Chapter 1

Octave

1.1 Symbolic solution

Input : Octave01SymbolicSolution.m
Output: Octave01SymbolicSolution.txt

1.1.1 DE

$$y' + 3y = 0, \qquad y' + 3 = 0 \;\; y(0) = 1,$$
$$y'' + 3y = 0, \qquad y'' + 3y = t^2, \;\; y(-1) = 0, \;\; y'(-1) = 0,$$
$$y' = e^{-t^2}, \;\; y(0) = 1, \qquad y(t) = 1 + \int_0^t e^{-s^2}\, ds = 1 + \frac{\sqrt{\pi}}{2}\mathrm{Erf}(t),$$
$$t^2y'' + ty' + t^2y = 0, \qquad y(t) = c_1 J_0(t) + c_2 Y_0(t),$$
$$\frac{d}{dt}\begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = \begin{pmatrix} y(t) \\ -4x(t) \end{pmatrix}$$

1.1.2 Script

```
clear all;
# graphics_toolkit ("gnuplot");
# pkg install -forge symbolic;  # if you need to install the package
pkg load symbolic;

#symbolic solution of DE
syms x(t) y(t) s

display("Diff.Eq.: y'+3y=0, y(0)=1 ")
de = diff(y,t) + 3*y(t)
dsolve( de == 0 )
dsolve( de == 0, y(0) == 1 )

display("Diff.Eq.: y''+3y=t^2, y(-1)=0, y'(-1)=0 ")
de = diff(y,t,2)+ 3*y(t)
dsolve( de == 0 )
dsolve( de == t^2 , y(-1) == 0, diff(y,t)(-1) == 0 )

display(["Diff.Eq.: y'=e^(-t^2)   Solution can be expressed by the ",
"    erf error function, well known in statistics (normal distribution) "])
de = diff(y,t) - e^(-t^2)
dsolve( de == 0, y(0) == 1 )
1 + int( e^(-s^2), s, 0, t )

display(["Diff.Eq.:  t^2 y'' + t y' + t^2 y = 0  Solution can be ",
"expressed by Bessel functions besselj and bessely,",
"OCTAVE just finds its series approximation"])
de = t^2 *  diff(y,t,2) + t *  diff(y,t) + t^2 * y(t)
dsolve( de == 0 )
b=t^2*diff(besselj(0,t),t,2)+t*diff(besselj(0,t),t)+t^2*besselj(0,t)
simplify(b)

display("    Systems of DE are rarely solvable, dsolve works for linear ones")
ode_sys = [diff(x(t),t) == y(t);  diff(y(t),t) == -4*x(t)]
soln = dsolve (ode_sys)
soln{1}
soln{2}
```

1.1.3 Output

```
>> Octave01SymbolicSolution
de(t) = (symfun)
  3 y(t) + d/dt (y(t))

ans = (sym)
             -3 t
```

```
  y(t) = C1 e

ans = (sym)
          -3 t
  y(t) = e

Diff.Eq.: y''+3y=t^2, y(-1)=0, y'(-1)=0
de(t) = (symfun)

  3 y(t) + d^2/dt^2 (y(t))

ans = (sym) y(t) = C1 sin(sqrt(3)t) + C2 cos(sqrt(3)t)
```

$$y(t) = \frac{t^2}{3} + \left(\frac{2\sqrt{3}}{9}\cos\left(\sqrt{3}\right) + \frac{1}{9}\sin\left(\sqrt{3}\right)\right)\sin\left(\sqrt{3}t\right) + \left(-\frac{1}{9}\cos\left(\sqrt{3}\right) + \frac{2\sqrt{3}}{9}\sin\left(\sqrt{3}\right)\right)\cos\left(\sqrt{3}t\right) - \frac{2}{9}$$

```
Diff.Eq.: y'=e^(-t^2)   Solution can be expressed by the
    erf error function, well known in statistics (normal distribution)
de(t) = (symfun)
                   2
                 -t
  d/dt (y(t)) - e

ans = (sym)

  y(t) = sqrt(pi) erf(t)/2 +1

ans = (sym)

  y(t) = sqrt(pi) erf(t)/2 +1

Diff.Eq.:  t^2 y'' + t y' + t^2 y = 0  Solution can be
expressed by Bessel functions besselj and bessely,
OCTAVE just finds its series approximation

 de(t) = (symfun)

  t^2 y(t) + t^2 d^2/dt^2 (y(t)) + t d/dt (y(t))

ans = (sym)

  y(t)= C1 (t^4/64 - t^2/4 + 1) + O(t^6)
```

$$\mathrm{b} = (\mathrm{sym})$$

$$t^2\left(-\frac{J_0(t)}{2} + \frac{J_2(t)}{2}\right) + t^2 J_0(t) - tJ_1(t)$$

```
bb = (sym) 0

    Systems of DE are rarely solvable, dsolve works for this one
ode_sys = (sym 2x1 matrix)

  d/dt (x(t)) = y(t)

  d/dt (y(t)) = -4 x(t)

soln =
{
(sym) x(t) = C1 sin(2 t) + C2 cos(2 t)
(sym) y(t) = 2 C1 cos(2 t) - 2 C2 sin(2 t)
}
ans = (sym) x(t) = C1 sin(2 t) + C2 cos(2 t)
ans = (sym) y(t) = 2 C1 cos(2 t) -  C2 sin(2 t)
```

1.2 Numerical solution, plotting 1.

Input : Octave02NumericalSolutionPlotting01.m
Output: Octave02NumericalSolutionPlotting01.txt

1.2.1 DE

$$y' = f(t,y) = (y-1)(y-2)e^{-t}, \quad y(0) = 1.9.$$

Plots:

1. Vector field $(t,y) \to (1, f(t,y))$,
2. Solution of the DE with initial condition $y(0) = 1.9$,
3. Solutions of the DE with initial conditions $y(0) = 1 + i/10,\ i = 0, \ldots, 10$.

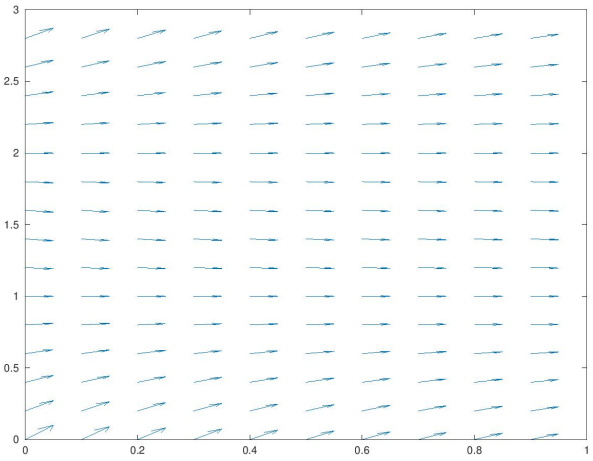
1.2.2 Script

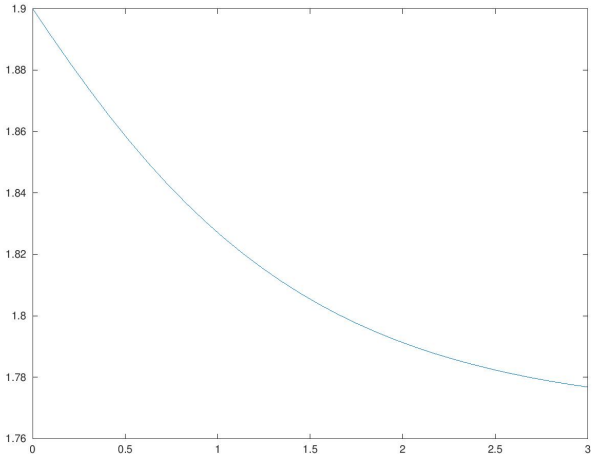
```
clear all;
graphics_toolkit ("gnuplot");
pkg load symbolic;
syms  y1(t) t

display("1. Diff.Eq.: y'=(y-1)(y-2)e^(-t), y(0)=1.9 ")

function ydot1 = f1 (y,t)
  ydot1=(y-1)*(y-2)*exp(-t);
endfunction

# symbolic solution
de1 = diff(y1,t)-f1(y1(t),t)
symsol1 = dsolve( de1==0 )

# the (1,f) vector field of the DE
figure(1)
yrange =  0:0.2:3;
trange =  0:0.1:1;
[tt,yy] = meshgrid(trange, yrange);
size = 0.05;
vtt = size*1;
vyy = size*exp(-tt) .* (yy.-1).*(yy.-2);  # observe the . dots !!!
quiver(trange, yrange, vtt, vyy);
axis([0 1 0 3]);
#  print -djpg fig1.jpg   # if you want to save the plot

# numerical solution, plot
ys1 = lsode ("f1", 1.9, (ts1 = linspace (0, 3, 30)'));
figure(2)
plot( ts1, ys1 )
display("1. Diff.Eq.: y'=(y-1)(y-2)e^(-t), y(0)=1.9, y(3)=???")
y3=ys1( length(ys1) )

# plot multiple solutions
figure(3)
for i=0:10
  ys1 = lsode ("f1", 1+i*0.1, (ts1 = linspace (0, 3, 30)'));
  hold on;
  plot( ts1, ys1 )
end
hold off;
```

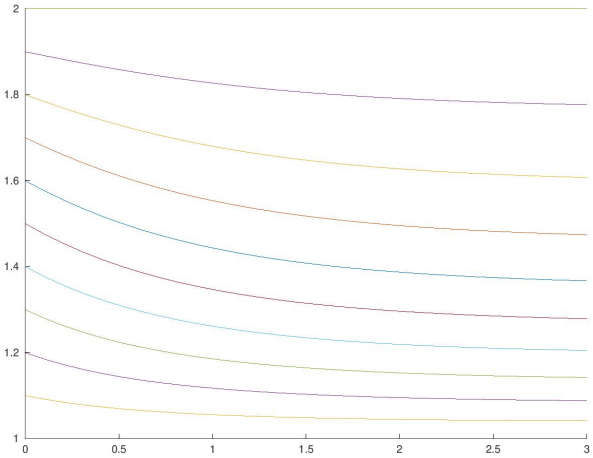
1.2.3 Output

```
>> Octave02NumericalSolutionPlotting01
1. Diff.Eq.: y'=(y-1)(y-2)e^(-t), y(0)=1.9
```

```
de1(t) = (symfun)
                                  -t
  - (y1 (t) - 2) (y (t) - 1)e    + d/dt (y1(t))

symsol1 = (sym)

  y1(t) =   ( C1 - 2 e^(e^(-t)) ) / ( C1 - e^(e^(-t)) )

1. Diff.Eq.: y'=(y-1)(y-2)e^(-t), y(0)=1.9, y(3)=???
y3 =  1.7768
```







1.3 Numerical solution, plotting 2.

Input : Octave02NumericalSolutionPlotting02.m
Output: Octave02NumericalSolutionPlotting02.txt

1.3.1 DE

$$\frac{d}{dt}\begin{pmatrix} y_1(t) \\ y_2(t) \end{pmatrix} = \begin{pmatrix} y_1 - y_1 y_2 \\ -y_2 + y_1 y_2 \end{pmatrix}.$$

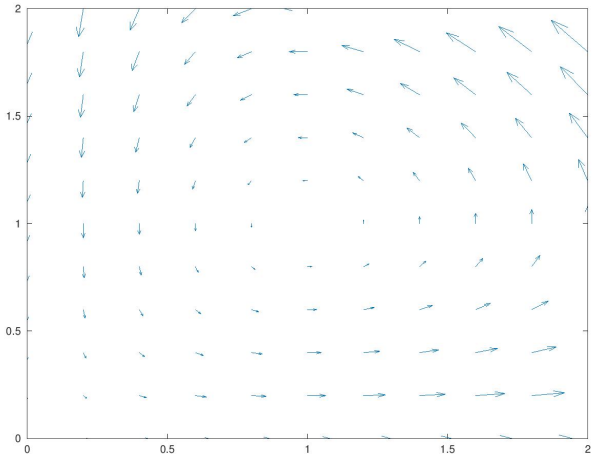
Plots:

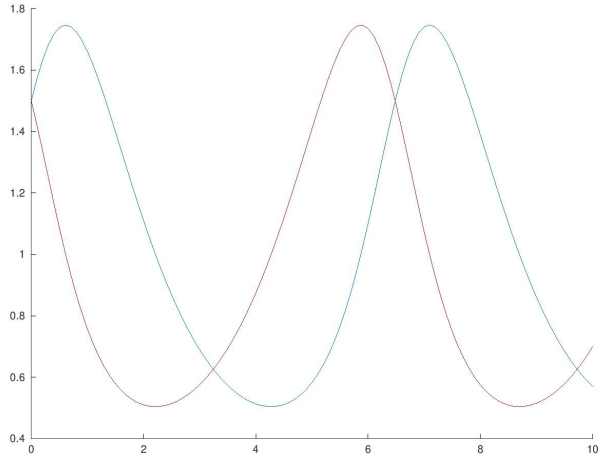
1. Vector field $(y_1, y_2)^T \to (y_1 - y_1 y_2,\ \ -y_2 + y_1 y_2)^T$,
2. $y_1(t)$ and $y_2(t)$ for the initial condition $\vec{y}(0) = (1.5,\, 1.5)^T$,
3. the trajectory $t \to (y_1(t),\, y_2(t))^T$ for the initial condition $\vec{y}(0) = (1.5,\, 1.5)^T$,
4. the trajectory $t \to (t,\, y_1(t),\, y_2(t))^T$ for the initial condition $\vec{y}(0) = (1.5,\, 1.5)^T$,
5. the trajectories $t \to (y_1(t),\, y_2(t))^T$ for the initial conditions $\vec{y}(0) = (1 + 0.1i,\, 1 + 0.1i)^T,\ \ i = 1, \ldots, 9$.

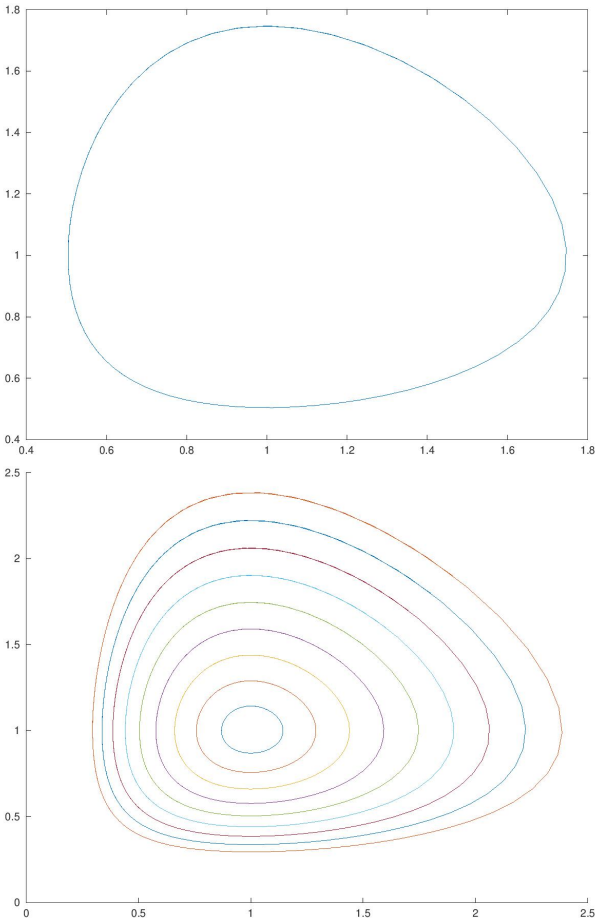
1.3.2 Script

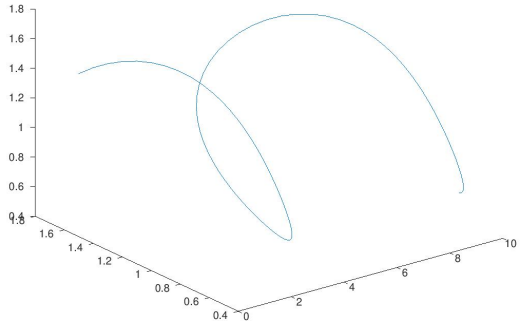
```
clear all;
graphics_toolkit ("gnuplot");

display("2. Lotka-Volterra (predator-prey) DE ")

function ydot = f (y,t)
  ydot(1) =  y(1)-y(1)*y(2);
  ydot(2) = -y(2)+y(1)*y(2);
endfunction

# the (f_1,f_2) vector field of the DE
figure(1)
y1range =  0:0.2:2;
y2range =  0:0.2:2;
[y1,y2] = meshgrid( y1range, y2range );
size = 0.05;
vy1 = size*( y1.-y1.*y2);
vy2 = size*(-y2.+y1.*y2);  # observe the . dots !!!
quiver( y1range,y2range,vy1,vy2 );
axis([0 2 0 2]);

# numerical solution, plots
ys = lsode ("f", [1.5,1.5], (ts = linspace (0, 10, 101)'));
figure(2)   # plots of t-->y1(t) and t-->y2(t)
hold on;
plot( ts, ys(:,1) )
plot( ts, ys(:,2) )
hold off;
figure(3)    # parametric plot y1(t)<-->y2(t)
plot( ys(:,1), ys(:,2) )
figure(4)    # 3 dim plot t--> [y1(t),y2(t)]
plot3( ts, ys(:,1), ys(:,2) )

# plot multiple solutions
figure(5)
hold on;
for i=1:9
  ys = lsode ("f", [1+i*0.1,1+i*0.1], (ts = linspace (0, 10, 101)'));
  plot( ys(:,1), ys(:,2) )
end
hold off;
```

1.3.3 Figures









1.4 Numerical derivation

Input : Octave03NumericalDerivation.m

1.4.1 Error estimates

$$error(\Delta x) = \left| f'(x) - \frac{f(x+\Delta x) - f(x)}{\Delta x} \right| \approx C \cdot \Delta x^1,$$

$$error(\Delta x) = \left| f'(x) - \frac{f(x+\Delta x) - f(x-\Delta x)}{2\Delta x} \right| \approx C \cdot \Delta x^2,$$

$$error(\Delta x) = \left| f'(x) - \frac{f(x+\Delta x) - f(x)}{\Delta x} \right| \approx C \cdot \Delta x^2 \qquad \text{if } x \text{ is an infection point.}$$

If $error(\Delta x) = C \cdot \Delta x^\alpha$, then $\ln(error(\Delta x)) = \ln(C) + \alpha \ln(\Delta x)$, so the plot of $\ln(\Delta x) \to \ln(error(\Delta x))$ is a straight line. Best fit can be found by linear regression, *polyfit* in Octave.

1.4.2 Script

```
clear all;
graphics_toolkit ("gnuplot");

min=4; max=15; dm=max-min;
for i = min : max
  dx(i-min+1) = 2.0^(-i);
end;

# sin'(1) = cos(1)
display(["sin'(1) = cos(1)",
"f->(f(x+dx)-f(x))/dx, error <--> dx^alpha"," "])
for i = 1:dm+1
  error(i) = abs( (sin(1.0+dx(i))-sin(1.0))/dx(i) - cos(1.0) );
end
```

```
pf = polyfit(log(dx),log(error),1)
alpha = pf(1)

# sin'(1) = cos(1)
display(["sin'(1) = cos(1)",
"f'->(f(x+dx)-f(x-dx)(2*d),"," error <--> dx^alpha"," "])
for i = 1:dm+1
  error(i) = abs( (sin(1.0+dx(i))-sin(1.0-dx(i)))/(2*dx(i)) - cos(1.0) );
end
pf = polyfit(log(dx),log(error),1)
alpha=pf(1)

# sin'(0) = cos(0)
display(["sin'(0) = cos(0)",
"f'->(f(x+dx)-f(x))/dx, error <--> dx^alpha"," "])
for i = 1:dm+1
  error(i) = abs( (sin(0.0+dx(i))-sin(0.0))/dx(i) - cos(0.0) );
end
pf = polyfit(log(dx),log(error),1)
alpha=pf(1)
```

1.4.3 Output

```
>> Octave03NumericalDerivation

sin'(1) = cos(1)
f->(f(x+dx)-f(x))/dx, error <--> dx^alpha
pf =

   1.00119  -0.85576

alpha =  1.0012
sin'(1) = cos(1)
f'->(f(x+dx)-f(x-dx)(2*d),
 error <--> dx^alpha
pf =

   2.0000  -2.4077

alpha =  2.0000
sin'(0) = cos(0)
f'->(f(x+dx)-f(x))/dx, error <--> dx^alpha
pf =

   2.0000  -1.7919

alpha =  2.0000
```

1.5 Errors of the Euler and Heun methods

Input : Octave03EulerHeunError.m

1.5.1 Euler and Heun methods

$$y'(t) = f(t,y), \quad y(t_0) = y_0, \qquad t_n = t_0 + n\Delta t, \quad y(t_n) \approx y_n.$$
$$error(\Delta t) = |y(T) - y_{T/\Delta t}| \sim \Delta t^{\alpha}.$$

Euler:

$$y_{n+1} = y_n + f(t_n, y_n)\Delta t.$$

Heun:

$$k_1 = f(t_n, y_n), \qquad k_2 = f(t + \Delta t, y_n + f(t_n, y_n)\Delta t),$$

$$y_{n+1} = y_n + \frac{1}{2}(k_1 + k_2)\Delta t.$$

We expect $\alpha = 1$ for Euler and $\alpha = 2$ for Heun.

If $error(\Delta x) = C \cdot \Delta x^{\alpha}$, then $\ln(error(\Delta x)) = \ln(C) + \alpha \ln(\Delta x)$, so the plot of $\ln(\Delta x) \to \ln(error(\Delta x))$ is a straight line. Best fit can be found by linear regression, *polyfit* in Octave.

1.5.2 Script

```
clear all;

min=9; max=15; dm=max-min;
for i = min : max
  dx(i-min+1) = 2.0^(-i);
end;

function ydot = f (y,t)
  ydot = t*y;
endfunction

global t0=0   y0=1   t1=3
ys = lsode ("f", y0, (ts1 = linspace (t0, t1, 100)'));
format long
ylsode = ys( length(ys) )
yexact = exp(3.0^2/2)
format short

function yeuler = euler (delta)
  global t0 y0 t1
  y=y0; t=t0;
  for i=1:(t1-t0)/delta
    y=y+f(y,t)*delta;
    t=t+delta;
  endfor;
  yeuler=y;
endfunction

for i = 1:dm+1
  error(i) = abs( yexact - euler( dx(i) ) );
end
display("Euler error");
pf = polyfit(log(dx),log(error),1);
alpha=pf(1)
display(dx)
display( error)

function yheun = heun (delta)
  global t0 y0 t1
  y=y0; t=t0;
  for i=1:(t1-t0)/delta
    k1=f(y,t);
    k2=f(y+k1*delta,t+delta);
    y=y+((k1+k2)/2)*delta;
    t=t+delta;
  endfor;
  yheun=y;
endfunction

for i = 1:dm+1
  error(i) = abs( yexact - heun( dx(i) ) );
end
display("Heun error");
pf = polyfit(log(dx),log(error),1);
alpha=pf(1)
display(error )
```

1.5.3 Output

```
>> Octave03EulerHeunError

ylsode =  90.0172578480779
yexact =  90.0171313005218

Euler error
alpha =  0.99816
dx =
   1.9531e-03   9.7656e-04   4.8828e-04   2.4414e-04   1.2207e-04   6.1035e-05   3.0518e-05

error =
   1.045688   0.525135   0.263144   0.131716   0.065894   0.032956   0.016480

Heun error
alpha =  1.9993
error =
   1.1551e-03   2.8926e-04   7.2374e-05   1.8101e-05   4.5262e-06   1.1317e-06   2.8293e-07
```

1.6 Taylor expansion of y(t+dt).

Input: Octave03Taylor.m

1.6.1 DE

$$\frac{d}{dt}y(t) = 5 + t \cdot y(t), \quad y(2) = 3.$$
$$y^{(n)}(t) = (\partial_t f + f \cdot \partial_y f)^{n-1} f$$

1.6.2 Script

```
clear all;
pkg load symbolic;
syms   y t y0 t0 c1 c2 c3 dt
y0 = sym(3)
t0 = sym(2)
f1 = sym(5) + t*y
f2 = diff( f1,t ) + f1 * diff( f1,y )
f3 = diff( f2,t ) + f1 * diff( f2,y )
c1 = subs( f1, {y,t},{y0,t0} )
c2 = subs( f2, {y,t},{y0,t0} )
c3 = subs( f3, {y,t},{y0,t0} )
taylor = y0 + c1*dt + c2*dt^2/factorial(sym(2)) + c3*dt^3/factorial(sym(3))
```

1.6.3 Output

```
y0 = (sym) 3
t0 = (sym) 2
f1 = (sym) t y + 5
f2 = (sym) t (t y + 5) + y
f3 = (sym) 2 t y + (t^2  + 1) (t y + 5) + 5
c1 = (sym) 11
c2 = (sym) 25
c3 = (sym) 72
taylor = (sym) 12 dt^3 +25 dt^2 / 2 + 11 dt + 3
```

1.7 Linearized DE

Input: Octave04Linearization.m

1.7.1 DE

$$\frac{d}{dt}\begin{pmatrix} y_1(t) \\ y_2(t) \end{pmatrix} = \begin{pmatrix} 2y_1 - 4y_1y_2 \\ -3y_2 + 2y_1y_2 \end{pmatrix}.$$

1.7.2 Script

```
clear all;
pkg load symbolic;
syms   y1 y2

S = solve( 2*y1-4*y1*y2 == 0,
          -3*y2+2*y1*y2 == 0, y1, y2)
jac = jacobian([2*y1-4*y1*y2 ; -3*y2+2*y1*y2], [y1;y2])

subs( jac, {y1 y2}, { sym(3)/sym(2) sym(1)/sym(2)} )
subs( jac, {y1 y2}, { S{1}.y1 S{1}.y2 } )
subs( jac, {y1 y2}, { S{2}.y1 S{2}.y2 } )

function ydot = f (y,t)
  ydot(1) =  2*y(1)-4*y(1)*y(2);
  ydot(2) = -3*y(2)+2*y(1)*y(2);
endfunction

[fsol1, info] = fsolve ("f", [0.1; 0.5])
[fsol2, info] = fsolve ("f", [5; 4])

jacf = function_handle(jac)
jacf(fsol1(1), fsol1(2))
jacf(fsol2(1), fsol2(2))
```

1.7.3 Output

```
>> Octave04Linearization
S =
{
  [1,1] =
    scalar structure containing the fields:
y1 = (sym) 0
y2 = (sym) 0

  [1,2] =
    scalar structure containing the fields:
y1 = (sym) 3/2
y2 = (sym) 1/2
}

jac = (sym 2x2 matrix)
  -4 y2 + 2   -4 y1

    2 y2     2 y1 - 3

ans = (sym 2x2 matrix)
  0  -6

  1  0

ans = (sym 2x2 matrix)
  2  0

  0  -3

ans = (sym 2x2 matrix)
  0  -6

  1  0

fsol1 =
   1.50000
   0.50000
```

```
info =
   1.4437e-07  -7.2187e-08

fsol2 =
   1.50000
   0.50000
info =
  -2.7756e-07   1.3878e-07

jacf =
@(y1, y2) [-4 * y2 + 2, -4 * y1; 2 * y2, 2 * y1 - 3]

ans =
   9.6250e-08  -6.0000e+00
   1.0000e+00  -1.4437e-07
ans =
  -1.8504e-07  -6.0000e+00
   1.0000e+00   2.7756e-07       (after minor editing)
```

1.8 Diagonalization, matrix exponential

Input: Octave05Diagonalization.m

1.8.1 Linear algebra

1. Solve $\det(A - \lambda E) = 0$.
2. Solve $(A - \lambda E)\vec{v}_k = \vec{0}$.
3. Set $S = [\vec{v}_1, \ldots, \vec{v}_n]$, $D = \mathrm{diag}(\lambda_1, \ldots . \lambda_n)$. Then $A = SDS^{-1}$.
4. $\exp(tA) = Se^{tD}S^{-1}$.
5. First column of $\exp(tA)$ solves $\vec{y}' = A\vec{y}$, $\vec{y}(0) = \vec{e}_1$, etc.

1.8.2 Script

```
clear all;
pkg load symbolic;
syms A lambda t y1(t) y2(t)

A=[4 -2; 1 1]
charpol = det(A-lambda*eye(2))
lambdas = solve( charpol == 0, lambda )
[evecs, evals] = eig(A)
SDSinv = evecs * evals * inv(evecs)           # should be A
dexp = evecs * expm(5*evals) * inv(evecs)     # exp(5*A) from lin.alg
matexp = expm( t*A )
matexp(:,1)                                   # first column of exp(tA)
dsolve( (evals * [y1(t) ; y2(t)])(1) - diff(y1(t))==0,
        (evals * [y1(t) ; y2(t)])(2) - diff(y2(t))==0,
         y1(0)==1, y2(0)==0 ) ;     #  should reproduce matexp(:,1),
                                    #  dsolve does not solve the DE
```

1.8.3 Output

```
>> Octave05Diagonalization
A =
   4  -2
   1   1

charpol = (sym) (-lambda + 1) (-lambda + 4) + 2
lambdas = (sym 2x1 matrix)
  2
  3
```

```
evecs =
   0.89443   0.70711
   0.44721   0.70711

evals =
Diagonal Matrix
   3   0
   0   2

SDSinv =
   4  -2
   1   1

dexp =
   6.5160e+06  -6.4940e+06
   3.2470e+06  -3.2250e+06

mexp = (sym 2x2 matrix)

     3t     2t           3t       2t
  2 e    - e        - 2 e    + 2 e

    3t     2t           3t       2t
   e    - e          - e    + 2 e

ans = (sym 241 matrix)

     3t     2t
  2 e   - e

    3t    2t
  e   - e
```

1.9 Matrix exponential, Jordan normal form

Input: Octave05Jordan.m

1.9.1 DE

Radioactive decay, spring-mass system.

$$\alpha \to \beta \to \emptyset, \qquad ||m_1 = 1|| \longleftarrow (k = 2) \longrightarrow ||m_2 = 1||$$

$$\frac{d}{dt}\vec{y} = A\vec{y}, \qquad A = \begin{pmatrix} 3 & 0 \\ 4 & 3 \end{pmatrix} \qquad \text{or} \qquad A = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -1/2 & 1/2 & 0 & 0 \\ 1/2 & -1/2 & 0 & 0 \end{pmatrix}$$

1.9.2 Script

```
clear all;
pkg load symbolic;
syms A lambda t

A = sym( [0    0   1 0;
          0    0   0 1;
         -1/2  1/2 0 0;
          1/2 -1/2 0 0] )   # generates a lot of warnings
[V, J] = jordan (A)
vjvi = V * J * inv(V)
etA = expm( t * A )
re7A = real( expm( sym(7)*A ) )
```

1.9.3 Output

```
>> Octave05Jordan
A = (sym 4x4 matrix)

   0     0    1  0

   0     0    0  1

  -1/2  1/2   0  0

  1/2   -1/2  0  0

V = (sym 4x4 matrix)

  1  0  -i  i

  1  0  i   -i

  0  1  -1  -1

  0  1  1   1

J = (sym 4x4 matrix)

  0  1  0   0

  0  0  0   0

  0  0  -i  0

  0  0  0   i

vjvi = (sym 4x4 matrix)

   0     0    1  0

   0     0    0  1

  -1/2  1/2   0  0

  1/2   -1/2  0  0
```

$$re7A = \begin{bmatrix} \frac{1}{2}\cos(7)+\frac{1}{2} & -\frac{1}{2}\cos(7)+\frac{1}{2} & \frac{1}{2}\sin(7)+\frac{7}{2} & -\frac{1}{2}\sin(7)+\frac{7}{2} \\ -\frac{1}{2}\cos(7)+\frac{1}{2} & \frac{1}{2}\cos(7)+\frac{1}{2} & -\frac{1}{2}\sin(7)+\frac{7}{2} & \frac{1}{2}\sin(7)+\frac{7}{2} \\ -\frac{1}{2}\sin(7) & \frac{1}{2}\sin(7) & \frac{1}{2}\cos(7)+\frac{1}{2} & -\frac{1}{2}\cos(7)+\frac{1}{2} \\ \frac{1}{2}\sin(7) & -\frac{1}{2}\sin(7) & -\frac{1}{2}\cos(7)+\frac{1}{2} & \frac{1}{2}\cos(7)+\frac{1}{2} \end{bmatrix}$$

1.10 Diagonalization, complex eigenvalues

1.10.1 DE

$$y'' = -y - 0.2\,y', \qquad \frac{d}{dt}\vec{y} = \begin{pmatrix} 0 & 1 \\ -1 & -0.2 \end{pmatrix}\vec{y}$$

Strategy:

- Use complex numbers and vectors.
- Or: Matrix is real $\Longrightarrow$ eigenvalues are either real or come in complex conjugate pairs. Then corresponding eigenvectors can be picked as complex conjugates. Choose one from each complex eigenvalue pair and use the real and imaginary parts of of the corresponding eigenvectors as basis vectors. In our case

$$A = \begin{pmatrix} 0 & 1 \\ -1 & -0.2 \end{pmatrix} = SDS^{-1}$$

$$= \begin{pmatrix} 0.707 + 0.i & 0.707 + 0.i \\ -0.0707 + 0.703i & -0.0707 - 0.703i \end{pmatrix} \begin{pmatrix} -0.1 + 0.994i & 0. \\ 0. & -0.1 - 0.994i \end{pmatrix} \begin{pmatrix} 0.707 + 0.i & 0.707 + 0.i \\ -0.0707 + 0.703i & -0.0707 - 0.703i \end{pmatrix}^{-1}$$

$$= TD_{real}T^{-1} = \begin{pmatrix} 0.707 & 0. \\ -0.0707 & 0.703 \end{pmatrix} \begin{pmatrix} -0.1 & 0.994 \\ -0.994 & -0.1 \end{pmatrix} \begin{pmatrix} 0.707 & 0. \\ -0.0707 & 0.703 \end{pmatrix}^{-1}$$

Furthermore

$$\exp(t \cdot D_{real}) = \exp\left[t \cdot \begin{pmatrix} -0.1 & 0.994 \\ -0.994 & -0.1 \end{pmatrix}\right] = \exp(-0.1 \cdot t) \begin{pmatrix} \cos(0.994\,t) & \sin(0.994\,t) \\ -\sin(0.994\,t) & \cos(0.994\,t) \end{pmatrix}$$

So $\exp(tA) = T \exp(t \cdot D_{real})\, T^{-1}$.

1.10.2 Code

```
clear all;
t=5
A=[0 1; -1 -0.2]
[evects, evals] =  eig(A)
SDSi = evects * evals * inv(evects)
etA = expm(t*A)
SetDSi = evects * expm(t*evals) * inv(evects)
lambda =evals(1,1)
B = [real(lambda) imag(lambda) ; -imag(lambda) real(lambda)]
etB = expm(t*B)
etBr = exp(t*real(lambda))*[cos(imag(lambda)*t) sin(imag(lambda)*t);
                           -sin(imag(lambda)*t) cos(imag(lambda)*t)]
T = [real(evects(:,1)) imag(evects(:,1))]
TBTi = T * B * inv(T)
TetBrTi = T * etBr * inv(T)
```

1.10.3 Output

```
>> Octave05KomplexDiagonalization
t =  5
A =
   0.00000   1.00000
  -1.00000  -0.20000

evects =
   0.70711 + 0.00000i   0.70711 - 0.00000i
  -0.07071 + 0.70356i  -0.07071 - 0.70356i

evals =
Diagonal Matrix
  -0.10000 + 0.99499i                   0
                    0  -0.10000 - 0.99499i

SDSi =
  -1.3878e-17   1.0000e+00
  -1.0000e+00  -2.0000e-01

etA =
```

```
   0.098551  -0.588697
   0.588697   0.216290

SetDSi =
   0.098551  -0.588697
   0.588697   0.216290

lambda = -0.10000 + 0.99499i
B =
  -0.10000   0.99499
  -0.99499  -0.10000

etB =
   0.15742  -0.58575
   0.58575   0.15742

etBr =
   0.15742  -0.58575
   0.58575   0.15742

T =
   0.70711   0.00000
  -0.07071   0.70356

TBTi =
  -1.3878e-17   1.0000e+00
  -1.0000e+00  -2.0000e-01

TetBrTi =
   0.098551  -0.588697
   0.588697   0.216290
```

1.11 State space portraits

Input: Octave06StateSpacePortraits.m

1.11.1 DE

$$\frac{d}{dt}\vec{y}(t) = A\vec{y}(t).$$

1.11.2 Script

```
clear all;
graphics_toolkit ("gnuplot");

global a b c d
a=0; b=1; c=-1; d=-0.4;  # underdamped oscillator
#a=0; b=1; c=-0.5; d=0;  # harminc oscillator
#a=-2; b=0; c=1; d=-4;   # stable node
#a=-2; b=1; c=2; d=3;    # saddle
#a=0; b=1; c=-1; d=-2;   # degenerate stable node

function ydot = f (y,t)
  global a b c d
  ydot(1) = a*y(1)+b*y(2);
  ydot(2) = c*y(1)+d*y(2);
endfunction

figure(1)
for i=0:9
  ys = lsode ("f",  1.4*[cos(6.28*i*0.1),sin(6.28*i*0.1)],
                    (ts = linspace (0, 10, 101)'));
  plot( ys(:,1) , ys(:,2) )
```

```
  if i == 1; hold on; end
  endfor
axis([-1 1 -1 1])
hold off;
```

1.11.3 Remarks

1. Underdamped oscillator.

$$A\vec{v} = (-0.2 + 0.97\,i)\vec{v}, \qquad \vec{v}_1 = Re(\vec{v}), \qquad T^{-1}AT = \begin{pmatrix} -0.2 & 0.97 \\ -0.97 & -0.2 \end{pmatrix}.$$

   $\vec{v} \to e^{i\alpha}\vec{v}$ rotates by α the $[\vec{v}_1, \vec{v}_2]$ coordinate system.

2. Harmonic oscillator.

$$A\vec{v} = 0.707\,i \cdot \vec{v}, \qquad \vec{v}_1 = Re(\vec{v}), \quad \vec{v}_2 = Im(\vec{v}).$$
$$T = [\vec{v}_1, \vec{v}_2], \qquad T^{-1}AT = \begin{pmatrix} 0 & 0.707 \\ -0.707 & 0 \end{pmatrix}.$$

   $\vec{v} \to e^{i\alpha}\vec{v}$ rotates by α the $[\vec{v}_1, \vec{v}_2]$ coordinate system.

3. Stable node. $\vec{v}_1, \vec{v}_2$ have fixed directions and arbitrary nonzero lengths.

4. Saddle. $\vec{v}_1, \vec{v}_2$ have fixed directions and arbitrary nonzero lengths.

5. Degenerate node. $\vec{v}_1$ has fixed direction and determines $\vec{v}_2$. In the $[\vec{v}_1, \vec{v}_2]$ coordinate system

$$J = [\vec{v}_1, \vec{v}_2]^{-1} A\,[\vec{v}_1, \vec{v}_2] = \begin{pmatrix} -1 & 1 \\ 0 & 1 \end{pmatrix}.$$

1.11.4 Output

Octave figure

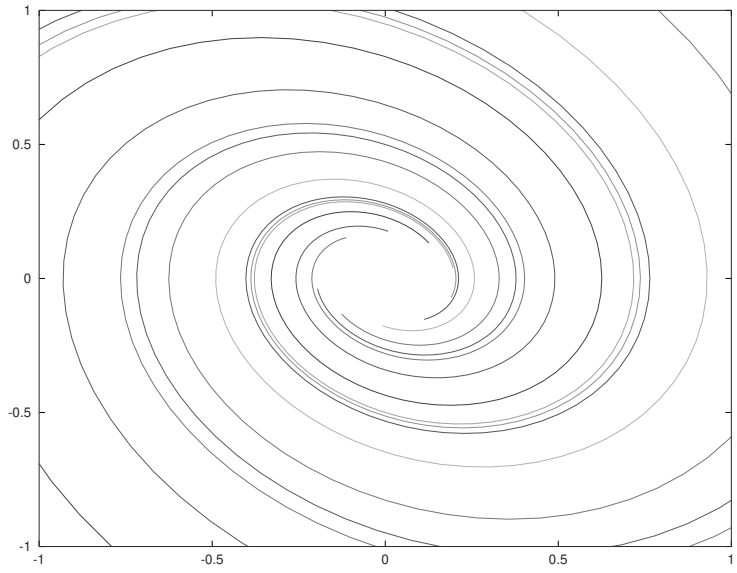


Figure 1.1: Underdamped oscillator, $\lambda_{1,2} = -0.2 \pm 0.97\,i$. Stable focus, spiral.

Better figures

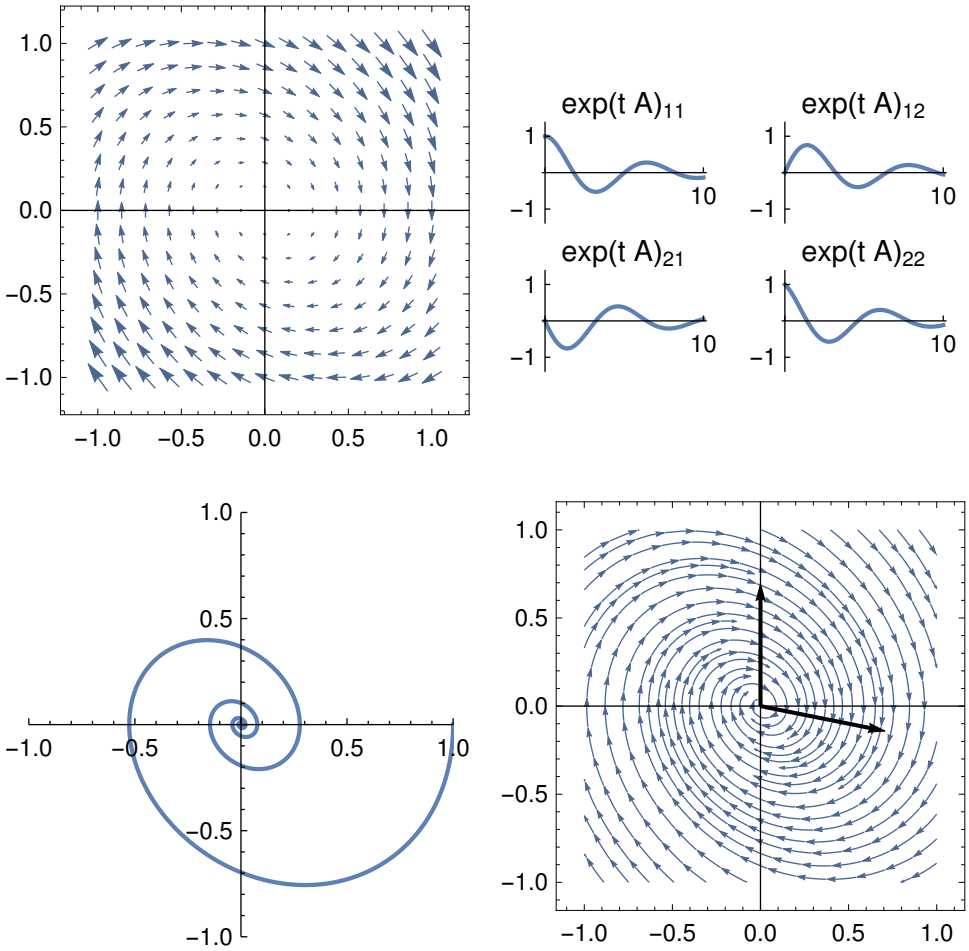


Figure 1.2: Underdamped oscillator, $\lambda_{1,2} = -0.2 \pm 0.97\,i$. Stable focus, spiral.

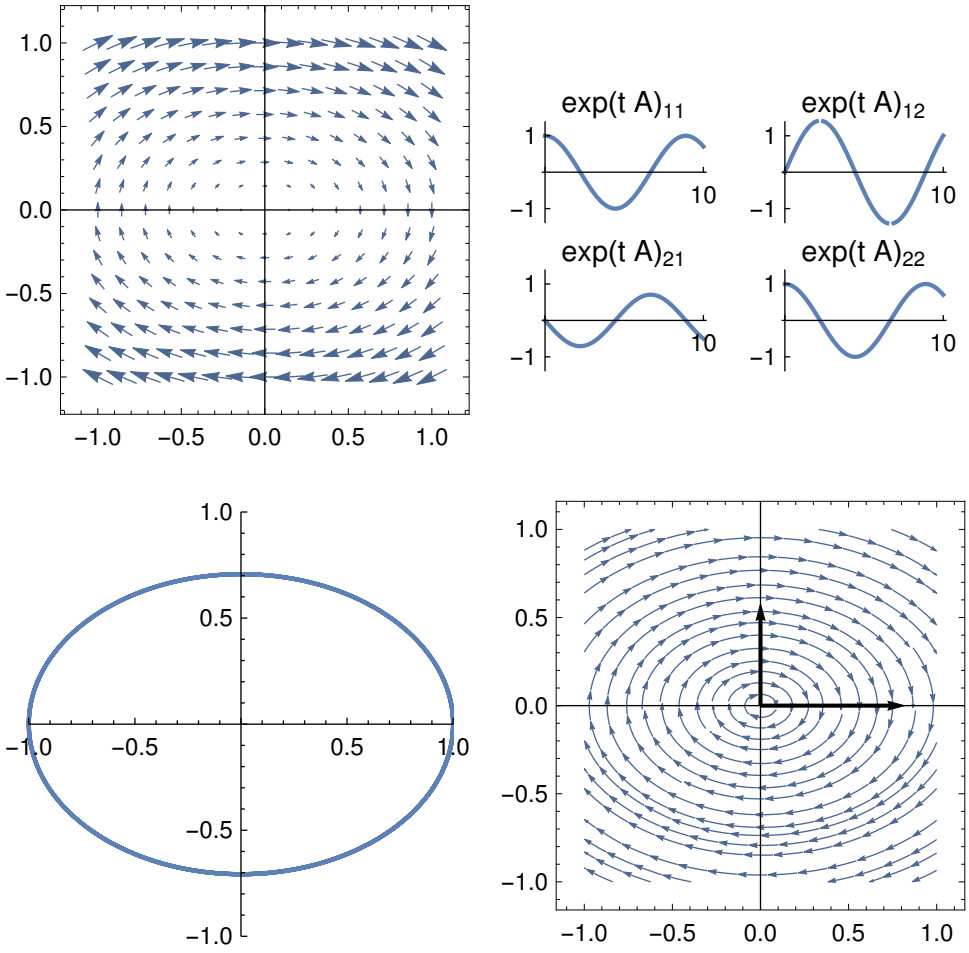


Figure 1.3: Harmonic oscillator, $\lambda_{1,2} = \pm 0.7071\, i$. Center.

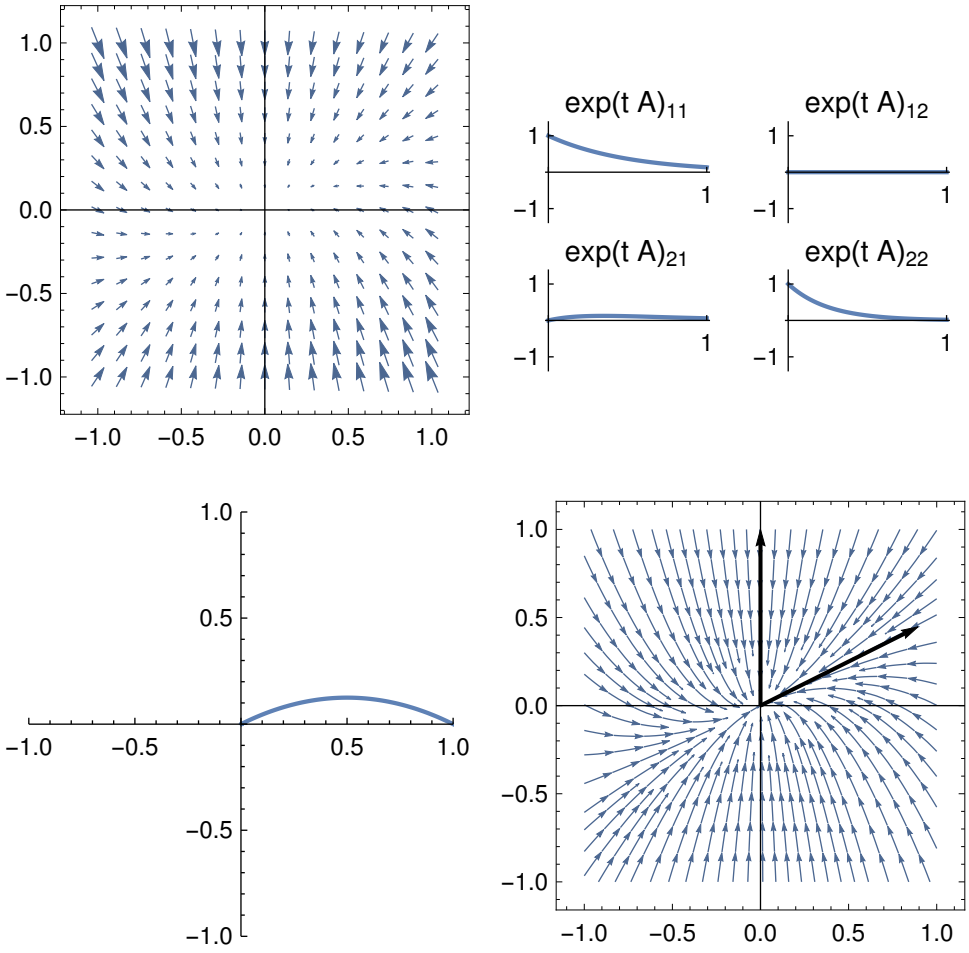


Figure 1.4: Drain, $\lambda_1 = -4, \lambda_2 = -2$. Stable node.

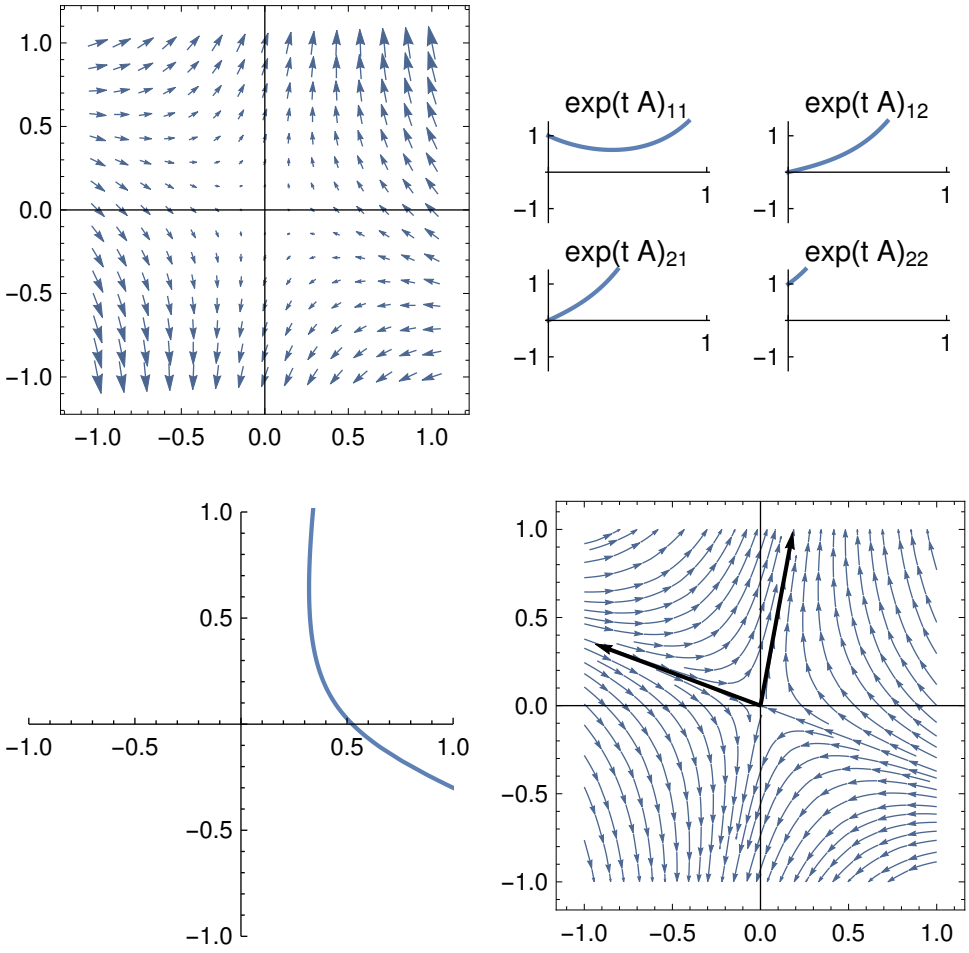


Figure 1.5: Saddle, $\lambda_1 = 3.372, \lambda_2 = -2.372$. Unstable.

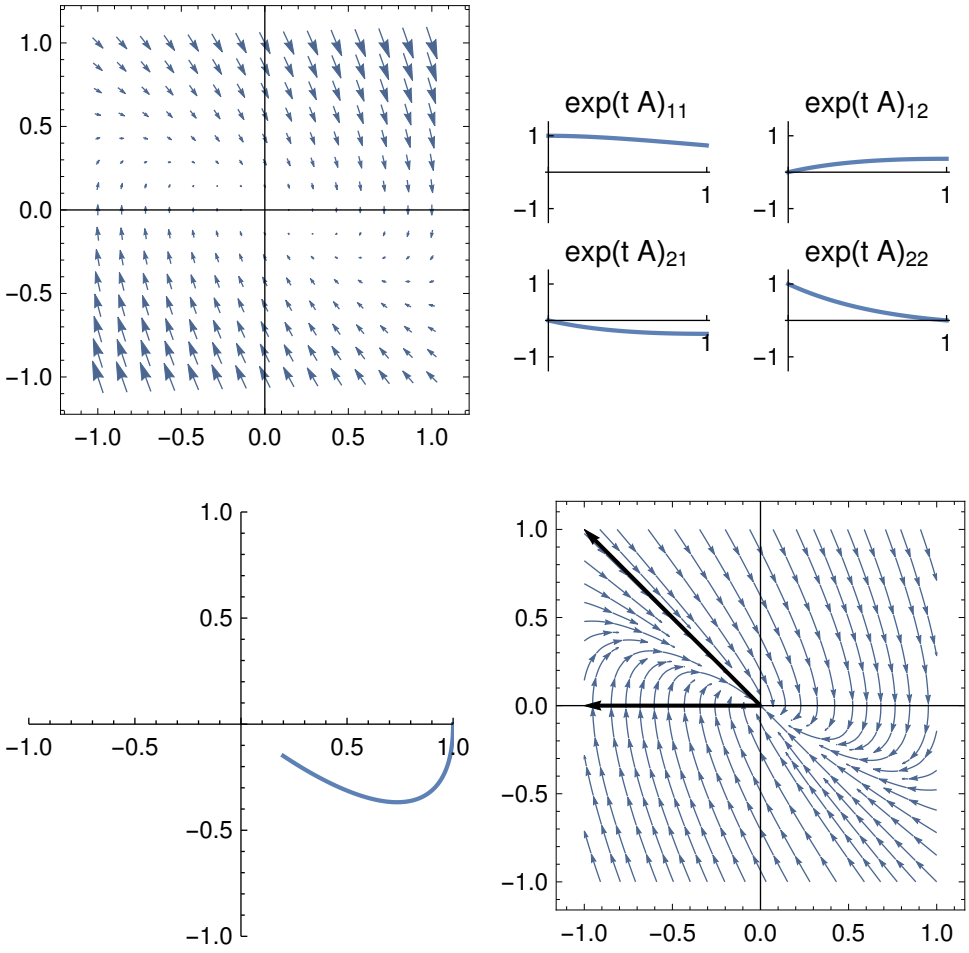


Figure 1.6: Critically damped oscillator, $\lambda_1 = -1$. Nontrivial Jordan decomposition. Stable degenerate node.

1.12 Rotation group $SO(3)$

1.12.1 Lie group, Lie algebra

- Lie group: Special orthogonal (rotation) matrices $SO(3) = \{O \in \mathrm{Mat}_3(\mathbb{R}) \mid \det O = 1,\ O^{-1} = O^T\}$.
- Lie algebra: $\mathfrak{so}(3) = \{A \in \mathrm{Mat}_3(\mathbb{R}) \mid A = -A^T\}$.
- Let

$$\vec{a} = (a_1, a_2, a_3), \qquad L(\vec{a}) = \begin{pmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{pmatrix} = A$$

  (and the same for the rest of the alphabet). Then

$$L(\vec{a} \times \vec{b}) = L(\vec{a})L(\vec{b}) - L(\vec{b})L(\vec{a}) = [L(\vec{a}), L(\vec{b})].$$

- $SO(3) = \exp(\mathfrak{so}(3))$. The rotation matrix around $\vec{a}$ by angle $|\vec{a}|$ is $R_a = \exp(L(\vec{a}))$.
- The product of rotation matrices is another rotation matrix, so

$$\exp(tL(\vec{a}))\exp(tL(\vec{b}))\exp(-tL(\vec{a}))\exp(-tL(\vec{b})) = \exp(t^2 L(\vec{c})),$$
$$L(\vec{c}) \approx L(\vec{a})L(\vec{b}) - L(\vec{b})L(\vec{a}) = [L(\vec{a}), L(\vec{b})].$$

- If

$$\vec{x} = (1,0,0), \quad \vec{y} = (0,1,0), \quad \text{then} \quad \vec{z} = \vec{x} \times \vec{y} = (0,0,1),$$

  so

$$\exp(tL(\vec{x}))\exp(tL(\vec{y}))\exp(-tL(\vec{x}))\exp(-tL(\vec{y})) \approx \exp(t^2 L(\vec{z})).$$

- Consequently by combining rotations around the x, y axes we can produce rotations around the z axis.

1.12.2 Code

```
clear all;
pkg load symbolic;
syms a a1 a2 a3 b1 b2 b3 x y z X Y Z

function mat = vectToMat ( v )
  mat = [ 0   -v(3) v(2);
          v(3) 0   -v(1);
         -v(2) v(1) 0    ];
endfunction

a = [a1 a2 a3];  b = [b1 b2 b3];
acb = cross(a,b)
macb = vectToMat(acb)
comm = vectToMat(a) * vectToMat(b) - vectToMat(b) * vectToMat(a)
x = sym([1 0 0]); y = sym([0 1 0]); z=sym([0 0 1]);
X = vectToMat(x);  Y = vectToMat(y);  Z = vectToMat(z);
t=0.01
Xd = double(X); Yd=double(Y); Zd=double(Z);
rotationXY = expm(t*Xd) * expm(t*Yd) * expm(-t*Xd) * expm(-t*Yd)
rotationZ = expm(t^2*Zd);
max max( abs( rotationXY -rotationZ ))
```

1.12.3 Output

```
acb = (sym) [a2 b3 - a3 b2  -a1 b3 + a3 b1  a1 b2 - a2 b1]  (1x3 matrix)
macb = (sym 3x3 matrix)

        0        -a1 b2 + a2 b1  -a1 b3 + a3 b1

  a1 b2 - a2 b1        0         -a2 b3 + a3 b2

  a1 b3 - a3 b1  a2 b3 - a3 b2         0

comm = (sym 3x3 matrix)

        0        -a1 b2 + a2 b1  -a1 b3 + a3 b1

  a1 b2 - a2 b1        0         -a2 b3 + a3 b2

  a1 b3 - a3 b1  a2 b3 - a3 b2         0

t =  0.010000
rotationXY =
   1.0000e+00  -9.9997e-05  -4.9996e-07
   9.9997e-05   1.0000e+00  -5.0001e-07
   5.0001e-07   4.9996e-07   1.0000e+00

ans =
   5.0001e-07
```

1.13 Third order Runge-Kutta method

1.13.1 Method

$$y'(t) = f(t,y), \quad y''(t) = (\partial_t + f(t,y)\partial_y)f(t,y), \quad y''(t) = (\partial_t + f(t,y)\partial_y)^2 f(t,y),$$
$$y(t+\Delta t) \approx y(t) + y'(t)\Delta t + \frac{y''(t)}{2!}\Delta t^2 + \frac{y''(t)}{3!}\Delta t,$$
$$k_1 = f(t,y), \quad k_2 = f(t + a\Delta t, y + bk_1\Delta t), \quad k_3 = f(t + c\Delta t, y + dk_1\Delta t + ek_2\Delta t),$$
$$y(t+\Delta t) = y(t) + (w_1k_1 + w_2k_2 + w_3k_3)\Delta t.$$

1.13.2 Script

```
clear all;
pkg load symbolic;
syms  f(t,y) f1 y t dt
syms  a b c d e w1 w2 w3

f1 = sym( f(t,y) );
f2 = diff( f1,t ) + f1 * diff( f1,y );
f3 = diff( f2,t ) + f1 * diff( f2,y );
#c1 = subs( f1, {y,t},{y0,t0} )
#c2 = subs( f2, {y,t},{y0,t0} )
#c3 = subs( f3, {y,t},{y0,t0} )
dytaylor = f1*dt + f2*dt^2/factorial(sym(2)) + f3*dt^3/factorial(sym(3));

k1 = f(t,y);
k2 = taylor( f(t+a*dt,y+b*dt*k1), dt, 0, 'order', 3 );
k3 = taylor( f(t+c*dt,y+d*dt*k1+e*dt*k2), dt, 0, 'order', 3 );

result = simplify( dytaylor/dt - ( w1*k1 + w2*k2 + w3*k3 ) );
clist = sym2poly( result, dt );
c1 = clist(1);
c2 = clist(2);
c3 = clist(3);
latex(c1)
latex(c2);
latex(c3);
```

1.13.3 Output

Unfortunately the output of $c1$ and $c2$ are quite unintelligible, the command "latex" turns at least $c3$ and $c2$ into more readable forms

$$c3 = -w_1 f(t,y) - w_2 f(t,y) - w_3 f(t,y) + f(t,y),$$

$$c2 = aw_2 \left.\frac{\partial}{\partial \xi_1} f(\xi_1, y)\right|_{\xi_1=t} - bw_2 f(t,y) \left.\frac{\partial}{\partial \xi_2} f(t,\xi_2)\right|_{\xi_2=y} - cw_3 \left.\frac{\partial}{\partial \xi_1} f(\xi_1, y)\right|_{\xi_1=t} - dw_3 f(t,y) \left.\frac{\partial}{\partial \xi_2} f(t,\xi_2)\right|_{\xi_2=y}$$
$$-ew_3 f(t,y) \left.\frac{\partial}{\partial \xi_2} f(t,\xi_2)\right|_{\xi_2=y} + \frac{1}{2} f(t,y) \frac{\partial}{\partial y} f(t,y) + \frac{1}{2} \frac{\partial}{\partial t} f(t,y)$$

The $c3 = c2 = 0$ equations are

$$0 = -1 + w_1 + w_2 + w_3,$$
$$0 = 1 - 2bw_2 - 2dw_3 - 2ew_3, \qquad 0 = 1/2 - aw_2 - cw_3.$$

In principle it is possible to get the $c3 = 0$ equations (taken from the Mathematica chapter)

$$0 = 1 - 6bew_3, \quad 0 = 1 - 3b^2 w_2 - 3d^2 w_3 - 6dew_3 - 3e^2 w_3,$$
$$0 = 1/6 - aew_3, \quad 0 = 1 - 3abw_2 - 3c(d+e)w_3, \quad 0 = 1 - 3a^2 w_2 - 3c^2 w_3.$$

Then "solve" might solve these equations, so in this way we end up with third order Runge-Kutta methods. The sad truth is that Octave is not very well suited for complicated symbolic manipulations, so we give it up.

1.14 Linear elasticity, plane wave solution

1.14.1 PDE

$$\vec{\phi}_{tt} - \mu \Delta \vec{\phi} - (\lambda + \mu) \nabla (\operatorname{div} \vec{\phi}) = \vec{0},$$
$$\vec{\phi}(t, x_1, x_1, x_3) = \vec{a} \cdot \exp\left(i[\vec{k} \cdot \vec{x} - \omega t] \right), \qquad \vec{k} = (k, 0, 0).$$

1.14.2 Script

```
clear all;
pkg load symbolic;
```

```
syms   x1 x2 x3 t a1 a2 a3 k omega

lambda = sym(7);
mu = sym(9);
phi = [a1 a2 a3]*exp(i*(k*x1-omega*t));

eq = diff( phi, t, t )   \
 -mu*[laplacian(phi(1),[x1,x2,x3]) , laplacian(phi(2),[x1,x2,x3]) ,  \
  laplacian(phi(3),[x1,x2,x3])]   -  (lambda+mu)*   \
   transpose(gradient(diff(phi(1),x1)+diff(phi(2),x2)+diff(phi(3),x3),[x1,x2,x3]));
#  ' : adjoint, not transpose. divergence is not implemented in symbolic

eq1 = simplify(eq(1)*exp(-i*(k*x1-omega*t)))
eq2 = simplify(eq(2)*exp(-i*(k*x1-omega*t)))
eq3 = simplify(eq(3)*exp(-i*(k*x1-omega*t)))
```

1.14.3 Output

```
eq1 = (sym)

         2        2
  a1 25 k  - omega

eq2 = (sym)

        2        2
  a2 9 k  - omega

eq3 = (sym)

        2        2
  a3 9 k  - omega
```

Chapter 2

Mathematica

2.1 Symbolic solution

2.1.1 Code

```
DSolve[y'[t] + 3 y[t] == 0, y[t], t]
DSolve[{y'[t] + 3 y[t] == 0, y[0] == 1}, y[t], t]
DSolve[y''[t] + 3 y[t] == 0, y[t], t]
DSolve[{y''[t] + 3 y[t] == t^2, y[-1] == 0, y'[-1] == 0}, y[t],   t] // Simplify
DSolve[{y'[t] == Exp[-t^2], y[0] == 1}, y[t], t]
DSolve[t^2 y''[t] + t y'[t] + t^2 y[t] == 0, y[t], t]
DSolve[{x'[t] == y[t], y'[t] == -4 x[t]}, {x[t], y[t]}, t]
```

2.2 Numerical solution, plotting 1.

2.2.1 Code

```
f[y_, t_] := (y - 1) (y - 2) Exp[-t]
DSolve[y'[t] == f[y[t], t], y[t], t]
VectorPlot[{1, f[y, t]}, {t, 0, 1}, {y, 0, 3}]
nds = NDSolve[{y'[t] == f[y[t], t], y[0] == 1.9}, y[t], {t, 0, 3}]
Plot[y[t] /. nds, {t, 0, 3}]
(y[t] /. nds) /. t -> 3
ndst = Table[
   NDSolve[{y'[t] == f[y[t], t], y[0] == 1 + 0.1 i}, y[t], {t, 0, 3}], {i, 0, 10}];
Plot[y[t] /. ndst, {t, 0, 3}]
```

2.3 Numerical solution, plotting 2.

2.3.1 Code

```
f[{y1_, y2_}] := {y1 - y1 y2, -y2 + y1 y2}
VectorPlot[f[{y1, y2}], {y1, 0, 2}, {y2, 0, 2}]
nds = NDSolve[{y1'[t] == f[{y1[t], y2[t]}][[1]],
                            y2'[t] == f[{y1[t], y2[t]}][[2]],
   y1[0] == 1.5, y2[0] == 1.5},   {y1[t], y2[t]}, {t, 0, 10}]
Plot[{y1[t], y2[t]} /. nds, {t, 0, 10}]
ParametricPlot[{y1[t], y2[t]} /. nds, {t, 0, 10}]
ParametricPlot3D[{t, y1[t], y2[t]} /. nds, {t, 0, 10}]
ndst = Table[NDSolve[{y1'[t] == f[{y1[t], y2[t]}][[1]],y2'[t] == f[{y1[t], y2[t]}][[2]],
     y1[0] == 1 + 0.1*i, y2[0] == 1 + 0.1*i},   {y1[t], y2[t]}, {t, 0,10}],{i, 1, 9}];
ParametricPlot[{y1[t], y2[t]} /. ndst, {t, 0, 10}]
```

2.4 Numerical derivation

2.4.1 Code

```
min = 4; max = 10; dm = max - min;
dx = Table[1/2^i, {i, min, max}] // N
x0 = 1;
logErrorTable = Table[{Log[dx[[i]]],
    Log[Abs[(Sin[x0 + dx[[i]]] - Sin[x0])/dx[[i]] - Cos[x0]]]},{i, 1, dm + 1}];
Fit[logErrorTable, {1, x}, x]
logErrorTable = Table[{Log[dx[[i]]],
    Log[Abs[(Sin[x0 + dx[[i]]] - Sin[x0 - dx[[i]]])/(2*dx[[i]]) - Cos[x0]]]},{i, 1, dm + 1}];
Fit[logErrorTable, {1, x}, x]
x0 = 0;
logErrorTable = Table[{Log[dx[[i]]],
    Log[Abs[(Sin[x0 + dx[[i]]] - Sin[x0])/dx[[i]] - Cos[x0]]]},{i, 1, dm + 1}];
Fit[logErrorTable, {1, x}, x]
```

2.5 Errors of the Euler and Heun methods

2.5.1 Code

```
min = 4; max = 15; dm = max - min;
dx = Table[1/2^i, {i, min, max}] // N;
f[y_, t_] := t y
t0 = 0; y0 = 1; t1 = 3;
ds = DSolve[{y'[t] == f[y[t], t], y[t0] == y0}, y[t], t]
Print["exact y(3)"];
exactY3 = (ds /. t -> t1 // N)[[1, 1, 2]]
Print["NDSolve y(3)"];
NDSolve[{y'[t] == f[y[t], t], y[t0] == y0}, y[t], {t, 0, t1}] /. t -> 3

Euler[dt_] := Module[{t = t0, y = y0, n = Round[(t1 - t0)/dt]},
  For[i = 1, i <= n, i++,
   y = y + f[y, t]*dt;
   t = t + dt;
   ];
   Return[y]]
Heun[dt_] :=
 Module[{t = t0, y = y0, n = Round[(t1 - t0)/dt], k1, k2},
  For[i = 1, i <= n, i++,
   k1 = f[y, t];
   k2 = f[y + f[y, t]*dt, t + dt];
   y = y + (k1 + k2)/2 *dt;
   t = t + dt;
   ];
   Return[y]]
elt = Log[Abs[
    Table[{dx[[i]], Euler[dx[[i]]] - exactY3}, {i, 1, 1 + (max - min)}]]];
Fit[elt, {1, x}, x]
hlt = Log[Abs[
    Table[{dx[[i]], Heun[dx[[i]]] - exactY3}, {i, 1, 1 + (max - min)}]]];
Fit[hlt, {1, x}, x]
```

2.6 Taylor expansion of y(t+dt).

2.6.1 DE

$$\frac{d}{dt}y(t) = f(t,y) = 5 + t \cdot y(t), \quad y(2) = 3.$$

$$y^{(n)}(t) = (\partial_t f + f \cdot \partial_y f)^{n-1} f$$

2.6.2 Code

```
f1 := 5 + t y
V[g_] := D[g, t] + f1 D[g, y]
f2 = V[f1]
f3 = V[f2]
initialCond = {t -> 2, y -> 3}
coeffs = {y, f1, f2, f3} /. initialCond
taylor =
 y + f1 dt + f2 dt^2/Factorial[2] + f3 dt^3/Factorial[3] /. initialCond
```

2.7 Linearization

2.7.1 DE

$$\frac{d}{dt}\begin{pmatrix} y_1(t) \\ y_2(t) \end{pmatrix} = \begin{pmatrix} 2y_1 - 4y_1y_2 \\ -3y_2 + 2y_1y_2 \end{pmatrix}.$$

2.7.2 Code

```
f = {2 y1 - 4 y1 y2, -3 y2 + 2 y1 y2}
eqs = {f[[1]] == 0, f[[2]] == 0}
(*  Reduce[eqs,{y1,y2}]                (* Try to find all exact sol. *)
    NSolve[eqs,{y1,y2}]                (* Try to find most of the numerical sol. *)
    FindRoot[eqs,{{y1,1.6},{y2,0.6}}]  (* Try to find a single num. sol.  *)         *)
sol = Solve[eqs, {y1, y2}]
jac = D[f, {{y1, y2}}];
jac // TableForm
j1 = jac /. sol[[1]];
j1 // TableForm
j2 = jac /. sol[[2]];
j2 // TableForm
```

2.8 Diagonalization, matrix exponential

2.8.1 Linear algebra

1. Solve $\det(A - \lambda E) = 0$.
2. Solve $(A - \lambda E)\vec{v}_k = \vec{0}$.
3. Set $S = [\vec{v}_1, \ldots, \vec{v}_n]$, $D = \mathrm{diag}(\lambda_1, \ldots . \lambda_n)$. Then $A = SDS^{-1}$.
4. $\exp(tA) = Se^{tD}S^{-1}$.
5. First column of $\exp(tA)$ solves $\vec{y}' = A\vec{y}$, $\vec{y}(0) = \vec{e}_1$, etc.

2.8.2 Code

```
B = {{2, 0}, {0, 3}};                          (* D is "protected", so we use B  *)
S = {{3, 2}, {3, 1}};
A = S.B.Inverse[S];
TableForm[A]                                   (* Now we have a nice matrix  *)

A = {{4, -2}, {1, 1}}
lambdas = Solve[Det[A - lambda IdentityMatrix[2]] == 0, lambda]
eval = Eigenvalues[A]
esys = Eigensystem[A]
evac = Eigenvectors[A]
S = Transpose[{evac[[1]], evac[[2]]}];         (* eigenvector basis *)
S // TableForm
Si = Inverse[S];
B = Si.A.S;                                    (* diagonalized matrix *)
B // TableForm
mexp = MatrixExp[t A];                         (* built in matexp *)
mexp // TableForm
dexp = S.DiagonalMatrix[Exp[t eval]].Si;       (* matexp from lin.alg. *)
dexp // TableForm
de = Join[Map[(# == 0) &, Flatten[A.{y1[t], y2[t]} - {y1'[t], y2'[t]}]],
            {y1[0] == 1, y2[0] == 0}]          (* y'=Ay,y(0)=e_1,plus some messy organization*)

DSolve[de, {y1[t], y2[t]}, t] // Expand
{mexp[[1, 1]], mexp[[2, 1]]}                   (* same as  DSolve...  *)
```

Try to repeat this exercise (and try to understand the peculiarities of the various cases) for A equal to

$$\begin{pmatrix} 2 & 0 \\ 0 & 3 \end{pmatrix}, \quad \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}, \quad \begin{pmatrix} 2 & 0 \\ 4 & 3 \end{pmatrix}, \quad \begin{pmatrix} 2 & 4 \\ 0 & 3 \end{pmatrix} \quad \begin{pmatrix} 2 & 5 \\ 5 & 2 \end{pmatrix}, \quad \begin{pmatrix} 2 & 5 \\ -5 & 2 \end{pmatrix}, \quad \begin{pmatrix} 2 & 0 & 0 \\ 4 & 3 & 0 \\ 0 & 0 & 7 \end{pmatrix}.$$

2.9 Matrix exponential, Jordan normal form

2.9.1 DE

Radioactive decay, spring-mass system.

$$\alpha \to \beta \to \emptyset, \qquad ||m_1 = 1|| \longleftarrow (k = 2) \longrightarrow ||m_2 = 1||$$

$$\frac{d}{dt}\vec{y} = A\vec{y}, \qquad A = \begin{pmatrix} 3 & 0 \\ 4 & 3 \end{pmatrix} \qquad \text{or} \qquad A = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -1/2 & 1/2 & 0 & 0 \\ 1/2 & -1/2 & 0 & 0 \end{pmatrix}$$

2.9.2 Code

```
(*    A = {{-3, 0}, {4, -3}}   *)
A = {{0, 0, 1, 0},
     {0, 0, 0, 1},
     {-1/2, 1/2, 0, 0},
     {1/2, -1/2, 0, 0}}
jform = JordanDecomposition[A];
S = jform[[1]];                  S //TableForm
J = jform[[2]];                  J //TableForm
SJSi = S.J.Inverse[S];        SJSi // TableForm
etJ = MatrixExp[t J];          etJ // TableForm
SetJSi = S.etJ.Inverse[S]; SetJSi // TableForm
mtA = MatrixExp[t A];          mtA // TableForm
```

2.9.3 Output

$$S = \begin{pmatrix} 1 & 0 & -i & i \\ 1 & 0 & i & -i \\ 0 & 1 & -1 & -1 \\ 0 & 1 & 1 & 1 \end{pmatrix}, \qquad J = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -i & 0 \\ 0 & 0 & 0 & i \end{pmatrix},$$

$$e^{tA} = \frac{1}{2}\begin{pmatrix} \cos(t)+1 & 1-\cos(t) & t+\sin(t) & t-\sin(t) \\ 1-\cos(t) & \cos(t)+1 & t-\sin(t) & t+\sin(t) \\ -\sin(t) & \sin(t) & \cos(t)+1 & 1-\cos(t) \\ \sin(t) & -\sin(t) & 1-\cos(t) & \cos(t)+1 \end{pmatrix}.$$

2.10 Diagonalization, complex eigenvalues

2.10.1 DE

$$y'' = -y - 0.2\,y', \qquad \frac{d}{dt}\vec{y} = \begin{pmatrix} 0 & 1 \\ -1 & -0.2 \end{pmatrix}\vec{y}$$

Strategy:

- Use complex numbers and vectors.
- Or: Matrix is real $\Longrightarrow$ eigenvalues are either real or come in complex conjugate pairs. Then corresponding eigenvectors can be picked as complex conjugates. Choose one from each complex eigenvalue pair and use the real and imaginary parts of of the corresponding eigenvectors as basis vectors. In our case

$$A = \begin{pmatrix} 0 & 1 \\ -1 & -0.2 \end{pmatrix} = SDS^{-1}$$

$$= \begin{pmatrix} 0.707 + 0.i & 0.707 + 0.i \\ -0.0707 + 0.703i & -0.0707 - 0.703i \end{pmatrix} \begin{pmatrix} -0.1 + 0.994i & 0. \\ 0. & -0.1 - 0.994i \end{pmatrix} \begin{pmatrix} 0.707 + 0.i & 0.707 + 0.i \\ -0.0707 + 0.703i & -0.0707 - 0.703i \end{pmatrix}^{-1}$$

$$= TD_{real}T^{-1} = \begin{pmatrix} 0.707 & 0. \\ -0.0707 & 0.703 \end{pmatrix} \begin{pmatrix} -0.1 & 0.994 \\ -0.994 & -0.1 \end{pmatrix} \begin{pmatrix} 0.707 & 0. \\ -0.0707 & 0.703 \end{pmatrix}^{-1}$$

Furthermore

$$\exp(t \cdot D_{real}) = \exp\left[t \cdot \begin{pmatrix} -0.1 & 0.994 \\ -0.994 & -0.1 \end{pmatrix}\right] = \exp(-0.1 \cdot t) \begin{pmatrix} \cos(0.994\,t) & \sin(0.994\,t) \\ -\sin(0.994\,t) & \cos(0.994\,t) \end{pmatrix}$$

So $\exp(tA) = T \exp(t \cdot D_{real})\, T^{-1}$.

2.10.2 Code

```
A = {{0, 1}, {-1, -0.2}};  t = 5;
Print["A:"]
A // TableForm
Print["Eigenvalues of A:"]
evals = Eigenvalues[A]
evects = Eigenvectors[A];
Print["S:"]
S = Transpose[evects];
S // TableForm
B = DiagonalMatrix[evals];
Print["Check diagonalization:"]
SBSi = S.B.Inverse[S];
SBSi // Chop // TableForm
Print["S exp(tA) Si:"]
eB = S.DiagonalMatrix[Exp[t evals]].Inverse[S] // Chop;
eB // TableForm

v = evects[[1]];
T = Transpose[{Re[v], Im[v]}];
Print["Real normal form, T:"]
T // TableForm
a = Re[evals[[1]]]; b = Im[evals[[1]]];
Br = {{a, b}, {-b, a}};
Print["Real normal form, B_real:"]
Br // TableForm
TBrTi = T.Br.Inverse[T];
Print["Check real normal form:"]
TBrTi // Chop // TableForm
eBr = Exp[t a] {{Cos[t b], Sin[t b]}, {-Sin[t b], Cos[t b]}};
Print["T exp( t B_real ) Ti:"]
TeBrTi = T.eBr.Inverse[T];
TeBrTi // TableForm
```

2.11 State space portraits

2.11.1 DE

$$\frac{d}{dt}\vec{y}(t) = A\vec{y}(t).$$

Plots:

1. vectorfield $\vec{y} \to A\vec{y}$,
2. $t \to \left(e^{tA}\right)_{i,j}$,
3. $t \to \vec{y}(t)$ parametric plot,
4. stream plots.

2.11.2 Code

```
type[n_] := Module[{}, Which[
   n == 1,  (*damped oscillator*)
   A = {{0, 1}, {-1, -0.4}};
   T = 10; y0 = {1, 0};
   v = Eigenvectors[A][[1]]/Norm[Eigenvectors[A][[1]]];
   v1 = Re[v];
   v2 = Im[v];,
   n == 2,   (*harmonic oscillator*)
   A = {{0, 1}, {-0.5, 0}};
   T = 10;  y0 = {1, 0};
   v = Eigenvectors[A][[1]]/Norm[Eigenvectors[A][[1]]];
   v1 = Re[v];
   v2 = Im[v];,
   n == 3,    (*drain*)
   A = {{-2, 0}, {1, -4}}; T = 1;  y0 = {1, 0};
   v1 = Eigenvectors[A][[1]]/Norm[Eigenvectors[A][[1]]];
   v2 = Eigenvectors[A][[2]]/Norm[Eigenvectors[A][[2]]];,
   n == 4,    (*saddle*)
   A = {{-2, 1}, {2, 3}}; T = 1;
   y0 = {1, -0.3};
   v1 = Eigenvectors[A][[1]]/Norm[Eigenvectors[A][[1]]];
   v2 = Eigenvectors[A][[2]]/Norm[Eigenvectors[A][[2]]];,
   n == 5,    (*critical damping*)
   A = {{0, 1}, {-1, -2}}; T = 1;
   y0 = {1, 0};
   jd = JordanDecomposition[A][[1]];
   v1 = Transpose[jd][[1]];
   v2 = Transpose[jd][[2]];
   ]]
n = 5; type[n];
etA[t_] := MatrixExp[t A];
gr0 = VectorPlot[A.{y1, y2}, {y1, -1, 1}, {y2, -1, 1}, Axes -> True];
gr1 = GraphicsGrid[Table[Plot[etA[t][[i, j]], {t, 0, T},
     PlotRange -> {-1.5, 1.5}, Ticks -> {{0, T}, {-1, 1}},
     PlotLabel ->
      Subscript["exp(t A)", ToString[i] <> ToString[j]]], {i, 2}, {j,
     2}]];
gr2 = ParametricPlot[etA[t].y0, {t, 0, 3 T},
   PlotRange -> {{-1, 1}, {-1, 1}}];
gr3 = StreamPlot[A.{y1, y2}, {y1, -1, 1}, {y2, -1, 1}, Axes -> True,
   Epilog -> {Thickness[0.01], Arrow[{{0, 0}, v1}],
     Arrow[{{0, 0}, v2}]}];
gr = GraphicsGrid[{{gr0, gr1}, {gr2, gr3}}]
```

2.11.3 Remarks

1. Underdamped oscillator.

$$A\vec{v} = (-0.2 + 0.97\,i)\vec{v}, \qquad \vec{v}_1 = Re(\vec{v}), \qquad T^{-1}AT = \begin{pmatrix} -0.2 & 0.97 \\ -0.97 & -0.2 \end{pmatrix}.$$

   $\vec{v} \to e^{i\alpha}\vec{v}$ rotates by α the $[\vec{v}_1, \vec{v}_2]$ coordinate system.

2. Harmonic oscillator.

$$A\vec{v} = 0.707\, i \cdot \vec{v}, \qquad \vec{v}_1 = Re(\vec{v}), \quad \vec{v}_2 = Im(\vec{v}).$$
$$T = [\vec{v}_1, \vec{v}_2], \qquad T^{-1}AT = \begin{pmatrix} 0 & 0.707 \\ -0.707 & 0 \end{pmatrix}.$$

$\vec{v} \to e^{i\alpha}\vec{v}$ rotates by α the $[\vec{v}_1, \vec{v}_2]$ coordinate system.

3. Stable node. $\vec{v}_1, \vec{v}_2$ have fixed directions and arbitrary nonzero lengths.

4. Saddle. $\vec{v}_1, \vec{v}_2$ have fixed directions and arbitrary nonzero lengths.

5. Degenerate node. $\vec{v}_1$ has fixed direction and determines $\vec{v}_2$. In the $[\vec{v}_1, \vec{v}_2]$ coordinate system

$$J = [\vec{v}_1, \vec{v}_2]^{-1} A\, [\vec{v}_1, \vec{v}_2] = \begin{pmatrix} -1 & 1 \\ 0 & 1 \end{pmatrix}.$$

2.11.4 Output

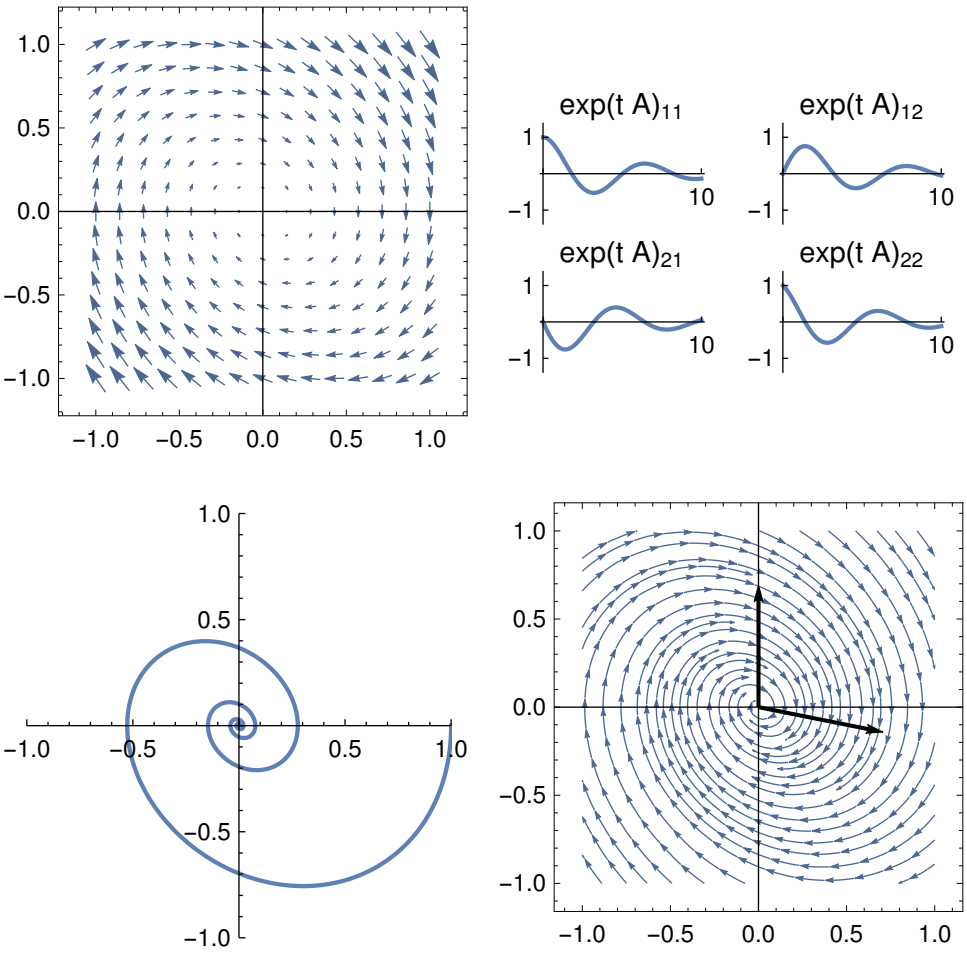


Figure 2.1: Underdamped oscillator, $\lambda_{1,2} = -0.2 \pm 0.97\, i$. Stable focus, spiral.

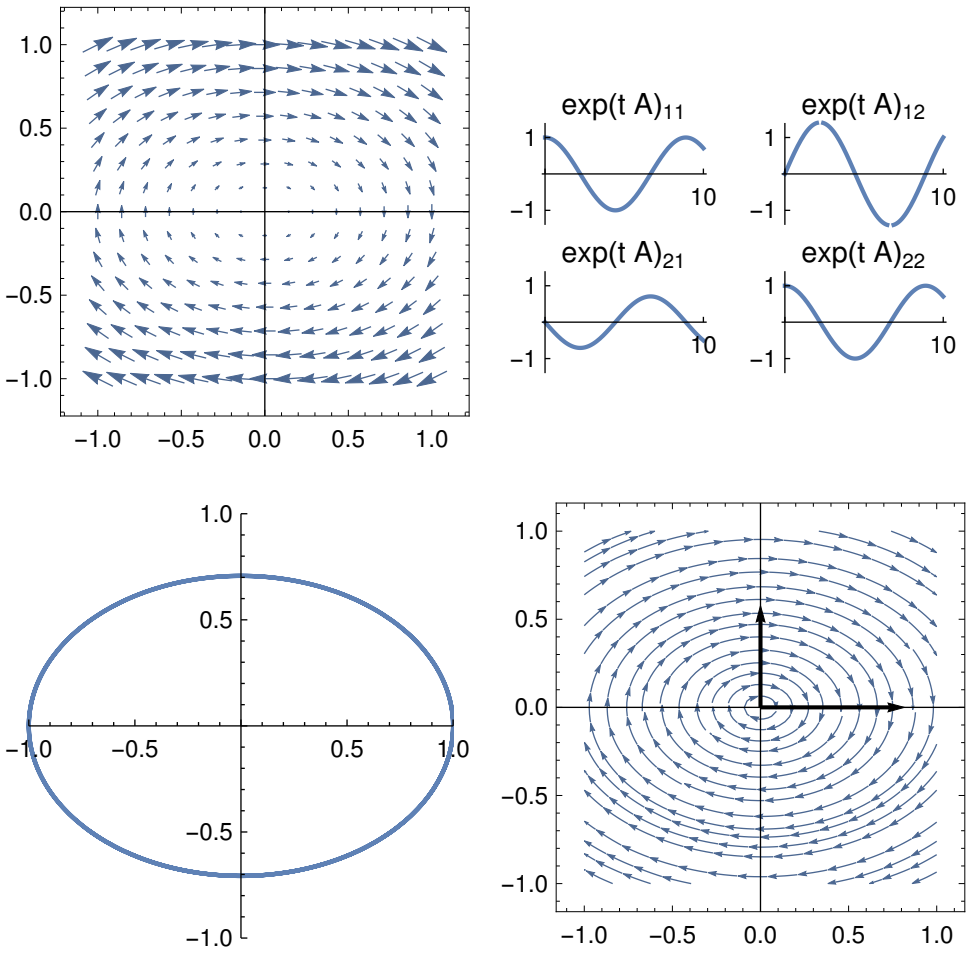


Figure 2.2: Harmonic oscillator, $\lambda_{1,2} = \pm 0.7071\, i$. Center.

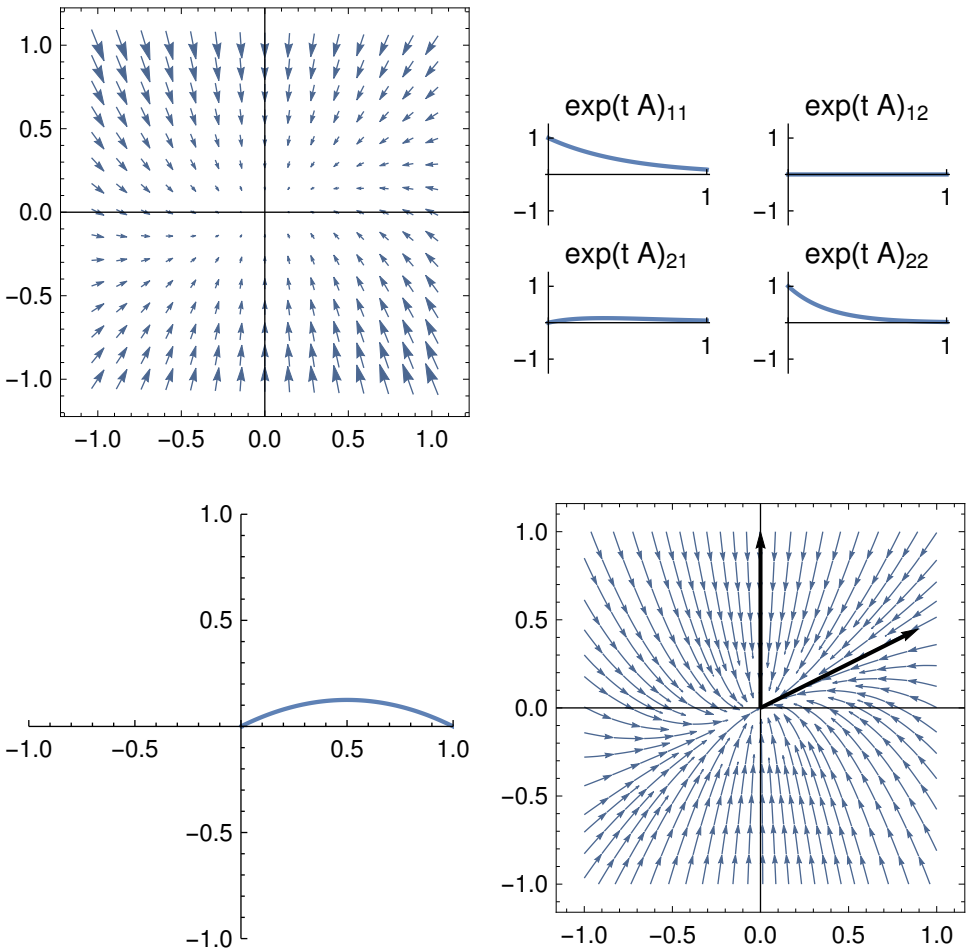


Figure 2.3: Drain, $\lambda_1 = -4, \lambda_2 = -2$. Stable node.

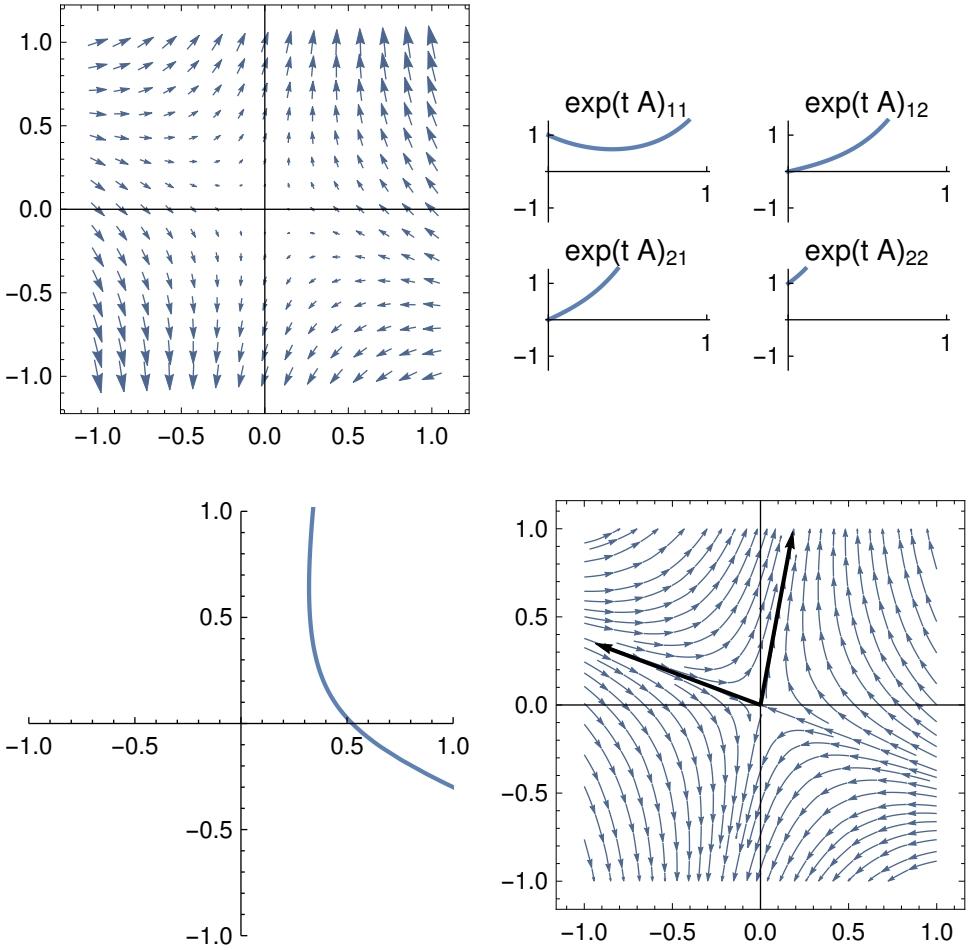


Figure 2.4: Saddle, $\lambda_1 = 3.372, \lambda_2 = -2.372$. Unstable.

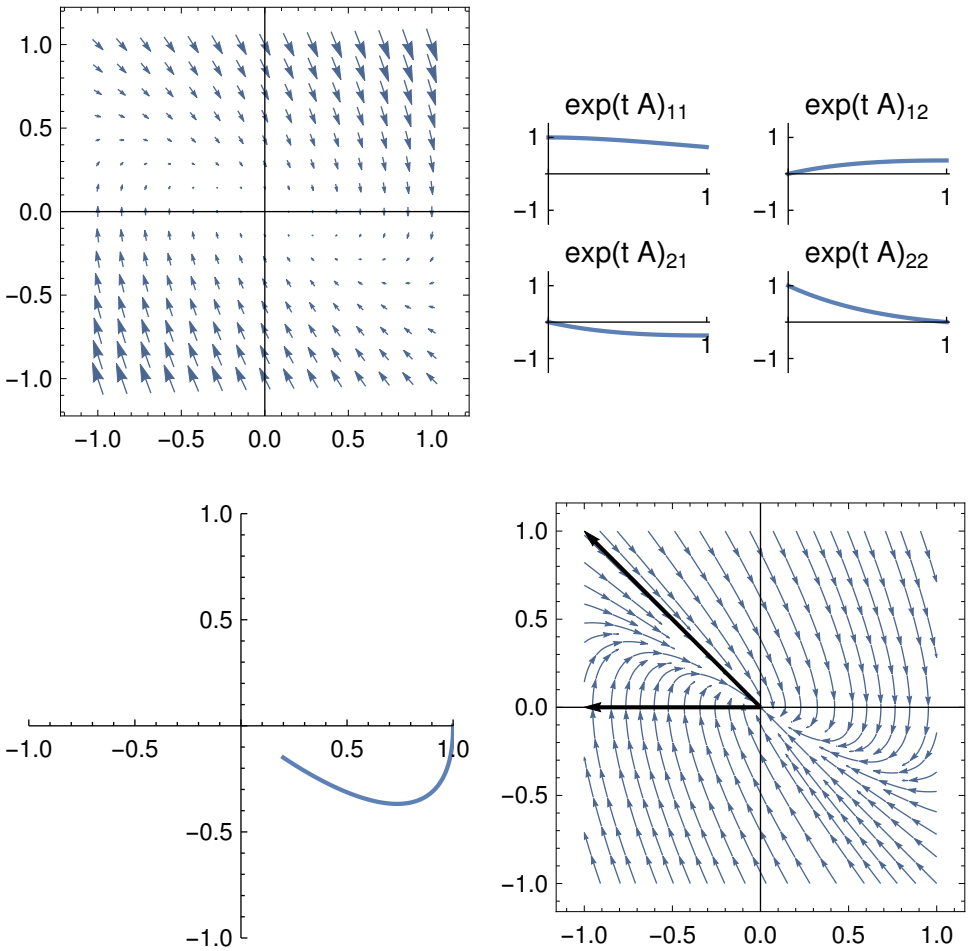


Figure 2.5: Critically damped oscillator, $\lambda_1 = -1$. Nontrivial Jordan decomposition. Stable degenerate node.

2.12 Rotation group $SO(3)$

2.12.1 Lie group, Lie algebra

- Lie group: Special orthogonal (rotation) matrices $SO(3) = \{O \in \mathrm{Mat}_3(\mathbb{R}) \mid \det O = 1,\ O^{-1} = O^T\}$.
- Lie algebra: $\mathfrak{so}(3) = \{A \in \mathrm{Mat}_3(\mathbb{R}) \mid A = -A^T\}$.
- Let
$$\vec{a} = (a_1, a_2, a_3), \qquad L(\vec{a}) = \begin{pmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{pmatrix} = A$$
(and the same for the rest of the alphabet). Then
$$L(\vec{a} \times \vec{b}) = L(\vec{a})L(\vec{b}) - L(\vec{b})L(\vec{a}) = [L(\vec{a}), L(\vec{b})].$$
- $SO(3) = \exp(\mathfrak{so}(3))$. The rotation matrix around $\vec{a}$ by angle $|\vec{a}|$ is $R_a = \exp(L(\vec{a}))$.
- The product of rotation matrices is another rotation matrix, so
$$\exp(tL(\vec{a}))\exp(tL(\vec{b}))\exp(-tL(\vec{a}))\exp(-tL(\vec{b})) = \exp(t^2 L(\vec{c})),$$
$$L(\vec{c}) \approx L(\vec{a})L(\vec{b}) - L(\vec{b})L(\vec{a}) = [L(\vec{a}), L(\vec{b})].$$
- If
$$\vec{x} = (1,0,0),\ \ \vec{y} = (0,1,0),\ \ \text{then}\ \ \vec{z} = \vec{x} \times \vec{y} = (0,0,1),$$
so
$$\exp(tL(\vec{x}))\exp(tL(\vec{y}))\exp(-tL(\vec{x}))\exp(-tL(\vec{y})) \approx \exp(t^2 L(\vec{z})).$$
- Consequently by combining rotations around the x, y axes we can produce rotations around the z axis.

2.12.2 Code

```
VectToMat[{v1_, v2_, v3_}] := ({
   {0, -v3, v2},
   {v3, 0, -v1},
   {-v2, v1, 0}
  })
a = {a1, a2, a3}; b = {b1, b2, b3};
acb = Cross[a, b]
macb = VectToMat[acb];
macb // TableForm
comm = VectToMat[a].VectToMat[b] - VectToMat[b].VectToMat[a];
comm // TableForm
x = {1, 0, 0}; y = {0, 1, 0}; z = {0, 0, 1};
X = VectToMat[x]; Y = VectToMat[y]; Z = VectToMat[z];
t = 0.01;
Comm[g1_, g2_] := g1.g2.Inverse[g1].Inverse[g2]
rotationXY = Comm[MatrixExp[t X], MatrixExp[t Y]];
rotationXY // TableForm
rotationZ = MatrixExp[t^2 Z];
rotationZ // TableForm
Max[rotationZ - rotationXY ]
```

2.13 Third order Runge-Kutta method

2.13.1 Method

$$y'(t) = f(t,y), \quad y''(t) = (\partial_t + f(t,y)\partial_y)f(t,y), \quad y''(t) = (\partial_t + f(t,y)\partial_y)^2 f(t,y),$$
$$y(t+\Delta t) \approx y(t) + y'(t)\Delta t + \frac{y''(t)}{2!}\Delta t^2 + \frac{y''(t)}{3!}\Delta t,$$
$$k_1 = f(t,y), \quad k_2 = f(t + a\Delta t, y + bk_1\Delta t), \quad k_3 = f(t + c\Delta t, y + dk_1\Delta t + ek_2\Delta t),$$
$$y(t+\Delta t) = y(t) + (w_1k_1 + w_2k_2 + w_3k_3)\Delta t.$$

2.13.2 Code

```
 ClearAll["Global`*"]
y1 = f[t, y];
y2 = D[y1, t] + y1 D[y1, y];
y3 = D[y2, t] + y1 D[y2, y] // Expand;
deltaY = y1 dt + y2/2 dt^2 + y3/6 dt^3;
k1 = f[t, y];
k2 = Series[f[t + a dt, y + b dt k1], {dt, 0, 2}] // Normal;
k3 = Series[f[t + c dt, y + d dt k1 + e dt k2], {dt, 0, 2}] // Normal;
cl = CoefficientList[deltaY/dt - (w1 k1 + w2 k2 + w3 k3) // Simplify, dt]

eqs = {0 == -1 + w1 + w2 + w3,
   0 == 1 - 2 b w2 - 2 d w3 - 2 e w3,
   0 == 1/2 - a w2 - c w3,
   0 == 1 - 6 b e w3,
   0 == 1 - 3 b^2 w2 - 3 d^2 w3 - 6 d e w3 - 3 e^2 w3,
   0 == 1/6 - a e w3,
   0 == 1 - 3 a b w2 - 3 c (d + e) w3,
   0 == 1 - 3 a^2 w2 - 3 c^2 w3};
sol = Solve[eqs];  (*explicitly solvable eqs, mostly complicated sol.*)
shortSol = Take[Solve[eqs], -2]
rk3a = Append[shortSol[[1]] /. w3 -> 1/2, w3 -> 1/2];
rk3b = Append[shortSol[[2]] /. e -> 1, e -> 1];
rk3Kutta =
  {a -> 1/2, c -> 1, b -> 1/2, d -> -1, e -> 2, w1 -> 1/6, w2 -> 2/3,  w3 -> 1/6};

min = 4; max = 10; dm = max - min;
dx = Table[1/2^i, {i, min, max}] // N;
f[y_, t_] := t y
t0 = 0; y0 = 1; t1 = 3;
ds = DSolve[{y'[t] == f[y[t], t], y[t0] == y0}, y[t], t]
Print["exact y(3)"];
exactY3 = (ds /. t -> t1 // N)[[1, 1, 2]]
Print["NDSolve y(3)"];
NDSolve[{y'[t] == f[y[t], t], y[t0] == y0}, y[t], {t, 0, t1}] /. t -> 3

RungeKutta3[dt_] := Module[{t = t0, y = y0, n = Round[(t1 - t0)/dt],
   k1, k2, k3, i},
  {a, b, c, d, e, w1, w2, w3} = {a, b, c, d, e, w1, w2, w3} /.
    rk3Kutta;
  For[i = 1, i <= n, i++,
   k1 = f[y, t];
   k2 = f[y + b k1 dt, t + a dt];
   k3 = f[y + ( d k1 + e k2)*dt, t + c dt];
   y = y + (w1 k1 + w2 k2 + w3 k3) *dt;
   t = t + dt;
   ]; Return[y]]
elt = Log[Abs[
    Table[{dx[[i]], RungeKutta3[dx[[i]]] - exactY3}, {i, 1, 1 + (max - min)}]]];
Print["error(dx) = C dx^alpha,   ln(error(dx)) = ln(C)+alpha ln(dx)"];
Fit[elt, {1, x}, x]
({
   {0, 0, 0, 0},
   {"a dt", "k2(k1)", 0, 0},
   {"c dt", "k3(k1)", "k3(k2)", 0},
   {"", "w1", "w2", "w3"}
  }) // TableForm
 ({
   {0, 0, 0, 0},
   {a, b, 0, 0},
   {c, d, e, 0},
   {"", w1, w2, w3}
  }) // TableForm
```

```
Print["dx:  ", dx[[dm]]];
Print["y(3):  ", RungeKutta3[dx[[dm]]]]
```

2.14 Linear elasticity (+ Maxwell equations), plane wave solution

2.14.1 PDE

$$\vec{\phi}_{tt} - \mu\Delta\vec{\phi} - (\lambda+\mu)\nabla(\operatorname{div}\vec{\phi}) = \vec{0},$$
$$\vec{\phi}(t,x_1,x_1,x_3) = \vec{a}\cdot\exp\left(i[\vec{k}\cdot\vec{x}-\omega t]\right), \qquad \vec{k} = (k,0,0).$$

2.14.2 Code

```
mu = 9; lambda = 7;
pde[phi_] := D[phi, t, t] - mu (Laplacian[phi, {x1, x2, x3}]) -
  (lambda + mu) Grad[Div[phi, {x1, x2, x3}], {x1, x2, x3}]
phi = {a1, a2, a3} Exp[I (k x1 - omega t)]
pdeEqs = pde[phi]/Exp[I (k x1 - omega t)] // Simplify;
eqs = Map[# == 0 &, pdeEqs]
Reduce[eqs, {omega, a1, a2, a3}]
```

2.14.3 Empty space Maxwell equations

$$\operatorname{div}\vec{E} = 0, \qquad \operatorname{div}\vec{B} = 0,$$
$$\operatorname{rot}\vec{E} + \frac{\partial\vec{B}}{\partial t} = 0, \qquad \operatorname{rot}\vec{B} - \mu_0\epsilon_0\frac{\partial\vec{E}}{\partial t} = 0,$$
$$\vec{E}(t,x_1,x_2,x_3) = \vec{e}\cdot\exp\left(i[\vec{k}\cdot\vec{x}-\omega t]\right), \qquad \vec{B}(t,x_1,x_2,x_3) = \vec{b}\cdot\exp\left(i[\vec{k}\cdot\vec{x}-\omega t]\right), \qquad \vec{k} = (k,0,0).$$

2.14.4 Code

```
e = {e1, e2, e3} Exp[I (k x1 - omega t)]
b = {b1, b2, b3} Exp[I (k x1 - omega t)]
eqDivE = Div[e, {x1, x2, x3}]/(I Exp[I (k x1 - omega t)])
eqDivB = Div[b, {x1, x2, x3}]/(I Exp[I (k x1 - omega t)])
eqRotEplusBdot =
 (Curl[e, {x1, x2, x3}] + D[b, t])/(I Exp[I (k x1 - omega t)]) // Simplify
eqRotBminusMuepsilonEdot =
 (Curl[b, {x1, x2, x3}] - m D[e, t])/(I Exp[I (k x1 - omega t)]) // Simplify
maxwellEqs = Map[(# == 0) &,
  Join[{eqDivE, eqDivB}, eqRotEplusBdot, eqRotBminusMuepsilonEdot]]
res = Reduce[Join[maxwellEqs, {k > 0, omega > 0, m > 0}],
  {e1, e2, e3, b1, b2, b3}, Reals]
```

2.14.5 Output

$$res =$$
$$\left(\left(\omega>0\wedge k>0\wedge m=\frac{k^2}{\omega^2}\wedge \mathrm{e1}=0\right)\vee\left(\omega>0\wedge k>0\wedge\left(0<m<\frac{k^2}{\omega^2}\vee m>\frac{k^2}{\omega^2}\right)\wedge \mathrm{e1}=0\wedge \mathrm{e2}=0\wedge \mathrm{e3}=0\right)\right)$$
$$\wedge\ \mathrm{b1}=0\ \wedge\ \mathrm{b2}=-\frac{\mathrm{e3}\,k}{\omega}\ \wedge\ \mathrm{b3}=\frac{\mathrm{e2}\,k}{\omega},$$

where $m=\mu_0\epsilon_0$. The relevant result is

$$\omega=\frac{k}{\sqrt{m}}, \quad e1=b1=0, \quad b2=-\sqrt{m}\,e3, \quad b3=\sqrt{m}\,e2.$$

So the speed of propagation of electromagnetic waves is $c=m^{-1/2}\approx 300,000\,km/sec$. A basis of the plane wave solutions in the x direction:

$$I: \qquad \vec{E}=\exp(i(ckt-kx))\cdot(0,1,0), \qquad \vec{B}=\exp(i(ckt-kx))\cdot(0,0,1/c),$$
$$II: \qquad \vec{E}=\exp(i(ckt-kx))\cdot(0,0,1), \qquad \vec{B}=\exp(i(ckt-kx))\cdot(0,-1/c,0).$$

These are transversally polarized waves, after taking the real parts, the electric and the magnetic fields are orthogonal to each other and to the velocity vector.

Plane wave pictures

Code:

```
s = 1.5;
eft1[x_] := s {Exp[I (0 - x)]*1, 0}
bft1[x_] := s {0, Exp[I (0 - x)]*1}
eft2[x_] := s {0, Exp[I (0 - x)]*1}
bft2[x_] := s {-Exp[I (0 - x)]*1, 0}
efc[x_] := (eft1[x] + I eft2[x])/Sqrt[2]
bfc[x_] := (bft1[x] + I bft2[x])/Sqrt[2]
gr1 = Graphics3D[
   Join[{Line[{{0, 0, 0}, {3 Pi, 0, 0}}]}, {Arrowheads[0.02]}, {Red},
    Table[Arrow[{{x, 0, 0}, {x, eft1[x][[1]], eft1[x][[2]]} // Re}], {x,0,3 Pi, 0.3}], {Green},
    Table[Arrow[{{x, 0, 0}, {x, bft1[x][[1]], bft1[x][[2]]} // Re}], {x, 0,3 Pi, 0.3}]]];
gr2 = Graphics3D[
   Join[{Line[{{0, 0, 0}, {3 Pi, 0, 0}}]}, {Arrowheads[0.02]}, {Red},
    Table[Arrow[{{x, 0, 0}, {x, efc[x][[1]], efc[x][[2]]} // Re}], {x, 0,3 Pi, 0.3}], {Green},
    Table[Arrow[{{x, 0, 0}, {x, bfc[x][[1]], bfc[x][[2]]} // Re}], {x, 0,3 Pi, 0.3}]    ]];
Show[GraphicsRow[{gr1, gr2}]]
Export["emplanewave.jpg", GraphicsRow[{gr1, gr2}]]
```

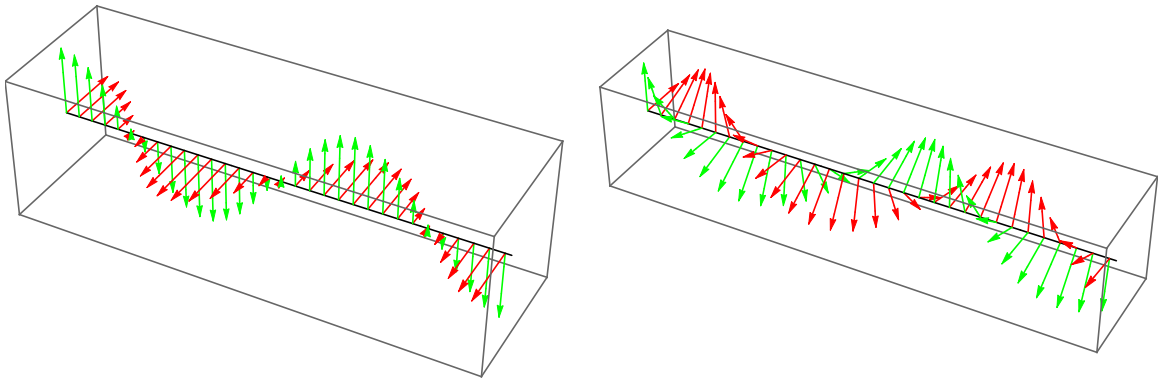


Figure 2.6: $\vec{E}, \vec{B}$ (red,green) at $t = 0$. Left: transversal polarization $\Re sol_I$, Right: circular polarization $\vec{E} = \Re\left(sol_I + i \cdot sol_{II}\right)/\sqrt{2}$.

2.15 Travelling wave

2.15.1 PDE

$$\phi_{tt} - \phi_{xx} = \phi - \phi^3,$$

$$\phi(t,x) = f(x - vt), \qquad (v^2-1)f'' = f - f^3, \qquad f_1(x) = f(x), \quad f_2(x) = f'(x),$$

$$H(f_1,f_2) = \frac{1}{2}f_2^2 + \frac{1}{v^2-1}\left(\frac{1}{4}f_1^4 - \frac{1}{2}f_1^2\right), \qquad \frac{d}{dx}\begin{pmatrix} f_1 \\ f_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \operatorname{grad} H(f_1,f_2).$$

This is a Hamiltonian equation for $\vec{f}$, so $H(f_1(x), f_2(x)) = const.$ on the solution curves. We are mostly interested in the

$$\lim_{x \to \pm\infty} \vec{f}(x) = \vec{0}$$

travelling wave solutions.

```
ClearAll["Global`*"]
H[f1_, f2_, v_] = 1/2 f2^2 + 1/(v^2 - 1) (1/4 f1^4 - 1/2 f1^2);
vs = 2/3; vl = 4/3;
GraphicsGrid[{{
   ContourPlot[ H[f1, f2, vs], {f1, -1.5, 1.5}, {f2, -1, 1}],
   ContourPlot[ H[f1, f2, vl], {f1, -1.5, 1.5}, {f2, -1, 1}]},
  {Plot[H[f1, 0, vs], {f1, -1.5, 1.5}],
   Plot[H[f1, 0, vl], {f1, -1.5, 1.5}]}}]
```

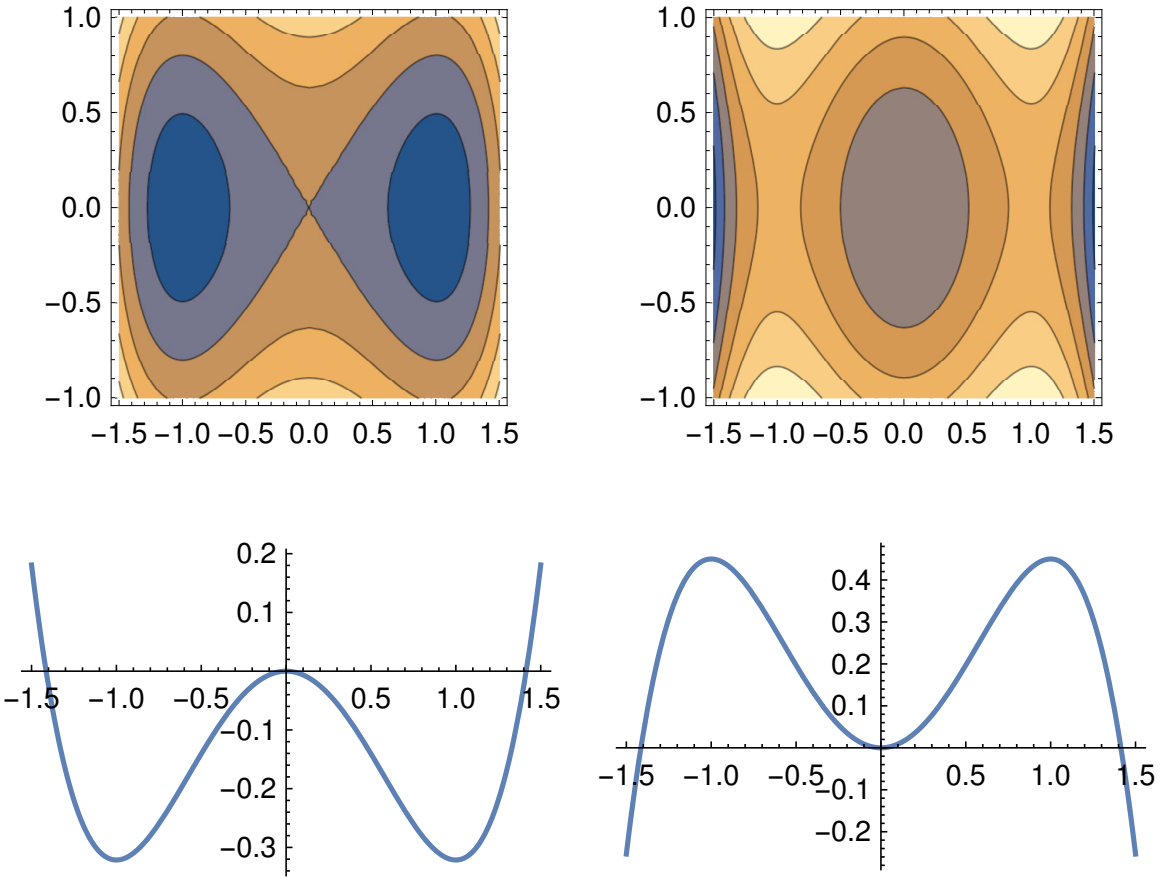


Figure 2.7: Plots of $H(f_1, f_2)$ for $2/3 = v < 1$, $v = 4/3 > 1$. The DE corresponds to Newtonian motions in potential fields.

Travelling wave, topological soliton

```
ClearAll["Global`*"]
H[f1_, f2_, v_] = 1/2 f2^2 + 1/(v^2 - 1) (1/4 f1^4 - 1/2 f1^2);
vs = 2/3; vl = 4/3;
gr = GraphicsGrid[{{ContourPlot[
     H[f1, f2, vl], {f1, -1.5, 1.5}, {f2, -1, 1}],
    ContourPlot[H[f1, f2, vs], {f1, -1.5, 1.5}, {f2, -1, 1}]},
   {Plot[H[f1, 0, vl], {f1, -1.5, 1.5}],
    Plot[H[f1, 0, vs], {f1, -1.5, 1.5}]}}]

 (* travelling wave  |v|>1 *)
f0p = Sqrt[2 Abs[H[1, 0, vl]]];
sol = DSolve[{(vl^2 - 1) f''[x] == f[x] - f[x]^3, f[0] == 1,
    f'[0] == f0p},
   f[x], x][[1, 1, 2]]
sol0 = NDSolve[{(vl^2 - 1) f''[x] == f[x] - f[x]^3, f[0] == 1,
    f'[0] == f0p}, f[x], {x, -5, 5}];
sol1 = NDSolve[{(vl^2 - 1) f''[x] == f[x] - f[x]^3, f[vl] == 1,
    f'[vl] == f0p}, f[x], {x, -5, 5}];
gr1 = Plot[{f[x] /. sol0, f[x] /. sol1}, {x, -5, 5},
  PlotLegends -> {"t=0", "t=1"}]

(* topological soliton |v|<1 *)
f0p = Sqrt[2 Abs[H[1, 0, vs]]];
sol0 = NDSolve[{(vs^2 - 1) f''[x] == f[x] - f[x]^3, f[0] == 0,
    f'[0] == f0p}, f[x], {x, -5, 5}];
sol1 = NDSolve[{(vs^2 - 1) f''[x] == f[x] - f[x]^3, f[vs] == 0,
    f'[vs] == f0p}, f[x], {x, -5, 5}];
gr2 = Plot[{f[x] /. sol0, f[x] /. sol1}, {x, -5, 5},
  PlotLegends -> {"t=0", "t=1"}]
Export["travelling2.eps", GraphicsRow[{gr1, gr2}]]
```

The explicit solution for the travelling wave is

$$f(x) = i\sqrt{2}\coth\left(\frac{1}{7}\left(3\sqrt{7}x - 7\coth^{-1}\left(\frac{1}{\sqrt{2}}\right)\right)\right)\sqrt{1-\tanh^2\left(\frac{1}{7}\left(3\sqrt{7}x - 7\coth^{-1}\left(\frac{1}{\sqrt{2}}\right)\right)\right)}$$

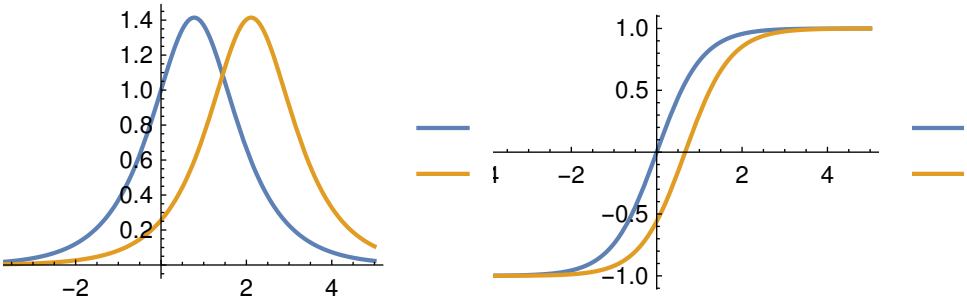


Figure 2.8: Plots of $\phi(t,x)$ for $t = 0,\ 1.$ a) $v = 4/3$, travelling wave, b) $v = 2/3$, topological soliton.

2.16 Shock wave (inviscid Burgers equation)

2.16.1 PDE

$$\phi_t + 7\phi_x = 0,\ \ \phi(0,x) = f(x) \quad \Longrightarrow \quad \phi(t,x) = \phi(0,x-7t) = f(x-7t),\ \ \phi(t,x+7t) = f(x),$$
$$\phi_t + \phi\phi_x = 0,\ \ \phi(0,x) = f(x) \quad \Longrightarrow \quad \phi(t,x+f(x)t) = f(x).$$
$$\vec{\gamma}_0 : \tau \to \begin{pmatrix} \tau \\ f(\tau) \end{pmatrix}, \qquad \vec{\gamma}_t : \tau \to \begin{pmatrix} 1 & t \\ 0 & 1 \end{pmatrix}\begin{pmatrix} \tau \\ f(\tau) \end{pmatrix}$$

2.16.2 Code

```
ClearAll["Global`*"]
f[x_] := Exp[-x^2]
eq = D[phi[t, x], t] + phi[t, x] D[phi[t, x], x] == 0
DSolve[eq, phi, {t, x}]
sol = NDSolve[{eq, phi[0, x] == f[x]}, phi, {t, 0, 1}, {x, -1, 2}];
gr = GraphicsGrid[{{
    Plot[Table[(phi[t, x] /. sol) /. t -> tt, {tt, 0, 1, 0.3}], {x, -1, 2}],
    Plot3D[phi[t, x] /. sol, {t, 0, 1}, {x, -1, 2}]},
   {ParametricPlot[Table[{{1, t}, {0, 1}}.{tau, f[tau]}, {t, 0, 2, 0.3}], {tau, -1, 2}],
    ParametricPlot[{{1, 1.8}, {0, 1}}.{tau, f[tau]}, {tau, -0.4, 2.3},
     Prolog -> {Line[{{1.945, 0.03}, {1.945, 0.93}}]}]}}]
Export["shockWave.eps", gr]
```

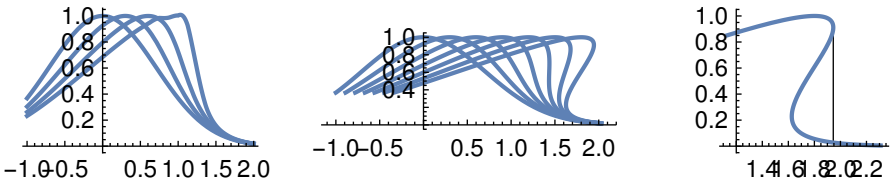


Figure 2.9: The wave propagation velocity equals to its height, so the top part moves faster. That leads to the overtopping of the wave, so the solution ceases to be a classical one. A shock front is formed (vertical line) and the PDE is solved only in the weak sense.

2.17 Discrete Fourier Transform

2.17.1 Definition, examples

DFT

$$\varepsilon = e^{2\pi/N}, \qquad \Delta x = 1/N, \qquad F_{k,l} = \frac{1}{\sqrt{N}}\varepsilon^{-kl}, \qquad F^{-1}_{k,l} = \frac{1}{\sqrt{N}}\varepsilon^{kl}.$$

Code:

```
ClearAll["Global`*"]
CPerm[n_] := Table[If[Mod[j - i, n] == 1, 1, 0], {i, 1, n}, {j, 1, n}]
n = 5; dx = 1/n; eps = Exp[2 Pi I/n]
P = CPerm[n] ; P // TableForm
evals = Table[eps^(k - 1), {k, n}]
Fv = Table[eps^(-(i - 1) (j - 1)), {i, n}, {j, n}];
Fv // TableForm
F = Fv/Sqrt[n];
FInv = Inverse[F];
FInv - Conjugate[Transpose[F]] // N // Chop
```

Command: Fourier

Code:

```
y1[x_] := If[x >= 1/3 && x <= 1/2, 1, 0]
y1vec = Table[y1[(i - 1) dx], {i, n}]/Sqrt[n] // N
y1HatF = F.y1vec // N
y1Hat = Fourier[y1vec]
FInv.y1HatF // Chop
InverseFourier[y1Hat] // Chop
```

Large N behaviour

Code:

```
n = 64; dx = 1/n; a = 1/10; b = 1/2;
y1[x_] :=  If[x >= a && x <= b, 1, 0]
y2[x_] :=  If[x >= a && x <= b, (2/(b - a)) (3/15 - Abs[x - (a + b)/2]), 0]
y3[x_] :=  If[x >= a && x <= b, (1/((b - a)/2))^4 (x - a)^2*(x - b)^2, 0]
ys = {y1, y2, y3};
gr = Table[{}, {i, 3}, {j, 4}];
For[i = 1, i <= 3, i++,
  y = ys[[i]];
   yHat = Fourier[Table[y[(i - 1) dx], {i, n}]];
  gr[[i, 1]] = Plot[y[x], {x, 0, 1}, Ticks -> {{0, 1}, {0, 1}}];
  gr[[i, 2]] =
   ListPlot[Re[yHat], Ticks -> {{0, n}, {0, 1}}, PlotRange -> All];
  gr[[i, 3]] =
   ListPlot[Abs[yHat], Ticks -> {{0, n}, {0, 1}}, PlotRange -> All];
  gr[[i, 4]] =
   ListPlot[Log[Abs[yHat]], Ticks -> {{0, n}, {0, 1}}, PlotRange -> All];
  ];
GraphicsGrid[gr]
```

2.17.2 Discrete diffusion

Code:

```
ClearAll["Global`*"]
n = 6; eps = Exp[2 Pi I/n];
CPerm[n_] :=
 Table[If[Mod[j - i, n] == 1, 1, 0], {i, 1, n}, {j, 1, n}]
A = -2 IdentityMatrix[n] + CPerm[n] + Transpose[CPerm[n]];
A // TableForm
yvec = {y1[t], y2[t], y3[t], y4[t]};
yvecDot = {y1'[t], y2'[t], y3'[t], y4'[t]};
yvec = Table[y[i][t], {i, n}];
yvecDot = Table[y[i]'[t], {i, n}];
de = Table[yvecDot[[i]] == (A.yvec)[[i]], {i, n}];
icondvec = Join[{0.9, 0.1}, Table[0, {n - 2}]];
icond = Map[# == 0 &, (yvec /. t -> 0) - icondvec];
dsol = DSolve[Join[de, icond], yvec, t]
g1 = Plot[yvec /. dsol, {t, 0, 3}, PlotRange -> All];
icondHat = Fourier[icondvec];
```

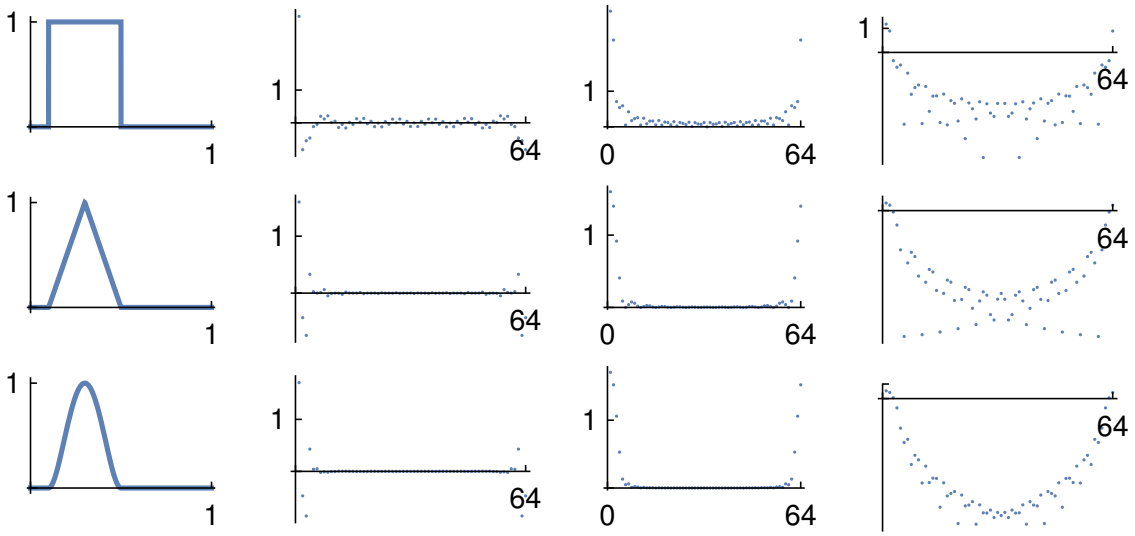


Figure 2.10: DFT of functions: a) with jumps, b) piecewise continuous , c) once differentiable. The smoother the function, the faster the decay rate of the coefficients in the regions k and $N-k$, where $0 \ll k \ll N$. Plotted are: a) function on $[0,1]$ whose samplings produce a vector of dimension N, b) real part of the DFT, c) absolute value of the DFT, d) logarithm of the absolute value.

```
Fsol = DiagonalMatrix[Table[Exp[(eps^k - 2 + eps^(-k)) t],
     {k, 0, n - 1}]].icondHat;
InvF = Table[eps^((i - 1) (j - 1))/Sqrt[n], {i, n}, {j, n}];
g2 = Plot[InvF.Fsol, {t, 0, 3}, PlotRange -> All];
GraphicsRow[{g1, g2}]
```

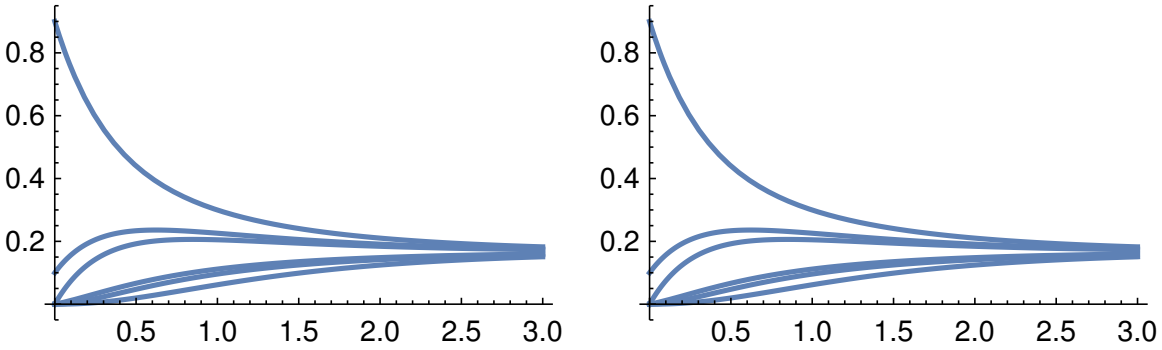


Figure 2.11: A six-state periodic stochastic system, with $w_{\delta t} = 1 \cdot \Delta t$ transition probabilities between neighbouring sites. Plots of the solutions with numerical solution of the DE and with DFT.

2.18 Fourier transform

2.18.1 Fourier series coefficients

Gibbs phenomenon

Code:

```
ClearAll["Global`*"]
a = -1; b = 2;
y1[x_] := If[x >= a && x <= b, 1, 0]
y2[x_] := Which[x >= a && x <= a + (b - a)/2, (2/(b - a)) (x - a),

  x <= b && x >= a + (b - a)/2, -(2/(b - a)) (x - b),
                                True, 0]
y3[x_] :=
 If[x >= a && x <= b, (1/((b - a)/2))^4 (x - a)^2*(x - b)^2, 0]
ys = {y1, y2, y3};
{FourierCoefficient[y1[x], x, n],
```

```
 (1/2 Pi) Integrate[Exp[-I n x] y1[x], {x, -Pi, Pi}]}

y = y1;
nns = {2, 6, 15, 50};
For[i = 1, i <= Length[nns], i++,
  yHatN[i] =
   Table[(1/(2 Pi)) NIntegrate[Exp[-I n x] y[x], {x, -Pi, Pi}],
    {n, -nns[[i]], nns[[i]]}];
  yN[i] = yHatN[i].Table[Exp[I n x], {n, -nns[[i]], nns[[i]]}]
  ];
funs = Prepend[Table[yN[i], {i, Length[nns]}], y[x]];
Plot[funs, {x, -Pi, Pi}, PlotLegends -> Automatic]
Map[Chop[#[[1]]] &, Map[NMaximize[{#, x < Pi, x > -Pi}, x] &, funs]]
"Gibbs Overshot: 8.948..%"
```

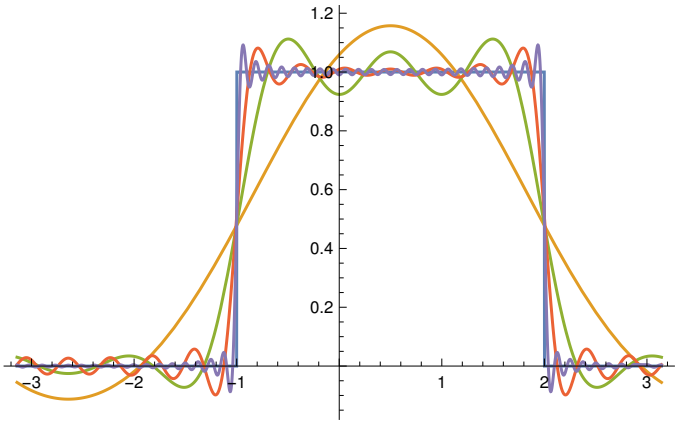


Figure 2.12: Fourier series approximations of the function which is 1 on $[-1, 2]$, otherwise zero. The overshot is 15.7, 11.2, 08.1, 09.2 percentages for $N = 2, 6, 15, 50$. That should tend to 8.948...

Truncated Fourier series

Code:

```
nn = 5;
f1 = FourierSeries[y1[x], x, nn]; g1 =
 Plot[{y1[x], f1}, {x, -Pi, Pi}, Ticks -> None];
f2 = FourierSeries[y2[x], x, nn]; g2 =
 Plot[{y2[x], f2}, {x, -Pi, Pi}, Ticks -> None];
f3 = FourierSeries[y3[x], x, nn]; g3 =
 Plot[{y3[x], f3}, {x, -Pi, Pi}, Ticks -> None];
GraphicsRow[{g1, g2, g3}]
```

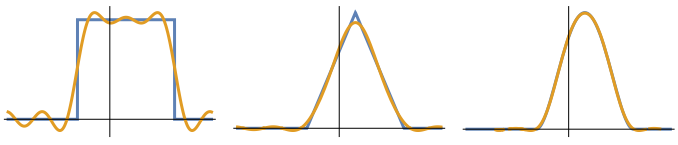


Figure 2.13: $N = 5$ Fourier approximations of functions: a) with jumps, b) continuous and piecewise smooth , c) once differentiable.

Large n behaviour of the Fourier coefficients

Code:

```
min = 3; max = 12;
yLogKLogHatK1 =
  Table[{Log[2^i], Log[Abs[FourierCoefficient[y1[x], x, 2^i]]]}, {i,
     min, max}] // N;
yLogKLogHatK2 =
```

```
  Table[{Log[2^i], Log[Abs[FourierCoefficient[y2[x], x, 2^i]]]}, {i,
     min, max}] // N;
yLogKLogHatK3 =
  Table[{Log[2^i], Log[Abs[FourierCoefficient[y3[x], x, 2^i]]]}, {i,
     min, max}] // N;
g1 = ListPlot[yLogKLogHatK1,
   Ticks -> {{}, {-20, 0}}, PlotRange -> {-20, 0},
   Prolog ->
    Line[{{Log[2^min], -(1/2 + 0) Log[2^min]}, {Log[
        2^max], -(1/2 + 0) Log[2^max]}}]];
g2 = ListPlot[yLogKLogHatK2,
   Ticks -> {{}, {-20, 0}}, PlotRange -> {-20, 0},
   Prolog ->
    Line[{{Log[2^min], -(1/2 + 1) Log[2^min]}, {Log[
        2^max], -(1/2 + 1) Log[2^max]}}]];
g3 = ListPlot[yLogKLogHatK3,
   Ticks -> {{}, {-20, 0}}, PlotRange -> {-20, 0},
   Prolog ->
    Line[{{Log[2^min], -(1/2 + 2) Log[2^min]}, {Log[
        2^max], -(1/2 + 2) Log[2^max]}}]];
GraphicsRow[{g1, g2, g3}]
```

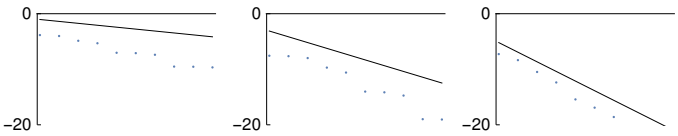


Figure 2.14: Large n behaviour of the Fourier cefficients of functions: a) with jumps, b) continuous and piecewise smooth , c) once differentiable. If $|\hat{f}_n| \sim 1/|n|^c$, then we expect at least a) $c = 0.5$, b) $c = 1.5$, c) $c = 2.5$. The straight lines corresponds to that expectations. (LogLog plots converts powers to straight lines).

Periodic heat equation

Code:

```
ClearAll["Global`*"]
a = -1; b = 2; nn = 10;
y2[x_] := Which[x >= a && x <= a + (b - a)/2, (2/(b - a)) (x - a),
                x <= b && x >= a + (b - a)/2, -(2/(b - a)) (x - b),
                True, 0]
y2Hat = Table[FourierCoefficient[y2[x], x, n], {n, -nn, nn}] // N;
phi = Table[
   Table[Exp[-t n^2] Exp[I n x], {n, -nn, nn}].y2Hat, {t, 0, 1, 0.2}];
phi = Prepend[phi, y2[x]];
GraphicsRow[{Plot[phi, {x, -Pi, Pi}],
  Plot3D[Re[Table[Exp[-t n^2] Exp[I n x], {n, -nn, nn}].y2Hat], {t, 0, 1}, {x, -Pi, Pi}]}]
```

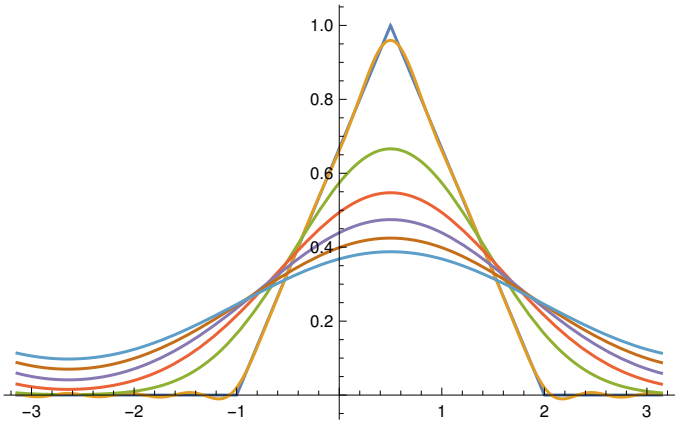


Figure 2.15: Solution of the periodic heat equation.

Dirichlet and Fejer kernels

$$f_N(x) = \sum_{n=-N}^{N} \frac{1}{\sqrt{2\pi}} e^{inx} \left(\int_{-\pi}^{\pi} \frac{1}{\sqrt{2\pi}} e^{-iny} f(y)\, dy \right) = \int_{-\pi}^{\pi} \left(\frac{1}{2\pi} \sum_{n=-N}^{N} e^{in(x-y)} \right) f(y)\, dy$$

$$= \int_{-\pi}^{\pi} D_N(x,y) f(y)\, dy,$$

$$D_N(0,y) = \frac{1}{2\pi} \sum_{n=-N}^{N} e^{-iny} = \frac{1}{2\pi} \frac{\sin\left((N+1/2)y\right)}{\sin((N/2)y)},$$

$$Fejer(0,y) = \frac{1}{N} \sum_{n=0}^{N} D_N(0,y) = \frac{1}{2\pi N} \frac{\sin^2\left((N+1/2)y\right)}{\sin^2((N/2)y)}$$

Code:

```
ClearAll["Global`*"]
dirichletN0y =
 1/(2 Pi) Sum[Exp[-I n y], {n, -nn, nn}] // ExpToTrig // Simplify
dfuns = Table[dirichletN0y /. nn -> n, {n, 5, 40, 15}];
GraphicsRow[Map[Plot[#, {y, -Pi, Pi}, PlotRange -> All] &, dfuns]]

fejerN0y = 1/mm Sum[1/(2 Pi) Sum[Exp[-I n y],
       {n, -nn, nn}], {nn, 0, mm}] // ExpToTrig // Simplify
dfuns = Table[fejerN0y /. mm -> n, {n, 5, 40, 15}];
GraphicsRow[Map[Plot[#, {y, -Pi, Pi}, PlotRange -> All] &, dfuns]]
```

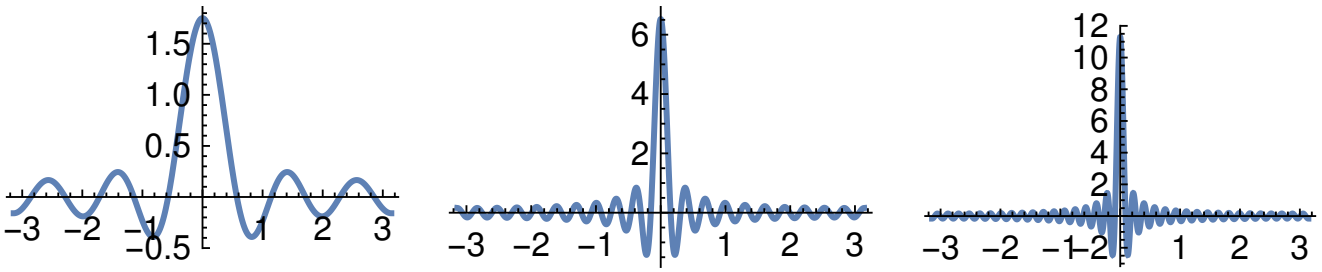


Figure 2.16: Dirichlet kernel.

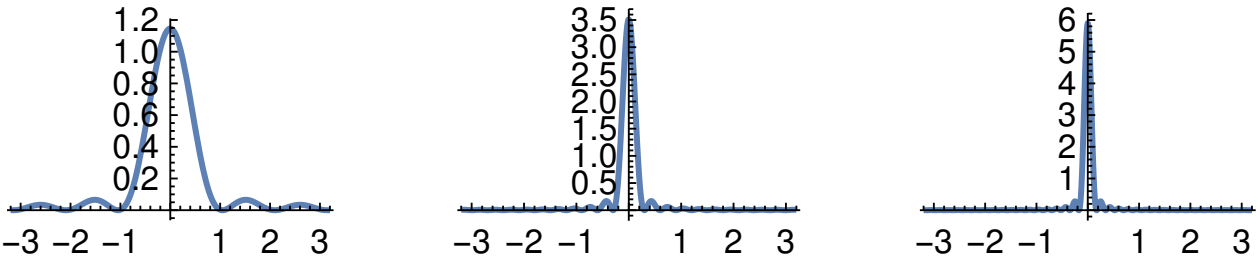


Figure 2.17: Fejér kernel.

2.19 Long term behaviour of dynamical systems, 1d, discrete time

$$x_0, \quad x_1 = f(x_0) = f^1(x_0), \quad x_2 = f(x_1) = f^2(x_0), \quad \dots \quad x_{n+1} = f(x_n) = f^{n+1}(x_0), \quad \dots$$
$$\text{logistic map:} \quad f(x) = rx(1-x), \quad x \in [0,1] \quad r \in [0,4],$$
$$\text{tent map:} \quad f(x) = r(1/2 - |x - 1/2|), \quad x \in [0,1] \quad r \in [0,2]$$

2.19.1 Logistic and tent maps

Iteration

```
f1[x_, r_] := r x (1 - x)
f2[x_, r_] := r (1/2 - Abs[x - 1/2])
CobWeb[f_, x0_, n_] := Module[{seq, gr},
  seq = NestList[f, x0, n];
  gr = Show[Plot[f[x], {x, 0, 1}, PlotRange -> {0, 1}],
    Graphics[Join[{Line[{{0, 0}, {1, 1}}]}, {Line[{{1, 0}, {1, 1}, {0, 1}}]},
      Table[Line[{{seq[[i]], seq[[i]]}, {seq[[i]], seq[[i + 1]]},
                  {seq[[i + 1]], seq[[i + 1]]}}], {i, 1, n}]]]]
gr1 = {Plot[Table[f1[x, r], {r, 2, 4, 0.5}], {x, 0, 1}],
       Plot[Table[f2[x, r], {r, 0.9, 2, 0.5}], {x, 0, 1}]};
gr2 = {CobWeb[f1[#, 3.6] &, 0.2, 6], CobWeb[f2[#, 1.6] &, 0.2, 6]};
gr = GraphicsGrid[{gr1, gr2}]
```

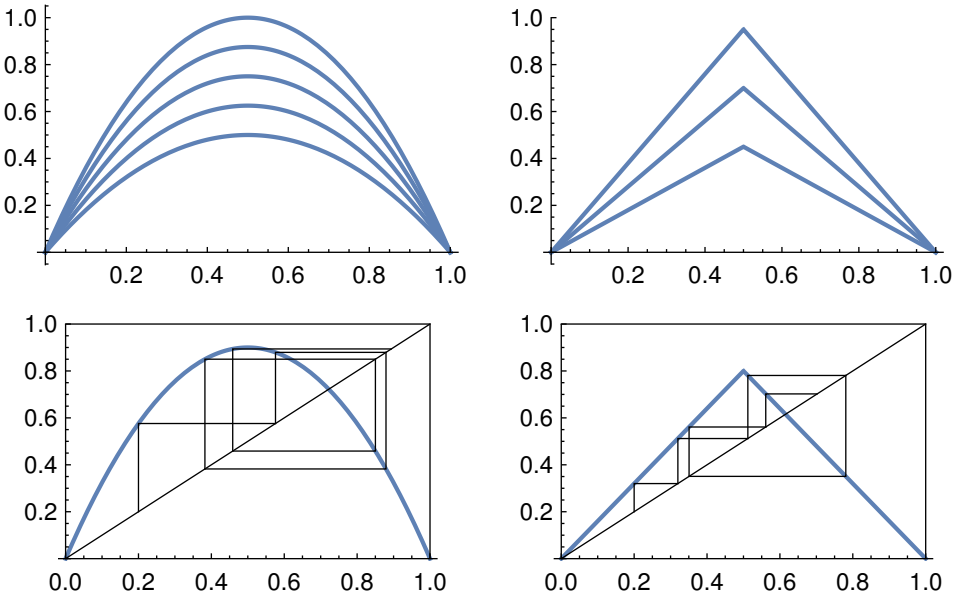


Figure 2.18: Graphical iteration.

Attractor

```
Attractor[f_, ra_, rb_, n_] :=
 Module[{att, k = 100, l = 250, dr = (rb - ra)/n, i, r, seq},
  att = {};
  For[i = 0, i < n, i++,
   r = ra + i dr;
   seq = NestList[f[#, r] &, RandomReal[], l];
   seq = Take[seq, {k, l}];
   att = Join[att, Map[{r, #} &, seq]];   ];
  Graphics[Join[{PointSize[0.0001]}, {Map[Point, att]}]]  ]
gr = GraphicsRow[{Attractor[f1, 3.3, 4, 200], Attractor[f2, 0.8, 2, 200]}]
```

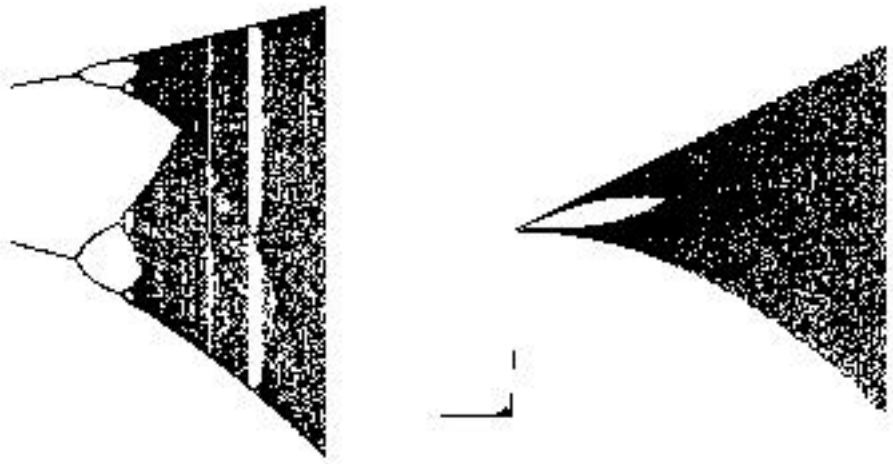


Figure 2.19: Plots of $x_{100} = f^{100}(x_0), \ldots, x_{250}$ for various values of r. Horizontal axis: r, Vertical axis: x_n.

Lyapunov exponent

```
LyapunovExponent[r_] := Module[{k = 100, l = 250, x0, seq},
  x0 = RandomReal[];
  seq = Take[ NestList[Function[x, r x (1 - x)], RandomReal[], l], {k, l}];
  seq = Map[Log[Abs[r (2 # - 1)]] &, seq];
  Return[Plus @@ seq/Length[seq]]
  ]
lyapunov = Table[{r, LyapunovExponent[r]}, {r, 2, 4, 0.001}];
gr1 = ListPlot[lyapunov];
r = 3.7; x0 = 0.3; delta = 10.0^(-9);
seq1 = NestList[Function[x, r x (1 - x)], x0, 60];
seq2 = NestList[Function[x, r x (1 - x)], x0 + delta, 60];
dseq = Map[Abs, seq1 - seq2];
gr2 = ListPlot[dseq, PlotRange -> All];
gr3 = ListPlot[Log[dseq]];
gr = GraphicsRow[{gr1, gr2, gr3}]
Fit[Log[Take[dseq, {10, 30}]], {1, x}, x]
LyapunovExponent[r]
Export["lyapunov.eps", gr]
```

Figure 2.20: a) Lyapunov exponent of the logistic map $f(x) = rx(1-x)$ as a function of r. It is measured as the average of $\ln(|f'(x_n)|)$ for $n = 100, \ldots, 250$. b) $|x_n - \tilde{x}_n|$, where $r = 3.7$, $x_0 = 0.3$, $\tilde{x}_0 = x_0 + 10^{-9}$. c) $\ln(|x_n - \tilde{x}_n|)$. We can estimate the Lyapunov exponent as the slope of a straight line approximation of this plot in a region where $|x_n - \tilde{x}_n|$ is small. The slope in the $n = 10, \ldots, 30$ region is 0.352. Compare this to the more precise value $\lambda = 0.3704$ from the first plot.

2.20 Long term behaviour of dynamical systems, 3d, continuous time

2.20.1 Lorenz equation

$$x' = \sigma(y - x), \qquad y' = x(\rho - z) - y, \qquad z' = xy - \beta z,$$

Code

```
Lorenz[{sigma_, rho_, beta_}, init_] :=
 Block[{T = 50, eq, x, y, z, t, initcond, sol},
  eq = {x'[t] == sigma (y[t] - x[t]),
        y'[t] == x[t] (rho - z[t]) - y[t],
        z'[t] == x[t] y[t] - beta z[t]};
  initcond = {x[0] == init[[1]], y[0] == init[[2]], z[0] == init[[3]]};
  sol = NDSolve[{eq, initcond}, {x[t], y[t], z[t]}, {t, 0, T}];
  Return[{sol,ParametricPlot3D[{x[t], y[t], z[t]} /. sol, {t, 0, T}, Ticks -> None]}]]
plist = {{10, 10, 8/3}, {10, 16, 8/3}, {10, 19.6976, 8/3}, {10, 28, 8/3}};
gr1 = GraphicsRow[Map[Lorenz[#, {10, 10, 10}][[2]] &, plist]]
```

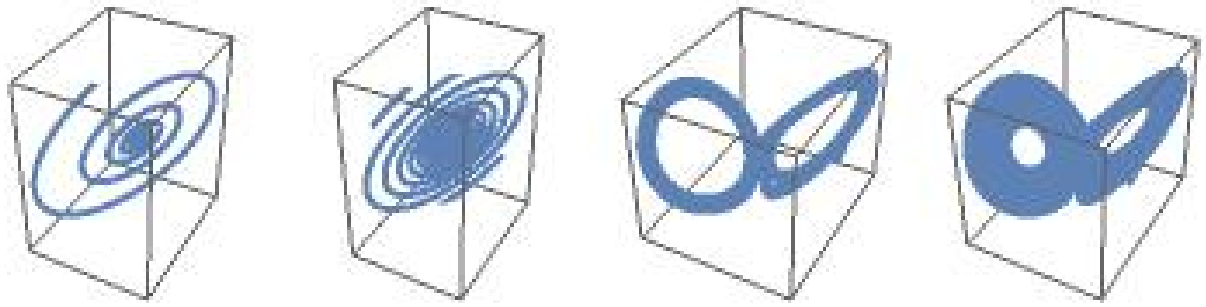


Figure 2.21: Trajectories of the Lorenz system for $\sigma = 10$, $\beta = 8/3$, $\rho = 10, 16, 19.7, 28$.

Sensitivity to the initial condition

Code

```
dLorenz[{sigma_, rho_, beta_}, init_, dinit_, T_] :=
 Block[{eq, x, y, z, t, initcond1, initcond2, sol1, sol2, data, fit, s, gr, dt},
  eq = {x'[t] == sigma (y[t] - x[t]),
        y'[t] == x[t] (rho - z[t]) - y[t],
        z'[t] == x[t] y[t] - beta z[t]};
  initcond1 = {x[0] == init[[1]], y[0] == init[[2]], z[0] == init[[3]]};
  initcond2 = {x[0] == init[[1]] + dinit[[1]],
               y[0] == init[[2]] + dinit[[2]],
               z[0] == init[[3]] + dinit[[3]]};
  sol1 = NDSolve[{eq, initcond1}, {x[t], y[t], z[t]}, {t, 0, T}];
  sol2 = NDSolve[{eq, initcond2}, {x[t], y[t], z[t]}, {t, 0, T}];
  dt = 0.01;
  data = Table[Log[Abs[Norm[
       ({x[t], y[t], z[t]} /. sol1) - ({x[t], y[t], z[t]} /. sol2)]]],{t, 3, 10, dt}];
  fit = Fit[data, {1, t}, t] /. t -> 1/dt t;
  gr = GraphicsRow[{
     Plot[Norm[({x[t], y[t], z[t]} /. sol1) - ({x[t], y[t], z[t]} /. sol2)] ,{t, 0, T}],
     Plot[Log[Abs[Norm[({x[t], y[t], z[t]} /. sol1) - ({x[t], y[t], z[t]} /. sol2)]]] ,
      {t, 0, T}]}];
  Return[{gr, fit}]
  ]
res = dLorenz[{10, 28, 8/3}, {10, 10, 10}, {0, 0, 10^(-6)}, 30];
res[[1]]
res[[2]]
```

Linearization aroud a trajectory

$$\frac{d}{dt}\vec{x_p}(t) = \vec{f}(\vec{x_p}(t)), \qquad \vec{x}(t) = \vec{x_p}(t) + \vec{\delta x}(t), \qquad \vec{\delta x}(t) = U(t)\vec{\delta x}(0),$$

$$\frac{d}{dt}U(t) = Jac(\vec{x_p}(t))U(t), \quad U(0) = E, \quad Jac = \frac{\partial \vec{f}(\vec{x})}{\partial \vec{x}},$$

$$\text{Lyapunov exponent:} \qquad \lambda = \lim_{t\to\infty} \frac{1}{t}\,||U(t)||.$$

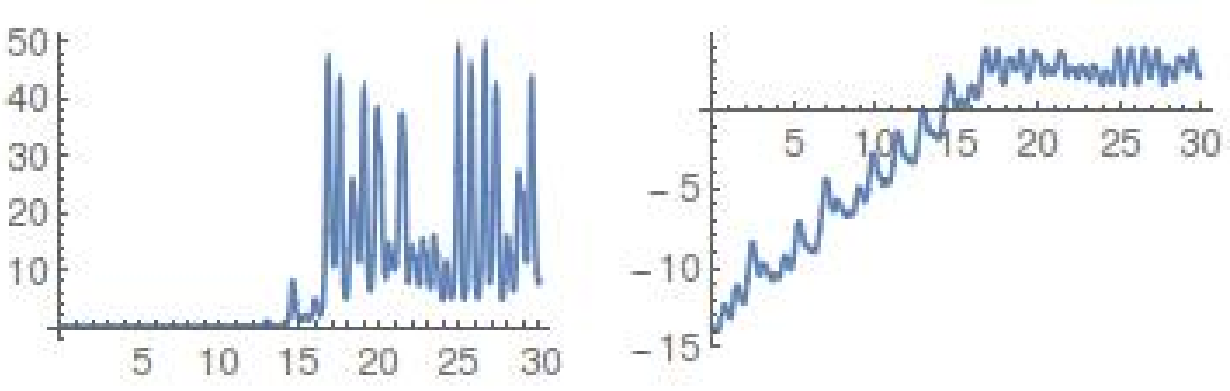


Figure 2.22: The a) distance b) the logarithm of the distance between two trajectories with almost equal initial conditions. The average slope of the second plot in the $[3, 10]$ interval is about 0.957. This number is an approximation of the Lyapunov exponent.

2.20.2 Code

```
f[{x_, y_, z_}, {sigma_, rho_, beta_}] :=
 {sigma (y - x),
  x (rho - z) - y, x y - beta z}
jac[{x_, y_, z_}, {sigma_, rho_, beta_}] :=
  D[f[{x, y, z}, {sigma, rho, beta}], {{x, y, z}}]
jac[{x, y, z}, {10, 28, 8/3}] // TableForm;
init = {10, 10, 10}; T = 50;
psol = Lorenz[{10, 28, 8/3}, init][[1]];
jacpsol[t_] := (jac[{x[t], y[t], z[t]}, {10, 28, 8/3}] /. psol)[[1]];
U[t_] := Table[u[i, j][t], {i, 3}, {j, 3}]
linde = Map[# == 0 &, Flatten[jacpsol[t].U[t] - D[U[t], t]]];
initcond = Map[# == 0 &, Flatten[U[0] - IdentityMatrix[3]]];
usol = NDSolve[Join[linde, initcond], Flatten[U[t]], {t, 0, T}];
gr = GraphicsRow[{
   Plot[Log[Max[Abs[U[t] /. usol]]], {t, 0, T}],
   Plot[Log[Max[Abs[Eigenvalues[U[t] /. usol]]]], {t, 0, T}],
   Plot[1/2 Log[Max[Abs[Eigenvalues[U[t] Transpose[U[t]] /. usol]]]], {t, 0, T}]}]
```

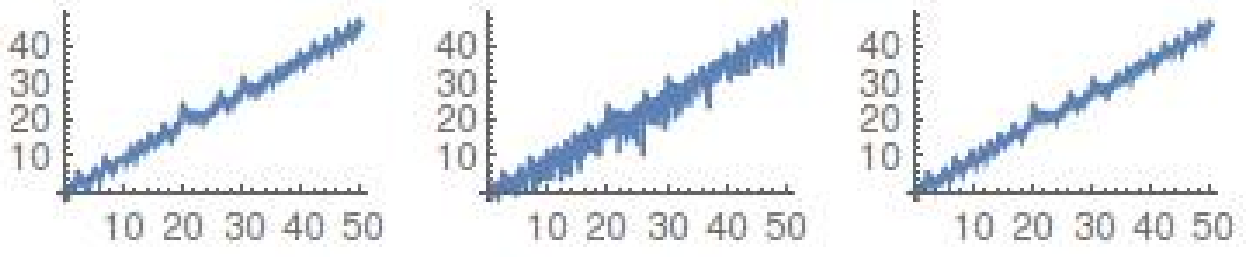


Figure 2.23: The logarithms of the norms of $U(t)$ for a) maximal absolute value of the elements of U, b) maximal absolute value of the eigenvalues, c)$||U||_2$.