Пакеты научных вычислений

Лекция 6

Программирование в Maple

Булева алгебра в Maple.

Условный оператор. Оператор цикла.

Итеративные команды.

Обзор операторов программирования.

Простые процедуры

Наседкина А. А.

Булева алгебра в Maple

- Логические константы и выражения
- Логические операторы и их таблицы истинности

Реализация булевой алгебры в Maple

- Двузначная (бинарная, двоичная) логика оперирует с двумя логическими значениями: истина и ложь (1 и 0).
- Трехзначная (троичная) логика оперирует с тремя логическими значениями: «истина», «ложь» и «не определено».
- В Maple реализована трехзначная логика с логическими значениями
  - **true** (истина)
  - **false** (ложь)
  - **FAIL** (истинность не определена)
- Логическим, или булевым, выражением в Maple называется любое выражение, которое имеет логическое значение. Для записи логического выражения в Maple можно использовать:
  - Логические значения **true, false, FAIL**
  - Операторы отношения <, >, <=, >=, =, <>
  - Логические операторы **not** (НЕ), **and** (И), **or** (ИЛИ), **xor** (исключающее ИЛИ), **implies** (импликация)

Вычисление логических выражений

- Для вычисления логических выражений служит функция **evalb.**

```
> evalb(1 > 2);
                              false
> evalb(not false)
                              true
```

- Логическое выражение является объектом булевского типа данных **boolean.** Этот тип данных включает в себя следующие подтипы: `**=**`, `<>`, `<`, `<=`, `**and**`, `**or**`, `**xor**`, `**implies**`, `**not**`.

```
> whattype(1 = 1);
                              `=`
> type(1 = 1, boolean);
                              true
```

- Вычисление логического выражения происходит автоматически, если оно:
  - содержит логический оператор
  - находится в условии **if** условного оператора
  - находится в условии **while** оператора цикла

Таблицы истинности логических операторов **not** и **and**

Оператор **not** (НЕ)	
a	**not** a
true	**false**
false	**true**
FAIL	**FAIL**

Оператор **and** (И)				
a **and** b		b		
		true	**false**	**FAIL**
a	**true**	**true**	**false**	**FAIL**
	false	**false**	**false**	**false**
	FAIL	**FAIL**	**false**	**FAIL**

Таблицы истинности логических операторов **or** и **xor**

Оператор or (ИЛИ)				
a **or** b		b		
		true	**false**	**FAIL**
a	**true**	**true**	**true**	**true**
	false	**true**	**false**	**FAIL**
	FAIL	**true**	**FAIL**	**FAIL**

Оператор xor (исключающее ИЛИ)				
a **xor** b		b		
		true	**false**	**FAIL**
a	**true**	**false**	**true**	**FAIL**
	false	**true**	**false**	**FAIL**
	FAIL	**FAIL**	**FAIL**	**FAIL**

Операторы программирования

- Условный оператор
- Оператор цикла

Условный оператор: виды

- В Maple можно реализовать базовую структуру программирования «ветвление» с помощью различных видов *условных операторов*.

Виды ветвлений:

- обход (если-то)
  if…**then**…**end if**
- разветвление (если-то-иначе)
  if…**then**…**else**…**end if**
- множественный выбор (если-то-иначе-если…)
  if…**then**…**elif**…**then**…<…> **end if**

- В качестве *условия* должно быть задано логическое выражение, в качестве *выражений* – одно или несколько выражений Maple. Ветка then выполняется, если результат проверки условия – истина (true). В противном случае (false или FAIL) выполняется ветка else (elif).

Условный оператор: простые формы

- **if** *condition* **then** *expressions* **end if;**
- **if** *condition* **then** *expressions1* **else** *expressions2* **end if;**

Команды для вывода сообщений на экран:

- **print(x)** –печатает на экран значение переменной **x**
- **print("xxx")** –печатает на экран строку **"xxx"**
- **printf("bbb %a cc",x)** –печатает на экран сообщение, где в строку **"bbb … cc"** вместо **%a** подставляется значение переменной **x**

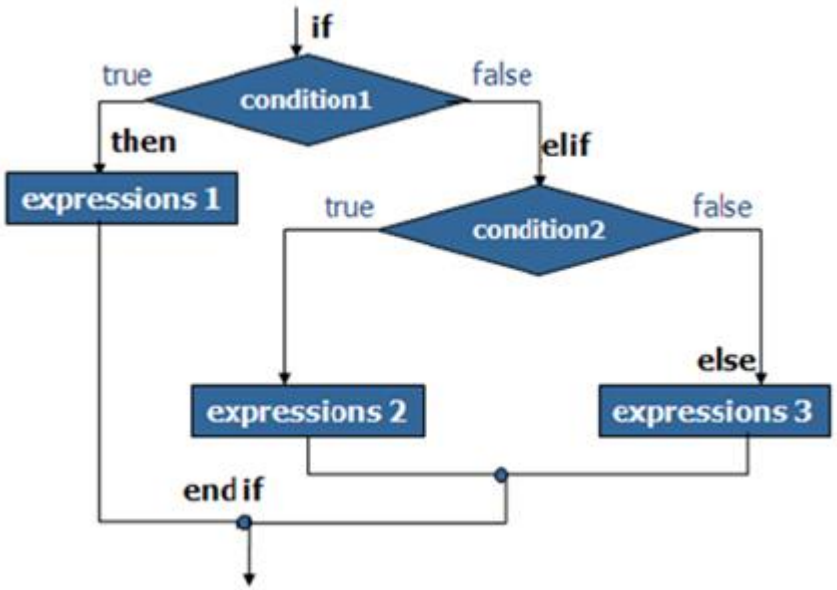
```
> a:=3; b:=5;
> if (a>b) then a else b end if;
                                    5

> x := 3 > 2 : print(x);
                                  2 < 3

> if type(x, boolean)
      then print("x – логическое выражение");
      printf("%a - булево выражение", x) end if;
                          "x - логическое выражение"
2 < 3 - булево выражение
```

Условный оператор: полный синтаксис

- **if** *condition1* **then** *expressions1* **elif** *condition2* **then** *expressions2* **elif** *condition3* **then** *expressions3 …* **else** *expressionsN* **end if;**




```
> x := 7 :
> if x < 2 then x := x^2; print(x) elif x = 2 then x
      := x^3; print(x) else x := x^4; print(x) end if
```

$x := 2401$

2401

Оператор цикла: виды

- В Maple можно реализовать базовую структуру программирования «повторение» с помощью различных видов *операторов цикла* с предусловием. Цикл с постусловием в Maple не реализован

Виды операторов цикла в Maple

- цикл-пока (while)
  while…do…end do
- циклы с параметром (for/from, for/in)
  for…from…do…end do
  for…in…do…end do
- смешанные циклы (for/from while, for/in while)
  for…from…while…do…end do
  for…in…while…do…end do

- В качестве *условия* **while** должно быть задано логическое выражение, в качестве *выражений* – одно или несколько выражений Maple. *Выражения*, расположенные между ключевыми словами **do** и **end do** составляют тело цикла.

Оператор цикла while

- **while** *condition* **do** *expressions* **end do;**

Пока результат проверки условия – истина (true), выполняются выражения Maple, составляющие тело цикла. Как только результат проверки условия становится ложным (false) или FAIL, то происходит выход из структуры цикла.

Пример. Четные числа от 8 до 2 в убывающем порядке

```
> j := 8:
> while j ≥ 2 do if irem(j, 2) = 0 then print(j) end if; j := j − 1 end do;
```

8 — 1-я итерация

$j := 7$

$j := 6$ — 2-я итерация

6 — 3-я итерация

$j := 5$

$j := 4$ — 4-я итерация

4 — 5-я итерация

$j := 3$

$j := 2$ — 6-итерация

2 — 7-я итерация

$j := 1$

Оператор цикла while: примеры

Пример. Алгоритм перевода десятичного числа в двоичное, остатки записываются в последовательность.

```
> x := 19; s := NULL;
```

$x := 19$

$s :=$

```
> while x>0 do printf("Остаток от деления %a на 2 равен %a", x,
irem(x,2)); s:=s,irem(x,2); printf("Частное от деления %a на 2 равно
%a", x, iquo(x,2)); x:=iquo(x,2); end do;
Остаток от деления 19 на 2 равен 1
```

$s := 1$

```
Частное от деления 19 на 2 равно 9
```

$x := 9$

```
Остаток от деления 9 на 2 равен 1
```

$s := 1, 1$

```
Частное от деления 9 на 2 равно 4
```

$x := 4$

```
Остаток от деления 4 на 2 равен 0
```

$s := 1, 1, 0$

```
Частное от деления 4 на 2 равно 2
```

$x := 2$

```
Остаток от деления 2 на 2 равен 0
```

$s := 1, 1, 0, 0$

```
Частное от деления 2 на 2 равно 1
```

$x := 1$

```
Остаток от деления 1 на 2 равен 1
```

$s := 1, 1, 0, 0, 1$

```
Частное от деления 1 на 2 равно 0
```

$x := 0$

```
> convert(19, base, 2);
```

$[1, 1, 0, 0, 1]$

```
> convert(19,'binary');
```

10011

Оператор цикла for/from

- **for** *counter* **from** *initial* **by** *increment* **to** *final* **do** *expressions* **end do;**

Сокращенная форма: **for** *counter* **to** *final* **do** *expressions* **end do;**

Значения по умолчанию:

initial=1, increment=1, final=∞

```
> for j from 4 to 8 do print(j) end do
                              4
                              5
                              6
                              7
                              8
```

Пример. Четные числа от 8 до 2 в убывающем порядке

```
> for j from 8 to 2 by -1 do if irem(j, 2) = 0 then print(j) end if end do
                              8
                              6
                              4
                              2
> j;
                              1
```

Оператор цикла for/in

- **for** *variable* **in** *expression* **do** *expressions* **end do;**

Параметр цикла последовательно принимает значения элементов выражения Maple (например, слагаемых в сумме, сомножителей в произведении, символов в строке, элементов списка, множества и т. д.)

```
> f := x^2 + 3*x + 1/x :
> for s in f do s; end do;
```

$$x^2$$

$$3\,x$$

$$\frac{1}{x}$$

```
> s := "цикл" :
> for i in s do print(i) end do
```

"ц"

"и"

"к"

"л"

Оператор цикла for/in: примеры

Пример. Вычисление суммы полных квадратов от 1 до 100.

```
> SQR := {i^2 $ i = 1..10};
```

$$SQR := \{1, 4, 9, 16, 25, 36, 49, 64, 81, 100\}$$

```
> s := 0:
> for i in SQR do s := s + i end do
```

$$s := 1$$
$$s := 5$$
$$s := 14$$
$$s := 30$$
$$s := 55$$
$$s := 91$$
$$s := 140$$
$$s := 204$$
$$s := 285$$
$$s := 385$$

```
> convert(SQR, `+`)
```

$$385$$

Полный синтаксис операторов цикла for/from…while, for/in…while (смешанные циклы)

- **for** *counter* **from** *initial* **by** *increment* **to** *final* **while** *condition* **do** *expressions* **end do;**
- **for** *variable* **in** *expression* **while** *condition* **do** *expressions* **end do;**

- Если в полном синтаксисе оператора цикла **for/from** задано конечное значение параметра цикла (**to** *final*) и присутствует проверка условия (**while** *condition*), то сначала выполняется сравнение текущего значения параметра цикла с конечным final, затем – проверка условия while. Если условие while станет ложным, произойдет выход из цикла.

- Если в полном синтаксисе оператора цикла **for/in** присутствует проверка условия (**while** *condition*), то для каждого значения параметра сначала выполняется проверка условия, а после этого выполняется тело цикла. Параметр последовательно принимает значение элементов выражения Maple до тех пор, пока не встретится элемент, для которого условие станет ложным. При этом выход из структуры цикла может произойти раньше, чем будут исчерпаны все элементы выражения Maple (см. пример далее).

Смешанные циклы: примеры

Пример 1.

```
> for x from 1 by 2 to 7 while x < 6 do x end do
                                  1
                                  3
                                  5
> x
                                  7
```

Пример 2. Вывод на экран тех элементов списка, которые являются числами.

```
> s := [2, a, b, 3.5, c] :
> for x in s while type(x, numeric) do x end do
                                  2
> x
                                  a
> s := [2, 3, 4, 3.5, c] :
> for x in s while type(x, numeric) do x : end do
      #двоеточие здесь не подавляет вывод на экран
                                  2
                                  3
                                  4
                                 3.5
> x
                                  c
```

Пример 3. Вывод на экран последовательности первых десяти простых чисел.

```
> COUNT:=0: PRIMES:=NULL:  (сначала задается пустая последовательность)
> for n from 1 while COUNT<10 do
if isprime(n) then COUNT:=COUNT+1; PRIMES:=PRIMES,n end if
end do;
> PRIMES;
                          2, 3, 5, 7, 11, 13, 17, 19, 23, 29
> COUNT;
```

Итеративные команды Maple

- seq
- add и mul
- select, remove, selectremove
- map
- zip

Итеративные команды: seq

- Оператор формирования последовательности

  expr $ name = initial .. final;
- Создание последовательности

  seq(expression, name = initial .. final,step); #аналог цикла **for/from**

  seq(expression, name in expression); #аналог цикла **for/in**

```
> $ 2..5;
```

2, 3, 4, 5

```
> a[i] $ i = 1..3;
```

a_1, a_2, a_3

```
> seq( i^2, i=1..5 );
```

1, 4, 9, 16, 25

```
> seq( i, i="a".."f" );
```

"a", "b", "c", "d", "e", "f"

```
> seq( x[i], i=1..5 );
```

x_1, x_2, x_3, x_4, x_5

```
> seq(u, u in [Pi/4, Pi^2/2, 1/Pi]);
```

$$\frac{\pi}{4}, \frac{\pi^2}{2}, \frac{1}{\pi}$$

Итеративные команды: add, mul

- Вычисление суммы

  add(expression, name = initial .. final); #аналог цикла **for/from**

  add(expression, name in expression); #аналог цикла **for/in**

- Вычисление произведения

  mul(expression, name = initial .. final); #аналог цикла **for/from**

  mul(expression, name in expression); #аналог цикла **for/in**

```
> add( i^2, i=1..5 );
```

$$55$$

```
> add(u, u in [Pi/4, Pi/2, Pi/6]);
```

$$\frac{11}{12}\,\pi$$

```
> mul(i, i=1..5 );
```

$$120$$

```
> mul(u, u in [Pi/4, Pi^2/2, 1/Pi]);
```

$$\frac{1}{8}\,\pi^2$$

Итеративные команды: select, remove

Команды извлечения и удаления (аналог цикла **for/in)**

- **select(proc,expression) -** извлекает те операнды из выражения *expression*, которые удовлетворяют булевой функции (процедуре) *proc.*
- **remove(proc,expression) -** извлекает те операнды из выражения *expression*, которые не удовлетворяют булевой функции (процедуре) *proc.*
- **selectremove(proc,expression) -** сначала извлекает те операнды из выражения *expression*, которые удовлетворяют булевой функции (процедуре) *proc*, а затем те, которые ей не удовлетворяют.
- Каждая команда возвращает объект того же типа, что и исходное выражение

Извлечем простые числа из списка

```
> integers := [$10..20];
                integers := [10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]

> select(isprime, integers);
                [11, 13, 17, 19]

> remove(isprime, integers);
                [10, 12, 14, 15, 16, 18, 20]

> selectremove(isprime, integers);
                [11, 13, 17, 19], [10, 12, 14, 15, 16, 18, 20]
```

```
> f := x + y + 2
                f := x + y + 2

> select(type, f, symbol)
                x + y
```

Итеративные команды: map

- Применение команды или процедуры к каждому члену выражения **map(proc,expression);** #аналог цикла **for/in**

```
> map(f, {a,b,c});
```

$$\{f(a), f(b), f(c)\}$$

```
> map(f, x + y*z);
```

$$f(x) + f(y\,z)$$

```
> map(f, y*z);
```

$$f(y)\,f(z)$$

```
> map(sqrt,a+b);
```

$$\sqrt{a} + \sqrt{b}$$

```
> f := x→sin(x + 3) :
> map(f, y·z);
```

$$\sin(y + 3)\,\sin(z + 3)$$

Разложим на множители все составные числа из списка `integers`.

```
> integers := [$10..20]: comp:=remove(isprime,integers);
map(ifactor,comp);
```

$$comp := [10, 12, 14, 15, 16, 18, 20]$$

$$[(2)\,(5), (2)^2\,(3), (2)\,(7), (3)\,(5), (2)^4, (2)\,(3)^2, (2)^2\,(5)]$$

Итеративные команды Maple: zip

- Применение команды или процедуры покомпонентно к паре элементов двух списков или векторов **zip.** Команда объединяет два списка или вектора в один.
- **zip(proc,a,b)**

  zip(proc,a,b,fill)
- Длина результирующего списка или вектора равна min(m,n), где m и n - длины исходных списков.
- Значение параметра *fill* используется для заполнения элементов результирующего списка в случае списков разной длины, при этом результирующий список будет иметь длину max(m,n)

```
> zip(`+`, [1, 2, 3], [4, 5, 6])
```

$$[5, 7, 9]$$

```
> zip(`+`, [1, 2, 3, 4, 5], [4, 5, 6], 10)
```

$$[5, 7, 9, 14, 15]$$

```
> zip(f, [a, b, c], [d, e])
```

$$[f(a, d), f(b, e)]$$

```
> zip((x, y) -> x + 100 y, [1, 2], [4, 5, 6])
```

$$[401, 502]$$

Обзор операторов программирования

Все операторы программирования в Maple 11.0

List of Maple statements

Description

• See ?topic for any of the following topics:

!	#	assignment	break	by
catch	description	do	done	elif
else	empty	end do	end if	error
export	finally	for	from	function
if	in	local	module	next
option	proc	quit	read	restart
return	save	separator	stop	then
to	try	use	uses	while

Пример

• **break** – оператор прерывания цикла

$L := [1, 2, \text{"abc"}, \text{"a"}, 7.0, \infty]$: **for** x **in** L **do if** *type*$(x, \text{'string'})$ **then**

print(x); **break end if end do**

"abc"

Пользовательские процедуры в Maple

- Функциональные операторы
- Простые процедуры

Функциональный оператор

- **Процедура** – это подпрограмма, состоящая из команд и выражений Maple. По сути, процедура также является командой Maple, но не встроенной, а созданной пользователем. Упрощенным понятием процедуры является функциональный оператор.

Общий синтаксис:

- **f:=var -> result**
- **f:=(var1,var2,..)->result**

С помощью функционального оператора можно задавать функции одной и многих переменных, а также вектор-функции:

- *f:=x -> x^2* – функция одной переменной $R \to R$
- *f:=(x,y) -> x^2 + y^2* – функция двух переменных $R^2 \to R$
- *f:= x -> (2*x, 3*x^4)* – вектор-функция одной переменной $R \to R^2$
- *f:= (x,y,z) -> (x*y, y*z)* – вектор-функция трех переменных $R^3 \to R^2$

Функциональный оператор: примеры

Создание функционального оператора из выражения и входящих в него переменных

- **unapply(expression, x,y…,); unapply(expression, list of variables);**

```
> g := (x,y) -> sin(x)*cos(y) + x*y;
```

$$g := (x, y) \rightarrow \sin(x)\cos(y) + x\,y$$

```
> g(Pi/2,Pi);
```

$$-1 + \frac{1}{2}\pi^2$$

С помощью функционального оператора можно создавать небольшие процедуры:

```
> p:=x-> if x<0 then -x; else sqrt(x); end if: p(-2); p(2);
```

$$2$$

$$\sqrt{2}$$

```
> p := x^2 + sin(x) + 1; f := unapply(p,x); f(Pi/6);
```

$$p := x^2 + \sin(x) + 1$$

$$f := x \rightarrow x^2 + \sin(x) + 1$$

$$\frac{1}{36}\pi^2 + \frac{3}{2}$$

```
> q:=a-b;g:=unapply(q,a,b);
```

$$q := a - b$$

$$g := (a, b) \rightarrow a - b$$

```
> q :=x^2+y^3+1; g:=unapply(q,[x,y]); g(2,3);
```

$$q := x^2 + y^3 + 1$$

$$g := (x, y) \rightarrow x^2 + y^3 + 1$$

$$32$$

Пользовательские процедуры в Maple

• Процедура – это пользовательская команда Maple.
• Процедура может не иметь аргументов, иметь один аргумент или несколько аргументов (тогда они перечисляются через запятую).
• Все описание процедуры должно находиться в одной выполнимой группе (execution group). Переход на новую строку осуществляется с помощью **Shift+Enter**

Краткий синтаксис процедуры

- *proc_name*:=**proc (***parameterSequence***)**

  statementSequence;

  end proc;

Слова, выделенные синим могут отсутствовать.

Пример простейшей процедуры без аргументов

```
> ex := proc( )
  √2;
  end proc;
```

ex := **proc( ) sqrt(2) end proc**

Вызов и выполнение процедуры:

```
> ex( );
```

$\sqrt{2}$

- *proc_name* – имя процедуры
- **proc … end proc** – служебные слова, начало и конец процедуры
- *statementSequence* – последовательность выражений, реализующих *тело* процедуры. Возвращается значение последнего выражения (команды) из этой последовательности или значение выражений, указанных после команды **return**.

Пример простой процедуры с параметром

- Процедура возвращает последовательность n первых простых чисел

```
> p:=proc(n)
  local i,COUNT, PRIMES;
  COUNT:=0: PRIMES:=NULL:
  for i from 1 while COUNT<n do
  if isprime(i) then COUNT:=COUNT+1; PRIMES:=PRIMES,i; end if
  end do;
  PRIMES;
  end proc;
```

$p := \mathbf{proc}(n)$
$\quad \mathbf{local}\ i, COUNT, PRIMES;$
$\quad COUNT := 0;$
$\quad PRIMES := NULL;$
$\quad \mathbf{for}\ i\ \mathbf{while}\ COUNT < n\ \mathbf{do}\ \mathbf{if}\ isprime(i)\ \mathbf{then}\ COUNT := COUNT + 1;\ PRIMES := PRIMES, i\ \mathbf{end\ if}\ \mathbf{end\ do};$
$\quad PRIMES$
end proc

$> p(5)$

$$2, 3, 5, 7, 11$$

$> p(10)$

$$2, 3, 5, 7, 11, 13, 17, 19, 23, 29$$