# WORKING PAPER

## INTEGRATION PRINCIPLES IN NUMERICAL SOFTWARE

*V. Mazourik*

April 1989
WP-89-010

IIASA
International Institute
for Applied Systems Analysis

# INTEGRATION PRINCIPLES IN NUMERICAL SOFTWARE

*V. Mazourik*

April 1989
WP-89-010

Computer Center of the USSR Academy of Sciences Moscow



INTERNATIONAL INSTITUTE FOR APPLIED SYSTEMS ANALYSIS
A-2361 Laxenburg, Austria

## Foreword

Since personal computers are currently being used by individuals who are not professional computer specialists, the need for a framework allowing quick and easy development of user-friendly software is necessary. One such framework is the concept of *software integration*. This concept is now reasonably popular and several software tools allowing for such integration are available on the market. Unfortunately, all of these tools are oriented towards office automation systems, word processing, telecommunications etc. No such tools are available for scientific users which would allow easy utilization of mathematical programming software, model building and analysis etc.

This paper presents the system DISO which can support dialogue solving mathematical programming problems. The system allows for the treatment of all mathematical programming modules in a uniform way, allowing easy interaction with the user utilizing various metaphors for data presentation and analysis of results as well as being open - i.e. new mathematical programming modules can be easily linked to the system. The DISO system has been designed utilizing modern software engineering concepts, like hierarchical program structuring, object-oriented paradigm and abstract data types oriented interface.

Alexander B. Kurzhanski
Chairman
System and Decision Sciences Progam

## Table of Contents

Page

# INTEGRATION PRINCIPLES IN NUMERICAL SOFTWARE

*V. Mazourik*

## 1. Integration in data processing.

A wide use of personal computers results in new research of software design for nonprofessional users. Some of the approaches, which were developed in this area, have the general meaning and can be applied for the numerical analysis software design as well.

One of these approaches is software integration. The principle of software integration is now the leading trend in commercial packages. This principle does not mean simple inclusion of different packages in one system. Actually, it means, that some unified approach is used for system design, which simplifies both the implementation of the system and, what is more important, its use.

Most of the well known integrated systems like LOTUS 1-2-3, SYMPHONY [1] etc. are based on spreadsheet approach. It means, that all the components of the system are oriented to spreadsheet capabilities. For example, database records in SYMPHONY are represented as lines in a spreadsheet.

FRAMEWORK [2] uses another approach. The integration basis in this system is a frame. The user deals with hierarchy of frames, which are texts, spreadsheets, databases or complicated structures of other frames.

The integration in JAVELIN [3] is perhaps the most impressive one. All the information in the system is saved in a highly unified format which is "I_don't_know_how". The idea is to supply the user with two sets of tools: one is a list of different forms to input data into the system and another is the list of patterns to retrieve information and to show it in the most convenient way. Which way to choose for representing the data inside the system is not the user's affair.

The usage of strictly defined integration concept in each of these systems results in their simplicity and clearness. The developement of new integration principles is extremely interesting task, because these principles are general in their nature and can be effectively used in different situations. There exists an analogy with programming languages. The structured programming approach is invariant and does not depend on the

language in use. This approach can be effectively used, say, in FORTRAN, which itself does not have any features of this style.

We can treat the developement of integration concepts for software packages as the use of structured programming approach in this new area of application.

Integrated systems which are available now at the software market are oriented to general data processing and include the list of components, which becomes traditional: text processor, spreadsheet, data base, graphics and telecommunication. They create the base of new information technology, promoting now due to the wide distribution of personal computers.

There exists another area of software packages, which deals with mathematical modeling. This type of software is much more sofisticated than ordinary data and text processing.

## 2. Mathematical integration.

Mathematical integration is based on the following principles:

- basic set of interrelated models
- hierarchy of models
- basic object-oriented software package
- numeric libraries
- unified model-oriented exception handling
- unified abstract data types-oriented interface.

This paper presents a short description of all these principles. They are valid for different applications. One of them is optimization problem solving. Mathematical integration approach which is described here was used in the implementation of the DISO system (DIalogue System for Optimation problem solving). The DISO system was developed in the Computer Center of the USSR Academy of Sciences for different types of computers including mainframes and PCs. We shall illustrate all the integration principles using the DISO case as an example.

### 2.1. Basic set of interrelated models.

There exist three fundamental optimization models which, being combined, create a powerfull instrument for numerous applications problem solving:

- unconstrained minimization (UCM),
- nonlinear programming (NLP),
- multicriteria optimization (MCO).

Actually, NLP can be treated as a simple case of MCO with two criteria: one is a goal function and another is some integrated measure of restrictions violation. There are two reasons to treat NLP as a basic model. The first is that there exists an advanced theory of this case. The second is that a lot of important applications come to this mathematical model.

Very many mathematical models such as optimal control, distributed systems equations solving, a boundary value problem for ordinary and partial differential equations etc. can be reduced to one of these three fundamental models of optimization.

For example, the following optimal control problem:

$$dx/dt = f(x,t,u); t0 <= t <= t1; x(t0) = x0;$$

$$\min b(x(t1))$$

$$u(t) <= P$$

is reduced to UCM problem by the discretization of vector $u(t)$. UCM goal function $u \rightarrow b(u)$ is evaluated for a given discrete approximation of $u(t)$ by ODE integration. If there are some trajectory restrictions like $g(x(t)) = 0$ then NLP problem occurs with the restriction $G(u) = 0$.

To solve the resulting UCM or NLP problem with the use of gradient methods one must have effective algorithms of $u \rightarrow db/du$ gradient function evaluation. Taking into account the nature of $b(u)$ function it is obvious, that its numerical differentiation is practically impossible. There exist effective algorithms of recurrent function differentiation which are proposed by Prof. Evtushenko [4]. They are the part of a more general approach which makes it possible to effectively differentiate a function (obtaining analytically accurate, but numerical results), defined by the algorithm rather than by analytical expression [5]. The fundamental nature of UCM, NLP and MCO models makes this set a full system. Using the analogy with linear spaces we can say, that these models are similar to coordinate axes and their "linear combinations" create the effective tools to solve almost any optimization problem.

The developer of integrated system which is based on mathematical models actually must investigate thoroughly the proper set of models to provide its fullness. Of cause in most cases this feature can not be proved for the simple reason that there is no strict

definition of it. But there always exists experience of problem solving in a given area, which must help in this question. Returning to optimization technique we can illustrate the necessity of integration principles the by following example. Typical situation in optimization problem solving is task redefinition in the process of the solution. The change of function status from goal function to restriction and back very often results in the change of the model itself between NLP, MCO and UCM. In a way, this is the change of the user point of view on his data. The same approach is an ordinary thing in traditional intrigrated systems, when the same data are viewed either as spreadsheet items or as database records. What must be done in the case of mathematical models is the same simplicity to change the view of the data.

### 2.2. Hierarchy of models.

In fact, there exists some intrinsic hierarchy of optimization models in a sense that numerical method of one class can create subordinate task of another class. Typical example is NLP methods, which are based on penalty function approach. They create UCM task making some convolution of goal function and restrictions.

Integration approach makes it possible to create the programming structure of the system in accordance with the mathematical structure of the interrelated models. In fact, there is no need to duplicate the code of UCM methods while implementing NLP library. The only thing to be done is to call UCM package inside NLP method yielding in invocation of all the means of UCM package (including, if necessary, interactive capabilities) in the process of NLP problem solving. This is the exact case of the DISO implementation. This source of hierarchy is due to the semantics of numerical methods. There exist some other potential sources of hierarchy. Global optimization problem solving which uses nonuniform covering technique can be substantially amplified if the estimation of function low value is improved. It makes it effective to combine local and global search in one session. Again the integration approach solves this problem automatically. The DISO system permits the user to change the optimization mode from global to local, and after descent within the local domain of the current point continue global search.

There exists potential possibility of more skillful hierarchical control of the solution. For example, the user can stop global search process, which currently uses some covering technique, and make the system continue the search in subarea with another covering scheme.

### 2.3. Basic object-oriented software package.

Integration approach provides the system designer with the unique capability to investigate and extract the common part of all the integrated mathematical models. Obviously, this common part exists, due to the similarity of the models.

Unnecessary duplication inevitably exists if the models are implemented independently. But this is not the main disadvantage. Independent implementation results in "local optimization" principle, when all the important solutions are restricted to the local needs of the particular model efficient implementation. Anyone who ones tried to combine such independent packages understands the problem.

Orientation to procedural, calculational part of the problem is typical for this approach. Data representation plays the secondary role and is subordinate to the procedure needs. It results in the fact, that both the system structure and user interface are unnatural. For example, presence of the field "current point" at the screen makes dangerous illusion, that the user has access to some vector space and can work with its points. What he actually has is access to the array of numbers in the program. The attempt to create, say, two points simultaneously (and to begin independent optimization processes from these points) proves immediately, that neither vector space, nor vectors are, in fact, available. Of course, it is very inconvenient for the user, who can not understand why such a natural and simple things are forbidden in the system. The user can easily handle an artificial manner of the language as far as its syntax is concerned. But not the semantics.

Basic object-oriented software package (BSP) development is an attempt to solve all these problems. The main question is to find the common part for optimization models, which represents their semantics and thus can play the role of implementation base for them. To find the main components of this basic package let's investigate the structure of optimization problem solving.

The main goal in optimization is, in fact, analysis of the functions, which creates task definition. Function analysis and modification at the beginning stage of the solution is much more a time consuming procedure than the use of optimization method afterwards. As it was mentioned above task modification can include function redefinition, change of their status, addition of new functions etc. The same capabilities are very useful at the next stage of problem solving, when the task definition functions are investigated and modified in a proper way, and mathematical model to solve the problem is chosen. Very often function modifications take place again. But if at the beginning stage of the solution functions are modified using the application point of view (for example, they do not meet some informal conditions), at this second stage mathematical properties of the functions are improved in the process of modifications in order to run optimization

methods more effectively.

Continuing this analysis one can easily understand that the notion of the function is one of the main semantic parts of the BSP. One more important conclusion is obvious. Only correct function definition as a programming object, which fully corresponds to the strict mathematical definition of it, will do. Any other solutions based on some "local optimization" approach must be rejected in principle.

The function notion, being the main in BSP, is not the only one. The BSP for the optimization package development is, in fact, abstract cluster, which includes a set of interrelated abstract data types of linear algebra, such as vector space, function, vector, basis etc., and procedures to manipulate objects of these types. The BSP approach has several goals. The first is a technological one. The use of the BSP cluster inbeds structured object-oriented style to the mathematical packages development in the frame of integrated system. Another BPS goal is to provide integrated system designer with effective tools to develop new ideas to control optimization problem solving. Any mathematical model can be treated as a frame, which defines all other parts of the system. For example, numerical methods are linked to the structure of the model. Being a theoretical scheme, mathematical model must not take into account any peculiarities of application task, which deviate from the scheme. But often these very deviations play decisive role in the most effective task solution. BSP is a tool to describe and analyze some special options of mathematical models. It means, that the user creates his own, outer methods of optimization in dialogue session, which are highly oriented on the specific properties of the task at a given point. This synthesis of new mathematical methods (or scenario) of task solution is possible only on the base of BSP capabilities.

This approach solves the well-known paradox which exists in the traditional systems, based on numerical methods libraries. Numerical methods development is very time consuming job which is made by skillful specialists. These methods are really very strong instrument ... but only if the task to be solved has some very special properties, like the existence of the first and the second derivatives etc. The problem is that to investigate those properties without special software instruments is often as time consuming as to solve initial optimization problem. So a strong point of the methods turn out to be their weak points. The reason is the closed, static nature of the methods. All the powerful mathematics inside the method is hidden from the user. And what he actually needs are simple things at hand, like the use of linear operator to change equipotential lines of the function.

The brief description of BSP is given below.

### 2.4. Numeric libraries.

Numeric libraries are a common part of different mathematical models in integrated systems. The methods themselves are of course different, but all the routines of methods initiation, run, termination and parameters control are common, which makes it possible to implement them once in the framework of integrated system.

Unified manner of methods handling procedures results in the unification of user's interface to control numerical algorithms, which is another useful feature of integrated approach. The unification of methods handling has the same origin as the unification of data handling procedures, such as copy, move, etc., in traditional integrated systems. The attributes of these procedures do not depend on the type of data to be processed: text, spreadsheet or database.

### 2.5. Unified model-oriented exception handling.

Let's investigate the goal of dialogue in the process of optimization problem solving. It is obvious that the dialogue in general is possible only if participants have nonempty cross of their world models. It is also obvious that dialogue makes sense only if their world models do not coincide. These simple remarks show the way to understand the interface logical structure in optimization packages.

Let's consider the innermost level of numerical methods. We can treat any numerical method for a given optimization model as a "software-being", which is specialized in particular, narrow area. For example, Newton method is the best in the vicinity of the solution point. When somebody (program or user) calls Newton method, it means that both the caller and the method have common understanding of unconstrained minimization model. At the programming level this common understanding is materialized in the form of the procedure interface specification. The difference between their models is that the caller does not need to understand the idea of Newton approach. On the other hand, Newton method does not know, whether initial point is in the vicinity of the solution or not. If it is not, then exception will be raised, which means, that the model state is not in the domain of Newton method. This example demonstrates model-oriented exception handling approach. Exceptions can be treated as current state transfer beyond the boundaries of model understanding by the method.

As an additional result, it helps to understand the role of optimization package for a given model. The package filters the exceptions, which were raised by methods. Any particular method can not have all the information about the model state, but the package can, and actually have to. When a method due to some reason fails to solve the problem the package must analyze the reason and make the decision how to handle the situation. One possible solution is to change parameters of the method or the method itself.

All this means that the package represents the whole semantics of the optimization model, which is implemented as data structures, numerical methods library and utility procedures of different types. The package must obviously include method descriptors which contain specification of parameters, domain, preferable model state to use the method etc. All the resources of the package are controlled by the monitor program, which provides both automatic and dialogue capabilities of optimization problem solving.

Optimization package for a given model creates the middle level of programming hierarchy. The outermost level is the caller itself. It is the last level of exception handling. There always exist some situations which are beyond the understanding of the package. They must be filtered by the caller, which means, that the caller must have common understanding of the model with the package. In fact, he must have additional knowledge to handle situations which were rejected by the package. It is exactly the case, when the caller is human being, and exceptions are due to his mistakes in task definition. Unified implementation technique of model-oriented exceptions handling create another area of integration. Taking into account hierarchical manner of optimization models this unification is the necessity.

### 2.6. Unified abstract data types-oriented interface.

The BSP provides the user with model-oriented high-level language, which includes abstract data types of linear algebra objects: vector space, basis, linear operator, vector etc. The user can create these objects, control their values, analyze and modify their properties with the help of cluster procedures. As usual, abstract approach means, that data types implementation is hidden from the user. Object's semantics is defined at the procedural level by all the procedures which can potentially be applied to the object.

Abstract data types approach can be extended to data visualization. Actually, it is a necessary extension, if we deal with abstract data. Abstract visualization is based on the following three ideas.

First, it is an abstract procedural interface. The user reports the system only his intention to see visual pattern of the object. He calls "demo" procedure which has the reference to the object as a parameter. The system itself will find the proper pattern to demonstrate the object.

Second, there exist several predefined patterns of visualization for every abstract data type. They may be linked to some situations or object values, which can be analyzed by the system. The user can extend the list of patterns and situation filters for a given object.

Third, there exists field management mechanism, which is independent of data patterns and provides the user with the means to organize his own layout of the screen. Fields are rectangle areas on the screen. Their number, shape and other attributes are under the user control in a dialogue session. The object is visible only if it is linked to one or several fields on the screen. The "demo" procedure reacts both on the object state and the field attributes, which makes it possible to see simultaneously several different patterns of the object in the screen. This approach makes the strict division between the static, abstract part of visualization, which the user creates when he writes application program, and dynamic part, which both the user and the application program can create and modify in the process of solution.

Abstract data visualization was implemented in Field Manager Package [6]. This package provides software designer with new interactive tools, which create significant part of integrated technology for model-oriented software development.

## 3. BSP description.

As it was mentioned above, the Basic Software Package (BSP) is abstract data types cluster, which provides the user with a set of interrelated mathematical notions of linear algebra.

### 3.1. Object types of BSP cluster.

The object types of BSP cluster are:

vector_space - finite-dimensional vector space over the field of real numbers (double precision numbers in implementation);

vector - element of a given vector space;

basis - basis of vector space;

| | |
|---|---|
| vc_system - | ordered set of vectors in vector space (not necessarily linear independent); |
| operator - | linear operator which maps one vector space into another; |
| function - | arbitrary function, i.e. functional mapping of linear space into field of real numbers. |

The BSP cluster has hierarchical structure. The innermost level is subcluster of matrix algebra, which has obvious data types like column, row, matrix etc.

### 3.2. BSP procedures specification.

Cluster includes procedures to create objects, evaluate their values, analyze attributes of objects (matrix rank, space dimensionality, etc). Some illustrative (not full) list of cluster functions specification is given below to make the general impression about the cluster contents.

#### 3.2.1. Vector space.

Create root vector space of a given dimension:

```
vector_space vs_root (dimension)
  int dimension;
```

Create vector space by ordered set of vectors:

```
vector_space vs_by_sy (sy)
  vc_system sy;
```

Create vector space by kernel of linear operator:

```
vector_space vs_by_ker (oper)
  operator oper;
```

Create vector space by image of linear operator:

```
vector_space vs_by_image (oper)
  operator oper;
```

Reference to parent vector space:

```
vector_space father_of_vs (vs)
  vector_space vs;
```

Reference to greatest common measure of two vector spaces:

```
vector_space vs_GCM (vs1, vs2)
  vector_space vs1, vs2;
```

Dimension of vector space:

```
int dimension_of_vs (vs)
  vector_space (vs);
```

Is vector space "sub_vs" subspace of "vs":

```
int is_sub_of_vs (sub_vs, vs)
  vector_space sub_vs, vs;
```

Reference to i-th subspace of a given vector space:

```
vector_space sub_of_vs (vs, i)
  vector_space vs;
  int i;
```

Reference to root basis of a given vector space:

```
basis root_bs_of_vs (vs)
  vector_space vs;
```

Delete vector space:

```
void vs_delete (vs)
  vector_space vs;
```

### 3.2.2. Vector.

Create vector:

```
vector vc_create (bas)
  basis bas;
```

Assign the value to vector:

```
void vc_value (vc, bs, val)
  vector vc;
  basis  bs;
  column val;
```

Get vector value in a given basis:

```
column value_of_vc (vc, bs)
  vector vc;
  basis bs;
```

Transform vector value to a given basis:

```
void vc__to__bs (vc, bs)
  vector vc;
  basis bs;
```

Reference to basis of current vector value:

```
basis bs__of__vc (vc)
  vector vc;
```

Reference to vector space of a given vector:

```
vector space vs__of__vc (vc)
  vector vc;
```

Standard vector space operations over vectors:

```
vector vc__plus (vc1, vc2)
  vector vc1, vc2;
vector vc__minus (vc1, vc2)
  vector vc1, vc2;
vector vc__inv (vc)
  vector vc;
vector vc__mult__scalar (vc, scal)
  vector vc;
  double scal;
```

### 3.2.3. Ordered set of vectors.

Create empty set of vectors:

```
vc__system sy__create (bas)
  basis bas;
```

Add vector to a given set:

```
void sy__include (sy, vc)
  vc__system sy;
  vector vc;
```

Eliminate vector from the set:

```
void sy__elim (sy, vc)
  vc__system sy;
  vector vc;
```

Is set of vectors empty:

```
int is_empty_sy (sy)
  vc_system sy;
```

Rank of set of vectors:

```
int rank_of_sy (sy)
  vc_system sy;
```

Transform set of vectors to a given basis:

```
void sy_to_bs (sy, bs)
  vc_system sy;
  basis bs;
```

### 3.2.4. Basis.

Create basis by set of vectors:

```
basis bs_by_sy (sy)
  vc_system sy;
```

Create basis by matrix:

```
basis bs_by_mt (mt)
  matrix mt;
```

Extract i-th vector of basis:

```
vector vc_extract_of_bs (bs, i)
  basis bs;
  int i;
```

Reference to vector space to which a given basis belongs:

```
vector_space vs_of_bs (bs)
  basis bs;
```

Reference to i-th linear operator with domain in a given basis:

```
operator op_domain_in_bs (bs, i)
  basis bs;
  int i;
```

Reference to i-th linear operator with image in a given basis:

```
operator op_image_in_bs (bs, i)
  basis bs;
  int i;
```

### 3.2.5. Linear operator.

Create linear operator by set of vectors:

```
operator op__by__sy (sy, bs__of__image__vs)
  vc__system sy;
  basis bs__of__image__vs;
```

Create linear operator by matrix:

```
operator op__by__mt (mt, bs__of__domain__vs, bs__of__image__vs)
  matrix mt;
  basis bs__of__domain__vs;
  basis bs__of__image__vs;
```

Is vector in domain of linear operator:

```
int vc__in__domain__of__op (vc, op)
  vector vc;
  operator op;
```

Reference to domain basis of linear operator: basis domain__bs__of__op (op) operator op;

Apply linear operator to vector:

```
vector op__call (op, vc)
  operator op;
  vector vc;
```

Linear operator value in a given pair of basis:

```
matrix value__of__op (op, bs__of__domain__vs, bs__of__image__vs)
  operator op;
  basis bs__of__domain__vs;
  basis bs__of__image__vs;
```

Transform linear operator to a given pair of basises:

```
void op__to__bs (op, bs__of__domain__vs, bs__of__image__vs)
  operator op;
  basis bs__of__domain__vs;
  basis bs__of__image__vs;
```

### 3.2.6. Function.

Create function with domain in a given basis:

```
function fn_create (bs, fn_code)
  basis bs;
  double (* fn_code) ();
```

Reference to basis of function domain:

```
basis bs_of_fn (fn)
  function fn;
```

Is vector in domain of function: k

```
int vc_in_domain_of_fn (vc, fn)
  vector vc;
  function fn;
```

Evalute function at a given point of vector space:

```
double fn_call (fn, vc)
  function fn;
  vector vc;
```

Evaluate gradient value of function at a given point:

```
row grad_of_fn (fn, vc)
  function fn;
  vector vc;
```

Evaluate hessian value of function at a given point:

```
matrix hess_of_fn (fn, vc)
  function fn;
  vector vc;
```

## 3.3. BSP semantic specifications.

BSP cluster specification includes list of equations, which define semantics of cluster objects and procedures. Here are some of them.

The property of root vector space:

```
father_of_vs (vs_root (int)) == NULL;
```

Dimension of vector space which is spanned on set of vectors is equal to the rank of this set:

dimension_of_vs (vs_by_sy (sy)) == rank_of_sy (sy);

Vector space, spanned on set of vectors which belong to some vector space, is subspace of this vector space:

is_sub_of_vs (vs_by_sy (sy), vs_of_sy (sy) == TRUE;

Dimension of vector space, which is created by kernel of linear operator, equals to the remainder of dimension of domain vector space of operator and rank of operator:

dimension_of_vs (vs_by_ker (op)) ==
dimension_of_vs (vs_of_bs (domain_bs_of_op (op))) -
  rank_of_mt (value_of_op (op));

Kernel of linear operator creates subspace of domain vector space of this operator:

is_sub_of_vs (vs_by_ker (op),
vs_of_bs (domain_bs_of_op (op))) == TRUE;

Dimension of vector space which is created by image of linear operator, equals the rank of operator:

dimension_of_vs (vs_by_image (op)) ==
rank_of_mt ( value_of _op (
op, domain_bs_of_op (op), image_bs_of_op (op)));

Rank of set of vectors, which create basis of vector space, equals the dimension of this space:

rank_of_sy (sy) ==
dimension_of_vs (vs_of_bs (bs_by_sy (sy)));

Domain of vc_to_bs procedure is restricted by the following necessary condition:

is_sub_of_vs (vs_of_vc (vc), vs_of_bs (bs)) == TRUE;

Greatest common measure reduction is automatically fulfilled (if necessary) when adding two vectors. This operation is ruled by the following equations:

vs_of_bs (bs_of_vc (vc_plus (vc1, vc2))) ==
vs_GCM (vs_of_vc (vc1), vs_of_vc (vc2));
bs_of_vc (vc_plus (vc1, vc2)) ==
if (is_sub_of_vs (vs_of_vc (vc1), vs_of_vc (vc2))) {
  bs_of_vc (vc2);
} else if (is_sub_of_vs (vs_of_vc (vc2), vs_of_vc (vc1))) {
  bs_of_vc (vc1);
} else {
  root_bs_of_vs (vs_GCM (vs_of_vc (vc1), vs_of_vc (vc2)));

```
}
```

Applicability of linear operator to vector is defined by the following condition (which means automatic reduction of linear operator to subspace of domain vector space):

```
vc_in_domain_of_op (vc, op) ==
is_sub_of_vs (vs_of_vc (vc), vs_of_op (op)));
```

Applicability of function to vector is defined by the following condition (which means automatic reduction of function to subspace of domain vector space):

```
vc_in_domain_of_fn (vc, fn) ==
is_sub_of_vs (vs_of_vc (vc), vs_of_fn (fn));
```

### 3.4. Example.

Let's consider the situation when goal function in unconstrained minimization problem has irregular equipotential lines. It means the existence of narrow caves which create great difficulties for optimization methods when the current point is at the slope of a cave. The optimization process might be very much simplified by the extraction of two subspaces in initial vector space, which are linked separately to "fast" and "slow" coordinates of gradient vector. It corresponds to descending the slope and then moving parallel to the bottom. Procedures, which are described below, give the idea, how to manage this situation with the use of BSP capabilities.

```
static vector_space V;
static function fn;
static int fast_dimens;
static vector_space V_fast;
static vector x_fast;

double fn_fast (x)
  double * x;
{
  vc_array_value (x_fast, x);
  return (fn_call (fn, x_fast));
}

void reduction_to_fast_coord (dimens, fun, point, fast_dimens)
  int dimens;
```

```
  double (* fun) ();
  double  * point;
  int     * fast__dimens;  /* output value */
{
extern void  coord__analysis ();

basis root__bs;
vector x;
vc__system sy;
int * fast__ind;
int i;

fast__ind = calloc (dimens, size of (int));
V = vs__root (dimens);
root__bs = root__bs__of__vs (V);
x = vc__create (root__bs);
sy = sy__create (root__bs);
fn = fn__create (fun, root__bs);
vc__array__value (x, point);
coord__analysis ( grad__of__fn (fn, x), fast__dimens, fast__ind );
for (i = 0; i < * fast__dimens; i++) {
sy__include ( sy, vc__extract__of__bs (root__bs, fast__ind [i]) );
}
V__fast  = vs__by__sy (sy);
x__fast = vc__create (root__bs__of__vs (V__fast));
free (fast__ind);
vc__delete (x);
sy__delete (sy);
}
```

Procedure "reduction__to__fast__coord" creates root vector space "V" and function "fn" with domain "V". Algorithm of function evaluation is given by a parameter "fun". Procedure "vc__array__value" assigns initial value to vector "x", which belongs to vector space "V". Procedure "coordinate analysis" evaluates gradient components of function and creates "fast__ind" array. Subspace "V__fast" is created by the extracted set of basis vectors. Vector "x__fast" is then created which belongs to "V__fast". Thus all the objects are ready for "fn__fast" function which is reduction of "fun" to the subspace of fast gra-

dient coordinates. Note, that "fn_fast" function interface does not contain any cluster types.

## 4. Integration, clusters and programming technology.

Structured programming principles can be transferred to the area of numerical packages implementation. In fact, in mathematical integration the use of structured programming technology is absolutely necessary because of the complexity of the problem. On the other hand, structured programming itself becomes more coherent and model-oriented.

Structured programming approach deals mostly with procedural part of programs, making them more clear and reliable. In fact, this approach was very useful to understand, that data and procedures are two sides of the same medal. They are so strongly interrelated, that some control structures are useless, if there are no corresponding data structures in program and vice versa. Abstract cluster approach greatly improves this feature of data and procedures interrelation. Mathematical integration gives new direction to structured programming. The order in programming technology is due now to the intrinsic structure of mathematical models which create the base of integrated system. Logics of programs and their links is derived from the semantics of model objects and their interrelations. Abstract nature of mathematical notions makes it natural to describe them using the language of abstract clusters.

The BSP which was described above has abstract cluster structure. The same approach is valid for the upper levels of integrated software hierarchy. These levels are: library level, package level and system level. As it was mentioned above, they are responsible for model-oriented exception filtering scheme, which is an essential part of their semantics. As an abstract cluster, integrated system consists of hierarchy of interrelated subclusters. Here is a brief review of their structure.

Main object types of cluster are:

| | |
|---|---|
| method - | numeric method; |
| library - | numeric methods library, which is a set of methods for a given model type; |
| package - | numeric package for a given model type; |
| task - | applied problem to be solved with the use of integrated system. |

There are sets of predefined attributes for every data type in this list. Attributes are accessible through procedural interface. For example, cluster includes procedural access to method parameters, exception lists, library contents, task attributes, comments, which are the base for "help" mechanism, etc.

### 4.1. Numeric method.

Create method for a given model:

```
method met_create (met_name, model_type)
char * met_name;
int model_type;
```

New method descriptor is created by this procedure. All the control mechanisms in system use this descriptor to address the method. The user can create several descriptors for the same method and make them differ in the values of control parameters. It means, that the same numeric algorithm can be linked to different control blocks thus creating different schemes of problem solving. Independent status of method descriptors is a good base for multiprocessing.

Initialize method:

```
met_init (met_descriptor, task_descriptor)
method met_descriptor;
task task_descriptor;
```

Initialization may include some preliminary calculations, memory allocation, initial state analysis etc. The second parameter provides information about the task to be solved. It may be useful for initialization process.

Run method:

```
met_run (met_descriptor, task_descriptor)
method met_descriptor;
task task_descriptor;
```

Several successive calls of this procedure result in the continuation of problem solving, data and method parameters being automatically transferred between these calls.

Terminate method:

```
met_terminate (met_descriptor, task_descriptor)
  method met_descriptor;
  task task_descriptor;
```

### 4.2. Numeric methods library.

Create empty library:

```
library lib__create (lib__name)
  char * lib__name;
```

Include method to library:

```
met__include (met__descriptor, lib__descriptor)
  method  met__descriptor;
  library lib__descriptor;
```

Eliminate method from library:

```
met__eliminate (met__descriptor, lib__descriptor)
  method  met__descriptor;
  library lib__descriptor;
```

Delete library:

```
lib__delete (lib__descriptor)
  library lib__descriptor;
```

The user can define simultaneously several different libraries for the same mathematical model. This kind of flexibility may have many applications. One of them is memory allocation algorithms with overlays, which is an important problem for small computers.

### 4.3. Package.

Package is the main resource to solve the problem of a given type, which corresponds to some mathematical model. It consists of the main control program, which supervises all data and procedure resources. Package provides the user with model-oriented interactive capabilities. In integrated system there are as many packages as it is predefined by hierarchical structure of corresponding mathematical models. For example, as it was mentioned above, the dialogue system for optimization problem solving DISO includes unconstrained minimization package, nonlinear programming package, multicriteria optimization package, linear programming package etc.

Call package of a given type (for example, unconstrained minimization) to solve the problem:

```
UCM__monitor (task__descriptor)
  UCM__task task__descriptor;
```

Actually, package must be treated as dialogue cluster, which includes not only data and procedural model-oriented resources, but interactive resources as well.

### 4.4. Task.

Task is collection of data and procedures, which define the problem to be solved. It has its own structure for a given mathematical model. The user can create task. It means, that task descriptor will appear, which contains all necessary information of its contents and properties. Task refers to its components. For example, nonlinear programming task consists of the following objects:

- dimension of independent variables vector;
- number of equality restrictions;
- number of inequality restrictions;
- function to be minimized;
- equality vector function;
- inequality vector function;
- vector of independent variables;
- vector of equality restrictions current values;
- vector of inequality restrictions current values;
- function current value;
- dual vector current values.

Create nonlinear programming task:

NLP_task NLP_task_create ()

This procedure without parameters creates NLP task descriptor. The content of the task is empty at the moment. The user can link some objects to this task later, filling its components.

Object-oriented approach, based on object descriptors, makes it possible to define several different tasks of the same type simultaneously. They may refer the same objects as task components, for example, function to be minimized. This is one more powerful potential source for multiprocessing which was mentioned above.

Not all the objects in a task structure must be necessarily defined by the user as input objects. For example, if a task descriptor has empty reference to, say, vector of independent variables, then the package itself will make memory allocation for it.

## 5. Dialogue clusters.

Supervising program of numeric package plays first of all role of a menu to access all the resources of a given model. But this role is not the only one. Problem solving can't be successfully fulfilled by simple package procedure calls and data access. Package must provide the user with a set of dialogue schemes and scenarios to analyze task properties and to solve the problem.

We introduce new programming item: dialogue cluster. Dialogue cluster is an extension of abstract data types for such cases, when, due to high complexity of task to be solved, it is not enough to supply the user with ordinary abstract cluster capabilities.

From the user point of view model consists of interrelated set of abstract objects: goal function, initial vector, gradient of function etc. There exists a set of procedures to evaluate objects values. The user can explore function properties or initiate numeric method. If a method succeeds then all is okay. But some exception may occur, and it is the most interesting (and, in fact, rather typical) situation for numeric packages.

As it was mentioned above, exception is raised, when current situation goes out of model domain of definition. But when a particular method rejects the task, it is not fatal, and the fact itself very often gives the user a good idea, how to continue problem solving. It explains the main goal of dialogue cluster: to ensure effective problem solving when any separate resource can't manage the situation by itself. In other words, dialogue cluster is a means to solve problems with the use of numeric algorithms, which do not have strict domain of definition.

Dialogue cluster semantic specification includes definition of its dialogue capabilities. Semantics of every cluster object is fully defined both by its procedural interface and by dialogue commands and scenarios, which use this object as an argument. For example, the notion of vector space is properly defined in dialogue cluster, if there exists a list of dialogue commands, such as "create vector space" etc.

From the implementation point of view every dialogue cluster for different optimization models is represented by BSP capabilities, numeric algorithms in the library, and by package resources. To invoke cluster the user calls package supervising program (dialogue monitor) with task descriptor as a parameter.

## 6. Conclusion.

Numeric software is an area, which greatly influences very many branches of science. Efficient numeric package can greatly improve and accelerate scientific research. Recent

efforts in numeric software is focused mostly on procedural part of algorithms implementation. The problems of system design are not so much developed. This paper shows some problems in numerical software design. Integrated model-oriented approach, which was described in this paper, proved to be highly efficient principle of software design. It was used as a leading principle in the DISO project, which is being developed since 1980 in the Computer Center of the USSR Academy of Sciences.

## REFERENCES.

[1]. Lotus Development Inc. (1984). LOTUS 1-2-3 and SYMPHONY Integrated Systems. User's Manuals.

[2]. Aston Tate Inc. (1985). FRAMEWORK Integrated System. User's Manual.

[3]. JAVELIN. (1986). User's Manual.

[4]. Evtushenko Yu.G. (1985). Numerical optimization technique. Springer Verlag. New-York.

[5]. Rall L.B. (1981). Automatic Differentiation: Technique and Applications. Lecture Notes in Computer Science. Vol. 120. Springer Verlag.

[6]. Mazourik V. (1987). Field Manager Application Package. IIASA Working Paper, to appear.