New features in Maple V Release 3

Edited by Michael Monagan[1]

1 Development of Improved User Interfaces

Most of the improvements and work done for this release has been to make the user interfaces consistant accross the main platforms, namely under Windows on the PC, the Macintosh, and under X Windows on Unix platforms. Many small improvements and fixes have been made. The new interfaces provide more tools and support for getting output from parts of worksheets into PostScript files for preparing LaTeX documents and books.

1.1 Export to LaTeX

New versions of Maple will support an "Export to LaTeX" facility that allows the user to convert his worksheet into a LaTeX file. The conversion includes converting Maple mathematical formulae into LaTeX. Large expressions will automatically be broken up. The conversion is faithful to the Maple style of output of Mathematical expressions. For example

```
# This is a line of text
> Int( exp(-t)*ln(t), t=0..infinity ) = int( exp(-t)*ln(t),t=0..infinity );

                      infinity
                         /
                        |
                        |      exp(- t) ln(t) dt = - gamma
                        |
                       /
                       0
```

[1] Institute for Scientific Computation, ETH Zentrum, CH 8092 Zurich, Switzerland.

would output the following LaTeX file

```
%% Created by Maple V Research
\documentstyle[maplems]{article}
\begin{document}
This is a line of text
\begin{mapleinput}
Int( exp(-t)*ln(t), t=0..infinity ) = int( exp(-t)*ln(t),t=0..infinity );
\end{mapleinput}
\begin{maplelatex}
\[
{\displaystyle \int_{0}^{{ \infty}}} {\rm e}^{(\, - {t}\,)}\,
{\rm ln}(\,{t}\,)\,{d}{t}= - { \gamma}
\]
\end{maplelatex}
\end{document}
%% End of Maple V Session Output
```

1.2 Toolbars

The Maple Windows user interface now have more Toolbars which are more convenient for operations such as interrupting Maple, cutting, pasting, and copying, and executing certain Maple commands.

1.3 Subscripts

Maple array and table subscripts will now print as proper subscripts in the X Window, Windows and Macintosh user interfaces.

1.4 Help Browser

The X Window version of Maple now has a help browser for easier access to the on-line help database, e.g. by subject.

1.5 Keyword Search

The Windows, Macintosh and X interfaces have a keyword search facility for searching the on-line help data base.

2 Graphics

Most of the improvements in the rendering software have been to fix problems on the various platforms. New plotting facilities include.

2.1 Plotting discontinuities

A facility for identifying discontinuities in an algebraic expression. For example

```
> plot( tan(x) , x = -10..10, y=-4..4, discont = true );
> plot( -1.5 + Heaviside(x) + Heaviside( 2 - x ) , x=-5..5, discont = true );
```

2.2 Statistical plotting routines

New statistical plotting routines.

```
> with(stats[statplots]);

  [boxplot, changecolour, histogram, notchedbox, quantile, quantile2,

    scatter1d, scatter2d, symmetry, xscale, xshift, xyexchange]
```

See section below on `stats` package for examples.

2.3 New options

See `?plot,options` and `?plot3d,options` and `?plot3d,structure` for details.

- `contours=n` specifies the number of contour lines to be drawn
- `thickness=n` where n is 0,1,2, or 3 for changing line thickness
- `symbol=s` specifies how a point is to be displayed where s is one of `BOX`, `CROSS`, `CIRCLE`, `POINT`, or `DIAMOND`
- `font=f` for changing the font of a TEXT object where f is a list of the form [family, style, size], family is one of `TIMES`, `COURIER` or `HELVITICA`, style is one of `BOLD`, `BOLDOBLIQUE`, `ITALIC`, or `BOLDITALIC` and var is the point size.

3 New and Enhanced Mathematical Facilities

3.1 Definite Integration

Work is proceeding on generalizing certain classes of integrals which yield solutions in terms of special functions. This solution of the following integral is computed in terms of the Beta function and its derivative w.r.t. to its first argument.

```
> f:=(x^(-1/3))*ln(x)*((1-x)^(1/2)):
> Int(f,x=0..1) = simplify(int(f,x=0..1));
  1
  /             1/2
 |  ln(x) (1 - x)
 |  --------------- dx = 3/98
 |         1/3
/         x
0

                       1/2
  (- 42 gamma + 7 Pi 3    - 63 ln(3) - 288 - 42 Psi(1/6)) GAMMA(2/3) GAMMA(5/6)
  ------------------------------------------------------------------------------
                                         1/2
                                       Pi
```

The solution of this integral involves the Beta and hypergeometric functions.

```
> assume(a>0): assume(b<0):
> f:=x^(a-1)*(1-Kt*x)^(-b):
> Int(f,x=0..1) = int(f,x=0..1);

              1
              /
             |   (a~ - 1)           (- b~)         Beta(a~, 1) a~
             |  x         (1 - Kt x)       dx = ---------------------
             |                                                    b~
            /                                  (a~ - b~) (1 - Kt)
            0

                  Beta(a~, 1) a~
             - ----------------------
                                  b~
               Kt (a~ - b~) (1 - Kt)

               Beta(a~, 1) a~ hypergeom([a~ - 1, b~], [a~], Kt)
             + ------------------------------------------------
                                 Kt (a~ - b~)
```

Is the answer correct? We include one numerical check by comparing the formula obtained with the result from numerical integration at some particular values.

```
> subs(a=1/2,Kt=1/3,b=-6,");

     1
    /            6
   |   (1 - 1/3 x)
   |   ------------ dx =
   |        1/2
  /        x
  0

          128
      - ---- Beta(1/2, 1) + 3/13 Beta(1/2, 1) hypergeom([-6, -1/2], [1/2], 1/3)
         9477

> simplify(");
                                    1
                                   /         6
                                  |   (x - 3)        845440
                         1/729    |   -------- dx = ------
                                  |      1/2         729729
                                 /      x
                                 0

> evalf(");

                          1.158567084 = 1.158567084

> a := 'a': b := 'b':
```

3.2 Indefinite Integration

Maple now knows how to integrate some Bessel Functions, for example

```
> int(x^9*BesselY(3,2*x^2),x);

                                   8               2
                                2 x  BesselY(4, 4 x )

> assume(a>0):
> int( x^(2*a)*BesselJ(a,x)*BesselJ(a-1,x),x);

      (2 a~ + 1)
  1/4 x
```

```
      (BesselJ(a~, x) BesselJ(a~ - 1, x) + BesselJ(a~ + 1, x) BesselJ(a~, x))/a~

> a := 'a':
```

Maple now supports Hermite reduction for integrals of the form

$$\int f(ux+v)\frac{p(x)}{q(x)}\,dx$$

where f is either exp, ln, sin or cos and $p(x)$ and $q(x)$ are polynomials. Examples

```
> f := exp(-z)/z^2/(z^2 + 6*z + 6)^2:
> Int(f,z) = int(f,z);
     /                                                      2
    |        exp(- z)                 exp(- z) (2 + 5 z + z )
    |  ------------------ dz = - 1/12 ----------------------- + 1/12 Ei(1, z)
    |   2   2           2                   2
   /   z  (z  + 6 z + 6)                  (z  + 6 z + 6) z

> f := exp(-z)/z/(z+1)/(z^3 + 2):
> Int(f,z) = int(f,z);

          /
         |        exp(- z)
         |  ------------------ dz = - 1/2 Ei(1, z) + exp(1) Ei(1, z + 1)
         |              3
        /   z (z + 1) (z  + 2)

              / -----                    2                          \
              |  \          (2 + 2 _R1 + _R1 ) exp(_R1) Ei(1, z + _R1)|
            - |   )     1/6 ------------------------------------------|
              |  /                             2                     |
              | -----                       _R1                      |
              \_R1 = %1                                              /

                                                3
 %1 :=                           RootOf(_Z  - 2)
> f := ln(x)/x/(x^2+1)^2:
> Int(f,x) = int(f,x);

      /
     |     ln(x)              2
     |  ----------- dx = 1/2 ln(x)  + 1/2 Pi arctan(x) + 1/2 dilog(- I x)
     |      2    2
    /    x (x  + 1)

                                                                2
                                           2             ln(x) x
          + 1/2 dilog(I x) + 1/4 ln(x  + 1) - 1/2 --------
                                                        2
                                                      x  + 1
```

```
> f := sin(z)/(z^3-z-1):
> Int(f,z) = int(f,z);
        /                -----
       |    sin(z)        \      Si(z - _R1) cos(_R1) + Ci(z - _R1) sin(_R1)
       |  ---------- dz =  )     ----------------------------------------
       |   3              /                          2
      /   z  - z - 1     -----                  3 _R1  - 1
                        _R1 = %1

                                      3
 %1 :=                      RootOf(_Z  - _Z - 1)
```

3.3 signum(0) = ?

The definition of signum has changed so that signum(0) is now undefined by default. The value of signum(<expr which evaluates to 0>) will return a result which could be 0 or any complex number of magnitude 1. Why not simply define signum(0) to be 0? Or 1 as it was previously? Because any specific definition will not permit certain desirable simplifications from taking place. For example, the transformation

signum(abs(x)) $\rightarrow$ 1

would be wrong at x=0 if signum(0)=0. Similarly, for real x and y, the transformation

signum(x y) $\rightarrow$ signum(x) signum(y)

would be wrong for $x < 0$ and $y = 0$ if signum(0)=1. Yet both simplifications are desired in many contexts e.g. definite integration. Thus making the value of signum(0) undefined appears to be the most useful definition. The value of signum(0) can however be fixed, either by assigning to the environment variable _Envsignum0 or by using a 3-argument form of signum:

signum(0,<expr>,s).

This will evaluate signum(expr) under the assumption that signum(0)=s . The leading argument "0" specifies that the 0'th derivative of signum is being computed. The `csgn` function (complex sign) been modified in the same way. Examples

```
> assume(y,real);
> signum(abs(x)), signum(-2*x^2), signum(-2*y^2); # using signum(0)=undefined

                                           2
                              1, - signum(x) , -1

> _Envsignum0 := 0; # using signum(0)=0
```

```
> signum(abs(x)), signum(-x^2), signum(-2*y^2);

                                          2               2
                signum(abs(x)), - signum(x) , - signum(y~)

> _Envsignum0 := 1; # using the old behaviour
> signum(abs(x)), signum(-x^2), signum(-2*y^2);

                                  2              2
                       1, signum(- x ), signum(- y~ )

> y := 'y': # clear y
```

3.4 Laplace transforms

The laplace functions have been extended to handle convolutions. E.g. given

```
> f := laplace(F(t),t,s):
> g := laplace(G(t),t,s):
> h := laplace(H(t),t,s):
```

Direct convolutions

```
> invlaplace(f*g,s,t);
                               t
                              /
                             |
                             |  G(_U1) F(t - _U1) d_U1
                             |
                            /
                            0

> laplace(",t,s); # check answer

                 laplace(F(t), t, s) laplace(G(t), t, s)
> invlaplace(f*g*h,s,t);

                t  _U1
               /   /
              |   |
              |   |   H(_U3) G(_U1 - _U3) d_U3 F(t - _U1) d_U1
              |   |
             /   /
             0   0
```

```
> invlaplace(exp(s)*f,s,t); # case where one of the products has a transform

                         Heaviside(t + 1) F(t + 1)

> laplace(",t,s);

                          laplace(F(t + 1), t, s)

> invlaplace(exp(-s)*f,s,t);

                         Heaviside(t - 1) F(t - 1)
> invlaplace(s*f,s,t); # derivative case for convolution

                        /  d      \
                        |---- F(t)| + F(0) Dirac(t)
                        \ dt      /

> laplace(",t,s);

                           s laplace(F(t), t, s)
> invlaplace(1/(s - 1)*f,s,t); # integration case of convolution

                           t
                           /
                          |
                          |  exp(_U) F(t - _U) d_U
                          |
                         /
                         0

> laplace(",t,s);

                            laplace(F(t), t, s)
                            -------------------
                                   s - 1
```

3.5 RootOf

The value RootOf$(a(x) = 0, x)$ specifies a root of $a(x)$. This is used to represent algebraic numbers and functions in Maple that cannot be expressed in terms of radicals. RootOf$(a(x) = 0, x)$ represents any of the roots. The extension in the new version (for algebraic numbers)

$$\text{RootOf}(a(x) = x, x, z)$$

specifies the complex root of $a(x)$ which is near the numerical approximation z. For example (further examples are given below for `minimize` and `solve`)

```
> RootOf(x^4-2,x,+1.2);

                                       4
                              RootOf(_Z  - 2, 1.2)

> evalf(");

                                  1.189207115
```

An example of a series expansion

```
> series(RootOf(x^4+k*x-16, x, 2.001*I), k);

           2                                             2                 2
    RootOf(_Z  + 4, 2.001 I) + 1/16 k + 1/2048 RootOf(_Z  + 4, 2.001 I) k

                                 2                 4              5      6
         + 7/8388608 RootOf(_Z  + 4, 2.001 I) k  - 1/8388608 k  + O(k )
```

3.6 The `minimize` command

The definition of minimize now supports the option **`infinite`** which means minimize will minimize over the closed real interval [infinity,infinity]. The default is to minimize over (-infinity,infinity). For example

```
> f := x^3-5*x^2+4;

                                     3      2
                               f := x  - 5 x  + 4

> minimize(f,x);

                                       392
                                     - ---
                                        27

> minimize(f,x,infinite);

                                   - infinity
```

For polynomials, if the minimum cannot be expressed in terms of radicals, it is now expressed in the RootOf notation where the root is identified as the second argument. For example

```
> g := x^4-x+4;

                                      4
                                g := x  - x + 4

> minimize(g,x);
```

```
                             3
      - 3/4 RootOf(4 _Z  - 1, .6299605249) + 4
```

```
> evalf(");
```

```
                         3.527529606
```

Also, minimize supports minimization over ranges, for example

```
> minimize( -2*y^2/((y-x^2)^4+1)+x^2+y^2, {x,y}, {x=-1..1,y=-1..1} );

                   10         6         8         2         4
        - 1/4 %1   - 5/2 %1  + 5/4 %1  - 1/2 %1  + 5/2 %1  - 1/2

 %1 := RootOf(

              2        4        6        8        10        12        14     16
   - 4 + 8 _Z  + 16 _Z  - 56 _Z  + 72 _Z  - 56 _Z   + 28 _Z   - 8 _Z   + _Z  ,

   -.6891910084)

> evalf(");

                         -.3837893698
```

3.7 Solving algebraic equations

A new method for solving large systems which have algebraic functions solutions. A better method for finding the solutions during back substitution not rationalizing the algebraic functions. An example of such a system

```
> e1 := H1^2 + (2*H2*H3/3 + H3^2*x1/3) *(x3 - 3/2*x1*x2 + x1^3/2):
> e2 := 2*H1*H2 - H2*H3*(x1^2 - x2) + H3^2*(-x1/2*(x1^2 - x2) +
>     (x3 - 3/2*x1*x2 + x1^3/2)/3):
> e3 := 2*H1*H3 + H2^2 + 2*H2*H3*x1 + H3^2*(x1^2 + x2)/2:
> eqns := {0 = a1 + a2*H1 + a3*e1, 1 = a2*H2 + a3*e2, 0 = a2*H3 + a3*e3}:
```

where we want to solve for $H1, H2, H3$ where the solutions are algebraic functions of the parameters $a1, a2, a3, x1, x2, x3$. The solutions involve the roots of an irreducible polynomial over degree 8.

Solving with radicals improved in general and also selects correct roots

```
> solve( sqrt(x+2)-sqrt(2*x-1)=1 );

                                1/2
                         6 - 2 7


> solve( {x^2-y+2, x*y=sqrt(75), x-sqrt(3)+y-5}, {x,y} );
```

```
                                      1/2
                                {x = 3   , y = 5}
```

Added power in solving systems of trigonometric equations

```
> e1 := x - (l1*cos(t1) + l2*cos(t1+t2)):
> e2 := y - (l1*sin(t1) + l2*sin(t1+t2)):
> solve({e1,e2}, {t1,t2});
{t1 = 2 arctan(%1),

                      3        2              2        2         2
 t2 = - 2 arctan((%1 x  + %1 l1  x + 2 %1 l1 x  - %1 l2  x + %1 y  x

              2       3    2       2       2      /      2          2         3
      + 2 %1 y  l1 - y  - x  y + l2  y - l1  y)  /  (%1 x  y - %1 l2  y + %1 y
                                                /

               2                               2       2       2      3
      - %1 y l1  + 2 %1 y l1 l2 + 2 l1 x l2 + y  x - l2  x - l1  x + x ))

 }

                 2              2    2    2    2                2    2     2
 %1 := RootOf((l1  + 2 l1 x - l2  + y  + x ) _Z  - 4 _Z y l1 + x  + y  - l2

                  2
     - 2 l1 x + l1 )
```

Equations involving RootOfs are now handled more generally. For example, here is an algebraic function in a.

```
> r := RootOf(x^2+a*x+1,x):
> solve( r^2+1/r=a^2-1/a, a);
                                4       3                2
                     RootOf(2 _Z  + 2 _Z  + 1 - 2 _Z + _Z )
```

More generality in the solutions of inequations

```
> solve( x^3+x > 5-2/x );
                                             4     2
           {0 < x, x < RootOf(- 5 _Z + 2 + _Z  + _Z , .4482633973)},

                                      4     2
               {RootOf(- 5 _Z + 2 + _Z  + _Z , 1.292690487) < x}
```

3.8 Complex evaluation of the Riemann Zeta function

Complex numerical evaluation of the Riemann Zeta function and its derivatives has been implemented. Note, Zeta(n, z) is the nth derivative of Zeta(z) and Zeta(n, z, q) is the nth derivative of the generalized Zeta function Zeta($0, z, q$). For Zeta($0, z, q$), for Re(z)<0 the index q is limited to the range $0 < q \leq 1$, and for Zeta(n, z, q), is only available for Re(z)>0. Examples

```
> Zeta(3.5+4.5*I);

                           .9241880340 + .01456451995 I

 # The 1st non-trivial 0 (approx):
> Zeta(1/2+14.13472514*I);
                                         -9                 -8
                      .2163160213*10   - .1358779595*10   I

 # Derivatives:
> Zeta(7, -3.5+1.9*I);
                          .01231658152 + .01218445174 I

 # 2 argument Zeta (aka, Generalized zeta function, aka Hurwitz zeta function):
> Zeta(0,3.5+4.5*I,.3);

                           43.96655549 - 51.81043075 I

 # Derivatives:
> Zeta(5,3.5+2.7*I, .7);
                         - .0002070976852 - .03085254975 I
```

3.9 Polynomial factorization over finite fields

Maple V Release 3 can factor univariate polynomials over GF(p^k). Maple represents elements of GF(p^k) as polynomials in α where α is a RootOf an irreducible polynomial. In the following example, elements of GF(3^3) are represented as polynomials over GF(2) modulo the $x^2 + 2 * x + 2$. This polynomial representation does not place any restriction on the size of the field, and it is quite feasible to compute large fields e.g. GF(2^{128}). Here is a univariate factorization

```
> alias( alpha=RootOf(x^2+2*x+2) ):
> f := 2*alpha*x^3+(2+2*alpha)*x^2+(alpha+2)*x+1;

                            3                  2
             f := 2 alpha x  + (2 + 2 alpha) x  + (alpha + 2) x + 1

> Factor(f) mod 3;

             2 alpha (x + alpha + 2) (x + 1 + 2 alpha) (x + alpha)
```

Release 3 can now factor multivariate polynomials over finite fields.

```
> f := x^3+alpha*x^2*y+alpha^2*x*y^2+alpha^3*y^3;

                          3          2          2    2        3  3
                    f := x  + alpha x  y + alpha  x y  + alpha  y

> Factor(f) mod 3;

                  (y + (alpha + 2) x) (y + alpha x) (y + 2 alpha x)
```

In prior releases we have been using the Cantor-Zassenhaus algorithm for factorization of univariate polynomials over GF(p) (p prime) and Berlekamps algorithm for GF(p^k) for $k > 1$. For GF(p) for large modulus p, Berlekamps algorithm performs better in general. We timed the two methods on the following problem taken from an article by von zur Gathen from the SIGSAM Bulletin (Vol 26, No. 2, April 1992.), namely to factor the polynomial $x^n + x + 1$ mod p for $p = 3982441812995697363688351114001$. We found that Berlekamps algorithm was approx. 2.6 times faster than the Cantor-Zassenhaus algorithm as implemented in Maple. More details comparing the two algorithms on von zur Gathen's tests can be found in a follow up article by Michael Monagan in the SIGSAM Bulletin (Vol 27, No. 2, April 1993). Note both algorithms are implemented for univariate factorization over GF(p^k) as well as GF(p).

3.10 Manipulation of sums, products, integrals and limits

We have provided more tools for manipulating unevaluated sums, integrals, products and limits. The `expand` and `simplify` commands will apply the following transformations

$$\text{expand}(\ F(\ a \times f(x), \ldots\ )\ ) \to \text{normal}(a) \times F(\text{expand}(f(x)), \ldots)$$
$$\text{simplify}(\ F(\ a \times f(x), \ldots\ )\ ) \to \text{simplify}(a) \times F(\text{simplify}(f(x)), \ldots))$$

where F is any of `sum, int, limit, product, Sum, Int, Limit, Product`. For example

```
> f := Int( sqrt(a)/Pi*exp(-u*x^2)*x^2, x ):
> f = expand(f);
                                                     /      2
                                               1/2  |      x
                                              a     |  --------- dx
                   /  1/2          2   2            |         2
                  |  a    exp(- u x ) x            /   exp(u x )
                  |  ------------------ dx = ---------------------
                  |           Pi                       Pi
                 /
```

```
> g := sum( a[i]*i/2^n, i=1..n ):
> g = simplify(g);
                           n                     /  n          \
                         -----                   |-----        |
                          \     a[i] i    (- n)  | \           |
                           )    ------ = 2       |  )   a[i] i |
                          /       n              | /           |
                         -----   2               |-----        |
                         i = 1                   \i = 1        /
```

4 Differential Equations

4.1 DESol - Solutions of Differential Equations

We have introduced a new structure `DESol` for representing the solutions of differential equations (DEs). It has the following structure

DESol(differential equation(s), variable(s), initial condition(s))

The DEs can be described either functionally using the D operator, or as an expression using the `diff` command. For example, the following describes the solutions to the equation $y'' + y = 0$.

```
> DESol( D(D(y)) = -y, y );

                                      (2)
                       de1 := DESol({D    (y) + y}, {y})

> DESol( diff(y(x),x,x) + y(x), y(x), {y(0)=0, D(y)(0)=1} );

                                 /   2      \
                                 |  d       |
                 de2 := DESol({|----- y(x)| + y(x)}, {y(x)})
                                 |   2      |
                                 \ dx       /
```

The purpose of `DESol` is to allow Maple to represent and manipulate the solution of a DE symbolically without first computing it's solution in closed form – which often is not possible. The `DESol` structure then is rather like the `RootOf` structure in Maple which represents a solution to an algebraic equation. Presently Maple knows how to differentiate, integrate, generate a series expansion, and numerically evaluate a `DESol`. Examples

```
> de := DESol( D(y)-a*y=1, y );

                       de := DESol({D(y) - a y - 1}, {y})

> D(de);
```

```
                         a DESol({D(y) - a y - 1}, {y}) + 1

> de(x);

                             /  d      \
                     DESol({|---- y(x)| - a y(x) - 1}, {y(x)})
                             \ dx      /

> int( 1/de(x), x );

                        /  d      \
              ln(DESol({|---- y(x)| - a y(x) - 1}, {y(x)}))
                        \ dx      /

                                /  d      \
                   - ln(a DESol({|---- y(x)| - a y(x) - 1}, {y(x)}) + 1)
                                \ dx      /
```

Series of `DESol` (simple ones) are now handled. By simple we mean that a Taylor series can be computed using the derivatives of the DE. This, together with the use of initial conditions, gives nice results. For example

```
> series(de(x),x=0);

                                                     2        2                  3
      y(0) + (1 + a y(0)) x + 1/2 a (1 + a y(0)) x  + 1/6 a  (1 + a y(0)) x

                  3                4          4                5      6
          + 1/24 a  (1 + a y(0)) x  + 1/120 a  (1 + a y(0)) x  + O(x )

> de := DESol( diff(x(t),t,t) = g/l*sin(x(t)), x(t), {x(0)=0, D(x)(0)=v0} ):
> series(de,t=0);

                              /            3          2   \
                   g v0   3   |        g v0          g  v0|  5      6
         v0 t + 1/6 ---- t  + |- 1/120 ----- + 1/120 -----| t  + O(t )
                     l        |          l             2  |
                              \                       l   /
```

4.2 The use of RootOfs

For ODE's where the characteristic polynomial cannot be factored, Maple now expresses the solution in terms of the roots of the characteristic polynomial using a RootOf. Example

```
> ode := diff(y(x),x$5 ) + 2*diff(y(x),x) + 2*y(x) = 0;
```

```
                    /   5      \
                    |  d       |     /  d      \
             ode := |----- y(x)| + 2 |---- y(x)| + 2 y(x) = 0
                    |   5      |     \ dx      /
                    \ dx       /

> dsolve(ode,y(x));
                                -----
                                 \
                     y(x) =       )    _C1[_R] exp(_R x)
                                 /
                                -----
                               _R = %1

                                                    5
 %1 :=                        RootOf(2 + 2 _Z + _Z )

> ode :=   diff(y(x),x$6) + 4*diff(y(x),x$4) +
>       4*diff(y(x),x$3) + 4*diff(y(x),x$2) + 8*diff(y(x),x):
> dsolve(ode,y(x));
                                                         / -----                  \
                      1/2                1/2             |  \                     |
  y(x) = _C2 + _C3 cos(2    x) + _C4 sin(2    x) + |   )    _C1[_R] exp(_R x)|
                                                         |  /                     |
                                                         | -----                  |
                                                         \_R = %1                 /

                                         3
 %1 :=                        RootOf(_Z  + 2 _Z + 4)
```

4.3 Bessel's Equation

Maple now handles Bessel's equation shifted from the origin.

```
> ode := diff(y(x),x,x) + (4*k+1)/(x+2)*\
> diff(y(x),x) + (4*(x+2)^6 + 3/(x+2)^2)*y(x);

                                        /  d      \
           /   2      \   (4 k + 1) |---- y(x)|
           |  d       |             \ dx      /   /         6       3   \
    ode := |----- y(x)| + --------------------- + |4 (x + 2)  + --------| y(x)
           |   2      |           x + 2           |                    2|
           \ dx       /                           \             (x + 2) /

> dsolve(ode,y(x));
```

```
                2           (- k)                  2     1/2             4
y(x) = _C1 (x  + 4 x + 4)      BesselJ(1/4 (4 k  - 3)   , 1/2 (x + 2) )

                2           (- k)                  2     1/2             4
     + _C2 (x  + 4 x + 4)      BesselY(1/4 (4 k  - 3)   , 1/2 (x + 2) )
```

4.4 Exponential solutions of ODE's

We have implemented the exponential linear ODE algorithm of Manuel Bronstein (Reference "Linear Ordinary Differential Equations: Breaking Through the Order 2 Barrier", Manuel Bronstein, Proceedings of ISSAC '92, ACM Press.) This algorithm finds solutions whose logarithmic derivative is in the coefficient field, i.e. $y'(x)/y(x)$ is a rational function. This includes solutions which themselves lie in the coefficient field. For example

```
> ode := (-2*x^2+x+n^2)*y(x)+(4*x^2-2*x-n^2)*diff(y(x),x)+
>        (-3*x^2+x)*diff(y(x),x$2)+x^2*diff(y(x),x$3);

                    2        2             2          2  / d      \
        ode := (- 2 x  + x + n ) y(x) + (4 x  - 2 x - n ) |---- y(x)|
                                                         \ dx      /

                              /  2       \       /  3       \
                    2         | d        |     2 | d        |
             + (- 3 x  + x) |----- y(x)| + x  |----- y(x)|
                              |   2      |       |   3      |
                              \ dx       /       \ dx       /

> dsolve(ode,y(x));
y(x) =

                         /                                  /
                        |                                  |
 _C1 exp(x) + _C2 exp(x)  |  BesselJ(n, x) dx + _C3 exp(x)  |  BesselY(n, x) dx
                        |                                  |
                       /                                  /

> subs(n=1,ode);

                 2                     2           / d      \
          (- 2 x  + x + 1) y(x) + (4 x  - 2 x - 1) |---- y(x)|
                                                   \ dx      /

                                /  2       \       /  3       \
                      2         | d        |     2 | d        |
               + (- 3 x  + x) |----- y(x)| + x  |----- y(x)|
                                |   2      |       |   3      |
                                \ dx       /       \ dx       /
```

```
> dsolve(",y(x));

    y(x) = _C1 exp(x) + _C2 exp(x) BesselJ(0, x) + _C3 exp(x) BesselY(0, x)

> ode := diff(y(x),x$2) + 3/x*diff(y(x),x) + (x^2-143)/x^2*y(x)-1/x;

                                      d
                  /   2      \       ---- y(x)      2
                  |  d       |        dx         (x  - 143) y(x)
           ode := |----- y(x)| + 3 --------- + --------------- - 1/x
                  |   2      |          x              2
                  \ dx       /                        x

> dsolve(ode,y(x));

                                   2              4             6           8
  y(x) = (980995276800 + 22295347200 x  + 278691840 x  + 2580480 x  + 20160 x

             10     12    /  13   _C1 BesselJ(12, x)   _C2 Y(12, x)
       + 144 x   + x  )  /  x   + ------------------ + ------------
                        /                 x                 x
```

4.5 Solutions in terms of DESol

If the solution cannot be computed in closed form, we can now return it in terms of the DESol data structure.

```
> ode := diff(y(x),x,x) + diff(y(x),x)*x^2 + y(x)*x;

                        /   2      \
                        |  d       |   /  d      \  2
                 ode := |----- y(x)| + |---- y(x)| x  + y(x) x
                        |   2      |   \ dx      /
                        \ dx       /

> dsolve(ode,y(x));

                    /   2       \
                    |  d        |     2 /  d       \
       y(x) = DESol({|----- _Y(x)| + x  |---- _Y(x)| + x _Y(x)}, {_Y(x)})
                    |   2       |      \ dx       /
                    \ dx        /
```

This example shows a case where a partial solution is obtained using Manuel Bronstein's algorithm.

```
> ode := diff(y(x),x$3) + 4*diff(y(x),x,x) + 4*diff(y(x),x) - 12*y(x)/x;

                 /   3      \     /   2      \
                 |  d       |     |  d       |     /  d      \      y(x)
         ode := |----- y(x)| + 4 |----- y(x)| + 4 |---- y(x)| - 12 ----
                 |   3      |     |   2      |     \ dx      /        x
                 \ dx       /     \ dx       /

> dsolve(ode,y(x));
                                                                           /
                                 2      3                  2      3   |
      y(x) = _C1 (27 x + 24 x  + 4 x ) + (27 x + 24 x  + 4 x )  |  DESol(
                                                                      |
                                                                     /

                                       /   2       \
                        2      3 |  d        |
         {(27 x + 24 x  + 4 x ) |----- _Y(x)|
                                 |   2       |
                                 \ dx        /

                                    2       3       /  d       \
              + (252 x + 132 x  + 16 x  + 81) |---- _Y(x)|
                                                  \ dx       /

                                    2       3
              + (564 x + 192 x  + 16 x  + 360) _Y(x)},    {_Y(x)}) dx
```

4.6 Solution Basis

An alternative output form is available in terms of a basis. For example

```
> ode := 2*diff(y(x),x,x) + 3*diff(y(x),x) + y(x) - exp(-3*x);

                   /   2      \
                   |  d       |     /  d      \
         ode := 2 |----- y(x)| + 3 |---- y(x)| + y(x) - exp(- 3 x)
                   |   2      |     \ dx      /
                   \ dx       /

> dsolve(ode,y(x),output=basis);

                [[exp(- x), exp(- 1/2 x)], 1/10 exp(- 3 x)]
```

4.7 Numerical Solutions

The output form now returns a list for every point (rather than a set).

```
> p := dsolve( {diff(y(x),x) = -y(x),y(0)=1},y(x),type=numeric);
p := proc(rkf45_x) ... end

> p(1);

                          [x = 1, y(x) = .3678794379225918]
```

Alternatively, the output can be a list of procedures

```
> p := dsolve({diff(x(t),t)=y(t),diff(y(t),t)=x(t)+y(t),x(0)=2,y(0)=1},
>    {x(t),y(t)}, type=numeric, output=listprocedure);

   p := [t = proc(t) ... end, x(t) = proc(t) ... end, y(t) = proc(t) ... end]
```

Now pick off the desired procedures and make a parametric plot.

```
> X := subs(p,x(t)): Y := subs(p,y(t)):
> plot([X,Y,0..3]);
```

A new numerical algorithm called dverk78 of W. Enright has been implemented which guarantees a preset accuracy for the answer.

4.8 Series Solutions

The series option to dsolve now accepts non-zero initial conditions, e.g.

```
> Order := 3:
> dsolve(diff(y(x),x$2) - y(x) = 1, y(x),series);

                                                      2       3
             y(x) = y(0) + D(y)(0) x + (1/2 + 1/2 y(0)) x  + O(x )
> dsolve({diff(y(x),x$2) - y(x) = 1,y(1)=y(1)},y(x),series);

                                                         2               3
y(x) = y(1) + D(y)(1) (- 1 + x) + (1/2 + 1/2 y(1)) (- 1 + x)  + O((- 1 + x) )
```

5 Sqrt and Radical Simplifications

5.1 Automatic radical transformations

The following transformations that Maple previously made automatically

$$(xy)^{n/d} \to x^{n/d} y^{n/d} \quad e.g. \;\; \sqrt{xy} \to \sqrt{x}\sqrt{y}$$

$$(x^r)^{n/d} \to x^{\frac{rn}{d}} \quad e.g. \;\; \sqrt{x^2} \to x, \; \sqrt{\frac{1}{x}} \to \frac{1}{\sqrt{x}}$$

which are incorrect for negative x,y, are no longer made automatically by the internal Maple simplifier. The only transformations made automatically are

(i) reduction of integer powers for integers n, a, b

$$n^{a/b} \to n^{q+r/b} \to n^q n^{r/b} \quad e.g. \;\; 2^{4/3} \to 2 \times 2^{1/3}$$

(ii) rational power simplification for integers n,d, and rational r

$$(n/d)^r \to \frac{n^r}{d^r} \quad e.g. \;\; \sqrt{2/3} \to \frac{\sqrt{2}}{\sqrt{3}}$$

(iii) combining radical powers for rational r, integers n,d

$$(x^r)^{n/d} \to x^{\frac{rn}{d}} \;\; iff \;\; -1 < r < 1 \quad e.g. \;\; (\sqrt{x})^{3/2} \to x^{3/4}$$

5.2 Radical transformations made by sqrt, combine, simplify and expand

The transformations made by the `sqrt`, `simplify`, and `combine` functions to square roots and radicals which are not correct for all values of the arguments are no longer made. Only provably correct transformations are made as determined by the `signum, csgn`, and `Re and Im` functions. For example, because of the following

```
> signum(Pi-3);
                                    1
```

i.e. Maple has proven this to be positive - it used interval arithmetic - then the following simplifications can be made.

```
> f := ((Pi-3)^2*x^2*y)^(1/2);
                                        2   2    1/2
                           f := ((Pi - 3)  x  y)

> simplify(f);
                                        2    1/2
                              (Pi - 3) (x  y)
```

Because Maple does not know anything about x, it cannot simplify this to $(\pi - 3)x\sqrt(y)$. Another example,

```
> f := 2^(1/3)*(x+1)^(1/3)*(y+1)^(1/3);

                           1/3        1/3        1/3
                     f := 2    (x + 1)    (y + 1)

> combine(f,radical);
                               1/3          1/3
                        (y + 1)    (2 x + 2)
```

Now that transformations that might be incorrect are no longer made, there needs to be mechanisms for the user to make these transformations when they are known to be correct. The user has two options. Using the symbolic option, the user can force Maple to make the transformation, i.e. effectively assuming that the sign of the expressions is positive. Hence

```
> sqrt( (Pi-3)^2*x^2, symbolic );

                               (Pi - 3) x

> sqrt( (3-Pi)^2*x^2, symbolic );

                               (Pi - 3) x

> sqrt( -a^3*b^2*c, symbolic );

                                        1/2
                             a b (- c a)
```

This is useful when you are solving an equation and you just need one root, the simplest one, and you will construct the other roots from this. For example

```
> solve(x^2+a^3*b^2*c,x);

                               1/2                1/2
                     a b (- c a)   , - a b (- c a)
```

Alternatively, the user can use the assume facility to tell Maple that expressions such as a,b,c here are real, real and positive, real and negative etc. For example

```
> assume(x>0);
> sqrt(x^2*y);
                                     1/2
                                 x~ y

> assume(x,real);
> sqrt(x^6);
                                 2    2 1/2
                               x~  (x~ )

> simplify(");
```

```
                                  3
                                x~  signum(x~)
```

5.3 The `simplify` command

The code for simplifying expressions containing square roots and general roots has been improved. Each (square) root is first simplified. The principal transformation made when $a > 0$ is

$$(a^n b)^{m/d} = (a^{q+r/d} b)^{m/d} \to a^{mq} (a^{r/d} b)^{m/d}$$

For example

```
> f := (a^8*b)^(1/3);
                                       8   1/3
                                f := (a   b)

> assume(a>0);
> simplify(f,radical);
                                  2    2   1/3
                                a~  (a~  b)
```

Then dependent integer roots are located in the expression. For example in the expression

```
> s := 6^(1/2)-2^(1/2)*3^(1/2)+1;

                                1/2    1/2  1/2
                          s := 6    - 3    2    + 1
```

It is determined that $\sqrt(6)$ can be written as $\sqrt(2)\sqrt(3)$ hence the expression is simplified to

```
> simplify(s,radical);
                                      1
```

Finally, the expression is rationalized. For example, writing the expression

```
> s := (2*x+2)^(3/2)-2*(2*x+2)^(1/2);

                                  3/2                1/2
                      s := (2 x + 2)    - 2 (2 x + 2)
```

as $(2x + 2)S - 2S$, then simplifying this as a rational expression, we obtain

```
> simplify(s,radical);
                                          1/2
                               2 (2 x + 2)    x
```

5.4 The `combine` command

Radicals can be combined together with

```
combine(expr,radical);
combine(expr,radical,symbolic);
```

The principal transformation made is

$$x^{m/d}y^{n/d} \rightarrow (x^m y^n)^{1/d} \; iff \; signum(x) > 0 \; or \; signum(y) > 0$$

For example

```
> e := 2^(1/2)*3^(1/2)*(x+1)^(3/2)*y^(3/2);

                          1/2  1/2        3/2  3/2
                  e := 2    3    (x + 1)    y

> combine(e,radical);
                               3/2          1/2
                    (x + 1) y    (6 x + 6)

> combine(e,radical,symbolic);
                                               1/2
                    (x + 1) y (6 x y + 6 y)
```

5.5 Denesting of radicals and the radnormal command

Nested square roots of the form $\sqrt{(}r + s\sqrt{(}n))$ where n is an integer, r and s are rationals are denested where possible by the `sqrt` and `simplify` functions. For example

```
> sqrt(3+4*I), sqrt(4-sqrt(12)), sqrt(1/2+sqrt(3)/4);

                         1/2            1/2        1/2
               2 + I, 3    - 1, 1/4 6    + 1/4 2
```

A more powerful facility for denesting and simplifying radicals is available with the radnormal command. Presently this facility only works for algebraic numbers, not algebraic functions. For example

```
> f := sqrt( 2*(3-sqrt(2)-sqrt(3)+sqrt(6)) );

                             1/2      1/2      1/2 1/2
                  f := (6 - 2 2    - 2 3    + 2 6    )

> readlib(radnormal)(f);
                             1/2        1/2  1/2
                    - 1 + 3    + 1/3 3    6
```

5.6 The rationalize command

The command **rationalize** will rationalize an expression containing radicals, i.e. given a rational expression of the form n/d, the result is a new expression n'/d' where no radicals appear in the new denominator d'. For example

```
> f := sqrt(2)/(1+sqrt(3));
                                          1/2
                                         2
                                f := --------
                                             1/2
                                        1 + 3

> rationalize(f);

                                  1/2          1/2
                             1/2 2    (- 1 + 3    )

> f := 1/(x^(1/3)+y^(3/2));
                                          1
                               f := -----------
                                       1/3    3/2
                                      x    + y

> rationalize(f);

                       3    3/2  1/3    2/3           9/2
                   (- y  + y    x    - x    ) (- x + y    )
                   -------------------------------------
                                      9    2
                                   - y  + x
```

5.7 The simplify command

The command **simplify(expr,assume=t)** will simplify an expression assuming all variables are of type t where t is typically real, positive. For example

```
> f := r^2 * sin (t) / sqrt(x^2+r^2-2*x*r*cos(t));
> g := int( int( f, t=0..Pi ), r=0..1 );

                         2 3/2              2 1/2  2              2 1/2
    g := 1/6 (2 ((1 + x) )    - 3 ((1 + x) )    x  - 3 ((1 + x) )    x

                         2 3/2                2 1/2  2                2 1/2
         - 2 ((- 1 + x) )    + 3 ((- 1 + x) )    x  - 3 ((- 1 + x) )    x)/x

> simplify(g,assume=real);
```

```
                                                                      3
 - 1/6 (- 2 signum(1 + x) - 3 signum(1 + x) x + signum(1 + x) x

                                                                          3
      - 2 signum(- 1 + x) + 3 signum(- 1 + x) x - signum(- 1 + x) x )/x
```

6 New and Enhanced System Facilities

6.1 Global statement

Procedures may now include an optional `global` statement which follows the `local` statement in a procedure. For example, the procedure

```
proc(n) local y; global A;
    y := 1;
    for i to n do y := x*y; A[i] := y od;
end;
```

has one parameter n, one local variable y, and one global variable A. The variables i and x are not defined, and in the older versions of Maple, would be assumed to be global variables. In this case, one can guess that x is meant to be global and i should be local. In the new version of Maple, in order to reduce the likelihood of errors, if a variable is not defined, then Maple will declare it to be local if

(i) it appears on the left hand side of an assignment statement
(ii) it is used as an index of a for statement or seq statement

otherwise Maple will declare it global. Applying these rules to the above procedure, the undeclared variable i will be declared to be local, and the undeclared variable x will be assumed global. Hence the following action is taken on the above procedure

```
Warning, i is assumed to be a local variable
proc(n) local y,i; global A;
    y := 1; for i to n do  y := x*y; A[i] := y od
end
```

The warning serves as a useful reminder to the user to declare variables.

6.2 Name protection

A protection facility has been added to prevent the user from accidentally assigning to Maple system variables. For example, the names “list” and “lhs” are now protected. Attempting to assign to them will result in

```
> lhs := x^2-1;
Error, attempting to assign to lhs which is protected
> list := [1,2,3];
Error, attempting to assign to list which is protected
```

A name can be protected using the `protect` function. For example

```
> height := proc(p,x) local t; max(seq(abs(t),t=[coeffs(p,x)])) end:
> protect(height);
> height := 1;
Error, attempting to assign to height which is protected
```

Names can be unprotected either by unassigning them, or by using the `unprotect` function. For example

```
> list := 'list':
> list := proc() convert([args],'list') end:
> list(1,2,3);

                                  [1, 2, 3]

> unprotect(height);
> height := 1;

                                 height := 1
```

6.3 `march` the Maple library archiver

Maple versions now come with a library archive facility called `march`. This tool can be used to create, insert and update ".m" files in a Maple library archive. A Maple library consists of two files, maple.lib and maple.ind. The maple.lib file contains the Maple code, a set of .m files, and the maple.ind file is a index into the file maple.lib for fast access. This facility yields faster access and is more economical in file space. It also provides a convenient way to distribute a library of Maple codes. The command

```
march -c archiveDir tableSize
```

creates a Maple archive in the directory <archiveDir> where <tableSize> is an estimate of the number of files to be put in the archive. The commands

```
march -a archiveDir fileName indexName
march -u archiveDir fileName indexName
```

adds, respectively updates, the archive in the directory <archiveDir> with the file <fileName> to be called <indexName> in the archive. There are further options for packing, extracting, removing .m files etc.

6.4 makehelp

This utility routine takes as input the name of a file which contains plain text and makes a Maple TEXT object out of a file for use with the on-line `help` command ? . The resulting TEXT object can be saved into a file to be included as on-help within a package. See `?makehelp` for details

6.5 The `fortran` command

The `fortran` function now accepts an optional argument mode = <modtype> where the mode type must be one of `single` (the default), `double`, `complex` or `generic` which specifies how function names are to be translated. For example

```
> fortran( ln(x)+sin(x) );
      t0 = alog(x)+sin(x)
> fortran( ln(x)+sin(x), mode=generic );
      t0 = log(x)+sin(x)
```

The handling of arrays has been improved (also in the `C` function) to output subscripts in lexicographical order and if an array entry is not assigned, the string `undefined` is output. For example

```
> A := array(symmetric,1..2,1..2,[(1,1)=ln(x),(2,1)=sin(x)]):
> fortran(A);
      A(1,1) = alog(x)
      A(1,2) = sin(x)
      A(2,1) = sin(x)
      A(2,2) = undefined
```

6.6 The `piecewise` function

The piecewise() function allows one to define a function with different forms on different intervals. One can also assert a global smoothness at the joints. At present, `diff/max` and `diff/min` produce answers in terms of piecewise. For example, the quadratic B-spline is given by the following piecewise quadratic polynomial. The first argument 1 states that the polynomial is C1 continuous

```
> b2 := piecewise( 1,
>                     x<0, 0,
>                     x<1, x^2/2,
>                     x<2, -3/2+x*(3-x),
>                     x<3, 9/2+x*(x-3),
>                     0 # otherwise
>                ):
> diff(b2,x);

     piecewise(0, x < 0, 0, x < 1, x, x < 2, - 2 x + 3, x < 3, 2 x - 3, 0)
```

6.7 help

The following commands have been added to display parts of on-line help files.

- The command `info`(foo) will display the first line of the help file i.e. the FUNCTION line.
- The command `usage`(foo) or ??foo will display the CALLING SEQENCE section.
- The command `example`(foo) or ???foo will display the EXAMPLES section.
- The command `related`(foo) will display the SEE ALSO section.

Note the old `example` function has been replaced by this functionality.

6.8 String and text processing capabilities

Maple strings now have no limitation on their length. Previously the length limitation was 499 characters. The `SearchText` and `searchtext` functions (case sensitive and insensitive respectively) can be used to search a string for a pattern. The functionality is

searchtext(t,s)
searchtext(t,s,a..b)

meaning search for the substring t in the string is (optionally starting at position a in the string s ending at position b).

```
> s := `maple and Maple and MAPLE`:
> SearchText(`Maple`,s);

                                      11

> searchtext(`Maple`,s);

                                      1
```

6.9 The interface command

The version of Maple is available from

```
> interface(version);

           Maple V, Release 3, Thu Aug 19 15:16:50 MET DST 1993
```

The number of bits in the machine word is available via

```
> interface(wordsize);

                                      32
```

7 Library Packages

The package mechanism has been extended to support subpackages. This works in an obvious way and is illustrated by the new stats package below.

7.1 numtheory - number theory package

The function `factorEQ`(m, d) factors the integer m in the Euclidean domain `Z`$[\sqrt(d)]$. For example

```
> numtheory[factorEQ](38477343,11);

                        1/2              1/2              1/2              1/2
       (3) (125 + 34 11    ) (125 - 34 11    ) (85 + 16 11    ) (85 - 16 11    )
```

The `isprime` function now uses one strong pseudo primality test and one Lucas test (not known to fail). numtheory[safeprime] has also been updated in the same way.

We have implemented a new iterative algorithm for numtheory[jacobi] that yields a modest improvement of 15

7.2 linalg - linear and vector algebra package

The functions `curl, diverge, grad`, and `laplacian` have been extended to work in spherical, cylindrical, and in orthogonally curvilinear coordinate systems in general (default cartesian).

```
> with(linalg,curl,grad):

> g := [r,sin(theta),z]: v := [r,theta,z]:
> curl(g,v,coords=cylindrical);

                                        sin(theta)
                                [ 0, 0, ---------- ]
                                            r

> g := r^2*sin(theta)*cos(phi): v := [r,theta,phi]:
> grad(g,v,coords=spherical);

          [ 2 r sin(theta) cos(phi), r cos(theta) cos(phi), - r sin(phi) ]
```

The functions `ihermite, ismith, hermite` and `smith` have been modified to optionally return the multiplier matrices. That is

H := ihermite(A,'U'); and H := hermite(A,x,'U');

compute the Hermite normal form matrix H in the Euclidean domains `Z` and $F[x]$ respectively and assigns U the multiplier matrix such that $H = UA$. (Hermite normal form is row reduction over a Euclidean ring). Similarly

S := ismith(A,'U','V'); and S := smith(A,x,'U','V');

compute the Smith normal form S in the Euclidean domains Z and $F[x]$ and assigns U and V the multiplier matrices such that $S = USV$. (Smith normal form is row and column reduction over a Euclidean ring).

```
> with(linalg,ismith,ihermite,inverse,hilbert):
> A := inverse(hilbert(3));

                              [  9    -36    30 ]
                              [                 ]
                         A := [ -36   192  -180 ]
                              [                 ]
                              [  30  -180   180 ]

> H := ihermite(A,'U'):
> print(H,U);

                      [ 3    0   30 ]  [ 13    9    7 ]
                      [             ]  [              ]
                      [ 0   12    0 ], [  6    4    3 ]
                      [             ]  [              ]
                      [ 0    0   60 ]  [ 20   15   12 ]

> evalm( H - U &* A );

                               [ 0  0  0 ]
                               [         ]
                               [ 0  0  0 ]
                               [         ]
                               [ 0  0  0 ]
> S := ismith(A,'U','V'):
> print(S,U,V);

               [ 3    0    0 ]  [  -3    0    1 ]  [ -5   -6  -30 ]
               [             ]  [               ]  [              ]
               [ 0   12    0 ], [  18    4   -1 ], [  1    1    0 ]
               [             ]  [               ]  [              ]
               [ 0    0   60 ]  [ -10   -5   -3 ]  [  1    1    1 ]

> evalm( S - U &* A &* V );

                               [ 0  0  0 ]
                               [         ]
                               [ 0  0  0 ]
                               [         ]
                               [ 0  0  0 ]
```

Note the new algorithm that is used for Hermite normal forms is, unlike the old algorithm, a polynomial time algorithm, and is in practice much faster for larger matrices.

7.3 GaussInt – Gaussian integer package

The routines `GIhermite` and `GIsmith` similarly have been added to compute the Hermite and Smith normal forms over the Euclidean domain $\mathtt{Z}[i]$

The routines `GIchrem` implements the Chinese Remainder Algorithm for $\mathtt{Z}[i]$

```
> GaussInt[GIchrem]([5+13*I,15-9*I],[3+4*I,7-11*I]);

                                             - 17 + 17 I
```

The routine `GInodiv` computes the number of non-associated divisors

```
> GaussInt[GInodiv](2+3*I);
                                                  2
```

8 The Statistics Package `stats`

The stats package has been completely redesigned. The data structure for the arguments to the functions has changed. This implies that programs using the previous version of the stats package will need to be updated. See ?stats[updates] on how to do this.

New types of data are now supported in addition to numbers.

- Missing data is represented by the keyword 'missing'.
- Data classes are represented as ranges $a..b$ where the bound a is inclusive and the bound b is exclusive.
- Weighted data is represented by the function Weight(data, weight). For example, the following are all valid data: 3, 3..4, missing, Weight(4,6), Weight(missing,7), Weight(3..4,9).

The stats package is subdivided into the following subpackages.

- describe: for descriptive statistics,
- transform: for data transformations,
- random: for generating numbers with a particular distribution,
- statevalf: for numerical evaluations of statistical functions,
- fit: for fitting data to a curve and
- statplots: for statistical graphics.

and the function **importdata** which supplements Maple's other input facilities.

Issuing **with(stats);** defines abbreviations for all the subpackages but not the subfunctions belonging to them. Then issuing with(describe); defines abbreviations for the functions in the describe subpackage.

```
> with(stats);

       [describe, fit, importdata, random, statevalf, statplots, transform]

> with(describe):
```

Now the following commands all refer to the same function on the data [1,5]

```
> mean([1,5]), describe[mean]([1,5]), stats[describe,mean]([1,5]);

                                   3, 3, 3
```

Certain functions require (and others allow) extra parameters. For example, describe[quartile[1]] is a function that gives the first quartile of the data. This allows one to do

```
> data:=[1,5,Weight(6,3), 9, 10, 15];

                  data := [1, 5, Weight(6, 3), 9, 10, 15]

> describe[quartile[1]](data);

                                     5
```

Functions like variance allow a parameter to specify if one is computing the sample variance or the population variance. (The parameter used the number to be subtracted from the item count, so variance[0] is the population variance and variance[1] is the sample variance).

8.1 The descriptive statistics subspackage `stats[describe]`

Contains the following functions

> coefficientofvariation, count, countmissing, covariance, decile, geometricmean, harmonicmean, kurtosis, linearcorrelation, mean, meandeviation, median, mode, moment, percentile, quadraticmean, quantile, quartile, range, skewness, standarddeviation, variance.

8.2 The transform subpackage stats[transform]

Contains

> apply, classmark, cumulativefrequency, deletemissing, divideby, frequency, moving, multiapply, remove, scaleweight, split, standardscore, statsort, statvalue, tally, tallyinto.

The function apply is used to apply a transformation on each observation, whereas moving is to apply a function across a sliding window over the data (and some can be used to smooth data using, for example, a weighted moving mean.) The function split is used to split a list of data into multiple lists, which can be required to be of equal weight. Data are counted with tally and regrouped into a specified pattern (exceptions can be reported if desired) with tallyinto.

8.3 Statistical distributions

The distributions are functions indexed by their parameters. For example, poisson[2] is a Poisson distribution with mean lambda = 2. The following distributions are supported:

> beta, binomiald (binomial), cauchy, chisquare, discreteuniform, empirical, exponential, fratio (variance ratio, Fisher F), gamma, hypergeometric laplaced (Laplace), logistic, lognormal, negativebinomial, normald (normal), poisson, studentst (T distribution), uniform, weibull.

Given a distribution, random numbers with that distribution can be obtained using the stats[random] subpackage. One can specify the quantity of numbers to be generated at once, or that a generator is to be returned.

```
> stats[random,normald[0,1]](5);

      1.175839568, -.5633641309, .2353939952, -1.442550291, -1.079196234

> T:=stats[random,studentst[3]]('generator'):
> seq(T(),i=1..5);

     -1.749151242, -.5845810002, -2.441129943, -.05425587235, 1.632365981
```

Numerical evaluations of statistical functions are computed by the stats[statevalf] subpackage. For example,

```
> statevalf[pdf, normald](0.5);

                              .3520653266
```

gives the height of the standard normal curve at the value 0.5. Similarly

```
> statevalf[cdf, chisquare[2]](0.6);

                              .2591817793
```

gives the area under the curve of the Chi square with 2 degrees of freedom lying to the left of 0.6. The inverse of this last function is obtained by using icdf – inverse cumulitive distribution function

```
> statevalf[icdf, chisquare[2]](");

                              .6000000000
```

Similar facilities are available for discrete distributions.

8.4 The regression subpackage stats[fit]

Provides for regressions. At the present, weighted least square fits of data to linear models is provided.

8.5 The plotting subpackage stats[statplots]

Contains various statistical plots and plotting utilities. The result of these functions can also be manipulated via the plots[display] function.

```
> with(statplots);

  [boxplot, changecolour, histogram, notchedbox, quantile, quantile2,

    scatter1d, scatter2d, symmetry, xscale, xshift, xyexchange]
> Xdata := [4.535, 4.029, 5.407, 1.605, 5.757, 3.527, 7.890, 8.159, 6.092,
>          13.442, 2.845, 5.172, 3.277, 8.810, 3.657, 7.226, 3.851, 2.162,
>           2.668, 4.692]:
> Ydata:=  [7.454, 4.476, 2.873, 5.476, 9.975,-1.476, 1.033, 1.140, 4.813,
>            .450, -.788, 9.389, 4.811,-3.107, 4.407, 5.534, 1.691, -.789,
>          1.684, 1.605]:
> p1 := scatter2d(Xdata,Ydata):   p1;  # scatter plot
> p2 := boxplot[15](Ydata):       p2;  # boxplot in right margin
> p3 := notchedbox[12](Xdata):    p3;  # notched box at top
> plots[display]({ p1, p2, xyexchange(p3)},
>               view =[0..17,-4..14], axes=FRAME );
```

9 New Share Library

The Maple share library is a library of Maple routines (the Maple src code included), Maple worksheets, and accompanying documentation files which have been contributed by users of Maple to the Maple user community.

The version of the share library that comes with Maple V Release 3 contains about 25 Maple applications packages, 75 Maple routines, 75 Maple worksheets and 25 additional PostScript, LaTeX and TeX articles documenting the routines and packages.

In this version of the share library we have divided the share library up into the following subject areas to make it easier for the user to find what s/he is looking for and provided a contents listing for each area with the given ? command

For each Maple routine and package in the share library we have simplified the loading mechanism to make it easier to access the share library. For example, to access the gfun package in the calculus directory in the share library, one does

Algebra	?share,algebra
Analysis (Calculus)	?share,analysis or ?share,calculus
Automatic Differentiation	?share,autodiff
Combinatorics	?share,combinat
Engineering	?share,engineer
Graphics and Geometry	?share,graphics or ?share,geometry
Linear Algebra	?share,linalg
Number Theory	?share,numtheory
Numerics	?share,numerics
Programming	?share,programming
Science	?share,science
Statistics	?share,statistics
System Tools	?share,system

```
> with(share);  # locate the share library
> readshare(gfun,calculus);  # load the gfun package
> ?gfun  # on-line help
```

In addition to Maple code, the share library also contains other files. The files ending with the suffix

1. .tex – are LaTeX or TeX src files
2. .dvi – are device independent file (output from latex or tex)
3. .ps – are PostScript files
4. .ms – are Maple worksheets
5. .in – are (plain text) Maple input files containing examples

Files with no suffix are Maple src code files. The original Maple src code is there and available to the user should s/he wish to study it or modify it. Users who would like to contribute Maple code or worksheets to the share library are encouraged to do so. Please contact

Dr. Michael Monagan
monagan@inf.ethz.ch

Here is a brief summary of some of the new routines/packages in the share library.

- perm: a package of routines for computing with permutation groups
- guesss: A routine to guess the next values in a sequence

- invar: a package of routines mainly for computing the invariant ring of permutation groups or finite linear groups over Q or $Q(\alpha)$
- macroC: a Maple package for generating C language code
- BesselH: Implementation of Hankel Functions in terms of BesselJ and BesselY
- algcurve:
- pade2: computes a generalised Pade approximation of functions $f1, f2, ..., fn$
- intpak: an experimental interval arithmetic package.
- fft, fht: fast Fourier and Hartley transforms
- surfaces: a set of procedures to calculate basic differential-geometric quantities of parametric surfaces in space
- sffge: row reduction of a sparse matrix of polynomials
- genus: the genus of an algebraic curve given by the polynomial $f(x, y)$
- normform: a package of routines for computing matrix normal forms Contains: ismithex, smithex, frobenius, jordan, and ratjordan
- puiseux: a Puiseux series expansion of an algebraic function
- ratlode: the rational solutions to a linear n'th order ODE in $y(x)$ over Q
- coxpoly: the coxeter polynomial of the rooted cycletree
- relpoly: the reliability polynomial of a network
- GB: Buchberger's Grobner bases algorithm over finite fields
- integral_basis: of an algebraic number or function
- FPS: tries to find a formula for the coefficients of a formal power series
- fields: uses Grobner bases methods to answer questions about degrees of algebraic field extensions and related questions
- polycon: a package for analysis of polynomial and rational control systems
- fjeforms: a new package for computing with differential forms
- IntSolve: an integral equation solver
- coxeter: procedures for studying roots systems and finite Coxeter groups
- weyl: for manipulating weight vectors and computing multiplicities for irreducible representations of semisimple Lie algebras

- ratinterp: rational function interpolation
- elliptic: determines the order of the group of points on a non-singular elliptic curve $y^2 = x^3 + A * x + B$ over a finite field Z mod p

In addition, new in this version of the share library are a selection of approximately 30 worksheets showing applications of Maple in various disciplines, in education and research.

10 Miscellaneous

- Convert/binomial converts expressions involving factorials and GAMMAs into binomials, for example

```
> f := (n+1)!/k!/(n+1-k)!*(2*n)!/n!^2:
> f = convert(f,binomial);

                (n + 1)! (2 n)!
             --------------------- = binomial(n + 1, k) binomial(2 n, n)
                                 2
             k! (n + 1 - k)! (n!)
```

- The combine/ln function takes an optional argument t to specify which type (default type rational) of logs you want to combine, e.g.

```
> f := (a*ln(x)+3*ln(x)-ln(1-x)+ln(1+x)/2:
> combine(f,ln);
                                          3        1/2
                                         x   (1 + x)
                           a ln(x) + ln(-------------)
                                             1 - x

> combine(f,ln,integer);
                                                          3
                                                         x
                       a ln(x) + 1/2 ln(1 + x) + ln(-----)
                                                     1 - x

> combine(f,ln,anything);
                                     a   3        1/2
                                    x   x   (1 + x)
                                 ln(----------------)
                                          1 - x
```

- numerical comparisons against infinity and -infinity are now allowed

```
> evalb( 5 < infinity );

                                      true
```

- new function: `numboccur`(a, b) count the number of occurrences of b in a

```
> numboccur( expand(sin(20*x)), x );

                                  20
```

- The system function `lprint` has changed a little so that it's output can be read back into Maple. The `lprint` form of series has changed to output a `series` command.