1, File='Newt-Kant.out')
      x10=2.d0
      x20=1.d0
      EPS=1.d-6
      Iter=0
      write(1,5)
 5    format(1h ,// 3x, 'i',7x,'x1(i)',9x,'x2(i)',
    * 9x,'f1(i)', 9x,'f2(i)'/)
      write(1,10)Iter, x10,x20,F1(x10,x20),F2(x10,x20)
 1    x11=x10-((32*x20+1)*f1(x10,x20)-(9*x10**2+8*x20)*
    * f2(x10,x20)) /Delta(x10,x20)
      x21=x20-(4*x10**3*f1(x10,x20)+18*x10*x20*f2(x10,x20))
    * /Delta(x10,x20)
      Iter=Iter+1
      write(1,10)Iter, x11,x21,F1(x11,x21),F2(x11,x21)
10    Format(1x,i3, 4D14.8,2x)
      If(Dabs(x10-x11).lt.EPS.and.Dabs(x20-x21).lt.EPS)stop
      If(Iter.gt.100)Stop
      x10=x11
      x20=x21
      go to 1
      End
```

and the output list of results is

```
  i       x1(i)         x2(i)         f1(i)         f2(i)
  0 .20000000D+01 .10000000D+01 .40000000D+01 .20000000D+01
  1 .19830508D+01 .92295840D+00 .73136345D-01 .88110835D-01
  2 .19837071D+01 .92074322D+00-.28694053D-04 .68348441D-04
  3 .19837087D+01 .92074264D+00-.10324186D-10-.56994853D-10
  4 .19837087D+01 .92074264D+00 .00000000D+00-.15543122D-14
```

Example 5.2.1.2. *Write program for approximative solving a system of equations*

$$F(x, y) = 0,$$
$$G(x, y) = 0,$$

where F and G are continuous differentiable functions, using Newton-Raphson method

$$\begin{aligned} x_{n+1} &= x_n - \Delta x_n \\ & \qquad\qquad (n = 0, 1, 2, \ldots) \\ y_{n+1} &= y_n - \Delta y_n, \end{aligned}$$

starting with some approximate values of x_0 and y_0, where

$$J(x_n, y_n) = \begin{vmatrix} F'_x(x_n, y_n) & F'_y(x_n, y_n) \\ G'_x(x_n, y_n) & G'_y(x_n, y_n) \end{vmatrix} \neq 0.$$

$$\Delta x_n = \frac{1}{J(x_n, y_n)} \begin{vmatrix} F_x(x_n, y_n) & F'_y(x_n, y_n) \\ G(x_n, y_n) & G'_y(x_n, y_n) \end{vmatrix},$$

$$\Delta y_n = \frac{1}{J(x_n, y_n)} \begin{vmatrix} F'_x(x_n, y_n) & F(x_n, y_n) \\ G'_x(x_n, y_n) & G(x_n, y_n) \end{vmatrix}.$$

Partial derivatives are to be obtained numerically. The iterations interrupt when the conditions

$$|x_{n+1} - x_n| \leq \varepsilon, \text{ and } |y_{n+1} - y_n| \leq \varepsilon,$$

are fulfilled. ε is accuracy given in advance. For test example take

$$F(x, y) = 2x^3 - y^2 - 1 = 0$$
$$G(x, y) = xy^3 - y - 4 = 0,$$

with initial values $x_0 = 1.2$, $y_0 = 1.7$ and $\varepsilon = 10^{-10}$.

For obtaining partial derivatives of function $f(x, y)$ we use following expressions

$$\frac{\partial f}{\partial x} \cong \frac{f(x+h, y) - f(x-h, y)}{2h},$$
$$\frac{\partial f}{\partial y} \cong \frac{f(x, y+h) - f(x, y-h)}{2h},$$

where h is small enough increment (here is taken $h = 10^{-5}$). Functions F and G are given in subprogram of type FUNCTION. Program is realized in double precision arithmetic. Program code is of following form:

```
C==========================================================
C     SOLVING OF SYSTEM OF NONLINEAR EQUATIONS
C     BY NEWTON-RAPHSON METHOD
C==========================================================
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION R(2)
      OPEN(6,File='NEWT-RAPH.OUT')
      OPEN(8,File='NEWT-RAPH.IN')
      READ(8,15) XP,YP,EPS,H
 15   FORMAT(4D10.0)
      WRITE(6,16)
 16   FORMAT(/10X,'NEWTON-RAPHSON''S METHOD FOR SYSTEMS OF'
     *' EQUATIONS'///30X,'2.*X**3-Y**2-1.=0.'//
     *30X,'X*Y**3-Y-4.=0.'///1X,'ITER.',6X,'X',15X,'Y',12X,
     *'F(X,Y)',10X,'G(X,Y)'/)
      ITER=0
 20   ITER=ITER+1
      CALL NEWT(XP,YP,XK,YK,FD,H,A,B)
      IF(FD) 5,6,5
  6   WRITE(6,17)
 17   FORMAT(/1X,'JACOBI MATRIX SINGULAR')
      GO TO 90
  5   DO 8 I=1,2
  8   R(I)=EEFF(I,XK,YK)
      WRITE(6,18) ITER,XK,YK,(R(J),J=1,2)
 18   FORMAT(1X,I3,4F15.10)
      IF(DABS(A)-EPS) 30,30,40
 30   IF(DABS(B)-EPS) 90,90,40
 40   XP=XK
      YP=YK
      GO TO 20
 90   STOP
      END
C
C
      SUBROUTINE NEWT(XP,YP,XK,YK,FD,H,A,B)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION R(2),DX(2),DY(2)
      X=XP
      Y=YP
      DO 4 I=1,2
      R(I)=EEFF(I,X,Y)
      DX(I)=0.5D0/H*(EEFF(I,X+H,Y)-EEFF(I,X-H,Y))
    4 DY(I)=0.5D0/H*(EEFF(I,X,Y+H)-EEFF(I,X,Y-H))
      FD=DX(1)*DY(2)-DY(1)*DX(2)
      IF(FD) 5,6,5
    5 A=(R(1)*DY(2)-R(2)*DY(1))/FD
      B=(R(2)*DX(1)-R(1)*DX(2))/FD
      XK=X-A
      YK=Y-B
    6 RETURN
      END
C
C
      FUNCTION EEFF(J,X,Y)
      IMPLICIT REAL*8 (A-H,O-Z)
      GO TO (50,60),J
   50 EEFF=2.D0*X**3-Y*Y-1.D0
      RETURN
   60 EEFF=X*Y**3-Y-4.D0
      RETURN
      END
```

and the output list of results is

```
          NEWTON-RAPHSON'S METHOD FOR SYSTEMS OF EQUATIONS
                                  2.*X**3-Y**2-1.=0.
                                  X*Y**3-Y-4.=0.
 ITER.      X              Y              F(X,Y)          G(X,Y)
   1   1.2348762633   1.6609796808    .0073200054    -.0022831784
   2   1.2342746753   1.6615262759    .0000023823    -.0000008838
   3   1.2342744841   1.6615264668    .0000000000     .0000000000
   4   1.2342744841   1.6615264668    .0000000000     .0000000000
```

5.2.2. Gradient method

Because Newton-Kantorowich method demands obtaining inverse operator $F'^{-1}_{(u)}$, what can be very complicated, and even impossible, it have been developed a whole class of quasi-Newton methods, which use some approximations of noted operator (see [21], [22], [23], [24]). One of this methods is gradient method.

Consider system of nonlinear equations (5.2.1.1), with matrix form

$$\vec{f}(\vec{x}) = \vec{0}. \tag{5.2.2.1}$$

The gradient method for solving a given system of equations is based on minimization of functional

$$U(\vec{x}) = \sum_{i=1}^{n} f_i(x_1, \ldots, x_n)^2 = (\vec{f}(\vec{x}), \vec{f}(\vec{x})).$$

It is easy to show that the equivalence $U(\vec{x}) = 0 \iff \vec{f}(\vec{x}) = \vec{0}$ holds.

Suppose that equation (5.2.2.1) has unique solution $\vec{x} = \vec{a}$, for which functional U riches minimum value. Let $\vec{x}^{(0)}$ be initial approximation of this solution. Let us construct series $\{\vec{x}^{(k)}\}$ such that $U(\vec{x}^{(0)}) > U(\vec{x}^{(1)}) > U(\vec{x}^{(2)}) > \cdots$. In a same way as at linear equations, we take

$$\vec{x}^{(k+1)} = \vec{x}^{(k)} - \lambda_k \nabla U(\vec{x}^{(k)}) \quad (k = 0, 1, \ldots), \tag{5.2.2.2}$$

where $\nabla U(\vec{x}) = \text{grad}(\vec{x}) = \left[\frac{\partial U}{\partial x_1} \cdots \frac{\partial U}{\partial x_n}\right]^T$. Parameter λ_k is to be determined from condition that scalar function S, defined with $S(t) = U(\vec{x}^{(k)} - t\nabla U(\vec{x}^{(k)}))$ has a minimum value in point $t = \lambda_k$. Having in mind that equation $S'(t) = 0$ is non-linear, proceed its linearization around $t = 0$. In this case we have

$$L_i^{(k)} = f_i(\vec{x}^{(k)} - t\nabla U(\vec{x}^{(k)})) = f_i(\vec{x}^{(k)}) - t(\nabla f_i(\vec{x}^{(k)}), \nabla U(\vec{x}^{(k)}))$$

so that linearized equation is

$$\sum_{i=1}^{n} L_i^{(k)} \frac{d}{dt} L_i^{(k)} = -\sum_{i=1}^{n} L_i^{(k)} (\nabla f_i \vec{x}^{(k)}), \nabla U(\vec{x}^{(k)})) = 0,$$

wherefrom we obtain

$$\lambda_k = t = \frac{\sum_{i=1}^{n} H_i f_i(\vec{x}^{(k)})}{\sum_{i=1}^{n} H_i^2}, \tag{5.2.2.3}$$

where we put $H_i = (\nabla f_i(\vec{x}^{(k)}),\ \nabla U(\vec{x}^{(k)}))$ $(i = 1, \ldots, n)$. Because of

$$\frac{\partial U}{\partial x_j} = \frac{\partial}{\partial x_j}\left\{\sum_{i=1}^{n} f_i(\vec{x})^2\right\} = 2\sum_{i=1}^{n} f_i(\vec{x}) \frac{\partial f_i(\vec{x})}{\partial x_j},$$

we have

$$\nabla U(\vec{x}) = 2\mathbf{W}^T(\vec{x})\vec{f}(\vec{x}), \tag{5.2.2.4}$$

where $\mathbf{W}(\vec{x})$ is Jacobian matrix.

According to previous, (5.2.2.3) reduces to

$$\lambda_k = \frac{1}{2} \cdot \frac{(\vec{f}^{(k)}, \mathbf{W}_k \mathbf{W}_k^T \vec{f}^{(k)})}{(\mathbf{W}_k \mathbf{W}_k^T \vec{f}^{(k)}, \mathbf{W}_k \mathbf{W}_k^T \vec{f}^{(k)})},$$

where $\vec{f}^{(k)} = \vec{f}(\vec{x}^{(k)})$ and $\mathbf{W}_k = \mathbf{W}(\vec{x}^{(k)})$. Finally, gradient method can be represented in the form

$$\vec{x}^{(k+1)} = \vec{x}^{(k)} - 2\lambda_k W_k^T \vec{f}(\vec{x}^{(k)}) \quad (k = 0, 1, \ldots). \tag{5.2.2.5}$$

As we see, in place of matrix $\mathbf{W}^{-1}(\vec{x}^{(k)})$ which appears in Newton-Kantorowich method, we have now matrix $2\lambda_k \mathbf{W}_k^T$.

Example 5.2.2.1. *System of nonlinear equations given in example 5.2.1.1 will be solved using gradient method, starting with the same initial vector $\vec{x}^{(0)} = [2\ 1]^T$, giving the following list of results*

```
i        x1(i)            x2(i)              2lam_k
0 .2000000000D+01 .1000000000D+01  .305787395D-03
1 .1975537008D+01 .9259994504D+00  .538747689D-03
2 .1983210179D+01 .9201871306D+00  .339553623D-03
3 .1983643559D+01 .9207840032D+00  .535596539D-03
4 .1983705230D+01 .9207387845D+00  .339328604D-03
5 .1983708270D+01 .9207429317D+00  .535573354D-03
6 .1983708709D+01 .9207426096D+00  .339332516D-03
7 .1983708731D+01 .9207426391D+00  .535990624D-03
8 .1983708234D+01 .9207426368D+00  .337793301D-03
9 .1983708734D+01 .9207426370D+00
```

Note that the convergence here is much slower than at Newton-Kantorowich method, due to fact that gradient method is of first order.

Gradient method is successfully used in many optimization problems of nonlinear programming, with large number of methods, especially of gradient type, which are basis for number of programming packages for solving nonlinear programming problems (see monograph [6] for symbolic implementation of nonlinear optimization).

The methods of unconstrained optimization, based on derivatives of goal function can be roughly separated in two classes. To the first group belong methods which use only first derivative of goal function and they are called *gradient methods of first order*. The most known gradient method of first order is Cauchy method of steepest descent. This method has linear convergence and characteristic of good progress to optimum point from "distant" initial approximations, but slow convergence in vicinity of optimum point. To the second class belong methods which use first and second derivative of goal function (or some approximation of them) which are called *gradient methods of second order*. The most known method of second order is Newton's method. It is characterized by square convergence, what means that, when converges, is faster than Cauchy's method. But, it is less reliable than Cauchy's method. Best characteristics of both methods belong to so known Quasi-Newton (Variable metric) methods. These methods have at least linear order of convergence, with quadratic asymptotic error.

Some basic terms, necessary for understanding of gradient methods of optimization are given below.

Vector-gradient ∇Q in n-dimensional space has n components, equal to partial derivatives in every governing parameter in point $\vec{x}_k$, i.e.

$$\nabla Q(\vec{x}^{(k)}) = \text{grad } Q(\vec{x}^{(k)}) = \left\{ \frac{\partial Q(\vec{x}^{(k)})}{\partial x_i} \right\}_{i=1,\dots,n} \tag{5.2.2.6}$$

where $Q(\vec{x}) = Q(x_1, x_2, \dots, x_n)$ Due to simplicity, we will denote $Q^{(k)} = Q(\vec{x}^{(k)})$, so that gradient in point $\vec{x}^{(k)}$ will be

$$\nabla Q(\vec{x}^{(k)}) = \text{grad } Q(\vec{x}^{(k)}) = \left\{ \frac{\partial Q^{(k)}}{\partial x_1}, \dots, \frac{\partial Q^{(k)}}{\partial x_n} \right\} \tag{5.2.2.7}$$

Gradient vector $\nabla Q(\vec{x})$ is in every point $\vec{x}^{(k)} = (x_1^{(k)}), \dots, x_n^{(k)})$ normal to the plane with constant value $Q(\vec{x})$ and goes through given point. This vector has in every point $\vec{x}^{(k)}$ orientation of fastest grow of $Q(\vec{x})$ from this point. Algorithm of gradient optimization methods consists in procedure that, starting from given or computed point $\vec{x}^{(k)}$, one goes to the next point $\vec{x}^{(k+1)}$ with step $\Delta\vec{x}^{(k)}$ in direction of gradient, during calculation of maximum

$$\vec{x}^{(k+1)} = \vec{x}^{(k)} + \Delta\vec{x}^{(k)} = \vec{x}^{(k)} + \vec{h}^{(k)} \nabla Q(\vec{x}^{(k)}), \tag{5.2.2.8}$$

or in opposite direction of gradient, when calculating minimum

$$\vec{x}^{(k+1)} = \vec{x}^{(k)} - \vec{h}^{(k)}\nabla Q(\vec{x}^{(k)}). \tag{5.2.2.9}$$

When given parameter of step $\vec{h}^{(k)} = (h_i^{(k)}), i = 1, \ldots, n$, move in direction of gradient is realized by formulas

$$x_i^{(k+1)} = x_i^{(k)} + h_i^{(k)}\frac{\partial Q^{(k)}}{\partial x_i} \quad (i = 1, \ldots, n), \tag{5.2.2.10}$$

during finding a maximum, and

$$x_i^{(k+1)} = x_i^{(k)} - h_i^{(k)}\frac{\partial Q^{(k)}}{\partial x_i} \quad (i = 1, \ldots, n), \tag{5.2.2.11}$$

during finding minimum of function $Q(x)$. In formulas (5.2.2.10) and (5.2.2.11) the move is in direction of gradient only if all $h_i^{(k)}, (i = 1, \ldots, n)$ are same. Nevertheless, in some methods are steps chosen arbitrary, or by some criteria.

At gradient methods the formulas with coordinates of normalized gradient vector can be uses.

$$x_i^{(k+1)} = x_i^{(k)} + h_i^{(k)}\frac{\dfrac{\partial Q^{(k)}}{\partial x_i}}{\sqrt{\sum_{i=1}^{n}(\dfrac{\partial Q^{(k)}}{\partial x_i})^2}}, \quad (i = 1, \ldots, n), \tag{5.2.2.12}$$

In formula (5.2.2.12) normed gradient-vector shows up to the direction of fastest change of goal function, but does not define the velocity of moving through extremum. This is given by steps, $h_k^{(k)}$, $(i = 1, \ldots, n)$. Normalization of gradient improves method stability.

Theoretically, the procedure of gradient search terminates in point in which all coordinates of gradient are equal to zero, i.e. in which is Euclid's norm of gradient equal to zero:

$$||\nabla Q(\vec{x})|| = \sqrt{\sum_{i=1}^{n}(\frac{\partial Q}{\partial x_i})^2} = 0. \tag{5.2.2.13}$$

Thus, the following criterion for termination of gradient search may be used:

$$\sqrt{\sum_{i=1}^{n}(\frac{\partial Q}{\partial x_i})^2} \leq \varepsilon, \tag{5.2.2.14}$$

where ε is given small number. For the same criterion can be used also Chebyshev gradient norm

$$\sum_{i=1}^{n}\left|\frac{\partial Q^{(k)}}{\partial x_i}\right| \leq \varepsilon. \tag{5.2.2.15}$$

The exposed formulas enable writing a code in procedural and symbolic languages for gradient methods, what is suggested to readers.

5.2.3. Globally convergent methods

We have seen that Newtons method and Newton-like methods (quasi-Newton methods) for solving nonlinear equations has an unfortunate tendency not to converge if the initial guess is not sufficiently close to the root. A global method is one that converges to a solution from almost any starting point. Therefore, it is our goal to develop an algorithm that combines the rapid local convergence of Newtons method with a globally convergent strategy that will guarantee some progress towards the solution at each iteration. The algorithm is closely related to the quasi-Newton method of minimization (see [5], p. 376).

From (5.2.1.3), Newton-Raphson method, we have so known Newton step in iteration formula

$$\vec{x}^{(k+1)} - \vec{x}^{(k)} = \delta\vec{x} = -\mathbf{W}^{-1}(\vec{x}^{(k)})\vec{f}(\vec{x}^{(k)}), \qquad (k = 0, 1, \ldots) \tag{5.2.3.1}$$

where $\mathbf{W}$ is Jacobian matrix. The question is how one should decide to accept the Newton step δx? If we denote $\mathbf{F} = \vec{f}(\vec{x}^{(k)})$, a reasonable strategy for step acceptance is that $|\mathbf{F}|^2 = \mathbf{F} \cdot \mathbf{F}$ decreases, what is the same requirement one would impose if trying to minimize

$$f = \frac{1}{2}\mathbf{F} \cdot \mathbf{F}. \tag{5.2.3.2}$$

Every solution of (5.2.1.1) minimizes (5.2.3.2), but there may be some local minima of (5.2.3.2) that are not solution of (5.2.1.1). Thus, simply applying some minimum finding algorithms can be wrong.

To develop a better strategy, note that Newton step (5.2.3.1) is a descent direction for f:

$$\nabla f \cdot \delta\vec{x} = (\mathbf{F} \cdot \mathbf{W}) \cdot (-\mathbf{W}^{-1} \cdot \mathbf{F}) = -\mathbf{F} \cdot \mathbf{F} < 0. \tag{5.2.3.3}$$

Thus, the strategy is quite simple. One should first try the full Newton step, because once we are close enough to the solution, we will get quadratic convergence. However, we should check at each iteration that the proposed step reduces f. If not, we go back (backtrack) along the Newton direction until we get acceptable step. Because the Newton direction is descent direction for f, we fill find for sure an acceptable step by backtracking.

It is to mention that this strategy essentially minimizes f by taking Newton steps determined in such a way that bring ∇f to zero. In spite of fact that this method can occasionally lead to local minimum of f, this is rather rare in practice. In such a case, one should try a new starting point.

Line Searches and Backtracking

When we are not close enough to the minimum of f, taking the full Newton step $\vec{p} = \delta\vec{x}$ need not decrease the function; we may move too far for the quadratic approximation to be valid. All we are guaranteed is that initially f decreases as we move in the Newton direction. So the goal is to move to a new point x_{new} along the direction of the Newton step $\vec{p}$, but not necessarily all the way (see [5], pp. 377-378):

$$\vec{x}_{new} = \vec{x}_{old} + \lambda\vec{p} \quad (0 < \lambda \leq 1) \tag{5.2.3.4}$$

The aim is to find λ so that $f(\vec{x}_{old} + \lambda\vec{p})$ has decreased sufficiently. Until the early 1970s, standard practice was to choose λ so that $\vec{x}_{new}$ exactly minimizes f in the direction $\vec{p}$. However, we now know that it is extremely wasteful of function evaluations to do so. A better strategy is as follows: Since $\vec{p}$ is always the Newton direction in our algorithms, we first try $\lambda = 1$, the full Newton step. This will lead to quadratic convergence when $\vec{x}$ is sufficiently close to the solution. However, if $f(\vec{x}_{new})$ does not meet our acceptance criteria, we backtrack along the Newton direction, trying a smaller value of λ, until we find a suitable point. Since the Newton direction is a descent direction, we are guaranteed to decrease f for sufficiently small λ. What should the criterion for accepting a step be? It is not sufficient to require merely that $f(\vec{x}_{new}) < f(\vec{x}_{old})$. This criterion can fail to converge to a minimum of f in one of two ways. First, it is possible to construct a sequence of steps satisfying this criterion with f decreasing too slowly relative to the step lengths. Second, one can have a sequence where the step lengths are too small relative to the initial rate of decrease of f. A simple way to fix the first problem is to require the average rate of decrease of f to be at least some fraction α of the initial rate of decrease $\nabla f \cdot \vec{p}$

$$f(\vec{x}_{new}) \leq f(\vec{x}_{old}) + \alpha \nabla f \cdot (\vec{x}_{new} - \vec{x}_{old}). \tag{5.2.3.5}$$

Here the parameter α satisfies $0 < \alpha < 1$. We can get away with quite small values of α; $\alpha = 10^{-4}$ is a good choice. The second problem can be fixed by requiring the rate of decrease of f at $\vec{x}_{new}$ to be greater than some fraction β of the rate of decrease of f at $\vec{x}_{old}$. In practice, we will not need to impose this second constraint because our backtracking algorithm will have a built-in cutoff to avoid taking steps that are too small.

Here is the strategy for a practical backtracking routine. Define

$$g(\lambda) \equiv f(\vec{x}_{old} + \lambda \vec{p}) \tag{5.2.3.6}$$

so that

$$g'(\lambda) = \nabla f \cdot \vec{p} \tag{5.2.3.7}$$

If we need to backtrack, then we model g with the most current information we have and choose λ to minimize the model. We start with $g(0)$ and $g'(0)$ available. The first step is always the Newton step, $\lambda = 1$. If this step is not acceptable, we have available $g(1)$ as well. We can therefore model $g(\lambda)$ as a quadratic:

$$g(\lambda) \approx [g(1) - g(0) - g'(0)]\lambda^2 + g(0). \tag{5.2.3.8}$$

By first derivative of this function we find the minimum condition

$$\lambda = -\frac{g'(0)}{2[g(1) - g(0) - g'(0)]}. \tag{5.2.3.9}$$

Since the Newton step failed, we can show that $\lambda \lesssim \frac{1}{2}$ for small α. We need to guard against too small a value of λ, however. We set $\lambda_{min} = 0.1$.

On second and subsequent backtracks, we model g as a cubic in λ, using the previous value $g(\lambda_1)$ and the second most recent value $g(\lambda_2)$.

$$g(\lambda) = a\lambda^3 + b\lambda^2 + g'(0)\lambda + g(0) \tag{5.2.3.10}$$

Requiring this expression to give the correct values of g at λ_1 and λ_2 gives two equations that can be solved for the coefficients a and b.

$$\begin{bmatrix} a \\ b \end{bmatrix} = \frac{1}{\lambda_1 - \lambda_2} \begin{bmatrix} 1/\lambda_1^2 & -1/\lambda_2^2 \\ -\lambda_2/\lambda_1^2 & \lambda_1/\lambda_2^2 \end{bmatrix} \cdot \begin{bmatrix} g(\lambda_1) - g'(0)\lambda_1 - g(0) \\ g(\lambda_2) - g'(0)\lambda_2 - g(0) \end{bmatrix}. \tag{5.2.3.11}$$

The minimum of the cubic (5.2.3.10) is at

$$\lambda = \frac{-b + \sqrt{b^2 - 3ag'(0)}}{3a}. \tag{5.2.3.12}$$

One should enforce that λ lie between $\lambda_{max} = 0.5\lambda_1$ and $\lambda_{min} = 0.1\lambda_1$. The corresponding code in FORTRAN is given in [5], pp. 378-381. It it suggested to reader to write the corresponding code in `Mathematica`.

Multidimensional Secant Methods: Broyden's Method

Newton's method as used previously is rather efficient, but it still has several disadvantages. One of the most important is that it needs Jacobian matrix. In many problems the Jacobian matrix is not available, i.e. there do not exist analytic derivatives. If the function evaluation is complicated, the finite-difference determination of Jacobian can be prohibitive. There are quasi-Newton methods which provide cheap approximation to the Jacobian for the purpose of zero finding. The methods are often called *secant methods*, because they reduce in one dimension to the secant method. One of the best of those methods is Broyden's method (see [21]).

If one denotes approximate Jacobian by $\mathbf{B}$, then the i-th quasi-Newton step $\delta\vec{x}_i$ is the solution of

$$\mathbf{B}_i \cdot \delta\vec{x}_i = -\mathbf{F}_i, \tag{5.2.3.13}$$

where $\delta\vec{x}_i = \vec{x}_{i+1} - \vec{x}_i$. Quasi-Newton or secant condition is that $\mathbf{B}_{i+1}$ satisfy

$$\mathbf{B}_{i+1} \cdot \delta\vec{x}_i = \delta\mathbf{F}_i, \tag{5.2.3.14}$$

where $\delta\mathbf{F}_i = \mathbf{F}_{i+1} - \mathbf{F}_i$. This is generalization of the one-dimensional secant approximation to the derivative, $\delta F/\delta x$. However, equation (5.2.3.14) does not determine $\mathbf{B}_{i+1}$ uniquely in more than one dimension. Many different auxiliary conditions to determine $\mathbf{B}_{i+1}$ have been examined, but the best one results from the Broyden's formula. This formula is based on idea of getting $\mathbf{B}_{i+1}$ by making a least change to $\mathbf{B}_i$ in accordance to the secant equation (5.2.3.14). Broyden gave the formula

$$\mathbf{B}_{i+1} = \mathbf{B}_i + \frac{(\delta\mathbf{F}_i - \mathbf{B}_i \cdot \delta\vec{x}_i) \otimes \delta\vec{x}_i}{\delta\vec{x}_i \cdot \delta\vec{x}_i}. \tag{5.2.3.15}$$

One can check that $\mathbf{B}_{i+1}$ satisfies (5.2.3.14).

Early implementations of Broyden's method used the Sherman-Morrison formula to invert equation analytically,

$$\mathbf{B}_{i+1}^{-1} = \mathbf{B}_i^{(-1)} + \frac{(\delta\vec{x}_i - \mathbf{B}_i^{-1} \cdot \delta\mathbf{F}_i) \otimes \delta\vec{x}_i \cdot \mathbf{B}_i^{-1}}{\delta\vec{x}_i \cdot \delta\mathbf{B}_i^{-1} \cdot \delta\mathbf{F}_i}. \tag{5.2.3.16}$$

Thus, instead of solving equation (5.2.3.1) by, for example, **LU** decomposition, one determined

$$\delta\vec{x}_i = -\mathbf{B}_i^{-1} \cdot \mathbf{F}_i \tag{5.2.3.17}$$

by matrix multiplication in $O(n^2)$ operations. The disadvantage of this method is that it cannot be easily embedded in a globally convergent strategy, for which the gradient of equation (5.2.3.2) requires $\mathbf{B}$, not $\mathbf{B}^{-1}$

$$\nabla(\frac{1}{2}\mathbf{F} \cdot \mathbf{F}) \simeq \mathbf{B}^T \cdot \mathbf{F} \tag{5.2.3.18}$$

Accordingly, one should implement the update formula in the form (5.2.3.15). However, we can still preserve the $\mathbf{O}(n^2)$ solution of (5.2.3.1) by using **QR** decomposition of $\mathbf{B}_{i+1}$ in $\mathbf{O}(n^2)$ operations. All needed is initial approximation $\mathbf{B}_0$ to start process. It is often accepted to take identity matrix, and then allow $\mathbf{O}(n)$ updates to produce a reasonable approximation to the Jacobian. In [5], p. 382-383, the first n function evaluations are spent on a finite-difference approximation in order to initialize **B**. Since **B** is not exact Jacobian, it is not guaranteed that $\delta\vec{x}$ is descent direction for $f = \frac{1}{2}\mathbf{F}\cdot\mathbf{F}$ (see eq. (5.2.3.3)). That has a consequence that the line search algorithm can fail to return the suitable step if **B** is far from the true Jacobian. In this case we simply reinitialize **B**.

Like the secant method in one dimension, Broyden's method converges superlinearly once you get close enough to the root. Embedded in a global strategy, it is almost as robust as Newton's method, and often needs far fewer function evaluations to determine a zero. Note that the final value of **B** is not always close to the true Jacobian at the root, in spite of fact that method converges.

The programme code ([5], pp. 383-385) of Broyden's method differs from Newtonian methods in using **QR** decomposition instead of **LU**, and determination of Jacobian by finite-difference approximation instead of direct evaluation.

More Advanced Implementations

One of the principal ways that the methods described above can fail is if matrix **W** (Newton-Kantorowich) or **B** (Broyden's method) becomes singular or nearly singular, so that Δx cannot be determined. This situation will not occur very often in practice. Methods developed so far to deal with this problem involve the monitoring of condition number of **W** and perturbing **W** if singularity or near singularity is detected. This feature is most easily implemented if **QR** decomposition instead of **LU** decomposition in Newton (or quasi-Newton) method is applied. However, in spite of fact that this method can solve problems when **W** is exactly singular and Newton's and Newton-like methods fail, it is occasionally less robust on other problems where **LU** decomposition succeeds. Implementation details, like roundoff, underflow, etc. are to be considered and taken in account.

In [5], considering effectiveness of strategies for minimization and zero finding, the global strategies have been based on *line searches*. Other global algorithms, like *hook step* and *dogleg step* methods, are based on the *model-trust region approach*, which is related to the Levenberg-Marquardt algorithm for nonlinear least-squares. In spite being more complicated than line searches, these methods have a reputation for robustness even when starting far from desired zero or minimum.

Numerous libraries and software packages are available for solving nonlinear equations. Many workstations and mainframe computers have such libraries attached to operating systems. Many commercial software packages contain nonlinear equation solvers. Very popular among engineers are `Matlab` and `Matcad`. More sophisticated packages like `Mathematica`, `IMSL`, `Macsyma`, and `Maple` contain programs for nonlinear equation solving. The book *Numerical recipes* [5] contains numerous programs for solving nonlinear equation.

For symbolic implementation of nonlinear optimization, see [6], containing not only very useful code but also corresponding theoretical background.

Bibliography (Cited references and further reading)

[1] Milovanović, G.V., *Numerical Analysis I*. Naučna knjiga, Beograd, 1988 (Serbian).

[2] Hoffman, J.D., *Numerical Methods for Engineers and Scientists.* Taylor & Francis, Boca Raton-London-New York-Singapore, 2001.

[3] Milovanović, G.V. and Djordjević, Dj.R., *Programiranje numeričkih metoda na FORTRAN jeziku.* Institut za dokumentaciju zaštite na radu "Edvard Kardelj", Niš, 1981 (Serbian).

[4] Stoer, J., and Bulirsch, R., *Introduction to Numerical Analysis*, Springer, New York, 1980.

[5] Press, W.H., Flannery, B.P., Teukolsky, S.A., and Vetterling, W.T., *Numerical Recepies - The Art of Scientific Computing.* Cambridge University Press, 1989.

[6] Milovanović, G.V., Stanimirović, P.S., *Symbolic Implementation of Nonlinear Optimization.* University of Niš, Faculty of Electronic Engineering, Niš, 2002 (Serbian).

[7] Djordjević, L.N., *An iterative solution of algebraic equations with a parameter to accelerate convergence.* Univ. Beograd. Publ. Elektrotehn. Fak. Ser. Mat.Fiz. No. 412- No. 460(1973), 179-182.

[8] Tihonov, O.N., *O bystrom vychyslnii najbolshih kornei mnogočlena.* Zap. Leningrad. gorn. in-ta 48, 3(1968), 36-41.

[9] Ostrowski, A., *Solution of Equations and System of Equations.* Academic Press, New Yurk, 1966.

[10] Wegstein, J.H., *Accelerating convergence of iterative processes.* Comm. ACM 1 (1958), 9-13.

[11] Ralston, A., *A First Course in Numerical Analysis.* McGraw-Hill, New York, 1965.

[12] Milovanović, G.V. & Petković, M.S., *On some modifications of a third order method for solving equations.* Univ. Beograd. Publ. Elektroteh. Fak. Ser. Mat. Fiz. No. 678 - No. 715 (1980), pp. 63-67.

[13] Milovanović, Kovačević, M.A., *Two iterative processes of third order without derivatives.* IV Znanstveni skup PPPR, Stubičke Toplice, 1982, Proceedings, pp. 63-67 (Serbian).

[14] Varjuhin, V.A. and Kasjanjuk, S.A., *Ob iteracionnyh metodah utočnenija kornei uravnenii.* Ž. Vychysl. Mat. i Mat. Fiz. 9(1969), 684-687.

[15] Lika, D.K., *Ob iteracionnyh metodah vissego porjadka.* Dokl. 2-i Nauč.-tehn. respubl. konf. Moldavii. Kishinev, 1965. pp.13-16.

[16] Djordjević, L.N. & Milovanović, G.V., *A combined iterative formula for solving equations.* Informatika 78, Bled 1978, 3(207).

[17] Petković, M.S., *Some iterative interval methods for solving equations.* Ph.D. thesis, University Niš, 1980.

[18] Petković, M.S. & Petković, D. Lj., *On a method for two-sided approaching for solving equations.* Freiburger Intervall-Berichte 10(1980), pp. 1-10.

[19] *IMSL Math/Library Users Manual* , IMSL Inc., 2500 City West Boulevard, Houston TX 77042

[20] *NAG Fortran Library*, Numerical Algorithms Group, 256 Banbury Road, Oxford OX27DE, U.K., Chapter F02.

[21] Broyden, C.G., *Quasi-Newton methods and their application to function minimization*, Math. Comp. 29(1967), 368-381.

[22] Kantorowich, L.V., *Funkcional'nyi analiz i prikladnaja matematika.*, Uspehi Mat. Nauk 3(1948), 89-185.

[23] Ortega, J.M. & Rheinboldt, W.C., *Iterative solution of nonlinear equations in several variables*, Academic Press, New York, 1970.

[24] Rall, L., *Computational solution of nonlinear operator equations*, New York, 1969.

[25] Kul'čickii, O.Ju. & Šimelevič, L.I., *O nahoždenii načal'nogo pribiženija.*, Ž. Vyčisl. Mat. i Mat. Fiz. 14(1974), pp. 1016-1018.

[26] Dennis, J.E. and Schnabel, R.B., *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Englewood Cliffs, NJ: Prentice Hall, 1983.

LECTURES

LESSON VI

6. Approximation and Interpolation

6.1. Introduction

This lesson is devoted to one of the most important areas of theory of approximation - interpolation of functions. In addition to theoretical importance in construction of numerical methods for solving a lot of problems like numerical differentiation, numerical integration and similar, it has practical application to many engineering problems, including FEM problems.

Theory of approximation deals with replacing of function f defined on some set X by another function Φ. Let function Φ depend on $n+1$ parameter $a_0, a_1, \ldots, a_n$, i.e.

$$\Phi(x) = \Phi(x; a_0, a_1, \ldots, a_n).$$

Problem of approximation of function f by function Φ reduces to determination of parameters a_i, $i = 1, \ldots, n$ according to some criterion. Depending on chosen criterion, we differ several sorts of approximations. Generally, depending on form of approximation function, they can be divided to linear and nonlinear approximation functions. The general form of linear approximation function is

(6.1.1) $$\Phi(x) = a_0\Phi_0(x) + a_1\Phi_1(x) + \ldots + a_n\Phi_n(x),$$

whereby system of functions $\{\Phi_k\}$ fulfills some given conditions. Linearity of function Φ means linearity regarding parameters a_i $(i = 0, 1, \ldots, n)$. When $\Phi_k = x^k$ $(k = 0, 1, \ldots, n)$, i.e.

$$\Phi(x) = a_0 + a_1 x + \ldots + a_n x^n,$$

we have approximation by algebraic polynomials. In the case when $\{\Phi_k\} = \{1, \cos x, \sin x, \cos 2x, \sin 2x, \ldots\}$ we have approximation by trigonometric polynomials, i.e. trigonometric approximation. For the case

$$\Phi_k(x) = (x - x_k)_+^m = \begin{cases} (x - x_k)^m & (x \geq x_k), \\ 0 & (x < x_k), \end{cases}$$

where m is fixed natural number, we have spline approximation.

We will mention two of nonlinear approximations:

1. Exponential approximation

$$\Phi(x) = \Phi(x; c_0, b_0, \ldots, c_r, b_r) = c_0 e^{b_0 x} + \ldots + c_r e^{b_r x},$$

where $n + 1 = 2(r + 1)$, i.e. $n = 2r + 1$.

2. Rational approximation

$$\Phi(x) = \Phi(x; b_0, \ldots, b_r, c_0, \ldots, c_s) = \frac{b_0 + b_1 x + \ldots + b_r x^r}{c_0 + c_1 x + \ldots + c_s x^s},$$

where $n = r + s + 1$.

Let function f be given on segment $[a, b]$ by set of pairs (x_k, f_k) $(k = 0, 1, \ldots, n)$, where $f_k \equiv f(x_k)$. If for approximation of function f by function Φ the criterion for choice of parameters $a_0, a_1, \ldots, a_n$ is given by system of equations

$$\Phi(x_k; a_0, a_1, \ldots, a_n) = f_k \quad (k = 0, 1, \ldots, n), \tag{6.1.2}$$

we have problem of function interpolation. Function Φ is called in this case interpolation function and points x_k $(k = 0, 1, \ldots, n)$ interpolation nodes.

Problem of interpolation could be more complicated then noted. More general case appears when, in addition to function values in interpolation nodes, the derivatives of function are also included.

6.2. Chebyshev systems

Let function f be given by its values $f_k \equiv f(x_k)$ in points x_k $(x_k \in [a, b])(k = 0, 1, \ldots, n)$. If we have linear interpolation problem, i.e. interpolation by function (6.1.1), system of equation (6.1.2) reduces to system of linear equations in parameters a_i $(i = 0, 1, \ldots, n)$,

$$a_0 \Phi_0(x_k) + a_1 \Phi_1(x_k) + \ldots + a_n \Phi_n(x_k) = f_k \quad (k = 0, 1, \ldots, n),$$

i.e.

$$\begin{bmatrix} \Phi_0(x_0) & \Phi_1(x_0) & \ldots & \Phi_n(x_0) \\ \Phi_0(x_1) & \Phi_1(x_1) & \ldots & \Phi_n(x_1) \\ \vdots & & & \\ \Phi_0(x_n) & \Phi_1(x_n) & \ldots & \Phi_n(x_n) \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_n \end{bmatrix}. \tag{6.2.1}$$

In order above given interpolation problem to have unique solution it is necessary that matrix of system (6.2.1) be regular.

To the system of functions (Φ_k) should be intruded such conditions under which there not exists linear combination

$$a_0 \Phi_0(x) + a_1 \Phi_1(x) + \ldots + a_n \Phi_n(x)$$

which has $n + 1$ different zeros on $[a, b]$. System of functions with such characteristic are called Chebyshev (Tchebyshev) systems, or T-systems. There exists extraordinary monograph regarding T-systems [6].

Theorem 6.2.1. *If the functions $\Phi_k : [a, b] \to R$ $(k = 0, 1, \ldots, n)$ are $n + 1$ times differentiable and if for every $k = 0, 1, \ldots, n$ the Wronsky determinant W_k is different from zero, i.e.*

$$W_k = \begin{vmatrix} \Phi_0(x) & \Phi_1(x) & \ldots & \Phi_k(x) \\ \Phi_0'(x) & \Phi_1'(x) & \ldots & \Phi_k'(x) \\ \vdots & & & \\ \Phi_0^{(k)}(x) & \Phi_1^{(k)}(x) & \ldots & \Phi_k^{(k)}(x) \end{vmatrix} \neq 0,$$

system of functions $\{\Phi_k\}$ is Chebyshev (T) system.

6.3. Lagrange interpolation

Let function f be given by its values $f_k \equiv f(x_k)$ in points x_k $(x_k \in [a,b])(k = 0,1,\ldots,n)$. Without decreasing the generality, assume

$$a \le x_0 < x_1 < \ldots < x_n \le b. \tag{6.3.1}$$

If we take points x_k for interpolation knots and put $\Phi_k(x) = x^k$ $(k = 0,1,\ldots,n)$ we have a problem of interpolation of function f by algebraic polynomial. Denote this polynomial with P_n, i.e.

$$P_n(x) = \Phi(x) = a_0 + a_1x + \ldots + a_nx^n.$$

Then we have the interpolating polynomial

$$P_n(x) = \sum_{k=0}^{n} f(x_k)L_k(x), \tag{6.3.2}$$

where

$$\begin{aligned} L_k(x) &= \frac{(x-x_0)\ldots(x-x_{k-1})(x-x_{k+1})\ldots(x-x_n)}{(x_k-x_0)\ldots(x_k-x_{k-1})(x_k-x_{k+1})\ldots(x_k-x_n)} \\ &= \frac{\omega(x)}{(x-x_k)\omega'(x_k)}, \\ \omega(x) &= (x-x_0)(x-x_1)\ldots(x-x_n), \\ \omega'(x_k) &= (x_k-x_0)\ldots(x_k-x_{k-1})(x_k-x_{k+1})\ldots(x_k-x_n). \end{aligned}$$

The formula (6.3.2) is called Lagrange interpolation formula, and polynomial P_n Lagrange interpolation polynomial. When programming, it is suggested to use the following form of Lagrange formula.

$$P_n(x) = \sum_{k=0}^{n}\left(f(x_k)\prod_{\substack{i=0\\ i\ne k}}^{n}\frac{x-x_i}{x_k-x_i}\right).$$

Having in mind that determinant of system (6.2.1) is Vandermonde determinant, i.e.

$$\begin{vmatrix} 1 & x_0 & \ldots & x_0^n \\ 1 & x_1 & \ldots & x_1^n \\ \vdots & & & \\ 1 & x_n & \ldots & x_n^n \end{vmatrix} = \prod_{i>j}(x_i - x_j),$$

and the assumption (6.3.1), it follows that the Lagrange polynomial (6.3.2) is unique.

Example 6.3.1. *For function values given in tabular form find the Lagrange interpolation polynomial.*

x_k	$f(x_k)$
-1	$-1.$
0	$2.$
2	$10.$
3	$35.$

$$P_3(x) = (-1)\frac{(x-0)(x-2)(x-3)}{(-1-0)(-1-2)(-1-3)} + 2\frac{(x+1)(x-2)(x-3)}{(0+1)(0-2)(0-3)}$$
$$+ 10\frac{(x+1)(x-0)(x-3)}{(2+1)(2-0)(2-3)} + 35\frac{(x+1)(x-0)(x-2)}{(3+1)(3-0)(3-2)},$$

i.e.

$$P_3(x) = \frac{5}{3}x^3 - \frac{4}{3}x^2 + 2.$$

Example 6.3.2. *Determine approximately zero of function given in example* 6.3.1.

Lagrange interpolation polynomial for function $y \to f^{-1}(y)$ is

$$P_3(y) = (-1)\frac{(y-2)(y-10)(y-35)}{(-1-2)(-1-10)(-1-35)} + 0\frac{(y+1)(y-10)(y-35)}{(2+1)(2-10)(2-35)}$$
$$+ 2\frac{(y+1)(y-2)(y-35)}{(10+1)(10-2)(10-35)} + 3\frac{(y+1)(y-2)(y-10)}{(35+1)(35-2)(35-10)},$$

wherefrom, for $y = 0$ we get zero of a function f

$$x \cong P_3(0) = -0.6508$$

6.4. Newton interpolation with divided differences

For function f given by its values $f_k \equiv f(x_k)$ in points x_k $(k = 0, 1, \ldots, n)$, define first divided differences. The ratio

$$\frac{f(x_1) - f(x_0)}{x_1 - x_0}$$

is called divided difference of first order (of function f in points x_0 and x_1) and denoted as $[x_0, x_1; f]$.

Divided difference of order r are defined recursively by

$$[x_0, x_1, \ldots, x_r; f] = \frac{[x_1, \ldots, x_r; f] - [x_0, \ldots, x_{r-1}; f]}{x_r - x_0}, \tag{6.4.1}$$

where $[x; f] \equiv f(x)$.

Relation (6.4.1) enables construction of table of divided differences

k	x_k	f_k	Δf_k	$\Delta^2 f_k$	$\Delta^3 f_k$
0	x_0	f_0			
			$[x_0, x_1; f]$		
1	x_1	f_1		$[x_0, x_1, x_2; f]$	
			$[x_1, x_2; f]$		$[x_0, x_1, x_2, x_3; f]$
2	x_2	f_2		$[x_1, x_2, x_3; f]$	
			$[x_2, x_3; f]$		
3	x_3	f_3			
$\vdots$	$\vdots$	$\vdots$			

One can show that divided difference of order r has characteristic of linearity, i.e.

$$[x_0, x_1, \ldots, x_r; c_1 f + c_2 g] = c_1[x_0, \ldots, x_r; f] + c_2[x_0, \ldots, x_r; g].$$

where c_1 and c_2 are arbitrary constants. Because of, based on (6.4.1),

$$[x_0, x_1; f] = \frac{f(x_0)}{x_0 - x_1} + \frac{f(x_1)}{x_1 - x_0},$$

one can prove by mathematical induction

$$[x_0, x_1, \ldots, x_r; f] = \sum_{i=0}^{r} \frac{f(x_i)}{\omega'(x_i)},$$

where $\omega(x) = (x - x_0)(x - x_1) \ldots (x - x_r)$.

Let $f \in C^n[a, b]$ and the condition (6.3.1) holds. Then, for every $r \leq n$, the formula

$$[x_0, x_1, \ldots, x_r; f] = \int_0^1 \int_0^{t_1} \ldots \int_0^{t_{r-1}} f^{(r)}(x_0 + \sum_{i=1}^{r} (x - x_{i-1}) t_i) dt_1\, dt_2 \ldots dt_r$$

holds. This can be proved by mathematical induction.

Applying theorem on integral mean value, from last expression follows

$$\begin{aligned}[x_0, x_1, \ldots, x_r; f] &= f^{(r)}(\xi) \int_0^1 \int_0^{t_1} \ldots \int_0^{t_{r-1}} dt_1\, dt_2 \ldots dt_r \\ &= \frac{1}{r!} f^{(r)}(\xi) \quad (a < \xi < b).\end{aligned}$$

Taking $x_i \to x_0 \quad (i = 1, \ldots, r)$ in last equality, we get

$$(6.4.2) \qquad [x_0, x_1, \ldots, x_r; f] \to \frac{1}{r!} f^{(r)}(x_0).$$

Let us express now value of function $f(x_r)$ $(r \leq n)$ by means of divided differences $[x_0, \ldots, x_i; f]$ $(i = 0, 1, \ldots, r)$.

For $r = 1$, based on definition (6.4.1), we have

$$f(x_1) = f(x_0) + (x_1 - x_0)[x_0, x_1; f].$$

In similar way, for $r = 2$,

$$\begin{aligned} f(x_2) &= f(x_1) + (x_2 - x_1)[x_1, x_2; f] \\ &= (f(x_0) + (x_1 - x_0)[x_0, x_1; f]) + (x_2 - x_1)([x_0, x_1; f] \\ &\quad + (x_2 - x_0)[x_0, x_1, x_2; f]),\end{aligned}$$

i.e.

$$f(x_2) = f(x_0) + (x_2 - x_0)[x_0, x_1; f] + (x_2 - x_0)(x_2 - x_1)[x_0, x_1, x_2; f].$$

In general case, it holds

$$\begin{aligned} f(x_r) = f(x_0) &+ (x_r - x_0)[x_0, x_1; f] + (x_r - x_0)(x_r - x_1)[x_0, x_1, x_2; f] \\ &+ \ldots + (x_r - x_0)(x_r - x_1) \ldots (x_r - x_{r-1})[x_0, x_1, \ldots, x_r; f].\end{aligned}$$

Using divided differences for set of data $(x_k, f(x_k))$ $(k = 0, \ldots, n)$ the interpolation polynomial of the following form can be constructed.

$$\begin{aligned} P_n(x) = f(x_0) + (x - x_0)[x_0, x_1; f] + (x - x_0)(x - x_1)[x_0, x_1, x_2; f] \\ + \ldots + (x - x_0)(x - x_1)\ldots(x - x_{n-1})[x_0, x_1, \ldots, x_n; f]. \end{aligned}$$

This polynomial is called Newton's interpolation polynomial.

Having in mind uniqueness of algebraic interpolation polynomial, we conclude that Newton's interpolation polynomial is equivalent to Lagrange polynomial. Note that construction of Newton's interpolation polynomial demands previous forming of table of divided differences, what was not the case with Lagrange polynomial. On the other hand, involving a new interpolation node in order to reduce interpolation error, is more convenient with Newton's polynomial, because do not demand repeating of whole calculation. Namely, at Newton's interpolation we have

$$P_{n+1}(x) = P_n(x) + (x - x_0)(x - x_1)\ldots(x - x_n)[x_0, x_1, \ldots, x_{n+1}; f].$$

If we put $x_i \to x_0$ in Newton's interpolation polynomial P_n, based on (6.4.2) it reduces to Taylor polynomial.

Example 6.4.1. *Based on table of values of function $x \to chx$ form table of divided differences and write Newton's interpolation polynomial.*

k	0	1	2	3
x_k $f(x_k)$	0.0 1.0000000	0.2 1.0200668	0.5 1.1276260	1.0 1.5430806

k	$[x_k, x_{k+1}; f]$	$[x_k, x_{k+1}, x_{k+2}; f]$	$[x_k, x_{k+1}, x_{k+2}, x_{k+3}; f]$
0			
	<u>0.1003338</u>		
1		<u>0.5163938</u>	
	0.3585307		<u>0.0740795</u>
2		0.5904733	
	0.8309093		
3			

Newton's interpolation polynomial is then

$$P_3(x) = 1. + 0.1003338x + 0.5163938x(x - 0.2) + 0.0740795x(x - 0.2)(x - 0.5).$$

For example, for $x = 0.3$, $ch\ 0.3 \cong P_3(0.3) = 1.0451474$.

6.5. Newton's interpolation formulas

By means of finite difference calculus the several interpolation formulas in equidistant nodes can be evaluated. The oldest one is Newton's interpolation formula.

Let function f be given on $[a, b]$ by pairs of values x_k, f_k, where $f_k = f(x_k)$ and $x_k = x_0 + kh$ $(k = 0, 1, \ldots, n)$. For given set of data the table of finite differences can be formed. In the following table is used operator Δ defined by $\Delta f(x) = f(x + h) -$

$f(x)$ $(h = const. > 0)$.

$$\begin{array}{cccccc}
x_0 & \underline{f_0} & & & & \\
 & & \underline{\Delta f_0} & & & \\
x_1 & f_1 & & \underline{\Delta^2 f_0} & & \\
 & & \Delta f_1 & & \underline{\Delta^3 f_0} & \\
x_2 & f_2 & & \Delta^2 f_1 & & \underline{\Delta^4 f_0} \\
 & & \Delta f_2 & & \Delta^3 f_1 & \\
x_3 & f_3 & & \Delta^2 f_2 & & \\
 & & \Delta f_3 & & & \\
x_4 & f_4 & & & & \\
\vdots & \vdots & & & &
\end{array}$$

Let $x = x_0 + ph$ $(0 \le p \le n)$, i.e. $p = \dfrac{x - x_0}{h}$. Because of

$$E^p = (1+\Delta)^p = \sum_{k=0}^{n} \binom{p}{k} \Delta^k,$$

where E is shift operator $(Ef(x) = f(x+h))$, we have

$$E^p f_0 = \sum_{k=0}^{\infty} \binom{p}{k} \Delta^k f_0 = \sum_{k=0}^{n} \binom{p}{k} \Delta^k f_0 + R_n(f;x),$$

i.e.

(6.5.1) $$f(x_0 + ph) = \sum_{k=0}^{n} \binom{p}{k} \Delta^k f_0 + R_n(f;x).$$

The remainder term R_n, having in mind the uniqueness of interpolation polynomial, is equal to remainder of Lagrange interpolation formula

$$R_n(f;x) = \frac{h^{n+1}}{(n+1)!} p(p-1)\ldots(p-n) f^{(n+1)}(\xi),$$

where ξ is point in interval (x_0, x_n).

The polynomial

(6.5.2) $$P_n(x) = \sum_{k=0}^{n} \binom{p}{k} \Delta^k f_0 \quad (ph = x - x_0)$$

obtained in a given way, is called first Newton's interpolation polynomial. This polynomial can be defined recursively as

$$P_k(x) = P_{k-1}(x) + \binom{p}{k} \Delta^k f_0 \quad (k = 1, \ldots, n),$$

starting with $P_0(x) = f_0$. The developed form of polynomial is

$$P_n(x) = f_0 + p\Delta f_0 + \frac{p(p-1)}{2!}\Delta^2 f_0 + \ldots + \frac{p(p-1)\ldots(p-n+1)}{n!}\Delta^n f_0,$$

i.e

$$P_n(x) = f_0 + \frac{\Delta f_0}{h}(x - x_0) + \frac{\Delta^2 f_0}{2!h^2}(x - x_0)(x - x_1) + \ldots$$
$$+ \frac{\Delta^n f_0}{n!h^n}(x - x_0)(x - x_1)\ldots(x - x_{n-1}).$$

First Newton's interpolation polynomial is used for interpolation on begin of unterval, i.e. in neighborhood of interval starting point x_0. Interpolation of function for $x < x_0$ is called extrapolation.

Remark. *The reader is encouraged to write a code in* `Mathematica` *for program realization of first Newton's interpolation formula, using all four previously given forms.*

For bigger tables of finite differences, usually by applying finite element method, it is very important knowing propagation of accidental error of function value in some interpolation nodes. Let us have the value $f_k + \epsilon$ in place of f_k, where ϵ is error. Then we have the following table obtained using operator Δ.

x_i	f_i	Δf_i	$\Delta^2 f_i$	$\Delta^3 f_i$	$\Delta^4 f_i$
x_{k-4}	f_{k-4}				
		Δf_{k-4}			
x_{k-3}	f_{k-3}		$\Delta^2 f_{k-4}$		
		Δf_{k-3}		$\Delta^3 f_{k-4}$	
x_{k-2}	f_{k-2}		$\Delta^2 f_{k-3}$		$\underline{\Delta^4 f_{k-4} + \varepsilon}$
		Δf_{k-2}		$\underline{\Delta^3 f_{k-3} + \varepsilon}$	
x_{k-1}	f_{k-1}		$\underline{\Delta^2 f_{k-2} + \varepsilon}$		$\underline{\Delta^4 f_{k-3} - 4\varepsilon}$
		$\underline{\Delta f_{k-1} + \varepsilon}$		$\underline{\Delta^3 f_{k-2} - 3\varepsilon}$	
x_k	$\underline{f_k + \varepsilon}$		$\underline{\Delta^2 f_{k-1} - 2\varepsilon}$		$\underline{\Delta^4 f_{k-2} + 6\varepsilon}$
		$\underline{\Delta f_k - \varepsilon}$		$\underline{\Delta^3 f_{k-1} + 3\varepsilon}$	
x_{k+1}	f_{k+1}		$\underline{\Delta^2 f_k + \varepsilon}$		$\underline{\Delta^4 f_{k-1} - 4\varepsilon}$
		Δf_{k+1}		$\underline{\Delta^3 f_k - \varepsilon}$	
x_{k+2}	f_{k+2}		$\Delta^2 f_{k+1}$		$\underline{\Delta^4 f_k + \varepsilon}$
		Δf_{k+2}		$\Delta^3 f_{k+1}$	
x_{k+3}	f_{k+3}		$\Delta^2 f_{k+2}$		
		Δf_{k+3}			
x_{k+4}	f_{k+4}				

The erroneous differences in table are underlined. We see the progressive propagation of error, so that error in difference $\Delta^m f_{k-m+i}$ $(i = 0, 1, \ldots, m)$ is $\binom{m}{i}(-1)^i\varepsilon$.

6.6. Spline functions and interpolation by splines

Physical device named spline consists of a long strip fixed in position at a number of points that relaxes to form a smooth curve passing through those points. Before computers were used for creating engineering designs, drafting tools were employed by designers drawing by hand. To draw curves, especially for shipbuilding, draftsmen often used long, thin, flexible strips of wood, plastic, or metal called a spline (or a lath, not to be confused with lathe). The splines were held in place with lead weights (called ducks because of their duck like shape). The elasticity of the spline material combined with the constraint of the control points, or knots, would cause the strip to take the shape which minimizes the energy required for bending it between the fixed points, and thus adopt the smoothest possible shape. One can recreate a draftsman's spline device with weights and a length of thin stiff plastic or rubber tubing. The weights are attached to the tube (by gluing or pinning). The tubing is then placed over drawing paper. Crosses are marked on the paper to designate the knots or control points. The tube is then adjusted so that it passes over the control points. Supposing uniform elasticity of spline, one can say that its potential energy, when bent, is proportional to the integral along it (curvilinear integral along curve) of quadrate of convolution K. Thus, if spline lies along plane curve $y = S(x)$, $a \le x \le b$, its potential energy is proportional to the integral

$$\int_L K(x)^2 ds = \int_a^b \frac{S''(x)^2}{(1 + S'(x)^2)^{5/2}} dx \tag{6.6.1}$$

and stabilized shape it takes is such that minimizes (6.6.1) under given limitations.

In the similar way is defined mathematical spline, by discarding $S'(x)^2$ in nominator of (6.6.1), what is close to previous case, when $S'(x) << 1$. Thus, now is to minimize the integral

$$\int_a^b S''(x)^2 dx. \tag{6.6.2}$$

Mathematical spline can be more generally defined by using higher derivative then two in (6.6.2).

First results regarding spline functions appeared in papers of Quade and Collatz ([11], 1938) and Courant([12]. In 1946 mathematicians started studying the spline shape, and derived the piecewise polynomial formula known as the spline curve or function (Schoenberg [13]). This has led to the widespread use of such functions in computer-aided design, especially in the surface designs of vehicles. Schoenberg gave the spline function its name after its resemblance to the mechanical spline used by draftsmen. The origins of the spline in wood-working may show in the conjectured etymology which connects the word spline to the word splinter. Later craftsmen have made splines out of rubber, steel, and other elastomeric materials. Spline devices help bend the wood for pianos, violins, violas, etc. The Wright brothers used one to shape the wings of their aircraft.

The extensive development of spline functions and usage of their approximation properties begun in sixties last century. The splines are greatly applied to numerical mathematics, in particular to interpolation, numerical differentiation, numerical integration, differential equations, etc. The extremal and approximative attributes of so known natural cubic spline are given in [1], pp. 81-86.

Let on segment $[a, b]$ given network of nodes

$$\Delta_n : a = x_0 < x_1 < \ldots < x_n = b. \tag{6.6.3}$$

Denote with $\mathcal{P}_m$ set of algebraic polynomials of order not greater than m.

Definition 6.6.1. *Function*

$$S_m(x) = S_{m,k}(x, \Delta_n)$$

is called polynomial spline of degree m and defect k $(1 \le k \le m)$ with nodes (6.6.3), *if satisfies the conditions*

1^0 $S_m \in \mathcal{P}_m$ on every subsegment $[x_{i-1}, x_i]$ $(i = 1, \ldots, n)$,
2^0 $S_m \in C^{m-k}[a, b]$.

Points x_i are called nodes of spline.

We will further consider polynomial splines of defect 1 and for $S_m(x) = S_{m,1}(x)$ say to be a spline of degree m. Very important kind of splines, interpolation cubic spline, with $m = 3$ are most frequently used and applied in engineering design. Therefore we join to the network nodes Δ_n real numbers $f_0, f_1, \ldots, f_n$.

Definition 6.6.2. *Function $S_3(x) = S_3(x; f)$ is called interpolation cubic spline for function f on the network Δ_n $(n \ge 2)$ if the following conditions are fulfilled:*

1^0 $S_3(x, f) \in \mathcal{P}_3$ if $x \in [x_{i-1}, x_i]$ $(i = 1, \ldots, n)$,
2^0 $S_3(x; f) \in C^2[a, b]$,
3^0 $S_3(x_i; f) = f_i = f(x_i)$ $(i = 0, \ldots, n)$.

We see that condition 3^0 does not appear in Definition 6.6.1. The spline defined in this way is called simple cubic spline. It interpolates function f in network nodes (condition 3^0), it is continuous on $[a, b]$ together with its derivatives $S_3'(x)$ and $S_3''(x)$ (condition 2^0) and defined on every subsegment between neighbor nodes with polynomial of degree not greater than 3. So, the third derivative of cubic spline is discontinuous, being, part by part of constant value.

Cubic spline has two free parameters, determined usually by some additional boundary conditions. The typical ones are:

$$S_3'(a) = S_3'(b), \quad S_3''(a) = S_3''(b); \tag{6.6.4}$$

$$S_3'(a) = a_n, \quad S_3'(b) = b_n; \tag{6.6.5}$$

$$S_3''(a) = A_n, \quad S_3''(b) = B_n; \tag{6.6.6}$$

$$S_3'''(x_1 - 0) = S_3'''(x_1 + 0), \quad S_3'''(x_{n-1} - 0) = S_3'''(x_{n-1} + 0), \tag{6.6.7}$$

where a_n, b_n, A_n, B_n are given real numbers.

Conditions (6.6.4) define so known periodic spline. These conditions are used when, for example, interpolating function f is periodic with period $b - a$.

If function f is differentiable and we know values of derivatives in boundary points a and b, then the additional conditions (6.6.5), $a_n = f'(a)$ and $b_n = f'(b)$, or (6.6.6), $A_n = f''(a)$ and $B_n = f''(b)$, are to be used, what is often case at mechanical models. The corresponding spline is called natural cubic spline.

The conditions (6.6.7) are most complicated, but they obtain continuity of third derivatives of spline in points $x = x_1$ and $x = x_{n-1}$.

The most interesting spline approximation is cubic spline interpolation. The algorithm for construction of cubic spline is given in ([1], pp. 73-81). To interested reader is suggested to write a code for construction of spline (taking care of Hermite interpolation) and, if possible, include graphical interface. For some programming details, see Fortran subroutines `Spline` and `Splint` in ([5], pp. 109-110). For obtaining a higher order of smoothness in two-dimensional interpolation (applicable in many areas of engineering, and specially in computer graphics), one can use bicubic spline and code given in ([5], pp. 120-121.)

6.7. Prony's interpolation

Dating from 1795, Prony's interpolation ([14]) is often known as Prony's exponential approximation, and until nowadays not applied as it by its sophisticated nature deserves. It is suggested to students and readers to apply the following formulas in developing algorithms for programming of Prony's method. Some benchmarking research for comparison of application of this method, cubic spline, and, for example, least square method to some physical problems is also strongly encouraged.

If we have interpolation function of form

$$\begin{aligned} f(x) &\cong c_1 e^{a_1 x} + c_2 e^{a_2 x} + \cdots + c_n e^{a_n x} \\ &= c_1 \mu_1^x + c_2 \mu_2^x + \ldots + c_n \mu_n^x, \end{aligned}$$

where $\mu_k = e^{a_k}$. If function f is given on set of equidistant points $\{(x_k, f_k)\}_{k=0,1,\ldots,2n-1}$, and $x_k - x_{k-1} = h = const$ $(k = 1, 2, \ldots, 2n-1)$, by replacing $x = x_0 + kh$ data set can be replaced by $\{(k, f_k)\}_{k=0,1,\ldots,2n-1}$, where $x = 0, 1, \ldots, 2n-1$. By setting interpolation problem

$$\Phi(k) = f_k \quad (k = 0, 1, \ldots, 2n-1), \tag{6.7.1}$$

we get the system of equations

(6.7.2)
$$\begin{aligned} c_1 + c_2 + \ldots + c_n &= f_0 \\ c_1\mu_1 + c_2\mu_2 + \ldots + c_n\mu_n &= f_1 \\ c_1\mu_1^2 + c_2\mu_2^2 + \ldots + c_n\mu_n^2 &= f_2 \\ &\vdots \\ c_1\mu_1^{N-1} + c_2\mu_2^{N-1} + \ldots + c_n\mu_n^{N-1} &= f_{N-1}. \end{aligned}$$

If μ's are known (or preassigned) and $N = n$, system (6.7.2) is to be solved exactly as system of linear equations, and if $N > n$ approximately by least squares method (see next chapter).

If μ's are to be determined, we need at least $2n$ equations, but we have system of nonlinear equation, which, as we know, in general case could be unsolvable. Therefore, we can assume that μ's are the roots of algebraic polynomial of form

(6.7.3)
$$\mu^n + \alpha_1\mu^{n-1} + \ldots + \alpha_{n-1}\mu + \alpha_n = 0,$$
i.e
$$(\mu - \mu_n)(\mu - \mu_{n-1}) \ldots (\mu - \mu_1) = 0.$$

By multiplying all equations in (6.7.21) by $\alpha_n, \alpha_{n-1}, \ldots, \alpha_1, 1$, we get the system

(6.7.4)
$$\begin{aligned} f_0\alpha_n + f_1\alpha_{n-1} + f_2\alpha_{n-2} + \cdots + f_{n-1}\alpha_1 &= -f_n \\ f_1\alpha_n + f_2\alpha_{n-1} + f_3\alpha_{n-2} + \cdots + f_n\alpha_1 &= -f_{n+1} \\ &\vdots \\ f_{N-n-1}\alpha_n + f_{N-n}\alpha_{n-1} + f_{N-n+1}\alpha_{n-2} + \cdots + f_{N-2}\alpha_1 &= -f_{N-1}. \end{aligned}$$

If determinant
$$\begin{vmatrix} f_0 & f_1 & \cdots & f_{n-1} \\ f_1 & f_2 & \cdots & f_n \\ \vdots & & & \\ f_{N-n-1} & f_{N-n} & \cdots & f_{N-2} \end{vmatrix} \neq 0,$$
the solution of system (6.7.4) is unique. If $N = 2n$ we get system of linear equations, and if $N > 2n$ we solve this system by least squares method.

After the α's are determined, the n μ's are found as the roots of (6.7.3). The equation (6.7.2) then become linear in the n c's, with known coefficients. Thus the nonlinearity of the system is concentrated in the single algebraic equation (6.7.3).

6.8. Packages for interpolation of functions

Many libraries and software packages are available for interpolation and extrapolation of functions. Many workstations and mainframe computers have such libraries attached to their operating systems.

Many commercial software packages contain algorithms and programs for interpolation. Some of more prominent packages are `Matlab` and `Mathcad`. Some more sophisticated packages, such as `IMSL`, `MATHEMATICA`, `MACSYMA`, and `MAPLE`, also contain routines for polynomial interpolation and approximation of functions. The book *Numerical Recipes* [5] contains a routines for interpolation (See chap. 3, Interpolation and Extrapolation), and the book *Numerical Methods for Engineers and Scientists* ([2]) program code for difference formulas and numerical differentiation.

For demands on interpolation in two or more dimensions, higher order for accuracy, ar higher order of smoothness (bicubic interpolation, bicubic spline) see code in ([5], p. 118-122).

Bibliography (Cited references and further reading)

[1] Milovanović, G.V., *Numerical Analysis II*, Naučna knjiga, Beograd, 1988 (Se bian).

[2] Hoffman, J.D., *Numerical Methods for Engineers and Scientists.* Tayl & Francis, Boca Raton-London-New York-Singapore, 2001.

[3] Milovanović, G.V. and Djordjević, Dj.R., *Programiranje numeričkih metoc na FORTRAN jeziku.* Institut za dokumentaciju zaštite na radu "Edvar Kardelj", Niš, 1981 (Serbian).

[4] Stoer, J., and Bulirsch, R., *Introduction to Numerical Analysis*, Springe New York, 1980.

[5] Press, W.H., Flannery, B.P., Teukolsky, S.A., and Vetterling, W.T., *Numeric Recepies - The Art of Scientific Computing.* Cambridge University Pres 1989.

[6] Karlin, S. and Studden, W.J., *Tchebyshev Systems With Applications Analysis and Statistics.* Interscience Publishers, New York, 1966.

[7] Milovanović, G.V. and Kovačević, M.A., *Zbirka rešenih zadataka iz nu meričke analize.* Naučna knjiga, Beograd, 1985. (Serbian).

[8] Goertzel, G., *An algorithm for the evaluation of finite trigonometr series.* Amer. Math. Monthly 65(1958).

[9] Cooley, J.W., Lewis, P.A.W, Welch, P.D., *Historical notes on the fast Fouri transform.* Proc. IEEE 55(1967), 1675-1677.

[10] Cooley, J.W., Tukey, J.W., *An algorithm for the machine calculation complex Fourier series.* Math. Comp. 19(1965), 297-301.

[11] Quade,W. and Collatz, L., *Zur Interpolationstheorie der reellen periodi chen Funktionen.* S. -B. Preus. Akad. Wiss. Phys.-Math. Kl. 30(1938), 383-42

[12] Courant, R., *Variational methods for the solution of problem of equili rium and vibrations.* Bull. Amer. Math. Soc. 49(1943), 1-23.

[13] Schoenberg,I.J., *Contributions to the problem of approximation of equ distant data by analytic functions.* Quart. Appl. Math. 4(1946), 45-99; 11 141.

[14] Prony, R. De, *Essai expérimentale et analytique...* J. Ec. Polytech. Pa 1(2) (1795), 24-76.

[15] Ralston,A., *A First Course in Numerical Analysis.* McGraw-Hill, New Yor 1965.

[16] *IMSL Math/Library Users Manual* , IMSL Inc., 2500 City West Boulevar Houston TX 77042

[17] *NAG Fortran Library*, Numerical Algorithms Group, 256 Banbury Road, O ford OX27DE, U.K., Chapter F02.

LECTURES

LESSON VII

7. Best Approximation of Functions

7.1. Introduction

This chapter is devoted to approximations of functions most applied in different areas of sciences and engineering.

Let function $f : [a, b] \to R$ given by set of value pairs (x_j, f_j) $(j = 0, 1, \ldots, m)$ where
$f_j \equiv f(x_j)$. Consider the problem of approximation of function f by linear approximation function

$$\Phi(x) \equiv \Phi(x; a_0, \ldots, a_n) = \sum_{i=0}^{n} a_i \phi_i(x).$$

where $m > n$ (for $m = n$ we have interpolation). Proceeding like at interpolation, we get so known overdefined system of equations

$$\sum_{i=0}^{n} a_i \phi_i(x_j) = f_j \quad (j = 0, 1, \ldots, m), \tag{7.1.1}$$

which in general case does not have solution, i.e. all equations of system (7.1.1) can not be contemporary satisfied. If we define δ_n by

$$\delta_n(x) = f(x) - \sum_{i=0}^{n} a_i \phi_i(x), \tag{7.1.2}$$

it is possible to search for "solution" of system (7.1.1) so that

$$||\delta_n^*||_r = \min_{a_i} ||\delta_n||_r, \tag{7.1.3}$$

where

$$||\delta_n||_r = \Big(\sum_{j=0}^{m} |\delta_n(x_j)|^r\Big)^{1/r} \quad (r \geq 1).$$

The equality (7.1.3) gives the criteria for determination of parameters $a_0, a_1, \ldots, a_n$ in approximation function Φ. The quantity $||\delta_n^*||_r$, which exists always, is called the value of best approximation in l^r. Optimal values of parameters $a_i = \bar{a}_i$ $(i = 0, 1, \ldots, n)$ in sense of (7.1.3) give the best l^r approximation function

$$\Phi^*(x) = \sum_{i=0}^{n} \bar{a}_i \phi_i(x).$$

Most frequently is taken

1. $r = 1$, $||\delta_n||_1 = \sum\limits_{j=0}^{m} |\delta_n(x_j)|$ (best l^1 approximation),
2. $r = 2$, $||\delta_n||_2 = (\sum\limits_{j=0}^{m} \delta_n(x_j)^2)^{1/2}$ (best l^2 approximation),
3. $r = +\infty$, $||\delta_n||_\infty = \max\limits_{0\le j\le m} |\delta_n(x_j)|$ (Tchebyshev min-max approximation).

In a similar way can be considered problem of best approximation of function f in space $L^r(a,b)$. Here we have

$$||\delta_n||_r = \Big(\int_a^b |\delta_n(x)|^r \, dx\Big)^{1/r} \qquad (1 \le r < +\infty)$$

and

$$||\delta_n||_\infty = \max_{a\le x\le b} |\delta_n(x)|.$$

By introducing weight function $p : (a,b) \to R^+$ the more general case of mean-square approximations can be considered, where the corresponding norms for discrete and continuous case are given as (see [1], pp. 90-91)

(7.1.4)
$$||\delta_n||_2 = ||\delta_n||_{2,p} = \Big(\sum_{j=0}^{m} p(x_j)\delta_n(x_j)^2\Big)^{1/2}$$

and

(7.1.5)
$$||\delta_n||_2 = ||\delta_n||_{2,p} = \Big(\int_a^b p(x)\delta_n(x)^2 \, dx\Big)^{1/2},$$

respectively.

Example 7.1.1. *Function $x \to f(x) = x^{1/3}$ is to approximate with function $x \to \phi(x) = a_0 + a_1 x$ in space*

1^0 $L^1(0,1)$, $\qquad$ 2^0 $L^2(0,1)$, $\qquad$ 3^0 $L^\infty(0,1)$.

Here we have $\delta_1(x) = x^{1/3} - a_0 - a_1 x \quad (0 \le x \le 1)$. (see [1], pp. 91-93).

1^0 We get the best $L^1(0,1)$ approximation by minimization of norm

$$||\delta_1||_1 = \int_0^1 |x^{1/3} - a_0 - a_1 x| dx.$$

Having $\dfrac{\partial \delta_1}{\delta a_0} = -1$, and $\dfrac{\partial \delta_1}{\delta a_1} = -x$, the optimal values of parameters a_0 and a_1 are to be determined from the system of equations

(7.1.6)
$$\int_0^1 \operatorname{sgn} \delta_1(x) \, dx = 0,$$
$$\int_0^1 x \operatorname{sgn} \delta_1(x) \, dx = 0.$$

Having in mind that δ_1 changes sign on segment $[0,1]$ in points x_1 and x_2 (see fig. 7.1.1), system of equations (7.1.6) reduces to system

$$x_2 - x_1 = \frac{1}{2}, \qquad x_2^2 - x_1^2 = \frac{1}{2},$$

wherefrom it follows $x_1 = \frac{1}{4}$ and $x_2 = \frac{3}{4}$.

Thus, determining the best $L^1(0,1)$ approximation reduces to interpolation, i.e. determining of interpolation polynomial Φ^* which satisfies the conditions

$$\Phi^*(1/4) = f(1/4) = \sqrt[3]{\frac{1}{4}}, \quad \Phi^*(3/4) = f(3/4) = \sqrt[3]{\frac{3}{4}},$$

i.e.

$$\Phi^*(x) = \frac{2}{\sqrt[3]{4}}(\sqrt[3]{3} - 1)x + \frac{1}{2\sqrt[3]{4}}(3 - \sqrt[3]{3})$$
$$\cong 0.55720x + 0.49066.$$

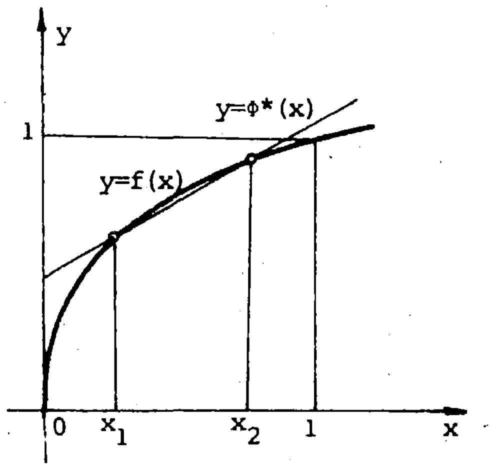


Figure 7.1.1

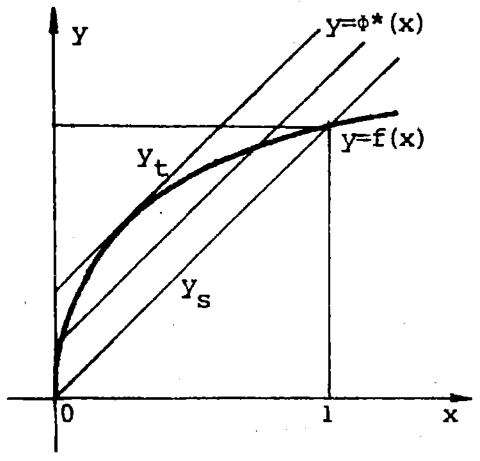


Figure 7.1.2

2^0 Let

$$I(a_0, a_1) = ||\delta_1||_2^2 = \int_0^1 (x^{1/3} - a_0 - a_1x)^2 dx.$$

From the conditions

$$\frac{\partial I}{\partial a_0} = -2\int_0^1 (x^{1/3} - a_0 - a_1x)dx = 0,$$

$$\frac{\partial I}{\partial a_1} = -2\int_0^1 x(x^{1/3} - a_0 - a_1x)dx = 0,$$

it follows

$$a_0 + \frac{1}{2}a_1 = \frac{3}{4}, \quad \frac{1}{2}a_0 + \frac{1}{3}a_1 = \frac{3}{7},$$

i.e. $a_0 = \overline{a_0} = \frac{3}{7}$, $a_1 = \overline{a_1} = \frac{9}{14}$, so that the best mean-square approximation is given with

$$\Phi^*(x) = \frac{3}{7} + \frac{9}{14}x \cong 0.42857x + 0.64286x.$$

3^0 For determining min-max approximation we will use the following simple geometrical procedure. Through the end-points of curve $y = f(x) = x^{1/3}$ $(0 \le x \le 1)$ we will draw the secant, and then tangent on the curve which is parallel to this secant (see Fig. 7.1.2). The corresponding equations for those straight lines are

$$y = y_{sec} = x \quad \text{and} \quad y = y_{tan} = x + \frac{2\sqrt{3}}{9},$$

so that the best min-max approximation is

$$\Phi^*(x) = \frac{1}{2}(y_{sec} + y_{tan}) = x + \frac{\sqrt{3}}{9} \cong x + 0.19245,$$

whereby the the value of best approximation is $||\delta_1^*||_\infty = \frac{\sqrt{3}}{9}$.

7.2. Best L^2 approximation

Here, we will consider the problem of best L^2approximation of function $f : [a,b] \to R$ using linear approximation function

$$\Phi(x) = \sum_{i=0}^{n} a_i \Phi_i(x),$$

where $\{\Phi_i\}$ is system of linear independent functions from the space $L^2(a,b)$, with scalar product introduced by

$$(f,g) = \int_a^b p(x)f(x)g(x)dx \quad (f,g \in L^2(a,b)),$$

where $p : (a,b) \to R^+$ is given weight function.

From the previous section we can conclude that for the best mean-square approximation for f it is necessary to minimize the norm (7.1.5) by parameters a_i $(i = 0,1,\ldots,n)$..

If we put $I(a_0, a_1, \ldots, a_n) = ||\delta_n||^2 = (\delta_n, \delta_n)$, then from

$$\frac{\partial I}{\partial a_j} = 2\int_a^b p(x)(f(x) - \sum_{i=0}^{n} a_i\Phi_i(x))(-\Phi_j(x))dx = 0 \quad (j = 0,1,\ldots,n)$$

it follows system of equations for determination of approximation parameters

$$\sum_{i=0}^{n}(\Phi_i, \Phi_j)a_i = (f, \Phi_j) \quad (j = 0,1,\ldots,n). \tag{7.2.1}$$

This system can be represented in matrix form as

$$\begin{bmatrix} (\Phi_0, \Phi_0) & (\Phi_1, \Phi_0) & \dots & (\Phi_n, \Phi_0) \\ (\Phi_0, \Phi_1) & (\Phi_1, \Phi_1) & & (\Phi_n, \Phi_1) \\ \vdots & & & \\ (\Phi_0, \Phi_n) & (\Phi_1, \Phi_n) & & (\Phi_n, \Phi_n) \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} (f, \Phi_0) \\ (f, \Phi_1) \\ \vdots \\ (f, \Phi_n) \end{bmatrix}.$$

Matrix of this system is known as Gram's matrix. It can be shown that this matrix is regular if system of functions $\{\Phi_i\}$ is linearly independent, having unique solution of given approximation problem.

System of equations (7.2.1) can be simple solved if the system of functions $\{\Phi\}$ is orthogonal. Namely, then all off-diagonal elements of matrix of system are equal to zero, i.e. matrix is diagonal one, having as solutions

$$a_i = \overline{a_i} = \frac{(f, \Phi_i)}{(\Phi_i, \Phi_i)} \quad (i = 0, 1, \dots, n). \tag{7.2.2}$$

It can be shown that by taking in the given way chosen parameters a_i $(i = 0, 1, \dots, n)$ the function I reaches its minimal value. Namely, because

$$\frac{\partial^2 I}{\partial a_j \partial a_k} = 2(\Phi_k, \Phi_j) = 2||\Phi_k||^2 \delta_{kj},$$

where δ_{kj} is Cronecker delta, we have

$$d^2 I = 2 \sum_{i=0}^{n} ||\Phi_k||^2 da_k^2 > 0.$$

Thus, the best L^2 approximation of function f in subspace $X_n = L(\Phi_0, \Phi_1, \dots, \Phi_n)$, where $\{\Phi_i\}$ is orthogonal system of functions, is given as

$$\Phi^*(x) = \sum_{i=0}^{n} \frac{(f, \Phi_i)}{||\Phi_i||^2} \Phi_i(x). \tag{7.2.3}$$

A very important class of mean-square approximations is approximation by algebraic polynomials. In this case, the orthogonal basis of subspace X_n is constructed by Gramm-Schmidt orthogonalization procedure, starting, for example, from natural basis $\{1, x, x^2, \dots, x^n\}$, or general methods for orthogonalization.

Example 7.2.1. *For function $x \to f(x) = |x|$ on segment $[-1, 1]$ determine in the set of polynomials not greater degree than three, best L^2 approximation, with weight function $x \to p(x) = (1 - x^2)^{3/2}$.*

Let us compute integral

$$N_k = \int_{-1}^{1} x^{2k} (1 - x^2)^{3/2} \, dx \quad (k \in N_0)$$

needed for further considerations (see [4], pp. 92-93). By partial integration over the integral

$$N_{k-1} - N_k = \int_{-1}^{+1} x^{2(k-1)} (1 - x^2)^{5/2} \, dx \quad (k \in N),$$

we get $N_{k-1}-N_k = \frac{5}{2k-1}N_k$, i.e. $N_k = \frac{2k-1}{2k+4}N_{k-1}$, $(k \in N)$ so that, with $N_0 = \frac{3\pi}{8}$ we have $N_k = 3\pi\frac{(2k-1)!!}{(2k+4)!!}$ $(k \in N)$. Starting from natural basis $\{1, x, x^2, \ldots\}$, using Gramm-Schmidt orthogonalisation, we get subsequently

$$\Phi_0(x) = 1$$
$$\Phi_1(x) = x - \frac{(x, \Phi_0)}{(\Phi_0, \Phi_0)}\Phi_0(x) = x,$$
$$\Phi_2(x) = x^2 - N_1 N_0^{-1} = x^2 - \frac{1}{6},$$
$$\Phi_3(x) = x^3 - N_2 N_1^{-1}x = x^3 - \frac{3}{8}x,$$

and corresponding norms

$$||\Phi_0|| = \sqrt{\frac{3\pi}{8}},\ ||\Phi_1|| = \frac{\sqrt{\pi}}{4},\ ||\Phi_2|| = \frac{1}{8}\sqrt{\frac{5\pi}{5}},\ ||\Phi_3|| = \frac{\sqrt{3\pi}}{32}.$$

Because of

$$(f, \Phi_0) = \frac{2}{5},\ (f, \Phi_1) = 0,\ (f, \Phi_2) = \frac{1}{21},\ (f, \Phi_3) = 0,$$

using (7.2.2) we get

$$a_0 = \frac{16}{15\pi},\ a_1 = 0,\ a_2 = \frac{128}{35\pi},\ a_3 = 0,$$

having, finally approximation in the form

$$\Phi^*(x) = \frac{16}{15\pi} + \frac{128}{35\pi}(x^2 - \frac{1}{6}) \cong 0.14551309 + 1.1641047x^2.$$

This function is, in addition, best approximation in the set of polynomials of degree not greater than two.

Some further very valuable considerations can be found in [1], pp. 96-99.

7.3. Best l^2 approximation

In previous sections we considered problem of best approximation of function in space $L^2(a, b)$. Now we will consider a particular case, mentioned in introductory section. Namely, let function $f : [a, b] \to R$ be given on set of pairs of values $\{(x_j, f_j)\}_{j=0,1,\ldots,m}$, where $f_j \equiv f(x_j)$. We will consider the problem of best approximation of given function by linear approximation function

$$\Phi(x) = \sum_{i=0}^{n} a_i \Phi_i(x) \quad (n < m) \tag{7.3.1}$$

in sense of minimization of norm (7.1.4), where $p : [a, b] \to R_+$ is given weight function and δ_n defined by (7.1.2). By involving the matrix notation

$$\mathbf{X} = \begin{bmatrix} \Phi_0(x_0) & \Phi_1(x_0) & \ldots & \Phi_n(x_0) \\ \Phi_0(x_1) & \Phi_1(x_1) & \ldots & \Phi_n(x_1) \\ \vdots & & & \\ \Phi_0(x_m) & \Phi_1(x_m) & \ldots & \Phi_n(x_m) \end{bmatrix},\ \vec{f} = \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_n \end{bmatrix},\ \vec{a} = \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix},$$

$$\mathbf{P} = \operatorname{diag}(p(x_0), p(x_1), \ldots, p(x_m)), \quad \vec{v} = \vec{f} - \mathbf{X}\vec{a},$$

square of norm, defined by (7.1.4), can be represented as

$$F = ||\delta_n||^2 = ||\delta_n||_2^2 = \sum_{j=0}^{m} p(x_j)\delta_n(x_j)^2 = \vec{v}^T\mathbf{P}\vec{v}. \tag{7.3.2}$$

For determination of best discrete mean-square approximation (7.3.1) it is necessary to minimize F, given by (7.3.2). Thus, based on

$$\frac{\partial F}{\partial a_i} = 2\sum_{j=0}^{m} p(x_j)\delta_n(x_j)\frac{\partial \delta_n(x_j)}{\partial a_i} = 0 \quad (i = 0, 1, \ldots, n)$$

we get normal system of equations

$$\sum_{j=0}^{m} p(x_j)\delta_n(x_j)\Phi_i(x_j) = 0 \quad (i = 0, 1, \ldots, n) \tag{7.3.3}$$

for determination of parameters a_i $(i = 0, 1, \ldots, n)$. The last system of equations can be given in matrix form

$$\mathbf{X}^T\mathbf{P}\vec{v} = \vec{0},$$

i.e.

$$\mathbf{X}^T\mathbf{P}\mathbf{X}\vec{a} = \mathbf{X}^T\mathbf{P}\vec{f}. \tag{7.3.4}$$

Note that normal system of equations (7.3.3), i.e (7.3.4) is obtained from overdefined system of equations (7.1.1), given in matrix form as

$$\mathbf{X}\vec{a} = \vec{f},$$

by simple multiplication by matrix $\mathbf{X}^T\mathbf{P}$ from the left side.

Diagonal matrix $\mathbf{P}$, which is called weight matrix, is of meaning so that larger weights $p_j \equiv p(x_j)$ are assigned to the values of function f_j with greater accuracy. This is of importance when approximating experimental data, which are obtained during measures by different accuracy. For example, for measurements realized with different dispersions, which relations are known, the weights p_j are chosen as inverse of dispersions, i.e, such that

$$p_0 : p_1 : \cdots : p_m = \frac{1}{\sigma_0^2} : \frac{1}{\sigma_1^2} : \cdots : \frac{1}{\sigma_m^2}.$$

When the measurements are realized with same accuracy, but with different numbers of measurements, i.e. for every value of argument x_j are proceeded m_j measurements, and for f_j taken arithmetic means of obtained results in series of measurements, then for weights are taken numbers of measurements in series, i.e. $p_j = m_j$ $(j = 0, 1, \ldots, m)$. Nevertheless, usually are the weights equal, i.e. $\mathbf{P}$ is unit matrix of order $m+1$. In this case, (7.3.4) reduces to

$$\mathbf{X}^T\mathbf{X}\vec{a} = \mathbf{X}^T\vec{f}. \tag{7.3.5}$$

Vector of coefficients $\vec{a}$ is determined from (7.3.4) or (7.3.5). From (7.3.5) it follows

$$\vec{a} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\vec{f}.$$

In case when the system of basic functions is chosen so that $\Phi_i(x) = x^i$ $(i = 0, 1, \ldots, n)$ we have

$$\mathbf{X} = \begin{bmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^n \\ 1 & x_1 & x_1^2 & & x_1^n \\ \vdots & & & & \\ 1 & x_m & x_m^2 & & x_m^n \end{bmatrix}.$$

The method considered is often called least-squares method. Interesting case is when $n = 1$, i.e. when the approximation function is of form $\Phi(x) = a_0 + a_1x$. Then the system (7.3.4) becomes

$$\begin{bmatrix} s_{11} & s_{12} \\ s_{21} & s_{22} \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \end{bmatrix},$$

where

$$s_{11} = \sum_{j=0}^{m} p_j, \; s_{12} = s_{21} = \sum_{j=0}^{m} p_j x_j, \; s_{22} = \sum_{j=0}^{m} p_j x_j^2,$$

$$b_0 = \sum_{j=0}^{m} p_j f_j, \; b_1 = \sum_{j=0}^{m} p_j x_j f_j.$$

The asked approximation parameters are

$$a_0 = \frac{1}{D}(s_{22}b_0 - s_{12}b_1), \; a_1 = \frac{1}{D}(s_{11}b_1 - s_{21}b_0),$$

where $D = s_{11}s_{22} - s_{12}^2$.

Example 7.3.1. *Find parameters a_0 and a_1 in approximation function $\Phi(x) = a_0 + a_1x$ using least-squares method. for function given in tabular form, as a set of values pairs*

$$\{(1.1, 2.5), \; (1.9, 3.2), \; (4.2, 4.5), \; (6.1, 6.0)\}.$$

For weight matrix $\mathbf{P}$ we can take unit matrix. The previously given formulas can be directly applied, but we can start from overdefined system of equations

$$\begin{bmatrix} 1 & 1.1 \\ 1 & 1.9 \\ 1 & 4.2 \\ 1 & 6.1 \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = \begin{bmatrix} 2.5 \\ 3.2 \\ 4.5 \\ 6.0 \end{bmatrix}.$$

By multiplying with matrix $\mathbf{X}^T = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1.1 & 1.9 & 4.2 & 6.1 \end{bmatrix}$ from the left side, we get the normal system of equations

$$\begin{bmatrix} 4 & 13.3 \\ 13.3 & 59.67 \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = \begin{bmatrix} 16.2 \\ 64.33 \end{bmatrix},$$

wherefrom it follows

$$\begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = \frac{1}{61.79} \begin{bmatrix} 59.67 & -13.3 \\ -13.3 & 4. \end{bmatrix} \cdot \begin{bmatrix} 16.2 \\ 64.33 \end{bmatrix} = \begin{bmatrix} 1.7974591 \\ 0.6774559 \end{bmatrix},$$

Thus, we have $\Phi(x) = 1.7974591 + 0.6774559x$.

In many areas of science and technology, dealing with experimental data, we have often problem of parameter determination in so known empirical formulas which express functional relation between two or more variables. For example, let functional relation be given as

$$y = f(x;\, a_0, a_1, \ldots, a_n),$$

where a_i $(i = 0, 1, \ldots, n)$ are parameters which are to be determined using the following tabulated data obtained by measurement.

i	0	1	$\ldots$	m
x_i	x_0	x_1	$\ldots$	x_m
y_i	y_0	y_1	$\ldots$	y_m

The measured data contain accidental errors of measurements, i.e. "noise" in experiment. Determination of parameters a_i $(i = 0, 1, \ldots, n)$ is, from the point of theory of approximation, possible only if $m \geq n$. In case of $m = n$, we have interpolation, which is, in general case nonlinear, what depends on function shape. In order to eliminate "noise" in data, and obtain greater accuracy and reliability, the number of measurements should be large enough. Then, the most used method for determination of parameters is least-square method, i.e. minimization of variable F defined by

$$F = \sum_{j=0}^{m} (y_j - f(x_j;\, a_0, a_1, \ldots, a_n))^2, \tag{7.3.6}$$

or using

$$F = \sum_{j=0}^{m} p_j\, (y_j - f(x_j;\, a_0, a_1, \ldots, a_n))^2,$$

where are included weights p_j. If functional relation between several variables is given as

$$z = f(x, y;\, a_0, a_1, \ldots, a_n)$$

for determination of approximation parameters we have to minimize

$$F = \sum_{j=0}^{m} p_j\, (z_j - f(x_j, y_j;\, a_0, a_1, \ldots, a_n))^2.$$

If f is linear approximation function (in parameters $a_0, a_1, \ldots, a_n$), i.e. of form (7.3.1), the problem is to be solved in previously explained way. Nevertheless, if f is nonlinear approximation function, then the corresponding normal system of equation

$$\frac{\partial F}{\partial a_i} = 0 \quad (i = 0, 1, \ldots, n) \tag{7.3.7}$$

is nonlinear. For solving of this system can be used some method for solution of system of nonlinear equations, like Newton-Kantorovich method, whereby this procedure is rather complicated. In order to solve problem in easier way, there are some simplified methods of transformation of such problems to linear approximation method. Namely, by introducing some substitutions, like

$$X = g(x), \quad Y = h(y) \tag{7.3.8}$$

nonlinear problem reduces to linear one.

For example, let $y = f(x; a_0, a_1) = a_0 e^{a_1 x}$. Then, by logaritmization and substitution

$$X = x, \quad Y = \log y, \quad b_0 = \log a_0, \quad b_1 = a_1,$$

the problem reduces to linear one, because now $Y = b_0 + b_1 X$. Thus, by minimization of

$$G = G(b_0, b_1) = \sum_{j=0}^{m} (Y_j - b_0 - b_1 X_j)^2, \tag{7.3.9}$$

where $X_j = x_j$ and $Y_j = \log y_j$ $(j = 0, 1, \ldots, n)$, we determine the parameters b_0 and b_1, and then

$$a_0 = e^{b_0} \quad \text{and} \quad a_1 = b_1.$$

Nevertheless, this procedure does not give the same result like the minimization of function

$$F = F(a_0, a_0) = \sum_{j=0}^{m} (y_j - a_0 e^{a_1 x_j})^2.$$

Moreover, the obtained results can significantly deviate, because the problem we are solving is different from stated one, having in mind transformation we have done ($Y = \log y$). But, for many practical engineering problems the parameters obtained in this way are satisfactory.

We will specify some typical functional dependencies with possible transformation of variables.

1^0 $y = a_0 x^{a_1}$, $X = \log x$, $Y = \log y$, $b_0 = \log a_0$, $b_1 = a_1$;

2^0 $y = a_0 a_1^x$, $X = x$, $Y = \log y$, $b_0 = \log a_0$, $b_1 = \log a_1$;

3^0 $y = a_0 + \dfrac{a_1}{x}$, $X = \dfrac{1}{x}$, $Y = y$, $b_0 = a_0$, $b_1 = a_1$;

4^0 $y = a_0 + \dfrac{a_1}{x}$, $X = x$, $Y = xy$, $b_0 = a_1$, $b_1 = a_0$;

5^0 $y = \dfrac{1}{a_0 + a_1 x}$, $X = x$, $Y = \dfrac{1}{y}$, $b_0 = a_0$, $b_1 = a_1$;

6^0 $y = \dfrac{x}{a_0 + a_1 x}$, $X = \dfrac{1}{x}$, $Y = \dfrac{1}{y}$, $b_0 = a_1$, $b_1 = a_0$;

7^0 $y = \dfrac{x}{a_0 + a_1 x}$, $X = x$, $Y = \dfrac{x}{y}$, $b_0 = a_0$, $b_1 = a_1$;

8^0 $y = \dfrac{1}{a_0 + a_1 e^{-x}}$, $X = e^{-x}$, $Y = \dfrac{1}{y}$, $b_0 = a_0$, $b_1 = a_1$;

9^0 $y = a_0 + a_1 \log x$, $X = \log x$, $Y = y$, $b_0 = a_0$. $b_1 = a_1$.

Example 7.3.2. *Result of measurements of values x and y are given in following tabular form.*

i	0	1	2	3	4	5
x_i	4.48	4.98	5.60	6.11	6.62	7.42
y_i	4.15	1.95	1.31	1.03	0.74	0.63

If $y = \dfrac{1}{a_0 + a_1 x}$, reduce to linear problem and approximate using least-square method.

By involving $X = x$. $Y = 1/y$ and using least-square method we get approximation function $\Phi(X) \cong 0.468X - 1.843$. wherefrom it follows $y \cong \dfrac{1}{0.468x - 1.843}$.

From the previous one can conclude that, depending on f the convenient replacements (7.3.8) should be chosen so that they enable reducing of

$$y = f(x; a_0, a_1, \dots, a_n)$$

to linear form of, for example, polynomial type

$$Y = b_0 + b_1 X + \dots + b_n X^n. \tag{7.3.10}$$

It is clear that functions g and h must have their inverse functions, so that (7.3.10) is, in fact, equivalent to

$$h^{-1}(Y) = f(g^{-1}(X); a_0, a_1, \dots, a_n),$$

whereby parameters b_i depend on parameters a_i in rather simple way.

7.4. Packages for approximation of functions

Procedures for developing polynomials for discrete data are very important in engineering practice. The direct fit polynomials, the Lagrange polynomial, and the divided difference polynomial work well for nonequally spaced data. For equally spaced data, polynomials based on differences are recommended.

Procedures for developing least squares approximations for discrete data are also valuable in engineering practice. Least squares approximations are useful for large sets of data and sets of rough data. Least square polynomial approximation is straightforward, for both one independent variable and more than one variable. The least squares normal equations corresponding to polynomial approximating functions are linear, which leads to very efficient solving procedures. For nonlinear approximating functions, the least squares normal equations are nonlinear, which leads to complicated solution procedures. As previously mentioned, convenient mapping of nonlinear approximating function to linear one (i.e. linearization) can solve this problem usually good enough. Least squares polynomial approximation is a straightforward, simple, and accurate procedure for obtaining approximating functions for large sets of data or sets of rough experimental data.

Numerous libraries and software packages are available for approximation of functions, especially for polynomial approximation.

Many commercial software packages contain routines for fitting approximating polynomials. Some of the more prominent packages are `Matlab` and `Mathcad`. More sophisticated packages, such as `IMSL`, `Mathematica`, and `Macsyma` contain also routines for fitting approximating polynomials. The book *Numerical Recipes* ([6]) contains numerous subroutines for fitting approximating polynomials (see Chapter 15, Modelling of Data), and the book *Numerical Methods for Engineers and Scientists* ([2]) program code for fitting approximating polynomials (see Chapter 4, Polynomial Approximation and Interpolation).

Bibliography (Cited references and further reading)

[1] Milovanović, G.V., *Numerical Analysis II*, Naučna knjiga, Beograd, 1988 (Serbian).

[2] Hoffman, J.D., *Numerical Methods for Engineers and Scientists.* Taylor & Francis, Boca Raton-London-New York-Singapore, 2001.

[3] Milovanović, G.V. and Djordjević, Dj.R., *Programiranje numeričkih metoda na FORTRAN jeziku.* Institut za dokumentaciju zaštite na radu "Edvard Kardelj", Niš, 1981 (Serbian).

[4] Milovanović, G.V., *Numerical Analysis I*, Naučna knjiga, Beograd, 1988 (Serbian).

[5] Stoer, J., and Bulirsch, R., *Introduction to Numerical Analysis*, Springer, New York, 1980.

[6] Press, W.H., Flannery, B.P., Teukolsky, S.A., and Vetterling, W.T., *Numerical Recepies - The Art of Scientific Computing.* Cambridge University Press, 1989.

[7] Milovanović, G.V. and Wrigge, S., *Least squares approximation with constraints.* Math. Comp. 46(1986), 551-565.

[8] Milovanović, G.V. and Kovačević, M.A., *Zbirka rešenih zadataka iz numeričke analize.* Naučna knjiga, Beograd, 1985. (Serbian).

[9] Đorđević, Đ.R., Ilić, Z.A., *Realization of Stieltjes procedure and Gauss-Christoffel quadrature formulas in system* Mathematica. Zb. rad. Gradj. fak. Niš 17(1996), pp. 29-40.

[10] Todd, J., *Introduction to the Constructive Theory of Functions.* New York, 1963.

[11] Schoenberg,I.J., *Contributions to the problem of approximation of equidistant data by analytic functions.* Quart. Appl. Math. 4(1946), 45-99; 112-141.

[12] Ralston,A., *A First Course in Numerical Analysis.* McGraw-Hill, New York, 1965.

[13] Brass, H., *Approximation durch Teilsummen von Orthogonalpolynomreihen.* In: Numerical Methods of Approximation Theory (L. Collatz, G. Meinarus, H. Werner, eds.), 69-83, ISNM 52, Birkhäuser, Basel, 1980.

[14] Brass, H., *Error estimates for least squares approximation by polynomials.* J. Approx. Theory 41(1984), 345-349.

[15] Collatz, L. and Krabs, W., *Approximationstheorie. Tschebyscheffsche Approximation with Anwendungen.* B.G. Teubner, Stutgart, 1973.

[16] Davis, P.J., *Interpolation and Approximation.* Blaisdell Publ. Comp., New York, 1963.

[17] Feinerman, R.P. & Newman, D.J., *Polynomial Approximation.* Williams & Wilkins, Baltimore, 1974.

[18] Fox, L. & Parker, I.B., *Chebyshev Polynomials in Numerical Analysis.* Oxford Univ. Press, 1972.

[19] Hildebrand, F.B., *Introduction to Numerical Analysis.*Mc.Graw-Hill, New York, 1974.

[20] Rivlin, T.J., *An Introduction to the Approximation od Functions.* Dover Publications, Inc., New York, 1981.

[21] *IMSL Math/Library Users Manual* , IMSL Inc., 2500 City West Boulevard, Houston TX 77042

[22] *NAG Fortran Library,* Numerical Algorithms Group, 256 Banbury Road, Oxford OX27DE, U.K., Chapter F02.

Faculty of Civil Engineering
Belgrade
Master Study
COMPUTATIONAL ENGINEERING

Faculty of Civil Engineering and Architecture
Niš
Doctoral Study

LECTURES

LESSON VIII

8. Numerical Differentiation and Integration

8.1. Numerical Differentiation

In this section the numerical differentiation of real functions defined on $[a,b]$ will be considered.

8.1.1. Introduction

The need for numerical differentiation appears in following cases:

a. When values of function are known only on discrete set of points on $[a,b]$, i.e. function f is given in tabular form;

b. When analytical expression for function f is rather complicated.

Numerical differentiation is chiefly based on approximation of function f by function Φ on $[a,b]$, and then differentiating Φ desirable times. Thus, based on $f(x) \sim \Phi(x)$ $(a \le x \le b)$), we have

$$f^{(k)}(x) \sim \Phi^{(k)}(x) \quad (a \le x \le b;\ k = 1, 2, \ldots).$$

For function Φ are mostly taken algebraic interpolation polynomials, because being simple differentiating. Let Φ be interpolating polynomial of $n-$th degree, i.e.

$$\Phi(x) = P_n(x).$$

If known error $R_n(f;x)$ of approximation polynomial

$$f(x) = P_n(x) + R_n(f;x) \quad (a \le x \le b) \tag{8.1.1.1}$$

it is possible to estimate error in formula for differentiation, i.e. from (8.1.1.1) it follows

$$f^{(k)}(x) = P_n^{(k)}(x) + R_n^{(k)}(f;x) \quad (a \le x \le b)$$

It is meaningful to take for order of derivative only $k < n$.

It is obvious that numerical differentiation has smaller accuracy than interpolation. So, for example, for interpolation is error in nodes equal to zero, what is not in case of differentiation.

8.1.2. Formulas for numerical differentiation

If known values of function f on set of equidistant points $\{x_0, x_1, \ldots, x_m\} \subset [a,b]$, with step h, let

$$f_k = f(x_k) = f(x_0 + k \cdot h) \quad (k = 0, 1, \ldots, m).$$

Construct over set $\{x_i, x_{i+1}, \ldots, x_{i+n}\}$ $(0 \le i \le m-n)$ first Newton interpolation polynomial (see Chapter 6)

(8.1.2.1) $$P_n(x) = f_i + p\,\Delta f_i + \frac{p\,(p-1)}{2!}\Delta^2 f_i + \frac{p\,(p-1)(p-2)}{3!}\Delta^3 f_i + \ldots + \frac{p\,(p-1)\ldots(p-n+1)}{n!}\Delta^n f_i,$$

where $p = (x - x_i)/h$. Because $P_n'(x) = \dfrac{1}{h}\dfrac{dP_n(x)}{dp}$, by differentiation (8.1.2.1) we get

(8.1.2.2) $$P_n'(x) = \frac{1}{h}(\Delta f_i + \frac{2p-1}{2}\Delta^2 f_i + \frac{3p^2-6p+2}{6}\Delta^3 f_i + \ldots).$$

By further differentiation of (8.1.2.2) we get, in turn P_n'', P_n''', and so on. For example,

(8.1.2.3) $$P_n''(x) = \frac{1}{h^2}(\Delta^2 f_i + (p-1)\Delta^3 f_i + \ldots).$$

For $x = x_i$, i.e. $p = 0$, formulas (8.1.2.2) and (8.1.2.3) reduce to

$$P_n'(x_i) = \frac{1}{h}(\Delta f_i - \frac{1}{2}\Delta^2 f_i + \frac{1}{3}\Delta^3 f_i - \ldots + \frac{(-1)^{n-1}}{n}\Delta^n f_i),$$
$$P_n''(x_i) = \frac{1}{h^2}(\Delta^2 f_i - \Delta^3 f_i + \frac{11}{12}\Delta^4 f_i - \ldots).$$

Some useful formulas for the first derivative are

$$f'(x_i) = \frac{1}{h}(f_i - f_{i-1}) + \frac{h}{2}f''(\eta_1), \quad (x_{i-1} < \eta_1 < x_i),$$
$$f'(x_i) = \frac{1}{2h}(3f_i - 4f_{i-1} + f_{i-2}) + \frac{h^2}{3}f'''(\eta_2), \quad (x_{i-2} < \eta_2 < x_i),$$
$$f'(x_i) = \frac{1}{6h}(11f_i - 18f_{i-1} + 9f_{i-2} - 2f_{i-3}) + \frac{h^3}{4}f^{IV}(\eta_3) \quad (x_{i-3} < \eta_3 < x_i),.$$

Previous formulas for first derivative in node x_i are obviously asymmetric and are usually applied on the interval $[a,b]$ boundaries. Typical application of these formulas is at approximation of differentiable boundary conditions in contour problems of differential equations.

For nodes inside of segment $[a,b]$ is better to use symmetric differentiation formulas.

(1^0) $$Df_i = \frac{1}{2h}(f_{i+1} - f_{i-1}) - \frac{1}{6}h^2 f'''(\xi_1) \quad (x_{i-1} < \xi_1 < x_{i+1});$$

(2^0) $$Df_i = \frac{1}{h}(-f_{i+2} + 8f_{i+1} - 8f_{i-1} + f_{i-2}) + r_2(f),$$

where

$$r_2(f) = \frac{1}{30}h^4 f^V(\xi_2) \quad (x_{i-2} < \xi_2 < x_{i+2}).$$

The most used and simplest formula for approximation of second derivative is

$$D^2 f_i = \frac{1}{h^2}(f_{i+1} - 2f_i + f_{i-1}) + r(f),$$

where, under condition $f \in C^4[a,b]$, the remainder term is

$$r(f) = -\frac{h^2}{12} f^{IV}(\xi).$$

8.2. Numerical integration - Quadrature formulas

8.2.1. Introduction

Numerical integration of functions is dealing with approximative calculation of definite integrals on the basis of the sets of values of function to be integrated, by following some formula.

Formulas for calculation of single integrals are called **quadrature formulas**. In similar way, formulas for double integrals (and multi-dimensional integrals, too) are called cubature formulas.

In our considerations, we will deal mainly with quadrature formulas.

The need for numerical integration appears in many cases. Namely, Newton-Leibnitz formula

$$\int_a^b f(x)dx = F(b) - F(a), \tag{8.2.1.1}$$

where F is primitive function of function f, cannot always be applied. Note some of these cases:

1. Function F cannot be represented by finite number of elementary functions (for example, when $f(x) = e^{-x^2}$).
2. Application of formula (8.2.1.1) leads often to very complicated expression, even at calculation of integral of rather simple functions, e.g.

$$\int_0^a \frac{dx}{1+x^3} = \log\sqrt[3]{|a+1|} - \frac{1}{6}\log(a^2-a+1) + \frac{1}{\sqrt{3}} arctg \frac{a\sqrt{3}}{2-a}.$$

3. Integration of functions with values known on discrete set of points (obtained, for example, by experiments), is not possible by applying formula (8.2.1.1).

Large number of quadrature formulas are of form

$$\int_a^b f(x)dx \cong \sum_{k=0}^{n} A_k f_k, \tag{8.2.1.2}$$

where $f_k = f(x_k)$ $(a \le x_0 < \ldots < x_n \le b)$. If $x_0 = a$ and $x_n = b$, formula (8.1.1.2) is of closed kind, and in other cases is of open kind.

For integration of differentiable functions are used also formulas which have, in addition to function values, values of its derivatives. The formulas for calculation of integrals of form

$$\int_a^b p(x)f(x)dx,$$

where $x \to p(x)$ is given weight function, are also of concern.

One simple way for construction of quadrature formulas is founded on application of interpolation. Formulas obtained in this way are called interpolating quadrature formulas.

Let the values of function f in given points $x_0, x_1, \ldots, x_n (\in [a,b])$ be $f_0, f_1, \ldots, f_n$ respectively, i.e.

$$f_k \equiv f(x_k) \quad (k = 0, 1, \ldots, n).$$

On the basis of these data, we can construct Lagrange interpolation polynomial

$$P_n(x) = \sum_{k=0}^{n} f_k \frac{\omega(x)}{(x - x_k)\omega'(x_k)},$$

where $\omega(x) = (x - x_0)(x - x_1)\cdots(x - x_n)$.

Then

$$\int_a^b p(x)f(x)dx = \int_a^b p(x)P_n(x)dx + R_{n+1}(f),$$

i.e.

$$\text{(8.2.1.3)} \qquad \int_a^b p(x)f(x)dx = \sum_{k=0}^{n} A_k f_k + R_{n+1}(f),$$

where we put

$$A_k = \int_a^b \frac{p(x)\omega(x)}{(x - x_k)\omega'(x_k)}dx \qquad (k = 0, 1, \ldots, n).$$

In formula (8.2.1.3), $R_{n+1}(f)$ is called remainder term, residue (rest, residuum) of quadrature formula and represents error done by replacing of integral by finite sum. Index $n+1$ in remainder term denotes that integral is approximate calculated based on values of function to be integrated in $n+1$ points.

Denote with π_n set of all polynomials of degree not greater than n.

Because $f(x) = x^k$ $(k = 0, 1, \ldots, n)$, $f(x) \equiv P_n(x)$, we have $R_{n+1}(x^k) \equiv 0$ $(k = 0, 1, \ldots, n)$, wherefrom we conclude that formula (8.2.1.3) is exact for every $f \in \pi_n$, regardless of choice of interpolation nodes x_k $(k = 0, 1, \ldots, n)$ and in this case we say that (8.2.1.3) has algebraic degree of accuracy n.

8.2.2. Newton-Cotes formulas

In this section we will develop quadrature formulas od closed type in which the interpolation nodes $x_k = x_0 + kh$ $(k = 0, 1, \ldots, n)$ are taken equidistantly with a step $h = \dfrac{b-a}{n}$.

If we introduce substitution $x - x_0 = ph$, we have

$$\text{(8.2.2.1)} \qquad \omega(x) = (x - x_0)(x - x_1)\ldots(x - x_n) = h^{n+1}p(p-1)\ldots(p-n)$$

and

$$\text{(8.2.2.2)} \qquad \begin{aligned}\omega'(x_k) &= (x_k - x_0)(x_k - x_1)\ldots(x_k - x_{k-1})(x_k - x_{k+1})\ldots(x_k - x_n)\\ &= h^n(-1)^{n-k}k!(n-k)!\end{aligned}$$

By introducing notation for generalized degree $x^{(s)} = x(x-1)\ldots(x-s+1)$, based on (8.2.2.1), (8.2.2.2) and results from previous section, we get

$$A_k = \int_0^n \frac{(-1)^{n-k}p^{(n+1)}h}{(p-k)k!(n-k)!}dp \quad (k = 0, 1, \ldots, n),$$

i.e.

$$A_k = (b-a)H_k \quad (k = 0, 1, \ldots, n),$$

where we put

$$(8.2.2.3) \qquad H_k \equiv H_k(n) = \frac{(-1)^{n-k}}{n!n}\binom{n}{k}\int_0^n \frac{p^{(n+1)}}{p-k}dp \quad (k = 0, 1, \ldots, n).$$

Coefficients H_k are known in literature as Newton-Cotes coefficients, and corresponding formulas

$$(8.2.2.4) \qquad \int\limits_{x_0=a}^{x_n=b} f(x)dx = (b-a)\sum_{k=0}^{n} H_k f(a + k\frac{b-a}{n}) \quad (k \in N)$$

as Newton-Cotes formulas.

Further on we will give survey of Newton-Cotes formulas for $n \leq 8$. Here we use denotations $h = \dfrac{b-a}{n}$, $f_k = f(x_k)$ $(k = 0, 1, \ldots, n)$.

1. $n = 1$ (Trapezoid rule)

$$\int\limits_{x_0}^{x_1} f(x)dx = \frac{h}{2}(f_0 + f_1) - \frac{h^3}{12}f''(\xi_1);$$

2. $n = 2$ (Simpson's rule)

$$\int\limits_{x_0}^{x_2} f(x)dx = \frac{h}{3}(f_0 + 4f_1 + f_2) - \frac{h^5}{90}f^{IV}(\xi_2);$$

3. $n = 3$ (Simpson's rule $\frac{3}{8}$)

$$\int\limits_{x_0}^{x_3} f(x)dx = \frac{3h}{8}(f_0 + 3f_1 + 3f_2 + f_3) - \frac{3h^5}{80}f^{IV}(\xi_3);$$

4. $n = 4$ (Boole's rule)

$$\int\limits_{x_0}^{x_4} f(x)dx = \frac{2h}{45}(7f_0 + 32f_1 + 12f_2 + 32f_3 + 7f_4) - \frac{8h^7}{945}f^{VI}(\xi_4);$$

5. $n = 5$

$$\int\limits_{x_0}^{x_5} f(x)dx = \frac{5h}{288}(19f_0 + 75f_1 + 50f_2 + 50f_3 + 75f_4 + 19f_5) - \frac{275h^7}{12096}f^{(6)}(\xi_5);$$

6. $n = 6$

$$\int\limits_{x_0}^{x_6} f(x)dx = \frac{h}{140}(41f_0 + 216f_1 + 27f_2 + 272f_3 + 27f_4$$

$$+ 216f_5 + 41f_6) - \frac{9h^9}{1400}f^8(\xi_6);$$

7. $n = 7$

$$\int\limits_{x_0}^{x_7} f(x)dx = \frac{7h}{17280}(751f_0 + 3577f_1 + 1323f_2 + 2989f_3 + 2989f_4$$
$$+ 1323f_5 + 3577f_6 + 751f_7) - \frac{8183h^9}{518400} f^{(8)}(\xi_7);$$

8. $n = 8$

$$\int\limits_{x_0}^{x_8} f(x)dx = \frac{4h}{14175}(989f_0 + 5888f_1 - 928f_2 + 10496f_3 - 4540f_4$$
$$+ 10496f_5 - 928f_6 + 5888f_7 + 989f_8) - \frac{2368h^{11}}{467775} f^{(10)}(\xi_8);$$

where $\xi_k \in (x_0, x_k)$ $(k = 1, \ldots, 8)$.
In general case, the residue $R_{n+1}(f)$ is of form

$$R_{n+1}(f) = C_n h^m f^{(m-1)}(\xi_n) \quad (x_0 < \xi_n < x_n),$$

where $m = 2[\frac{n}{2}] + 3$. Given equality has a meaning if function $f \in C^{m-1}[a, b]$.

8.2.3. Generalized quadrature formulas

In order to compute value of integrals more accurate it is necessary to divide segment $[a, b]$ to the set of subsegments, and then to apply to each of them some quadrature formula. In this way we get generalized or composite formulas. In this section we will consider generalized formulas obtained on the basis of trapezoid or Simpson's formula.

Divide the segment $[a, b]$ on set of subsegments $[x_{i-1}, x_i]$ so that $x_i = a + ih$ and $h = (b - a)/n$ (see Fig. 8.2.3.1)).

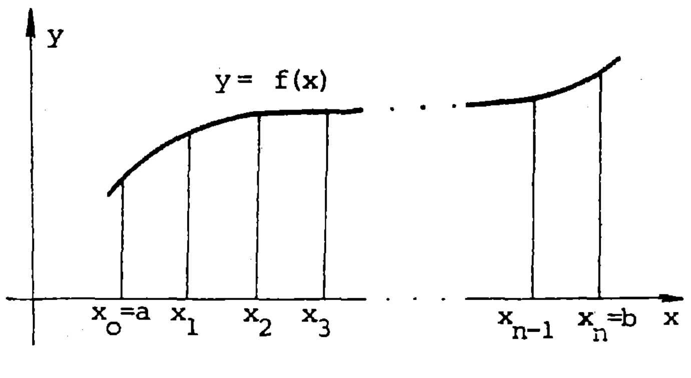


Figure 8.2.3.1

By applying the trapezoid formula on every subsegment, we get

$$\int\limits_a^b f(x)dx = \sum_{i=1}^{n} \int\limits_{x_{i-1}}^{x_i} f(x)dx = \sum_{i=1}^{n} (\frac{h}{2}(f_{i-1} + f_i) - \frac{h^3}{12} f''(\xi_i)),$$

i.e.

$$\int\limits_a^b f(x)dx = T_n - \frac{h^3}{12} \sum_{i=1}^{n} f''(\xi_i),$$

where

$$T_n \equiv T_n(f;h) = h(\frac{1}{2}f_0 + f_1 + \cdots + f_{n-1} + \frac{1}{2}f_n)$$

and

$$x_{i-1} < \xi_i < x_i \quad (i = 1, 2, \ldots, n).$$

Theorem 8.2.3.1. *If $f \in C^2[a,b]$ the equality*

$$\int_a^b f(x)dx - T_n = -\frac{(b-a)^2}{12n^2}f''(\xi) \quad (a < \xi < b)$$

holds.

Quadrature formula

$$\int_a^b f(x)dx \cong T_n(f;h) \quad (h = \frac{b-a}{n})$$

is called generalized trapezoid formula.

Suppose now that $h = \dfrac{b-a}{2n}$, i.e. $x_i = a + ih$ $(i = 0, 1, \ldots, 2n)$ (See Fig. 8.2.3.2), and then apply Simpson's rule to subsegments

$$[x_0, x_2], \ldots, [x_{2n-2}, x_{2n}].$$

In this way we get generalized Simpson's formula

$$\int_a^b f(x)dx \cong S_n(f;h) \quad (h = \frac{b-a}{2n}),$$

where

$$S_n \equiv S_n(f;h) = \frac{h}{3}\{f_0 + 4(f_1 + \ldots + f_{2n-1}) + 2(f_2 + \ldots + f_{2n-2}) + f_{2n}\}.$$

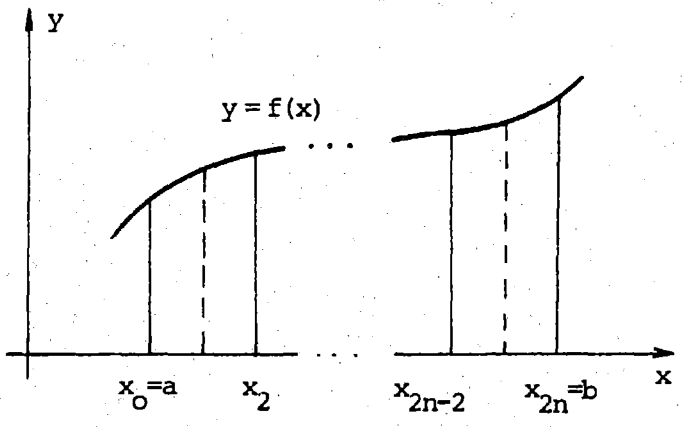


Figure 8.2.3.2

Theorem 8.2.3.2. *If $f \in C^4[a,b]$ the equality*

$$\int_a^b f(x)dx - S_n = -\frac{(b-a)^2}{2880n^4} f^{(IV)}(\xi) \quad (a < \xi < b)$$

holds.

8.2.4. Romberg integration

For calculation of definite integrals in practice is most frequently used generalized trapezoid formula in a special form, known as Romberg integration.

Denote with $T_k^{(0)}$ trapezoid approximation $T_n(f;h)$ $(n = 2^k)$, i.e. $h = \frac{(b-a)}{2^k})$. Romberg integration consists of construction of two-dimensional set $T_k^{(m)}$ $(m = 0,1,\ldots,k;\ k = 0,1,\ldots)$ using

$$T_k^{(m)} = \frac{4^m T_{k+1}^{(m-1)} - T_k^{(m-1)}}{4^m - 1}. \tag{8.2.4.1}$$

Using (8.2.4.1) one can construct so known T table

$$\begin{tikzcd}
T_0^{(0)} \arrow[r] & T_0^{(1)} \arrow[r] & T_0^{(2)} \arrow[r] & T_0^{(3)} \\
T_1^{(0)} \arrow[r] \arrow[ur] & T_1^{(1)} \arrow[r] \arrow[ur] & T_1^{(2)} \arrow[r] \arrow[ur] & \vdots \\
T_2^{(0)} \arrow[r] \arrow[ur] & T_2^{(1)} \arrow[r] \arrow[ur] & \vdots & \\
T_3^{(0)} \arrow[r] \arrow[ur] & \vdots & &
\end{tikzcd}$$

by taking $k = 0,1,\ldots$ and $m = 1,2,\ldots$. In first column of this table are in turn approximate values of integral I obtained by means of trapezoid formula with $h_k = (b-a)/2^k$ $(k = 0,1,\ldots)$. Second column is obtained based on the first, using formula (8.2.4.1), third from second, and so on.

Iterative process, defined by (8.2.4.1) is the standard Romberg method for numerical integration. One can prove that series $\{T_k^{(m)}\}_{k\in N_0}$ and $\{T_k^{(m)}\}_{m\in N_0}$ (by columns and rows in T-table) converge to I. At practical application of Romberg integration, iterative process (8.2.4.1) is usually interrupted when $|T_0^{(m)} - T_0^{(m-1)}| \le \varepsilon$, where ε is in advance allowed error, and then as result is taken $I \cong T_0^{(m)}$.

8.2.5. Program realization

In this section we give program realization of Simpson's and Romberg integration.

Program 8.2.5.1.

For integration using generalized Simpson's formula the subroutine `INTEG` is written. Parameters in parameter list are of meaning explained in C- comments of subprogram source code. Function to be integrated is given in subroutine `FUN`, and depends on one parameter Z. By integer parameter J is provided simultaneous specifying more functions to integrate.

Subroutine INTEG is organized in this way that initial number of subsegments can be improved (by reduction of step h to $h/2$) up to MAX=1000. In case when relative difference in integrals values, obtained by steps h and $h/2$, is less than 10^{-5}, the calculation interrupts and value of integral calculated with the smallest step is taken as definitive value of integral. If this criterion cannot be fulfilled with less than MAX subsegments, the message KBR=1 is printed (and in opposite case KBR=0).

As a test examples for this subroutine, the following integrals are taken:

$$\int_0^1 \frac{e^{zx}}{x^2+z^2}dx \qquad (z = 1.0(0.1)1.5),$$

$$\int_0^{1/2} \pi \sin(\pi z x)\, dx \qquad (z = 1.0(0.2)1.4)$$

$$\int_1^2 \frac{\log(x+z)}{z^2+e^x}\frac{\sin x}{x}dx \qquad (z = 0.0(0.1)0.5).$$

Subroutines, main program, and output listing are of form:

```
C=========================================================
C COMPUTATION OF DEFINITE INTEGRAL OF FUNCTION F(X,Z,J)
C BY SIMPSON'S RULE
C=========================================================
      SUBROUTINE INTEG(A, B, S, F, J, KBR, Z)
C  A - LOWER LIMIT OF INTEGRAL
C  B - UPPER LIMIT OF INTEGRAL
C  S - VALUE OF INTEGRAL WITH ACCURACY  EPS=1.E-5
C  KBR - CONTROL NUMBER
C  KBR=0 INTEGRAL CORRECTLY COMPUTED
C  KBR=1 INTEGRAL NOT COMPUTED WITH SPECIFIED ACCURACY
C  Z - PARAMETAR OF INTEGRATED FUNCTION
C  INITIAL NUMBER OF SEGMENTS IS 2*MP MAXIMAL IS MAX=1000
      MP=15
      MAX=1000
      KBR=0
      N=2.*MP
      S0=0.
      SAB=F(A,Z,J)+F(B,Z,J)
      H=(B-A)/FLOAT(N)
      X=A
      S1=0.
      N2=N-2
      DO 5 I=2, N2, 2
      X=X+2.*H
5     S1=S1+F(X,Z,J)
10    S2=0.
      X=A-H
      N1=N-1
      DO 15 I=1, N1, 2
      X=X+2.*H
15    S2=S2+F(X,Z,J)
      S=H/3.*(SAB+2.*S1+4.*S2)
      REL=(S-S0)/S
      IF (ABS(REL)-1.E-5) 35,35,20
20    IF (N-MAX) 25,25,30
```

```
25      N=2*N
        H=0.5*H
C  NUMBER OF INTERVALS IS DOUBLED AND
C  NEW VALUE FOR S1 IS COMPUTED
        S1=S1+S2
        S0=S
        GO TO 10
30      KBR=1
35      RETURN
        END
        FUNCTION FUN(X,Z,J)
        GO TO (10,20,30),J
10      FUN=EXP(Z*X)/(X*X+Z*Z)
        RETURN
20      PI=3.1415926535
        FUN=PI*SIN(PI*X*Z)
        RETURN
30      FUN=ALOG(X+Z)/(Z*Z+EXP(X))*SIN (X)/X
        RETURN
        END
        EXTERNAL FUN
        OPEN(8,File='Simpson.IN')
        OPEN(6,File='Simpson.out')
        WRITE(6,5)
5       FORMAT (1H1,2X, 'IZRACUNAVANJE VREDNOSTI INTEGRALA',
     1 ' PRIMENOM SIMPSONOVE FORMULE ' //14X,
     2 'TACNOST IZRACUNAVANJA EPS=1.E-5'
     3 ///11X,'J',4X,'DONJA',5X,'GORNJA',3X,'PARAMETAR',
     4 3X,' VREDNOST'/ 16X, 'GRANICA', 3X,'GRANICA',
     5  5X,'Z',7X,'INTEGRALA'//)
        DO 40 J=1,3
        READ (8,15) DG, GG, ZP, DZ, ZK
15      FORMAT(5F5.1)
        Z=ZP-DZ
18      Z=Z+DZ
        IF (Z.GT.ZK+0.000001) GO TO 40
        CALL INTEG (DG,GG,S,FUN,J,KBR,Z)
        IF(KBR) 20,25,20
20      WRITE (6,30)
30      FORMAT (/11X, 'INTEGRAL NIJE KOREKTNO IZRACUNAT'/)
        GO TO 18
25      WRITE (6,35) J,DG,GG,Z,S
35      FORMAT (11X,I1,F8.1,2F10.1,F15.6/)
        GO TO 18
40      CONTINUE
        STOP
        END
```

```
0.,1.,1.,0.1,1.5
0.,0.5,1.,0.2,1.4
1.,2.,0.,0.1,0.5
```

```
IZRACUNAVANJE VREDNOSTI INTEGRALA PRIMENOM SIMPSONOVE FORMULE
              TACNOST IZRACUNAVANJA EPS=1.E-5
           J    DONJA     GORNJA   PARAMETAR    VREDNOST
                GRANICA   GRANICA      Z        INTEGRALA
           1      .0        1.0       1.0        1.270724
```

```
        1       .0        1.0        1.1        1.153890
        1       .0        1.0        1.2        1.059770
        1       .0        1.0        1.3         .983069
        1       .0        1.0        1.4         .920013
        1       .0        1.0        1.5         .867848
        2       .0         .5        1.0        1.000000
        2       .0         .5        1.2        1.090848
        2       .0         .5        1.4        1.134133
        3      1.0        2.0         .0         .048047
        3      1.0        2.0         .1         .059595
        3      1.0        2.0         .2         .069940
        3      1.0        2.0         .3         .079052
        3      1.0        2.0         .4         .086920
        3      1.0        2.0         .5         .093558
```

Program 8.2.5.2.

Now we give program realization of Romberg integration in double arithmetic computer precision DOUBLE PRECISION. List in subroutine is of following meaning:
DG - lower limit of integral;
GG - upper limit of integral;
FUN - name of function subroutine which defines function to be integrated;
EPS - demanded accuracy of computation;
VINT - value of integral for given accuracy EPS, if KB=0;
KB - control number (KB=0 - integral correctly computed; KB=1 - accuracy of computing not reached after 15 proposed steps, i.e. with numbers of subsegments 2^{15}). For testing of this subroutine is taken tabulating of function

$$F(x) = \int_0^x e^{-t^2} dt \qquad (x = 0.1(0.1)1.0),$$

with accuracy 10^{-5}. Routines codes and output listings are of form:

```
C==================================================================
C                 ROMBERGOVA INTEGRACIJA
C==================================================================
      DOUBLE PRECISION GG, FUN, VINT
      EXTERNAL FUN
      open(6,file='romberg.out')
EPS=1.E-8
      WRITE (6,11)
11    FORMAT(1H0,5X,'X',7X,'INTEGRAL(0.,X)'/)
      DO 10 I=1, 10
      GG=0.1*I
      CALL ROMBI(0.DO,GG,FUN,EPS,VINT,KB)
      IF (KB) 5,15,5
  5   WRITE (6,20) GG
20    FORMAT (5X,F3.1,4X,'TACNOST NE ZADOVOLJAVA'//)
      GO TO 10
15    WRITE(6,25)GG,VINT
25    FORMAT(5X,F3.1,4X,F14.9)
10    CONTINUE
      STOP
      END
      SUBROUTINE ROMBI (DG,GG,FUN,EPS,VINT,KB)
      DOUBLE PRECISION FUN,VINT,T(15),DG,GG,H,A,POM,B,X
      KB=0
      H=GG-DG
```

```
          A=(FUN(DG)+FUN(GG))/2.
          POM=H*A
          DO 50 K=1, 15
          X=DG+H/2.
10        A=A+FUN (X)
          X=X+H
          IF (X.LT.GG) GO TO 10
          T(K)=H/2.*A
          B=1.
          IF (K.EQ.1) GO TO 20
          K1=K-1
          DO 15 M=1, K1
          I=K-M
          B=4.*B
15        T(I)=(B*T(I+1)-T(I))/(B-1.)
20        B=4.*B
          VINT=(B*T(1)-POM)/(B-1.)
          IF(DABS(VINT-POM).LE.EPS) RETURN
          POM=VINT
50        H=H/2.
          KB=1
          RETURN
          END
          FUNCTION FUN(X)
          DOUBLE PRECISION FUN,X
          FUN=DEXP(-X*X)
          RETURN
          END
```

```
0         X          INTEGRAL(0.,X)
          .1            .099667666
          .2            .197365034
          .3            .291237887
          .4            .379652845
          .5            .461281012
          .6            .535153533
          .7            .600685674
          .8            .657669863
          .9            .706241521
         1.0            .746824138
```

8.2.6. On numerical computation of one class of double integrals

In this section we will point out to one way for approximate calculation of double integrals of form

$$\iint_G f(x,y)\, dxdy, \tag{8.2.6.1}$$

where area of integration is unit circle, i.e. $G = \{(x,y)\,|\,x^2+y^2 \le 1\}$. Namely, for numerical computation of the integral (8.2.6.1) in literature is known formula

$$\iint_G f(x,y)\, dxdy \cong \frac{\pi}{8}(2f(0) + \sum_{i=1}^{n} f(M_i)), \tag{8.2.6.2}$$

where O is origin, i.e. $0 = (0,0)$, and points M_i have polar coordinates

$$r_i = \sqrt{\frac{2}{3}}, \quad \Phi_i = \frac{\pi}{3}(i-1) \quad (i = 1, 2, \ldots, 6).$$

According to formula (8.2.6.2) we will realize program for computation of double integrals, with unit circle as area of integration. Program organization will be such that by function subroutine EF can be defined several different functions f to be integrated. Parameters in list of parameters are of following meaning:

X - value of argument x;
Y - value of argument y;
K - integer that defines different functions to be integrated.

Formula (8.2.6.2) is realized by subroutine DVINT, which parameters in list are of following meaning:

EF - name of function subroutine;
K - integer with same meaning like in subroutine EF;
VRINT - computed value of integral, obtained by using formula (8.2.6.2).

```
      SUBROUTINE DVINT(EF, K,VRINT)
      PI=3.1415926535
      RO=SQRT(2./3)
      PI3=PI/3
      FI=-PI3
      VRINT=2.*EF(0.,0.,K)
      DO 10 I=1,6
      FI=FI+PI3
      X=RO*COS(FI)
      Y=RO*SIN(FI)
10    VRINT=VRINT+EF(X,Y,K)
      VRINT=PI/8.*VRINT
      RETURN
      END
```

Main program is of form:

```
C==================================================
C      IZRACUNAVANJE DVOSTRUKOG INTEGRALA
C==================================================
      EXTERNAL EF
      OPEN(6,FILE='DVINT.OUT')
      WRITE (6,5)
5     FORMAT (1H1//10X,'IZRACUNAVANJE DVOSTRUKOG',
     1' INTEGRALA'//)
      DO 10 K=1,3
      CALL DVINT(EF, K, VRINT)
10    WRITE (6,15)K,VRINT
15    FORMAT (15X,I1,' PRIMER'// 10X,
     1 'VREDNOST INTEGRALA =',F12.6//)
      STOP
      END
```

By using this program we calculated approximately values of the following integrals:

$$1^0 \quad \iint\limits_G \frac{16x^2y^2}{1+x^2+y^2}\, dxdy;$$

$$2^0 \quad \iint\limits_G \sqrt{1+(1+x)^2+y^2}\, dxdy;$$

$$3^0 \quad \iint\limits_G \frac{24x^2}{\sqrt{2-x^2-y^2}}\, dxdy.$$

Function subroutine EF and output listing are of form:

```
      FUNCTION EF(X,Y,K)
GO TO (10,20,30),K
10    EF=(16.*X*X*Y*Y)/(1.+X*X+Y*Y)
      RETURN
20    EF=SQRT(1.+Y*Y+(1.+X)**2)
      RETURN
30    EF=(24.*X*X)/SQRT(2.-X*X-Y*Y)
      RETURN
      END
```

```
1
           IZRACUNAVANJE DVOSTRUKOG INTEGRALA
                1 PRIMER
           VREDNOST INTEGRALA =    1.256637
                2 PRIMER
           VREDNOST INTEGRALA =    4.858376
                3 PRIMER
           VREDNOST INTEGRALA =   16.324200
```

8.2.7. Packages for Numerical Integration

Numerical integration of both discrete data and known functions are needed in engineering practice. The procedures for first case are based on fitting approximating polynomials to the data and integrating the approximating polynomials. The direct fit polynomial method works well for both equally spaced data and non-equally spaced data. Least squares fit polynomials can be used for large sets of data or sets of rough data. The Newton-Cotes formulas, which are based on Newton forward-difference polynomials, give simple integration formulas for equally spaced data. Romberg integration, which is extrapolation of the trapezoid rule is of important practical use. An example of multiple integration is presented as illustrative case.

Of presented simple methods it is likely that Romberg integration is most efficient. Simpson's rules are elegant, but the first extrapolation of Romberg integration gives comparable results. Subsequent extrapolation of Romberg integration increase the order at a very satisfactory rate. Simpson's rules could be developed into an extrapolation procedure, but with no advantage over Romberg integration.

Many commercial software packages contain solvers for numerical integration. Some of the more prominent systems are `Matlab` and `Mathcad`. More sophisticated systems, such as `Mathematica`, `Macsyma` (VAX UNIX MACSYMA, Reference Manual, Symbolics Inc., Cambridge, MA), and `Maple` (MAPLE V Library Reference Manual, Springer, NY, 1991) also contain numerical integration solvers.

Some organizations have own packages - collection of high-quality routines, like `ACM` (Collected algorithms), `IMSL` (Houston, TX), `NAG` (Numerical Algorithms Group, Downers Grove, IL), and some famous individual packages are `QUADPACK` (R. Piessens, et all.), *QUADPACK, A Subroutine Package for Automatic Integration*, Springer, Berlin, 1983), `CUBTRI` (Cubature Formulae Over Triangle), `SSP` (IBM Numerical Software).

The book *Numerical Recipes* ([4], Chap. 4) contains several subroutines for integration of functions. Some algorithms, from which some are codded, are given in book *Numerical Methods for Engineers and Scientists* ([3], Chap. 6).

On the end, in order to give some hints for software own development or usage of software packages, we will give standard test examples for testing or benchmarking.

Standard test examples (Indefinite integrals):

$1^0 \int \sin x\, dx$; $2^0 \int \sqrt{\tan x}\, dx$; $3^0 \int \frac{x}{x^3-1}\, dx$; $4^0 \int \frac{x}{\sin^2 x}\, dx$; $5^0 \int \frac{\log x}{\sqrt{x+1}}\, dx$;
$6^0 \int \frac{x}{\sqrt{1+x}+\sqrt{1-x}}\, dx$; $7^0 \int e^{-ax^2}\, dx$; $8^0 \int \frac{x}{\log^3 x}\, dx$; $9^0 \int \frac{\sin x}{x^2}\, dx$;
$10^0 \int \frac{1}{2+\cos x}\, dx$;

Standard test examples (Definite integrals):

$1^0 \int_0^{4\pi} \frac{1}{2+\cos x}\, dx$; $2^0 \int_{-\infty}^{\infty} \frac{\sin x}{x}\, dx$; $3^0 \int_0^{\infty} \frac{e^{-x}}{\sqrt{x}}\, dx$; $4^0 \int_0^{\infty} \frac{x^2 e^{-x}}{1-e^{-2x}}\, dx$;
$5^0 \int_0^{\infty} e^{-x^2} \log^2 x\, dx$; $6^0 \int_1^{\infty} e^{-x} x^3 \log^2 x\, dx$; $7^0 \int_0^{\infty} \frac{x^2}{1+x^3}\, dx$; $8^0 \int_{-1}^{1} \frac{1}{x^2}\, dx$;
$9^0 \int_1^{\infty} e^{-x} x^{11/3}\, dx$;

Bibliography (Cited references and further reading)

[1] Milovanović, G.V., *Numerical Analysis II*, Naučna knjiga, Beograd, 1988 (Serbian).
[2] Milovanović, G.V. and Djordjević, Dj.R., *Programiranje numeričkih metoda na FORTRAN jeziku.* Institut za dokumentaciju zaštite na radu "Edvard Kardelj", Niš, 1981 (Serbian).
[3] Hoffman, J.D., *Numerical Methods for Engineers and Scientists.* Taylor & Francis, Boca Raton-London-New York-Singapore, 2001.
[4] Press, W.H., Flannery, B.P., Teukolsky, S.A., and Vetterling, W.T., *Numerical Recepies - The Art of Scientific Computing.* Cambridge University Press, 1989.
[5] Kronrod, A. S., *Nodes and Weights of Quadrature Formulas.* Consultants Bureau, New York, 1965.
[6] Clenshaw, C.W. and Curtis, A.R., *A method for numerical integration on an automatic computer* Num. Math. 2(1960), pp. 197–205.
[7] Engels, H., *Numerical Quadrature and Cubature.* Academic Press, London, 1980.
[8] Abramowitz, M., *On the practical evaluation of integrals.* SIAM J. Appl. Math. 2(1954)20-35.
[9] Davis, P.J., and P. Rabinowitz, P., *Methods of Numerical Integration.* Academic Press, New York, 1975.
[10] Krylov, V.I., *Approximate Calculation of Integrals.* MacMillan, New York, 1962 (Russian, transl. A.H. Stroud).
[11] Stroud, A.H., *Approximative Calculation of Multiple Integrals.* Prentice-Hall, Englewood Cliffs, N.J. 1971.
[12] Mysovskih, I.P., *Interpolyacionnye Kubaturnye Formuly.* Nauka, Moskva, 1981.
[13] Stroud, A.H., *Gaussian Quadrature Formulas.* Prentice Hall, Englewood Cliffs, N.J., 1966.
[14] Ralston,A., *A First Course in Numerical Analysis.* McGraw-Hill, New York, 1965.

[15] Hildebrand, F.B., *Introduction to Numerical Analysis.* Mc.Graw-Hill, New York, 1974.

[16] Acton, F.S., *Numerical Methods That Work* (corrected edition). Mathematical Association of America, Washington, D.C., 1990.

[17] Abramowitz, M., and Stegun, I.A., *Handbook of Mathematical Functions.* National Bureau of Standards, Applied Mathematics Series, Washington, 1964 (reprinted 1968 by Dover Publications, New York).

[18] Rice, J.R., *Numerical Methods, Software, and Analysis.* McGraw-Hill, New York, 1983.

[19] Forsythe, G.E., Malcolm, M.A., and Moler, C.B., *Computer Methods for Mathematical Computations.* Englewood Cliffs, Prentice-Hall, NJ, 1977.

[20] Kahaner, D., Moler, C., and Nash, S., 1989, *Numerical Methods and Software.* Englewood Cliffs, Prentice Hall, NJ, 1989.

[21] Hamming, R.W., *Numerical Methods for Engineers and Scientists.* Dover, New York, 1962 (reprinted 1986).

[22] Ferziger, J.H., *Numerical Methods for Engineering Applications.* Stanford University, John Willey & Sons, Inc., New York, 1998.

[23] Pearson, C.E., *Numerical Methods in Engineering and Science.* University of Washington, Van Nostrand Reinhold Company, New York, 1986.

[24] Stephenson, G. and Radmore, P.M., *Advanced Mathematical Methods for Engineering and Science Students.* Imperial College London, University College, London Cambridge Univ. Press, 1999.

[25] Milovanović, G.V. and Kovačević, M.A., *Zbirka rešenih zadataka iz numeričke analize.* Naučna kujiga, Beograd, 1985. (Serbian).

[26] *IMSL Math/Library Users Manual* , IMSL Inc., 2500 City West Boulevard, Houston TX 77042.

[27] *NAG Fortran Library*, Numerical Algorithms Group, 256 Banbury Road, Oxford OX27DE, U.K.

[28] Piessens, R., de Doncker-Kapenga, E., Überhuber, C.W., Kahaner, D.K., *QUADPACK, A Subroutine Package for Automatic Integration.* Springer-Verlag, Berlin, 1983.

[29] Laurie, D. P., *Algorithm 584 CUBTRI: Automatic cubature over a triangle.* ACM Trans. Math. Software 8(1982), 210.-218.

[30] Stoer, J., and Bulirsch, R., *Introduction to Numerical Analysis.* Springer-Verlag, New York, 1980.

[31] Johnson, L.W., and Riess, R.D., *Numerical Analysis, 2nd ed.* Addison- Wesley, Reading, MA, 1982.

[32] Ralston, A., and Rabinowitz, P., *A First Course in Numerical Analysis, 2nd ed.* McGraw-Hill, New York, 1978.

[33] Isaacson, E., and Keller, H.B., *Analysis of Numerical Methods.* Wiley, New York, 1966.

LECTURES

LESSON IX

9. Ordinary Differential Equations

9.1. Introduction

Problems involving ordinary differential equations (ODEs) can always be reduced to the set of first-order differential equations. For example the second order equation

$$\frac{d^2y}{dx^2} + q(x)\frac{dy}{dx} = r(x) \tag{9.1.1}$$

can be rewritten as two first-order equations

$$\begin{aligned} \frac{dy}{dx} &= z(x) \\ \frac{dz}{dx} &= r(x) - q(x)z(x), \end{aligned} \tag{9.1.2}$$

where z is a new variable. This exemplifies the procedure for an arbitrary ODE. The usual choice for the new variables is to let them be just derivatives if each other, and, of course, of original variable. Occasionally, it is useful to incorporate into their definition some other factors in the equation, or some powers of the independent variable, for the purpose of the mitigating singular behavior that could result in overflows or increased roundoff error. Thus, involving new variables should be carefully chosen. The possibility of a formal reduction of a differential equation system to an equivalent set of first-order equations means that computer programs for the solution of differential equation sets can be directed toward the general form

$$\frac{dy_i(x)}{dx} = f_i(x, y_1, \cdots, y_n) \quad (i = 1, \ldots, n), \tag{9.1.3}$$

where the f_i functions are known and $y_1, y_2, \ldots, y_n$ are dependent variables.

A problem involving ODEs is not completely specified by its equations. Even more crucial in determining how to start solving problem numerically is the nature of the problem's boundary conditions. Boundary conditions are algebraic conditions on the values of the functions y_i in (9.1.3). Generally, they can be satisfied at discrete specified points, but do not hold between those points, i.e. are not preserved automatically by the differential equations. Boundary conditions can be as simple as requiring that certain variables have certain numerical values, or as complicated as a set of nonlinear algebraic equations among the variables. Usually, it is the nature of the boundary conditions that determines which numerical methods will be applied. Boundary conditions divide into two broad categories.

- **Initial value problems**, where all the y_i are given at some starting value x_s, and it is desired to find the the y_i's at some final point x_f, or at some discrete list of points (for example, to generate a table of results).
- **Two-point boundary value problems**, where the boundary conditions are specified at more than one x. Usually some conditions are specified at x_s and the remainder at x_f.

In considering methods for numerical solution of Cauchy problem for differential equations of first order, we will note two general classes of those methods:

a) Linear multi-step methods,
b) Runge-Kutta methods.

The first class of methods has a property of linearity, in contrary to Runge-Kutta methods, where the increasing of method order is realized by involving nonlinearity. The common "predecessor" of both classes is Euler's method, which belongs to both classes.

In newer times there appeared a whole series of methods, so known hybrid methods, which use good characteristics of mentioned basic classes of methods.

9.2. Euler's method

Euler's method is the simplest numerical method for solving Cauchy's problem

$$y' = f(x, y), \quad y(x_o) = y_0 \tag{9.2.1}$$

and is based on approximative equality

$$y(x) = y(x_0) + (x - x_0)y'(x_0),$$

i.e.

$$y(x) = y(x_0) + (x - x_0)f(x_0, y_0), \tag{9.2.2}$$

in regard to (9.2.1). If we denote with y_1 the approximate value for $y(x_1)$, based on (9.2.2) we have

$$y_1 = y_0 + (x_1 - x_0)f(x_0, y_0).$$

In general case, for arbitrary set of points $x_0 < x_1 < x_2 < \ldots$, the approximate values for $y(x_n)$, denoted as y_n, can be determined using

$$y_{n+1} = y_n + (x_{n+1} - x_n)f(x_n, y_n) \quad (n = 0, 1, \ldots). \tag{9.2.3}$$

The last formula defines Euler's method, which geometric interpretation is given in the Fig. 9.2.1.

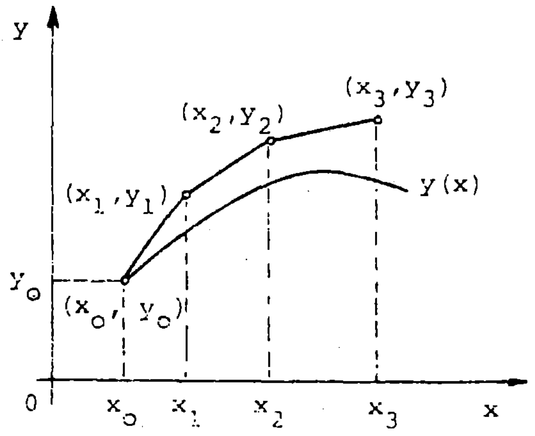


Figure 9.2.1

Polygonal line $(x_0, y_0) - (x_1, y_1) - (x_2, y_2) - \ldots$ is known as Euler's polygon.
The points x_n are usually chosen equidistantly, i.e. $x_{n+1} - x_n = h = const.(> 0)$ $(n = 0, 1, \ldots)$ in which case (9.2.3) reduces to

$$y_{n+1} = y_n + hf(x_n, y_n) \quad (n = 0, 1, \ldots).$$

9.3. General linear multi-step method

In this and following sections a general method for solving Cauchy problem

$$y' = f(x, y), \quad y(x_0) = y_0 \quad (x_0 \le x \le b). \tag{9.3.1}$$

will be considered. If we divide the segment $[x_0, b]$ to N subsegments of length $h = \frac{b - x_0}{N}$, we get a string of points x_n determined with

$$x_n = x_0 + nh \quad (n = 0, 1, \ldots, N).$$

Let y_n denote sequence of approximate values of solutions of problem (9.3.1) in points x_n and let $f_n \equiv f(x_n, y_n)$. It is our task to determine a set y_n. In order to solve this problem a number of methods have been developed. One of them is Euler's method, which has been considered in previous section. At Euler's method series y_n is computed recursively using

$$y_{n+1} - y_n = hf_n \quad (n = 0, 1, \ldots), \tag{9.3.2}$$

whereby the linear relation among y_n, y_{n+1} and f_n exists. In general case, for evaluation of series more complicated recurrence relations than (9.3.2) can be used. Among the methods originated from these relations, important role have the methods with linear relation between y_{n+i}, f_{n+i} $(i = 0, 1, \ldots k)$ and they form the class of linear multi-step methods.

General linear multi-step method can be represented in form

$$\sum_{i=0}^{k} \alpha_i y_{n+1} = h \sum_{i=0}^{k} \beta f_{n+i} \quad (n = 0, 1, \ldots), \tag{9.3.3}$$

where α and β are constant coefficients determined by accuracy up to multiplicative constant. In order to obtain their uniqueness we will take $\alpha_k = 1$.

If $\beta_k = 0$, we say that method (9.3.3) is of open type or that is explicit; in counterpart we say that it is of closed type or implicit.

In general case (9.3.3) represents nonlinear difference equation, because of $f_{n+i} \equiv f(x_{n+i}, y_{n+i})$.

For determination of series y_n using method (9.3.3) it is necessary to know initial values y_i $(i = 0, 1, \ldots, k - 1)$. Knowing in advance only value y_0, a particular problem in application of multi-step methods (9.3.3) is determination of other initial values. A special section will be devoted to this problem.

Supposing that initial values y_i $(i = 0, 1, \ldots, k - 1)$ are known, at explicit methods are directly computed $y_k, y_{k+1}, \ldots, y_N$ using

$$y_{n+k} = h \sum_{i=0}^{k-1} \beta_i f_{n+i} - \sum_{i=0}^{k-1} \alpha_i y_{n+i} \quad (n = 0, 1, \ldots, N - k).$$

Nevertheless, at implicit methods for determination of values y_{n+k} the equation

$$y_{n+k} = h\beta f(x_{n+k}, y_{n+k}) + \Phi, \tag{9.3.4}$$

where

$$\Phi = h\sum_{i=0}^{k-1}\beta_i f_{n+i} - \sum_{i=0}^{k-1}\alpha_i y_{n+i},$$

shell be solved. When $(x,y) \to f(x,y)$ is nonlinear function which satisfies Lipschitz condition in y with constant L, the equation (9.3.4) can be solved by iterative process

$$y_{n+k}^{[s+1]} = h\beta_k f(x_{n+k}, y_{n+k}^{[s]}) + \Phi, \tag{9.3.5}$$

starting from arbitrary value $y_{n+k}^{[0]}$ if

$$h|\beta_k|L < 1.$$

The condition given by this inequality ensures convergence of iterative process (9.3.5).

Let us for method (9.3.3) define difference operator $L_h : C^1[x_0, b] \to C[x_0, b]$ by

$$L_h[y] = \sum_{i=0}^{k}[\alpha_i y(x+ih) - h\beta_i y'(x+ih)]. \tag{9.3.6}$$

Let function $g \in C^\infty[x_0, b]$. Then $L_h[g]$ can be presented in form

$$L_h[g] = C_0 g(x) + C_1 h g'(x) + C_2 h^2 g''(x) + \cdots, \tag{9.3.7}$$

where C_j $(j = 0, 1, \ldots)$ are constants not depending on h and g.

Definition 9.3.1. *Linear multi-step method* (9.3.3) *is of order p if in development* (9.3.7)

$$C_0 = C_1 = \ldots = C_p = 0 \text{ and } C_{p+1} \neq 0.$$

Let $x \to y(x)$ be exact solution of problem (9.3.1) and y_n series of approximate values of this solution in points $x_n = x_0 + nh$ $(n = 0, 1, \ldots, N)$ obtained by method (9.3.3), with initial values $y_i = s_i(h)$ $(i = 0, 1, \ldots, k-1)$.

Definition 9.3.2. *For linear multi-step method* (9.3.3) *one says to be convergent if for every $x \in [x_0, b]$*

$$\lim_{\substack{x \to 0 \\ x - x_0 = nh}} y_n = y(x)$$

and for initial values hold

$$\lim_{h\to 0} s_i(h) = y_0 \quad (i = 0, 1, \ldots, k-1).$$

Linear multi-step method (9.3.3) can be characterized by first and second characteristic polynomials given by

$$\rho(\xi) = \sum_{i=0}^{k}\alpha_i \xi^i \quad \text{and} \quad \sigma(\xi) = \sum_{i=0}^{k}\beta_i \xi^i,$$

respectively.

Two important classes of convergent multi-step methods, which are met in practice are:

1. Methods at which $\rho(\xi) = \xi^k - \xi^{k-1}$;
2. Methods at which $\rho(\xi) = \xi^k - \xi^{k-2}$.

Explicit methods of first class are called Adam-Bashforth methods, and the implicit Adam-Moulton methods. Similarly, explicit methods of second class are called Nystrom's methods and corresponding implicit methods Milne-Simpson's.

Of course, there are methods that do not belong to neither of these classes.

9.4. Choice of initial values

As earlier mentioned, at application of linear multi-step methods on solving problem (9.3.1), it is necessary knowledge on initial values $y_i = s_i(h)$, such that

$$\lim_{h\to 0} s_i(h) = y_0 \quad (i = 1, \ldots, k-1).$$

Certainly, this problem is stated when $k > 1$.

If method (9.3.3) is of order p, then initial values $s_i(h)$ are obviously to be chosen such that

$$s_i(h) - y(x_i) = \mathbf{O}(h^{p+1}) \quad (i = 1, \ldots, k-1),$$

where $x \to y(x)$ is exact solution of problem (9.3.1).

In this section we will show one class of methods for determination of necessary initial values.

Suppose that function f in differential equation (9.3.1) is enough times differentiable. Than, based on Tailor's method we have

$$y(x_0 + h) = y(x_0) + hy'(x_0) + \frac{h^2}{2!}y''(x_0) + \cdots + \frac{h^p}{p!}y^{(p)}(x_0) + \mathbf{O}(h^{p+1}).$$

Last equation points out that it can be taken

$$s_i(h) = y(x_0) + hy'(x_0) + \frac{h^2}{2!}y''(x_0) + \cdots + \frac{h^p}{p!}y^{(p)}(x_0),$$

because of $s_i(h) - y(x_1) = O(h^{p+1})$ $(x_1 = x_0 + h)$. The same procedure can be applied to determination of other initial values. Namely, in general case, we have

$$s_i(h) = y(x_{i-1}) + hy'(x_{i-1}) + \frac{h^2}{2!}y''(x_{i-1}) + \cdots + \frac{h^p}{p!}y^{(p)}(x_{i-1}) \quad (i = 1, \ldots, k-1),$$

whereby for $y(x_{i-1})$ we take $s_{i-1}(h)$.

9.5. Predictor-corrector methods

As mentioned in section 9.3., application of implicit methods is in connection with solution of equation (9.3.4) in every integration step, whereby for this solution is used iterative process (9.3.5). Regardless to this difficulty in implicit method, they are often used for solving Cauchy problem, because they have a number of advantages over explicit methods (higher order, better numerical stability). The initial value $y_{n+k}^{[0]}$ is determined in practice using some explicit method, which is then called predictor. Implicit method (9.3.4) is then called corrector. Method obtained by such combination is called predictor-corrector method.

For determination y_{n+k}, the iterative procedure (9.3.5) should be applied until fulfilment of the condition

$$|y_{n+k}^{[s+1]} - y_{n+k}^{[s]}| < \varepsilon,$$

where ε tolerable error, usually of order of local round-off error. Then for y_{n+k} can be taken $y_{n+k}^{[s+1]}$.

Nevertheless, this method is usually not applied in practice, due to demanding large number of function f evaluations by step of calculation and, in addition, this number is varying from step to step. In order to reduce this number of calculations, number of iterations in (9.3.5) is fixed. Thus, one takes only $s = 0, 1, \ldots, m-1$.

9.6. Program realization of multi-step methods

In this section we will give program realization of explicit as well as implicit methods. The presented programs are tested on the example (with $h = 0.1$).

$$y' = x^2 + y, \quad y(1) = 1 \quad (1 \le x \le 2).$$

The exact solution of this problem is $y(x) = 6e^{x-1} - x^2 - 2x - 2$.

9.6.1. Euler's method

Euler's method is given by expression

$$y_{n+1} - y_n = hf_n \quad (n = 0, 1, \ldots),$$

of order $p = 1$, and Adams-Bashforth method of third order

$$y_{n+3} - y_{n+2} = \frac{h}{12}(23f_{n+2} - 16f_{n+1} + 5f_n) \quad (n = 0, 1, \ldots),$$

are realized by subroutines EULER i ADAMS, respectively.

```
      SUBROUTINE EULER (XP,XK,H,Y,FUN)
      DIMENSION Y(1)
      N=(XK-XP+0.00001)/H
      X=XP
      DO 11 I=1,N
      Y(I+1)=Y(I)+H*FUN(X,Y(I))
   11 X=X+H
      RETURN
      END
C
      FUNCTION FUN (X,Y)
      FUN=X*X+Y
      RETURN
      END
C
      SUBROUTINE ADAMS (XP, XK, H, Y, FUN)
      DIMENSION Y(1)
      N=(XK-XP+0.00001)/H
      X=XP
      FO=FUN (X,Y(1))
      F1=FUN (X+H,Y(2))
      N2=N-2
      DO 11 I=1,N2
      F2=FUN(X+2.*H,Y(I+2))
      Y(I+3)=Y(I+2)+H*(23.*F2-16.*F1+5.*FO)/12.
      FO=F1
      F1=F2
   11 X=X+H
      RETURN
      END
```

Parameters in list of subroutine parameters are of following meaning:
XP and XK - start and end point of integration interval;
H - step of integration;
Y - vector of approximate values of solution obtained by multi-step method, where at Euler's method Y(1) represents given initial value, and at Adam's method initial values are given by Y(1), Y(2) and Y(3);
FUN - name of function subroutine which defines right hand size of differential equation $f(x,y)$. Initial values for Adam's method we determine by using Taylor's method for $p = 3$ (see section 9.3.4). Namely, being

$$y(1) = 1, \quad y'(1) = 2, \quad y''(1) = 4, \quad y'''(1) = 6, \quad h = 0.1,$$

we get Y(1)=1., Y(2)=1.221, Y(3)=1.48836.
Main program and output listing are of form:

```
C
C=================================================
C       RESAVANJE DIFERENCIJALNIH JEDNACINA
C             EKSPLICITNIM METODIMA
C=================================================
      EXTERNAL FUN
      DIMENSION Y(100),Z(100)
      F(X)=6.*EXP(X-1.)-X*X-2.*X-2.
      OPEN(5,FILE='EULER.OUT')
      WRITE (5,10)
   10 FORMAT(3X,'RESAVANJE DIFERENCIJAL.JED.',
     1'EKSPLICITNIM METODIMA'//8X,'XN',8X,'YN(I)',
     15X,'GRESKA(%)',3X,'YN(II)',4X,'GRESKA (%)'/)
      XP=1.
      XK=2.
      H=0.1
      Y(1)=1.
      CALL EULER (XP,XK,H,Y,FUN)
      Z(1)=Y(1)
      Z(2)=1.221
      Z(3)=1.48836
      CALL ADAMS (XP,XK,H,Z,FUN)
      N=(XK-XP+0.00001)/H
      NN=N+1
      X=XP
      DO 22 I=1,NN
      G1=ABS((Y(I)-F(X))/F(X))*100.
      G2=ABS((Z(I)-F(X))/F(X))*100.
      WRITE (5,20)X,Y(I),G1,Z(I),G2
   22 X=X+H
   20 FORMAT (8X,F3.1,2(4X,F9.5,4X,F5.2))
      CLOSE(5)
      STOP
      END
```

```
RESAVANJE DIFERENCIJAL.JED.EKSPLICITNIM METODIMA
XN        YN(I)     GRESKA(%)   YN(II)    GRESKA (%)
1.0      1.00000       .00     1.00000       .00
1.1      1.20000      1.72     1.22100       .00
1.2      1.44100      3.19     1.48836       .00
1.3      1.72910      4.42     1.80883       .02
1.4      2.07101      5.47     2.19028       .03
1.5      2.47411      6.37     2.64126       .04
```

```
  1.6        2.94652      7.13        3.17116        .05
  1.7        3.49717      7.79        3.79040        .06
  1.8        4.13589      8.36        4.51045        .06
  1.9        4.87348      8.87        5.34403        .07
  2.0        5.72183      9.32        6.30518        .07
```

9.6.2. Taking Euler's method as predictor and trapezoid rule ($p = 2$)

$$y_{n+1} - y_n = \frac{h}{2}(f_n + f_{n+1}) \quad (n = 0, 1, \ldots),$$

as corrector (with number of iterations $m = 2$) the subroutine PREKOR is written. Main program, subprogram, and output results are of form:

```
C=====================================================
C     RESAVANJE DIF.JED. METODOM PREDIKTOR-KOREKTOR
C=====================================================
      EXTERNAL FUN
      DIMENSION Y(100)
      F(X)=6.*EXP(X-1.)-X*X-2.*X-2.
      OPEN(5,FILE='PREKOR.OUT')
      OPEN(8,FILE='PREKOR.TXT')
      WRITE(5,10)
 10   FORMAT(8X,'RESAVANJE DIF. JED. METODOM',
     1' PREDIKTOR-KOREKTOR'//15X,'XN',13X,'YN'
     2,10X,'GRESKA(%)'/)
      READ(8,5)XP,XK,YP,H
    5 FORMAT(4F6.1)
      CALL PREKOR(XP,XK,YP,H,Y,FUN)
      N=(XK-XP+0.00001)/H
      NN=N+1
      X=XP
      DO 11 I=1,NN
      G=ABS((Y(I)-F(X))/F(X))*100.
      WRITE(5,15)X,Y(I),G
   15 FORMAT(15X,F3.1,8X,F9.5,8X,F5.2)
   11 X=X+H
      STOP
      END
C
C
      SUBROUTINE PREKOR(XP,XK,YP,H,Y,FUN)
      DIMENSION Y(100)
      N=(XK-XP+0.00001)/H
      X=XP
      Y(1)=YP
      DO 10 I=1,N
C   PROGNOZIRANJE VREDNOSTI
      FXY=FUN(X,Y(I))
      YP=Y(I)+H*FXY
C   KOREKCIJA VREDNOSTI
      DO 20 M=1,2
   20 YP=Y(I)+H/2.*(FXY+FUN(X+H,YP))
      Y(I+1)=YP
   10 X=X+H
      RETURN
      END
C
```

```
C
      FUNCTION FUN(X,Y)
      FUN=X*X+Y
      RETURN
      END
```

```
RESAVANJE DIF. JED. METODOM PREDIKTOR-KOREKTOR
       XN               YN              GRESKA(%)
       1.0              1.00000              .00
       1.1              1.22152              .04
       1.2              1.48952              .07
       1.3              1.81097              .10
       1.4              2.19363              .12
       1.5              2.64602              .14
       1.6              3.17760              .15
       1.7              3.79881              .17
       1.8              4.52118              .18
       1.9              5.35747              .18
       2.0              6.32177              .19
```

9.7. Runge-Kutta methods

In previous sections are considered linear multi-step methods for solving Cauchy problem (9.3.1). The order of these methods can be enlarged by increasing number of steps. Nevertheless, by sacrifice of linearity these methods posses, it is possible to construct single-step methods of arbitrary order.

For solving Cauchy problem of form (9.3.1) with enough times differentiable function f, it is possible to construct single-step methods of higher order (e.g. Taylor's method).

Consider general explicit single-step method

$$y_{n+1} - y_n = h\Phi(x_n, y_n, h) \tag{9.7.1}$$

Definition 9.7.1. *Method* (9.7.1) *is of order* p *if* p *is greatest integer for which holds*

$$y(x+h) - y(x) - h\Phi(x, y(x), h) = \mathbf{O}(h^{p+1}),$$

where $x \to y(x)$ *is exact solution of problem* (9.3.1).

Definition 9.7.2. *Method* (9.7.1) *is consistent if* $\Phi(x, y, 0) \equiv f(x, y)$.

Note that Taylor's method is special case of method (9.7.1). Namely, at Taylor's method of order p we have

$$\Phi(x, y, h) = \Phi_T(x, y, h) = \sum_{i=0}^{p-1} \frac{h^i}{(i-1)!}(\frac{\partial}{\partial x} + f\frac{\partial}{\partial y})^i f(x, y). \tag{9.7.2}$$

In special case, at Eulerov's method is $\Phi(x, y, h) = f(x, y)$.

In this section we will consider a special class of methods of form (9.7.1), which was proposed in 1895. year by C. Runge. Later on, this class of methods was developed by W. Kutta i K. Heun.

As we will see later, all these methods contain free parameters. Considering time in which these methods appeared, the free parameters have been chosen in such a way to obtain as simple as possible formulas for practical calculation. Nevertheless, such values of parameters do not ensure optimal characteristics of observed methods. In further text these methods will be called classical. General explicit Runge-Kutta method is of form

$$y_{n+1} - y_n = h\Phi(x_n, y_n, h) \tag{9.7.3}$$

where

$$\Phi(x, y, h) = \sum_{i=1}^{m} c_i k_i,$$

$$\begin{aligned} k_1 &= f(x, y), \\ k_i &= f(x + a_i, y + b_i h) \quad (i = 2, \ldots, m). \end{aligned}$$

(9.7.4) $$a_i = \sum_{j=1}^{i-1} \alpha_{ij}, \quad b_i = \sum_{j=1}^{i-1} \alpha_{ij} k_j.$$

Note that from the condition of consistence of method (9.7.3) it follows

$$\sum_{i=1}^{m} c_i = 1.$$

Unknown coefficients which appear in this method are to be determined from the condition that method has a maximal order. Here, we use the following fact: If $\Phi(x, y, h)$, developed by degrees of h, can be presented in form

$$\Phi(x, y, h) = \Phi_T(x, y, h) = \mathbf{O}(h^p),$$

where Φ_T is defined by (9.7.2), then method (9.7.3) is of order p.

Find previously development $\Phi_T(x, y, h)$ by degrees of h. Using Monge's notations for partial derivative, we have

$$(\frac{\partial}{\partial x} + \frac{\partial}{\partial y})f = f_x + f f_y = F$$

and

$$(\frac{\partial}{\partial x} + f\frac{\partial}{\partial y})^2 f = (\frac{\partial}{\partial x} + f\frac{\partial}{\partial y})F = G + f_y F,$$

where we put $G = f_{xx} + 2ff_{xy} + f^2 f_{yy}$. Then from (9.7.2) it follows

(9.7.5) $$\Phi_T(x, y, h) = f + \frac{1}{2}hF + \frac{1}{6}h^2(G + f_y F) + \mathbf{O}(h^3).$$

Consider now only Runge-Kutta methods of order $p \le 3$. One shows that for obtaining method of third order it is enough to take $m = 3$. In this case, formulas (9.7.3) reduce to

$$\begin{aligned} \Phi(x, y, h) &= c_1 k_1 + c_2 k_2 + c_3 k_3 \\ k_1 &= f(x, y) \\ k_2 &= f(x + a_2 h, y + b_2 h), \\ k_3 &= f(x + a_3 h, y + b_3 h) \end{aligned}$$

and

$$\begin{aligned} a_2 &= \alpha_{21}, \quad b_2 = \alpha_{21} k_1, \\ a_3 &= \alpha_{31} + \alpha_{32}, \quad b_3 = \alpha_{31} k_1 + \alpha_{32} k_2. \end{aligned}$$

By developing of function k_2 in Taylor's series in neighborhood of point (x, y), we get

$$k_2 = f + a_2Fh + \frac{1}{2}a_2^2Gh^2 + \mathbf{O}(h^3).$$

Because of

$$b_3 = \alpha_{31}k_1 + \alpha_{32}k_2 = \alpha_{31}f + \alpha_{32}(f + a_2Fh + \frac{1}{2}a_2^2Gh^2) + \mathbf{O}(h^3)$$

we have

$$b_3 = a_3f + a_2\alpha_{32}Fh + \mathbf{O}(h^2)$$

and

$$b_3^2 = a_3^2f^2 + \mathbf{O}(h).$$

By developing of function k_3 in neighborhood of point (x, y) and by using last equalities we have

$$k_3 = f + a_3Fh + \frac{1}{2}(2a_3\alpha_{32}Ff_y + a_3^2G)h^2 + \mathbf{O}(h^3).$$

Finally, by substituting the obtained expressions for k_1, k_2, k_3 in expression for $\Phi(x, y, h)$ we get

$$\begin{aligned}\Phi(x, y, h) =&(c_1 + c_2 + c_3)f + (c_2a_2 + c_3a_3)Fh\\ &+ (c_2a_2^2G + 2c_3a_2\alpha_{32}Ff_y + c_3a_3^2G)\frac{h^2}{2} + \mathbf{O}(h^3).\end{aligned}$$

Last equality enables construction of methods for $m = 1, 2, 3$.

Case m=1. Being $c_2 = c_3 = 0$, we have

$$\Phi(x, y, h) = c_1f + \mathbf{O}(h^3).$$

By comparison with (9.7.5) we get

$$\Phi_T(x, y, h) - \Phi(x, y, h) = (1 - c_1)f + \frac{1}{2}h^2(G + f_yF) + \mathbf{O}(h^3),$$

wherefrom we conclude that for $c_1 = 1$ the method

$$y_{n+1} - y_n = hf_n,$$

of order $p = 1$ is obtained. Considering that it is Euler's method, we see that it belongs to the class of Runge-Kutta methods too.

Case m=2. Here is $c_3 = 0$ and

$$\Phi_T(x, y, h) = (c_1 + c_2)f + c_2a_2Fh + \frac{1}{2}c_2a_2^2Gh^2 + \mathbf{O}(h^3).$$

Because of

$$\begin{aligned}\Phi(x, y, h) - \Phi_T(x, y, h) = &(c_1 + c_2 - 1)f + (c_2a_2 - \frac{1}{2})Fh\\ &+ \frac{1}{6}[(3c_2a_2^2 - 1)G - f_yF]h^2 + \mathbf{O}(h^3),\end{aligned}$$

we conclude that under condition

$$c_1 + c_2 = 1 \quad \text{and} \quad c_2 a_2 = \frac{1}{2}, \tag{9.7.6}$$

one obtains method of second order with one free parameter. Namely, from system of equations (9.7.6) it follows

$$c_2 = \frac{1}{2a_2} \quad \text{and} \quad c_1 = \frac{2a_2 - 1}{2a_2},$$

where $a_2 (\neq 0)$ is free parameter. Thus, with $m = 2$ we have single-parametric family of methods

$$\begin{aligned} y_{n+1} - y_n &= \frac{h}{2a_2}((2a_2 - 1)k_1 + k_2), \\ k_1 &= f(x_n, y_n), \\ k_2 &= f(x_n + a_2 h, y_n + a_2 k_1 h). \end{aligned}$$

In special case, for $a_2 = \frac{1}{2}$, we get Euler-Cauchy method

$$y_{n+1} - y_n = hf(x_n + \frac{1}{2}h, y_n + \frac{1}{2}hf(x_n, y_n)).$$

Similarly, for $a_2 = 1$, we get so known improved Euler-Cauchy method

$$y_{n+1} - y_n = \frac{h}{2}[f(x_n, y_n) + f(x_n + h, y_n + hf(x_n, y_n)].$$

On geometric interpretation of obtained methods see, e.g. [6].

Case m=3. According to

$$\begin{aligned} \Phi(x, y, h) - \Phi_T(x, y, h) = (c_1 + c_2 + c_3 - 1)f + (c_2 a_2 + c_3 a_3 - \frac{1}{2})Fh \\ + [(c_2 a_2^2 + c_3 a_3^2 - \frac{1}{3})G + (2c_3 a_2 \alpha_{32} - \frac{1}{3})F f_y]\frac{h^2}{2} + O(\frac{h}{3}), \end{aligned}$$

we conclude that for obtaining of methods of third order the satisfactory conditions are

$$\begin{aligned} c_1 + c_2 + c_3 &= 1, \\ c_2 a_2 + c_3 a_3 &= \frac{1}{2}, \\ c_2 a_2^2 + c_3 a_3^2 &= \frac{1}{3}, \\ c_3 a_2 \alpha_{32} &= \frac{1}{6}. \end{aligned} \tag{9.7.7}$$

Having four equations with six unknowns, it follows that, in case $m = 3$, we have two-parametric family of Runge-Kutta methods. One can show that among methods of this family does not exists not single method with order greater than three.

In special case, when $a_2 = \frac{1}{3}$ and $a_3 = \frac{2}{3}$, from (9.7.7) it follows $c_1 = \frac{1}{4}, c_2 = 0, c_3 = \frac{3}{4}, \alpha_{32} = \frac{2}{3}$. Thus, we obtained the method

$$\begin{aligned} y_{n+1} - y_n &= \frac{h}{4}(k_1 + 3k_3),\\ k_1 &= f(x_n, y_n),\\ k_2 &= f(x_n + \frac{h}{3}, y_n + \frac{h}{3}k_1),\\ k_3 &= f(x_n + \frac{2h}{3}, y_n + \frac{2h}{3}k_2), \end{aligned}$$

which is known in bibliography as Heun's method.

For $a_2 = \frac{1}{2}, a_3 = 1 (\Rightarrow c_1 = c_3 = \frac{1}{6}, c_2 = \frac{2}{3}, \alpha_{32} = 2)$ we get the method

$$\begin{aligned} y_{n+1} - y_n &= \frac{h}{6}(k_1 + 4k_2 + k_3),\\ k_1 &= f(x_n, y_n),\\ k_2 &= f(x_n + \frac{h}{2}, y_n + \frac{h}{2}k_1),\\ k_3 &= f(x_n + h, y_n - hk_1 + 2hk_2), \end{aligned}$$

which is most popular among the methods of third order from the point of view of hand calculations.

In case when $m = 4$, we get two-parameter family of methods of fourth order. Namely, here, analogously to system (9.7.7), appears system of 11 equations in 13 unknowns.

Now we quote, without proof, Runge-Kutta method of fourth order.

$$\begin{aligned} y_{n+1} - y_n &= \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4),\\ k_1 &= f(x_n, y_n),\\ k_2 &= f(x_n + \frac{h}{2}, y_n + \frac{h}{2}k_1),\\ k_3 &= f(x_n + \frac{h}{2}, y_n + \frac{h}{2}k_2),\\ k_4 &= f(x_n + h, y_n + hk_3), \end{aligned} \tag{9.7.8}$$

which is traditionally most used in applications.

From methods of fourth order it is often used so known Gill's variant, which can be expressed as the following recursive procedure:

$$\begin{aligned} &n := 0, \quad Q_0 := 0\\ (*)\ &Y_0 := y_n,\\ &k_1 := hf(x_n, Y_0), \quad Y_1 := Y_0 + \frac{1}{2}(k_1 - 2Q_0),\\ &Q_1 := Q_0 + \frac{3}{2}(k_1 - 2Q_0) - \frac{1}{2}k_1,\\ &k_2 := hf(x_n + \frac{h}{2}, Y_1), \quad Y_2 := Y_1 + (1 - \sqrt{1/2})(k_2 - Q_1),\\ &Q_2 := Q_1 + 3(1 - \sqrt{1/2})(k_2 - Q_1)(1 - \sqrt{1/2})k_2,\\ &k_3 := hf(x_n + \frac{h}{2}, Y_2), \quad Y_3 := Y_2 + (1 + \sqrt{1/2})(k_3 - Q_2),\\ &Q_3 := Q_2 + 3(1 + \sqrt{1/2})(k_3 - Q_2) - (1 + \sqrt{1/2})k_3, \end{aligned}$$

$$k_4 := hf(x_n + h, Y_3), \quad Y_4 := Y_3 + \frac{1}{6}(k_4 - 2Q_3),$$
$$Q_0 := Q_3 + \frac{1}{2}(k_4 - 2Q_3) - \frac{1}{2}k_4,$$
$$y_{n+1} := Y_4,$$
$$n := n + 1$$
skip to $(*)$.

In contrary to linear multi-step methods, Runge-Kutta methods do not demand knowledge of initial values (except $y(x_0) = y_0$, what, by the way, defines Cauchy problem), but for practical application are more complicated, because they demand m calculations of function f values in every step.

9.8. Program realization of Runge-Kutta methods

In this section we present program realization of Euler-Cauchy method, improved Eueler-Cauchy method, as well as method of fourth order (9.7.8) and Gill's variant of Runge-Kutta method. The obtained software will be tested on the example from section 9.3.6.

Program 9.8.1.

By subroutine EULCAU are realized Euler-Cauchy and improved Euler-Cauchy method. Parameters in parameter list have the following meaning:

XP - start point of integration interval;

H - integration step;

N - integer, such that $N + 1$ is lenght of vector Y;

M - integer which defines way of construction of vector Y. Namely, in vector Y is stored in turn every M-th value of solution obtained during integration process.

Y - vector containing solutions of length N+1, whereby Y(1) is given initial condition y_0, Y(2) is value of solution obtained by integration in point XP + M*H, etc.

FUN - name of function subroutine, which defines right-hand side of differential equation $f(x, y)$;

K - integer with values K=1 and K=2, which governs integration according to Euler-Cauchy and improved Euler-Cauchy method, respectively.

Subroutine EULCAU is of form:

```
      SUBROUTINE EULCAU(XP,H,N,M,Y,FUN,K)
      DIMENSION Y(1)
      X=XP
      Y1=Y(1)
      NN=N+1
      DO 10 I=2,NN
      DO 20 J=1,M
      YO=Y1
      Y1=FUN(X,YO)
      GO TO (1,2),K
  1   Y1=YO+H*FUN(X+0.5*H,YO+0.5*H*Y1)
      GO TO 20
  2   Y1=YO+H*(Y1+FUN(X+H,YO+H*Y1))/2.
 20   X=X+H
10    Y(I)=Y1
      RETURN
      END
C
      FUNCTION FUN(X,Y)
      FUN=X*X+Y
      RETURN
      END
```

Main program and output listing are given in further text. As input parameters for integration we have taken `H=0.1`, `N=10`, `M=1`, and in second case `H=0.05`, `N=10`, `M=2`. Columns `Y1N` and `Y2N` in output listing give values for solution of given Cauchy problem, according to regular and improved Euler-Cauchy method, respectively. In addition to those columns, in output listing are given columns with corresponding errors (as relation to exact solution, expressed in %)

```
C==================================================
C RESAVANJE DIF. JED. EULER-CAUCHYEVIM
C      I POBOLJSANIM METODOM
C==================================================
      EXTERNAL FUN
      DIMENSION Y(100), Z(100)
      F(X)=6.*EXP(X-1.)-X*X-2.*X-2.
      OPEN(5,FILE='EULCAU.OUT')
      OPEN(8,FILE='EULCAU.IN')
      WRITE(5,10)
10    FORMAT(10X,'RESAVANJE DIF.JED.EULER-CAUCHYEVIM'
     1 ' I POBOLJSANIM METODOM')
20    READ(8,25,END=99)XP,Y(1),H,N,M
25    FORMAT(3F6.1,2I3)
      CALL EULCAU(XP,H,N,M,Y,FUN,1)
      Z(1)=Y(1)
      CALL EULCAU(XP,H,N,M,Z,FUN,2)
      WRITE(5,30)H
30    FORMAT(1H0,30X,'(H=',F6.4,')'//15X,'XN',8X,
     1'Y1N',4X,'GRESKA(%)',5X,'Y2N',4X,'GRESKA(%)'/)
      NN=N+1
      X=XP
      DO 11 I=1,NN
      G1=ABS((Y(I)-F(X))/F(X))*100.
      G2=ABS((Z(I)-F(X))/F(X))*100.
      WRITE(5,15)X,Y(I),G1,Z(I),G2
15    FORMAT(15X,F3.1,3X,F9.6,2X,F7.5,3X,F9.6,2X,
     1 F7.5)
11    X=X+H*M
      GO TO 20
99    CLOSE(5)
      CLOSE(8)
      STOP
      END
```

```
RESAVANJE DIF.JED.EULER-CAUCHYEVIM I POBOLJSANIM METODOM
                          (H= .1000)
    XN         Y1N     GRESKA(%)      Y2N     GRESKA(%)
    1.0    1.000000     .00000    1.000000     .00000
    1.1    1.220250     .06352    1.220500     .04304
    1.2    1.486676     .11693    1.487203     .08157
    1.3    1.806227     .16173    1.807059     .11576
    1.4    2.186581     .19934    2.187750     .14599
    1.5    2.636222     .23109    2.637764     .17274
    1.6    3.164526     .25808    3.166479     .19650
    1.7    3.781851     .28125    3.784260     .21773
    1.8    4.499645     .30138    4.502557     .23685
    1.9    5.330558     .31907    5.334026     .25422
    2.0    6.288567     .33483    6.292649     .27013
```

```
                    (H= .0500)
XN        Y1N      GRESKA(%)       Y2N     GRESKA(%)
1.0    1.000000    .00000      1.000000    .00000
1.1    1.220824    .01655      1.220888    .01130
1.2    1.487963    .03046      1.488098    .02140
1.3    1.808391    .04213      1.808604    .03034
1.4    2.189811    .05192      2.190111    .03824
1.5    2.640738    .06019      2.641133    .04523
1.6    3.170581    .06721      3.171082    .05143
1.7    3.789740    .07324      3.790357    .05696
1.8    4.509705    .07848      4.510451    .06195
1.9    5.343177    .08309      5.344066    .06647
2.0    6.304192    .08719      6.305238    .07061
```

Program 9.8.2.

According to formulas (9.7.8) for standard Runge-Kutta method of fourth degree, the following subroutine RK4 is written:

```
      SUBROUTINE RK4(X0,Y0,H,M,N,YVEK,F)
C==============================================
C     METOD RUNGE-KUTTA CETVRTOG REDA
C==============================================
      DIMENSION YVEK(1)
      T=H/2.
      X=X0
      Y=Y0
      DO 20 I=1,N
      DO 10 J=1,M
      A=F(X,Y)
      B=F(X+T,Y+T*A)
      C=F(X+T,Y+T*B)
      D=F(X+H,Y+H*C)
      X=X+H
 10   Y=Y+H/6.*(A+2.*B+2.*C+D)
 20   YVEK(I)=Y
      RETURN
      END
```

Parameters in list of subroutine parameters are of following meaning:
X0,Y0 - define given initial condition (Y0=y(X0));
H - step of integration;
M, N - integers with meanings similar to ones in subroutine EULCAU;
YVEK - vector of length N which is obtained as result of numerical integration, whereby Y(1) is value obtained in point X0+M*H, Y(2) value in point X0+2M*H, etc.
F - name of function subroutine which defines right-hand side of differential equation $f(x,y)$.
Main program is of form:

```
C===========================================
C    RESAVANJE DIF.JED. METODOM RUGE-KUTTA
C===========================================
      EXTERNAL FUN
      DIMENSION Y (100)
      F(X)=6.*EXP(X-1.)-X*X-2.*X-2.
      OPEN(5,FILE='RK4.OUT')
      OPEN(8,FILE='RK4.IN')
      WRITE(5,10)
 10   FORMAT (14X,'RESAVANJE DIF.JED. METODOM',
     1 ' RUNGE-KUTTA')
 20   READ (8,5,END=99)X0,Y0,H,N,M
 5    FORMAT (3F6.1,2I3)
      CALL RK4(X0,Y0,H,M,N,Y,FUN)
      G=0.
      WRITE (5,25) H,X0,Y0,G
25    FORMAT( 28X,'(H=',F6.4,')'//15X,'XN',13X,'YN',
     110X,'GRESKA(%)'//15X,F3.1,8X,F9.6,7X,F7.5)
      X=X0
      DO 11 I=1,N
      X=X+H*M
      G=ABS((Y(I)-F(X))/F(X))*100.
11    WRITE (5,15)X,Y(I),G
15    FORMAT (15X,F3.1,8X,F9.6,7X,F7.5)
      GO TO 20
99    CLOSE(5)
      CLOSE(8)
      STOP
      END
C
      FUNCTION FUN(X,Y)
      FUN=X*X+Y
      RETURN
      END
```

Taking H=0.1, N=10, M=1 the following results are obtained:

```
RESAVANJE DIF.JED. METODOM RUNG-KUTTA
               (H= .1000)
  XN             YN           GRESKA(%)
  1.0          1.000000        .00000
  1.1          1.221025        .00002
  1.2          1.488416        .00005
  1.3          1.809152        .00007
  1.4          2.190946        .00009
  1.5          2.642325        .00011
  1.6          3.172709        .00012
  1.7          3.792512        .00014
  1.8          4.513240        .00015
  1.9          5.347611        .00017
  2.0          6.309682        .00018
```

Program 9.8.3.

The Gill's variant of Runge-Kutta method is realized in double precision. Parameters in parameter list of subroutine GILL, X0, H, N, M, Y, FUN have the same meaning as the parameters HP, H, N, M, Y, FUN in subroutine EULCAU, respectively. Note that this subroutine is realized in such a way that the optimization of number of variables has been performed.

Input parameters are taken like in program **9.8.1.**

```
C==================================================
C  RESAVANJE DIF.JED. METODOM RUNGE-KUTTA
C              (GILLOVA VARIJANTA)
C==================================================
      EXTERNAL FUN
      REAL*8 Y(100),F,FUN,X0,X,H,G
      F(X)=6.*DEXP(X-1.)-X*X-2.*X-2.
      OPEN(8,FILE='GILL.IN')
      OPEN(5,FILE='GILL.OUT')
      WRITE(5,10)
   10 FORMAT(8X,'RESAVANJE DIF.JED.METODOM'
     1' RUNGE-KUTTA (GILLOVA VARIJANTA)' )
   20 READ(8,25,END=99)X,Y(1),H,N,M
   25 FORMAT(3F6.1,2I3)
      X0=X
      CALL GILL(X0,H,N,M,Y,FUN)
      WRITE(5,30)H
   30 FORMAT(/28X,'(H=',F6.4,')'//15X,'XN',13X,'YN',
     1 10X,'GRESKA(%)'/)
      NN=N+1
      DO 11 I=1,NN
      G=DABS((Y(I)-F(X))/F(X))*100.
      WRITE(5,15)X,Y(I),G
   15 FORMAT(15X,F3.1,8X,F9.6,6X,D10.3)
   11 X=X+H*M
      GO TO 20
   99 CLOSE(5)
      CLOSE(8)
      STOP
      END
C
C
      SUBROUTINE GILL(X0,H,N,M,Y,FUN)
      REAL*8 Y(1),H,FUN,X0,Y0,Q,K,A,B
      B=DSQRT(0.5D0)
      Q=0.D0
      Y0=Y(1)
      NN=N+1
      DO 10 I=2,NN
      DO 20 J=1,M
      K=H*FUN(X0,Y0)
      A=0.5*(K-2.*Q)
      Y0=Y0+A
      Q=Q+3.*A-0.5*K
      K=H*FUN(X0+H/2.,Y0)
      A=(1.-B)*(K-Q)
      Y0=Y0+A
      Q=Q+3.*A-(1.-B)*K
      K=H*FUN(X0+H/2,Y0)
      A=(1.+B)*(K-Q)
      Y0=Y0+A
      Q=Q+3.*A-(1.+B)*K
      K=H*FUN(X0+H,Y0)
      A=(K-2.*Q)/6.
      Y0=Y0+A
      Q=Q+3.*A-K/2.
   20 X0=X0+H
   10 Y(I)=Y0
      RETURN
      END
C
      FUNCTION FUN(X,Y)
      REAL*8 FUN,X,Y
      FUN=X*X+Y
      RETURN
      END
```

```
RESAVANJE DIF.JED.METODOM RUNGE-KUTTA (GILLOVA VARIJANTA)
                         (H= .1000)
        XN                 YN            GRESKA(%)
        1.0             1.000000          .000D+00
        1.1             1.221025          .246D-04
        1.2             1.488416          .460D-04
        1.3             1.809152          .647D-04
        1.4             2.190946          .808D-04
        1.5             2.642325          .949D-04
        1.6             3.172709          .107D-03
        1.7             3.792512          .118D-03
        1.8             4.513240          .128D-03
        1.9             5.347611          .136D-03
        2.0             6.309682          .144D-03
                         (H= .0500)
        XN                 YN            GRESKA(%)
        1.0             1.000000          .000D+00
        1.1             1.221025          .162D-05
        1.2             1.488417          .303D-05
        1.3             1.809153          .425D-05
        1.4             2.190948          .531D-05
        1.5             2.642327          .623D-05
        1.6             3.172713          .704D-05
        1.7             3.792516          .775D-05
        1.8             4.513245          .838D-05
        1.9             5.347618          .894D-05
        2.0             6.309690          .946D-05
```

9.9. Solution of system of equations and equations of higher order

Methods considered in previous sections can be generalized in that sense to be applicable in solution of Cauchy problem for system of p equations of first order

$$y_i' = f_i(x; y_1, \ldots, y_p), \quad y_i(x_0) = y_{i0} \quad (i = 1, \ldots, p). \tag{9.9.1}$$

In this case, system of equations (9.9.1) shell be represented in vector form

$$\vec{y}\,' = \vec{f}(x, \vec{y}), \quad \vec{y}(x_0) = \vec{y}_0, \tag{9.9.2}$$

where

$$\vec{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_p \end{bmatrix}, \quad \vec{y}_0 = \begin{bmatrix} y_{10} \\ y_{20} \\ \vdots \\ y_{p0} \end{bmatrix}, \quad \vec{f}(x, \vec{y}) = \begin{bmatrix} f_1(x; y_1, \ldots, y_p) \\ \vdots \\ f_p(x; y_1, \ldots, y_p) \end{bmatrix}.$$

It is of our interest the solution of Cauchy problem for differential equations of higher order. Note, nevertheless, that this problem can be reduced to previous one. Namely, let be given the differential equation of order p

$$y^{(p)} = f(x, y, y', \ldots, y^{(p-1)}) \tag{9.9.3}$$

with initial conditions

$$y^{(i)}(x_0) = y_{i0} \quad (i = 0, 1, \ldots, p-1). \tag{9.9.4}$$

Then, by substitution

$$z_1 = y, \quad z_2 = y', \ldots, z_p = y^{(p-1)},$$

equation (9.9.3) with conditions (9.9.4), reduces to system

$$\begin{aligned} z_1' &= z_2 \\ z_2' &= z_3 \\ &\vdots \\ z_{p-1}' &= z_m \\ z_p' &= f(x; z_1, z_2, \ldots, z_p), \end{aligned}$$

with conditions $z_i(x_0) = z_{i0} = y_{i0} \quad (i = 1, \ldots, p)$.

Linear multi-step methods considered up to now, can be formally generalized to vector form

$$\sum_{i=0}^{k} \alpha_i \vec{y}_{n+i} = h \sum_{i=0}^{k} \beta_i \vec{f}_{n+i},$$

where $\vec{f}_{n+i} = \vec{f}(x_{n+i}, \vec{y}_{n+i})$, and then as such can be applied to solution of Cauchy problem (9.9.2).

Also, the Runge-Kutta methods for solution of Cauchy problem (9.9.2) are of form

$$\vec{y}_{n+1} - \vec{y}_n = h\vec{\Phi}(x_n, \vec{y}_n, h),$$

where

$$\begin{aligned} &\vec{\Phi}(x, \vec{y}, h) = \sum_{i=1}^{m} c_i \vec{k}_i, \\ &\vec{k}_1 = \vec{f}(x, \vec{y}), \\ &\vec{k}_i = \vec{f}(x + a_i h, \ \vec{y} + \vec{b}_i h) \\ &a_i = \sum_{j=1}^{i-1} \alpha_{ij}, \quad \vec{b}_i = \sum_{j=1}^{i-1} \alpha_{ij} \vec{k}_j \quad (i = 2, \ldots, m). \end{aligned}$$

All analysis given in previous sections can formally be translated to noted vector methods.

As an example, realize standard Runge-Kutta method of forth order (9.7.8) for solving of system of two differential equations

$$y' = f_1(x, y, z), \quad z' = f_2(x; y, z),$$

with conditions $y(x_0) = y_0$ and $z(x_0) = z_0$.

The corresponding subroutine is of form:

```
SUBROUTINE RKS(XP,XKRAJ,YP,ZP,H,N,YY,ZZ)
REAL KY1,KY2,KY3,KY4,KZ1,KZ2,KZ3,KZ4
DIMENSION YY(1),ZZ(1)
K=(XKRAJ-XP)/(H*FLOAT(N))
```

```
      N1=N+1
      X=XP
      Y=YP
      Z=ZP
      T=H/2.
      YY(1)=Y
      ZZ(1)=Z
      DO 6 I=2,N1
      DO 7 J=1,K
      KY1=FUN(1,X,Y,Z)
      KZ1=FUN(2,X,Y,Z)
      KY2=FUN(1,X+T,Y+T*KY1,Z+T*KZ1)
      KZ2=FUN(2,X+T,Y+T*KY1,Z+T*KZ1)
      KY3=FUN(1,X+T,Y+T*KY2,Z+T*KZ2)
      KZ3=FUN(2,X+T,Y+T*KY2,Z+T*KZ2)
      KY4=FUN(1,X+H,Y+H*KY3,Z+H*KZ3)
      KZ4=FUN(2,X+H,Y+H*KY3,Z+H*KZ3)
      Y=Y+H*(KY1+2.*(KY2+KY3)+KY4)/6.
      Z=Z+H*(KZ1+2.*(KZ2+KZ3)+KZ4)/6.
    7 X=X+H
      YY(I)=Y
    6 ZZ(I)=Z
      RETURN
      END
```

Using this subroutine we solved system of equations

$$y' = xyz, \quad z' = xy/z,$$

under conditions $y(1) = 1/3$ and $z(1) = 1$ on segment $[1, 2.5]$ taking for integration step $h = 0.01$, and printing on exit x with step 0.1 and corresponding values of y, y_T, z, z_T, where y_T and z_T are exact solutions of this system, given with

$$y_T = \frac{72}{(7-x^2)^3} \quad \text{and} \quad z_T = \frac{6}{7-x^2}.$$

The corresponding program and output listing are of form:

```
C=====================================================
C   RESAVANJE SISTEMA DIF. JED. METODOM RUNGE-KUTTA
C=====================================================
      DIMENSION YT(16),ZT(16),YY(16),ZZ(16),X(16)
      YEG(P)=72./(7.-P*P)**3
      ZEG(P)=6./(7.-P*P)
      OPEN(8,FILE='RKS.IN')
      OPEN(5,FILE='RKS.OUT')
      READ(8,15)N,XP,YP,ZP,XKRAJ
15    FORMAT(I2,4F3.1)
      YP=YP/3.
      H=0.1
      N1=N+1
      DO 5 I=1,N1
      X(I)=XP+H*FLOAT(I-1)
      YT(I)=YEG(X(I))
    5 ZT(I)=ZEG(X(I))
      WRITE(5,22)
      H=0.01
      CALL RKS(XP,XKRAJ,YP,ZP,H,N,YY,ZZ)
      WRITE(5,18)H,(X(I),YY(I),YT(I),ZZ(I),ZT(I),
     1     I=1,N1)
18    FORMAT(//7X,'KORAK INTEGRACIJE H=',F6.3//7X,
     1'X',11X,'Y',10X,'TACNO',11X,'Z',10X,'ZTACNO'//
     2(F10.2,4F14.7))
22    FORMAT(1H1,9X,'RESAVANJE SISTEMA SIMULTANIH',
     1' DIFERENCIJALNIH JEDNACINA'//33X,'Y''=XYZ'//
     1 33X, 'Z''=XY/Z')
      CLOSE(5)
      CLOSE(8)
      STOP
      END
C
      FUNCTION FUN(J,X,Y,Z)
      GO TO (50,60),J
   50 FUN=X*Y*Z
      RETURN
   60 FUN=X*Y/Z
      RETURN
      END
```

```
1         RESAVANJE SISTEMA SIMULTANIHDIFERENCIJALNIH JEDNACINA
                                 Y'=XYZ
                                 Z'=XY/Z
       KORAK INTEGRACIJE H=  .010
       X           Y            TACNO           Z             ZTACNO
      1.00      .3333333      .3333333     1.0000000     1.0000000
      1.10      .3709342      .3709342     1.0362690     1.0362690
      1.20      .4188979      .4188979     1.0791370     1.0791370
      1.30      .4808936      .4808935     1.1299430     1.1299430
      1.40      .5623943      .5623943     1.1904760     1.1904760
      1.50      .6718181      .6718181     1.2631580     1.2631580
      1.60      .8225902      .8225904     1.3513510     1.3513510
      1.70     1.0370670     1.0370680     1.4598540     1.4598540
      1.80     1.3544680     1.3544680     1.5957440     1.5957450
      1.90     1.8481330     1.8481340     1.7699110     1.7699110
      2.00     2.6666650     2.6666670     2.0000000     2.0000000
      2.10     4.1441250     4.1441260     2.3166020     2.3166020
      2.20     7.1444800     7.1444920     2.7777760     2.7777780
      2.30    14.3993600    14.3993900     3.5087690     3.5087710
      2.40    37.7628900    37.7631300     4.8387000     4.8387110
      2.50   170.6632000   170.6667000     7.9999230     8.0000000
```

9.10. Boundary problems

In this section we will point out to difference method for solution boundary problem

(9.10.1) $$y'' + p(x)y' + q(x)y = f(x); \quad y(a) = A, \; y(b) = B,$$

where functions p, q, f are continuous on $[a, b]$.

Let us divide segment $[a, b]$ to $N + 1$ subsegments of length $h = \dfrac{b-a}{N+1}$, so that $x_n = a + nh$ $(n = 0, 1, ..., N + 1)$. In points x_n $(n = 1, \ldots, N)$ we approximate the differential equation from (9.10.1) with

(9.10.2) $$\frac{y_{n+1} - 2y_n + y_{n-1}}{h^2} + p_n \frac{y_{n+1} - y_{n-1}}{2h} + q_n y_n = f_n \quad (n = 1, \ldots, N),$$

where $p_n \equiv p(x_n)$, $q_n \equiv q(x_n)$, $f_n \equiv f(x_n)$.

If we involve substitutions

$$a_n = 1 - \frac{h}{2}p_n,\ b_n = h^2 q_n - 2,\ c_n = 1 + \frac{h}{2}p_n,$$

(9.10.2) can be represented in form

$$a_n y_{n-1} + b_n y_n + c_n y_{n+1} = h^2 f_n \quad (n = 1, \ldots, N). \tag{9.10.3}$$

In regards to boundary conditions $y_0 = A$ and $Y_{N+1} = B$, we have the problem of solving system of linear equations $\mathbf{T}\vec{y} = \vec{d}$, where

$$\vec{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix},\quad \vec{d} = \begin{bmatrix} h^2 f_1 - Aa_1 \\ h^2 f_2 \\ \vdots \\ h^2 f_N - Bc_N \end{bmatrix},\quad \mathbf{T} = \begin{bmatrix} b_1 & c_1 & 0 & \ldots & 0 \\ a_2 & b_2 & c_2 & 0 & \\ \vdots & & & & \\ 0 & 0 & 0 & \ldots & b_N \end{bmatrix}.$$

System matrix is tri-diagonal. For solving of this system it is convenient to perform decomposition of matrix **T** as **T=LR** (see Chapter 2), whereby the problem is reduced to successive solution of two triangular systems of linear equations. This procedure for solution boundary problem (9.10.1), is known as matrix factorization.

The following program is written in accordance to explained procedure.

```
      DIMENSION A(100),B(100),C(100),D(100)
C=================================================
C  MATRICNA FAKTORIZACIJA ZA RESAVANJE
C  KONTURNIH PROBLEMA KOD  LINEARNIH
C  DIFERENCIJALNIH JEDNACINA II REDA
C  Y''+ P(X)Y'+ Q(X)Y = F(X)
C  Y(DG) = YA, Y(GG) = YB
C ================================================
      OPEN(8,FILE='KONTUR.IN')
      OPEN(7,FILE='KONTUR.OUT')
      READ(8,5) DG,YA,GG,YB
 5    FORMAT(4F10.5)
C UCITAVANJE BROJA MEDJUTACAKA
 10   WRITE(*,14)
 14   FORMAT(1X,'UNETI BROJ MEDJUTACAKA'
     1' U FORMATU I2'/ 5X,'(ZA N=0 => KRAJ)')
      READ(5,15) N
 15   FORMAT(I2)
      N1=N+1
      IF(N.EQ.0) GO TO 60
      H=(GG-DG)/FLOAT(N1)
      HH=H*H
      X=DG
      DO 20 I=1,N
      X=X+H
      Y=H/2.*PQF(X,1)
      A(I)=1.-Y
      C(I)=1.+Y
      B(I)=HH*PQF(X,2)-2.
 20   D(I)=HH*PQF(X,3)
      D(1)=D(1)-YA*A(1)
      D(N)=D(N)-YB*C(N)
```

```
         C(1)=C(1)/B(1)
         DO 25 I=2,N
         B(I)=B(I)-A(I)*C(I-1)
      25 C(I)=C(I)/B(I)
         D(1)=D(1)/B(1)
         DO 30 I=2,N
      30 D(I)=(D(I)-A(I)*D(I-1))/B(I)
         NM=N-1
         DO 35 I=1,NM
         J=NM-I+1
      35 D(J)=D(J)-C(J)*D(J+1)
         WRITE(7,40)N,(I,I=1,N1)
      40 FORMAT(///5X,'BROJ MEDJUTACAKA N='
        1 ,I3///5X,'I',6X,'0',9I10)
         DO 45 I=1,N
         C(I)=DG+H*FLOAT(I)
      45 B(I)=PQF(C(I),4)
         WRITE(7,50)DG,(C(I),I=1,N),GG
         WRITE(7,55)YA,(D(I),I=1,N),YB
         WRITE(7,65)YA,(B(I),I=1,N),YB
      50 FORMAT(/5X,'X(I)',10(F6.2,4X))
      55 FORMAT(/5X,'Y(I)',10F10.6)
      65 FORMAT(/5X,'YEGZ',10F10.6)
         GO TO 10
      60 CLOSE(7)
         CLOSE(8)
         STOP
         END
```

Note that this program is so realized that number of inner points N is read on input. In case when $N = 0$ program ends. Also, in program is foreseen tabulating of exact solution in observing points, as control. It is clear that last has meaning for scholastic examples when solution is known. So, for example, for boundary problem

$$y'' - 2xy' - 2y = -4x; \quad y(0) = 1, \; y(1) = 1 + e \cong 3.7182818,$$

the exact solution is $y = x + exp(x^2)$.

For this contour problem function subroutine for defining functions p, q, f, as for as for exact solution, is named PQF. In case N=4, we got the results given in continuation.

```
         FUNCTION PQF(X,M)
         GO TO (10,20,30,40),M
      10 PQF=-2.*X
         RETURN
      20 PQF=-2.
         RETURN
      30 PQF=-4.*X
         RETURN
      40 PQF=X+EXP(X*X)
         RETURN
         END
```

```
 BROJ MEDJUTACAKA N=  4
 I      0        1        2        3        4        5
X(I)   .00      .20      .40      .60      .80     1.00
Y(I) 1.000000 1.243014 1.576530 2.035572 2.695769 3.711828
YEGZ 1.000000 1.240811 1.573511 2.033329 2.696481 3.711828
```

9.11. Packages for ODEs

Numerous libraries and software packages are available for integrating initial-value ordinary differential equations. Many work stations and main frame computers have such libraries attached to their operating systems.

Many commercial software packages contain routines for integrating initial-value ODEs. One of the oldest and very known among senior scientist is SSP (Scientific Subroutine Package) of IBM. For ODEs it has subroutines `RK1` (integral of first-order differential equation by Runge-Kutta method), `RK2` (integral of first-order differential equation by Runge-Kutta method in tabulated form) using in both subroutines fourth order Runge-Kutta method, and `RKGS` (solution of system of first-order differential equations with given initial values by the Runge-Kutta method) using evaluation by means of fourth order Runge-Kutta formulae in the modification due to Gill. Some of the more prominent packages are `Matlab` and `Mathcad`. More sophisticated packages, such as `IMSL`, `Mathematica`, and `Macsyma` contain also algorithms for integrating initial-value ODEs. The book *Numerical Recipes*([12]) contains numerous subroutines for integrating initial-value ordinary differential equations and the book *Numerical Methods for Engineers and Scientists*([3]) program code for solving single first-order ODEs, higher order ODEs, and systems of first-order ODEs, by using single-point methods, extrapolation methods, and multi-point methods (see Chapter 7, One-Dimensional Initial-Value Ordinary Differential Equations).

Bibliography (Cited references and further reading)

[1] Milovanović, G.V., *Numerical Analysis III*, Naučna knjiga, Beograd, 1988 (Serbian).

[2] Milovanović, G.V., *Numerical Analysis I*, Naučna knjiga, Beograd, 1988 (Serbian).

[3] Hoffman, J.D., *Numerical Methods for Engineers and Scientists.* Taylor & Francis, Boca Raton-London-New York-Singapore, 2001.

[4] Milovanović, G.V. and Djordjević, Dj.R., *Programiranje numeričkih metoda na FORTRAN jeziku.* Institut za dokumentaciju zaštite na radu "Edvard Kardelj", Niš, 1981 (Serbian).

[5] Milovanović, G.V., *Numerical Analysis II*, Naučna knjiga, Beograd, 1988 (Serbian).

[6] Bertolino, M., *Numerička analiza*, Beograd, 1977.

[7] Runge, C., *Über die numerische Auflösung von Differentialgleichungen*, Math. Ann. 46(1985), 167-178.

[8] Kutta, W., *Beitrag zur Näherungsweisen Integration totaler Differentialgleichungen*, Z. Math. Phys. 46(1901), 435-453.

[9] Neun, K., *Neue Methode zur approximativen Integration der Differentialgleichungen einer unabhängigen Veränderlichen*, Z. Math. Phys. 45(1900), 23-38.

[10] Gill, S., *A Process for the step-by-step integration of differential equations in an automatic computing machine*, Proc. Cambridge Phil. Soc. 47(1951), 96-108.

[11] Stoer, J., and Bulirsch, R., *Introduction to Numerical Analysis*, Springer, New York, 1980.

[12] Press, W.H., Flannery, B.P., Teukolsky, S.A., and Vetterling, W.T., *Numerical Recepies - The Art of Scientific Computing.* Cambridge University Press, 1989.

[13] Milovanović, G.V. and Kovačević, M.A., *Zbirka rešenih zadataka iz numeričke analize.* Naučna knjiga, Beograd, 1985. (Serbian).

[14] Ralston,A., *A First Course in Numerical Analysis.* McGraw-Hill, New York, 1965.

[15] Hildebrand, F.B., *Introduction to Numerical Analysis.* Mc.Graw-Hill, New York, 1974.

[16] Gear, C.W., *Numerical Initial Value Problems in Ordinary Differential Equations.* Englewood Cliffs, NJ: Prentice-Hall, 1971.

[17] Acton, F.S., *Numerical Methods That Work* (corrected edition). Mathematical Association of America, Washington, D.C., 1990.

[18] Lambert, J., *Computational Methods in Ordinary Differential Equations*, Wiley, New York, 1973.

[19] Lapidus, L., and Seinfeld, J., *Numerical Solution of Ordinary Differential Equations*, Academic Press, New York, 1971.

[20] Abramowitz, M., and Stegun, I.A., *Handbook of Mathematical Functions.* National Bureau of Standards, Applied Mathematics Series, Washington, 1964 (reprinted 1968 by Dover Publications, New York).

[21] Rice, J.R., *Numerical Methods, Software, and Analysis.* McGraw-Hill, New York, 1983.

[22] Forsythe, G.E., Malcolm, M.A., and Moler, C.B., *Computer Methods for Mathematical Computations.* Englewood Cliffs, Prentice-Hall, NJ, 1977.

[23] Kahaner, D., Moler, C., and Nash, S., 1989, *Numerical Methods and Software.* Englewood Cliffs, Prentice Hall, NJ, 1989.

[24] Hamming, R.W., *Numerical Methods for Engineers and Scientists.* Dover, New York, 1962 (reprinted 1986).

[25] *IMSL Math/Library Users Manual* , IMSL Inc., 2500 City West Boulevard, Houston TX 77042

[26] *NAG Fortran Library*, Numerical Algorithms Group, 256 Banbury Road, Oxford OX27DE, U.K., Chapter F02.

LECTURES

LESSON X

10. Partial Differential Equations - PDE

10.1. Introduction

Partial differential equations (PDEs) arise in all fields of engineering and science. Most real physical processes are governed by partial differential equations. In many cases, simplifying approximations are made to reduce the governing PDEs to ordinary differential equations (ODEs) or even algebraic equations. However, because of the ever increasing requirements for more accurate modelling of physical processes, engineers and scientists are more and more required to solve the actual PDEs that govern the physical problem being investigated. Physical problems are governed by many different PDEs. A few problems are governed by a single first-order PDE. Numerous problems are governed by a system of first order PDEs. Some problems are governed by a single second-order PDE, and numerous problems are governed by a system of second-order PDEs. A few problems are governed by fourth order PDEs. The two most frequent types of physical problems described by PDEs are equilibrium and propagation problems.

The classification of PDEs is most easily explained for a single second order linear PDE of form

$$A\frac{\partial^2 u}{\partial x^2}+B\frac{\partial^2 u}{\partial x\partial y}+C\frac{\partial^2 u}{\partial y^2}+D\frac{\partial u}{\partial x}+E\frac{\partial u}{\partial y}+Fu=G, \tag{10.1.1}$$

where A, B, C, D, E, F, G are given functions which are continuous in area S of plane xOy. The area S is usually defined as inside part of some curve Γ. Of course, the area S can be as finite as well as infinite. Typical problem is finding two times continuous differentiable solution $(x, y) \to u(x, y)$ which satisfies equation (10.1.1) and some conditions on curve (contour) Γ.

Linear PDEs of second order can be classified as eliptic, parabolic and hyperbolic, depending on the sign of the discriminant $B^2 - 4AC$ in given area S, as follows:

1^0 $B^2 - 4AC < 0$ Elliptic
2^0 $B^2 - 4AC = 0$ Parabolic
3^0 $B^2 - 4AC < 0$ Hyperbolic

The terminology elliptic, parabolic, and hyperbolic chosen to classify PDEs reflects the analogy between the form of the discriminant, $B^2 - 4AC$, for PDEs and the form of the discriminant, $B^2 - 4AC$, which classifies conic sections, described by the general second-order algebraic equation

$$Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0,$$

where we for negative, zero, and positive value of discriminant have ellipse, parabola, and hyperbola, respectively. It is easy to check that the Laplace equation

$$\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}=0, \tag{10.1.2}$$

is of elliptic type, heat conduction equation

$$\frac{\partial u}{\partial t} - a^2\frac{\partial^2 u}{\partial x^2} = 0, \tag{10.1.3}$$

is of parabolic type, and wave equation

$$\frac{\partial^2 u}{\partial t^2} - c^2\frac{\partial^2 u}{\partial x^2} = 0 \tag{10.1.4}$$

of hyperbolic type. In this chapter we will show one way for numerical solution of PDEs, for Laplace and wave equation by grid method. In the similar way can be solve heat conduction equation, what we leave to the reader.

10.2. Grid method

Grid method or difference method, or finite-difference grid method, is basic method for solution of equations of mathematical physics (partial equations which appear in physics and science)

Let be given linear PDE

$$Lu = f \tag{10.2.1}$$

and let in area D, which is bounded by curve $\Gamma(D = int\ \Gamma)$, look for such its solution on curve Γ that satisfies given boundary condition

$$Ku = \Psi \quad ((x, y) \in \Gamma). \tag{10.2.2}$$

In application of grid method, at first, one should chose discrete set of points D_h, which belongs to area $\overline{D}(= D \cup \Gamma)$, called grid. Most frequently, in applications is for grid taken family of parallel straight lines $x_i = x_0+ih$, $y_j = y_0+jl$ $(i, j = 0, \pm 1, \pm 2, \ldots)$. Intersection points of these families are called nodes of grid, and h and l are steps of grid. Two nodes of grid are called neighbored if the distance between them along x and y axes is one step only. If all four neighbor nodes of some node belong to area $\overline{D}$, then this node is called interior or inner; in counterpart node of grid D_h is called boundary node. In addition to rectangular grids, in practice are also used other grid shapes.

Grid method consists of approximation of equations (10.2.1) and (10.2.2) using corresponding difference equations. Namely, we can approximate operator **L** by difference operator very simple, by substituting derivative with corresponding differences in inner nodes of grid. Thereby are used the following formulas

$$\frac{\partial u(x_i, y_j)}{\partial x} \cong \frac{u_{i+1,j} - u_{i,j}}{h}$$
$$\frac{\partial u(x_i, y_j)}{\partial y} \cong \frac{u_{i+1,j} - u_{i-1,j}}{2h}$$
$$\frac{\partial^2 u(x_i, y_j)}{\partial x^2} \cong \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2}, \quad \text{etc.}$$

Formulas for partial derivatives in variable y are absolutely symmetric. Approximation of boundary conditions can be in some cases very complicated problem, what depends on form of operator K and contour Γ. At so known boundary conditions of first kind, where $Ku = u$, one practical way for approximation was proposed by L. Collatz and comprises of the following:

Let the closest point from contour Γ to boundary node A be point B and let their distance be δ (see Fig. 10.2.1).

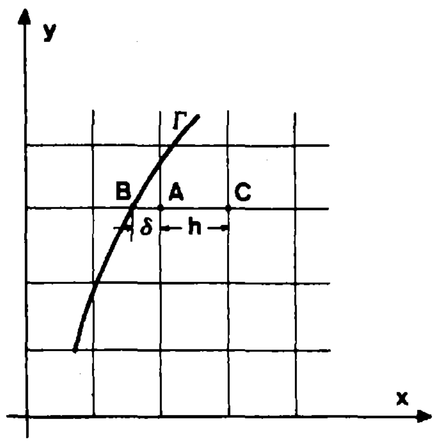


Figure 10.2.1

Based on function values in points B and C, we get by linear interpolation

$$u(A) \cong \frac{h\Psi(B) + \delta u(C)}{h+\delta}.$$

Approximation of boundary condition (10.2.2) in this case comprises of defining equations of above form for every boundary node.

The equations obtained by approximation of equation (10.2.1) and boundary condition (10.2.2) form system of linear equations, by which solution are obtained numerical solutions of given problem.

In further consideration we will give two basic examples.

10.3. Laplace equation

Let it be needfully to find solution of Laplace equation

$$\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \quad ((x,y) \in D),$$

which on the contour of square $D = \{(x,y)|0 < x < 1,\ 0 < y < 1\}$ fulfills given condition $u(x,y) = \Psi(x,y)$ $((x,y) \in \Gamma)$. Let's chose the grid in D_h at which is $l = h = \dfrac{1}{N-1}$, so that grid nods are points $(x_i, y_i) = ((i-1)h,\ (j-1)l)$ $(i,j = 1,\ldots,N)$. The standard difference approximation scheme for solving Laplace equation is of form

$$\frac{1}{h^2}u_{i+1,j} + u_{i-1,j} + u_{i,j-1} - 4u_{i,j} = 0,$$

or

$$u_{i,j} = \frac{1}{4}(u_{i,j+1} + u_{i,j-1} + u_{i-1,j} + u_{i+1,j}).$$

Taking $i,j = 2,\ldots,N-1$ in last equality we get the system of $(N-2)^2$ linear equations. For solving this system usually is used method of simple iterations, or, even more simpler, Gauss-Seidel method.

The corresponding program for solving problem in consideration is of form

```
C==================================================
C  RESAVANJE LAPLACE-OVE JEDNACINA
C==================================================
      DIMENSION U(25,25)
      OPEN(8,FILE='LAPLACE.IN')
      OPEN(5,FILE='LAPLACE.OUT')
      READ(8,4)N
 4    FORMAT(I2)
      M=N-1
      READ(8,1)(U(1,J),J=1,N),(U(N,J),J=1,N),
     1(U(I,1),I=2,M),(U(I,N),I=2,M)
 1    FORMAT(8F10.0)
      DO 10 I=2,M
      DO 10 J=2,M
 10   U(I,J)=0.
      IMAX=0
 20   WRITE(*,5)
  5   FORMAT(5X,'UNETI MAKSIMALNI BROJ ITERACIJA'/
     110X, '(ZA MAX=0 => KRAJ)')
      READ(*,4)MAX
      IF(MAX.EQ.0) GOTO 100
      DO 30 ITER=1,MAX
      DO 30 I=2,M
      DO 30 J=2,M
 30   U(I,J)=(U(I,J+1)+U(I,J-1)+U(I-1,J)+U(I+1,J))/4.
      IMAX=IMAX+MAX
      WRITE(5,65) IMAX,(J,J=1,N)
 65   FORMAT(//26X,'BROJ ITERACIJA JE',I3//17X,
     14(5X,'J=',I2))
      DO 60 I=1,N
 60   WRITE(5,66) I,(U(I,J),J=1,N)
 66   FORMAT(13X,'I =',I2,6F10.4)
      GO TO 20
100   CLOSE(8)
      CLOSE(5)
      STOP
      END
```

For solving system of linear equations we used Gauss-Seidel method with initial conditions $u_{i,j} = 0$ $(i, j = 2, \ldots, N-1)$, whereby one can control number of iterations on input. For N=4 and boundary conditions

$$u_{11} = 0,\ u_{1,2} = 30,\ u_{13} = 60,\ u_{1,4} = 90,$$
$$u_{41} = 180,\ u_{4,2} = 120,\ u_{43} = 60,\ u_{4,4} = 0,$$
$$u_{21} = 60,\ u_{3,1} = 120,\ u_{24} = 60,\ u_{3,4} = 30,$$

the following results are obtained:

```
                BROJ ITERACIJA JE  2
            J= 1      J= 2      J= 3      J= 4
I = 1       .0000   30.0000   60.0000   90.0000
I = 2     60.0000   47.8125   53.9063   60.0000
I = 3    120.0000   83.9063   56.9531   30.0000
I = 4    180.0000  120.0000   60.0000     .0000
                BROJ ITERACIJA JE  7
            J= 1      J= 2      J= 3      J= 4
I = 1       .0000   30.0000   60.0000   90.0000
```

```
I = 2    60.0000    59.9881    59.9940    60.0000
I = 3   120.0000    89.9940    59.9970    30.0000
I = 4   180.0000   120.0000    60.0000      .0000
              BROJ ITERACIJA JE  9
          J= 1       J= 2       J= 3       J= 4
I = 1      .0000    30.0000    60.0000    90.0000
I = 2    60.0000    59.9993    59.9996    60.0000
I = 3   120.0000    89.9996    59.9998    30.0000
I = 4   180.0000   120.0000    60.0000      .0000
              BROJ ITERACIJA JE 10
          J= 1       J= 2       J= 3       J= 4
I = 1      .0000    30.0000    60.0000    90.0000
I = 2    60.0000    59.9998    59.9999    60.0000
I = 3   120.0000    89.9999    60.0000    30.0000
I = 4   180.0000   120.0000    60.0000      .0000
              BROJ ITERACIJA JE 21
          J= 1       J= 2       J= 3       J= 4
I = 1      .0000    30.0000    60.0000    90.0000
I = 2    60.0000    60.0000    60.0000    60.0000
I = 3   120.0000    90.0000    60.0000    30.0000
I = 4   180.0000   120.0000    60.0000      .0000
```

10.4. Wave equation

Consider wave equation

$$\frac{\partial^2 u}{\partial x^2} = \frac{1}{a^2}\cdot\frac{\partial^2 u}{\partial x^2} \tag{10.4.1}$$

with initial conditions

$$u(x,0) = f(x),\ u_1(x,0) = g(x) \quad (0 < x < h) \tag{10.4.2}$$

and boundary conditions

$$u(0,t) = \Phi(t),\ u(b,t) = \Psi(t) \quad (t \geq 0). \tag{10.4.3}$$

Using finite differences, the equation (11.4.1) can be approximated by

$$u_{i+1,j} - 2u_{i,j} + u_{i-1,j} = \frac{1}{r^2}(u_{i,j+1} - 2u_{i,j} + u_{i,j-1}), \tag{10.4.4}$$

where $r = a\frac{1}{h}$ (h and l are steps along x and t axes respectively), and $u_{i,j} \cong u(x_i,t_j)$. Based on first equality in (10.4.2) we have

$$u_{i,0} = f(x_i) = f_i. \tag{10.4.5}$$

By introducing fictive layer $j = -1$, second initial condition in (10.4.2) can simple be approximated using

$$u_i(x_i,0) = g(x_i) = g_i \cong \frac{u_{i,1} - u_{i,-1}}{2l}. \tag{10.4.6}$$

If we put in (10.4.4) $j = 0$ we get

$$f_{i+1} - 2f_i + f_{i-1} - \frac{1}{r^2}(u_{i,1} - 2f_i + u_{i,-1}) = 0,$$

wherefrom, in regard to (10.4.6) it follows

$$u_{i,1} = lg_i + f_i + \frac{1}{2}r^2(f_{i+1} - 2f_i + f_{i-1}),$$

i.e.

$$u_{i,1} = lg_i + (1 - r^2)f_i + \frac{1}{2}r^2(f_{i+1} + f_{i-1}). \tag{10.4.7}$$

On the other hand, from (10.4.4) it follows

$$u_{i,j+1} = \frac{1}{r^2}(u_{i+1,j} + u_{i-1,j}) - u_{i,j-1} + 2(\frac{1}{r^2} - 1)u_{i,j}. \tag{10.4.8}$$

If we put $h = b/N$ and $x_i = (i-1)h$ $(i = 1, 2, \ldots, N+1)$, due to boundary conditions (10.4.3) we have

$$u_{1,j} = \Phi_j, \; u_{N+1,j} = \Psi(t_j) = \Psi_j, \tag{10.4.9}$$

where $j = 0, 1, \ldots$. For determining of solution inside of rectangle $P = \{(x,t)|0 < x < b, 0 < t < T_{max}\}$, maximal value of index j is integer part of T_{max}/l i.e. $j_{max} = M = [T_{max}/l]$.

Based on equalities (10.4.5), (10.4.7), (10.4.8), (10.4.9) the approximate solutions of given problem in grid nodes of rectangle P, are simple to obtain. This algorithm is coded in the following program.

```
C=====================================================
C RESAVANJE PARCIJALNE DIF. JED. HIPERBOLICNOG TIPA
C=====================================================
      DIMENSION U(3,9)
      OPEN(8,FILE='TALAS.IN')
      OPEN(5,FILE='TALAS.OUT')
      READ (8,5)N,A,B,R,TMAX
 5    FORMAT(I2,4F5.2)
      N1=N+1
      WRITE (5,10) (I,I=1,N1)
 10   FORMAT(10X,1HJ,<N+1>(4X,'U(',I1,',J)')/)
      H=B/FLOAT(N)
      EL=R*H/A
      M=TMAX/EL
      T=0.
      DO 15 K=1,2
      U(K,1)=FF(T,B,3)
      U(K,N1)=FF(T,B,4)
 15   T=T+EL
      X=0.
      R2=R*R
      DO 20 I=2,N
      X=X+H
      U(1,I)=FF(X,B,1)
 20   U(2,I)=EL*FF(X,B,2)+(1.-R)*U(1,I)
      DO 25 I=2,N
 25   U(2,I)=U(2,I)+R2/2.*(U(1,I+1)+U(1,I-1))
      J=0
 30   WRITE(5,35)J,(U(1,I),I=1,N1)
 35   FORMAT(7X,I5,<N1>F10.4)
```

```
      IF(J.EQ.M)GO TO 50
      J=J+1
      U(3,1)=FF(T,B,3)
      U(3,N1)=FF(T,B,4)
      DO 40 I=2,N
   40 U(3,I)=(U(2,I+1)+U(2,I-1))/R2-U(1,I)-2.
     1*(1./R2-1.)*U(2,I)
      T=T+EL
      DO 45 I=1,N1
      U(1,I)=U(2,I)
   45 U(2,I)=U(3,I)
      GO TO 30
   50 CLOSE(5)
      CLOSE(5)
      STOP
      END
```

Note that the values of solution in three successive layers $j-1, j, j+1$, are stored in first, second, and third row of matrix U, respectively. .

Functions f, g, Φ, Ψ are defined by function subroutine FF for I=1,2,3,4, respectively.

In considered case for $a = 2$, $b = 4$, $T_{max} = 6$, $f(x) = x(4-x)$, $g(x) = 0$, $\Phi(t) = 0$, $\Psi(t) = 0$, $N = 4$, and $r = 1$, subroutine FF and corresponding output listing with result have the following form:

```
      FUNCTION FF(X,B,I)
      GO TO(10,20,30,40),I
   10 FF=X*(B-X)
      RETURN
   20 FF=0.
      RETURN
   30 FF=0.
      RETURN
   40 FF=0.
      RETURN
      END
```

```
 J    U(1,J)     U(2,J)     U(3,J)     U(4,J)     U(5,J)
 0     .0000     3.0000     4.0000     3.0000      .0000
 1     .0000     2.0000     3.0000     2.0000      .0000
 2     .0000      .0000      .0000      .0000      .0000
 3     .0000    -2.0000    -3.0000    -2.0000      .0000
 4     .0000    -3.0000    -4.0000    -3.0000      .0000
 5     .0000    -2.0000    -3.0000    -2.0000      .0000
 6     .0000      .0000      .0000      .0000      .0000
 7     .0000     2.0000     3.0000     2.0000      .0000
 8     .0000     3.0000     4.0000     3.0000      .0000
 9     .0000     2.0000     3.0000     2.0000      .0000
10     .0000      .0000      .0000      .0000      .0000
11     .0000    -2.0000    -3.0000    -2.0000      .0000
12     .0000    -3.0000    -4.0000    -3.0000      .0000
```

10.5. Packages for PDEs

Elliptic PDEs govern equilibrium problem, which have no preferred paths of information propagation. The domain of dependence and range of influence of every point is the entire closed solution domain. Such problems are solved numerically by relaxation methods. Finite difference methods, as typified by five-point method, yield a system of finite difference equations, called the system equations, which have to be solved by relaxation methods. The successive-over-relaxation method (SOR)method is generally method of choice. The multigrid method (Brandt, 1977) shows the best potential for rapid convergence. Nonlinear PDEs yield nonlinear finite difference equations (FDE). System of nonlinear FDEs can be very difficult to solve. The multigrid method can be applied directly to nonlinear PDEs. Three-dimensional PDEs are approximated simply by including the finite difference approximations of the spatial derivatives in the third direction. The relaxation techniques used to solve two-dimensional problems generally can be used to solve three-dimensional problems, at the expense of a considerable increase of computational time.

Parabolic PDEs govern propagation problems which have an infinite physical information propagation speed. They are usually solved numerically by marching method. Explicit finite difference methods, like FTCS (Forward-Time Centered-Space method, see [3], pp. 633-635) are conditionally stable and require relatively small step size in the marching direction to satisfy the stability criteria. Implicit methods, like BTCS (Backward-Time Centered-Space method, see [3], pp. 635-637) are unconditionally stable. The marching step size is restricted by accuracy requirements, not stability requirements. For accurate solution of transient problems, the marching step-size for implicit methods cannot be very much larger than the stable step size for explicit methods. Consequently, explicit methods are generally preferred for obtaining accurate transient solutions. Asymptotic steady state solutions can be obtained very efficiently by BTCS method with a large marching step size. Nonlinear PDEs can be solved directly by explicit methods. When solved by implicit methods, system of nonlinear FDEs must be solved. Multidimensional problems can be solved directly by explicit methods. When solved by implicit methods, large banded systems of FDEs result.

Hyperbolic PDEs govern propagation problems, which have a finite physical information propagation speed. They are solved numerically by marching method. Explicit finite difference methods are conditionally stable and require a relatively small step size in marching direction to satisfy the stability criteria. Implicit methods, as typified by the BTCS method, are unconditionally stable. The marching step size is restricted by accuracy requirements, not stability requirements. For accurate solution of transient problems, explicit methods are recommended. When steady state solutions are to be obtained as the asymptotic solution in time of an appropriate unsteady propagation problem, BTCS with a large step size is recommended.

Nonlinear PDEs can be solved directly by explicit methods. When solved by implicit methods, system of nonlinear FDEs must be solved. Multidimensional problems can be solved directly by explicit methods. When solved by implicit methods, large banded systems of FDEs result.

Numerous libraries and software packages are available for integrating the Laplace and Poisson equations, diffusion type (i.e. parabolic) and convection type (i.e. hyperbolic) PDEs. Many work stations and main frame computers have such libraries attached to their operating systems.

Many commercial software packages contain routines for integrating Laplace and Poisson equations. Due to the wide variety of elliptic, parabolic, and hyperbolic PDEs governing physical problems, many PDE solvers (programs) have been developed.

The book *Numerical Recipes* ([7]) contains a lot of algorithms for integrating PDEs. For some of them is given programming code in `Fortran` (available also in `C`). Survey of methods for solving different classes of PDEs accompanied with algorithms, from which some are codded, is given in book *Numerical Methods for Engineers and Scientists* ([3], Chapter 9, 10 and 11).

Bibliography (Cited references and further reading)

[1] Milovanović, G.V., *Numerical Analysis III*, Naučna knjiga, Beograd, 1988 (Serbian).

[2] Milovanović, G.V., *Numerical Analysis I*. Naučna knjiga, Beograd, 1988 (Serbian).

[3] Hoffman, J.D., *Numerical Methods for Engineers and Scientists*. Taylor & Francis, Boca Raton-London-New York-Singapore, 2001.

[4] Milovanović, G.V. and Djordjević, Dj.R., *Programiranje numeričkih metoda na FORTRAN jeziku.* Institut za dokumentaciju zaštite na radu "Edvard Kardelj". Niš, 1981 (Serbian).

[5] Milovanović, G.V., *Numerical Analysis II.* Naučna knjiga, Beograd, 1988 (Serbian).

[6] Stoer, J., and Bulirsch, R., *Introduction to Numerical Analysis*, Springer, New York, 1980.

[7] Press, W.H., Flannery, B.P., Teukolsky, S.A., and Vetterling, W.T., *Numerical Recepies - The Art of Scientific Computing.* Cambridge University Press, 1989.

[8] Milovanović, G.V. and Kovačević, M.A., *Zbirka rešenih zadataka iz numeričke analize.* Naučna knjiga, Beograd, 1985. (Serbian).

[9] Ralston,A., *A First Course in Numerical Analysis.* McGraw-Hill, New York, 1965.

[10] Hildebrand, F.B., *Introduction to Numerical Analysis.* Mc.Graw-Hill. New York. 1974.

[11] Acton, F.S., *Numerical Methods That Work* (corrected edition). Mathematical Association of America, Washington, D.C., 1990.

[12] Abramowitz, M., and Stegun, I.A., *Handbook of Mathematical Functions.* National Bureau of Standards, Applied Mathematics Series, Washington, 1964 (reprinted 1968 by Dover Publications, New York).

[13] Rice, J.R., *Numerical Methods, Software, and Analysis.* McGraw-Hill, New York, 1983.

[14] Forsythe, G.E., Malcolm, M.A., and Moler, C.B., *Computer Methods for Mathematical Computations.* Englewood Cliffs, Prentice-Hall, NJ, 1977.

[15] Kahaner, D., Moler, C., and Nash, S., 1989, *Numerical Methods and Software.* Englewood Cliffs, Prentice Hall, NJ, 1989.

[16] Hamming, R.W., *Numerical Methods for Engineers and Scientists.* Dover, New York, 1962 (reprinted 1986).

[17] Ferziger, J.H., *Numerical Methods for Engineering Applications.* Stanford University, John Willey & Sons, Inc., New York, 1998

[18] Pearson, C.E., *Numerical Methods in Engineering and Science.* University of Washington, Van Nostrand Reinhold Company, New York, 1986.

[19] Stephenson, G. and Radmore, P.M., *Advanced Mathematical Methods for Engineering and Science Students.* Imperial College London, University College, London Cambridge Univ. Press, 1999

[20] Ames, W.F., *Numerical Methods for Partial Differential Equations.* 2nd ed. Academic Press, New York, 1977.

[21] Richtmyer, R.D. and Morton, K.W., *Difference Methods for Initial Value Problems.* 2nd ed., Wiley-Interscience, New York, 1967.

[22] Mitchell, A.R., and Griffiths, D.F. , *The Finite Difference Method in Partial Differential Equations.*, Wiley, New York, 1980.

[23] *IMSL Math/Library Users Manual* , IMSL Inc., 2500 City West Boulevard, Houston TX 77042

[24] *NAG Fortran Library.* Numerical Algorithms Group, 256 Banbury Road, Oxford OX27DE, U.K., Chapter F02.

LECTURES

LESSON XI

11. Integral Equations

11.1. Introduction

In spite the fact that integral equations are almost never treated in numerical analysis textbooks, there is a large and growing literature on their numerical solution. One reason for the sheer volume of this activity is that there are many different kinds of equations, each with many different possible pitfalls. Often many different algorithms have been proposed to deal with a single case. There is a close correspondence between linear integral equations, which specify linear, integral relations among functions in an infinite-dimensional function space, and plain old linear equations, which specify analogous relations among vectors in a finite-dimensional vector space. This correspondence lies at the heart of most computational algorithms, as we will see in program realization of their numerical solution.

The equation

$$y(x) = f(x) + \lambda \int_a^b K(x,t)y(t)\,dt, \tag{11.1.1}$$

where f i K are known functions, y unknown function, and λ numerical parameter, is called a Fredholm integral equation of second kind. Fredholm equations involve definite integrals with fixed upper and lower limits.

The function in two variables K is called kernel of integral equation (11.1.1). In our considerations we will always suppose that kernel is defined and continuous on $D = \{(x,t)|a \le x \le b,\ a \le t \le b\}$.

If $f(x) \not\equiv 0$, the equation (11.1.1) is called inhomogeneous, and in case when $f(x) \equiv 0$, equation is homogenous.

Integral equation of form

$$f(x) + \lambda \int_a^b K(x,t)y(t)\,dt = 0$$

is called Fredholm integral equation of first kind. This equation can be written in analogous form, as matrix equation

$$\mathbf{K} \cdot \vec{y} = -\vec{f}$$

which solution is

$$\vec{y} = -\mathbf{K}^{-1} \cdot \vec{f},$$

and $\mathbf{K}^{-1}$ is matrix inverse. Both equation are solvable when function f and $\vec{f}$ are nonzero, respectively (the homogeneous case with f=0 is almost never useful), and $K(\mathbf{K})$ is invertible.

The analogous matrix form of Fredholm equation of second kind (11.1.1) is

$$(\mathbf{K} - \frac{1}{\lambda}\mathbf{I}) \cdot \vec{y} = -\frac{\vec{f}}{\lambda}.$$

Again, if f or $\vec{f}$ is zero, then the equation is said to be homogeneous. If the kernel $K(x,t)$ is bounded, then, like in matrix form, the equation (11.1.1) has the property that its homogeneous form has solutions for at most a denumerably infinite set $\lambda \doteq \lambda_n$, $n = 1, 2\ldots$, the eigenvalues. The corresponding solutions $y_n(x)$ are the eigenfunctions. The eigenvalues are real if the kernel is symmetric. In the inhomogeneous case of nonzero f or $\vec{f}$, both equations are solvable except when λ or $1/\lambda$ is an eigenvalue - because the integral operator (or matrix) is singular then. In integral equations this dichotomy is called the Fredholm alternative.

Fredholm equations of the first kind are often extremely ill-conditioned. Applying the kernel to a function is generally a smoothing operation, so the solution, which requires inverting the operator, will be extremely sensitive to small changes or errors in the input. Smoothing often actually loses information, and there is no way to get it back in an inverse operation. Specialized methods have been developed for such equations, which are often called inverse problems. The idea is that method must augment the information given with some prior knowledge of the nature of the solution. This prior knowledge is then used, in some way, to restore lost information.

Volterra integral equations of first and second kind are of forms

$$f(x) + \lambda \int_a^x K(x,t)y(t)\,dt = 0$$

and

$$y(x) = f(x) + \lambda \int_a^x K(x,t)y(t)\,dt,$$

respectively. Volterra equations are a special case of Fredholm equations with $K(x,t) = 0$ for $t > x$. Chopping off the unnecessary part of the integration, Volterra equations are written in a form where the upper limit of integration is the independent variable x. The analogous matrix form of Volterra equation of first kind (written out in components) is

$$\sum_{j=1}^{k} K_{kj}y_j = f_k,$$

wherefrom we see that Volterra equation corresponds to a matrix $\mathbf{K}$ that is lower (left) triangular. As we already know, such matrix equations are trivially soluble by forward substitution. Techniques for solving Volterra equations are similarly straightforward. When experimental measurement noise does not dominate, Volterra equations of the first kind tend not to be ill-conditioned. The upper limit to the integral introduces a sharp step that conveniently spoils any smoothing properties of the kernel. The matrix analog of Volterra equation of the second kind is

$$(\mathbf{K} - \mathbf{I}) \cdot \vec{y} = \vec{f},$$

with $\mathbf{K}$ lower triangular matrix. The reason there is no λ in these equations is that in inhomogeneous case (nonzero f) it can be absorbed into K or $\mathbf{K}$, while in the homogeneous case ($f = 0$), there is a theorem that Volterra equations of the second kind with bounded kernels have no eigenvalues with square-integrable eigenfunctions.

We have considered only the case of linear integral equations. The integrand in a nonlinear version of given equations of first kind (Fredholm and Volterra) would be $K(x,t,y(t))$ instead of $K(x,t)y(t)$, and a nonlinear versions of equations of second kind would have an integrand $K(x,t,y(x),y(t))$. Nonlinear Fredholm equations are considerably more complicated than their linear counterparts. Fortunately, they do not occur as frequently in practice. By contrast, solving nonlinear Volterra equations usually involves only a slight modification of the algorithm for linear equations. Almost all methods for solving integral equations numerically make use of quadrature rules, frequently Gaussian quadratures.

11.2. Method of successive approximations

For solving Fredholm equation (11.1.1) it is often used method of successive approximations based on equality

$$y_n(x) = f(x) + \lambda \int_a^b K(x,t)y_{n-1}(t)\,dt \quad (n = 1,2,\ldots), \tag{11.2.1}$$

whereby is taken $y_0 = f(x)$. Namely, if we define sequence of functions $\{\overline{y}_k\}$ by using

$$\overline{y}_0(x) = y_0(x) = f(x),\ \overline{y}_k(x) = \int_a^b K(x,t)\overline{y}_{k-1}(t)\,dt \quad (k = 1,2\ldots),$$

then (11.2.1) can be presented in the form

$$y_n(x) = \sum_{k=0}^{n} \lambda^k \overline{y}_k(x) \quad (n = 1,2,\ldots). \tag{11.2.2}$$

One can show that sequence y_n converges to exact solution of equation (11.1.1) if fulfilled the condition $|\lambda| < \dfrac{1}{M(b-a)}$ where

$$M = \max_{x,t\in[a,b]} |K(x,t)|$$

11.3. Application of quadrature formulas

In order to solve Fredholm equation (11.1.1) let's take quadrature formula

$$\int_a^b F(x)\,dx = \sum_{j=1}^{n} A_j F(x_j) + R_n(F), \tag{11.3.1}$$

where abscissas $x_1,\ldots,x_n$ are from $[a,b]$, A_j are weight coefficients not depending on F, and $R_n(F)$ corresponding remainder term.

If we put in (11.1.1) successively $x = x_i$ $(i = 1,\ldots,n)$, we obtain

$$y(x_i) = f(x_i) + \lambda \int_a^b K(x,t)y(t)\,dt \quad (1 = 1,\ldots,n),$$

wherefrom by using quadrature formula (11.3.1) it follows

$$(11.3.2)\qquad y(x_i) = f(x_i) + \lambda\sum_{k=1}^{n} A_j K(x_i, x_j)\, y(x_j) + R_n(F) \quad (i = 1, \ldots, n),$$

where $F_i(t) = K(x_i, t)\, y(t)$ $(i = 1, \ldots, n)$. By discarding members $R_n(F_i)$ $(i = 1, \ldots, n)$, based on (11.3.2) we get system of linear equations

$$(11.3.3)\qquad y_i - \lambda\sum_{i=1}^{n} A_j K_{ij} y_j = f_i \quad (i = 1, \ldots, n),$$

where we put $y_i = y(x_i)$, $f_i = f(x_i)$, $K_{ij} = K(x_i, x_j)$. System (11.3.3) can also be given in matrix form

$$\begin{bmatrix} 1-\lambda A_1K_{11} & -\lambda A_2K_{12} & \ldots & -\lambda A_nK_{1n} \\ -\lambda A_1K_{21} & 1-\lambda A_2K_{22} & \ldots & -\lambda A_nK_{2n} \\ \vdots & & & \\ -\lambda A_1K_{n1} & -\lambda A_2K_{n2} & \ldots & 1-\lambda A_nK_{nn} \end{bmatrix} \cdot \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{bmatrix}.$$

By solving the obtained system of linear equations in $y_1, \ldots, y_n$, the approximative solution of equation (11.1.1) can be presented in the form

$$(11.3.4)\qquad \tilde{y}(x) = f(x) + \lambda\sum_{j=1}^{n} A_j K(x, x_j)\, y_j.$$

11.4. Program realization

Method explained in previous section will be realized by using generalized Simpson quadrature formula, at which we have

$$h = \frac{b-a}{2m},\ n = 2m+1,\ x_i = a + (i-1)h \quad (i = 1, \ldots, n),$$

$$A_1 = A_{2m+1} = \frac{h}{3},\ A_2 = A_4 = \ldots = A_{2m} = \frac{4h}{3},$$

$$A_3 = A_5 = \ldots = A_{2m-1} = \frac{2h}{3}.$$

For solving system of linear equations (11.3.3) we will use subroutines LRFAK and RSTS The code of subroutines and description of subroutines parameters are given in Chapter 2.

In subroutine FRED is formed system of equations (11.3.3). Parameters in subroutine parameter list are of following meaning:

X - vector of abscissas of quadrature formula;
A - vector of weight coefficients of quadrature formula;
FK - name of function subroutine with function f and kernel K;
PL - parameter λ;
C - matrix of system (11.3.3), stored as vector in columnwise way (column by column);
F - vector of free members in system of equation (11.3.3).
Subroutine code is of form:

```
      SUBROUTINE FRED(X,A,N,FK,PL,C,F)
      DIMENSION X(1), A(1),C(1),F(1)
      IND=-N
      DO 15 J=1,N
      IND=IND+N
      DO 10 I=1,N
      IJ=IND+I
      C(IJ)=-PL*A(J)*FK(X(I),X(J),2)
      IF(I-J)10,5,10
5     C(IJ)=1+C(IJ)
10    CONTINUE
15    F(J)=FK(X(J),X(I),1)
      RETURN
      END
```

Function subroutine FK has a following parameters in the parameter list:
X and T - values of arguments x and t respectively.
M - integer which governs calculation of function f (M=1) and kernel K (M=2) for given values of arguments. Subroutine code is of form:

```
      FUNCTION FK(X,T,M)
      GO TO (10,20), M
10    FK=EXP(X)
      RETURN
 20   FK=X*EXP(X*T)
      RETURN
      END
```

Main program is organized in such a way that at first in FRED is formed system of equation, and then is matrix of system factorized by subroutine LRFAK, what enables solving of system of equations by subroutine RSTS.

Taking as an example equation

$$y(x) = e^x - \int_0^1 xe^{xt}\, y(t)\, dt$$

and M=1,2 (N=3,5), the corresponding results are obtained and presented below main program code. Note that exact solution of given equation is $y(x) = 1$.

```
      EXTERNAL FK
      DIMENSION X(10), A(10), C(100),B(10),IP(9)
      OPEN(8,FILE='FRED.IN')
      OPEN(5,FILE='FRED.OUT')
      READ(8,5)PL,DG,GG
 5    FORMAT(3F5.0)
 10   READ(8,15,END=60) M
15    FORMAT(I2)
      N=2*M+1
      H=(GG-DG)/(2.*FLOAT(M))
      X(1)=DG
      DO 20 I=2,N
 20   X(I)=X(I-1)+H
      Q=H/3.
      A(1)=Q
      A(N)=Q
      DO 25 I=1,M
```

```
25    A(2*I)=4.*Q
      DO 30 I=2,M
30    A(2*I-1)=2.*Q
      CALL FRED(X,A,N,FK,PL,C,B)
      CALL LRFAK(C,N,IP,DET,KB)
      IF(KB) 35,40,35
 35   WRITE(5,45)
 45   FORMAT(1H0,'MATRICA SISTEMA SINGULARNA'//)
      GO TO 60
40    CALL RSTS(C,N,IP,B)
      WRITE(5,50)(B(I),I=1,N)
 50   FORMAT(/5X,'RESENJE'//(10F10.5))
      GO TO 10
60    CLOSE(5)
      CLOSE(8)
      STOP
      END
```

```
  RESENJE
1.00000   0.94328   0.79472
  RESENJE
1.00000   1.00000   1.00000   1.00000   0.99998
```

Bibliography (Cited references and further reading)

[1] Milovanović, G.V., *Numerical Analysis III*, Naučna knjiga, Beograd, 1988 (Serbian).

[2] Milovanović, G.V., *Numerical Analysis I*, Naučna knjiga, Beograd, 1988 (Serbian).

[3] Milovanović, G.V. and Djordjević, Dj.R., *Programiranje numeričkih metoda na FORTRAN jeziku.* Institut za dokumentaciju zaštite na radu "Edvard Kardelj", Niš, 1981 (Serbian).

[4] Milovanović, G.V., *Numerical Analysis II*, Naučna knjiga, Beograd, 1988 (Serbian).

[5] Press, W.H., Flannery, B.P., Teukolsky, S.A., and Vetterling, W.T., *Numerical Recepies - The Art of Scientific Computing.* Cambridge University Press, 1989.

[6] Delves, L.M., and Mohamed,J.L., *Computational Methods for Integral Equations.* Cambridge University Press, Cambridge, 1985.

[7] Linz, P., *Analytical and Numerical Methods for Volterra Equations.* S.I.A.M., Philadelphia:, 1985.

[8] Atkinson, K.E., *A Survey of Numerical Methods for the Solution of Fredholm Integral Equations of the Second Kind.* S.I.A.M., Philadelphia, 1976.

[9] Smithies, F., *Integral Equations.* Cambridge University Press, Cambridge,1958.

[10] Kanwal, R.P., *Linear Integral Equations.* Academic Press, New York, 1971.

[11] Green, C.D., *Integral Equation Methods* . Barnes & Noble, New York, 1969.

[12] Stoer, J., and Bulirsch, R., *Introduction to Numerical Analysis*, Springer, New York, 1980.

[13] Ralston,A., *A First Course in Numerical Analysis.* McGraw-Hill, New York, 1965.

[14] Hildebrand, F.B., *Introduction to Numerical Analysis.* Mc.Graw-Hill, New York, 1974.

[15] Acton, F.S., *Numerical Methods That Work* (corrected edition). Mathematical Association of America, Washington, D.C., 1990.

[16] Kahaner, D., Moler, C., and Nash, S., 1989, *Numerical Methods and Software.* Englewood Cliffs, Prentice Hall. N.J. 1989.

[17] *IMSL Math/Library Users Manual* . IMSL Inc., 2500 City West Boulevard, Houston TX 77042

[18] *NAG Fortran Library,* Numerical Algorithms Group, 256 Banbury Road. Oxford OX27DE, U.K., Chapter F02.