Maxima — Numbers, Variables and Expressions

April 20, 2007

Contents

7 Maxima — Numbers, Variables and Expressions 2

7.1 Numbers ................................................. 2

7.1.1 Integers ........................................... 2

7.1.2 Primes and Factorization ...................... 3

7.1.3 Exact Arithmetic ................................. 3

7.1.4 Floating Point Numbers ......................... 5

7.1.5 Complex Numbers ............................... 6

7.2 Variables ............................................. 9

7.2.1 Uses of Variables ............................... 9

7.2.2 Variables and Values ......................... 10

7.2.3 An Important Example ....................... 11

7.2.4 Substitution ................................... 12

7.3 Expressions ......................................... 13

7.3.1 The Structure of Expressions ............... 13

7.3.2 Parts of an Expression ....................... 14

7.4 Lists ................................................. 16
With these Lectures on Maxima it is probably best to have a quick glance through the Lecture to get an idea of what it is about, and then start Maxima and type in the commands as you go. Maxima is an interactive system, and it is important that you become familiar with how it works.

For installing and working with Maxima, see Practical 3 and Lecture 6.

7.1 Numbers

7.1.1 Integers

Maxima works with arbitrarily large integers:\footnote{This is not quite true. There are practical limits, for example about 10,000 digits in the version of Maxima I use.}

(\%i2) \(a : 20!;\)

(\%o2) \(2432902008176640000\)

(\%i3) \(b : 12^{34};\)

(\%o3) \(4922235242952026704037113243122008064\)

Quotients and remainders are found with \texttt{quotient} and \texttt{remainder}:

(\%i4) \(q : \text{quotient}(b,a);\)

(\%o4) \(2023195026519395077\)

(\%i5) \(r : \text{remainder}(b,a);\)

(\%o5) \(774112680720728064\)

(\%i6) \(b - (a*q + r);\)

(\%o6) \(0\)

\texttt{gcd} computes the greatest common divisor:

(\%i7) \(g : \text{gcd}(a,b);\)

(\%o7) \(1719926784\)

(\%i8) \(a/g;\)

(\%o8) \(1414538125\)

(\%i9) \(b/g;\)

(\%o9) \(2861886499322070388803895296\)
7.1.2 Primes and Factorization

Two important problems in computational number theory are prime testing and factorization. The relevant Maxima functions are `primep` and `factor`:

```
(%i10) primep(216091);
(%o10) TRUE

(%i11) d : 2^63 - 1;
(%o11) 9223372036854775807

(%i12) primep(d);
(%o12) FALSE

(%i13) factor(d);
(%o13) 2
    7  73  237 92737 649657
```

With regard to prime testing and factorization, note:

1. Factorizations into primes is a very difficult computational problem. Many modern techniques for data encryption, for example the RSA cryptosystem, depend for their security on the difficulty of factoring large numbers or related problems. Therefore don’t expect `factor` to be able to factorize, say, 200 digit numbers.

2. Prime testing, while not as computationally difficult as factorization, is still a very hard problem. However there is a very efficient probabilistic algorithm for testing for primality but which can (at least in theory) occasionally fail.

3. The algorithms used by Maxima for prime testing and factorization are fairly inefficient. For serious work in number theory it is better to use a specialized system such as PARI/GP\(^2\) designed especially for number theoretic computations.

7.1.3 Exact Arithmetic

Where possible, Maxima uses exact arithmetic. Thus division of integers results in a rational number rather than a numerical approximation:

```
(%i14) a;
(%o14) 2432902008176640000
```

\(^2\)URL: pari.math.u-bordeaux.fr
\[
\begin{align*}
\%i15\ b; & \quad \text{(no output)} \\
\%i15\ & 4922235242952026704037113243122008064 \\
\%i16\ c : a/b; & \quad \frac{1414538125}{2861886499322070388803895296}
\end{align*}
\]

\[
\begin{align*}
\%i17\ c + 3/4; & \quad \frac{2146414874491552793017459597}{2861886499322070388803895296}
\end{align*}
\]

The numerator and denominator of a fraction are obtained with \texttt{num} and \texttt{denom}:

\[
\begin{align*}
\%i18\ & \texttt{num(c);} \\
\%i18\ & 1414538125 \\
\%i19\ & \texttt{denom(c);} \\
\%i19\ & 2861886499322070388803895296
\end{align*}
\]

Square roots and similar operators are not replaced by numerical approximations:

\[
\begin{align*}
\%i20\ e : \sqrt{1001}; & \quad \sqrt{1001} \\
\%i20\ & \texttt{e*e;} \\
\%i21\ & 1001 \\
\%i22\ f : \cos(\sqrt{1001}); & \quad \cos(\sqrt{1001})
\end{align*}
\]

The following example is interesting:

\[
\begin{align*}
\%i23\ & \sin(\pi/4); \\
\%o23\ & \frac{\sqrt{2}}{2}
\end{align*}
\]
(%i24) sin(%pi/8);

\%PI
(%o24) SIN(---)
\hfill 8

Note that \( \sin(\pi/8) \) can also be expressed in terms of roots. (Can you explain why?). This is another example of Maxima making a choice of how much simplification to perform.

The common constants \( \pi, e \) and \( i = \sqrt{-1} \) are written \%pi, \%e and \%i. Maxima knows that \( e^{i\pi} = -1 \):

(%i25) \%e^(%i*%pi);

(%o25) - 1

7.1.4 Floating Point Numbers

Numerical values can be obtained with float:

(%i26) f;

(%o26) COS(SQRT(1001))

(%i27) float(%);

(%o27) 0.9753140580386898

(%i28) c;

1414538125
(%o28) ----------------------------
2861886499322070388803895296

(%i29) float(c);

(%o29) 4.94267723522606e-19

By default, floating point numbers use IEEE double precision arithmetic which works with about 16 decimal digits. Floating point numbers are written in scientific notation with \( e \) as the scale factor. Thus

\[ 1.2345 \times 10^{-67} \] is written \( 12.345 \times 10^{-67} \).

Arbitrary precision floating point arithmetic is available. The number of digits is controlled by fpprec and expressions are evaluated with bfloat instead of float.

(%i30) fpprec : 100;

(%o30) 100
7.1.5 Complex Numbers

Maxima can do arithmetic with complex numbers:

(%i35) z1 : 4 + 19*%i;

(%o35) 19 %I + 4
Sometimes Maxima needs coaxing to get things in the form you want:

```
(%i40) rectform(%);
(%o40) 125 %I - 311

(%i41) z1/z2;
(%o41) 19 %I + 4

(%i42) rectform(%);
(%o42) 335 11 %I
     --- - ----- 
     298 298

(%i43) polarform(%);
- %I ATAN(11/335) 
     SQRT(377) %E
     -----------------------------
     SQRT(298)
```

Absolute values and complex conjugates are found using abs and conjugate. Before using conjugate you must load the eigen package:

```
(%i44) abs(%o42);
(%o44) SQRT(377)

(%i44) conjugate
```
(%i45) load(eigen);

Warning - you are redefining the MACSYMA function EIGENVALUES
Warning - you are redefining the MACSYMA function EIGENVECTORS
(%o45) /usr/share/maxima/5.9.1/share/matrix/eigen.mac
(%i46) conjugate(%o42);  

11 %I 335
----- + ---
298 298

Here is another example showing how to get Maxima to put an expression
in the form you want:

(%i47) %o43;

- %I ATAN(11/335)
  SQRT(377) %E
  -----------------------------
  SQRT(298)
(%o47) conjugate(%);

%I ATAN(11/335)
  SQRT(377) %E
  -----------------------------
  SQRT(298)
(%o48) rectform(%);

11 SQRT(377) %I 335 SQRT(377)
---------- + ----------------------
SQRT(298) SQRT(112346) SQRT(298) SQRT(112346)

This looks as though it can be simplified. The radcan command per-
forms the simplification we want:

(%i50) radcan(%);

11 %I + 335
-----------
298
7.2 Variables

7.2.1 Uses of Variables

Variables in Maxima, as in Mathematics itself, are used in a number of different ways. A simple use is when variables have already been assigned values:

(%i2) a : 10;
(%o2) 10
(%i3) b : 6;
(%o3) 6
(%i4) c : a+b;
(%o4) 16

In this situation, we are just using the variables a and b as names for the numbers 10 and 6. This is no different to how variables are used in other programming languages.

What sets computer algebra systems like Maxima apart is that variables can be used as they are in algebra, that is as names for unknowns which might never be assigned as values. Some familiar examples occur in calculus, where variables are often used as place holders, and in algebra itself when variables can be things to solve for.

Here are some examples which showing how Maxima performs typical algebraic calculations:

(%i5) e : x - k;
(%o5) x - k
(%i6) f : x + k;
(%o6) x + k
(%i7) g : e*f;
(%o7) (x - k) (x + k)
(%i8) expand(%);
(%o8) x - k
(%i9) factor(%);
(%o9) (x - k) (x + k)
(%i10) diff(%,x);

(%o10) 2 x

(%i11) solve(diff(g,x) = 3, x);

(%o11) [x = -2]

(%i12) solve(g = 1, k);

(%o12) [k = -%I, k = %I]

7.2.2 Variables and Values

As pointed above, sometimes variables are used to stand for values and sometimes they are used to stand for unknowns.

The values command tells which variables have been assigned values. Given the calculations of the previous subsection we have:

(%i14) values;

(%o14) [a, b, c, e, f, g]

These are the variables which have been assigned values by assignment statements. We can see the values assigned to these variables:

(%i15) [a,b,c,d,e,f,g];

(%o15) [10, 6, 16, d, x - k, x + k, (x - k) (x + k)]

Thus a has been assigned the value 100, f has been assigned the value x + k etc.

Earlier we solved equations for the variables x and k, but this is not the same as assigning values. To confirm this we check the values assigned to x and k:

(%i16) [x,k];

(%o16) [x, k]

indicating no values have been assigned to these variables.
Values assigned to variables are not permanent:

(%i17) a;
(%o17) 10
(%i18) a : a + c;
(%o18) 26
(%i19) a;
(%o19) 26

The `kill` command can be used to “unassign” variables:

(%i20) kill(a,b,c);
(%o20) DONE
(%i21) values;
(%o21) [e, f, g]
(%i22) a;
(%o22) a

7.2.3 An Important Example

The relation between variables and values can be a bit tricky. Consider a situation where variables \(a, b\) and \(c\) are yet to be assigned values:

(%i23) [a,b,c];
(%o23) [a, b, c]

Now consider:

(%i24) c : a + b;
(%o24) b + a
(%i25) a : 10;
(%o25) 10
(%i26) b : 6;
(%o26) 6

What value should \(c\) have now? We can see what Maxima does:
You might think that \( c \) should now have the value 16. This is done by some other computer algebra systems but it leads to serious problems. The rule used by Maxima is clear and consistent:

Once a variable has been assigned a value this value is not changed by subsequent assignments, except, of course, to the variable itself.

This means, for example, that when once \( c \) is assigned the value \( a+b \) subsequent assignments to \( a \) and \( b \) will not change the value assigned to \( c \).

### 7.2.4 Substitution

The command

\[
\text{subst}(a = b, \text{expr})
\]
substitutes the expression \( b \) for the variable \( a \) in the expression \( \text{expr} \). Here are some examples

\[
\begin{align*}
(\%i42) \quad f & : \sin((x+y+z)/2); \\
& \quad \frac{z + y + x}{2} \\
& \quad \text{SIN}(-1) \\
(\%o42) & \\
(\%i43) \quad \text{subst}(z = 10, f); \\
& \quad \frac{y + x + 10}{2} \\
& \quad \text{SIN}(-1) \\
(\%o43) & \\
(\%i44) \quad \text{subst}(x = \cos(a+b), f); \\
& \quad \frac{z + y + \cos(b + a)}{2} \\
& \quad \text{SIN}(-1) \\
(\%o44) & \\
\end{align*}
\]

Note here that previous substitution of \( z=10 \) in \( f \) did not changed \( f \), rather it created a new expression.

\[
\begin{align*}
(\%i45) \quad \text{subst}(\sin = \cos, f); \\
& \quad \frac{z + y + x}{2} \\
& \quad \text{COS}(-1) \\
(\%o45) & \\
\end{align*}
\]
To perform multiple substitutions use

\[
\text{subst([eqn}_1, \ldots, \text{eqn}_n], \text{expr})
\]

where each of the \text{eqn}_i are equations indicating the substitutions to be made.

\[
\text{(146) } c : a + b;
\]

\[
\text{(146) } b + a
\]

\[
\text{(147) } \text{subst([a=10, b=6], c)};
\]

\[
\text{(147) } 16
\]

\[
\text{(148) } \text{subst([x = \%pi, y = \%pi, sin = cos], f)};
\]

\[
\text{(148) } z - \cos(-)
\]

\[
\text{(149) } \frac{\text{sqrt}(2)}{2}
\]

\[
\text{(149) } \text{subst(z = \%pi/2, \%)};
\]

\[
\text{(149) } \frac{\text{sqrt}(2)}{2}
\]

### 7.3 Expressions

The most general sort of objects Maxima works with are \textit{expressions}. These are built up from numbers, variables and operators like + and \textit{sin}.

#### 7.3.1 The Structure of Expressions

Maxima itself is written in Lisp. To understand how expressions are structured in Maxima it is helpful to understand how these expressions are represented in the Lisp which underlies Maxima.

Lisp uses a prefix notation for function application. For example

\[
f(x, y, z)
\]

is written in Lisp as

\[
(f \ x \ y \ x)
\]

Similarly,

\[
a + b + c + d
\]

is written in Lisp as
Note that Lisp (and Maxima) treat algebraic operators like + and * just like any other functions.

The parts of an expression may themselves be expressions containing operators. For example consider the expression

\[ \sin((x+y+z)/2) \]

Here the operator \( \sin \) is applied to the expression \((x+y+z)/2\) which in turn consists of the division operator / applied to the expressions \((x+y+z)\) and 2. Thus the Lisp representation of

\[ \sin((x+y+z)/2) \]

is therefore

\[ (\sin (/ (+ x y z) 2)) \]

The same ideas apply to more complicated expressions. For example

\[ k \cos(a*x + b*y + c) \sin(a*x + b*y + c) \]

is written in Lisp as

\[ (* k (\cos (+ (* a x) (* b y) c)) (\sin (+ (* a x) (* b y) c))) \]

### 7.3.2 Parts of an Expression

To understand how parts of an expression are extracted it is handy to keep in mind the Lisp representation of expressions described above.

Applied to an expression, \texttt{length(expr)} gives the number of subexpressions, not counting the initial operator. Subexpressions are extracted by \texttt{part(expr,n)}, with \texttt{part(expr,0)} giving the operator. Here is our first example:

\begin{verbatim}
(%i2) f : x + y + z;
(%o2) z + y + x
(%i3) length(f);
(%o3) 3
(%i4) [part(f,0), part(f,1), part(f,2), part(f,3)];
(%o4) [+, z, y, x]
\end{verbatim}

Note how this ties in with the Lisp representation

\[ f = (+ z y x) \]
Here is another example:

(%i5) g : sin((x+y+z)/2);

\[
\frac{z + y + x}{2}
\]

(%o5) \text{SIN}\left(\frac{z + y + x}{2}\right)

(%i6) length(g);

(%o6) 1

(%i7) part(g,0);

(%o7) \text{SIN}

(%i8) h : part(g,1);

\[
\frac{z + y + x}{2}
\]

We can look at the parts of \( h \):

(%i9) part(h,0);

(%o9) //

(%i10) part(h,1);

(%o10) \frac{z + y + x}{2}

(%i11) part(h,2);

(%o11) 2

There is a second form \( \text{part}(\text{expr}, n_1, \ldots, n_k) \) which selects part \( n_k \) of part \( \ldots \) of part \( n_2 \) of part \( n_1 \) of \( \text{expr} \):

(%i12) g;

\[
\frac{z + y + x}{2}
\]

(%o12) \text{SIN}\left(\frac{z + y + x}{2}\right)

(%i13) part(g,1,1,2);

(%o13) \text{y}

In this example you would find:

\[
\begin{align*}
\text{part}(g,1) &= (z+y+x)/2 \\
\text{part}(g,1,1) &= (z+y+x) \\
\text{part}(g,1,1,1) &= z \\
\text{part}(g,1,1,2) &= y \\
\text{part}(g,1,1,3) &= x
\end{align*}
\]
A handy command is `dpart` which marks a part of an expression:

```
(%i14) dpart(g,1,1);

"z + y + x"
```

```
(%o14) SIN(--------------)
    2
```

The `substpart` command is used to substitute for a part of an expression rather than a variable.

```
(%i15) substpart(x*y*z,g,1,1);

x y z
```

```
(%o15) SIN(-----)
    2
```

7.4 Lists

Lists are quite useful in Maxima and we have already seen a few examples. Lists are written by enclosing comma separated elements in square brackets, for example:

```
(%i2) x : [a, b, c, d];
```

```
(%o2) [a, b, c, d]
```

`length` gives the length of the list:

```
(%i3) length(x);
```

```
(%o3) 4
```

Elements of a list can be extracted as follows:

```
(%i4) x[1];
```

```
(%o4) a
```

```
(%i5) x[4];
```

```
(%o5) d
```