An Introduction to Ordinary Differential Equations
MATLAB files

James C. Robinson

1

MATLAB files referred to in the book

This document contains listings of all the MATLAB files available on the web, and brief descriptions of what they do and how to use them. They were all written to help create the figures and exercises in the book; as such they are in no way intended to stand up as worthy examples of elegant programming. Nevertheless, I hope that they will be useful.

Note that many of these programs use the same variable names, so it is a good idea to use the `clear` command between different programs to clear all the variables.

`besselseries.m`

This program computes the series expansion of the Bessel function of order ν that has the form

$$x^\nu \sum_{n=0}^{\infty} a_n x^n,$$

using the recurrence relation

$$a_n = -\frac{a_{n-2}}{n(n+2\nu)},$$

as discussed in Section 20.4. The program then plots this series solution, along with the true solution (the normalisation has to be changed to make the two coincide, see comments after Example 20.3, and Exercise 20.8).

```
%% Bessel function of order nu from its series expansion
%% Sum up to power x^(2N)

nu=input('nu = ');
N=input('N= ');
```

```
a(1)=-1/(4*(1+nu));

%% a(i) is coefficient of x^(2i)

for i=2:N;
   a(i)=-a(i-1)/(4*i*(i+nu));
end

x=linspace(0,10,1000);

y=1+0.*x;

hold on

for k=1:N;
   y=y+a(k)*x.^(2*k);
   ay=y.*x.^nu;
   if (k/2)~=floor(k/2) & k>1;
      plot(x,ay);
   end
end

%% Change the normalisation

z=besselj(nu,x)*2^nu*gamma(1+nu);

plot(x,z,'linewidth',2)

ylim([1.5*min(z),1.5*max(z)])
```

bifurcation.m

This short program draws the bifurcation diagram for the logistic map $x_{n+1} = f(x_n) := rx_n(1 - x_n)$. For a selection of values of r between the values you input it computes 100 iterates, and then plots the next 30 on the vertical axis. This can take a long time to run.

```
%% bifurcation.m
%% draw bifurcation diagram for logistic map

clf
hold on

r1=input('Smallest value of r: ');
r2=input('Largest value of r: ');

%% N is the number of r values used
%% lowering/increasing N will decrease/increase the resolution
%% with a consequent decrease/increase in the time needed

N=400;

for t=0:N

%% for each value of r start at x_0=0.5

   z=0.5;

   r=r1+((r2-r1)*t/N);

%% compute one hundred iterates of f

   for j=1:100
      z=r*z*(1-z);
   end

%% and then plot the next 30

   for j=1:30
      z=r*z*(1-z);
      plot(r,z,'.','MarkerSize',2)
   end

end

xlabel('r','FontSize',20)

xlim([r1 r2])
```

`cylinder.m`

This plots animated trajectories of the nonlinear pendulum equation, but moving in their natural phase space, the cylinder. The '`*`' represents the stable fixed point (the pendulum hanging vertically downwards) and the '`x`' represents the unstable fixed point (the pendulum balanced vertically upwards). If you tilt the three-dimensional figure correctly you can see something very like the swings of the pendulum. This program uses the file `pendde.m`. The program ends when the initial condition is '`999`'.

```
%%  Animate trajectories of the nonlinear pendulum
%%  on the 'phase cylinder'

%%  Requires pendde.m

hold off
```

```
%% Draw portion of cylinder

[tpx,tpz]=meshgrid(-pi:.1:pi+.1,-3:.1:3);
tpcx=cos(tpx);
tpcy=sin(tpx);
surfl(tpcx,tpcy,tpz), shading flat

hold on

axis off

%% * is pendulum pointing down
%% x is pendulum pointing up

plot3(-1,0,0,'*')
plot3(1,0,0,'x')

X0=0;

%% program ends when initial condition is 999

while X0~=999

   X0=input('Initial condition:   ');
   if imag(X0)~=0
      X0=[u(2001,1) u(2001,2)];
   end
   if X0~=999
    T=input('Maximum time:   ');
    if X0~=999

          tspan=linspace(0,T,2001);

          [t,u]=ode45('pendde',tspan,X0,[],1);

          c1(:,1)=cos(pi+u(:,1));
       c2(:,2)=sin(pi+u(:,1));
        c3(:,3)=u(:,2);

            comet3(c1(:,1),c2(:,2),c3(:,3))

        end

   end

end

hold off
```

`darrow.m`

This routine, which is handy for phase diagrams, draws an arrow at the point (`xo`, `yo`) in the direction (`nx`, `ny`) of size `s`.

```
function darrow(xo,yo,nx,ny,s)

%% Draw an arrow at (xo,yo) in the (nx,ny)
%% direction, of size s

if s==0;   s=0.4;   end

%% theta is (half) the angle the sides of the arrow make
%% to each other

theta=pi/9;

l=sqrt(nx^2+ny^2); if l~=0; nx=nx/l; ny=ny/l; end

if nx~=0
   ppx=-(ny/nx); ppy=1;
   nl=sqrt(ppx^2+ppy^2);
   ppx=ppx/nl; ppy=ppy/nl;
else
   ppx=1; ppy=0;
end

plot([xo-s*nx+s*tan(theta)*ppx xo],[yo-s*ny+s*tan(theta)*ppy yo])
plot([xo-s*nx-s*tan(theta)*ppx xo],[yo-s*ny-s*tan(theta)*ppy yo])
```

`euler.m`

This is the MATLAB implementation of Euler's method,

$$x_{n+1} = x_n + hf(x_n, t_n),$$

discussed in Section 21.3, applied as there to the simple differential equation $\dot{x} = t - x^2$.

```
%% Euler's method

T=12;         %% final time

h=0.5;        %% timestep

%% MATLAB does not allow an index 0
%% on a vector, so x_n is x(n+1) here

t(1)=0;      %% initial time
x(1)=0;      %% initial condition
```

```
for n=1:T/h;

    t(n+1)=n*h;
    x(n+1)=x(n) + h * (t(n) - x(n)^2);

end

[t x]       %% display values

%% Plot crosses at numerical values, and join these

plot(t,x,'x','MarkerSize',20)
hold on
plot(t,x)
```

f2.m

This program draws the pictures for Exercise 24.3, which asks you to investigate the periodic-doubling cascade by looking at a 'renormalised' version of the graph of f^2. First it draws graphs of f and f^2, then finds the right-hand point `nxr` of the small box (where $f^2(\texttt{nxr}) = 1 - (1/r)$) using an interval-halving method. It then draws a blown-up version of the function that appears in this small box.

```
%%  Renormalisation of logistic map f

r=input('value of r:  ');

clf reset

%%  The left-hand plot shows f and f^2

subplot(121)

x0=0:0.025:1;

%% y=f(x0) and z=f^2(x0)

y=r.*x0.*(1-x0);
z=r.*y.*(1-y);
plot(x0,y)
hold on
plot(x0,x0)
plot(x0,z, 'LineWidth', 2)
axis equal

%% Now find where f^2(nxl)=1-(1/r)
%% using the interval halving method

nxl=1-(1/r);
g=inline('(r^2*x*(1-x)*(1-(r*x*(1-x))))-1+(1/r)','x','r');
xl=1-(1/r)+0.00001; xr=1;
d=1;
while d>0.0001
   m=(xl+xr)/2;
   if g(xl,r)*g(m,r)<0
      xr=m;
   else
      xl=m;
   end
   d=(xr-xl);
end
nxr=m;

%% Draw the little 'invariant box'

plot([nxr nxr],[nxr nxl])
plot([nxr nxl],[nxl nxl])
plot([nxl nxr],[nxr nxr])
plot([nxl nxl],[nxr nxl])

xlim([0 1])
ylim([0 1])

%%  On the right-hand side draw an expanded version of little box

subplot(122)

x2=linspace(nxl,nxr,100);
y2=r.*x2.*(1-x2); y2=r.*y2.*(1-y2);
plot(x2,y2, 'LineWidth', 2);
hold on
plot(x2,x2);
axis equal
xlim([nxl nxr])
ylim([nxl nxr])
```

`flies.m`

This program, inspired by the MMath project of Warwick student James Macdonald, animates the 'swarm of flies' produced by solving the Lorenz equations for a cluster of different initial conditions using `solvem.m`. As noted below, any more than 6^3 solutions will strain most computers. [It is important *not* to use the `clear` command between running `solvem.m` and `flies.m`.]

```
%% Swarm of flies from the Lorenz equations
%% Run M-file solvem.m first

hold on

xlim([-20 20])
ylim([-30 30])
zlim([0 50])

for m=1:(d+1)^3;
   hend(m)=plot3(y0(1,m),y0(2,m),y0(3,m),'Erase','background');
   hstart(m)=plot3(y0(1,m),y0(2,m),y0(3,m),'Erase','none');
end

pause=input('Rotate axes now, then press [RETURN]');

for i=floor(length(yi)/3):length(yi)-1,

   for m=1:(d+1)^3
      set(hstart(m),'xdata',yi(i,1,m),'ydata',yi(i,2,m),'zdata',yi(i,3,m));
      drawnow
      set(hend(m),'xdata',yi(i,1,m),'ydata',yi(i,2,m),'zdata',yi(i,3,m));
      drawnow
   end

end
```

`linearde.m`

This file contains the linear equation

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

for use in `lportrait.m` (via the `ode45` routine).

```
function yprime=linearde(t,y,flag,a,b,c,d)
yprime=[a*y(1)+b*y(2); c*y(1)+d*y(2)];
```

`logistic.m`

This program will draw cobweb diagrams for the logistic map

$$x_{n+1} = rx_n(1 - x_n),$$

which forms the subject of Chapter 24. Having chosen r the program draws the graph of $f(x) = rx(1 - x)$ for $0 \leq x \leq 1$ and the diagonal ('$y = x$'). Then given an initial condition `x_0` it will plot the selected number of iterates on the diagram. When asked for the number of iterates, 0 will finish the program, .1 will clear all lines from the cobweb diagram, .2 will change the initial condition, and .3 will erase everything and allow you to choose a new value of r. The 'clear' option is particularly useful, for example, to find a stable periodic orbit: start anywhere, perform 100 iterations (say), clear the picture, and then compute a few iterations to show a clean picture of the orbit.

```
%%  logistic.m
%%  draw cobweb diagrams for logistic map

hold off
r=input('r=:  ');
x=0:0.01:1;
y=r.*x.*(1-x);
plot(x,x)
hold on
plot(x,y, 'LineWidth', 2)
z=input('x_0=:  ');
n=1; while n~=0
   n=input( ...
'# iterations (0=stop, .1=clear, .2=change IC, .3=change r):  ');
   if n==0.1
      hold off; plot(x,x); hold on; plot(x,y, 'LineWidth', 2);
   end
   if n==0.2
      z=input('new initial condition:  ');
   end
   if n==0.3
      r=input('new value of r:   ');
      hold off
      y=r.*x.*(1-x);
      plot(x,x)
      hold on
      plot(x,y)
      z=input('z_0=:  ');
   end
   if n~=0
      for j=1:abs(n)
         nz=r*z*(1-z);
         if n>0
            plot([z z],[z nz])
            ms=min(8,25*abs(nz-z));
            if z<nz plot(z,(z+nz)/2,'^','markersize',ms)
            else
               plot(z,(z+nz)/2,'v','markersize',ms)
            end
         plot([z nz],[nz nz])
            if z<nz plot((z+nz)/2,nz,'>','markersize',ms)
            else
               plot((z+nz)/2,nz,'<','markersize',ms)
            end
        end
      z=nz;
    end
    end
end
```

logisticn.m

This simple program iterates the logistic map, and plots successive iterates x_n against n. The program finishes when the initial condition lies outside the interval $[0, 1]$.

```
%% Plot iterates of logsitic map
%% against n

clf
hold on

r=input('r = ');
N=input('N = ');

xlim([0 N]); ylim([0 1]);
xlabel('n','FontSize',20); ylabel('x','FontSize',20)

x0=input('initial condition = ');

while x0>=0 & x0<=1

   x(1)=x0; t(1)=0;

   for n=2:N;

      t(n)=n-1;
      x(n)=r*x(n-1)*(1-x(n-1));

   end

   plot(t,x)
   plot(t,x,'x','MarkerSize',15)

   x0=input('initial condition = ');

end
```

`lorenzdraw.m`

This program integrates the Lorenz equations (using MATLAB's `ode45` command) up to time $t = 50$, starting at the initial condition `yzero`. It then draws the trajectory in $\mathbb{R}^3$. The equations themselves are contained in the file `lorenzde.m`.

```
%% Integrate the Lorenz equations
%% Requires lorenzde.m

tspan=[0 50];
yzero=input('yzero = ');
[t,y]=ode45('lorenzde',tspan,yzero);

%% Draw the trajectory in R^3

plot3(y(:,1),y(:,2),y(:,3))
xlabel('x'); ylabel('y'); zlabel('z');
title('Lorenz attractor')
```

`lorenzde.m`

This file contains the Lorenz equations,

$$\begin{aligned}\dot{x} &= \sigma(-x+y)\\ \dot{y} &= rx-y-xz\\ \dot{z} &= -bz+xy\end{aligned}$$

for use in `lorenzdraw.m`, `lorenz37.m`, and `solvem.m` (via the `ode45` command).

```
function yprime=lorenzde(t,y)
yprime=[10*(y(2)-y(1)); 28*y(1)-y(2)-y(1)*y(3); y(1)*y(2)-8*y(3)/3];
```

`lorenz37.m`

This long program is essentially a presentation of various topics that are covered in Chapter 37 that discusses the Lorenz equations. It would be useful to run the program as you read that chapter. Note that the program requires the `lorenzde.m` file.

Having set up the parameters it gives explicit values for the non-zero stationary points, and plots them. It then computes the eigenvalues and eigenvectors at the origin and at the non-zero stationary points.

After this it plots the direction field near each stationary point, and then, unhelpfully all these direction fields together on one plot. After this it shows three components of the solution, along with the function $V(x, y, z) = x^2 + y^2 + (z - r - \sigma)^2$ which can be used to show that the solutions remain bounded for all time.

Then it plots the trajectories in $\mathbb{R}^3$, showing the Lorenz attractor, and finally draws a graph of successive maxima of z against each other.

```
%%  Lorenz ‘presentation’
%%  Requires lorenzde.m

clf
echo

% parameters

sigma=10; r=28; b=8/3;

echo off
p=input('To continue press [RETURN]'); echo

% non-zero fixed points

x=sqrt(b*(r-1)); y=x; z=r-1;

echo off
p=input('draw the fixed points (press [RETURN])');

% draw the three fixed points & some axes

plot3([0 0],[0 0],[-40 40],'g')
hold on
plot3([0 0],[-40 40],[0 0])
plot3([-40 40],[0 0],[0 0])
plot3(0,0,0,'x','MarkerSize',15)
plot3(x,y,z,'x','MarkerSize',15)
plot3(-x,-y,-z,'x','MarkerSize',15)
xlabel('x'); ylabel('y'); zlabel('z');

p=input('To continue press [RETURN]');
echo

% eigenvalues and eigenvectors at the origin
% L gives values, V gives vectors

[V L]=eig([-sigma sigma 0; r -1 0; 0 0 -b])

% two stable directions and one unstable direction

echo off
p=input('look close to origin (press [RETURN])');
echo
```

```
% draw local direction field near origin
% the picture is nice if you rotate to -141 30

[x0,y0,z0]=meshgrid(-2:1:2,-2:1:2,-2:1:2);
xd=sigma.*(-x0+y0);
yd=r.*x0-y0-x0.*z0;
zd=-b.*z0+x0.*y0;
quiver3(x0,y0,z0,xd,yd,zd)
xlim([-2.5 2.5]);
ylim([-2.5 2.5]);
zlim([-2.5 2.5])

a1=3.5*V(:,1);
a2=3.5*V(:,2);
a3=3.5*V(:,3);
plot3([-a1(1) a1(1)],[-a1(2) a1(2)],[-a1(3) a1(3)],'g')
plot3([-a2(1) a2(1)],[-a2(2) a2(2)],[-a2(3) a2(3)],'r')
plot3([-a3(1) a3(1)],[-a3(2) a3(2)],[-a3(3) a3(3)],'w')
plot3([-a3(1) a3(1)],[-a3(2) a3(2)],[-a3(3) a3(3)],'g')

echo off
p=input('To continue press [RETURN]');
echo

% eigenvalues and eigenvectors at the non-zero points

[V L]=eig([-sigma sigma 0; r-z -1 -x; y x -b])

% a stable direction and a '2d unstable focus'

echo off
p=input('look close to one of these fixed points (press [RETURN])');
echo

% draw local direction field near one of these fixed points

[x0,y0,z0]=meshgrid(x-2:1:x+2,y-2:1:y+2,z-2:1:z+2);
xd=sigma.*(-x0+y0);
yd=r.*x0-y0-x0.*z0;
zd=-b.*z0+x0.*y0;
quiver3(x0,y0,z0,xd,yd,zd)
xlim([x-3 x+3]);
ylim([y-3 y+3]);
zlim([z-3 z+3])

% draw some local axes to make picture easier to see

a1=3.5*V(:,1); a2=3.5*real(V(:,2)); a3=3.5*imag(V(:,2));
plot3([x-a1(1) x+a1(1)],[y-a1(2) y+a1(2)],[z-a1(3) z+a1(3)],'g')
```

```
plot3([x-a2(1) x+a2(1)],[y-a2(2) y+a2(2)],[z-a2(3) z+a2(3)],'r')
plot3([x-a3(1) x+a3(1)],[y-a3(2) y+a3(2)],[z-a3(3) z+a3(3)],'r')

% rotate to -85 44 to see ‘rotational part’

echo off

% draw local direction field near the other one of these fixed points

[x0,y0,z0]=meshgrid(-x-2:1:-x+2,-y-2:1:-y+2,z-2:1:z+2);
xd=sigma.*(-x0+y0);
yd=r.*x0-y0-x0.*z0;
zd=-b.*z0+x0.*y0;
quiver3(x0,y0,z0,xd,yd,zd)

% draw some local axes to make picture easier to see

plot3([-x-a1(1) -x+a1(1)],[-y-a1(2) -y+a1(2)],[z-a1(3) z+a1(3)],'g')
plot3([-x-a2(1) -x+a2(1)],[-y-a2(2) -y+a2(2)],[z-a2(3) z+a2(3)],'r')
plot3([-x-a3(1) -x+a3(1)],[-y-a3(2) -y+a3(2)],[z-a3(3) z+a3(3)],'r')

p=input('look at whole picture again (press [RETURN])');

echo

% and "join the dots"?
% show whole picture

xlim([-35 35]); ylim([-35 35]); zlim([-2 35])

% oh dear -- try looking at components of solutions
% as functions of time, and also the ‘bounding function’
% V(x,y,z)=x^2+y^2+(z-sigma-r)^2

echo off
p=input('To continue press [RETURN]');

figure(2)

tt=1; cla

yzero=[0;1;0];

while yzero~=999

   if tt==1; echo; end

   tspan=[0 50];    % length of time
   yzero=input('initial condition (input 999 to continue):    ');
   if yzero~=999
```

```
      [t,y]=ode45('lorenzde',tspan,yzero);
      figure(2)
      subplot 411
      plot(t,y(:,1)); ylabel('x')
      subplot 412
      plot(t,y(:,2)); ylabel('y')
      subplot 413
      plot(t,y(:,3)); ylabel('z')
      V=sqrt(y(:,1).^2+y(:,2).^2+(y(:,3)-sigma-r).^2);
      subplot 414
      plot(t,V); ylabel('V(x,y,z)^{1/2}')
   end

   if tt==1; echo off; end
   tt=tt+1;


end

echo

% this is no good -- look at trajectories moving in R^3

echo off

figure(1)

yzero=[0;1;0];

while yzero~=999

   if tt==1; echo; end

   tspan=[0 50];    % length of time
   yzero=input('initial condition (input 999 to continue):    ');
   if yzero~=999
      [t,y]=ode45('lorenzde',tspan,yzero);
      comet3(y(:,1),y(:,2),y(:,3))
   end

   if tt==1; echo off; end
   tt=tt+1;

end

echo

% still looks complicated... try plotting successive maximum
% values of z against each other...
```

```
echo off

p=input('To continue press [RETURN]');


yzero=[0 1 0]
figure
hold on

for k=1:25

tspan=[0 20]; [t,y]=ode45('lorenzde',tspan,yzero);

j=0;
for i=2:length(y)-1
   if y(i,3)>y(i+1,3) & y(i,3)>y(i-1,3)
      j=j+1;
      m(j)=y(i,3);
   end
end

for i=1:j-1
   plot(m(i),m(i+1),'x')
end

yzero=y(940,:);

end
```

lotkade.m

This final contains a general form of the Lotka-Volterra equations

$$\begin{aligned}\dot{x} &= x(A + ax + by)\\ \dot{y} &= y(B + cx + dy)\end{aligned}$$

for use in `lotkaplane.m` (via the `ode45` command). The inclusion of the extra parameter `sn` enables `lotkaplane.m` to integrate the trajectory both forwards (if `sn`$= 1$) and backwards (if `sn`$= -1$) in time.

```
function yprime=lotkade(t,y,flag,sn,A,B,a,b,c,d)
yprime=sn*[y(1)*(A+a*y(1)+b*y(2)); y(2)*(B+c*y(1)+d*y(2))];
```

lotkaplane.m

This program integrates the Lotka-Volterra model (above) for a given choice of the parameters, and then plots the trajectory in the phase plane. It integrates both forwards and backwards from the initial condition for a time

T, and as a default places an arrow indicating the direction of movement at the initial condition. However, there are other options: if $T < 0$ then the equations are integrated for a time $|T|$ but no arrow is used. If T has an imaginary part (e.g. $10 + \mathrm{i}$) then the solution is integrated for a time $\mathrm{Re}(T)$ and drawn as a thicker line (this is useful for highlighting stable manifolds). If a complex initial condition is entered then the program will continue integrating from the end of the previous trajectory, while it will finish when the initial condition is '`999`'.

The program uses `lotkade.m` (which contains the equations) and `darrow.m` (a routine to add arrows to the trajectories).

```
%%  lotkaplane.m
%%  draw phase plane diagram of the
%%  Lotka-Volterra model of two species
%%  Requires lotakde.m & darrow.m

hold on

%%  input parameters

A=input('A = ');
a=input('a = ');
b=input('b = ');
B=input('B = ');
c=input('c = ');
d=input('d = ');

X0=0;

%% mX is largest value of x and y in the phase diagram
%% change this number if the stationary points lie
%% outside 0<=x,y<=5

mX=5;

%% X0 is initial condition
%% if X0=999 then program finishes
%% if X0 has non-zero imaginary part then the integration
%%    continues from the end of the previous trajectory

while X0~=999

   X0=input('Initial condition:   ');
   if imag(X0)~=0
      X0=[u(2001,1) u(2001,2)];
   end

%% T is maximum time
%% if T is negative there is no arrow
%% if T has non-zero imaginary part the trajectory is
%%    drawn in bold

   T=input('Maximum time:   ');
   if X0~=999

      RX=X0;

%%  trajectory is calculated both forwards and backwards from X0

      for k = 1:2,

         X0=RX;

         sn=(-1)^k;

         rT=2*abs(real(T)); rN=4001; u(1,1)=mX*2; u(2,2)=mX*2;

%%  make sure that trajectory doesn't get too large

         while (max([max(u(:,1)) max(u(:,2))])>mX)

            rT=rT/2; rN=(rN-1)/2+1;

            tspan=linspace(0,rT,rN);

            [t,u]=ode45('lotkade',tspan,X0,[],sn,A,B,a,b,c,d);

         end

         if imag(T)==0
            plot(u(:,1), u(:,2))
         else
            plot(u(:,1), u(:,2), 'LineWidth',2)
         end


      end

%%  put the arrow on the trajectory (unless T is imaginary)

      D=lotkade(0,X0,[],1,A,B,a,b,c,d);

      if real(T)>0; darrow(X0(1), X0(2), D(1), D(2), 0.05); end

   end
end

hold off
```

`lportrait.m`

This lengthy file draws phase portraits for linear equations

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \mathbf{x}.$$

Once the parameters are fixed it calculates and displays the eigenvalues and eigenvectors of the matrix, and then (if the eigenvalues are not complex) these eigenvectors, labelling them with arrows. If the eigenvalues have the same sign then the eigenvector corresponding to the eigenvalue of larger modulus has two arrows. The figure window then shows the region $-1 \leq x, y \leq 1$. The program needs the files `linearde.m` and `darrow.m`

```
%%   lportrait.m
%%   phase portraits for linear equations
%%   Requires linearde.m & darrow.m

mng=5;  % maximum number of integrations from each IC

a=input('a=');
b=input('b=');
c=input('c=');
d=input('d=');

ew=1;   % width of eigenvectors

%%  calculate eigenvalues and eigenvectors

[V,D]=eig([a b;c d])

[x,y]=meshgrid(-1:.25:1,-1:.25:1);
DxDt=a*x+b*y; DyDt=c*x+d*y;

hold on

%%  if the eigenvalues are real then draw the eigenvectors

if imag(D(1))==0

   x1=linspace(-1,1,101)*V(1,1);
   y1=linspace(-1,1,101)*V(2,1);
   plot(x1,y1,'LineWidth',ew)
   x2=linspace(-1,1,101)*V(1,2);
   y2=linspace(-1,1,101)*V(2,2);
   plot(x2,y2,'LineWidth',ew)

%% draw arrows on eigenvectors

   s=0.05;
   xo=.5*V(1,1); yo=.5*V(2,1); nx=D(1,1)*xo; ny=D(1,1)*yo;
   darrow(xo,yo,nx,ny,s)
   xo=-xo; yo=-yo; nx=-nx; ny=-ny;
   darrow(xo,yo,nx,ny,s)
   xo=.5*V(1,2); yo=.5*V(2,2); nx=D(2,2)*xo; ny=D(2,2)*yo;
   darrow(xo,yo,nx,ny,s)
   xo=-xo; yo=-yo; nx=-nx; ny=-ny;
   darrow(xo,yo,nx,ny,s)
   if D(1,1)*D(2,2)>0
      if abs(D(1,1))>abs(D(2,2))
         xo=.55*V(1,1);yo=.55*V(2,1); nx=D(1,1)*xo; ny=D(1,1)*yo;
         darrow(xo,yo,nx,ny,s)
         xo=-xo; yo=-yo; nx=-nx; ny=-ny;
         darrow(xo,yo,nx,ny,s)
      else
         xo=.55*V(1,2); yo=.55*V(2,2); nx=D(2,2)*xo; ny=D(2,2)*yo;
         darrow(xo,yo,nx,ny,s)
         xo=-xo; yo=-yo; nx=-nx; ny=-ny;
         darrow(xo,yo,nx,ny,s)
      end
   end

end

axis equal

%%   now draw trajectories

tspan=linspace(0,1,101);

X0=0;

%%   enter '999' for initial condition to stop

while X0~=999
   X0=input('Initial condition:   ');

%%   trajectories starting at both X0 and -X0 are drawn,
%%   both forwards and backwards in time

   if X0~=999
      for yp=1:2;
          xm=(-1)^yp; ym=(-1)^yp;
          Z0=[xm ym].*X0;

          dir=linearde(0,Z0,[],a,b,c,d);

          xo=Z0(1); yo=Z0(2); nx=dir(1); ny=dir(2); s=.05;
          darrow(xo,yo,nx,ny,s)
          for p=1:2;
              m=(-1)^p;

              j=102; Y0=Z0; gn=0;

              while j==102 & gn<mng

                [t,u]=ode45('linearde',tspan,Y0,[],a*m,b*m,c*m,d*m);
                u(102,:)=[0 0];

                j=2;

                while abs(u(j,1))<=1 & abs(u(j,2))<=1 & ...
                (abs(u(j,1))+abs(u(j,2)))>=0.001 & j<=101
                    plot([u(j-1,1); u(j,1)],[u(j-1,2); u(j,2)])
                    j=j+1;
                end

                Y0=u(101,:); gn=gn+1;

              end

           end

       end

   end
end

axis off
axis equal
xlim([-1 1])
ylim([-1 1])
```

makematrix.m

The program uses the results of Exercises 28.5, 29.5, and 30.4 to construct matrices with specified eigenvalues and eigenvectors. After inputting the first eigenvector and eigenvalue, the program only asks for a second if they are real. Otherwise it takes the second eigenvalue and eigenvector to be the complex conjugate of the first. The matrix $\mathbb{A}$ is constructed as follows:

- if the two eigenvalues λ_1 and λ_2 are real and distinct then

$$\mathbb{A} = \mathbb{P}\begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix}\mathbb{P}^{-1},$$

where

$$\mathbb{P} = [\mathbf{v}_1 \quad \mathbf{v}_2]; \tag{M1.1}$$

- if the eigenvalues and eigenvectors are imaginary, with $\lambda_\pm = \rho \pm \mathrm{i}\omega$ and $\boldsymbol{\eta}_\pm = \mathbf{v}_1 \pm \mathrm{i}\mathbf{v}_2$ then

  $$\mathbb{A} = \mathbb{P}\begin{pmatrix} \rho & \omega \\ -\omega & \rho \end{pmatrix}\mathbb{P}^{-1},$$

  where $\mathbb{P}$ is as in (M1.1);
- if the eigenvalues are the same then the first eigenvector is taken as *the* eigenvector, and the second 'eigenvector' is taken to be the $\mathbf{v}_2$ used in the coordinate transformation in Section 30.2. In this case

  $$\mathbb{A} = \mathbb{P}\begin{pmatrix} \lambda & 1 \\ 0 & \lambda \end{pmatrix}\mathbb{P}^{-1},$$

  with $\mathbb{P}$ again given by (M1.1).

```
%% makematrix.m
%% construct matrix with specified
%% eigenvalues & eigenvectors

l1=input('First eigenvalue = ');
v1=input('First eigenvector = '); v1=conj(v1');

if imag(l1)~=0
   v2=imag(v1); v1=real(v1);
   M=[real(l1) imag(l1); -imag(l1) real(l1)];
else
   l2=input('Second eigenvalue = ');
   v2=input('Second eigenvector = '); v2=v2';
   M=[l1 0; 0 l2];
   if l1==l2 & norm(v1-v2)~=0;
      M=[l1 1; 0 l1];
   end

end

P=[v1, v2];

A=P*M*inv(P)

[V D]=eig(A)
```

`newtonde.m`

This file contains a variety of 'Newtonian equations' for use in `newtonplane.m` (via the `ode45` routine). As in `lotkade.m`, the variable `s` allows integration both forwards and backwards in time.

```
function yprime=newtonde(t,y,flag,s,k)

%% Comment out unwanted equations

%% Particles moving in the following potentials:

%%      V(x) = x-x^3/3
yprime=s.*[y(2); -k*y(2)-1+y(1)^2];

%%      V(x) = x^2/2
%% yprime=s.*[y(2); -k*y(2)-y(1)];

%%      V(x) = x^4/2-x^2
%% yprime=s.*[y(2); -k*y(2)-2*y(1)^3+2*y(1)];

%%      V(x) = x^6/6 - 5x^4/4 + 2x^2
%% yprime=s.*[y(2); -k*y(2)-y(1)^5+5*y(1)^3-4*y(1)];

%% Particles moving on a wire in the following shapes

%%      V(x) = x-x^3/3
%% yprime=s.*[y(2); -k*y(2)-(1-y(1)^2)*(1-2*y(1)*y(2)^2)/(2-2*y(1)^2+y(1)^4)];

%%      V(x) = x^2/2
%% yprime=s.*[y(2); -k*y(2)-y(1)*(1+y(2)^2)/(1+y(1)^2)];

%%      V(x) = x^3/2-x^2
%% n=2*y(1)-2*y(1)^3-(2*y(1)^3-y(1))*(6*y(1)^2-1)*y(2)^2;
%% yprime=s.*[y(2); -k*y(2)+n/(1+(2*y(1)^3-y(1))^2)];
```

`newtonplane.m`

Another phase plane program, this time using the file `newtonde.m` to provide the equation. It also allows a damping term $-k\dot{x}$ to be added (choose $k = 0$ for no damping). The program has the same options as `lotkaplane.m`, and also requires `darrow.m`

```
%%  newtonplane.m
%%  draw phase plane diagrams in whole plane
%%  requires newtonde.m & darrow.m

clf
hold on
```

```
dmp=input('damping k = ');

mX=5;

X0=0;

while X0~=999
   X0=input('Initial condition:   ');
   if imag(X0)~=0
      X0=[u(2001,1) u(2001,2)];
   end
   T=input('Maximum time:   ');
   if X0~=999
 %     tspan=linspace(0,T,2001);

      RX=X0;

      for k = 1:2,

         X0=RX;

         sn=(-1)^k;

         rT=2*abs(real(T)); rN=4001; u(1,1)=mX*2; u(2,2)=mX*2;

         while (max([max(u(:,1)) max(u(:,2))])>mX)

            rT=rT/2; rN=(rN-1)/2+1;

            tspan=linspace(0,rT,rN);

            [t,u]=ode45('newtonde',tspan,X0,[],sn,dmp);

         end

            if imag(T)==0
             plot(u(:,1), u(:,2))
          else
             plot(u(:,1), u(:,2), 'linewidth', 2)
          end


      end

      D=newtonde(0,X0,[],1,dmp);

      if real(T)>0; darrow(X0(1), X0(2), D(1), D(2), .1); end
   end
end
```

`pendde.m`

This file has the pendulum equation $\ddot{x} = -\sin x$ written in the coupled form

$$\begin{aligned} \dot{x} &= y \\ \dot{y} &= -\sin x \end{aligned}$$

for use in `piphase.m` and `cylinder.m` (via the `ode45` routine).

```
function yprime=pendde(t,y,flag,s)
yprime=s*[y(2); -sin(y(1))];
```

`piphase.m`

This phase plane program draws the direction field for the nonlinear pendulum on the phase space between $-\pi$ and π, and then animated trajectories on this restricted region of the phase space: if trajectories leave this region to one side then they will come back on the other. The program requires `pendde.m` to run.

```
%%  piphase.m
%%  phase portrait for the nonlinear pendulum
%%  on -pi<=x<=pi

hold off

[x,y]=meshgrid(-pi:.5:pi,-3:.5:3);
DxDt=y; DyDt=-sin(x);
quiver(x,y,DxDt,DyDt)
hold on

plot([-pi -pi],[-4 4],'--')
plot([pi pi],[-4 4],'--')

X0=0;

while X0~=999
   X0=input('Initial condition  [999 to end]:   ');
   if imag(X0)~=0
      X0=[u(2001,1) u(2001,2)];
   end
   if X0~=999
      T=input('Maximum time:   ');

      tspan=linspace(0,T,2001);
      [t,u]=ode45('pendde',tspan,X0,[],1);

      js=1; je=1;

      while je<2001
         je=je+1;
         if u(je,1)>pi
            comet(u(js:je,1),u(js:je,2))
            for j=je:2001
               u(j,1)=u(j,1)-2*pi;
            end
            js=je;
         end
         if u(je,1)<-pi
            comet(u(js:je,1),u(js:je,2))
            for j=je:2001
               u(j,1)=u(j,1)+2*pi;
            end
            js=je;
         end

      end

      if js~=2001;
         comet(u(js:je,1),u(js:je,2))
      end

   end

end

hold off
```

solvem.m

This program integrates $(d+1)^3$ copies of the Lorenz equations on a collection of initial conditions starting in a very small ball (one pixel) near $(0,1,0)$. The output can then be viewed using `flies.m`. Watching the evolution of 6^3 solutions will probably push the capabilities of most systems. The program needs `lorenzde.m` in order to work.

```
%%  solvem.m
%%  solve d^3 copies of the Lorenz equations
%%  starting in a small ball
%%  Requires lorenzde.m

clf
clear
```

```
d=input('d = ');

for i=0:1999;
   tspan(i+1)=i*35/1999;
end

for i=0:d;
   for j=0:d;
      for k=0:d;
         m=1+k+(j*(d+1))+(i*(d+1)^2)
         y0(:,m)=[(i-4.5)/100000; 1+(j-4.5)/100000; (k-4.5)/100000];
         [t,y]=ode45('lorenzde',tspan,y0(:,m));
         yi(2000,3,m)=1;
         yi(:,:,m)=y;
      end
   end
end
```

2

MATLAB files used in the exercise solutions

`backeuler.m`

This program implements the backwards Euler method

$$x_{n+1} = x_n + f(x_{n+1})$$

using the iterative method

$$y_{j+1} = x_n + f(y_j) \qquad y_0 = x_n \tag{M2.1}$$

in order to find a good approximation (y_j with j large) to x_{n+1} given x_n. It is applied here to the equation $\dot{x} = x(1-x)$ (i.e. with $f(x) = x(1-x)$) as in Exercise 21.7. You need to be careful to make sure that h is small enough that the iterative method in (M2.1) converges, see Exercise 21.5.

```
%% backwards Euler method

T=8;        %% final time

h=0.5;      %% timestep

%% MATLAB does not allow an index 0
%% on a vector, so x_n is x(n+1) here

t(1)=0;     %% initial time
x(1)=.5;    %% initial condition

for n=1:T/h;

t(n+1)=n*h;

%% Use iterative method to find x(n+1) given x(n)

gn=x(n);
g=gn+2*h^3;
```

```
while abs(gn-g)>h^3;

%% method is O(h) so approximate x(n+1) to within O(h^3)

g=gn;
gn=x(n)+h*g*(1-g);

end

x(n+1)=gn;

end

%% Plot crosses at numerical values, and join these

plot(t,x,'x','MarkerSize',20)
hold on
plot(t,x)
```

`exint.m`

This is the solution to Exercise 4.1(iii). It draws a graph of

$$T(t) = \int_0^t \mathrm{e}^{-(t-s)} \sin s \, \mathrm{d}s$$

against t for $0 \le t \le 7$.

```
%% Solution of Exercise 4.1(iii)

f=inline('exp(-(t-s)).*sin(s)','t','s');

for j=0:100;
     t(j+1)=7*j/100;
     T(j+1)=quad(f,0,t(j+1),[],[],t(j+1));
end

plot(t,T)
```

`findk.m`

This program finds the solution of the equation

$$k = -\ln\left\{\frac{12(k^2+\omega^2) - 5k(k\cos 6\omega + \omega \sin 6\omega)}{17(k^2+\omega^2) - 5k(k\cos 5\omega + \omega\sin 5\omega)}\right\} \tag{M2.2}$$

which occurs in Exercise 9.5 by choosing an initial ‘guess’ $k_0 = 0$ and then setting

$$k_{n+1} = -\ln\left\{\frac{12(k_n^2+\omega^2) - 5k_n(k_n\cos 6\omega + \omega\sin 6\omega)}{17(k_n^2+\omega^2) - 5k_n(k_n\cos 5\omega + \omega\sin 5\omega)}\right\}.$$

If $k_{n+1} = k_n$ then this gives a solution of (M2.2). The program performs 20 iterates and ends up with a reasonable approximation.

```
%% find k for Exercise 9.5(ii)
%% first guess

k=0;

w=pi/12;

%% change N for more iterations and so
%% a more accurate value for k

N=20;

for i=1:N;

    dn=12-(5*k*(k*cos(6*w)+w*sin(6*w))/(k^2+w^2));
    nn=17-(5*k*(k*cos(5*w)+w*sin(5*w))/(k^2+w^2));

    x=dn/nn;

    nk=-log(x);

    k=nk

end
```

`findt0.m`

Also used for Exercise 9.5, this program finds the solution t of the equation

$$t = 7 + \frac{1}{k}\ln\left\{\frac{17(k^2+\omega^2) - 5k(k\cos 5\omega + \omega\sin 5\omega)}{34(k^2+\omega^2) - 5k(k\cos\omega(t-2) + \omega\sin\omega(t-2))}\right\}$$

by choosing an initial guess $t_0 = 2$ and then setting

$$t_{n+1} = 7 + \frac{1}{k}\ln\left\{\frac{17(k^2+\omega^2) - 5k(k\cos 5\omega + \omega\sin 5\omega)}{34(k^2+\omega^2) - 5k(k\cos\omega(t_n-2) + \omega\sin\omega(t_n-2))}\right\}.$$

Although the first line sets the value of k to the first four decimal places of that found by `findk.m`, you could erase this line and run `findt0.m` straight after `findk.m` for a more accurate value.

```
%% find time of death for Exercise 9.5(iv)

%% value of k from findk.m

k=0.3640;

%% initial guess

t0=2;

w=pi/12;

%% change N for more accurate value

N=20;

for i=1:N;

   dn=17-(5*k*(k*cos(5*w)+w*sin(5*w)))/(k^2+w^2);
   nn=34-(5*k*(k*cos(w*(t0-2))+w*sin(w*(t0-2))))/(k^2+w^2);

   x=dn/nn;

   nt=7+(log(x)/k);

   t0=nt

end
```

param.m

This is the solution to Exercise 4.1(v). It draws a number of curves given parametrically by

$$x(t) = B\mathrm{e}^{-t} + At\mathrm{e}^{-t} \qquad \text{and} \qquad y(t) = A\mathrm{e}^{-t},$$

for A and B taking integer values between -3 and 3.

```
%%  Solution to Exercise 4.1(v)

t=linspace(0,5);

hold on

for A=-3:3;
   for B=-3:3;
      x=B*exp(-t)+A.*t.*exp(-t);
      y=A*exp(-t);
      plot(x,y)
   end
```

```
end

hold off
```

renormalised.m

This program combines parts of `f2.m` and `logistic.m` in order to draw cobweb diagrams for the renormalised version of the logistic map (see Exercise 24.3 and its solution).

```
%% Draw cobweb diagrams for renormalised version of logistic map

hold off
r=input('r=:  ');
plot([0 1],[0 1])

nxl=1-(1/r);
g=inline('(r^2*x*(1-x).*(1-(r*x*(1-x))))-1+(1/r)','x','r');
xl=1-(1/r)+0.00001; xr=1;
d=1;
while d>0.0001
   m=(xl+xr)/2;
   if g(xl,r)*g(m,r)<0
      xr=m;
   else
      xl=m;
   end
   d=(xr-xl);
end
nxr=m;

xp=linspace(0,1,100);
xv=nxl+xp*(nxr-nxl);
yv=r.*xv.*(1-xv); yv=r.*yv.*(1-yv);
yp=(yv-nxl)/(nxr-nxl);

hold on

plot(xp,yp,'linewidth',2)

axis equal; xlim([0 1]); ylim([0 1]);


z=input('z_0=:  ');
n=1;
while n~=0
   n=input('number of iterations (0 to stop, .1 to clear, .2 to change IC):  ');
   if n==0.1
      hold off; plot(x,x); hold on; plot(xp,yp, 'LineWidth', 2); axis equal;
      xlim([0 1]); ylim([0 1])
   end
   if n==0.2
      z=input('new initial condition:  ');
   end
   if n~=0
      for j=1:abs(n)
         za=nxl+z*(nxr-nxl);
         nza=r*za*(1-za); nza=r*nza*(1-nza);
         nz=(nza-nxl)/(nxr-nxl);
         if n>0
         plot([z z],[z nz])
         ms=min(8,25*abs(nz-z));
         if z<nz plot(z,(z+nz)/2,'^','markersize',ms)
         else
            plot(z,(z+nz)/2,'v','markersize',ms)
         end
         plot([z nz],[nz nz])
         if z<nz plot((z+nz)/2,nz,'>','markersize',ms)
         else
            plot((z+nz)/2,nz,'<','markersize',ms)
         end
      end
      z=nz;
   end
end
end
```

rungekutta.m

This file implements the Runge-Kutta scheme

$$x_{n+1} = x_n + \frac{h}{6}(f_1 + 2f_2 + 2f_3 + f_4)$$

with $f_1, \ldots, f_4$ given by

$$\begin{aligned} f_1 &= f(x_n, t_n) \\ f_2 &= f(x_n + \tfrac{1}{2}hf_1, t_n + \tfrac{1}{2}h) \\ f_3 &= f(x_n + \tfrac{1}{2}hf_2, t_n + \tfrac{1}{2}h) \\ f_4 &= f(x_n + hf_3, t_n + h), \end{aligned}$$

when $f(x,t) = t - x^2$, as required by Exercise 21.8.

```
%% Runge-Kutta scheme

T=12;
h=0.5;
```

```
t(1)=0;     %% initial time
x(1)=0;     %% initial condition

for n=1:T/h;

   t(n+1)=n*h;

   f1=t(n)-x(n)^2;
   f2=(t(n)+(h/2))-(x(n)+(h*f1/2))^2;
   f3=(t(n)+(h/2))-(x(n)+(h*f2/2))^2;
   f4=(t(n)+h)-(x(n)+h*f3)^2;
   x(n+1)=x(n) + h * (f1+(2*f2)+(2*f3)+f4)/6;

end

[t x]       %% display values

%% Plot crosses at numerical values, and join these

plot(t,x,'x','MarkerSize',20)
hold on
plot(t,x)
```

`temperature.m`

This program, also for use with Exercise 9.5 (as are `findk.m` and `findt0.m`) will calculate the temperature at time `t2` given the temperature at a time `t1` with a specified value of the parameter `k`, given by the formula

$$\begin{aligned} T(t_2) &= \mu + ak\left[\frac{k}{k^2+\omega^2}\cos\omega(t_2-\phi) + \frac{\omega}{k^2+\omega}\sin\omega(t_2-\phi)\right] \\ &+ \left[T(t_1) - \mu - \frac{ak}{k^2+\omega^2}(k\cos\omega(t_1-\phi)+\omega\sin\omega(t_1-\phi))\right]\mathrm{e}^{-k(t_2-t_1)}. \end{aligned}$$

(which is the solution of Exercise 9.4). This can be used to check the value of k found in part (ii), and then that the initial temperature found in part (iv) is also correct.

```
%% Calculate temperature when given by solution of Exercise 9.4
%% input is k, t1 (initial time), and t2 (final time), and T(t1)

m=3; a=5; f=2; w=pi/12;

% k=0.3640; % this is value from Exercise 9.5(ii)
k=input('k = ');

t1=input('t1 = ');
t2=input('t2 = ');
```

```
Tt1=input('T(t1) = ');

u=k^2+w^2;

T=m+a*k*(k*cos(w*(t2-f))+w*sin(w*(t2-f)))/u;
T=T+((Tt1-m-(a*k*(k*cos(w*(t1-f))+w*sin(w*(t1-f)))/u))*exp(-k*(t2-t1)))
```