Mathematical Computing

IMT2b2β

Department of Mathematics
University of Ruhuna

A.W.L. Pubudu Thilan

Programming in Maxima

Introduction

- Maxima contains all the programming structures required to build programs of any complexity.
- Maxima's base language is **Lisp**, but Maxima can either be programmed in **Lisp**, or in its own language.

Comments in Maxima

- A comment in Maxima input is any text between /* and */.
- Comments can be nested to arbitrary depth.
- The /* and */ delimiters form matching pairs.
- There must be the same number of /* as there are */.

Eg:

1. **/* i is a variable of interest */ i : 12;**
2. **/* Comments /* can be nested /* to any depth */ */ */ 1 + uvt;**

Branching

About branching

- Branching controls the execution of a program.
- Statements in a program are executed only if a **condition** holds.
- The **condition** determines the choice of branch we want our program to execute.
- ";" or "$" is only used after the complete if-else statement.

Formal syntax

if cond_1 then
    expr_1
else
    expr_0

The value of this expression:

- evaluates to **expr_1** if **cond_1** evaluates to true.
- otherwise the expression evaluates to **expr_0**.

Formal syntax
Example code

```
user:17$
if (user < 18) then
   print ("User is 18 or younger")
else
   print ("User is older than 18")$
```

Output is

User is 18 or younger

The condition

- Must be able to be evaluated to either true or false.
- Uses operators for comparing things.
- In nearly all situations, operators are one or more of **relational operators** or **logical operators**.

Relational operators

Operator	Symbol
Less than	$<$
Less than or equal to	$<=$
Greater than	$>$
Greater than or equal to	$>=$
Equality	$=$
Negation of equality	$\#$

```
if(1 < 2)then
   print("I like Maxima")
else
   print("I like Mathematica")$
```

Output is

I like Maxima

```
if(1 >= 2) then
   print("I like Maxima")
else
   print("I like Mathematica")$
```

Output is

   I like Mathematica

Logical operators

Operator	Symbol
and	and
or	or
not	not

```
if ((1 < 2)and(2 < 3)) then
   print("I like Maxima")
else
   print("I like Mathematica")$
```

Output is

  I like Maxima

```
if ((1 < 2)and(2 > 3)) then
    print("I like Maxima")
else
    print("I like Mathematica");
```

Output is

    I like Mathematica

More complex if-else statements

if cond_1 then
    expr_1
elseif cond_2 then
    expr_2
elseif ...
else expr_0

- Evaluates to **expr_k** if **cond_k** is true and all preceding conditions are false.
- If none of the conditions are true, the expression evaluates to **expr_0**.

More complex if-else statements
Excercise

Write a programme to assign a grade based on the value of a test score: an **A** for a score of 90% or above, a **B** for a score of 80% or above, and so on.

Omiting ending else in if-else

- **if cond_1 then**
  expr_1

  is equivalent to

- **if cond_1 then**
  expr_1
  else
  false

Omiting ending else in if-else

Example code

- **if (1 = (2 + 4 + 6)/(6 + 4 + 2)*(2-1)) then**
  print("You are welcome");

- **if (1 = (2 + 4 + 6)/(6 + 4 + 2) * 2) then**
  print("You are welcome");

Functions and if statements

Define following function in Maxima.

$$f(x) = \begin{cases} 0 & x < 0 \\ x & 0 \leq x < 1 \\ 1 & x > 1 \end{cases}$$

code

```
f(x):= if (x < 0) then
            0
       else if (0 <= x) and (x < 1) then
            x
       else
            1
```

Define following function in Maxima and plot the function in the interval [-10,10].

$$f(x) = \begin{cases} (x+5)^2 & x < -5 \\ \dfrac{x+5}{2} & -5 \leq x < 0 \\ \dfrac{-x+5}{2} & 0 \leq x < 5 \\ (x-5)^2 & x \geq 5 \end{cases}$$

Iteration

About iteration

- The **do** statement is used for performing iteration.
- The **do** statement can be used in Maxima analogous to that used in several other programming language.
- Also it can be used in different ways in Maxima.
- There are three variants of this form that differ only in their terminating conditions

Three form of **do** statement

1. **for** *variable*:*initial_value* **step** *increment* **thru** *limit* **do** *body*
2. **for** *variable*:*initial_value* **step** *increment* **while** *condition* **do** *body*
3. **for** *variable*:*initial_value* **step** *increment* **unless** *condition* **do** *body*

Three form of **do** statement
Cont...

- The reserved words for the loop are **for**, **step**, **thru**, **while**, **unless**, and **do**.
- The *initial_value*, *increment*, *limit*, and *body* can be any expressions.
- If the increment is 1 then **step** 1 may be omitted.
- The **step** may be given after the termination *condition* or *limit* as well.

The execution of the **do** statement proceeds by first assigning the *initial_value* to the *variable*. Then:

1. If the *variable* has exceeded the *limit* of a **thru** specification, or if the *condition* of the **unless** is true, or if the *condition* of the **while** is false then the **do** terminates.
2. The *body* is evaluated.
3. The *increment* is added to the *variable*.

The execution of the **do** statement
Cont...

- The process from (1) to (3) is performed repeatedly until the termination condition is satisfied.
- One may also give several termination conditions in which case the **do** terminates when any of them is satisfied.

The execution of the **do** statement
Cont...

- In general the **thru** test is satisfied when the *variable* is greater than the *limit* if the *increment* was non-negative, or when the *variable* is less than the *limit* if the *increment* was negative.
- The *increment* and *limit* may be non-numeric expressions as long as this inequality can be determined.
- However, unless the *increment* is syntactically negative at the time the **do** statement is input, Maxima assumes it will be positive when the **do** is executed.
- If it is not positive, then the **do** may not terminate properly.

Example code

Find the sum of the integers from 1 to 10.

<u>code</u>

```
sum:0; /* initialize */
for i: 1 step 1 thru 10 do
sum : sum + i ;/* accumulate */
print(sum); /* output */
```

55

Excercise

(i) Write a program to find summation of numbers from 1 to 100.

(ii) Write a program to find summation of even numbers from 1 to 2000.

(iii) Write a program to find summation of odd numbers from 1 to 2000.

(iv) Write a program to plot $\sin(nx)$ for $n = 1, 2, 3, 4, 5$ in the range of $-\pi \leq x \leq \pi$.

Different implementation

Same results can be obtained from different implementation of **do**.

1. **for i: 1 step 1 thru 10 do**
   print(i);

2. **for i: 1 step 1 while (i $<=$ 10) do**
   print(i);

3. **for i: 1 step 1 unless (i $>$ 10) do**
   print(i);

Execute block of code

- In Maxima the **block** construct simply groups together a list of commands and treats them as a single statement.

  for *variable*:*initial_value* **step** *increment* **thru** *limit*
  block(
  statement 1,
  statement 2,
  statement 3,
  .,
  .,
  .,
  statement n
  );

Execute block of code

Example code

The segment below generates and displays random numbers between 0.0 and 1.0 as long as the values are less than 0.7. The segment also counts how many of the random values are in that range.

```
r:random(1.0);
count:0;

for i: 1 step 1 while(r < 0.7) do
block(
count:count+1,
r:random(1.0),
print(r)
);
print(count);
```

(i) Write a programe to find 5!.

(ii) Implement following algorithm using Maxima.
x=1 , y=0 , z=2
while $(0 \leq (x - y) < 5)$
y=zx
x=y+z
z=z+1
end while

(a) Use Maxima to define a function to get the volume of a sphere when its radius is given.

(b) Use the above defined function to get the volume of the sphere when its radius is given as 6.

Parameter and argument

- In Maxima, we can define a function to get the volume of a sphere as follows:

$$\mathbf{volume}(\mathbf{r}) := \frac{\mathbf{4}}{\mathbf{3}} * \%\mathbf{pi} * \mathbf{r}^\wedge\mathbf{3};$$

- The **argument** is the input passed to a function, whereas the **parameter** is the variable inside the implementation of the function.

- Therefore, in our example, **r** is the parameter, while if this is called as **volume**(**6**), then **6** is an argument.

Parameter and argument
Example

- The following statement defines a function that is named **tax** and has one parameter named **price**.

$$\textbf{tax}(\textbf{price}) := \textbf{price} * \left(\frac{\textbf{10}}{\textbf{100}}\right);$$

- After the function has been defined, it can be invoked as follows by passing an argument.

$$\textbf{tax}(\textbf{1000});$$

- When this happens, **1000** will be assigned to **price**, and the function begins calculating its result.

Motivating example

(a) Use Maxima to define a function to get both the volume and the surface area of a sphere when its radius is given.

(b) Use the above defined function to get the volume and the surface area of a sphere when its radius is given as 6.

$$
\begin{aligned}
&sphere(r) := block([area],\\
&area : bfloat(4 * \%pi * r * r),\\
&print("Area" = area),\\
&volume : bfloat((4/3) * \%pi * r^3),\\
&print("Volume" = volume)\\
&)\$\\
&sphere(6)\$
\end{aligned}
$$

Identify global and local variables in the following program.

Global and local variables

- The value of a global variable can be accessed in anywhere.
- But the value of a local variable can only be accessed in the block where it is declared.
- When execution of the block starts the local variable is available, and when the block ends the local variable 'dies'.

Blocks and local variables

- The first statement in your block should normally be $[\mathbf{v_1}, \mathbf{v_2}, ..., \mathbf{v_n}]$, where $\mathbf{v_1}, \mathbf{v_2}$, etc., are variables that you wish to be local.
- If you do not want any local variables, then omit the local statement.
- When Maxima enters **block**(), it saves the current values of the variables in the $[\mathbf{v\_1}, \mathbf{v\_2}, ..., \mathbf{v\_n}]$ statement.
- When Maxima exits the **block**() in which the variable was declared as local, its current properties are removed and the saved values/properties are restored.

Example 1

Consider the initial value problem

$$y' + y = x, \qquad y(0) = 1.$$

(i) Find exact solution of the above initial value problem.

(ii) Use Euler's method with step size 0.2 to get numerically approximated solutions in the interval $0 \le x \le 1$.

Example 1

Euler Method Algorithm

define $f(x, y)$
input $x0$ and $y0$
input $xend$
input the number of steps, n
calculate step size h
set $x = x0$
set $y = y0$
for i from 1 to n do
$y : f(x, y) * h + y$,
$x : x + h$,
print(x, y)
end

Example 2

Use the Runge-Kutta method of order four to obtain approximations to the solution of the intial-value problem

$$y' = \frac{(1+y)}{x}, \quad y(1) = 1,$$

in the range $1 \le t \le 10$ with $h = 0.1$.

Example 2
Code

$$runge(f, x0, y0, x1, n) := block([h, x, y, vx, vy, k1, k2, k3, k4],$$

$$h : bfloat((x1 - x0)/(n - 1)),$$

$$x : x0,$$

$$y : y0,$$

$$vx : makelist(0, n + 1),$$

$$vy : makelist(0, n + 1),$$

$$vx[1] : x0,$$

$$vy[1] : y0,$$

Example 2
Code ⇒ Cont...

$$\text{for i from 1 thru } n \text{ do(}$$
$$k1 : bfloat(h * f(x, y)),$$
$$k2 : bfloat(h * f(x + h/2, y + k1/2)),$$
$$k3 : bfloat(h * f(x + h/2, y + k2/2)),$$
$$k4 : bfloat(h * f(x + h, y + k3)),$$
$$vy[i + 1] : y : y + (k1 + 2 * k2 + 2 * k3 + k4)/6,$$
$$vx[i + 1] : x : x + h$$
$$),$$
$$[vx, vy]$$
$$)\$$$
$$[x, y] : runge(lambda([x, y], (1 + y)/x), 0, 1, 10, 101)\$$$

The End!