# POLYE SYSTEM. COMPUTER IMPLEMENTATION OF THE R-FUNCTIONS METHOD

September 12, 2016

# Contents

# List of Tables

# Chapter 1

# PROBLEM - ORIENTED LANGUAGE FOR DESCRIPTION OF BOUNDARY VALUE PROBLEMS

## 1.1 General characteristic of the RL language

At present there exist several hundred languages intended for automation of programming and solution of various scientific and engineering problems. The multitude of developed and being developed languages is accounted for by the great variety of types of computers and classes of problems being solved on them. Of special interest are problem-oriented languages and systems which reflect the peculiarities of the problems being solved and the methods therefore, and which do not require detailed synthesizing of the algorithms of their solution. The users of such systems need only to specify the problem formulation, the initial data and the required form of presentation of the results. In other words, an expert in some field of knowledge does not solve the problems of detailed specification of algorithms and need not develop the sequence of operators for solving the stated problem. He must only describe the properties of the objects being considered and the computational procedures in a professional language inherent to the given field of application.

The problem oriented language RL and the POLYE system are based on the methods and tools of the $R$-functions theory, extending the capabilities of variational and projection methods of solving boundary problems in complex-shape domains. Such complex of program and language facilities.is used effectively for solving boundary problems in mathematical physics, and to develop application program packages for solving the problems of heat conduction, theory of elasticity, theory of plates and shells, electrodynamics, etc.

The input language of the POLYE system is oriented for users of different qualification:

for developer of software, the input language provides the entire spectrum of facilities (all the standard facilities of the operational system, problem-oriented languages, a special macroprocessor);

for users non-programmers, familiar with the methods of solving boundary problems used by the system, the input language comprises a simplified set of special operators;

for experts familiar with the mathematical statement of boundary value problems, but not familiar with the methods of their solution, the input language consists of one or two operators, with the help of which one or other parameter characterizing the domain shape, boundary conditions, field exciters, etc. is modified;

for users-engineers unfamiliar with the POLYE system, a special input language can be set up in terms which are characteristic for the given speciality.

The input language of the POLYE system allows to describe all the necessary information about the boundary value problem in mathematical physics.

Information about the general problem equations contains data in analytical form about the kind of differential equations, space dimensions, number of sought functions or sought vector- functions, linearity or non-linearity of equation coefficients, field exciters and shape of the domains of their action, shape of subdomains with different media characteristics, etc.

Information about the boundary and initial conditions of the problem is also analytical and is accompanied by data about the boundary and initial conditions, their linearity or non-linearity, shapes of subdomains with different media and their characteristics. Besides, the boundary conditions (depending on the use of the $R$-functions theory constructive means) may be accompanied by additional information: about the approximation means and their carriers or regions of operation, boundary problem solution structures and the details of their construction, etc.

Information about the shape of the domains and sections of its boundaries are specified by logical formulas and data about the base subdomains. The need for this information is accounted for by necessity of solving boundary problems in complex-shape domains and used both for constructing approximate solutions and for defining domains where exciters of a field or function, characterizing heterogeneous media and others, are effective.

Information about the solution method is intended for forming an "instrumental base" for solving a boundary value problem. It contains information about the methods of reducing a boundary problem to an algebraic one, the methods of solving a system of algebraic equations or finding the eigenvalues and eigenvectors. Besides, each of the methods is accompanied by additional information, e.g. about the methods of the integration over a domain or a contour and the formation of the Ritz matrix, etc.

Information about the computational results determines the output of results on the basis of the result obtained and contains data about the kinds of computational formulas, subdomains, in which it is required to find the sought-for functions, and also includes indications how the results are to be presented (tables, contour lines, axonometric projections, graphs, etc.).

Users who master the POLYE system for the first time, distinguish its following advantages:

high level of resemblance of the input language to common ways of specifying the information about the problem being solved;

easy of using the language and its simple mastering;

no programming in the usual sense for practically all categories of users;

convenient language facilities for description of the geometric and analytical components of the boundary problems;

wide possibilities for development and creation of new high-level languages and application program packages;

comprehensive analysis and availability of developed means of debugging and diagnostics;

high adaptability of the POLYE system to various computers;

high quality of working programs (compactness, speed, etc.).

The users of the POLYE system are practically free from programming. They need only to describe the boundary problems in a high-level language in which the program is automatically synthesized. This significantly increases the productivity of researchers to simulate various physical-mechanical fields.

## 1.2 Components and basic constructions of the language

The possibilities of the problem-oriented language RL, when preparing a program for solving boundary value problems, shall be considered in the sequel. It is expedient to characterize preliminary its one generalized property which should be related to the basic results of developing the RL language.

The problem-oriented language RL is substantially oriented to the methods of the $R$-functions theory of solving boundary value problems. Firstly, the language RL is very convenient for describing geometric information and transforming it to the analytical form. Secondly, the programs in the RL language for realizing the structure of a solution determining the kind of sought-for solution and the whole course of the computational process in the system are presented in the form of simple mathematical formulas. Finally, the RL language is used for writing the algorithms for solution of a series of interrelated boundary value problems by the projection or variation methods in combination with the $R$-functions method. So the RL language can be considered as a fairly good means of user interaction with the system when solving a wide class of boundary value problems in mathematical physics.

All the language constructions are expressed with the help of a basic set of symbols comprising 31 Cyrillic and 26 Latin characters, 10 numerals and special symbols. The program text is divided into lexical units, being identifiers (including keywords), numbers and limiters.

Identifiers are used for designating simple variables, arrays, corteges, modules and others. They can be selected arbitrarily. The identifier length is not more than 5 symbols. The keywords are used for determining the operations being performed, data properties, purpose of names, etc. The keywords have standard identifiers which cannot be used in another sense. Comments begin with the symbol $>$ and continue to a semicolon. Comments do not influence the program sense and substantially facilitate its understanding and analysis.

In the language the elementary parts of the program, describing the algorithmic processes, are expressions. An expression represents an arbitrary sequence of operands (constants and variables) separated by the operations symbol $(+, -,$ etc.) and parentheses, so as to form a sensible mathematical expression. Any expression can be classified as an arithmetic (scalar) expression, expression on corteges or their arrays (cortege expressions). Expression on corteges or their arrays is a cortege or array of corteges which compute by expanding the expression into a series of scalar expressions on the components of the unions.

The functions define separate numeric or cortege values. The functions have standard identifiers which cannot be used in another sense. Practically all the functions, admissible in FORTRAN , can be used in the arithmetic expressions. The standard functions (Table 1.1) can be used in expressions on differential corteges or there arrays, issuing a differential cortege (array of differential corteges) as a result. In this table NAME means the name of the function for which the superposition is realized, N - the number of coefficients.

An expression is the rule for calculating values. In the case of scalar expressions, this value is obtained by executing an arithmetic operation $( +, -, *, /, ** )$ on actual numeric values of the primary expressions. An expression on differential corteges is a separate variable or expression including, at least, one operand - and a differential cortege. All operations in expressions on arrays of differential corteges are performed on an element-by-element basis.

The list of basic operations in cortege expressions is extended by additional means for implementing $R$-operations $(\&, !)$. The operation of the joining of arrays of differential corteges (#) results to the array

| DEFINITION | FUNCTION |
|---|---|
| Natural logarithm | LN(X) |
| Square root | SQRT(X) |
| Exponent | EXP(X) |
| Sines | SIN(X) |
| Cosines | COS(X) |
| Tangent | TG(X) |
| Hyperbolic sines | SH(X) |
| Hyperbolic cosines | CH(X) |
| Arc sin | ARSIN(X) |
| Arc tg | ARTG(X) |
| Function normalization | NORMA(X) |
| Inverse value | INV(X) |
| (x+abs(x))/2 | REG(X) |
| Operator D1 | D1(X,Y) |
| Operator T1 | T1(X,Y) |
| Derivative over X | DX(X) |
| Derivative over Y | DY(X) |
| Sending | AUT(X) |
| xy/(x+x) | MULDIV(X,Y) |
| Gluing (xb+ya)/(x+y) | PASTE(X,Y,A,B) |
| Superposition | SU(X,Y,NAME) |
| Sum | SUM(N,U0,U1) |

Table 1.1 Cortege functions

of differential corteges which consist of two initial arrays.

The operations within one expression are performed from left to right with account of the following order of significance: raising to n-th power ($**$), multiplicative operations ($*, /, \&$), additive operators ($+, -, !$), joining of arrays of differential corteges ($\#$). The desirable order of executing the operations within an expression may be specified by arranging the parentheses respectively.

## 1.3 Language operators

Units of operations in a language are known as operators. They may be united into the following classes: descriptive, transfer of data and computations, program structures, special, output of results.

The descriptions operator is intended for specifying the name attributes. It is used for describing geometric and analytic objects ($LINE$, $CIRCLE$, $POL1$ and others). A list of keywords for describing variables is given in Table 1.2.

Internal moving of data includes in assigning an expression value to a specified variable. An expression may be a constant or variable, or it may be an expression which specifies the necessary computations. The assignment operator is identified by symbol $=$ . The l.h. variable may be the name of a cortege or array of differential corteges; an expression may produce a scalar or cortege value. Hence, the assignment operator may be used to move data aggregates in the same manner as scalar elements.

Program structure operators are used for delineating the program into sections and groups, being the following ones: $DECLARE$, $OMEGA$, $FUNCTION$, $PROGRAM$, $FORTR$
The program in the system input language consists of sections ordered as follows: section of descriptions; section of operators; section of $PROGRAMS$; section of $FORTRAN$ (if necessary); section of $VALUE$. An indication of the beginning of a section is the program structure operator, and the section ends with the operator $END$ or the indication of the beginning of a new section.

Operator $DECLARE$ serves as an indication of the beginning of the program descriptions section, after which there follows the list of description operators. The descriptions section should be first in the

| DEFINITION | NAME |
|---|---|
| Straight normalized line | LINE |
| Circumference normalized (plus inside) | CIRCL |
| Circumference normalized (minus inside) | MCIRCL |
| Strip arbitrary | BAND |
| Strip parallel to axis X (plus inside) | BANDX |
| Strip parallel to axis Y (plus inside) | BANDY |
| Strip parallel to axis X (minus inside) | MBANDX |
| Strip parallel to axis Y (minus inside) | MBANDY |
| Parabola arbitrary | PRBL |
| Parabola along axis X | PRBLX |
| Parabola along axis Y | PRBLY |
| Ellyps | ELLY |
| Ellyps along axis X | ELLYX |
| Ellyps along axis Y | ELLYY |
| Hyperbola arbitrary | HIPER |
| Hyperbola along axis X | HIPERX |
| Hyperbola along axis Y | HIPERY |
| Polynom of the 0 power | POL0 |
| Polynom of the 1-st power | POL1 |
| Polynom of the 2-nd power | POL2 |
| System of power polynoms | SF |
| System of harmonic polynoms | GARM |
| System of Chebyshev's polynoms | SFTH |
| System of splines | SPLI |
| System of zeros | SF0 |
| Subroutine cortege | MODUL |
| Subroutine array of corteges | MODULC |
| Cortege | CORTEG |
| Array of corteges | CORRAY |
| Array of words | ARRAY |

Table 1.2 Key words for describing variables

program.

Operator $OMEGA$ is an indication of the beginning of the section wherein the geometric information about the domain form and the sections of its boundary is described. This is followed by a list of operators comprising computational operators. The expressions in the r.h. side of these operators are expressions on differential corteges. It should be noted that this section is obligatory in the program. The set of operators of this section comprises a separate program block, viz. a subroutine with a name defined by the variable in the left side of the last assignment operator. The name of this subroutine can be used in other subroutines as an operand in cortege expressions. They program can be called also by the name $OMEGA$.

Operator $FUNCTION$ serves as an indication of the beginning of the section containing information about some analytical components of the problem solution structure. It is wanted or required one may organize sections as much as needed (vector problems, regional structures, refinement of the solution, etc.). The set of operators of this section comprises a separate program unit - $MODUL$, whose name appears only once in the left side of the last assignment operator. The this name can be used in other subroutines as an operand in cortege expressions.

Operator $PROGRAM$ serves as an indication of the beginning of the sequence of operators for forming a matrix, solving system and output of computation results. Operator $FORTRAN$ is an indication of the beginning of the section which contains complete FORTRAN subroutines. Operator $NAMES$ may be encountered in the subroutine. It is replaced in the FORTRAN subroutine by the description of the common block $DATA$ which gives access to all variables described in the descriptions section. This section is to be used only for access to these variables from FORTRAN subroutine, otherwise it is better to enter it to the library in the usual way.

Operator $VALUE$ is the indication of the beginning of the sequence of sentences for describing the values of variables and formation of data for calculating different variants of the problem. Operator $END$ is the indication of the section end.

Special operators are used only in the section $PROGRAM$ for organizing the processes of formation of matrices, solving systems of linear algebraic equations, output of calculation results, program control, and so forth (Table 1.3).

The integration operator is used for computing integrals when forming systems of linear algebraic equations, for out-put of results, etc. The operator may have two or three parameters: first - array name ($ARRAY$), containing information necessary for computing integrals (the domain of the integration, order of quadrature formula), the second and third are the names of the functions under the integral signs. If the third parameter is absent, the operator is computed for one function, otherwise computation is performed for two functions at the same time.

The operator for generating the spline nodes computes the integration nodes, if splines are used as the coordinate functions. The first parameter is an array containing information about the spline nodes, the second and the third are the names of the functions under the integral sign which are used for computing the coefficients of the system of linear algebraic equations.

The approximation operator is used for treating the numeric values of the arguments and functions when forming systems of linear algebraic equations. The first parameter is the name of the array containing information about the approximation nodes and values of functions. The second and the third parameters are the names of the functions under the integral sign which are used for computing the coefficients of the system of equations. Operator $POIS$ is used during spline approximation.

The system solution operator is used for solving the system of linear algebraic equations and finding the indefinite elements of the boundary value problem solution structure. Two parameters are names of functions used for forming the matrix and vectors of the r.h. sides of the system of equations. When using the $FCL$ operator, the functional value is printed out.

The presentation operators are used for output of computation results in the form of tables, graphs, contour lines. The first parameter is the name of the array containing information necessary for presenting the results, then follow the names of the functions which are to be computed and the results printed out.

The functions have the form of an assignment operator, the l.h. side

| KEY WORD | PURPOSE OF OPERATOR |
|---|---|
| GAUSS | Integral calculation as to Gauss |
| GAUSP | Integral calculation as to Gauss in polar co-ordinates |
| GIL | Integral calculation along straight line |
| GIC | Integral calculation along circumference |
| KONTUR | Integral calculation along contour |
| POINT | Generation of spline nodes |
| INSPLI | Generation of spline nodes for complex-shape region |
| POIP | Processing approximation nodes (polynoms) |
| POIS | Processing approximation nodes (splines) |
| SIS | Solving systems of linear algebraic equations |
| SISV | Solving systems for various approximations |
| SISVS | Solving systems with approximation averaging |
| FCL | Printing functional value |
| PRO | Solving system for strip matrix |
| VAL | Solving problem of proper numbers and vectors |
| KORNI | Calculating roots of equations |
| IADRO | Kernel of integral conversion |
| ATA | Increasing current cortege order |
| ATS | Decreasing current cortege order |
| NEW | Jump to forming new matrix |
| KMAT | Determining the number of matrices |
| PREDI | Setting new predicate |
| PR1 | Selecting next proper vector as coefficients |
| PR2 | Calling subroutine |
| MAZ | Clear matrices |
| AMA | Setting new attributes of matrices |
| RESULT | Calling graphical integrator Creating charts, figures, etc. |
| PINT | Printing integral value |
| MATR | Printing matrices |
| KOEF | Printing coefficients |
| TIME | Printing current time value |
| TITLE | Printing heading (character string) |

Table 1.3 Special operators of the language

is the name of the function used as a parameter of other functions as an operand in expressions. The function name may be followed by the module name (in parentheses) which is to be called before computing the function value. The r.h. side of the function is a scalar expression in which all FORTRAN operations and standard functions may be used. Besides, it is allowed to use functions determined in the program and standard operands - identificators $I,\ J,\ X,\ Y$, where:

$I,\ J$ are indicators of the cortege array element;

$X$, $Y$ are the current values of arguments specified in the system.

Note that it is necessary to thoroughly consider where to call the module at a great depth of nesting of functions. For instance, if the names of the modules are specified in each function, then this will lead to repeated computation of the function being sought for.

## 1.4 Description of the values of variables (section of values)

The section $VALUE$ is used for describing the values of variables and formation of different variants of the problem. The sentences have the form of an assignment operator, in whose r.h.side the sequence of numbers and variables should be written. There are two types of sentences: for description of an array of data and for describing different variants of the problem (in the second case the list is put in brackets and consists only of numbers). Sentences are used for formation of values of variables described in the section $DECLARE$ (values are not assigned to variables described by means of $CORTEG$, $CORRAY$, $MODUL$, $ARG$). When describing a variable, identificators may occur in the list, which should be assigned variables later on.

The l.h. part of the operator uses standard identificators $CONST$ and $TABL$, which should be obligatory assigned the following values:

$CONST = K, L, IK, DER, NMC, KM, KK$;

$K$ is the number of indefinite sequences described in $TABL$;

$L$is the number of descriptions of matrices (described in $TABL$);

$IK$ is the number of successive approximations which shall be used for comparing results, averaging, etc. (at splines $IK = 1$);

$DER$ is the maximum order of a differential cortege (maximum order of the derivative $+1$);

$NMC$ is the maximum length of the array of differential corteges;

$KM$ is the number of matrices ($KM = 1 \div 3$);

$KK$ is the number of roots ($1 \div 60$) (at integral transforms).

$TABL = N1, P1, N2, P2, ..., NK, PK, SM1, Q1, SM2, Q2, ..., SML, QL$;

$NI$ is the increased by 1 order of the polynomial in the $I$-th sequence;

$PI$ is four numbers $IXI, IYI, IHXI, IHYI$ ;

$IX$ is initial polynomial order as to $X + 1$ ;

$IY$ is initial polynomial order as to $Y + 1$ (for splines this is the number of nodes as to $X$ and $Y$ );

$IHX$ is the step of changing the polynomial order as to $X$ ;

$IHY$ is the step of changing the polynomial order as to $Y$ (for splines these are zeros);

$SMI$ is the method of storing the matrix (0 -complete, 1 - symmetric, 2 - band );

$QI$ is four number for describing the matrices: $NP1I, KP1I, NP2I, KP2I$ ;

$NP1$ is a number of the sequence, beginning from which the functions are used for forming matrices ;

$KP1$ is the number of sequences, used for forming matrices ;

$NP2$ is the number of the sequence, beginning from which the functions are used for forming vectors ($NP2 = 0$ is one vector);

$KP2$ is the number of sequences, used for forming vectors ;

Information about the assignment of values to different types of data is given in Table 1.4. The arrays of numbers ($ARRAY$) are used in the operators of integration, printing and others, hence for each statement there exist rules for assignment of values (Table 1.5).

## 1.5 Using the system for solving a boundary value problem

To illustrate the possibilities of the $RL$ language, we shall give examples of its use for describing different problems. For this, let us discuss the

| TYPE | LENGTH | KIND OF LIST | DESCRIPTION OF ELEMENTS |
|---|---|---|---|
| LINE | 4 | $x_1, y_1, x_2, y_2$ | co-ordinates of points |
| CIRCL | 3 | $x_0, y_0, r$ | co-ordinates of center |
| MCIRCL | 3 | | and radius |
| BAND | 5 | $x_0, y_0, a, fi$ | co-ordinates of center |
| BANDX | 2 | $y_0, a$ | and half-width |
| BANDY | 2 | $x_0, a$ | |
| MBANDX | 2 | $y_0, a$ | fi -tilt angle towards |
| MBANDY | 2 | $x_0, a$ | axis x |
| PRBL | 5 | $x_0, y_0, x_1, y_1, fi$ | co-ordinates of center |
| PRBLX | 4 | $x_0, y_0, x_1, y_1$ | and arbitrary point |
| PRBLY | 4 | | |
| HIPER | 5 | $x_0, y_0, a, b, fi$ | co-ordinates of center |
| HIPERX | 4 | $x_0, y_0, a, b$ | and dimension of |
| HIPERY | 4 | | half-axis |
| | | | fi - tilt angle |
| ELLY | 5 | $x_0, y_0, a, b, fi$ | co-ordinates of center |
| ELLYX | 4 | $x_0, y_0, a, b$ | a - large half-axis |
| ELLYY | 4 | | b - small half-axis |
| | | | fi - tilt angle |
| POL0 | 1 | $a_0$ | polynom coefficients |
| POL1 | 3 | $a_0, a_1, a_2$ | |
| POL2 | 6 | $a_0, a_1, ..., a_5$ | |
| SF | 5 | $n, x_1, y_1, x_2, y_2$ | rectangle for polynom |
| SFTH | 5 | | normalization |
| SPLI | 5 | | n - number of sequence |
| SF0 | 1 | $n$ | |

Table 1.4 Values of variables

| TYPE | LENGTH | KIND OF LIST | DESCRIPTION OF ELEMENTS |
|---|---|---|---|
| GAUSS | 4n+1 | $k, p_1, ..., p_n$ | k - number of quadrature |
| GAUSP | 5n+1 | $k, o_1, ..., o_n$ | $p_i$ are rectangles $(x_1, y_1, x_2, y_2)$ |
| KONTUR | 5n+1 | $k, o_1, ..., o_n$ | $t_i$ are points $(x_1, y_1)$ |
| GIL | 2n+1 | $k, t_1, ..., t_n$ | $o_i$ - arcs of circumferences |
| GIC | 5n+1 | $k, o_1, ..., o_n$ | $(x_1, y_1, r, fi_1, fi_2)$ |
| | | | $fi_1, fi_2$ -angles in degrees |

Table 1.5 Values of ARRAY variable

problem statement, program texts in the *RL* language, commentaries to operators and the results obtained.

As known, the problem of natural oscillations of an isotropic plate of constant thickness is reduced to the standard problem of eigenvalues for the equation

$$\frac{\partial^4 u}{\partial x^4} + \frac{\partial^4 u}{\partial x^2 \partial y^2} + \frac{\partial^4 u}{\partial y^4} + \lambda u = 0.$$

In case of a rigid attachment, the function of deflections $u$ on the boundary should satisfy the following boundary conditions

$$u = 0 \ , \ \frac{\partial u}{\partial n} = 0 \ on \ \partial\Omega.$$

It is necessary to find the eigenvalues $\lambda$ which make up the frequency spectrum of the plate natural vibrations. To each frequency there correspond the eigen functions $u\,(x, y)$determining some mode of oscillations.

The domain is a square with a triangular cut-out in its upper part (a flag). The coordinates of the lower l.h. and upper r.h. vertices are equal to $-0.5$, $-0.5$, $0.5$, $0.5$ respectively and two variants coordinates of the cut-out point $C : (0, 0);\ (0, 0.4)$.

Print out the eigen numbers and contour lines, determining different modes of plate vibrations.

### 1.5.1 Program text in the RL language

```
1 DECLARE > VIBRATION OF PLATE;
   2 LINE F1,F2,F3;
   3 BANDY F4; SF P;
   4 OMEGA W=(((F1!F2)&F3)&F4);
   5 FUNCTION OM=W; W1=OM*OM; U1=W1*P;
   6 FUNCTION U=SUM(1,0,U1);
   7 PROGRAM
   8 GAUSS(S1,FA); TIME; NEW; ATS; ATS;
   9 GAUSS(S1,FB); TIME; VAL; TIME;
   10 TITLE('EIGEN FORM OF VIBRATION');
   11 PR1; RESULT (R1,FU);
   12 >PR1; >RESULT(R1,FU); >PR1; >RESULT(R1,FU);
   13 FA(U1)=(U1(I,4)+U1(I,6))*(U1(J,4)+U1(J,6));
   14 FB(U1)=U1(I)*U1(J); FU(U)= U;
   15 VALUE CONST=1,1,1,3,55,2,1;
   16 TABL=NP,1,1,2,1,0,1,1,0,0;P=1,AM,BM,A,B;
   17 S1=K,0,BM,A,C,0,C,A,B;
   18 F1=0,C,AM,B; R1=31,AM,BM,A,B;
   19 F2=A,B,0,C; F3=AM,BM,A,BM; F4=0,A;
   20 A=.5; AM=-.5; B=.5; BM=-.5;
   21 C= (0,0.4); NP= 5; K=1;
   22 END
```

### 1.5.2 Commentary to operators

1) Beginning of the descriptions section.

2) Description of straight lines $F1$, $F2$ and $F3$ (normal equations).

3) Description of strip $F4$ parallel to axis $OY$ (normal equation, plus inside). Description of the system of coordinate functions (power polynomials).

4) Beginning of $OMEGA$ section. Computation of the value of the differential cortege $W$ - value of function and its all partial derivatives. Only straight lines $F1$, $F2$, $F3$ and strip $F4$ are used.

5) Beginning of section $FUNCTION$. The values of corteges $OM$ and $W1$ are computed. Then the structure of solution $U1$ and all its partial derivatives ($U1$ - module) are computed.

6) Beginning of section $FUNCTION$. Computation of value $U = U1$. Operator $SUM$ is used to compute the sum.

7) Beginning of section $PROGRAM$.

8) Computation of the value of the matrix A elements. Subroutine $GAUSS$ is accessed for computing integrals over the domain (information about the domain is given by parameter $S1$) of the integrand function $FA$ ( the function is described lower). The operator operation result is matrix $A1$. Computation time is printed out (operator $TIME$). New matrix ($NEW$) is ordered. Reducing the derivative order for further computations by 2 (operator $ATS$ is repeated twice). $S1$ is described in $VALUE$, operator 17.

9) Computation of the values of the matrix $B$ elements. Subroutine $GAUSS$ is accessed for computing integrals over the domain (information about the domain is given by parameter $S1$ - operator 17) of the integrand function $FB$ (the function is described lower). The result of operator operation is matrix $B1$. Problem solution time is printed out. Solution of the problem of eigen numbers and vectors (operator $VAL$). The square of the eigenvalues of $L$ are printed out. The problem solution time is printed out.

10) Heading printout.

11) Retrieval of the first eigenvector for summation (operator $PR1$). Every new usage of operator $PR1$ effects retrieval of the following vector. Output of the contour lines of the sought for function $FU$ is carried out with the help of subroutine $RESULT$. Parameter $R1$ is the information about the results output zone.

12) Commentary line. If symbol > is removed, the second and third modes of oscillations shall be output.

13) Computation of the integrand function values used for forming matrix $A1$ by the Ritz method. The matrix $A1$ elements for the given problem are determined as follows:

$$a_{ij} = \int\limits_{\Omega} \left( \frac{\partial^2 u_{1i}}{\partial x^2} + \frac{\partial^2 u_{1i}}{\partial y^2} \right) \left( \frac{\partial^2 u_{1j}}{\partial x^2} + \frac{\partial^2 u_{1j}}{\partial y^2} \right) d\Omega.$$

14) Computation of the integrand function values used for forming the matrix $B1$ by the Ritz method. The matrix $B1$ elements for the given problem are determined as follows:

$$b_j = \int\limits_{\Omega} u_{1i} u_{1j} d\Omega.$$

Computation of the sought for function $u$ values. Access to subroutine $u$ takes place. The value of $u$ is taken from the differential cortege.

15) Beginning of the section $VALUE$. Standard variable $CONST$ is assigned the following values: 1 - one indefinite sequence, 1 - one matrix description, 1 - one approximation, 3 - maximum order of differential corteges, 55 - maximum length of arrays of corteges, 2- two matrices $A1$ and $B1$, last constant $\neq 1$ when using integral transforms.

16) Standard variable $TABL$ is assigned the following values: $NP$ - power of approximating polynomial, then four numbers determining the sequence structure; 0 - method of matrix storage, then four numbers for describing the matrix and vectors of the r.h. parts; variable $P$ (system of functions) is assigned values. The system is related to the first sequence (here it is only one) and is defined in the rectangle$AM$, $BM$, $A$, $B$.

17) Variable $S1$ is assigned a value (information about the quadrature formula and integration domain). K is the number of the quadrature formula. Then follow quads of numbers determining the integration rectangles $C$ coordinates of the lower l.h. and upper r.h. rectangle vertices).

18) Variable $F1$ (straight line). The values are presented by a quad of numbers - the coordinates of two points on the straight line. Variable $R1$ is the information for presentation of results (number of points for output of results and the output rectangle).

19) Variables $F2$ and $F3$ (straight lines). The variables are presented as a quad of numbers - coordinates of two points of the straight line. Variable $F4$ is a strip whose variables are determined by a pair of numbers (center coordinates and semiwidth).

20) Variables $A$, $AM$, $B$, $BM$ are assigned values.

21) Variable $C$ is assigned a value. Note that two different variants

are formed in this operator (1- point ordinate equal to 0, 2 - point ordinate equal to 0.4). Then variables $NP$ and $K$ are assigned values (polynomial power and quadrature formula order).

22) End of section $VALUE$.

# Chapter 2

# PORTABLE SOFTWARE FOR SIMULATION OF FIELDS

## 2.1 General characteristics of the POLYE system

The specialized system POLYE is intended for solving problems in investigation, analysis and optimization of strain, force, temperature, electromagnetic, filtration, magnetohydro-dynamic and other physical-mechanical fields. The POLYE system is a complex software package intended for solving two-dimensional (plane and axisymmetrical) boundary value problems for partial derivative differential equations with no constraints on the character of the boundary conditions, shape of the domain and the sections of its boundaries.

The mathematical tool is the $R$-functions method allowing to take into account the boundary and geometric information on the analytical level. In world practice there are no methods sufficiently competitive to the given one as to the extended possibilities of variational and projection methods of solving boundary value problems in complex-shape domains when developing, in the first place, special-purpose software for modern computers. As in the case of $R$-functions, the specialized POLYE system is being developed for the first time in world practice. In many scientific centers of the country and abroad software is being developed, which has the same problems orientation though it is based on other methods.

The POLYE system allows to obtain an approximate solution of the boundary value problems in the analytical form, exactly satisfy (if necessary) all the boundary conditions, take into account the apriori information about the solution behaviour, ets. The input languages of the POLYE system are a convenient means for describing boundary problems in their "natural" mathematical statement, and practically entirely exclude the processes of programming in the common sense of this word. The POLYE system makes it possible to reduce the time of programming and solving boundary problems in mathematical physics by 50 to 100 times.

It may be said that the POLYE programming system maintains its linkage with existing software, supports the technology of modular programming and consists of two parts, viz. the functional and system filling. The functional filling reflects the specific character of the system subject field and includes the library of modules used for developing working programs, as well as collections of standard computing schemes which determine the main stages of one or other typical problem. The system filling includes the assignment languages, the translator, the monitor program, system archives, etc.

The POLYE system integrates different program and language facilities which not only extend the capabilities of operating systems, but also imparts them a specialized orientation. Thus, for instance, the package of DIFOR programs extends the FORTRAN language with means for numerical-analytic differentiation of function; problem-oriented language RL provides means for programming boundary value problems solution structures and their other components. The system also includes a specialized macroprocessor whose facilities allow to organize a library of subroutines in the RL language.

The system input languages are maximally close to a conventional mathematical notation of the problem statement and the algorithm of its solution. The problem-oriented languages proposed have developed facilities for describing both geometric and analytical components of boundary value problems and allow to specify all the necessary required information more efficiently than conventional software. As a result, the process of writing programs, their debugging, modification and development is simplified.

The principles of construction of program and language facilities allow to formulate the system assignment both as an order to solve a specific boundary value problem in a given class of problems, and as a number of directions allowing to form a new solution algorithm. The experience of using the problem-oriented language RL and the POLYE system shows that they essentially simplify and speed up the most time-consuming stages of computational experiments, viz.: programming and debugging of modules, problem solving on a computer and analysis of the results obtained.

The problem-oriented language RL and the specialized POLYE system have a wide range of facilities for describing the methods of solving boundary problems (Ritz, Galerkin, least-squares, Kurant, Treftz and others) and linear algebra problems (solution of systems of equations, eigen numbers and eigen vectors problems); approximating polynomials ( Chebyshev, trigonometric, splines and atomaric functions, and others); integration methods (n-point Gauss formula, Monte-Carlo and others); various constructive means of the $R$-functions theory ($R$-operations, special operators, solution structures and others). The possibility of representing approximate solutions in the form of formulae containing the characteristics of physical and geometric values as alpha parameters allows to carry out a series of computational experiments within the framework of the POLYE system with changing of necessary parameters, as well as to solve problems of an optimization character. The results of solution of boundary value problems (different differential and integral characteristics) are displayed on the information output devices in the form of tables, graphs, contour lines, axonometric constructions.

A high level of the technology of programming is provided by automated synthesis of working programs. Thus, for instance, a computational experiment with the use of the POLYE system can be carried out in several hours or days (depending on the problem complexity, user's expertise, system capabilities). At the same time, when using conventional software, the solution of a problem in some cases requires several months. At this, there arise great difficulties both of a technical and psychological character connected with large volumes of information being processed, complexity of computational algorithms, multivariantness of mathematical experiments, multimodelness of the problems stated.

At present, the POLYE system is used widely in scientific research and in the tutorial process at a number of R&D organizations and higher education institutions in the country for solving the following problems:

boundary value problems in mathematical physics and developing application problems in mathematical physics and developing application program packages for analysis and optimization of various physical-mechanical fields (temperature, strain, force, electromagnetic) when solving problems in the theory of elasticity, theory of plates and shells, thermal physics, electrostatics and electrodynamics, etc.;

when delivering lectures in the theory of $R$-functions and the POLYE system; when fulfilling practical tasks, course and diploma works both in the solution of computational mathematics problems (approximation of functions, integration and differentiation, solution of systems of equations and others) and in the simulation of different physical-mechanical fields in CAD-systems.

## 2.2 System architecture

Organizationally the POLYE system consists of two parts: the system and functional filling.

The system filling is the administrative body of the POLYE system and includes the following components:

input language - means for users interface with the system (description of the analytical and geometric components of the boundary problem, description of the form of output of computational results, service and debugging operators, etc.);

archive, comprising a system for storing the elements of the functional filling and service information of the POLYE system;

monitor - software package ensuring the service and operational capabilities of the system (control program, translator from the input language, etc.).

The functional filling reflects the specificity of the system subject fields and includes the following components:

library of modules used for creating working programs for a given subject field;

collections of standard computational schemes which determine the basic stages of solution of one or other typical problem (construction of the solution structure, conversion of geometric information into analytical one, selection of approximating sequences, etc.).

When implementing the specialized POLYE system, a strategy of its embedding into the respective computer operating system has been accepted. This means that the POLYE system does not duplicate those facilities which are available in the operating system. The principle of embedding into the operating system means that the POLYE system, in essence, is its extension oriented to programming and solution of boundary value problems in mathematical physics and automated output of application program packages for solving the problems of research, analysis and optimization of physical-mechanical fields.

The problem-oriented languages of the POLYE system are distinguished by tree hierarchical levels:

non-procedural languages for describing the problem with the use of the terminology of a definite field of knowledge; problem-oriented high-level language RL oriented to a class of problems reflecting the specific features of the methods used;

superstructure above the procedure-oriented programming language - language DIFOR (differential FORTRAN).

The lower hierarchical level comprises the superstructure language above the procedure-oriented language (base language) whose feature of implementation consists in purposeful extension of the procedure-oriented language by the specialized means for processing information about the objects of the mathematical model. The software package implementing the superstructure language may be realized in a procedure-oriented languages whose operators are the operators calling subroutines in the procedure-oriented language. In essence, we have developed a new language, since a such an extension increases the capabilities of the procedure-oriented language and leads to a new interpretation of some of its components. As a result, the procedure-oriented language becomes more richer semantically. The advantages of such a method of implementing the language facilities are the following ones:

implementation only of the specific elements of the problems;

possibility of unlimited extension by special means;

easy implementation on other computers by extracting the required subject of the procedure-oriented language;

the language is easy to masters.

The growth of speed and volume of memory of modern computers allows to implement this language on the basis of existing programming systems by developing a special library of subroutines determining the additional orientation of the procedure-oriented language. It is expedient to use FORTRAN as the base language. This is explained by its wide use by different users. There exists the possibility of extracting a subset of the language which is implemented practically on all mainframes and minicomputers. Besides, FORTRAN has at its disposal means for such extensions by providing the apparatus of interaction of programs for this purpose. Implementation of separate system subroutines may be performed in a machine-oriented language for increasing the overall system efficiency.

Using further the complex of programs realizing the superstructure above the procedure-oriented language as a base for developing new language facilities it is easy to realize more-higher-level languages. The hierarchical second-level problem-oriented language represents an assemblage of means for writing down the problem solution algorithm as formulae relating the objects of the mathematical model used by means of algebraic operations. The processor purpose is to analyse and translate expressions, therefore its implementation poses no problem.

The upper hierarchical level is occupied by the non-procedural languages oriented to solution of a definite class of problems by specialists of different qualification (physicists, mathematicians, engineers and others). These languages are realized on the basis of languages of a more lower level (RL, DIFOR) in the sense of creating a special library of macrodefinitions. For this, the system macroprocessor and the problem-oriented language RL are used.

When solving boundary problems in the framework of a specialized system, the user is provided with the following possibilities:

description of boundary value problems in the RL language;

programming of separate modules in the DIFOR language;

creation of new subroutines in FORTRAN;

use of system facilities for organizing libraries and files.

The complex of software and language facilities is implemented on PC's compatible with the IBM PC. The problem statement in the POLYE system requires a minimal configuration of hardware and peripherals.

Volume of implementation:

processor - 70 subroutines in the FORTRAN-77 language (3500 operators);

DIFOR software package - about 220 subroutine in FORTRAN (17 000 operators).

The time of solution of a boundary value problem depends on many factors (kind of boundary conditions, method of solution, accuracy of computing the integrals and solving the problem on the whole, etc.) and varies from 1 min. to several hours.

The minimal volume of RAM is 320 K, external disc memory - about 500 K. When solving some large problems, up to 1 M of RAM and up to 20-30 M of external disc memory is used.

To start the programs on PC's, the instruction file POLYE is used. The parameter is the name of the file containing the program text in the RL language. Files with programs in the RL language have the extension RL.

## 2.3 Processor for the RL language

The source program for solving boundary value problems in mathematical physics written in the problem-oriented language RL is converted by the processor into an object program in the DIFOR language. The program is processed at first by the processor, then by the FORTRAN translator, and finally it is executed on a general-purpose computer.

The processes of translation and program execution are usually separated in time. In this case, the processor for processing the source program is related to translators of the compiling type. A processor can also be realized, in which translation and execution are concurrent, i.e. a translator of the interpreting type. It is expedient to use the interpreter as a dialog translator in a special-purpose computer ensuring a man-machine interactive mode. When implementing the system on a general-purpose computer, interpretation of the source text is practically of little efficiency, since the time needed for execution of the RL language operators (such as integration, solution of equations) does not allow to organize a man-machine interactive mode.

In the end, during translation, the processor has to solve the same problems as any translator from a procedure-oriented language, viz.:

identify the base elements of the language, the basic syntactic units (expressions, operators) and interpret their meaning;

translate the source program operators to the internal language;

allocate memory for variables;

generate the respective object codes;

locate syntactic and semantic errors in the program;

generate and output a printed document.

The basic construction of the RL language are comparatively short cortege expressions. That is why, without an overall loss of efficiency of translation, the processor is realized in the high-level language FORTRAN-77. In so doing direct methods of translation are used, in which for each construction of the input language an individual translation algorithm is selected.

The processor consists of four independent blocks for lexical analysis, syntactic analysis, optimization and generation of an object program.

The lexical analysis program scans the source program text and makes up information tables. At the given stage, identifiers, constants and keywords are packed, preliminary analysis of separate lexical elements is carried out, and the program text is printed out.

As a result of lexical analysis of the source program, a table of lexical units (lexemes) is formed, in which the key-words, identifiers and constants are reduced to one format and replaced with references to the respective tables. The program in the table of lexemes contains no commentaries, since they are not needed for further processing.

After this, syntactic analysis is performed, which discriminates separate language constructions (operators, expressions, etc.). The results of syntactic analysis are used for forming the internal program representation (its text in the internal language). The internal representation of the source program has the form of a list of instructions (operator, operand, result) generated by a special subroutine for parsing and analysis of cortege expressions. In the course of lexical and syntactic analysis, a full syntactic check of the source program is carried out and the respective error messages are printed out.

After translation into the internal language is over, a part of the information about the source program is stored as an internal program representation, and the other part is located in the different tables. All this information is used further at the stage of semantic analysis for program optimization, memory allocation and generation of the object program text.

At the stage of the program generation, the source program internal representation is translated into the DIFOR language. In other words, at this stage, semantic analysis is performed, in the course of which each operator of the source language is analysed and semantically equivalent sentences of the object language are generated. Besides, at this stage, memory is allocated for variables and constants, information about which is located in the tables of identifiers and numerals.

## 2.4 DIFOR software package

The function filling consists of modules used at different stages of solving the boundary value problem (formation of a system of algebraic equations, solution of linear algebra problems, presentation of results) and reflects the specificity of the system subject field. The functional filling comprises two libraries , viz. constant and temporary modules. The constant modules make up the system base and are called automatically by means of the temporary library modules formed as a result of translation of the source language from the RL language and DIFOR.

The modules of the POLYE system go through definite stages of processing, viz. generation, text editing, translation and loading. All this installation is performed by a special program which modifies the form and content of modules according to the input information.

The DIFOR language (differential FORTRAN) is oriented to solution of problems connected with differentiation of complex functions. The basic types of data in the language are differential corteges and their arrays on which the cortege algebra operations are defined. The DIFOR language is a purposeful extension of FORTRAN by a special library of subroutines which organize complex structure data, perform dynamic allocation of memory therefore, as well as execute cortege algebra operations (arithmetic, elementary functions, operators $D$, $T$, $R$-operations, etc.). Thus, for instance, some subroutines perform formation of corteges corresponding to the analytical and geometric components of the boundary value problem (circle, straight line, polynomial, etc.). In other words, these subroutines compute the values of some functions and their partial derivatives in the given point. Other subroutines are intended for realizing cortege operations ($+$, $-$, $*$, $/$, $**$, $sin$, $exp$, $ln$, $sqrt$ and others) whose execution yields new corteges or their arrays.

Let us give a brief characteristic of the library of subroutines realizing the algorithms of the algebra of differential corteges. The library is oriented to a programmer and is used, as a rule, with a high-level language. The subroutines contain no input/output operators and the user himself has to write the main program for initial data input, calling the library subroutines and output of computational results. The library of subroutines may serve as a basis for creating specialized packages of a more complex kind.

The library of subroutines for differentiating functions has been developed in portable FORTRAN and is intended to be used on modern computers for executing the basic cortege operations. The library contains about 50 subroutines presented in standard form and includes the following sections:

creation of differential corteges - geometric objects ($LINE$, $CIRCLE$, $BAND$) and analytical objects ($POL1$, $POL2$);

creation of arrays of corteges - sequence of classical polynomials ($THEB$, $LEGAN$, $GARMON$) and sequence of finite functions ($SPLI$, $UP$);

arithmetic operations with differential corteges and their arrays ($ADD$, $SUB$, $MULT$, $DIV$, $POWER$);

cortege operations for elementary functions ($ROOT$, $DERLN$, $DEREXP$, $DERSIN$, $DERCOS$, $DERTG$, $DERASN$, $DERATG$, $DERSH$, $DERCH$);

cortege $R$-operations and other operators ($AND$, $OR$, $NORMA$, $D1$, $T1$, $DX$, $DY$);

subroutine for differentiating a complex function ($SUP$, $DER$).

Let us consider the problems of presenting differential corteges in the computer memory, which are considered as one-dimensional arrays of real numbers. The length of such an array depends on the number of arguments and maximum order m of differentiating functions, and for the two-dimensional case it is determined as

$$d = (m+1)(m+2)/2.$$

The cortege components are arranged in groups in the order of increase of derivatives, and in each group the priority belongs to derivatives with a lesser index, i.e. the cortege has the following structure:

$$(u)\left(\left(\frac{\partial u}{\partial x}\right)\left(\frac{\partial u}{\partial y}\right)\right)\left(\left(\frac{\partial^2 u}{\partial x^2}\right)\left(\frac{\partial^2 u}{\partial x \partial y}\right)\left(\frac{\partial^2 u}{\partial y^2}\right)\right)\cdots\left(\cdots\left(\frac{\partial^m u}{\partial y^m}\right)\right).$$

Such mapping of corteges on one-dimensional arrays provides the possibility of direct access to each cortege component. The relationship between the serial number $P$ in the array of numbers and indices of the derivative $(1,k)$ is found from the formula

$$p = (l+k+1)\,(l+k)\,/2 + k + 1.$$

To use of the library of subroutines for differentiating functions user must have a sufficiently foundational knowledge of the operating system and the procedure-oriented language FORTRAN. Such use is very time-consuming, since the user has to not only write the main program, but also debug it, which, as a rule, requires considerable time and manpower expenditures.

All the library modules may be called with the help of the subroutines $CALL$ access operator. They mainly implement only computational functions and do not contain input/output devices access operators. Therefore, the functions of input of initial data, construction of the sequence of calling the necessary subroutines and output of results in convenient form are placed on the user. On the one hand, the availability of such a library significantly facilitates the problem solution, and practically does not depend on the type of computer and OS, and on the other hand, the labour input needed for obtaining the necessary results remains fairly high.

When using the library subroutines it is necessary to follow the rules of writing subroutines in FORTRAN. In so doing, all the arrays required for storing corteges are described in the main program by operator DIMENSION. Besides, the binomial coefficients (array $BINOM$) are computed in the main program, as well as the numbers of the components of the corteges which correspond to different partial derivatives (array $INDEX$), whose elements are computed by the formula .

Such preliminary calculations of expressions common for all subroutines allow to increase significantly the speed of subroutines for differentiating functions (e.g. when calculating fifth-order corteges of two-variable function the speed increases by a factor of two).

When accessing the library subroutines, the formal parameters are replaced by actual ones, thereat their quantity, type and sequence of parameters should coincide. The parameters of the differentiation subroutines are the following variables and arrays:

$M$ is the cortege order (required order of function differentiation incremented by 1);

$X$, $Y$ is the current value of arguments (for subroutines which generate the values of corteges or their arrays);

$A$, $B$, $C$ are the differential corteges (arrays of length $(M+1)*M/2$);

$BINOM$ is the arrays for storing binomial coefficients;

$INDEX$ is the array for storing the control vector used for providing access to the cortege elements.

Let us consider an example. It is necessary to calculated the derivatives up to the $m$-th order of a two-variable function:

$$f(x,y) = \frac{g(x,y) * p(x,y)}{u(x,y)} \sin\left(\lg\left(u(x,y)\right)\right).$$

We shall put array $G$ into correspondence to the cortege of function $g(x,y)$, and arrays $P$, $U$, $F$ to the corteges of function $p$, $u$, $f$ whose components are calculated in point $x$, $y$. During calculation we shall need working arrays $R1$, $R2$, $R3$ for storing the intermediate differential corteges. A fragment of the main program for differentiating function $f(x,y)$ has the form:

```
C
      DIMENSION G(10),P(10),U(10),F(10),
     * R1(10),R2(10),R3(10)
```

```
C
      DIMENSION BINOM(10),INDEX(4)
C
C   HERE ARE COMPLETED ARRAY
C   G, P, U, BINOM AND INDEX
C
C   M - CORTEGE ORDER IS DEFINED
C
      CALL MULT(M,G,P,R1,BINOM,INDEX)
      CALL DIV(M,R1,U,R2,BINOM,INDEX)
      CALL DERLN(M,U,R1,BINOM,INDEX)
      CALL DERSIN(M,R1,R3,BINOM,INDEX)
      CALL MULT(M,R2,R3,F,BINOM,INDEX)
C
```

Let us comment the results which will be obtained when computing the program fragment given above. The program begins with a description of the arrays used for storing the differential corteges (in the given case the corteges are not higher than the fourth order). After forming the initial arrays (input, computation with the use of the library subroutines, transfer by assignment operators) the corteges $G$ and $P$ multiplication subroutines are called. As a result, array $R1$ , is formed, which contains the cortege of function. Further calls of subroutines $DIV$, $DERLN$, $DERSIN$, $MULT$ perform step-by-step calculation of the function $f(x, y)$ cortege. Hence, after executing the given program fragment, array $F$ shall contain the values of function $f(x, y)$ and its all partial derivatives up to the $(M - 1)$-th order.