Interval Computations
No 2, 1991

Паскаль–XSC: новый язык для научных вычислений

Р. Хаммер, М. Неага, Д. Рац, Д. Ширяев

Представлен новый язык программирования Паскаль–XSC, упрощающий разработку программ для инженерно–научных вычислений за счет модульной структуры программ, определяемых пользователем операторов, совмещения функций, процедур и операторов, динамических массивов, стандартных модулей для дополнительных численных типов данных с операторами максимальной аккуратности, стандартных функций высокой аккуратности и вычисления выражений без ошибок округления.

PASCAL–XSC: A new language for scientific computing

R. Hammer, M. Neaga, D. Ratz, D. Shiryaev

A new programming language PASCAL–XSC is discussed. The language simplifies the design of programs in engineering/scientific computations by modular program structure, user-defined operators, overloading of functions, procedures and operators, dynamic arrays, standard modules for additional numerical data types with operators of highest accuracy, standard functions of high accuracy and exact evaluation of expressions. It is available for supercomputers, mainframes, workstations and personal computers by means of a portable implementation in ANSI C.

1. Введение

В настоящее время элементарные арифметические операции электронных компьютеров обычно аппроксимируются операциями с плавающей точкой максимальной аккуратности. (Определение термина аккуратность смотри в разделе 3.6.) В частности это означает, что при любом выборе операндов вычисленный

© R. Hammer, M. Neaga, D. Ratz, D. Shiryaev, 1991.

результат совпадает с округленным точным результатом операции. (Смотри стандарт арифметики IEEE [3] в качестве примера.) Этот стандарт требует также наличия четырех основных арифметических операций $+, -, *$, и / с направленными округлениями. Большое количество имеющихся процессоров предоставляет эти операции. Однако до сих пор ни один обычный язык программирования не обеспечивает доступ к ним.

В последнее время в научных вычислениях произошел значительный сдвиг от компьютеров общего назначения к использованию векторных и параллельных компьютеров. Так называемые супер-компьютеры обеспечивают дополнительные арифметические операции типа "умножить и сложить" и "аккумулировать" или "умножить и аккумулировать" (смотри [9]). Названные аппаратные операции должны всегда выдавать результат максимальной аккуратности, однако до сих пор не существует ни одного процессора, выполняющего эти требования. Иногда результаты численного алгоритма, вычисленного на векторном процессоре, полностью отличаются от результатов, вычисленных на скалярном процессоре (смотри [11],[27]).

Не раз предпринимались попытки расширить мощность языков программирования. Были спроектированы новые мощные языки, например, как ADA, а расширение существующих языков, таких как Фортран, непрерывно продолжается. Однако, поскольку в этих языках до сих пор отсутствует точное определение их арифметики, одинаковые программы могут выдавать различный результат на различных процессорах.

Язык Паскаль-XSC – это результат долговременной работы коллектива ученых с целью создания средства для корректного решения научных проблем. Математическое определение арифметики является неотъемлемой частью языка. В нем прямо доступны оптимальные арифметические операции с направленными округлениями. Кроме того, с максимальной аккуратностью согласно законам семиморфизма (см. [21]) определены арифметические операции для интервалов и комплексных чисел и даже векторно-матричные операции, представляемые прекомпилированными арифметическими модулями.

2. Язык Паскаль–XSC

Паскаль–XSC (PASCAL eXtension for Scientific Computation) – это расширение языка программирования Паскаль для научных вычислений. Первая реализация такого расширения (Паскаль–SC) появилась в 1980 году. Спецификации расширения непрерывно улучшались, добавлялись существенные языковые концепции, в результате чего родился новый язык Паскаль–XSC [16],[17]. Новый транслятор [1], реализованный на языке C и использующий C в качестве целевого языка, делает Паскаль–XSC доступным на персональных компьютерах, рабочих станциях, больших ЭВМ, и суперкомпьютерах. Паскаль–XSC отличается следующими чертами:

- Стандартный Паскаль
- Универсальная концепция оператора (определяемые пользователем операторы)
- Функции и операторы с произвольным типом результата
- Совмещение процедур, функций и операторов
- Концепция модульности
- Динамические массивы
- Доступ к подмассивам
- Концепция строк
- Контролируемое округление
- Оптимальное скалярное произведение
- Стандартный тип *dotprecision* (формат с фиксированной точкой для покрытия всего возможного диапазона произведений с плавающей точкой)
- Дополнительные стандартные арифметические типы, такие как *complex*, *interval*, *rvector*, *rmatrix* и т.д.
- Высокоаккуратная арифметика для всех стандартных типов
- Высокоаккуратные встроенные функции
- Вычисление выражений с максимальной аккуратностью (#-выражения)

3. Реализация Паскаля–XSC

Начиная с 1976 года, определение и развитие Паскаля для научных вычислений осуществляется в Институте Прикладной Математики университета города Карлсруэ (ФРГ). Был разработан компилятор с Паскаля-SC для нескольких типов компьютеров (процессоры Z80, 8088, и 68000) под различными операционными системами. Этот компилятор уже продавался для IBM PC/AT и ATARI-ST (смотри [18], [19]).

В настоящее время новый транслятор Паскаля–XSC [1] доступен для персональных компьютеров, рабочих станций, больших ЭВМ и суперкомпьютеров. Основной целью новой реализации было создание полностью переносимой системы программирования. Компилятор реализован на языке ANSI C, и в качестве его целевого языка использован также язык ANSI C. Этот выбор объясняется чрезвычайно широкой его распространенностью и заложенными в стандарт ANSI C возможностями создания переносимого кода. Благодаря этому, а также библиотеке поддержки, реализованной на языке C, программы на языке Паскаль–XSC могут быть использованы на всех системах практически в неизменном виде. Таким образом, существует возможность разработки программы, например, на персональном компьютере и дальнейшей работы с ней на большой ЭВМ, используя тот же самый компилятор. Впервые реализована возможность сменной вещественной арифметики в следующих вариантах:

- аппаратная арифметика используемого компьютера
- программная эмуляция арифметики стандарта IEEE 754
- произвольная определенная пользователем арифметика

В настоящее время система программирования Паскаль–XSC доступна на следующих компьютерах: IBM PS/2, IBM RT PC, HP 9000, HP VECTRA, SUN, CONVEX, PARSYTEC (TRANSPUTER), ATARI-ST, VAX, IBM 370. Под MS- или PC-DOS для MICROSOFT C, WATCOM C, TURBO C++, METAWARE C, GNU C. Описание системы можно найти в [1] и [2].

Полное описание языка Паскаль–XSC вместе с набором учебных программ дано в [16] и [17].

3.1. Стандартные типы данных, предопределенные операторы и функции

Кроме типов данных стандартного Паскаля в Паскале–XSC имеются следующие численные типы данных:

interval	*complex*	*cinterval*	
rvector	*cvector*	*ivector*	*civector*
rmatrix	*cmatrix*	*imatrix*	*cimatrix*

где префиксы *r*, *i*, и *c* обозначают (по начальным буквам) слова *real*, *interval*, и *complex*. Таким образом, *cinterval* означает *complex interval*, *cimatrix* комплексные интервальные матрицы, и *rvector* вещественные векторы. Векторные и матричные типы определены как динамические массивы и могут быть использованы при любом диапазоне индексов.

Для этих типов в арифметических модулях Паскаля–XSC предопределено большое количество операторов (смотри раздел 3.8). Все эти операторы вырабатывают результат максимальной аккуратности. В таблице 1 перечислены 29 предопределенных стандартных операторов Паскаля–XSC в соответствии с приоритетом.

Тип	Приоритет	Операторы
унарный	3 (самый высокий)	унарный +, унарный −, **not**
мультипликативный	2	**and**, **div**, **mod** *, *<, *>, /, /<, />, **
аддитивный	1	**or** +, +<, +>, −, −<, −>, +*
условные	0 (самый низкий)	**in** =, <>, <=, <, >=, >, ><

Таблица 1. Старшинство встроенных операторов.

По сравнению со стандартным Паскалем появляется 11 новых

операторных символов. Это операторы ∘< и ∘>, ∘ ∈ {+, −, *, /} для операций с вниз и вверх направленными округлениями и операторы **, +*, >< , необходимые в интервальных вычислениях для проверок на пересекаемость, выпуклую оболочку и непересекаемость.

В таблицах 2 и 3 перечислены все предопределенные арифметические и условные операторы относительно возможных комбинаций типов операндов.

правый операнд / левый операнд	integer real complex	interval cinterval	rvector cvector	ivector civector	rmatrix cmatrix	imatrix cimatrix
унарный[1]	+, −	+, −	+, −	+, −	+, −	+, −
integer real complex	∘, ∘<, ∘>, +* [2]	+, −, *, /, +*	*, *<, *>	*	*, *<, *>	*
interval cinterval	+, −, *, /, +*	+, −, *, /, +*, **	*	*	*	*
rvector cvector	*, *<, *>, /, /<, />	*, /	∘, ∘<, ∘>, +* [3]	+, −, *, /, +* [4]		
ivector civector	*, /	*, /	+, −, *, /, +* [4]	+, −, *, /, +*, ** [4]		
rmatrix cmatrix	*, *<, *> /, /<, />	*, /	*, /, *<, *>	*	∘, ∘<, ∘>, +* [3]	+, −, *, +* [4]
imatrix cimatrix	*, /	*, /	*	*	+, −, *, +* [4]	+, −, *, +*, ** [4]

[1]) Операторы в этом ряду унарные (т.е. отсутствует левый операнд).

[2]) ∘ ∈ {+, −, *, /}

[3]) ∘ ∈ {+, −, *}, где * обозначает скалярное или матричное произведение.

[4]) * обозначает скалярное или матричное произведение.

+* : интервальная оболочка.

** : интервальное пересечение.

Таблица 2. Предопределенные арифметические операторы.

В отличие от стандартного Паскаля, Паскаль–XSC располагает более широким набором встроенных математических функций (смотри таблицу 4). Эти функции с родовым именем доступны для типов *real*, *complex*, *interval*, и *cinterval* и вырабатывают результат максимальной аккуратности. Функции для типов *complex*, *interval*, и *cinterval* предоставляются арифметическими модулями Паскаля–XSC.

правый операнд / левый операнд	integer real complex	interval cinterval	rvector cvector	ivector civector	rmatrix cmatrix	imatrix cimatrix
integer real complex	=, <>, <=, <, >=, >	in, =, <>				
interval cinterval	=, <>	in, ><, [1] =, <>, <=, <, >=, >				
rvector cvector			=, <>, <=, <, >=, >	in, =, <>		
ivector civector			=, <>	in, ><, [1] =, <>, <=, <, >=, >		
rmatrix cmatrix					=, <>, <=, <, >=, >	in, =, <>
imatrix cimatrix					=, <>	in, ><, [1] =, <>, <=, <, >=, >

[1] Операторы <= и < означают отношение “подмножества”
>= и > означает отношение “супермножества”.
V ∈ {=, <>, <, <=, >, >=}.
>< : Проверка непересекаемости интервалов.
in : Проверка наличия точки в интервале или проверка на строгое включение интервала в другой интервал.

Таблица 3 Предопределенные условные операторы.

	Функция	Родовое имя	Тип аргумента
1	Абсолютное значение	abs	*
2	Арккосинус	arccos	*
3	Арккотангенс	arccot	*
4	Обратный гиперболический косинус	arcosh	*
5	Обратный гиперболический котангенс	arcoth	*
6	Арксинус	arcsin	*
7	Арктангенс	arctan	*
8	Обратный гиперболический синус	arsinh	*
9	Обратный гиперболический тангенс	artanh	*
10	Косинус	cos	*
11	Котангенс	cot	*
12	Гиперболический косинус	cosh	*
13	Гиперболический котангенс	coth	*
14	Экспонента	exp	*
15	Степенная функция (По основанию 2)	exp2	*
16	Степенная функция (По основанию 10)	exp10	*
17	Натуральный логарифм (По основанию e)	ln	*
18	Логарифм (По основанию 2)	log2	*
19	Логарифм (По основанию 10)	log10	*
20	Синус	sin	*
21	Гиперболический синус	sinh	*
22	Возведение в квадрат	sqr	*
23	Квадратный корень	sqrt	*
24	Тангенс	tan	*
25	Гиперболический тангенс	tanh	*

Таблица 4. Стандартные математические функции
(* включает типы *integer*, *real*, *complex*, *interval*, и *cinterval*)

Кроме стандартных математических функций, Паскаль–XSC предоставляет необходимые функции преобразования типа *intval*, *inf*, *sup*, *compl*, *re*, и *im* для взаимных преобразований численных типов данных (для скалярных и векторных типов).

3.2. Общая концепция оператора

Продемонстрируем на простом примере сложения интервалов преимущества общей концепция оператора. При невозможности определять операторы пользователем имеются два способа реализации сложения двух интервалов, задаваемых следующим описанием типа

```
type interval = record inf,sup: real end;
```

Можно использовать описание процедуры

```
procedure intadd(a,b: interval; var c: interval);
begin
        c.inf := a.inf +<b.inf;
        c.sup := a.sup +>b.sup
end;
```

математическая нотация	соответствующий оператор программы

$z := a + b + c + d$	intadd(a,b,z);
	intadd(z,c,z);
	intadd(z,d,z);

или описание функции (что возможно только в Паскале–XSC, но не в стандартном Паскале)

```
function intadd(a,b: interval): interval;
begin
        intadd.inf := a.inf +<b.inf;
        intadd.sup := a.sup +>b.sup
end;
```

математическая нотация	соответствующий оператор программы
$z := a + b + c + d$	z := intadd(intadd(intadd(a,b),c),d);

В обоих случаях описания математических формул выглядят достаточно громоздко. Реализуя оператор в Паскале–XSC

```
operator + (a,b: interval) intadd: interval;
begin
        intadd.inf := a.inf +<b.inf;
        intadd.sup := a.sup +>b.sup
end;
```

математическая нотация	соответствующий оператор программы
$z := a + b + c + d$	z := a+b+c+d;

мы видим, что многократное сложение интервалов описывается в традиционной математической нотации. Можно не только совмещать операторные символы, но и использовать именованные операторы. Этим операторам должно предшествовать описание приоритета. Существуют четыре различных уровня приоритета, каждый представляемый своим собственным символом:

- унарный : ↑ уровень 3 (самый высокий)
- мультипликативный: * уровень 2
- аддитивный : + уровень 1
- условный : = уровень 0

Оператор для вычисления биномиального коэффициента $\binom{n}{k}$ может быть определен следующим образом:

```
priority choose = *;            {описание приоритета }
operator choose (n,k: integer) binomial: integer;
var i,r : integer;
begin
        if k > n div 2 then k := n−k;
        r := 1;
        for i := 1 to k do
                r := r * (n − i + 1) div i;
        binomial := r;
end;
```

математическая нотация	соответствующий оператор программы
$c := \binom{n}{k}$	c := n choose k

Концепция оператора, реализованная в Паскале–XSC, предоставляет следующие возможности:

- определение произвольного количества операторов
- совмещение операторных символов или имен операторов произвольное количество раз
- реализацию рекурсивно определенных операторов

Распознавание подходящего оператора зависит прежде всего от числа операндов, а при совпадении числа операндов от их типов согласно следующему правилу взвешивания:

Если список фактических параметров подходит для двух различных операторов, то выбирается тот оператор, у которого первый параметр "более подходящий". Это означает, что типы операндов должны совпадать, а не просто приводиться друг к другу.

Пример:

```
operator +* (a: integer; b: real) irres: real;
          ⋮
operator +* (a: real; b: integer) rires: real;
          ⋮
var x : integer;
    y, z : real;
          ⋮
    z := x +* y; ⟹ 1. оператор
    z := y +* x; ⟹ 2. оператор
    z := x +* x; ⟹ 1. оператор
    z := y +* y; ⟹ невозможно !
```

Паскаль–XSC также делает возможным совмещение оператора присваивания :=. Благодаря этому математическая нотация может быть использована и для присваиваний.

Пример:

```
var
        c : complex;
        r : real;
        ⋮
operator := (var c: complex; r: real);
        begin
                c.re := r;
                c.im := 0;
        end;
        ⋮
        r := 1.5;
        c := r; {комплексное число с вещественной частью 1.5
           и мнимой частью 0}
```

3.3. Совмещение подпрограмм

Стандартный Паскаль предоставляет стандартные математические функции

sin, cos, arctan, exp, ln, sqr, и *sqrt*

только для чисел типа *real*. Для реализации функции синус с интервальным аргументом, необходимо использовать имя функции, подобное *isin(...)* , поскольку переопределение стандартного имени функции *sin* не разрешено в стандартном Паскале.

В Паскале–XSC разрешено совмещение имен функций и процедур при помощи концепции родовых имен. Таким образом, имена

sin, cos, arctan, exp, ln, sqr и *sqrt*

могут быть использованы не только для чисел типа *real*, но и для интервалов, комплексных чисел и других математических пространств. Для распознавания различия между совмещенны-

ми функциями или процедурами с одним и тем же именем используются число, тип и взвешивание их аргументов подобно методу для операторов. Тип результата, однако, *не* учитывается.

Пример:

```
procedure rotate (var a,b: real);
procedure rotate (var a,b,c: complex);
procedure rotate (var a,b,c: interval);
```

В слегка модифицированном виде концепция совмещения применима также к стандартным процедурам *read* и *write*. Первым параметром вновь объявленной процедуры ввода/вывода должен быть **var**-параметр типа файл, второй параметр представляет величину, предназначенную к вводу/выводу. Все последующие параметры интерпретируются как спецификации формата.

Пример:

```
procedure write (var f: text; c: complex; w: integer);
begin
        write (f, '(', c.re : w, ',', c.im : w, ')');
end
```

При вызове совмещенной процедуры ввода/вывода первый параметр может быть опущен, если происходит обращение к стандартному файлу *input* или *output*. Спецификации формата вводятся и разделяются двоеточием. Кроме того, несколько операторов ввода или вывода могут быть скомбинированы в один оператор, как и в стандартном Паскале.

Пример:

```
var
        r: real;
        c: complex;
        :
write (r : 10, c : 5, r/5);
```

3.4. Концепция модульности

В стандартном Паскале предполагается, что программа состоит из одного куска текста, который должен быть полностью подготовлен перед компиляцией и выполнением. Во многих случаях бывает удобнее разделить программу на несколько частей, называемых модулями, разрабатывающиеся и компилирующиеся независимо друг от друга. Кроме того, несколько разных программ могут использовать компоненты модуля без их копирования в исходный код и без перекомпиляции. Для этой цели в Паскаль–XSC была введена концепция модульности, которая делает возможным

- модульное программирование
- синтаксический и семантический анализ за пределами модуля
- реализацию арифметических пакетов как стандартных модулей

module	: начинает новый модуль
global	: указывает, что объект является глобальным
use	: указывает импортируемые модули

Модуль начинается словом **module**, за которым следуют имя и точка с запятой. Тело модуля строится подобно телу нормальной программы, однако слово **global** может стоять перед словами **const**, **type**, **var**, **procedure**, **function**, **operator** и непосредственно после **use** и знака равенства в объявлении типа.

Таким образом, можно объявлять личные и глобальные типы. Структура личного типа неизвестна за пределами модуля и к ней можно обращаться, только вызывая подпрограммы. Если, например, внутренняя структура, также как и имя типа, должны стать глобальными, то слово **global** повторяется после знака равенства. При объявлении

global type complex = **global record** re, im : real **end;**

complex и его внутренняя структура как запись с компонентами *re* и *im* становятся глобальными.

Личный тип *complex* может быть объявлен как

global type complex = **record** re, im: real **end;**

Пользователь, импортирующий личный тип, не может ссылаться на компоненты записи, потому что структура типа скрыта внутри модуля.

Модуль строится по следующему образцу:

module M1;

use < другие модули >;

< глобальные и локальные объявления >

begin

< инициализация модуля >

end.

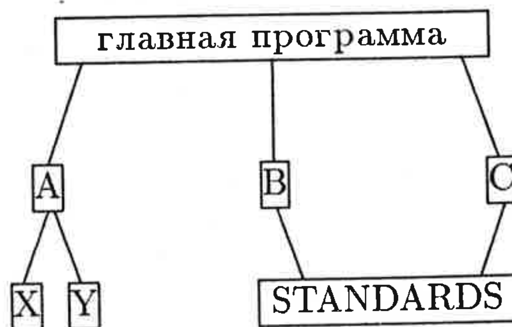
При импорте модулей с помощью **use** или **use global** выполняются следующие законы транзитивности:

M1 **use** M2 и M2 **use global** M3 $\Rightarrow$ M1 **use** M3,

но

M1 **use** M2 и M2 **use** M3 $\not\Rightarrow$ M1 **use** M3.

Пример: Допустим, иерархия модулей выглядит следующим образом:



Все глобальные объекты в модулях A, B, и C видимы в главной программе, но доступ к глобальным объектам X, Y и STANDARDS отсутствует. Существуют две возможности сделать последние видимыми также в этой главной программе :

1) написать

use X, Y, STANDARDS

в главной программе

2) написать

use global X, Y

в модуле A и

use global STANDARDS

в модуле B или C.

3.5. Динамические массивы

В стандартном Паскале невозможно объявлять динамические типы или переменные. В частности, пакет программ с векторными и матричными операциями может быть реализован только для фиксированной (максимальной) размерности. Поэтому используется лишь часть отведенной памяти, если требуется только меньшая размерность. Концепция динамических массивов снимает это ограничение и позволяет:

- Динамику внутри процедур и функций
- Автоматическое размещение и освобождение локальных динамических переменных
- Экономное использование памяти
- Доступ к строкам и столбцам динамических массивов
- Совместимость статических и динамических массивов

Динамические массивы вводятся словом **dynamic**. Большой недостаток схемы *конформантных массивов* в стандартном Паскале в том, что они могут быть использованы только в качестве параметров, но не переменных или результата функции. Подобное использование не является, естественно, полностью динамическим.

В Паскале–XSC динамические и статические массивы равноправны. В настоящий момент динамические массивы не могут быть компонентами других структур данных. Синтаксически это означает, что слово **dynamic** может стоять только непосредственно после знака равенства в определении типа или непосредственно после двоеточия при объявлении переменной. Динами-

ческий массив не может быть компонентой записи.

Тип, соответствующий двумерному массиву, может быть определен следующим образом:

type matrix = **dynamic array**[*,*] **of** real;

Возможно также определение различных динамических типов с совпадающей синтаксической структурой. Например, в некоторых ситуациях могло бы быть полезным отождествлять коэффициенты полинома с компонентами вектора и наоборот. Поскольку Паскаль строго типизированный язык, подобные структурно-эквивалентные массивы совместимы только, если их типы предварительно приведены. Следующий пример показывает определение типов полином и вектор (обратите внимание, что функции приведения типа *polynomial(...)* и *vector(...)* определены неявно):

```
type vector = dynamic array[*] of real;
type polynomial = dynamic array[*] of real;
operator + (a,b: vector) res: vector[lbound(a)..ubound(a)];
          ⋮
var v : vector[1..n];
    p : polynomial[0..n-1];
          ⋮
          v := vector(p);
          p := polynomial(v);
          v := v + v;
          v := vector(p) + v; { но не v := p + v; }
```

Доступ к верхней и нижней границе размерности обеспечивается новыми встроенными функциями *lbound(...)* и *ubound(...)*, второй необязательный аргумент которых указывает искомую размерность динамической переменной. Используя эти функции, указанный выше оператор может строиться следующим образом.

```
operator + (a,b: vector) res: vector[lbound(a)..ubound(a)];
var i : integer;
begin
        for i := lbound(a) to ubound(a) do
                res[i] := a[i] + b[lbound(b) + i – lbound(a)]
end;
```

Введение динамических типов требует расширения условий совместимости. Как и в стандартном Паскале, в Паскале–XSC два векторных типа несовместимы, если они не одного и того же типа. Следовательно, динамический векторный тип не совместим со статическим типом. В Паскале–XSC присваивание значений всегда возможно в случаях, приведенных в таблице 5.

Тип левой стороны	Тип правой стороны	Присваивание разрешено
анонимный динамический	произвольный *векторный*	если структурно эквивалентны
известный динамический	известный динамический	если типы совпадают
анонимный статический	произвольный *векторный*	если структурно эквивалентны
известный статический	известный статический	если типы совпадают

Таблица 5. Совместимость присваиваний.

В остальных случаях присваивание возможно только при совпадении типов левой и правой стороны (детали смотри в [16] или в [17]).

Кроме доступа к каждой компоненте массива, Паскаль–XSC дает возможность доступа к подмассивам. Если индекс содержит вместо выражения $*$, то он ссылается на подмассив с диапазоном индексов соответствующей размерности, например, j-й столбец матрицы m доступен через $m[*, j]$. Следующий пример демонстрирует доступ к строкам и столбцам динамического массива:

```
type vector = dynamic array[*] of real;
type matrix = dynamic array[*] of vector;
        ⋮
var v : vector[1..n];
```

```
m : matrix[1..n,1..n];
        ⋮
    v := m[i];
    m[i] := vector(m[*, j]);
```

В первом присваивании не нужно использовать функцию, адаптирующую тип, поскольку оба операнда, слева и справа, одного и того же *известного динамического* типа. Во втором случае операнд слева *известного динамического* типа, а операнд справа *анонимного динамического* типа, поэтому необходимо использовать встроенную функцию преобразования типа *vector(...)*.

Паскаль–XSC – программа с использованием динамических массивов строится по следующей схеме:

```
program dynprog (input,output);
type
        vector = dynamic array[*] of real;
        <различные динамические определения>
var n : integer;
{-----------------------------------------}
procedure main (dim: integer);
var a,b,c : vector[1..dim];
        ⋮
begin
        <ввод/вывод в зависимости от значения dim>
                ⋮
        c := a + b;
                ⋮
end;
```

```
{-------------------------------------}
begin {главная программа}
        read(n);
        main(n);
end. {главная программа}
```

Только первоначальная главная программа должна быть оформлена в виде процедуры (здесь: *main*), к которой обращаются, передавая размерность динамического массива в качестве параметра.

3.6. Аккуратные выражения

Введем несколько определений . **Точность** системы с плавающей точкой (и числа, представимого в ней) прямо пропорциональна числу цифр в мантиссе и является таким образом понятием относительным. Поскольку нас интересует качество результата вычисления, инвариантное относительно используемого формата числа, введем термин *аккуратность* результата вычисления. **Аккуратность** обратно пропорциональна относительной ошибке результата. **Высокая аккуратность** результата подразумевает гарантированно небольшую относительную ошибку по сравнению с точным математическим решением, которое является *абстрактным вещественным* числом. В вычислениях с плавающей точкой это обычно означает ошибку не большую, чем две единицы последнего разряда мантиссы. **Максимально аккуратный** результат вычисления должен быть или точным математическим решением, если оно представимо, или ближайшим представимым числом снизу или сверху от него. Выбор может зависеть от режима округления. Таким образом, между точным математическим решением и вычисленным результатом не должно быть какого-либо представимого числа.

Разработка алгоритмов включения с автоматической верификацией результата (смотри [14],[20],[24],[29]) делает необходимым обширное использование максимально аккуратного вычисления скалярных произведений, обладающего следующим свойством (смотри [21]):

$$\text{(RG)}\ a \odot b := \bigcirc \sum_{i=1}^{n} a_i \cdot b_i,\ \bigcirc \in \{\square, \triangle, \triangledown\},\ n \in \mathbf{N}.$$

Для вычисления подобных выражений был предложен тип данных *dotprecision*. Этот тип данных покрывает весь диапазон чисел с плавающей точкой с двойной экспонентой и называется иногда *длинным аккумулятором (ДА)*. (смотри [21],[20]). Основываясь на этом типе, так называемые *аккуратные выражения* (#-выражения) могут быть обозначены указанием символа (#, #*, #<, #> или ##), за которым следует *выражение, вычисляемое без ошибок округления*, заключенное в скобки. Это выражение должно иметь форму, подобную скалярному произведению (СП–выражение). Следующие встроенные операции доступны для *ДА*–типа:

- преобразование *вещественных* и *целых* значений к *ДА* (#)
- округление *ДА*–значений до *вещественных*; в особенности вниз направленное округление (#<), вверх направленное округление (#>) и округление к ближайшему. (#*)
- округление *ДА*–выражения до ближайшего включающего интервала (##)
- сложение *вещественных* чисел или произведения двух *вещественных* чисел с переменной типа *ДА*
- суммирование скалярного произведения с переменной типа *ДА*
- сложение и вычитание *ДА*–чисел
- унарный минус для *ДА*–чисел
- стандартная функция *sign*, возвращающая −1, 0 или +1, в зависимости от знака *ДА*–числа

Для получения неокругленного или правильно округленного результата СП-выражения нужно заключить выражение в скобки и поставить перед ним символ #, за которым может следовать необязательный символ, указывающий режим округления. В таблице 6 даны возможные режимы округлений в зависимости от формы СП–выражения (Детали даны в приложении на странице 29).

Символ	Форма выражения	Режим округления
#*	скалярное, векторное или матричное	к ближайшему
#<	скалярное, векторное или матричное	вниз
#>	скалярное, векторное или матричное	вверх
##	скалярное, векторное или матричное	наименьший объемлющий интервал
#	только скалярное	точно, без округления

Таблица 6. Режимы округления для аккуратных выражений.

На практике СП–выражения могут содержать большое число слагаемых, что сильно затрудняет их написание. Для устранения этого неудобства в математике используется символ $\sum$. Если, например, A и B n-размерные матрицы, то

$$\sum_{k=1}^{n} A(i,k) \cdot B(k,j)$$

является СП–выражением. Паскаль–XSC обеспечивает для этой цели эквивалентную сокращенную нотацию **sum**. Соответствующий этому выражению Паскаль–XSC – оператор выглядит следующим образом:

D := #(**for** k:=1 **to** n **sum** (A[i,k]*B[k,j]))

где D является *ДА*– переменной .

Аккуратные выражения используются главным образом при вычислении невязок. Пусть, например, $Ay \approx b$ в системе линейных уравнений $Ax = b$, $A \in \mathbb{R}^{n \times n}$, $x, b \in \mathbb{R}^n$. Тогда включением невязки является $\Diamond(b - Ay)$, реализованное в Паскале–XSC через

##(b – A*y);

только с одним интервальным округлением для каждой компоненты. Для получения верифицированного включения решения системы линейных уравнений необходимо вычислить выражение

$\Diamond(E - RA)$,

где E – единичная матрица и $R \approx A^{-1}$. В Паскале-XSC это выражение программируется следующим образом:

(id(A) – R*A);

причем интервальная матрица вычисляется только с одним округлением для каждой компоненты. Функция *id*(...) является частью модуля для вещественной матрично/векторной арифметики и генерирует единичную матрицу соответствующей размерности в зависимости от формы своего аргумента (смотри раздел 3.8).

3.7. Концепция строк

Средства для обработки строк, предоставляемые стандартным Паскалем, не обеспечивают удобной обработки текста. По этой причине в определение языка была введена концепция строк, позволяющая не только более удобную обработку текстовой информации, но и символические вычисления. Используя новый тип данных *string*, можно работать со строками длиной до 255 символов. При условии, что длина строки не выходит за этот диапазон, она может быть указана явно при определении строки. Таким образом, строка *s*, определенная посредством

var s: string[40];

может иметь длину до 40 символов. Возможны следующие стандартные операции:

- конкатенация

  operator + (a,b: string) conc: string;
- фактическая длина строки

  function length(s: string): integer;
- преобразование *string* → *real*

  function rval(s: string): real;
- преобразование *string* → *integer*

  function ival(s: string): integer;
- преобразование *real* → *string*

  function image(r: real; width,fracs,round: integer): string;
- преобразование *integer* → *string*

  function image(i,len: integer): string;
- выделение подстроки

  function substring(s: string; i,j: integer): string;

- позиция первого вхождения подстроки

  function pos(sub,s: string): integer;

- условные операторы <=, <, >=, >, <>, =, и **in**

3.8. Стандартные модули

Доступны следующие стандартные модули:

- интервальная арифметика (I_ARI)
- комплексная арифметика (C_ARI)
- комплексно-интервальная арифметика (CI_ARI)
- вещественная матрично–векторная арифметика (MV_ARI)
- интервальная матрично–векторная арифметика (MVI_ARI)
- комплексная матрично–векторная арифметика (MVC_ARI)
- комплексно-интервальная матрично–векторная арифметика (MVCI_ARI).

Эти модули могут быть импортированы оператором **use**, описанном в разделе 3.4. В таблице 7 перечислены все операторы, обеспечиваемые модулем для интервальной матрично–векторной арифметики .

правый операнд / левый операнд	integer real	interval	rvector	ivector	rmatrix	imatrix
унарный				+, −		+, −
integer real				*		*
interval			*	*	*	*
rvector		*, /	+*	+*, +, −, *, **in**, =, <>		
ivector	*, /	*, /	+*, +, −, *, =, <>	+*, **, +, −, *, **in**, =, <>, ><, <=, <, >=, >		
rmatrix		*, /		*	+*	+*, +, −, *, **in**, =, <>
imatrix	*, /	*, /	*	*	+*, +, −, *, =, <>	+*, **, +, −, *, **in**, =, <>, ><, <=, <, >=, >

Таблица 7. Предопределенные арифметические и условные операторы модуля MVI_ARI.

В дополнение к этим операторам модуль MVI_ARI предоставляет следующие встроенные операторы, функции и процедуры с родовыми именами

intval, inf, sup, diam, mid, blow, transp, null, id, read и *write*.

Функция *intval* используется для генерации интервальных векторов и матриц соответственно, в то время как функции *inf* и *sup* выбирают инфимум и супремум интервального объекта. Диаметр и середина интервальных векторов и матриц определяется через *diam* и *mid*; *blow* раздувает интервал; а *transp* используется для получения транспонированной матрицы.

Нулевые векторы и матрицы генерируются функцией *null*, в то время как *id* возвращает единичную матрицу подходящей формы. Наконец, имеются родовые процедуры ввода/вывода *read* и *write*, которые могут быть использованы для всех матрично/векторных типов данных, определенных в вышеупомянутых модулях.

3.9. Прикладные подпрограммы

Паскаль-XSC – подпрограммы для решения традиционных численных задач предоставляются дополнительной библиотекой модулей. Используются методы, вычисляющие чрезвычайно аккуратное включение решения задачи и в то же время доказывающие существование и единственность решения в заданном интервале. Перечислим преимущества новых подпрограмм :

- Решение вычисляется с максимальной или большой, но всегда контролируемой аккуратностью, даже в случае плохой обусловленности.
- Правильность результата автоматически верифицируется, т.е. включающее множество вычисляется с гарантией существования и единственности точного решения внутри его.
- Если решения не существует или задача очень плохо обусловлена, выдается сообщение об ошибке.
- Эти подпрограммы могут быть использованы неспециалистами, поскольку у пользователя практически отсутствует возможность сделать ошибку или неправильно интерпретировать что либо.

В частности, Паскаль–XSC - подпрограммы покрывают следующие области:

- системы линейных уравнений
  - полные системы (*real*, *complex*, *interval*, *cinterval*)
  - обращение матриц (*real*, *complex*, *interval*, *cinterval*)
  - проблема наименьших квадратов (*real*, *complex*, *interval*, *cinterval*)
  - вычисление псевдообращений (*real*, *complex*, *interval*, *cinterval*)
  - ленточные матрицы (*real*)
  - разреженные матрицы (*real*)
- вычисление полиномов
  - одной переменной (*real*, *complex*, *interval*, *cinterval*)
  - многих переменных (*real*)
- нули полиномов (*real*, *complex*, *interval*, *cinterval*)
- собственные числа и собственные векторы
  - симметрические матрицы (*real*)
  - произвольные матрицы (*real*, *complex*, *interval*, *cinterval*)
- Краевые задачи и задачи Коши для обыкновенных дифференциальных уравнений
  - линейных
  - нелинейных
- вычисление арифметических выражений
- системы нелинейных уравнений
- нечисленные квадратуры
- интегральные уравнения
- автоматическое дифференцирование

4. Примеры Паскаль–XSC программ

Приведем пример полной Паскаль–XSC – программы, которая демонстрирует использование некоторых арифметических модулей. Используя модуль LIN_SOLV, решение системы линейных уравнений заключается в интервальный вектор путем последовательных интервальных итераций.

Процедура *main*, которая вызывается в теле *lin_sys*, нужна только для ввода размерности системы и размещения динамических переменных. Численный метод начинается вызовом процедуры *linear_system_solver*, определенной в модуле LIN_SOLV. Эта процедура может быть вызвана с массивом произвольной размерности в качестве параметра.

Более подробную информацию об итеративных методах с автоматической верификацией результата можно найти, например, в [14],[20], [24], или [28].

Главная программа

```
program lin_sys (input,output);
{     Программа для верифицированного решения системы        }
{     линейных уравнений.                                     }
{     Должны быть введены матрица A и вектор b.               }
{     Программа возвращает или верифицированое решение        }
{     или соответствующее сообщение об ошибке                 }
use                         { lin_solv : Решатель систем линейных уравнений }
    lin_solv, mv_ari, mvi_ari; { mv_ari   : матрично/векторная арифметика    }
                            { mvi_ari  : матрично/векторная интервальная    }
                            {            арифметика                         }

var
    n : integer;
{-----------------------------------------------------------------}
procedure main (n : integer);
{     Матрица A и вектор b размещаются динамически при вызове  }
```

```
{    этой подпрограммы. Вводятся матрица A и вектор b      }
{    и вызывается linear_system_solver                      }
var
      ok  : boolean;
      b   : rvector[1..n];
      x   : ivector[1..n];
      A   : rmatrix[1..n,1..n];

begin

      writeln('Введите матрицу A:');
      read(A);

      writeln('Введите вектор b:');
      read(b);

      linear_system_solver(A,b,x,ok);

      if ok then

        begin
          writeln('Данная матрица A не вырождена и решение ');
          writeln('системы линейных уравнений содержится в:');
          write(x);

        end

      else

        writeln('Решение не найдено!');

      end;        { главная процедура}

      {-----------------------------------------}
      begin

        write('Введите размерность системы: ');
        read(n);
        main(n);

      end. { программа lin_sys}
```

модуль LIN_SOLV

```
module lin_solv;
```

```
{ Верифицированное решение системы линейных уравнений Ax = b. }
use                          { i_ari    : интервальная арифметика            }
     i_ari, mv_ari, mvi_ari; { mv_ari   : матрично/векторная арифметика      }
                             { mvi_ari  : матрично/векторная интервальная    }
                             {            арифметика                         }
priority
     inflated = *;          { приоритет уровня 2 }
{-----------------------------------------------------------}
operator inflated (a : ivector; eps : real) infl:ivector[1..ubound(a)];

{ Вычисление так называемого эпсилон–расширения
интервального вектора. }

var
  i    : integer;
  x    : interval;

begin
     for i:= 1 to ubound(a) do
     begin
          x:= a[i];
          if (diam(x) <> 0) then
               a[i] := (1+eps)*x − eps*x
          else
               a[i] := intval( pred (inf(x)), succ (sup(x)) );
     end; {for}

infl := a;
end;          { оператор inflated}
{-----------------------------------------------------}

function approximate_inverse (A: rmatrix): rmatrix[1..ubound(A),1..ubound(A)];
     {          Приблизительное вычисление матрицы обратной          }
     {          к (n,n)-матрице A методом Гаусса.                      }
var
     i, j, k, n    : integer;
     factor        : real;
     R, Inv, E     : rmatrix[1..ubound(A),1..ubound(A)];
begin
     n := ubound(A);          { размерность A }
```

```
    E := id(E);          { единичная матрица}
    R := A;

    {        Шаг метода Гаусса с единичным вектором справа         }
    {        Деление на R[i,i]=0 указывает на                      }
    {        возможную вырожденность матрицы A.                    }
    for i:= 1 to n do
        for j:= (i+1) to n do
        begin
            factor := R[j,i]/R[i,i];
            for k:= i to n do R[j,k] := #*(R[j,k] - factor*R[i,k]);
            E[j] := E[j] - factor*E[i];
        end;        {for j:=...}

    { Вычисление строк обратной матрицы обратной подстановкой. }
    for i:= n downto 1 do
        Inv[i] := #*(E[i] - for k:= (i+1) to n sum(R[i,k]*Inv[k]))/R[i,i];
    approximate_inverse := Inv;
end;        { функция approximate_inverse}
{-------------------------------------------------------------}

global procedure linear_system_solver (A : rmatrix; b : rvector;
                              var x : ivector; var ok : boolean);
{   Вычисление верифицированного вектора, включающего }
{   решение системы линейных уравнений.               }
{   Если включение не достигается после определенного }
{   числа итераций, алгоритм останавливается          }
{   и параметр ok устанавливается в false.            }
const

    epsilon     = 0.25;  { Константа для эпсилон-расширения }
    max_steps   = 10;    { Максимальное число итераций      }

var
    i           : integer;
    y,z         : ivector[1..ubound(A)];
    R           : rmatrix[1..ubound(A),1..ubound(A)];
    C           : imatrix[1..ubound(A),1..ubound(A)];

begin
    R := approximate_inverse(A);
```

```
    { R*b – решение системы линейных уравнений и }
    { z – включение этого вектора. Однако, }
    { оно обычно не содержит истинного решения. }

    z := ##(R*b);

    { Включение I − R*A вычисляется с максимальной аккуратностью. }
    { Вызов функции id(A) генерирует единичную матрицу (n,n).   }
    C := ##(id(A) − R*A);

    x := z;        i := 0;
    repeat
      i := i + 1;
      y := x inflated epsilon;   { Для получения истинного включения   }
                                 { интервала вектор x слегка раздувается.}

      x := z + C*y;              { Вычисляется новая итерация.           }
      ok := x in y;              { Содержится ли x в y?                  }
    until ok or (i = max_steps);
end;   { процедура linear_system_solver }
{-------------------------------------------------------------}
end.   { модуль lin_solv}
```

Приложение

Обзор вещественных и комплексных #-выражений

Синтаксис: #-Символ (Точное выражение)

#-Символ	Тип Результата	Слагаемые разрешенные в точном выражении
#	*dotprecision*	● переменные, константы и специальные функции типа *integer*, *real* или *dotprecision* ● произведения типа *integer* или *real* ● скалярные произведения типа *real*
#* #< #>	*real*	● переменные, константы и специальные функции типа *integer*, *real* или *dotprecision* ● произведения типа *integer* или *real* ● скалярные произведения типа *real*
	complex	● переменные, константы и специальные функции типа *integer*, *real* или *dotprecision* ● произведения типа *integer* или *real* или *complex* ● скалярные произведения типа *real* или *complex*
	rvector	● переменные и специальные функции типа *rvector* ● произведения типа *rvector* (например, *rmatrix* * *rvector*, *real* * *rvector* и т.д.)
	cvector	● переменные и специальные функции типа *rvector* или *cvector* ● произведения типа *rvector* или *cvector* (например, *cmatrix***rvector*, *real***cvector* и т.д.)
	rmatrix	● переменные и специальные функции типа *rmatrix* ● произведения типа *rmatrix*
	cmatrix	● переменные и специальные функции типа *rmatrix* или *cmatrix* ● произведения типа *rmatrix* или *cmatrix*

Обзор вещественных и комплексно–интервальных #-выражений

Синтаксис: ## (Точное выражение)

#-Символ	Тип Результата	Слагаемые разрешенные в точном выражении
##	*interval*	● переменные, константы и специальные функции типа *integer*, *real*, *interval* или *dotprecision* ● произведения типа *integer*, *real* или *interval* ● скалярные произведения типа *real* или *interval*
	cinterval	● переменные, константы и специальные функции типа *integer*, *real*, *complex*, *interval*, *cinterval* или *dotprecision* ● произведения типа *integer*, *real*, *complex*, *interval* или *cinterval* ● скалярные произведения типа *real*, *complex*, *interval* или *cinterval*
	ivector	● переменные и специальные функции типа *rvector* или *ivector* ● произведения типа *rvector* или *ivector*
	civector	● переменные и специальные функции типа *rvector*, *cvector*, *ivector* или *civector* ● произведения типа *rvector* *cvector*, *ivector* или *civector*
	imatrix	● переменные и специальные функции типа *rmatrix* или *imatrix* ● произведения типа *rmatrix* или *imatrix*
	cimatrix	● переменные и специальные функции типа *rmatrix*, *cmatrix*, *imatrix* или *cimatrix* ● произведения типа *rmatrix*, *cmatrix*, *imatrix* или *cimatrix*

References

1. Allendörfer, U., Shiriaev, D., *PASCAL-XSC to C. A portable PASCAL-XSC compiler.* IMACS Annals on Computing and Applied Mathematics. Proceedings of SCAN-90, Albena (Sep. 24-28, 1990), to appear 1991
2. Allendörfer, U., Shiriaev, D., *A portable PASCAL-XSC Development System.* Proceedings of IMACS 13th World Congress on Computation and Applied Mathematics, Dublin (Jul. 22-26, 1991), p. 111-112.
3. American National Standards Institute / Institute of Electrical and Electronic Engineers: *A Standard for Binary Floating-Point Arithmetic.* ANSI/IEEE Std. 754-1985, New York, 1985.
4. Bleher, J. H., Rump, S. M., Kulisch, U., Metzger, M., Ullrich, Ch., and Walter, W.: *FORTRAN-SC: A Study of a FORTRAN Extension for Engineering/Scientific Computation with Access to ACRITH.* Computing **39**, 93 - 110, 1987.
5. Bohlender, G., Grüner, K., Kaucher, E., Klatte, R., Krämer, W., Kulisch, U., Rump, S., Ullrich, Ch., Wolff von Gudenberg, J., and Miranker, W.: *PASCAL-SC: A PASCAL for Contemporary Scientific Computation.* Research Report RC 9009, IBM Thomas J. Watson Research Center, Yorktown Heights, New York, 1981.
6. Bohlender, G., Grüner, K., Kaucher, E., Klatte, R., Kulisch, U., Neaga, M., Ullrich, Ch., and Wolff von Gudenberg, J.: *PASCAL-SC Language Definition.* Internal Report of the Institute for Applied Mathematics, University of Karlsruhe, 1985.
7. Bohlender, G., Rall, L., Ullrich, Ch., and Wolff von Gudenberg, J.: *PASCAL-SC: A Computer Language for Scientific Computation.* Academic Press, New York, 1987.
8. Bohlender, G., Rall, L., Ullrich, Ch. und Wolff von Gudenberg, J.: *PASCAL-SC – Wirkungsvoll programmieren, kontrolliert rechnen.* Bibliographisches Institut, Mannheim, 1986.
9. Buchholz, W.: *The IBM System/370 Vector Architecture.* IBM Systems Journal 25/1, 1986.
10. Däßler, K. und Sommer, M.: *PASCAL, Einführung in die Sprache.* Norm Entwurf DIN 66256, Erläuterungen. Springer, Berlin, 1983.
11. Hammer, R.: *How Reliable is the Arithmetic of Vector Computers?* In: [29], 1990.
12. IBM *High-Accuracy Arithmetic Subroutine Library* (ACRITH). General Information Manual, GC 33-6163-02, 3rd Edition, 1986.
13. IBM *High-Accuracy Arithmetic Subroutine Library* (ACRITH). Program Description and User's Guide, SC 33-6164-02, 3rd Edition, 1986.
14. Kaucher, E., Kulisch, U., and Ullrich, Ch. (Eds.): *Computer Arithmetic – Scientific Computation and Programming Languages.* Teubner, Stuttgart, 1987.
15. Kirchner, R. and Kulisch, U.: *Accurate Arithmetic for Vector Processors.*

Journal of Parallel and Distributed Computing **5**, 250-270, 1988.

16. Klatte, R., Kulisch, U., Neaga, M., Ratz, D. und Ullrich, Ch.: *PASCAL-XSC Sprachbeschreibung mit Beispielen.* Springer, Heidelberg, 1991.
17. Klatte, R., Kulisch, U., Neaga, M., Ratz, D. und Ullrich, Ch.: *PASCAL-XSC Language Reference with Examples.* To be published by Springer, Heidelberg, 1991/92.
18. Kulisch, U. (Ed.): *PASCAL-SC: A PASCAL Extension for Scientific Computation*, Information Manual and Floppy Disks, Version ATARI ST. Teubner, Stuttgart, 1987.
19. Kulisch, U. (Ed.): *PASCAL-SC: A PASCAL Extension for Scientific Computation*, Information Manual and Floppy Disks, Version IBM PC/AT (DOS). Teubner, Stuttgart, 1987.
20. Kulisch, U. (Hrsg.): *Wissenschaftliches Rechnen mit Ergebnisverifikation – Eine Einführung.* Akademie Verlag, Ost-Berlin, Vieweg, Wiesbaden, 1989.
21. Kulisch, U. and Miranker, W. L.: *Computer Arithmetic in Theory and Practice.* Academic Press, New York, 1981.
22. Kulisch, U. and Miranker, W. L.: *The Arithmetic of the Digital Computer: A New Approach.* SIAM Review, Vol. 28, No. 1, 1986.
23. Kulisch, U. and Miranker, W. L. (Eds.): *A New Approach to Scientific Computation.* Academic Press, New York, 1983.
24. Kulisch, U. and Stetter, H. J. (Eds.): *Scientific Computation with Automatic Result Verification.* Computing Suppl. **6**, Springer, Wien, 1988.
25. Neaga, M.: *Erweiterungen von Programmiersprachen für wissenschaftliches Rechnen und Erörterung einer Implementierung.* Dissertation, Universität Kaiserslautern, 1984.
26. Neaga, M.: *PASCAL-SC – Eine PASCAL-Erweiterung für wissenschaftliches Rechnen.* In: [20], 1989.
27. Ratz, D.: *The Effects of the Arithmetic of Vector Computers on Basic Numerical Methods.* In: [29], 1990.
28. Rump, S. M.: *Solving Algebraic Problems with High Accuracy.* In: [23], 1983.
29. Ullrich, Ch. (Ed.): *Contributions to Computer Arithmetic and Self-Validating Numerical Methods.* J. C. Baltzer AG, Scientific Publishing Co., IMACS, 1990.
30. Wolff von Gudenberg, J.: *Einbettung allgemeiner Rechnerarithmetik in PASCAL mittels eines Operatorkonzeptes und Implementierung der Standardfunktionen mit optimaler Genauigkeit.* Dissertation, Universität Karlsruhe, 1980.

Institute for Applied Mathematics
University of Karlsruhe
W-7500 Karlsruhe
Germany
e-mail: dima@iamk4508.rz.uni-karlsruhe.de