Useful Maple Functions and Hints

Douglas R. Stinson
School of Computer Science
University of Waterloo
Waterloo, Ontario, N2L 3G1, Canada

March 15, 2016

Commands

`with(numtheory)`	loads the Maple number theory package
`ifactors(n)`	factors the integer n into primes
`igcd(m,n)`	finds the greatest common denominator of integers m and n
`order(a,n)`	finds the order of a modulo n
`isprime(n)`	returns "true" if n is prime
`nextprime(n)`	finds the smallest prime greater than n
`phi(n)`	computes $\phi(n)$, where $\phi(\cdot)$ is the Euler phi-function
`primroot(g,n)`	finds the smallest primitive root of n greater than g
`msqrt(a,n)`	finds a square root of a modulo n (if it exists)
`a mod n`	reduces a modulo n, returning a value between 0 and $n-1$
`1/a mod n`	computes a^{-1} modulo n (if it exists)
`Power(a,b) mod n`	computes a^b modulo n using the square-and-multiply algorithm
`a &^ b mod n`	equivalent to `Power(a,b) mod n`

Functions

Chinese Remainder Theorem

The function `chrem` uses the Chinese remainder theorem to solve a system of linear equations. That is, `chrem([a1,a2,a3],[m1,m2,m3])` finds x such that $x \equiv a_j$ modulo m_j, $j = 1, 2, 3$. For example,

```
chrem([5,3,10],[7,11,13])
```

returns 894.

Lagrange Interpolation

`interp([x1,x2,x3],[y1,y2,y3],x) mod n` uses Lagrange interpolation to find the polynomial $f(x)$ such that $f(x_j) \equiv y_j$ modulo n, $j = 1, 2, 3$. For example,

```
interp([1,3,5],[8,10,11],x) mod 17
```

returns $2x^2 + 10x + 13$.

Solving Equations

The function `solve(eqn,var)` will find all solutions to the equation `eqn` in the variable `var`. For example,

```
solve(x^2 - 10*x + 24 = 0, x);
```

yields the two solution $6, 4$. If we execute the command

```
soln := solve(x^2 - 10*x + 24 = 0, x);,
```

then $soln[1] = 6$ and $soln[2] = 4$.

Factoring Integers

The command

```
L:=ifactors(123456);
```

yields the result

```
L := [1, [[2, 6], [3, 1], [643, 1]]].
```

Here $L[1] = 1$; this indicates that the number being factored is positive. The list $L[2]$ is

```
[[2, 6], [3, 1], [643, 1]],
```

which means that the factorization is $123456 = 2^6 3^1 643^1$. The distinct prime factors of 123 are specified by $L[2, 1, 1] = 2$, $L[2, 2, 1] = 3$ and $L[2, 3, 1] = 643$.

Continued Fractions

The continued fraction expansion of a number m is obtained from the command `cfrac(m,'quotients')`. For example, the command

```
M:=cfrac(60728973/160523347,'quotients');
```

yields the result

```
M := [0, 2, 1, 1, 1, 4, 12, 102, 1, 1, 2, 3, 2, 2, 36].
```

The nth convergent of M is obtained from the command `nthconver(M,n);`. For example, the command `C:=nthconver(M,6);` yields `C := 171/452`. The numerator of this fraction (171) can be obtained as `op(1,C);` and the denominator (452) is `op(2,C);`.

Checking Running Times

To check the running time of a series of operations in Maple, use the following commands:

```
printlevel := -1;
start_time:= time();
perform desired operations
elapsed_time := (time() - start_time)*seconds;
print(elapsed_time);
printlevel := 1;
```

Random Number Generation

To generate a random integer in the range from `lower_bound` to `upper_bound`, use the following commands:

```
with(RandomTools);
seed := any desired value;
SetState( state = seed );
lower_bound := any desired value;
upper_bound := any desired value;
Generate(integer(range = lower_bound..upperbound));
```

Loops

Maple supports various types of loop structures, if-then-else constructs, etc. For example:

```
while condition do
statements
end do;
```

and

```
if condition then
statements
else
more statements
end if;
```

Procedures

Procedures have the following format:

```
Proc_name := proc( parameter list )
local  local variables ;
procedure statements
return a,b,c
end proc;
```

To invoke a procedure and assign the outputs, use the following commands:

```
Out_params := Proc_name( parameter list )
a1:= Out_params[1];
b1:= Out_params[2];
c1:= Out_params[3];
```

Printing

The variable `printlevel` controls how much output is produced by a Maple program. The default value of `printlevel` is 1; this causes all variables at the "top" level to be printed. To automatically print variables in (nested) loops, set `printlevel := 2` (or a higher level, if desired). To turn off all output except for user-defined print statements, set `printlevel := 0`.

Miscellaneous

To include an inline comment in a Maple program, begin the line with `#`. To continue input to the next line (this is useful for assigning large integer values), use a backslash character (\) at the end of a line.