USING MAPLE

Math Software

The following is a collection of examples and explanations from the textbook and the Maple help files:

Solving a differential equation: p162

This solves the equation $y^{(3)} - 3y'' + y = 0$. The **first statement** defines the function. The **second statement** results in the solution in terms of sine and cosine. The **third statement** recalculates the solution in terms of floating point approximations. The terminating colons suppress output and the semicolon permits output:

```
diffeq := diff(y(x),x$3) - 3*diff(y(x),x$2) + y(x) = 0:
dsolve( diffeq, y(x) );
evalf(");
```

This solves the equation $y'' + y' = \tan(x)$ using the method of *Variation of Parameters*. The **first and second statement** are the homogeneous solutions which we must find ourselves using the characteristic equation to find the roots (refer to the topic *Characteristic Equations* in a separate document). The **third and fourth statements** instruct the software to find the differentials of y_1 and y_2. The **fifth statement** sets up the simultaneous equations for finding u_1 and u_2 and the **sixth statement** instructs that they be solved and the result placed in variable **`soln`**. The **eighth and ninth statements** instruct the software to select the derivatives of u_1 and u_2 from the solution. The **tenth and eleventh statements** instruct the software to integrate u_1' and u_2' to determine u_1 and u_2. The **final statement** forms the particular solution $y_p = u_1y_1 + u_2y_2$. The terminating colons suppress output and the semicolon permits output:

```
y1 := cos(x):
y2 := sin(x):
f := tan(x):
y1p := diff(y1, x):

y2p := diff(y2, x):
eqs := { u1p*y1 + u2p*y2 = 0, u1p*y1p + u2p*y2p = f }:
soln := solve( eqs, {u1p, u2p} );
u1p := rhs(soln[2]):

u2p := rhs(soln[1]):
u1 := int(u1p, x):
u2 := int(u2p, x);
y := simplify( u1*y1 + u2*y2);
```

Solving systems of differential equations:

This solves the system $x' = -x + 3y$, $y' = 2y$. The terminating colons suppress output and the semicolon permits output:

```
deq1 := diff(x(t),t) = -x(t) + 3*y(t):
deq2 := diff(y(t),t) = 2*y(t):
dsolve({deq1,deq2}, {x(t),y(t)});
```

We can solve for initial values by adding the line:

```
dsolve({deq1,deq2, x(0)=1, y(0)=0}, {x(t),y(t)});
```

Finding eigenvectors: p290

I'm not sure what the **first statement** does, maybe calls a function which is required when using certain commands. The **second statement** defines the 3 x 3 matrix. The **third statement** instructs the software to find the eigenvectors.

$$\mathbf{A} = \begin{bmatrix} -.05 & 0 & 0 \\ 0.5 & -0.25 & 0 \\ 0 & 0.25 & -0.2 \end{bmatrix}$$

```
with(linalg):
A := matrix(3,3,[-0.5,0,0,0.5,-0.25,0,0,0.25,-0.2]);
eigenvects(A);
```

Plotting simple functions:

This command plots y = x + 1 and y = 2x on the same graph in Maple:

```
plot({x+1,2*x},x=-5..5,y=-5..5);
```

Plotting a vector field: (I haven't tried this)

This command plots the vector field for $y' = y - \sin(x)$ with domain -4 to 4 and range -3 to 3:

```
dfieldplot( diff(y(x),x) = y - sin(x), [x,y], -4..4, y=-3..3 );
```

Function: DEtools[DEplot] - plot solutions to a system of DE's (from the Maple help files)

Calling Sequences:

```
DEplot(deqns, vars, trange, eqns)
DEplot(deqns, vars, trange, inits, eqns)
DEplot(deqns, vars, trange, yrange, xrange, eqns)
DEplot(deqns, vars, trange, inits, xrange, yrange, eqns)
```

Parameters:

```
deqns  - list or set of first order ordinary differential
  equations,
```

or a single differential equation of any order

```
vars   - dependent variable, or list or set of dependent variables
trange - range of the independent variable
inits  - initial conditions for solution curves
yrange - range of the first dependent variable
xrange - range of the second dependent variable
```

`eqns` - optional equations of the form keyword=value

Description:

Given a set or list of initial conditions (see below), and a system of first order differential equations or a single higher order differential equation, DEplot will plot solution curves, by numerical methods. A two element system of first order differential equations will also produce a direction field plot, provided the system is determined to be autonomous. For non-autonomous systems, no direction field will be produced (only solution curves will be possible in such instances). There can be ONLY one independent variable.

The default method of integration is method=classical[rk4]. Other methods may be specified in the optional equations. Note that because numerical methods are used to generate plots, the output is subject to the characteristics of the numerical method in use. In particular, unusual output may occur when dealing with asymptotes. The direction field presented consists of a grid of arrows tangential to solution curves. For each grid point, the arrow centred at (x,y) will have slope dy/dx. This slope is computed using (dy/dt)/(dx/dt), where these two derivatives are specified in the first argument to DEplot. A system is determined to be autonomous when all terms and factors, other than the differential, are free of the independent variable. See the help page for autonomous for more information. For a single higher order differential equation, only solution curves can be plotted.

By default, the two dependent variables will be plotted, unless otherwise specified in the scene option. deqns may be given as a procedure, but must conform to the specification as given in dsolve/numeric. In this instance, deqns must be of the form:

```
proc( N, ivar, Y, YP )
    ...
    YP[1] := f1(ivar,Y);
    YP[2] := f2(ivar,Y);
    ...
end
```

where N represents the number of first order equations, ivar is the independent variable, Y is a vector of length N, and YP is a vector of derivatives which should be updated by the procedure (for the equivalent first order system), also of length N. See DEtools[convertsys] for information on determining first order systems. inits should be specified as:

```
[ [x(t0)=x0,y(t0)=y0], [x(t1)=x1,y(t1)=y1], ... ]
```

where the above is a list (or set) of lists, each sublist specifying one group of initial conditions. yrange and xrange should be specified as follows:

```
y(t) = y1..y2,  x(t) = x1..x2  or
   y = y1..y2,     x = x1..x2
```

By default, integration along a solution curve stops one mesh point after the specified range is exceeded. This may be overridden by the obsrange option. The optional equations eqns consist of the following, as well as a restricted set of plot and dsolve[numeric] options:

```
'arrows'     =  arrowtype
'colour'     =  arrowcolour
'dirgrid'    =  dirarray
'iterations' =  iter
'linecolour' =  line_info
'obsrange'   =  observe
'scene'      =  scenearray
'stepsize'   =  h
```

`arrows = type` where type consists of 'SMALL', 'MEDIUM', 'LARGE', 'LINE', or 'NONE'

arrowtype indicates the type of arrow to be used in a direction field. The default, when a direction field is possible, is 'SMALL'. Specifying an arrowtype of 'NONE' will suppress the calculation or display of any direction fields.

`colour, color = arrowcolour`

arrowcolour may take a variety of forms: 1) as a plot[color] name; 2) as COLOUR('HUE',realcons); 3) as COLOUR('RGB',realcons,realcons, realcons); 4) as an expression in two variables; 5) as a procedure in two variables; 6) as a three element list of expressions in two variables; 7) and as a three element list of procedures in two variables. In the case of 4) and 5), coordinates of the arrows in the plane are passed to the expression or procedure. The resulting values are then normalized on the range [0,1] and used as 'HUE' values. In the case of 6) and 7), the resulting values are normalized on [0,1] and used as RGB values. Note that 4) and 6) must contain expressions of the plotted variables. This form of colour handling may be useful in differentiating features of the direction field. The default arrow colour is red.

`dirgrid = [integer,integer]`

dirgrid is an array of two integer values, specifying the number of horizontal and vertical mesh points to use for arrows. [2,2] is the minimum, and [20,20] is the default.

`iterations = integer`

iterations indicates the number of steps to be taken, of size defined by stepsize, between stored points. This is useful in gaining a higher accuracy of plotted points by a small stepsize, without storing and plotting a large number of points.

`linecolour, linecolor = line_info`

line_info may take one of five forms: 1) as a plot[color] name; 2) as COLOUR('HUE',realcons); 3) as COLOUR('RGB',realcons,realcons, realcons); 4) as an expression of the independent variable; 5) as a procedure of the independent variable. In the case of 4) and 5), values of the independent variable are passed to the expression or procedure. The resulting values are then normalized on [0,1] and applied as 'HUE' values. This colour feature may be useful in displaying variation of the solution curve as the independent variable changes value.

In addition, 1)-5) may be combined into a list, one element for each of the solution curves considered. Each list element will then be applied to a corresponding solution curve. The default line colour is yellow.

`obsrange = TRUE,FALSE`

obsrange indicates if the integrator should halt once the solution curve has passed outside of the specified range. This may be useful in the plotting of functions with asymptotic behavior. The default is TRUE.

`scene = [name,name]`

scene specifies the plot to be viewed. For example, scene=[x,y] indicates that the plot of x versus y (x horizontal) is to be plotted, with t implicit, while scene=[t,y] plots t versus y with t explicit. This option can also be used to change the order in which to plot the variables. There is no default ordering when vars is indicated as a set; if vars is given as a list, the given ordering will be used.

`stepsize = realcons`

stepsize specifies the distance between mesh points to be used in generating the graph. For trange=a..b, the default h value will be abs((b-a))/20. If stepsize is specified to be too large, the default will be used.

There are many other optional equations. See ?plot[options] and the individual dsolve[numeric] help pages for more information.

Examples:

```
with(DEtools):
DEplot({cos(x)*diff(y(x),x$3)-diff(y(x),x$2)+Pi*diff(y(x),x)=y(x)-
     x},\

{y(x)},x=-2.5..1.4,[[y(0)=1,D(y)(0)=2,(D@@2)(y)(0)=1]],y=-
     4..5,stepsize=.05);
```

Same plot as above, but using the procedure form of intput.

```
F1 := proc(N,x,Y,YP)                  # First order system
    YP[1] := Y[2];
    YP[2] := Y[3];
    YP[3] := (Y[3]-Pi*Y[2]+Y[1]-x)/cos(x)
end;

DEplot(F1,{y(x)},x=-2.5..1.4,[[y(0)=1,D(y)(0)=2,(D@@2)(y)(0)=1]],y=-
     4..5,\

stepsize=.05);

DEplot({D(x)(t)=y(t)-z(t), D(y)(t)=z(t)-x(t), D(z)(t)=x(t)-y(t)*2},
     {x(t), y(t),\


z(t)},t=-2..2, [[x(0)=1,y(0)=0,z(0)=2]], stepsize=.05, scene=[z(t),
     x(t)],\
   linecolour=sin(t*Pi/2),method=classical[foreuler]);

DEplot({diff(x(t),t)=x(t)*(1-y(t)),diff(y(t),t)=.3*y(t)*(x(t)-1)},
     [x(t), y(t)],\

t=-2..2,x=-1..2,y=-1..2,arrows=LARGE,title=`Lotka-Volterra model`,\
   color=[.3*y(t)*(x(t)-1),x(t)*(1-y(t)),.1]);
DEplot(diff(y(x),x)=1/2*(-x-(x^2+4*y(x))^(1/2)),y(x),x=-3..3,y=-
     3..2,\

title=`Restricted domain`,color=1/2*(-x-(x^2+4*y)));

DEplot({diff(x(t),t)=x(t)*(1-y(t)),diff(y(t),t)=.3*y(t)*(x(t)-
     1)},[x(t),\

y(t)],t=-7..7,[[x(0)=1.2,y(0)=1.2],[x(0)=1,y(0)=.7]],stepsize=.2,\
title=`Lotka-Volterra model`,color=[.3*y(t)*(x(t)-1),x(t)*(1-y(t)),\
.1],linecolor=t/2,arrows=MEDIUM,method=rkf45);
```

Produce the same plot, but use the procedure form.

```
F2 := proc(N,t,X,XP)                   # First order system
   XP[1] := X[1]*(1-X[2]);
   XP[2] := .3*X[2]*(X[1]-1)
end;
DEplot(F2,[x(t),y(t)],t=-7..7, [[x(0)=1.2,y(0)=1.2], [x(0)=1,
     y(0)=.7]],\

stepsize=.2,title=`Lotka-Volterra model`,color=[.3*y(t)*(x(t)-1), \
   x(t)*(1-y(t)),.1],linecolor=t/2,arrows=MEDIUM,method=rkf45);

DEplot(D(y)(x)=-y(x)-x^2,y(x),x=-1..2.5,[[y(0)=0],[y(0)=1],[y(0)=-
     1]],\
title=`Asymptotic solution`,colour=magenta,linecolor=[gold,yellow,\
   wheat]);
```

```
f1 := proc(N,x,Y,YP) YP[1] := -Y[1]-x^2 end:
DEplot(f1,y(x),x=-1..2.5,[[y(0)=0],[y(0)=1],[y(0)=-1]],\
   title=`Asymptotic solution using procedure`);
```