Fall 2019

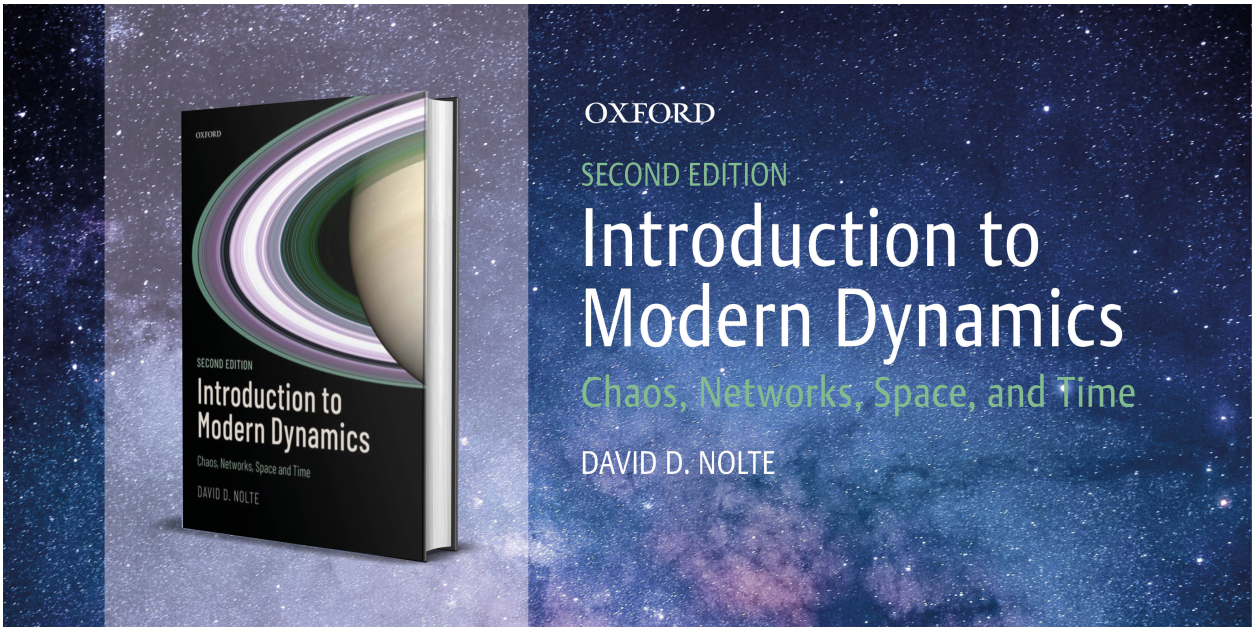
Program codes for Introduction to Modern Dynamics

David D Nolte

Program Codes for

Introduction to Modern Dynamics

Chaos, Networks, Space and Time



Python Scripts for 2D, 3D and 4D Flows

These Python programs can be downloaded from GitHub at
https://github.itap.purdue.edu/nolte/Python-Programs-for-Nonlinear-Dynamics

DampedDriven.py
Driven-damped oscillators. Options are: driven-damped pendulum and driven-damped double well potential. Plots a two-dimensional Poincaré section.

DoublePendulum.py
Double pendulum.

Duffing.py
Duffing oscillator.

DWH.py
Biased double well.

Flow2D.py
Simple flows for 2D autonomous dynamical systems. Options are: Medio, van der Pol, and Fitzhugh-Nagumo models.

Flow2DBorder.py
Same as Flow2D.py but with initial conditions set on the boarder of the phase portrait.

Flow3D.py
Flows for 3D autonomous dynamical systems. Options are: Lorenz, Rössler and Chua's Circuit.

gravlens.py
Gravitational lensing

Hamilton4D.py
Hamiltonian flows for 4D autonomous systems. Options are: Henon-Heiles potential, and the crescent potential. Plots a two-dimensional Poincaré section.

Heiles.py
Henon-Heiles and also a crescent model

HenonHeiles.py
Standalone Henon-Heiles model.

Hill.py

Hill potentials for 3-body problem.

Kuramoto.py
Kuramoto synchronization of phase oscillators on a complete graph.

logistic.py
Logistic discrete map, plus some other choices.

Lozi.py
Discrete iterated Lozi map conserves volume.

NetDynamics.py
Coupled phase oscillators on various network topologies. Has more options than coupleNdriver.py.

NetSIR.py
SIR viral infection model on networks

NetSIRS.py
SIRS viral infection model on networks

PenInverted.py
Inverted pendulum.

Perturbed.py
Driven undampded oscillators with a plane-wave perturbation. Options are: pendulum and double-well potential. These are driven nonlinear Hamiltonian systems. When driven at small perturbation amplitude near the separatrix, chaos emerges. These systems do not conserve energy, because there is a constant input and output of energy as the system reacts against the drive force. Plots a two-dimensional Poincaré section.

raysimple.py
Eikonal equation simulator.

SIR.py
SIR homogeneous COVID-19 model

SIRS.py
SIRS homogeneous COVID-19 model

SIRWave.py
Covid-19 second wave model

StandMap.py
The Chirikov map, also known as the standard map, is a discrete itereated map with winding numbers and islands of stability.

StandMapHom.py
Homoclinic tangle for the standard map.

StandMapTwist.py
The Standard Map in twist format

trirep.py
Replicator dynamics in 3D simplex format.

UserFunction.py
Growing library of user functions
linfit.py – linear regression function

WebMap.py
The discrete map of a periodically kicked oscillator displays a web of dynamics.

(Selected Python programs can be found at the Galileo Unbound Blog Site:
https://galileo-unbound.blog/tag/python-code/)

Flow2D.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Apr 16 07:38:57 2018

@author: David Nolte
"""
import numpy as np
from scipy import integrate
from matplotlib import pyplot as plt

plt.close('all')

# model_case 1 = Medio
# model_case 2 = vdP
# model_case 3 = Fitzhugh-Nagumo
model_case = int(input('Input Model Case (1-3)'))

def solve_flow(param,lim = [-3,3,-3,3],max_time=10.0):

   if model_case == 1:
# Medio 2D flow
      def flow_deriv(x_y, t0, a,b,c,alpha):
      #"""Compute the time-derivative of a Medio system."""
         x, y = x_y
         return [a*y + b*x*(c - y**2),-x+alpha]
      model_title = 'Medio Economics'

   elif model_case == 2:
# van der pol 2D flow
      def flow_deriv(x_y, t0, alpha,beta):
      #"""Compute the time-derivative of a Medio system."""
         x, y = x_y
         return [y,-alpha*x+beta*(1-x**2)*y]
      model_title = 'van der Pol Oscillator'

   else:
# Fitzhugh-Nagumo
      def flow_deriv(x_y, t0, alpha, beta, gamma):
      #"""Compute the time-derivative of a Medio system."""
         x, y = x_y
         return [y-alpha,-gamma*x+beta*(1-y**2)*y]
      model_title = 'Fitzhugh-Nagumo Neuron'
```

```
    plt.figure()
    xmin = lim[0]
    xmax = lim[1]
    ymin = lim[2]
    ymax = lim[3]
    plt.axis([xmin, xmax, ymin, ymax])


    N=144
    colors = plt.cm.prism(np.linspace(0, 1, N))

    x0 = np.zeros(shape=(N,2))
    ind = -1
    for i in range(0,12):
        for j in range(0,12):
            ind = ind + 1;
            x0[ind,0] = ymin-1 + (ymax-ymin+2)*i/11
            x0[ind,1] = xmin-1 + (xmax-xmin+2)*j/11

    # Solve for the trajectories
    t = np.linspace(0, max_time, int(250*max_time))
    x_t = np.asarray([integrate.odeint(flow_deriv, x0i, t, param)
                      for x0i in x0])

    for i in range(N):
        x, y = x_t[i,:,:].T
        lines = plt.plot(x, y, '-', c=colors[i])
        plt.setp(lines, linewidth=1)

    plt.show()
    plt.title(model_title)
    plt.savefig('Flow2D')

    return t, x_t



if model_case == 1:
    param = (0.9,0.7,0.5,0.6)    # Medio
    lim = (-7,7,-5,5)
elif model_case == 2:
    param = (5, 0.5)             # van der Pol
    lim = (-7,7,-10,10)
else:
```

```
    param = (0.02,0.5,0.2)        # Fitzhugh-Nagumo
    lim = (-7,7,-4,4)

t, x_t = solve_flow(param,lim)
```

Flow3D.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Apr 16 07:38:57 2018

@author: nolte
"""
import numpy as np
import matplotlib as mpl
from mpl_toolkits.mplot3d import Axes3D
from scipy import integrate
from matplotlib import pyplot as plt

plt.close('all')
fig = plt.figure()
ax = fig.add_axes([0, 0, 1, 1], projection='3d')
ax.axis('on')

# model_case 1 = Lorenz
# model_case 2 = Rossler
# model_case 3 = Chua
model_case = int(input('Enter Model Case (1-3)'));

def solve_lorenz(param, max_time=8.0, angle=0.0):

    if model_case == 1:
# Lorenz 3D flow
        def flow_deriv(x_y_z, t0, sigma, beta, rho):
        #"""Compute the time-derivative of a Lorenz system."""
            x, y, z = x_y_z
            return [sigma * (y - x), x * (rho - z) - y, x * y - beta * z]
        model_title = 'Lorenz Attractor'

    elif model_case == 2:
# Rossler 3D flow
```

```
        def flow_deriv(x_y_z, t0, sigma, beta, rho):
        #"""Compute the time-derivative of a Medio system."""
            x, y, z = x_y_z
            return [-y-z, x + sigma*y, beta + z*(x - rho)]
        model_title = 'Rossler Attractor'

    else:
# Chua 3D flow
        def flow_deriv(x_y_z, t0, alpha, beta, c, d):
        #"""Compute the time-derivative of a Medio system."""
            x, y, z = x_y_z
            f = c*x + 0.5*(d-c)*(abs(x+1)-abs(x-1))
            return [alpha*(y-x-f), x-y+z, -beta*y]
        model_title = 'Chua Attractor'



    N=12
    colors = plt.cm.prism(np.linspace(0, 1, N))


    # Choose random starting points, uniformly distributed from -15 to 15
    np.random.seed(1)
    x0 = init1 + init2*np.random.random((N, 3))

    # Settle-down Solve for the trajectories
    t = np.linspace(0, max_time/4, int(250*max_time/4))
    x_t = np.asarray([integrate.odeint(flow_deriv, x0i, t, param)
                      for x0i in x0])

    # Solve for trajectories
    x0 = x_t[0:N,int(250*max_time/4)-1,0:3]
    t = np.linspace(0, max_time, int(250*max_time))
    x_t = np.asarray([integrate.odeint(flow_deriv, x0i, t, param)
                      for x0i in x0])


    # choose a different color for each trajectory
    # colors = plt.cm.viridis(np.linspace(0, 1, N))
    # colors = plt.cm.rainbow(np.linspace(0, 1, N))
    # colors = plt.cm.spectral(np.linspace(0, 1, N))
    colors = plt.cm.prism(np.linspace(0, 1, N))

    for i in range(N):
        x, y, z = x_t[i,:,:].T
        lines = ax.plot(x, y, z, '-', c=colors[i])
```

```
        plt.setp(lines, linewidth=0.5)

    ax.view_init(30, angle)
    plt.show()
    plt.title(model_title)
    plt.savefig('Flow3D')

    return t, x_t

if model_case == 1:
    param = (10, 8/3, 28)    # Lorenz
    ax.set_xlim((-25, 25))
    ax.set_ylim((-35, 35))
    ax.set_zlim((5, 55))
    max_time = 50.0
    init1 = -15
    init2 = 30
elif model_case == 2:
    param = (0.2, 0.2, 5.7)        # Rossler
    ax.set_xlim((-15, 15))
    ax.set_ylim((-15, 15))
    ax.set_zlim((0, 20))
    max_time = 200
    init1 = -15
    init2 = 30
else:
    param = (15.6, 28.0, -0.7, -1.14 )        # Chua
    ax.set_xlim((-3, 3))
    ax.set_ylim((-1, 1))
    ax.set_zlim((-3, 3))
    max_time = 100
    init1 = 0
    init2 = 0.1


t, x_t = solve_lorenz(param, max_time,angle=30)

plt.figure(2)
lines = plt.plot(t,x_t[1,:,0],t,x_t[1,:,1],t,x_t[1,:,2])
plt.setp(lines, linewidth=1)

for i in range(4):
    plt.figure(3)
    lines = plt.plot(x_t[i,:,0],x_t[i,:,1])
    plt.setp(lines,linewidth=0.5)
```

```
    plt.figure(4)
    lines = plt.plot(x_t[i,:,1],x_t[i,:,2])
    plt.setp(lines,linewidth=0.5)

    plt.figure(5)
    lines = plt.plot(x_t[i,:,0],x_t[i,:,2])
    plt.setp(lines,linewidth=0.5)
```

DampedDriven.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed May 21 06:03:32 2018

@author: nolte
"""

import numpy as np
import matplotlib as mpl
from mpl_toolkits.mplot3d import Axes3D
from scipy import integrate
from matplotlib import pyplot as plt
from matplotlib import cm
import time
import os

plt.close('all')

# model_case 1 = Pendulum
# model_case 2 = Double Well
model_case = int(input('Enter the Model Case (1-2)'))

if model_case == 1:
    F = 1.2         # 0.6
    delt = 0.5       # 0.1
    w = 2/3         # 0.7
    def flow_deriv(x_y_z,tspan):
        x, y, z = x_y_z
        a = y
        b = F*np.cos(w*tspan) - np.sin(x) - delt*y
        c = w
        return[a,b,c]
else:
```

```
    alpha = -1       # -1
    beta = 1         # 1
    gam = 0.3        # 0.3
    delta = 0.15     # 0.15
    w = 1
    def flow_deriv(x_y_z,tspan):
        x, y, z = x_y_z
        a = y
        b = gam*np.cos(w*tspan) - alpha*x - beta*x**3 - delta*y
        c = w
        return[a,b,c]


T = 2*np.pi/w

px1 = .1
xp1 = .1
w1 = 0

x_y_z = [xp1, px1, w1]

# Settle-down Solve for the trajectories
t = np.linspace(0, 1000, 10000)
x_t = integrate.odeint(flow_deriv, x_y_z, t)
x0 = x_t[9999,0:3]

tspan = np.linspace(1,40000,400000)
x_t = integrate.odeint(flow_deriv, x0, tspan)
siztmp = np.shape(x_t)
siz = siztmp[0]

if model_case == 1:
    y1 = np.mod(x_t[:,0]-np.pi,2*np.pi)-np.pi
    y2 = x_t[:,1]
    y3 = x_t[:,2]
else:
    y1 = x_t[:,0]
    y2 = x_t[:,1]
    y3 = x_t[:,2]



plt.figure(2)
lines = plt.plot(y1,y2,'ko',ms=1)
plt.setp(lines, linewidth=0.5)
plt.show()
```

```
repnum = 5000
px = np.zeros(shape=(2*repnum,))
xvar = np.zeros(shape=(2*repnum,))
cnt = -1
testwt = np.mod(tspan,T)-0.5*T;
last = testwt[1]
for loop in range(2,siz):
   if (last < 0)and(testwt[loop] > 0):
      cnt = cnt+1
      del1 = -testwt[loop-1]/(testwt[loop] - testwt[loop-1])
      px[cnt] = (y2[loop]-y2[loop-1])*del1 + y2[loop-1]
      xvar[cnt] = (y1[loop]-y1[loop-1])*del1 + y1[loop-1]
      last = testwt[loop]
   else:
      last = testwt[loop]


plt.figure(3)
lines = plt.plot(xvar,px,'ko',ms=1)
plt.show()


if model_case == 1:
   plt.savefig(Pendulum)
else:
   plt.savefig(DoubleWell)
```

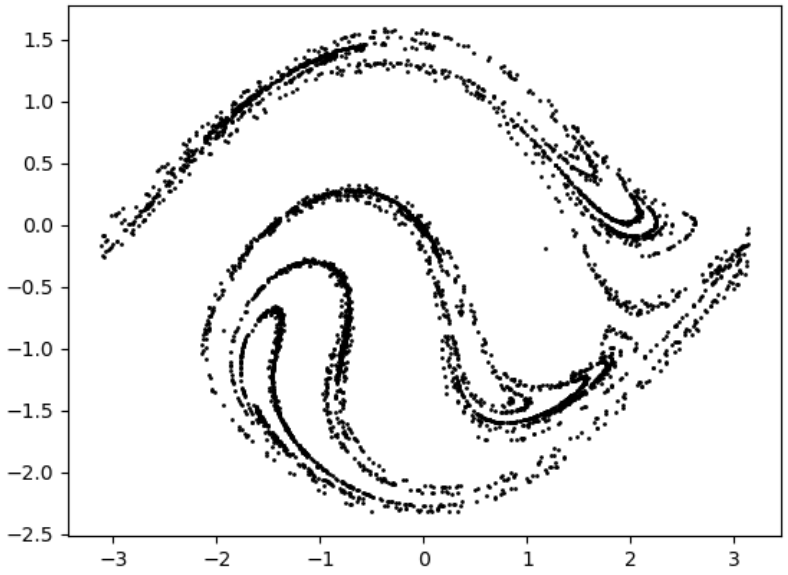


Fig. Driven damped pendulum

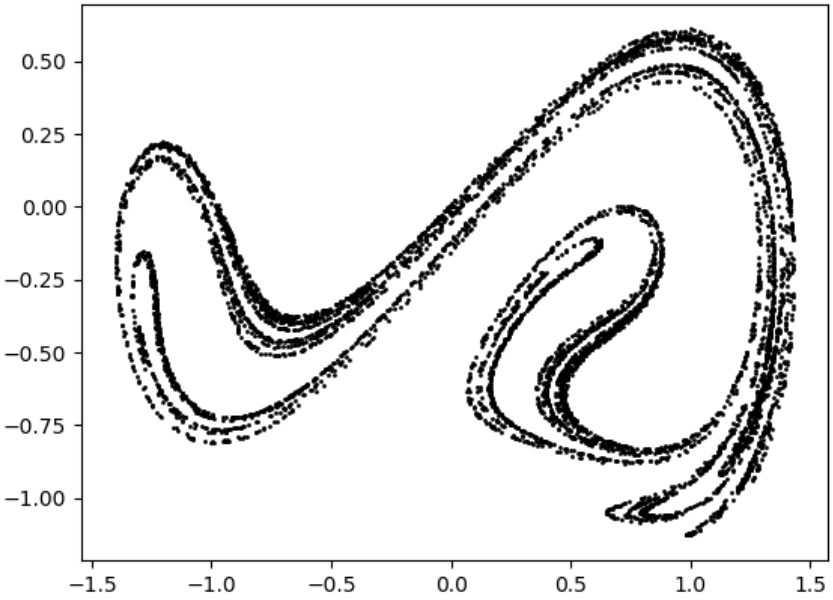


Fig. Driven damped double-well potential.

Hamilton4D.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Apr 18 06:03:32 2018

@author: nolte
"""

import numpy as np
import matplotlib as mpl
from mpl_toolkits.mplot3d import Axes3D
from scipy import integrate
from matplotlib import pyplot as plt
from matplotlib import cm
import time
import os

plt.close('all')

# model_case 1 = Heiles
# model_case 2 = Crescent
model_case = int(input('Enter the Model Case (1-3)'))

if model_case == 1:
    E = 1       # Heiles: 1, 0.3411   Crescent: 0.05, 1
    epsE = 0.3411   # 3411
    def flow_deriv(x_y_z_w,tspan):
        x, y, z, w = x_y_z_w
        a = z
        b = w
        c = -x - epsE*(2*x*y)
        d = -y - epsE*(x**2 - y**2)
        return[a,b,c,d]
else:
    E = .05       # Heiles: 1, 0.3411   Crescent: 0.05, 1
    epsE = 1   # 3411
    def flow_deriv(x_y_z_w,tspan):
        x, y, z, w = x_y_z_w
        a = z
        b = w
        c = -(epsE*(y-2*x**2)*(-4*x) + x)
```

```
        d = -(y-epsE*2*x**2)
        return[a,b,c,d]

prms = np.sqrt(E)
pmax = np.sqrt(2*E)


# Potential Function
if model_case == 1:
    V = np.zeros(shape=(100,100))
    for xloop in range(100):
        x = -2 + 4*xloop/100
        for yloop in range(100):
            y = -2 + 4*yloop/100
            V[yloop,xloop] = 0.5*x**2 + 0.5*y**2 + epsE*(x**2*y - 0.33333*y**3)
else:
    V = np.zeros(shape=(100,100))
    for xloop in range(100):
        x = -2 + 4*xloop/100
        for yloop in range(100):
            y = -2 + 4*yloop/100
            V[yloop,xloop] = 0.5*x**2 + 0.5*y**2 + epsE*(2*x**4 - 2*x**2*y)

fig = plt.figure(1)
contr = plt.contourf(V,100, cmap=cm.coolwarm, vmin = 0, vmax = 10)
fig.colorbar(contr, shrink=0.5, aspect=5)
fig = plt.show()

repnum = 250
mulnum = 64/repnum


np.random.seed(1)
for reploop  in range(repnum):
    px1 = 2*(np.random.random((1))-0.499)*pmax
    py1 = np.sign(np.random.random((1))-0.499)*np.real(np.sqrt(2*(E-px1**2/2)))
    xp1 = 0
    yp1 = 0

    x_y_z_w0 = [xp1, yp1, px1, py1]

    tspan = np.linspace(1,1000,10000)
    x_t = integrate.odeint(flow_deriv, x_y_z_w0, tspan)
    siztmp = np.shape(x_t)
    siz = siztmp[0]
```

```
    if reploop % 50 == 0:
        plt.figure(2)
        lines = plt.plot(x_t[:,0],x_t[:,1])
        plt.setp(lines, linewidth=0.5)
        plt.show()
        time.sleep(0.1)
        #os.system("pause")



    y1 = x_t[:,0]
    y2 = x_t[:,1]
    y3 = x_t[:,2]
    y4 = x_t[:,3]

    py = np.zeros(shape=(2*repnum,))
    yvar = np.zeros(shape=(2*repnum,))
    cnt = -1
    last = y1[1]
    for loop in range(2,siz):
        if (last < 0)and(y1[loop] > 0):
            cnt = cnt+1
            del1 = -y1[loop-1]/(y1[loop] - y1[loop-1])
            py[cnt] = y4[loop-1] + del1*(y4[loop]-y4[loop-1])
            yvar[cnt] = y2[loop-1] + del1*(y2[loop]-y2[loop-1])
            last = y1[loop]
        else:
            last = y1[loop]


    plt.figure(3)
    lines = plt.plot(yvar,py,'o',ms=1)
    plt.show()


if model_case == 1:
    plt.savefig('Heiles')
else:
    plt.savefig('Crescent')
```

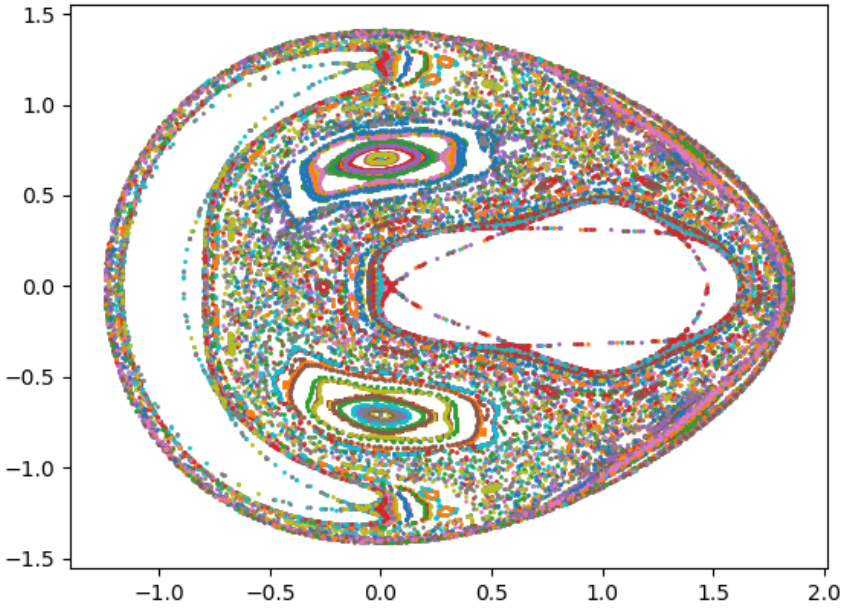
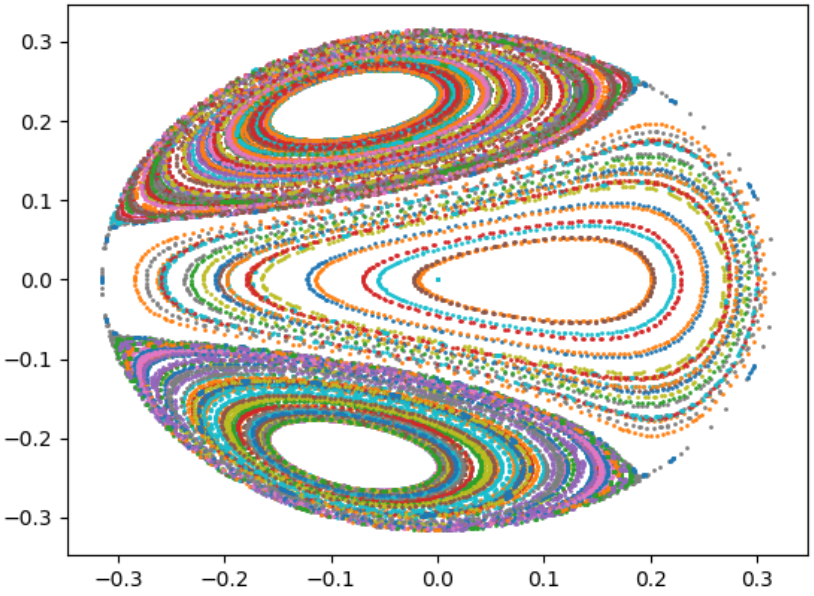


Fig. Heiles



Figl Crescent

Perturbed.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed May 21 06:03:32 2018

@author: nolte
"""
from IPython import get_ipython
get_ipython().magic('reset -f')

import numpy as np
import matplotlib as mpl
from mpl_toolkits.mplot3d import Axes3D
from scipy import integrate
from matplotlib import pyplot as plt
from matplotlib import cm
import time
import os


plt.close('all')


# model_case 1 = Pendulum
# model_case 2 = Double Well
print(' ')
print('DampedDriven.py')
print('Case: 1 = Pendulum 2 = Double Well')
model_case = int(input('Enter the Model Case (1-2)'))

if model_case == 1:
    F = 0.02          # 0.6
    delt = 0.0        # 0.1
    w = 3/4           # 0.7
    k = 2
    phase = 0
    px1 = 1.9635
    xp1 = 0
    w1 = 0
    def flow_deriv(x_y_z,tspan):
        x, y, z = x_y_z
        a = y
        b = F*np.cos(-w*tspan + k*x + phase) - np.sin(x) - delt*y
        c = w
```

```
        return[a,b,c]

else:
    alpha = -1       # -1
    beta = 1         # 1
    F = 0.002        # 0.3
    delta = 0.0    # 0.15
    w = 1
    k = 1
    phase = np.random.random()
    px1 = 0
    xp1 = 0
    w1 = 0
    def flow_deriv(x_y_z,tspan):
        x, y, z = x_y_z
        a = y
        b = F*np.cos(-w*tspan + k*x + phase) - alpha*x - beta*x**3 - delta*y
        c = w
        return[a,b,c]


T = 2*np.pi/w

x_y_z = [xp1, px1, w1]

# Settle-down Solve for the trajectories
t = np.linspace(0, 2000, 20000)
x_t1 = integrate.odeint(flow_deriv, x_y_z, t)
x0 = x_t1[9999,0:3]

tlim = 200000  # number of points
nt = 40000   #stop time
tspan = np.linspace(1,nt,tlim)
x_t = integrate.odeint(flow_deriv, x0, tspan, rtol=1e-8)
siztmp = np.shape(x_t)
siz = siztmp[0]

y1 = np.zeros(shape=(2*tlim,))
y2 = np.zeros(shape=(2*tlim,))
if model_case == 1:
    y1tmp = np.mod(x_t[:,0]-np.pi,2*np.pi)-np.pi
    y2tmp = x_t[:,1]

    y1[0:tlim] = y1tmp
    y1[tlim:2*tlim] = y1tmp+2*np.pi
    y2[0:tlim] = y2tmp
```

```
    y2[tlim:2*tlim] = y2tmp

    y3 = x_t[:,2]
    Energy = 0.5*x_t[:,1]**2 + 1 - np.cos(x_t[:,0])
else:
    y1 = x_t[:,0]
    y2 = x_t[:,1]
    y3 = x_t[:,2]
    Energy = 0.5*x_t[:,1]**2 + 1 + 0.5*alpha*x_t[:,0]**2 + 0.25*beta*x_t[:,0]**4



plt.figure(1)
lines = plt.plot(y1,y2,'ko',ms=1)
plt.setp(lines, linewidth=0.5)
plt.title('Phase Portrait')
plt.show()


plt.figure(2)
lines = plt.plot(y3[0:3000],y2[0:3000])
plt.setp(lines, linewidth=0.5)
plt.title('Velocity')
plt.show()

plt.figure(3)
lines = plt.plot(y3[0:3000],Energy[0:3000])
plt.setp(lines, linewidth=0.5)
plt.title('Energy')
plt.show()


# First-Return Map
repnum = 5000
px = np.zeros(shape=(2*repnum,))
xvartmp = np.zeros(shape=(2*repnum,))
cnt = -1
testwt = np.mod(tspan,T)-0.5*T;
last = testwt[0]
for loop in range(1,siz):
    if (last < 0)and(testwt[loop] > 0):
        cnt = cnt+1
        del1 = -testwt[loop-1]/(testwt[loop] - testwt[loop-1])
        px[cnt] = (y2[loop]-y2[loop-1])*del1 + y2[loop-1]
        xvartmp[cnt] = (x_t[loop,0]-x_t[loop-1,0])*del1 + x_t[loop-1,0]
        #xvar[cnt] = y1[loop]
```

```
        last = testwt[loop]
    else:
        last = testwt[loop]



# Plot First Return Map
if model_case == 1:
    xvar = np.mod(xvartmp-np.pi,2*np.pi)-np.pi
    pxx = np.zeros(shape=(2*cnt,))
    xvarr = np.zeros(shape=(2*cnt,))

    xvarr[0:cnt] = xvar[0:cnt]
    xvarr[cnt:2*cnt] = xvar[0:cnt]+2*np.pi
    pxx[0:cnt] = px[0:cnt]
    pxx[cnt:2*cnt] = px[0:cnt]

    plt.figure(4)
    lines = plt.plot(xvarr,pxx,'ko',ms=0.5)
    plt.xlim(xmin=0, xmax=2*np.pi)
    plt.title('First Return Map')
    plt.show()

    plt.savefig('PPendulum')
else:
    xvar = xvartmp
    plt.figure(4)
    lines = plt.plot(xvar,px,'ko',ms=0.5)
    #mpl.pyplot.xlim(xmin=0, xmax=2*np.pi)
    plt.title('First Return Map')
    plt.show()


    plt.savefig('PDoubleWell')
```

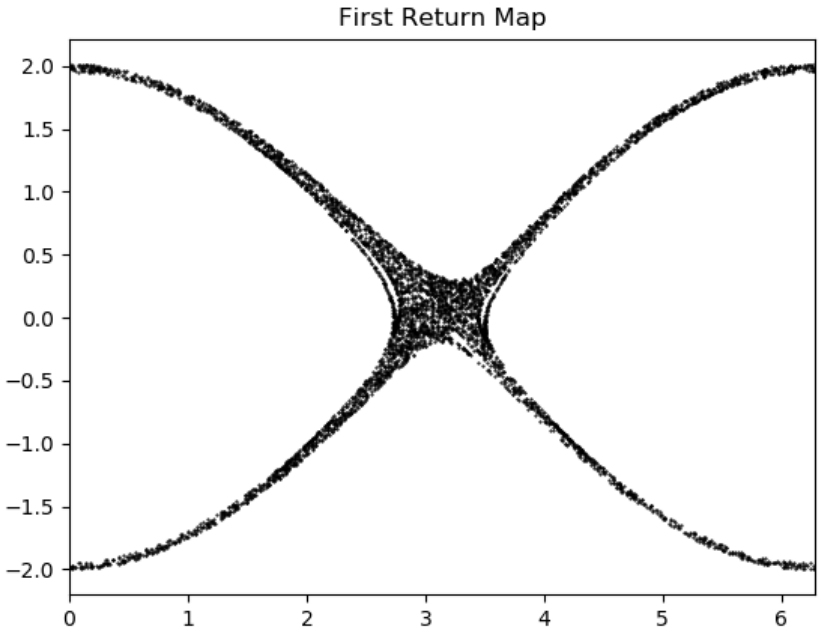


Fig. Perturbed pendulum

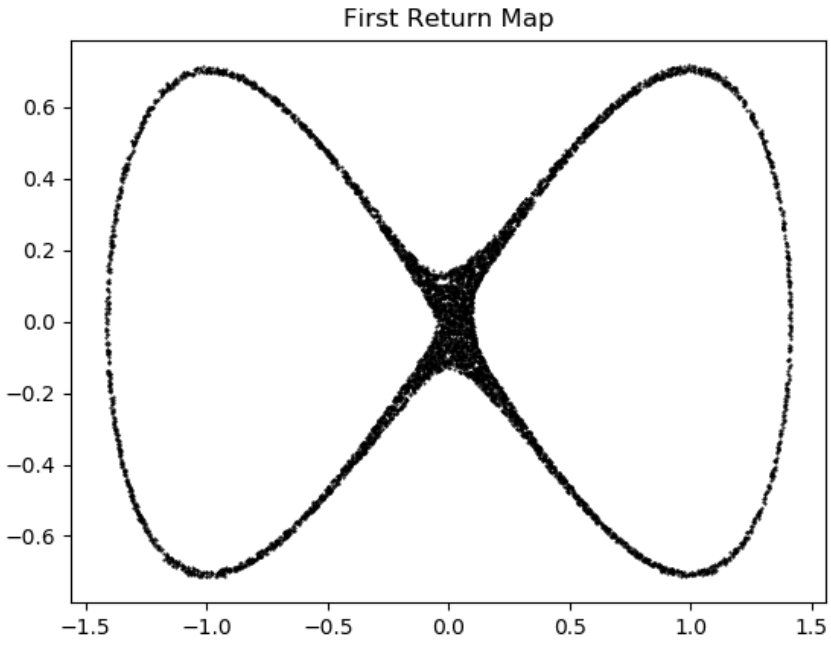


Fig. Perturbed double well

Lozi.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed May  2 16:17:27 2018

@author: nolte
"""

import numpy as np
from scipy import integrate
from matplotlib import pyplot as plt

#plt.close('all')

B = -1
C = 0.5

np.random.seed(2)

plt.figure(1)

for eloop in range(0,100):

    xlast = np.random.normal(0,1,1)
    ylast = np.random.normal(0,1,1)

    xnew = np.zeros(shape=(500,))
    ynew = np.zeros(shape=(500,))
    for loop in range(0,500):
        xnew[loop] = 1 + ylast - C*abs(xlast)
        ynew[loop] = B*xlast
        xlast = xnew[loop]
        ylast = ynew[loop]

    plt.plot(np.real(xnew),np.real(ynew),'o',ms=1)
    plt.xlim(xmin=-1.25,xmax=2)
    plt.ylim(ymin=-2,ymax=1.25)


plt.savefig('Lozi')
```

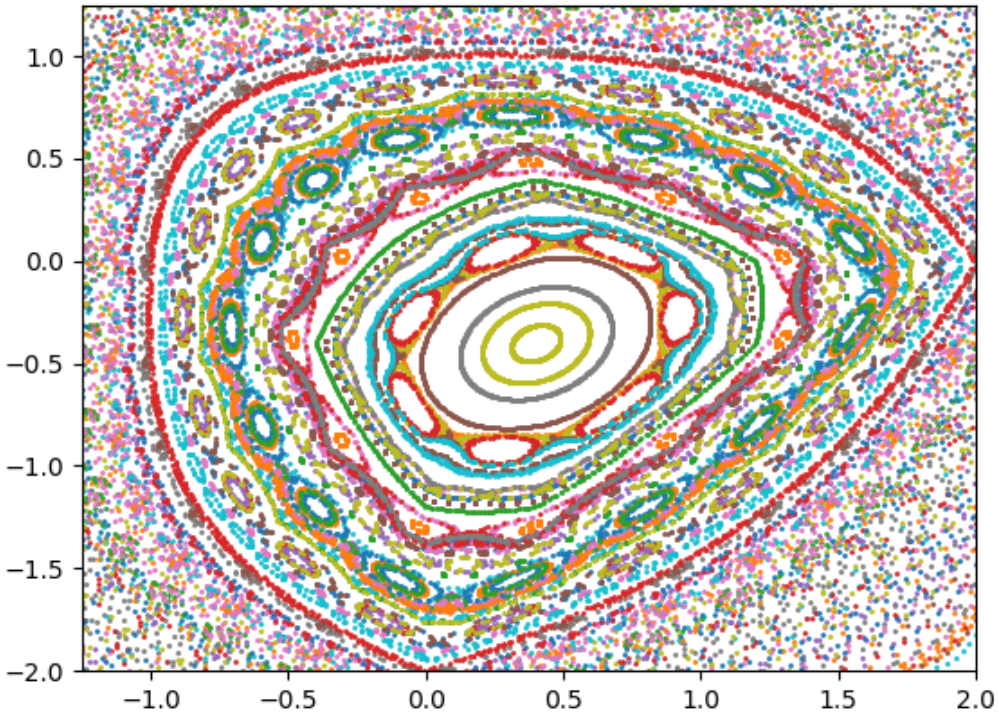


Fig. Lozi map. B = -1, C = 0.5.

StandMap.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed May  2 16:17:27 2018

@author: nolte
"""

import numpy as np
from scipy import integrate
from matplotlib import pyplot as plt

plt.close('all')

eps = 0.97

np.random.seed(2)

plt.figure(1)

for eloop in range(0,200):

    rlast = 2*np.pi*(0.5-np.random.random())
    thlast = 2*np.pi*np.random.random()

    #    rold = 2.0*pi*(0.5-rand);
    #    thetold = 2.0*pi*rand;


    rplot = np.zeros(shape=(200,))
    thetaplot = np.zeros(shape=(200,))
    for loop in range(0,200):
        rnew = rlast + eps*np.sin(thlast)
        thnew = np.mod(thlast+rnew,2*np.pi)

        thetaplot[loop] = np.mod(thnew-np.pi,2*np.pi) - np.pi
        if rnew > np.pi:
            rtemp = rnew-2*np.pi
        elif rnew < -np.pi:
            rtemp = 2*np.pi + rnew
        else:
            rtemp = rnew

        rplot[loop] = np.mod(0.5 + (rtemp + np.pi)/2/np.pi,1)
```

```
        rlast = rnew
        thlast = thnew

    plt.plot(np.real(thetaplot),np.real(rplot),'o',ms=1)
#    plt.xlim(xmin=-np.pi,xmax=np.pi)
#    plt.ylim(ymin=-np.pi,ymax=2*np.pi)


plt.savefig('StandMap')
```

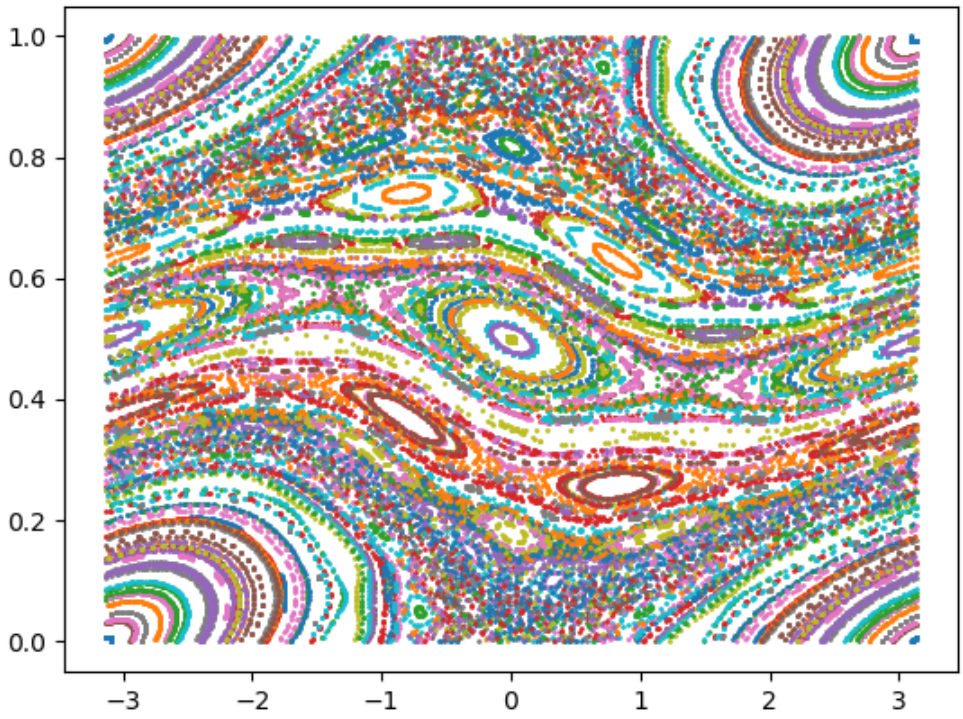


Fig. Standard map. ε = 0.97

WebMap.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed May  2 16:17:27 2018

@author: nolte
"""

import numpy as np
from scipy import integrate
from matplotlib import pyplot as plt

plt.close('all')
phi = (1+np.sqrt(5))/2
K = 1-phi      # (0.618, 4) (0.618,5) (0.618,7) (1.2, 4)
q = 4          # 4, 5, 6, 7
alpha = 2*np.pi/q

np.random.seed(2)

plt.figure(1)

for eloop in range(0,1000):

    xlast = 50*np.random.random()
    ylast = 50*np.random.random()

    xnew = np.zeros(shape=(300,))
    ynew = np.zeros(shape=(300,))
    for loop in range(0,300):
        xnew[loop] = (xlast + K*np.sin(ylast))*np.cos(alpha) + ylast*np.sin(alpha)
        ynew[loop] = -(xlast + K*np.sin(ylast))*np.sin(alpha) + ylast*np.cos(alpha)
        xlast = xnew[loop]
        ylast = ynew[loop]

    plt.plot(np.real(xnew),np.real(ynew),'o',ms=1)
    plt.xlim(xmin=-60,xmax=60)
    plt.ylim(ymin=-60,ymax=60)


plt.title('WebMap')
plt.savefig('WebMap')
```

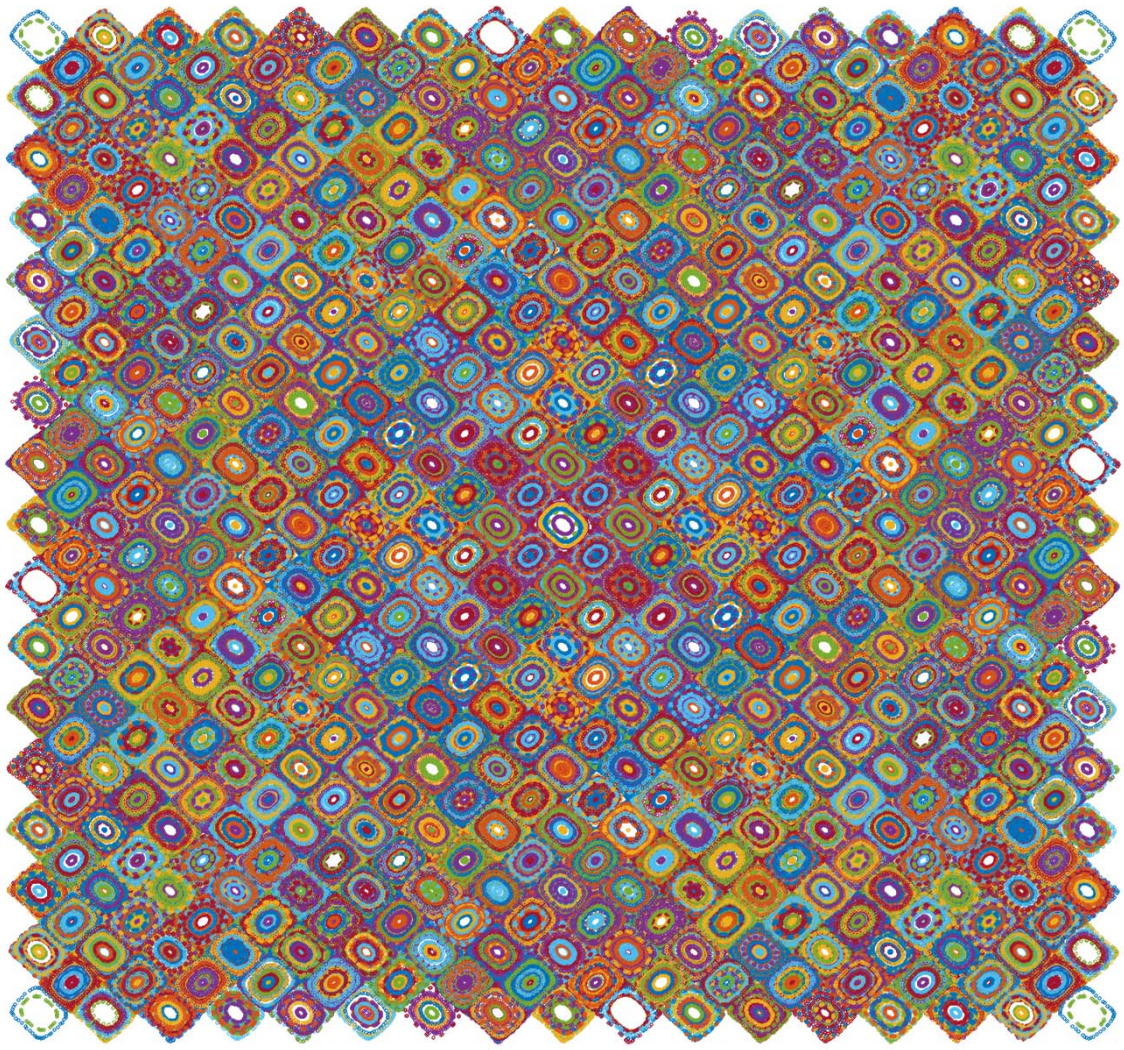
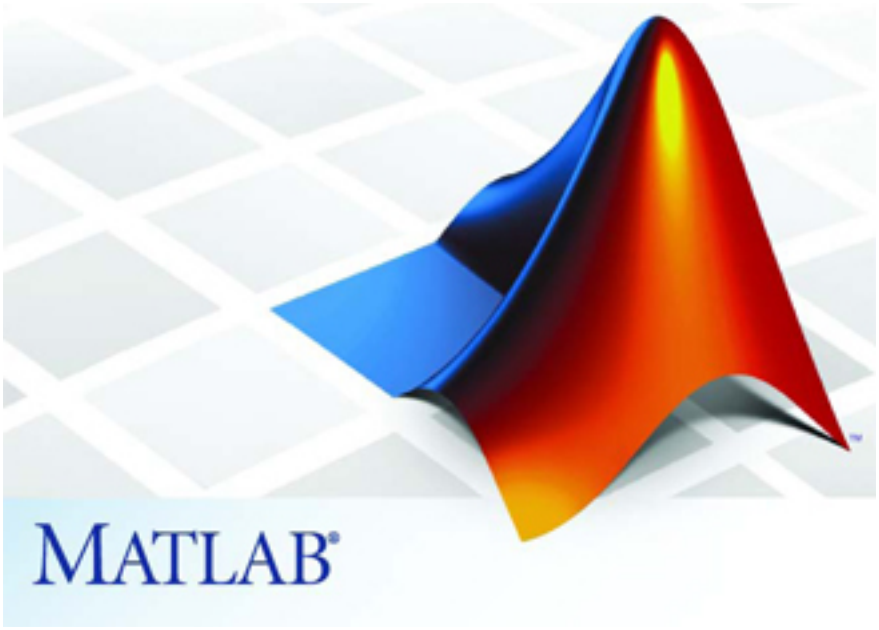


Fig. Web map. $K = \phi - 1$. $q = 4$

MatLab Programs

These programs can help with the computational homework problems at the end of the chapters. They are "scratch" examples meant to be modified by the student, and have only been debugged at a rudimentary level. Use with caution...and expand at will.



Some of these programs can be downloaded from GitHub at

https://github.itap.purdue.edu/nolte/Matlab-Programs-for-Nonlinear-Dynamics

General Functions

aheaviside.m
Cramer.m
cuberoot.m
deriv.m
displine.m
gauss.m
heaviside1.m
heaviside0.m
maskbilevel.m
metropolis.m
newcolormap.m
nonuniformrandom.m
randbin2.m
rowvec.m
where.m
zerodiag.m

aheaviside.m

Asymmetric Heaviside function (y = 0 at x = 0)

```
%function y = aheaviside(x)
% asymmetric Heaviside function
% From -1 to 1. Evaluation at x = 0 is 0
function y = aheaviside(x)

y = -1 + 2*heaviside(x);
```

Cramer.m

Cramers rule is always the best way to solve linear equations of order 2 and higher.

```
% function y = Cramer(M,v)
% Cramer's rule for finding linear solutions
% M is the matrix
% v is the vector

function y = Cramer(M,v)

[sz dum] = size(v);

detM = det(M);
```

```
for loop = 1:sz
    Mp = M;
    Mp(:,loop) = v;

    y(loop) = det(Mp)/det(M);
end
```

cuberoot.m

Roots of general cubic function

```
% function y = cuberoot(a,b,c,d)
% roots of general cubic equation: ax^3 + bx^2 + cx + d = 0

function y = cuberoot(a,b,c,d)

zeta = -1/2 + i*sqrt(3)/2;
delta = 18*a*b*c*d - 4*b^3*d + b^2*c^2 - 4*a*c^3 - 27*a^2*d^2;
delta0 = b^2 - 3*a*c;
delta1 = 2*b^3 - 9*a*b*c + 27*a^2*d;
C = ((delta1+sqrt(-27*a^2*delta))/2)^(1/3);

y(1) = -(1/3/a)*(b + zeta^0*C + delta0/zeta^0/C);
y(2) = -(1/3/a)*(b + zeta^1*C + delta0/zeta^1/C);
y(3) = -(1/3/a)*(b + zeta^2*C + delta0/zeta^2/C);
```

deriv.m

Numerical derivative (computationally unstable on noisy data)

```
% function y = deriv(f)
function y = deriv(f)

[dum sz] = size(rowvec(f));

ytmp = diff(f);

for loop = 2:sz-1
    yav(loop) = (ytmp(loop-1) + ytmp(loop))/2;
end

yav(1) = ytmp(1);
yav(sz) = ytmp(sz-1);

y = yav;
```

displine.m

```
% function displine(txt,val)
function displine(txt,val)
```

```
disp(strcat(txt,num2str(val)))
```

gauss.m

Generic gaussian

```
% Function Gauss(x)
% returns y = exp(-x.^2).

function y=gauss(x)

y = exp(-x.^2);
```

heaviside1.m

Heaviside function with midpoint set equal to 1.

```
% function y = heaviside1(x)
% y = 1 at x = 0
function y = heaviside1(x)
[sy, sx] = size(x);
for yloop = 1:sy
    for xloop = 1:sx

        if x(yloop,xloop)<0
            y(yloop,xloop) = 0;
        elseif x(yloop,xloop) == 0
            y(yloop,xloop) = 1;
        elseif x(yloop,xloop) > 0
            y(yloop,xloop) = 1;
        end
    end
end
```

heaviside0.m

Heaviside function with midpoint set equal to 0.

```
% function y = heaviside0(x)
% y = 0 at x = 0
function y = heaviside0(x)
[sy, sx] = size(x);
for yloop = 1:sy
    for xloop = 1:sx

        if x(yloop,xloop)<0
            y(yloop,xloop) = 0;
        elseif x(yloop,xloop) == 0
            y(yloop,xloop) = 0;
        elseif x(yloop,xloop) > 0
```

```
            y(yloop,xloop) = 1;
        end
    end
end
```

maskbilevel.m

Set a binary mask based on hi and lo values

```
% maskbilevel.m
% function [y mask] = masklevel(A,lowcut,hicut,vallo,valhi)
% Assigns values vallo and valhi to outside of cuts
% mask is binary 1's where inside cutvalues
function [y mask] = maskbilevel(A,lowcut,hicut,vallo,valhi)
% slope = 0.00001*(hicut-lowcut);
%mtemp = clip(A,lowcut,hicut,slope);
mtemp = heaviside0(A-lowcut).*heaviside1(hicut-A);
y = A.*mtemp + valhi*heaviside1(A-hicut).*(1-mtemp) +
vallo*heaviside0(lowcut-A).*(1-mtemp);
mask = mtemp;
```

metropolis.m

Generate a random sampling from a defined distribution function

```
% function x = metropolis(N,F,xmax)
% Metropolis algorithm for general probabilistic distribution functions
% N is number of samples
% F is called as @function_name with single input variable
% For example ...
% v = metropolis(N,@cauchyfcn,100); % Cauchy distribution

function x = metropolis(N,F,xmax)

x(1) = xmax*(2*(0.5-rand));
ind = 1;
while ind < 20
    ind = ind + 1;
    xstar = xmax*(2*(0.5-rand));      % the domain is -xmax to +xmax

    ucomp = min(1,F(xstar)/F(x(ind-1)));
    u = rand;

    if u < ucomp
        x(ind) = xstar;
    else
        x(ind) = x(ind-1);
    end

end

x(1) = x(ind);
ind = 1;
```

```
while ind < N
    ind = ind + 1;
    xstar = xmax*(2*(0.5-rand));      % the domain is -xmax to +xmax

    ucomp = min(1,F(xstar)/F(x(ind-1)));
    u = rand;

    if u < ucomp
        x(ind) = xstar;
    else
        x(ind) = x(ind-1);
    end
end
% function y = cauchyfcn(x)
% y = (1/pi)./(x.^2 + 1);
function y = cauchyfcn(x)
n = 2.;
y = (1/pi)./(abs(x).^n + 1);
```

newcolormap.m

New colormaps

```
% function y = newcolormap(name)
% name = 'heated'
% name = 'graycolor'
% name = 'green'
% name = 'whitegreen'
% name = 'red'
% name = 'whitered'
% name = 'blue'
% name = 'whiteblue'
% name = 'darkjet'
% name = 'bandjet'
% name = 'fluoro'
% name = 'jetwhite'
% name = 'rgmerge'

function y = newcolormap(name)

if strncmp(name,'heated',6)

    % Set colormap for red/blue
    for loop = 1:32
        h(loop,1) = loop/32;
        h(loop,2) = loop/32;
        h(loop,3) = 1;
    end
    for loop = 33:64
        h(loop,1) =1;
        h(loop,2) = 1- (loop-33)/32;
        h(loop,3) = 1-(loop-33)/32;
    end
```

```
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%
elseif strncmp(name, 'darkheat',8)
    for loop = 1:32
        h(loop,1) = 0;
        h(loop,2) = 0;
        h(loop,3) = 1 - loop/32;
    end
    for loop = 33:64
        h(loop,1) =(loop-33)/32;
        h(loop,2) = 0;
        h(loop,3) = 0;
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%
elseif strncmp(name, 'graycolor',6)
    temp = colormap(gray);
    for loop = 1:64
        h(loop,1) = 1-temp(loop,1);
        h(loop,2) = 1-temp(loop,2);
        h(loop,3) = 1-temp(loop,3);
    end
    %h(127,:) = [0 0 1];
    %h(256,:) = [1 0 0];

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%
elseif strncmp(name,'green',5)
    h = zeros(64,3);
    for loop = 1:64
        h(loop,2) = (loop-1)/63;
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%
elseif strncmp(name,'whitegreen',7)
    h = ones(64,3);
    for loop = 33:64
        h(loop,1) = (64-loop)/31;
        h(loop,3) = (64-loop)/31;
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%
elseif strncmp(name,'red',3)
    h = zeros(64,3);
    for loop = 1:64
        h(loop,1) = (loop-1)/63;
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%
elseif strncmp(name,'whitered',7)
    h = ones(64,3);
    for loop = 33:64
        h(loop,2) = (64-loop)/31;
        h(loop,3) = (64-loop)/31;
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%
elseif strncmp(name,'blue',3)
    h = zeros(64,3);
```

```
    for loop = 1:64
        h(loop,3) = (loop-1)/63;
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%
elseif strncmp(name,'whiteblue',7)
    h = ones(64,3);
    for loop = 33:64
        h(loop,1) = (64-loop)/31;
        h(loop,2) = (64-loop)/31;
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%
elseif strncmp(name,'darkjet',6)
    h = colormap(jet);
    for loop = 1:64
        h(loop,1) = (loop-1)/63;
        h(loop,3) = (64-loop)/63;
    end
    for loop = 1:32
        h(loop,2) = (loop-1)/31;
        h(32+loop,2) = (32-loop)/32;
    end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%
elseif strncmp(name,'bandjet',6)
    for loop = 1:64
        h(2*loop-1,1) = 0.5*(cos(1*pi*(loop-1)/63 + 2.0*pi/3) + 1);
        h(2*loop-1,2) = 0.5*(cos(1*pi*(loop-1)/63 + 4.0*pi/3) + 1);
        h(2*loop-1,3) = 0.5*(cos(1*pi*(loop-1)/63 + 0.0*pi/3) + 1);

        h(2*loop,1) = 0.5*(cos(-1*pi*(loop-1)/63 + 2.0*pi/3) + 1);
        h(2*loop,2) = 0.5*(cos(-1*pi*(loop-1)/63 + 4.0*pi/3) + 1);
        h(2*loop,3) = 0.5*(cos(-1*pi*(loop-1)/63 + 0.0*pi/3) + 1);
    end

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%
elseif strncmp(name,'fluoro',6)
    for loop = 1:64
        t(loop,1) = (0.5*(cos(2*pi*(loop-1)/63 + 2.0*pi/3) + 1))^1;
        t(loop,2) = (0.5*(cos(2*pi*(loop-1)/63 + 4.0*pi/3) + 1))^1;
        t(loop,3) = (0.5*(cos(2*pi*(loop-1)/63 + 0.0*pi/3) + 1))^1;
    end
    intens = sum(t,2);
        h(:,1) = 1.49*t(:,1)./intens;
        h(:,2) = 1.49*t(:,2)./intens;
        h(:,3) = 1.49*t(:,3)./intens;


            %%%%%%%%%%%%%%%%%%%%%%%%%%%%
elseif strncmp(name,'jetwhite',6)
    x = 1:16;
    y = 1:24;
    z = 1:8;
    h = zeros(64,3);

%     % red
```

```
%     h(9:32,1) = y'/24;
%     h(33:48,1) = ones(16,1);
%     h(49:64,1) = 1 - x'/24;
%     % green
%     h(1:24,2) = y'/24;
%     h(25:40,2) = ones(16,1);
%     h(41:64,2) = 1-y'/24;
%     % blue
%     h(1:16,3) = (x'+ 8)/24;
%     h(17:32,3) = ones(16,1);
%     h(33:56,3) = 1 - y'/24;

    % red
    h(17:32,1) = x'/16;
    h(33:56,1) = ones(24,1);
    h(57:64,1) = 1 - (z'-1)/16;
%     % green
%     h(9:24,2) = x'/16;
%     h(25:40,2) = ones(16,1);
%     h(41:56,2) = 1-x'/16;
    % green
    h(1:16,2) = x'/16;
    h(17:40,2) = ones(24,1);
    h(41:56,2) = 1-(x'-1)/16;
    % blue
    h(1:8,3) = (z'+ 8)/16;
    h(9:32,3) = ones(24,1);
    h(33:48,3) = 1 - (x'-1)/16;
%     % blue
%     h(1:24,3) = (y'+ 12)/36;
%     h(25:32,3) = ones(8,1);
%     h(33:48,3) = 1 - (x'-1)/16;

elseif strncmp(name,'rgmerge',6)
    x = 1:32;
    y = 1:16;
    h = zeros(64,3);

    h(1:16,1) = zeros(16,1);
    h(1:16,2) = 0.5 + y'/32;

    h(17:32,1) = y'/16;
    h(17:32,2) = ones(16,1);

    h(33:48,1) = ones(16,1);
    h(33:48,2) = 1 - y'/16;

    h(49:64,1) = 1 - y'/32;
    h(49:64,2) = zeros(16,1);

    h(1,:) = ones(1,3);

else
    disp('Not specified in newcolormap')
    h = colormap(jet);
end
```

```
y = h;
```

nonuniformrand.m

Whole number between 1 and N with prob given by wt(N)

```
% function nur = nonuniformrand(N,wt)
% wt is size N vector with sum(wt) = 1
% output is whole number between 1 and N with prob given by wt(N)

function nur = nonuniformrand(N,wt)

[Y,I] = sort(wt);

S(1) = wt(I(1));
for loop = 2:N
    S(loop) = S(loop-1) + wt(I(loop));
end

R = rand;

nur = 0;
flag = 1;
i = 1;
while flag == 1
    if max(S(i),R) == R
        i = i+1;
    else
        nur = I(i);
        flag = 0;
    end

end % end while flag
```

randbin2.m

Random binary number generator

```
% function y = randbin2(M,N,thresh)
% random binary matrix of length M by N
% thresh between 0 and 1
% p(1) = 1-thresh
% p(0) = thresh
function y = randbin(M,N,thresh)
temp = rand(M,N);
y = zeros(M,N);
for mloop = 1:M
    for nloop = 1:N
        if temp(mloop,nloop) > thresh
            y(mloop,nloop) = 1;
```

```
        end
    end
end
```

rowvec.m

Make sure a vector is a row vector

```
% function y = rowvec(a)
% guarantees/converts a vector as a row vector

function y = rowvec(a)

[sy sx] = size(a);
if (sy~=1)&(sx~=1)
    disp('a is not a vector (from rowvec.m)')
end

if sx == 1
    y = a';
else
    y = a;
end
```

where.m

Find where a number is located in a vector

```
% function [w,N] = where(invec,num)
% finds where num is located in 1D vector invec
% returns all N instances
% See also where2.m for 2D matrix

function [w,N] = where(invec,num)

[sy,sx] = size(invec);
if sx == 1
    invec = invec';
end
[dum,sz] = size(invec);

w = -1;
ind = 0;
for loop = 1:sz
    if num == invec(loop);
        ind = ind+1;
        w(ind) = loop;
    end
end
N = ind;

[dum,Ntemp] = size(w);
if (Ntemp ==1&w==-1)
```

```
    N = 0;
end
```

zerodiag.m

Zero the diagonal of a matrix

```
% function B = zerodiag(A)
% Sets diagonal terms to zero

function B = zerodiag(A,altmin,varargin)

[sy,sx] = size(A);
B = A;
if sy~=sx
    disp('in zerodiag not square');
    B = A;
else
    if nargin == 1
        for loop = 1:sx
            B(loop,loop) = 0;
        end
    else
        for loop = 1:sx
            B(loop,loop) = altmin;
        end
    end
end
```

Chapter 1

The geometric spiral of Problem 1.6 can be solved numerically very easily (but work at the analytical solution for good practice)

gspiral.m

```
clear
format compact

b = 0.2;
a = 1;
r0 = 1;
dtheta = 0.01;

theta = 0:dtheta:6*pi;
r = r0*exp(b*theta);
%r = r0*theta;
x = r.*cos(theta);
y = r.*sin(theta);
z = a*theta;

Norm = sqrt(a^2 + (b^2 + 1)*r.^2);
ds = Norm.*deriv(theta);

% The Norm can be obtained for general r(theta) using
% xp = deriv(x)./deriv(theta);
% yp = deriv(y)./deriv(theta);
% zp = deriv(z)./deriv(theta);
% Norm2 = (xp.^2 + yp.^2 + zp.^2);
% ds = Norm2.*deriv(theta);


figure(1)
plot3(x,y,z)

figure(2)
plot(x,y);

% Tangent
Tx = deriv(x)./ds;
Ty = deriv(y)./ds;
Tz = deriv(z)./ds;

T2 = Tx.^2 + Ty.^2 + Tz.^2;

figure(3)
plot(theta,Tx,theta,Ty,theta,Tz)
title('Tangent')
legend('x','y','z')

% Normal
kNx = deriv(Tx)./ds;
kNy = deriv(Ty)./ds;
kNz = deriv(Tz)./ds;
```

```
k2 = kNx.^2 + kNy.^2 + kNz.^2;
k = sqrt(k2);

Nx = kNx./k;
Ny = kNy./k;
Nz = kNz./k;

figure(4)
plot(theta,Nx,theta,Ny,theta,Nz)
title('Normal')
legend('x','y','z')

T(1,:) = Tx;
T(2,:) = Ty;
T(3,:) = Tz;

N(1,:) = Nx;
N(2,:) = Ny;
N(3,:) = Nz;

% Binormal
B = cross(T,N);

Bx = B(1,:);
By = B(2,:);
Bz = B(3,:);

figure(5)
plot(theta,Bx,theta,By,theta,Bz)
title('BiNormal')
legend('x','y','z')
```

Chapter 2

body3.m
RTB.m

Three-body problem for Sun, Earth and Jupiter

body3.m

```
function body3

clear

chsi = 0.001;
wj = 2*pi/11.86;
rj = 5.203;

rsun = chsi*rj/(1+chsi);
rjup = (1/chsi)*rj/(1+1/chsi);

% Earth: r0 = 1;
% Mars: r0 = 1.524;
% Asterpods: r0 = 2.92;
% 2:1 resonance r0 = 3.15
% 3:1 resonance r0 = 2.405
% 3:2 r0 = 3.82
% 5:1 r0 = 1.1
% Try chsi = 0.989 and r0 = 1.1
% Try chsi = 0.9091 and r0 = 1.1

r0 = 1-rsun;
y0 = [r0 0 0 2*pi/sqrt(r0)];

tspan = [0 150];
options = odeset('RelTol',1e-5,'AbsTol',1e-6);
[t,y] = ode45(@f5,tspan,y0,options);

figure(1)
plot(t,y(:,1),t,y(:,3))

figure(2)
plot(y(:,1),y(:,3),'k')
axis equal
axis([-6 6 -6 6])

print -dtiff -r800 threebody


    function yd = f5(t,y)
```

```
        xj = rjup*cos(wj*t);
        yj = rjup*sin(wj*t);
        xs = -rsun*cos(wj*t);
        ys = -rsun*sin(wj*t);
        rj32 = ((y(1) - xj).^2 + (y(3) - yj).^2).^1.5;
        r32 = ((y(1) - xs).^2 + (y(3) - ys).^2).^1.5;

        yp(1) = y(2);
        yp(2) = -4*pi^2*((y(1)-xs)/r32 + chsi*(y(1)-xj)/rj32);
        yp(3) = y(4);
        yp(4) = -4*pi^2*((y(3)-ys)/r32 + chsi*(y(3)-yj)/rj32);

        yd = [yp(1);yp(2);yp(3);yp(4)];

    end     % end f5

end
```

Restricted Three-body problem and Lagrange points

RTB.m

```
% RTB.m Restricted 3-Body problem
% Equipotentials and Lagrange points

clear
format compact

M1 = 100;
M2 = 10;
d = 1;
fac = 35;
eps = .0001;

x1 = -M2*d/(M1+M2);
x2 = x1 + d;

T2 = 4*pi^2*d^3/(M1+M2);
omega = 2*pi/sqrt(T2);

N = 100;
for yloop = 1:N
    for xloop = 1:N

        y = (yloop - N/2)/fac;
        x = (xloop - N/2)/fac;
        r = sqrt(y^2 + x^2);

        d1 = sqrt(y^2 + (x-x1)^2);
        d2 = sqrt(y^2 + (x-x2)^2);

        temp = M1/(d1+eps) + M2/(d2+eps) + 0.5*omega^2*r^2;

        V(yloop,xloop) = -temp;
```

```
    end
end

for loop = 1:40
    L(loop) = -389.8 + 750*loop/(50+loop);
end

figure(1)
contour(V,L,'k')
set(gcf,'Color','White')
axis square

print -dtiff -r600 L45
```

Chapter 3

AHBif.m
dblwellhysteresis.m
DoubleWellrk.m
flowsimple.m
logistic.m
Pendulum3.m
lorenz.m
rossler.m
Henon.m
Lozi2.m
Heiles.m
rksimple.m
S147ext.m
standclinic.m
vdp.m
vandpol.m

AHBif.m

Aronov-Hopf bifurcation simulator.

```
%function AHBif
% a<0 and c>0 stable orbit (supercritical)
% a>0 and c<0 unstable orbit (subcritical)
% Try: a = 1 and c = -0.5 (stable fixed point and unstable limit cycle)
% Try: a = -1 and c = +0.5 (unstable fixed point and stable limit cycle)


function AHBif

a = -1;
c = 0.5;

xrange = [-3 3];
yrange = [-3 3];
rngx = xrange(2) - xrange(1);
rngy = yrange(2) - yrange(1);

[X,Y] = meshgrid(xrange(1):0.125:xrange(2), yrange(1):0.125:yrange(2));

[x,y] = f5(X,Y);

clf
figure(1)
for xloop = 1:20
```

```
    xs = xrange(1) + xloop*rngx/20;
    for yloop = 1:20
        ys = yrange(1) + yloop*rngy/20;

        streamline(X,Y,x,y,xs,ys)

    end
end
hold on
[XQ,YQ] = meshgrid(-3:0.5:3,-3:0.5:3);
[xq,yq] = f5(XQ,YQ);
quiver(XQ,YQ,xq,yq,5,'r')
hold off

axis([-2 2 -2 2])


    function [x,y] = f5(X,Y)

        r2 = X.^2 + Y.^2;
        x = Y + X.*(c+a*r2);
        y = - X +Y.*(c+a*r2);


    end     % end f5


end % end flowsimple
```

dblwellhysteresis.m

Double-well hysteresis.

```
% Companion to IMD: Demonstration of double-well bistability
clear
format compact
eps = 0.0001;

for loop = 1:100
    c = -0.8 + 1.6*loop/100 + eps;
    xc(loop) = c;

    y = roots([-4 0 2 c]);

    xtmp = real(y(1));
    ytmp = real(y(2));
    X(loop) = min(xtmp,ytmp);
    Y(loop) = max(xtmp,ytmp);
    Z(loop) = real(y(3));

    xx = -3:0.1:3;
    yy = c + 2*xx - 4*xx.^3;
    figure(1)
```

```
    plot(xx,yy)
    axis([-3 3 -3 3])
    line([-3 3],[0 0],'Color','r')
    pause(0.01)


end

figure(2)
plot(xc,X,xc,Y,xc,Z)
legend('1','2','3')
axis([-1 1 -1 1])

x = -1.5:0.01:1.5;
y0 = -0.7*x + x.^4 - x.^2;
y1 = -0.4*x + x.^4 - x.^2;
y2 =  x.^4 - x.^2;
y3 = 0.4*x + x.^4 - x.^2;
y4 = 0.7*x + x.^4 - x.^2;

figure(3)
plot(x,y0,x,y1,x,y2,x,y3,x,y4)
axis([-1.5 1.5 -1 0.6])

Printfile6('DWpoten.txt',x,y0,y1,y2,y3,y4)
```

DoubleWellrk.m

Double-well time dependence.

```
% Companion to IMD: Demonstration of double-well bistability
function DoubleWell

inc = 1;

if inc == +1
    c0 = -0.7;
    y0 = [-0.2 0];
else
    c0 = 0.7;
    y0 = [0.898 0];
end


X = y0(1); Y = y0(2); T = 0;

delt = 2.5;
tloopmax = 100;

for tloop = 1:tloopmax

    tlo = (tloop - 1)*delt;
    thi = tloop*delt;
```

```
    if inc == +1
        c = c0 + 3*abs(c0)*tloop/tloopmax;
    else
        c = c0 - 3*abs(c0)*tloop/tloopmax;
    end

    tspan = [tlo thi];

    [t,y] = ode45(@f5,tspan,y0);

    [sz,dum] = size(t);

    X = append(X,y(:,1)');
    Y = append(Y,y(:,2)');
    T = append(T,t');

    y0(1) = y(sz,1);
    y0(2) = y(sz,2) + 0.001*rand + 0.001;

end

Tthresh = delt*tloopmax/2;

if inc == +1
    C = c0 + 3*abs(c0)*T/(delt*tloopmax);
else
    C = c0 - 3*abs(c0)*T/(delt*tloopmax);
end

Cthresh = 0;

figure(1)
plot(C,X,'k','LineWidth',1)

figure(2)
plot(X,Y)

Printfile2('DWH.txt',C,X)

    function yd = f5(t,y)

        yp(1) = y(2);
        yp(2) = -0.1*y(2)-4*y(1).^3 + 2.4*y(1) + c;

        yd = [yp(1);yp(2)];

    end     % end f5

end
```

flowsimple.m

This is one of the workhorses of IMD. It displays the vector flow fields of an ODE flow. Many HW problems will start with flowsimple.m and modify it to the specific problem.

```
function flowsimple

% xrange = [-3 3];
% yrange = [-3 3];
xrange = [0 3];
yrange = [0 3];
rngx = xrange(2) - xrange(1);
rngy = yrange(2) - yrange(1);

[X,Y] = meshgrid(xrange(1):0.1:xrange(2), yrange(1):0.1:yrange(2));

[x,y] = f5(X,Y);

clf
figure(1)
for xloop = 1:10
    xs = xrange(1) +xloop*rngx/10;
    for yloop = 1:10
        ys = yrange(1) +yloop*rngy/10;

        streamline(X,Y,x,y,xs,ys)

    end
end
hold on
[XQ,YQ] = meshgrid(xrange(1):0.25:xrange(2),yrange(1):0.25:yrange(2));
[xq,yq] = f5(XQ,YQ);
quiver(XQ,YQ,xq,yq,5,'r')
hold off

axis([xrange(1) xrange(2) yrange(1) yrange(2)])
set(gcf,'Color','White')


    function [x,y] = f5(X,Y)

        x = X.*(1-2*X+Y);
        y = Y.*(0.5 + 0.5*X - Y);

%         x = Y+0.75*X;
%         y = X.^3-Y-X;


%         x=X.*(2-X-Y);
%         y = X-Y;


        % x = X.*(X-Y);
        % y = Y.*(2*X-Y);
```

```
        %          a = 2; b = 2;
        %          x = Y - a*X;
        %          y = -X - b*Y;

        %      x = -X.^2-Y;
        %      y = X;

        %x = X.^2 - Y;
        %y = X - Y;

        %x = X.*(3-X-2*Y);
        %y = Y.*(2-X-Y);

    end     % end f5

end % end flowsimple
```

Logistic map

logistic.m

```
% logistic.m

clear

xold = 0.231;
maxg = 900;         % 4000 for good coverage (pause off)
ming = 2.9;
figure(1);
clf
axis([2.7 4 0 1]);
hold on
for gainloop = 1:maxg

    g=(4.00-ming)*gainloop/maxg + ming;

    for sloop = 1:100           %settle-down loop
        xnew = g*xold*(1-xold);
        xold = xnew;
    end                     % end settle-down loop

    rep = 64;
    ind = 0;
    for rloop = 1:rep
        xnew = g*xold*(1-xold);
        xold = xnew;

        ind = ind + 1;
        x(ind) = g + (4.00-ming)*(rloop-1)/(maxg*rep);
        y(ind) = xnew;

    end     % end rloop
```

```
    plot(x,y,'o','MarkerSize',2);
     pause(0.01)  % Take pause off to watch real-time, but put maxg to 300


end     % end gainloop
hold off
```

Chaotic pendulum.

Pendulum3.m

```
function Pendulum3

F = 0.7;                     % driving strength (0.7) 0.5  0.56  0.58
w = 0.7;                     % drive frequency (0.7)
c = 0.1;                    % damping (0.05)   0.5 0.45 0.4 0.39 0.35

% (0.7,0.7,0.27) roughly divides regular from chaotic long-term behavior(??)
% The sequence (0.7,0.7,0.5), (0.7,0.7,0.45), (0.7,0.7,0.4), (0.7,0.7,0.39),
(0.7,0.7,0.35) is interesting

tspan = [0 50];
y0 = [1 1 w];
[t,y] = ode45(@f5,tspan,y0);
[sy,dum] = size(y);

tspan = [50 20000];           % 8000 for a good return map, 1000 for demo
purposes
y0 = [y(sy,1) y(sy,2) y(sy,3)]

figure(1)
options = odeset('OutputFcn',@odephas3);
% options = odeset('OutputFcn',@odeplot);

[t,y] = ode45(@f5,tspan,y0,options);

siz = size(t)

theta = mod(y(:,1)-pi,2*pi)-pi;
thetadot = y(:,2);
wt = y(:,3);

figure(2)
plot(t(400:siz),y(400:siz,2),'k')
xlabel('Time')
legend('speed')

figure(3)
plot(theta(400:siz),y(400:siz,2),'ok','MarkerSize',2,'MarkerFaceColor','k')
```

```
% Power Spectrum
Pow = 0;
if Pow == 1
    [st dum] = size(t);
    tshort = t(50:st,1);
    yshort = y(50:st,1);
    [st dum] = size(tshort);
    dt = (max(tshort)-min(tshort))/st;
    df = 1/st/dt;
    fmax = 1/dt;
    freq = df:df:fmax;

    yf = yshort(:,1);
    F = fft(yf);
    Pow = F.*conj(F);

    mid = round(st/4);
    figure(4)
    loglog(freq(1:mid),Pow(1:mid))
    title('Power Spectrum')
    xlabel('Frequency (Hz)')


    [Pmax I] = max(Pow(2:mid));
    f0 = freq(I)
    T0 = 1/f0
end


%First Return Map
Fst = 1;
if Fst == 1
    cnt = 0;
    last = wt(399);
    for loop = 400:siz
        if (last < 0)&(wt(loop) > 0)
            cnt = cnt+1;
            th(cnt) = theta(loop);
            thd(cnt) = thetadot(loop);
            last = wt(loop);
        else
            last = wt(loop);
        end
    end

    figure(5)
    plot(th,thd,'ok','MarkerSize',2,'MarkerFaceColor','k')
    axis([-pi pi -4 2])
end

set(gcf,'color','white')

keyboard
```

```
% Model function
    function dy = f5(t,y)

        dy = zeros(3,1);
        dy(1) = y(2);
        %dy(2) = F*sin(y(3)) - sin(y(1)) - c*y(2);
        dy(2) = F*y(3) - sin(y(1)) - c*y(2);
        %dy(3) = w;
        dy(3) = -w*sin(w*t);

    end     % end f5


end % end ltest
```

Lorenz chaotic attractor.

lorenz.m

```
function lorenz

a = 10.;         % 10
b = 100;          % 50
c = 8./3;        % 8/3

y0 = [1 1 1];
%y0 = [12.8 10.955 53.455];
y0 = [-4.3426   -9.3985 21.389];
tspan = [1 50];

options = odeset('OutputFcn',@odephas3);
% options = odeset('OutputFcn',@odeplot);

[t,y] = ode45(@f5,tspan,y0,options);

figure(2)
plot(t,y(:,1),t,y(:,2),t,y(:,3))

figure(3)
plot(y(:,1),y(:,2))

d = y(:,1);
e = y(:,2);
f = y(:,3);
Printfile4('lout',t',d',e',f')


    function yd = f5(t,y)

        yp(1) = a*(y(2) - y(1));
        yp(2) = y(1)*(b - y(3)) - y(2);
        yp(3) = y(1)*y(2) - c*y(3);

        yd = [yp(1);yp(2);yp(3)];
```

```
    end     % end f5

end % end ltest
```

Rössler chaotic attractor.

rossler.m

```
% function rossler

function y = rossler

a = 0.3;         % 0.15 (defined phase)     0.3 (chaotic phase)
b = 0.4;          % 0.4
c = 8;        % 8

y0 = [1 1 0];
tspan = [1 30];
[t,y] = ode45(@f5,tspan,y0);
[sz dum] = size(y);

y0 =[y(sz,1) y(sz,2) y(sz,3)];
tspan = [1 200];
options = odeset('OutputFcn',@odephas3);
[t,y] = ode45(@f5,tspan,y0,options);

figure(2)
plot(t,y(:,1),t,y(:,2),t,y(:,3))

figure(3)
plot(y(:,1),y(:,3))
title('x-z')

figure(4)
plot(y(:,2),y(:,3))
title('y-z')

figure(5)
plot(y(:,1),y(:,2))
title('x-y')



d = y(:,1);
e = y(:,2);
f = y(:,3);
Printfile4('lout',t',d',e',f');


    function yd = f5(t,y)
```

```
        yp(1) = -(y(2) + y(3));
        yp(2) = y(1) + a* y(2);
        yp(3) = b + y(3)*(y(1) - c);

        yd = [yp(1);yp(2);yp(3)];

    end     % end f5


end % end ltest
```

Henon map.

Henon.m

```
% Henon.m
% B = 0.3; C from 0.3 to 1.1 goes through bifurcation cascade
% B = 0.3; C = 1.2 and look at tip of attractor (self-similar)

clear
tic

B = 0.4252;                    %0.4252    1.0 (Area-preserving)
C = 1.2;                        %1.2            0.13

x = 1:1000;

x0 = 2*(rand-0.5);           % 0.7878
y0 = (rand-0.5);             % -0.3009

xlast = x0;
ylast = y0;

for transloop = 1:100
    xnew = 1 + ylast - C*xlast^2;
    ynew = B*xlast;
    xlast = xnew;
    ylast = ynew;
end

figure(1)
clf
hold on
for loopout = 1:40

    for loopin = 1:30000
        xnew(loopin) = 1 + ylast - C*xlast^2;
        ynew(loopin) = B*xlast;
        xlast = xnew(loopin);
        ylast = ynew(loopin);
    end

    plot(xnew,ynew,'o','MarkerSize',2)
    %axis([-2 2 -2 2])
```

```
    pause(0.1)
end

hold off
toc
```

Lozi map.

Lozi2.m

```
% Lozi.m

clear

B = -1.00;
C = 0.5;

h = colormap(hsv);

for itloop = 1:200
    itloop
rn = ceil(63*rand);

xlast = randn;
ylast = randn;

for transloop = 1:100
    xnew = 1 + ylast - C*abs(xlast);
    ynew = B*xlast;
    xlast = xnew;
    ylast = ynew;
end

figure(1)
axis([-2.5 2.5 -2.5 2.5])
%clf
hold on
for loop = 1:500
    xnew = 1 + ylast - C*abs(xlast);
    ynew = B*xlast;
    xlast = xnew;
    ylast = ynew;

plot(real(xnew),real(ynew),'o','MarkerSize',6,'LineWidth',2,'Color',h(rn,:))
end
%plot(real(xnew),real(ynew),'o','MarkerSize',2,'Color','r')


%hold off

end
```

Heiles.m

Henon-Heiles Hamiltonian chaos.

```
function Heiles

c(1) = 'r';
c(2) = 'b';
c(3) = 'g';
c(4) = 'k';
c(5) = 'c';
c(6) = 'm';
c(7) = 'y';

h = colormap(lines);


E = 0.1;        % 0.1

epsE = 1;         % 1.0

prms = sqrt(E);
pmax = sqrt(2*E);

% Potential Function
for xloop = 1:100
    x = -2 + 4*xloop/100;
    for yloop = 1:100
        y = -2 + 4*yloop/100;

        V(yloop,xloop) = 0.5*x^2 + 0.5*y^2 + epsE*(x^2*y - 0.33333*y^3);

    end
end
figure(6)
colormap(jet)
imagesc(V)
hold on
contour(V,30,'LineColor','k')
hold off
caxis([0 1])
colorbar


figure(5)
clf
hold on
repnum = 32;                % 32
mulnum = 64/repnum;
for reploop = 1:repnum-6

    clear yvar py

    px = 2*(rand-0.499)*pmax;
    py = sign(rand-0.5)*sqrt(2*E - px^2);
    y0 = [0 0 px py];
```

```
    tspan = [1 200];            % 500

    figure(1)
    options = odeset('OutputFcn',@odephas3);
    % options = odeset('OutputFcn',@odeplot);

    [t,y] = ode45(@f1,tspan,y0,options);

    siz = size(t)
    y1 = y(:,1);
    y2 = y(:,2);
    y3 = y(:,3);
    y4 = y(:,4);

    figure(2)
    plot(t(1:siz),y(1:siz,1))
    xlabel('Time')
    legend('speed')

    figure(3)
    plot(y(1:siz,1),y(1:siz,2))

% Power Spectrum
    Pow = 0;
    if Pow == 1
        yf = y(:,1);
        F = fft(yf);
        Pow = F.*conj(F);

        figure(4)
        plot(Pow)
    end


    %First Return Map
    Fst = 1;
    if Fst == 1
        cnt = 0;
        last = y1(1);
        for loop = 2:siz
            if (last < 0)&(y1(loop) > 0)
                cnt = cnt+1;
                py(cnt) = y4(loop);
                yvar(cnt) = y2(loop);
                last = y1(loop);
            else
                last = y1(loop);
            end
        end

        figure(5)

plot(yvar,py,'o','MarkerSize',4,'Color',h(mulnum*reploop,:),'MarkerFaceColor'
,h(mulnum*reploop,:))
```

```
    end

end     % end reploop

figure(5)
hold off


% Model function
    function dy = f1(t,y)

        B = 0.000;          %0.002
        dy = zeros(4,1);
        dy(1) = y(3);
        dy(2) = y(4);
        dy(3) = -y(1) - epsE*(2*y(1)*y(2) + B*y(3));
        dy(4) = -y(2) - epsE*(y(1)^2 - y(2)^2 + B*y(4));

    end     % end f5


end % end ltest
```

rksimple.m

This is another of the workhorses of IMD. It solves for the time dependence of an ODE flow. Many HW problems will start with flowsimple.m and modify it to the specific problem.

```
function rksimple

clear


y0 = [0.01 0.01];
tspan = [0 400];

[t,y] = ode45(@f5,tspan,y0);

figure(1)
plot(t,y(:,1),t,y(:,2))

figure(2)
plot(y(:,1),y(:,2))

    function yd = f5(t,y)

        yp(1) = y(2);
        yp(2) = -y(1) + 0.05*y(2)*(1-y(1).^2);

        %mu = 1; beta = 1; w02 = 1;
        %yp(1) = y(2);
```

```
        %yp(2) = mu*y(2)*(1-beta*y(1)^2)-w02*y(1);


        yd = [yp(1);yp(2)];

    end     % end f5

end
```

S147.m

Simple demo of nullclines and separatrixes

```
% Companion to IMD: Program for Nullclines and Separatrixes
function S147

 [X,Y] = meshgrid(-10:0.1:10, -10:0.1:10);

[x,y] = f5(X,Y);

clf
figure(1)

 [XQ,YQ] = meshgrid(-4:0.25:2,-2:0.25:6);
[xq,yq] = f5(XQ,YQ);
quiver(XQ,YQ,xq,yq,5,'r')
hold on
for xloop = -4:0.125:2
    xs = xloop;
    for yloop = 1:2
        ys = -2 + 8*(yloop-1);
        streamline(X,Y,x,y,xs,ys)
    end
end

hold off
h = gca;
set(h,'FontSize',14)

axis([-4 2 -2 6])
set(gcf,'Color','White')

xold = -4;
yold = exp(-xold);
for xloop = -4:0.25:2
    xnew = xloop;
    ynew = exp(-xloop);
    line([xold xnew],[yold ynew])
    xold = xnew;
    yold = ynew;
end
```

```
yold = -2;
xold = -exp(-yold);
for yloop = -2:0.25:6
    xnew = -exp(-yloop);
    ynew = yloop;
    line([xold xnew],[yold ynew])
    xold = xnew;
    yold = ynew;
end

    function [x,y] = f5(X,Y)
        x = X + exp(-Y);
        y = -Y + exp(-X);
    end     % end f5

end % end S147
```

standclinic.m

Homoclinic orbits in the Standard Map

```
% Companion to IMD: standclinic.m
% plots homoclinic orbit onto standard map

clear
close all

eps = 1.0;      % 0.3   1.0

stable = 0;

h = newcolormap('fluoro');

figure(1)
axis([-pi pi -0.5 0.5])
hold on
for icloop = 1:100      % 200 initial conditions
    rn = ceil(63*rand);

    rold = 2.0*pi*(0.5-rand);
    thetold = 2.0*pi*rand;

    for nloop = 1:100       % 200  iterates for each orbit
        r = rold + eps*sin(thetold);
        theta = mod(thetold + r,2*pi);


        thetaplot = mod(theta-pi,2*pi) - pi;
        rplot = amod(r,pi)/(2*pi);


plot(thetaplot,rplot,'o','MarkerSize',2,'LineWidth',0.5,'Color',h(rn,:))

        rold = r;
```

```
        thetold = theta;

    end

    pause(0.1)

end %icloop

K = eps;

J = [1 1+K;1 1];

[V,D] = eig(J);

Vu = V(:,1);
Vs = V(:,2);
My = D(1,1);

eps0 = 5e-7;

for eloop = 1:100
    rn = ceil(63*eloop/100);

    eps = eps0*eloop;

    roldu1 = eps*Vu(1);
    thetoldu1 = eps*Vu(2);

    Nloop = ceil(-6*log(eps0)/log(eloop+1));
    flag = 1; cnt = 0;
    while (flag == 1)&(cnt < Nloop)
        cnt = cnt + 1;

        ru1 = roldu1 + K*sin(thetoldu1);
        thetau1 = thetoldu1 + ru1;

        roldu1 = ru1;
        thetoldu1 = thetau1;

        if thetau1 > 4*pi
            flag = 0;
        end

        Hr(eloop,cnt) = roldu1;
        Ht(eloop,cnt) = thetoldu1;

    end

end

clear x y
x = Ht(:,13)-2*pi;
x2 = -x;
y = Hr(:,13)/(2*pi);
```

```
y2 = -y;

plot(x,y,'k')

clear x y
x = Ht(5:100,16)-2*pi;
x2 = -x;
y = Hr(5:100,16)/(2*pi);
y2 = -y;
plot(x,y,'k')

clear x y
x = Ht(8:100,17)-2*pi;
x2 = -x;
y = Hr(8:100,17)/(2*pi);
y2 = -y;
plot(x,y,'k')

clear x y
x = Ht(16:100,18)-2*pi;
x2 = -x;
y = Hr(16:100,18)/(2*pi);
y2 = -y;
plot(x,y,'k')

if stable == 1

    % Now Stable manifold
    for eloop = 1:100
        rn = ceil(63*eloop/100);

        eps = eps0*eloop;

        roldu1 = eps*Vs(1);
        thetoldu1 = eps*Vs(2);

        Nloop = ceil(-6*log(eps0)/log(eloop+1));
        flag = 1; cnt = 0;
        while (flag == 1)&(cnt < Nloop)
            cnt = cnt + 1;

            ru1 = roldu1 - K*sin(thetoldu1);
            thetau1 = thetoldu1 - ru1;

            roldu1 = ru1;
            thetoldu1 = thetau1;

            if thetau1 > 4*pi
                flag = 0;
            end

            Hr(eloop,cnt) = roldu1;
            Ht(eloop,cnt) = thetoldu1;

        end
```

```
    end

    clear x y
    x = Ht(:,13)-2*pi;
    x2 = -x;
    y = Hr(:,13)/(2*pi);
    y2 = -y;

    plot(x,y,'k')

    clear x y
    x = Ht(5:100,16)-2*pi;
    x2 = -x;
    y = Hr(5:100,16)/(2*pi);
    y2 = -y;
    plot(x,y,'k')

    clear x y
    x = Ht(8:100,17)-2*pi;
    x2 = -x;
    y = Hr(8:100,17)/(2*pi);
    y2 = -y;
    plot(x,y,'k')

    clear x y
    x = Ht(16:100,18)-2*pi;
    x2 = -x;
    y = Hr(16:100,18)/(2*pi);
    y2 = -y;
    plot(x,y,'k')

    plot(0,0,'o','MarkerSize',4,'LineWidth',0.1,'MarkerFaceColor','k')

end

hold off
```

vdp.m

Van der Pol oscillator.

```
function vdp

beta =1;     % 1.0 nonlinear term
F = 0.0;        % 0.0 DC drive amplitude
mu = 10.0;       % 1.0 gain
w02 = (2*pi*1.0)^2;      % 1.0 natural frequency

PowS = 1;

y0 = [0 10];
tspan = [0 100];
```

```
figure(1)
options = odeset('OutputFcn',@odephas2);
 %options = odeset('OutputFcn',@odeplot);

[t,y] = ode45(@f5,tspan,y0,options);

figure(2)
plot(t,y(:,1),t,y(:,2))
legend('y1','y2')
xlabel('Time')

figure(3)
plot(y(:,1),y(:,2),'LineWidth',1.5)
title('Configuration Space')
xlabel('x1','FontSize',14)
ylabel('x2','FontSize',14)



% Power Spectrum
if PowS == 1        % Turn this off by setting PowS = 0
    [st dum] = size(t);
    tshort = t(400:st,1);
    yshort = y(400:st,1);
    [st dum] = size(tshort);
    dt = (max(tshort)-min(tshort))/st;
    df = 1/st/dt;
    fmax = 1/dt;
    freq = df:df:fmax;

    yf = yshort(:,1);
    F = fft(yf);
    Pow = F.*conj(F);

    mid = round(st/4);
    figure(4)
    semilogy(freq(1:mid),Pow(1:mid))
    title('Power Spectrum')
    xlabel('Frequency (Hz)')


    [Pmax I] = max(Pow);
    f0 = freq(I)
    T0 = 1/f0
    T0nat = 2*pi/sqrt(w02)
end



    function yd = f5(t,y)

        yp(1) = y(2);
        yp(2) = F + mu*y(2)*(1-beta*y(1)^2)-w02*y(1);
        yd = [yp(1);yp(2)];
```

```
    end     % end f5


end % end vandpol
```

vandpol.m

Van der pol oscillator.

```
function vandpol

% 0.1, 0, 1, 1  [0, 1]
% 0.1, 0, 1, 1  [0, 100]

beta =2;     % 1.0 nonlinear term
F = 0;        % 0.0 DC drive amplitude
mu = 0.1;       % 1.0 gain
w02 = (2*pi*1.0)^2;      % 1.0 natural frequency

PowS = 1;

y0 = [0 8.5];
tspan = [0 20];     %100

figure(1)
options = odeset('OutputFcn',@odephas2);
 %options = odeset('OutputFcn',@odeplot);

[t,y] = ode45(@f5,tspan,y0,options);

figure(2)
plot(t,y(:,1),t,y(:,2))
legend('y1','y2')
xlabel('Time')


% First Return

[sz dum] = size(t);
ind = 0;
xplast = 1;
for loop = 2:sz
    xpnow = y(loop,1);
    xprev = y(loop-1,1);
    if (xpnow < 0)&(xprev > 0)
        %keyboard
        ind = ind + 1;
        Poinc(ind) = y(loop,2);
        xplast = y(loop-1,1)
    end
end

[dum szP] = size(Poinc);
```

```
for loop = 2:szP-1
    ratio(loop) = (Poinc(szP) - Poinc(loop))/(Poinc(szP)-Poinc(loop-1))
end

figure(10)
plot(ratio,'o','MarkerFaceColor','r')
set(gcf,'Color','White')


keyboard



figure(3)
plot(y(:,1),y(:,2),'LineWidth',1.5)
title('Configuration Space')
xlabel('x1','FontSize',14)
ylabel('x2','FontSize',14)


% Power Spectrum
if PowS == 1        % Turn this off by setting PowS = 0
    [st dum] = size(t);
    tshort = t(400:st,1);
    yshort = y(400:st,1);
    [st dum] = size(tshort);
    dt = (max(tshort)-min(tshort))/st;
    df = 1/st/dt;
    fmax = 1/dt;
    freq = df:df:fmax;

    yf = yshort(:,1);
    F = fft(yf);
    Pow = F.*conj(F);

    mid = round(st/4);
    figure(4)
    semilogy(freq(1:mid),Pow(1:mid))
    title('Power Spectrum')
    xlabel('Frequency (Hz)')


    [Pmax I] = max(Pow);
    f0 = freq(I)
    T0 = 1./f0
    T0nat = 2*pi/sqrt(w02)


end

    function yd = f5(t,y)

        yp(1) = y(2);
        yp(2) = F + mu*y(2)*(1-beta*y(1)^2)-w02*y(1);
        yd = [yp(1);yp(2)];
```

```
    end     % end f5

end % end vandpol
```

Chapter 4

couplext.m
sinecircle1.m
coupledvdp.m
coupledrossler.m
couplextlor.m
couplelorenz.m

couplext.m

External drive on a single phase oscillator.

```
%coupleNdriver.m
%External drive on a single phase oscillator

function couplext

for omegloop = 1:60
    fac = -0.15 + 0.1*omegloop/20;
    omega = 2*pi*0.0;
    wdrive = 2*pi*fac;
    g = 2*pi*0.025;

    y0 = 0;
    tspan = [0 20];
    [t,y] = ode45(@f2,tspan,y0);
    [sy,dum] = size(y);

    y0 = y(sy,1);

    tspan = [0 200];

    [t,y] = ode45(@f2,tspan,y0);

    [m,b] = linfit(t,y(:,1));

    om(omegloop) = fac;
    omegout(omegloop) = m/2/pi;

    figure(1)
    plot(t,y(:,1))
    legend('drive','oscillator')
    xlabel('Time')

    pause(0.2)
end

figure(2)
plot(om,omegout)
axis([-0.15 0.15 -0.025 0.025])
```

```
figure(3)
plot(om,(om - omegout))
function yd = f2(t,y)
        yp(1) = omega + g*sin(y(1) - wdrive*t);
        yd = [yp(1)];
    end     % end f2

    function [m,b] = linfit(x,y)
        meanx = mean(x);
        meany = mean(y);
        meanxy = mean(x.*y);
        meanx2 = mean(x.^2);

        m = (meanxy - meanx*meany)/(meanx2 - meanx^2);
        b = meany - m*meanx;

        f = m*x + b;
    end

end % end vandpol
```

coupledvdp.m

Coupled van der Pol oscillators.

```
function coupledvdp

beta1 = 1.0;      % 1.0 nonlinear term
F1 = 0.0;         % 0.0 DC drive amplitude
mu1 = 2.5;        % 2.5 gain
w021 = (2*pi*1)^2;       % 1.0 natural frequency

dnu = 2*pi*0.15;       % Frequency offset (0.1/1.618)
eps =2*pi*0.;       % Position coupling (8.4)
epsv = 2*pi*0.15;     % Velocity coupling

beta2 = 1.0;
F2 = 0.0;
mu2 = 2.5;
w022 = (sqrt(w021) + dnu).^2;



PowS = 1;

y0 = [1 -8 -1 8];
tspan = [0 20];
[t,y] = ode45(@f5,tspan,y0);
[sy,dum] = size(y);

y0 = [y(sy,1) y(sy,2) y(sy,3) y(sy,4)]
```

```
tspan = [0 100];

figure(1)
options = odeset('OutputFcn',@odephas2);
% options = odeset('OutputFcn',@odeplot);

[t,y] = ode45(@f5,tspan,y0,options);

figure(2)
plot(t,y(:,1),t,y(:,3))
legend('y1','y3')
xlabel('Time')

% %extract phase
% ph1 = asin(y(:,1)/max(y(:,1)));
% ph2 = asin(y(:,2)/max(y(:,2)));
% phdiff = ph2-ph1;
% figure(6)
% plot(phdiff)


figure(3)
plot(y(:,1),y(:,3),'LineWidth',1.5)
title('Configuration Space')
xlabel('x1','FontSize',14)
ylabel('x3','FontSize',14)

% Printfile2('vdp.out',y(501:800,1)',y(501:800,2)')
% Printfile3('vdpt.out',t(501:1000)',y(501:1000,1)',y(501:1000,2)')

% d = y(:,1);
% e = y(:,2);
% Printfile3('lout',t',d',e')

% Power Spectrum
if PowS == 1
    [st dum] = size(t);
    tshort = t(400:st,1);
    yshort1 = y(400:st,1);
    yshort3 = y(400:st,3);
    [st dum] = size(tshort);
    dt = (max(tshort)-min(tshort))/st;
    df = 1/st/dt;
    fmax = 1/dt;
    freq = df:df:fmax;

    yf1 = yshort1;
    F1 = fft(yf1);
    Pow1 = F1.*conj(F1);
%     lpPow1 = lpfilter(Pow1,0.5);

    yf3 = yshort3;
    F3 = fft(yf3);
    Pow3 = F3.*conj(F3);
%     lpPow3 = lpfilter(Pow3,0.5);
```

```
    mid = round(st/4);
    figure(4)
    semilogy(freq(1:mid),Pow1(1:mid),freq(1:mid),Pow3(1:mid))
    title('Power Spectrum')
    xlabel('Frequency (Hz)')


    [Pmax1 I] = max(Pow1);
    f01 = freq(I)
    [Pmax3 I] = max(Pow3);
    f03 = freq(I)

    T01 = 1/f01;
    T03 = 1/f03;
    T0nat = 2*pi/sqrt(w021);
end

Printfile3('phaselock',t',y(:,1)',y(:,3)')

    function yd = f5(t,y)

        yp(1) = y(2) - eps*(y(1)-y(3));
        yp(2) = F1 + 2*mu1*y(2)*(1-beta1*y(1)^2)-w021*y(1)  - epsv*(y(2)-
y(4));
        yp(3) = y(4) + eps*(y(1)-y(3));
        yp(4) = F2 + 2*mu2*y(4)*(1-beta2*y(3)^2)-w022*y(3)  + epsv*(y(2)-
y(4));
        yd = [yp(1);yp(2);yp(3);yp(4)];

    end     % end f5


end % end vandpol
```

coupledrossler.m

Coupled Rössler systems.

```
% function y = coupledrossler(node)
% identical linearly-coupled Rossler osccilators

function [y,t] = coupledrossler(node)

 [N,e,avgdegree,maxdegree,mindegree,numclus,meanclus,Lmax,L2,LmaxL2] =
clusterstats(node);
displine('numclus =',numclus)

a = 0.15;         % 0.15 is phase coherent   0.25 is phase incoherent
b = 0.4;          % 0.4
c = 8;           % 8
g = 0.2;         %(a=0.15 b=0.4 c=0.8 gc=0.2 for SW(50,4,0.1))

y0 =10*rand(3*N,1);
```

```
tspan = [1 20];
options = odeset('OutputFcn',@odephas3);
[t,y] = ode45(@f5,tspan,y0,options);
[sy,sx] = size(y);
y0 = y(sy,:);


tspan = [1 200];
options = odeset('OutputFcn',@odephas3);
[t,y] = ode45(@f5,tspan,y0,options);

% figure(2)
% plot(t,y(:,3),t,y(:,3*round(N/3)),t,y(:,3*round(2*N/3),t,y(:,3*N)))

figure(2)
plot(t,y(:,1),t,y(:,3*N/2-2))

figure(3)
%plot(y(:,1),y(:,3*N/2-1),'k','LineWidth',1.25)
plot(y(:,3*(13-1)+1),y(:,3*(33-1)+2),'k','LineWidth',1.25)  % Note: find 2
nodes that are 1 net diameter distant
set(gcf,'color','white')

figure(4)
plot(y(:,3*(13-1)+1),y(:,3*(13-1)+2),'k','LineWidth',1.25)    % The first
node
set(gcf,'color','white')


    function yd = f5(t,y)

        for nloop = 1:N
            ind1 = 3*nloop-2;
            ind2 = 3*nloop-1;
            ind3 = 3*nloop;

            yp(ind1) = -(y(ind2) + y(ind3));
            yp(ind2) = y(ind1) + a* y(ind2);
            yp(ind3) = b + y(ind3)*(y(ind1) - c);

            linksz = node(nloop).numlink;
            temp = 0;
            tmp1 = yp(ind1);
            for linkloop = 1:linksz
                cindex = node(nloop).link(linkloop);
                temp = temp + g*(y(3*cindex-2) - y(ind1));
            end % linkloop
            yp(ind1) = tmp1 + temp;


        end % nloop

        for nloop = 1:N
            ind1 = 3*nloop-2;
```

```
            ind2 = 3*nloop-1;
            ind3 = 3*nloop;
            yd(ind1,1) = yp(ind1);
            yd(ind2,1) = yp(ind2);
            yd(ind3,1) = yp(ind3);
        end

    end     % end f5


end % end
```

listfrac.m

Rational fractional resonances.

```
% Companion to IMD
% listfrac.m  Lists rational fractions
% Diophantine analogy
% Calls:
% convergent.m

clear
format compact

N = 23;   % max denominator

f = ones(20,1000);

for aloop = 1:20
    alpha = 0.05*(aloop - 1);

    ind = 0;
    for qloop = 2:N
        for ploop = 1:qloop-1

            [p q] = convergent(ploop/qloop,8);

            if (p == ploop)&(q == qloop)
                ind = ind + 1;
                frac(ind).p = p;
                frac(ind).q = q;
            end

        end
    end

    sz = ind;

    delx = 1/1000;
    for xloop = 1:1000;
        x = xloop*delx;
```

```
        for fracloop = 1:sz
            rat = frac(fracloop).p/frac(fracloop).q;
            del = alpha/(2*frac(fracloop).q^2.5);

            if abs(x-rat)<del
                f(aloop,xloop) = 0;
            end

        end
    end

end     % end aloop

figure(1)
pcolor(f)
shading interp
colormap(gray)
```

convergent.m

Subroutine for continued fractions.

```
%function [p q] = convergent(r,N)
% Nth convergent of real number r
% p and q are relatively prime
%
% see also continued.m


function [p q] = convergent(r,N)

a = continued(r,N);

num = 1;
denom = a(N);

flag = 1;
while flag == 1
    if denom == 0;
        N = N-1;
        denom = a(N);
    else
        flag = 0;
    end
end

for loop = 1:N-1

    if(loop<(N-1))
        num = a(N-loop)*denom + num;
        tmp = denom;
        denom = num;
        num = tmp;
    else
        num = a(N-loop)*denom + num;
```

```
    end
end

p = num;
q = denom;
```

continued.m

Continued fractions.

```
% a = continued(r,N)
% continued fraction
% see also convergent.m

function a = continued(r,N)

rtmp = r;

for loop = 1:N

    tmp = fix(rtmp);
    if abs(rtmp - tmp) > 1e-6
        rtmp = 1./(rtmp - tmp);
    else
        rtmp = 0;
    end

    a(loop) = tmp;

end
```

couplextlor.m

External drive for Lorenz chaotic system.

```
% Companion to IMD
% Externally driven Lorenz
%
function couplextlor

aa = 10.;         % 10
bb = 50;          % 50
cc = 8./3;        % 8/3


for omegloop = 200:400

    fac = 1.0 + 2*omegloop/240;

    omega = 2*pi*1.0;
    wdrive = 2*pi*fac;
    g = 2*pi*1;
```

```
    y0 = [1,1,1];
    tspan = [0 50];
    [t,y] = ode45(@fode,tspan,y0);
    [sy,dum] = size(y);

    y0 = y(sy,:);

    tspan = [0 200];

    [t,y] = ode45(@fode,tspan,y0);

    om(omegloop) = fac;

    z = y(:,3);
    u = sqrt(y(:,1).^2 + y(:,2).^2);
    xp = u - 20;
    yp = z - 50;

    theta = atan2(yp,xp);
    signal = sin(theta);

    figure(2)
    subplot(1,2,1),plot(u,y(:,3))
    text(5,95,num2str(fac),'FontSize',18)
    axis([0 60 0 100])

    PowS = 1;
% Power Spectrum
if PowS == 1        % Turn this off by setting PowS = 0
    [st dum] = size(t);
    tshort = t(400:st,1);
    %yshort = u(400:st) - mean(u(400:st));
    yshort = signal(400:st) - mean(signal(400:st));
    [st dum] = size(tshort);
    dt = (max(tshort)-min(tshort))/st;
    df = 1/st/dt;
    fmax = 1/dt;
    freq = df:df:fmax;

    yf = yshort(:,1);
    F = fft(yf);
    Pow = F.*conj(F);

    mid = round(st/4);

    subplot(1,2,2),loglog(freq(2:mid),Pow(2:mid))
    title('Power Spectrum','FontSize',18)
    xlabel('Drive Frequency (Hz)','FontSize',16)

    minP(omegloop) = min(log(abs(Pow(2:mid)) + 1e-6))/10;

    Fmean = sum(freq(1:mid).*Pow(1:mid)')/sum(Pow(1:mid));
    F2 = sum(freq(1:mid).^2.*Pow(1:mid)')/sum(Pow(1:mid));

    [Pmax I] = max(Pow);
```

```
    [Psortmp,Isort] = sort(Pow(2:mid),'descend');
    Psort(omegloop,1:20) = freq(Isort(1:20));

    f0(omegloop) = freq(I);
    f1(omegloop) = Fmean;
    f2(omegloop) = sqrt(F2 - Fmean.^2);
end

    pause(0.01)
    clear t y

end

figure(4)
plot(om,Psort,'o')
hold on
plot(om,0.92*f1,'LineWidth',2,'Color','k')
hold off
set(gcf,'Color','White')
h = gca;
set(h,'FontSize',14)
xlabel('Drive Frequency (Hz)','FontSize',16)
ylabel('Response Frequency (Hz)','FontSize',16)
title('External Drive on Lorenz','FontSize',18)
print -dtiff -r600 lorensynch

figure(5)
plot(om,(om - 0.92*f1 + 1.95))
axis([1 3 1 3])
line([1 3],[1 3],'LineStyle','--','Color','k')
Printfile2('lorensynch.txt',om,f1)


    function yd = fode(t,y)


        yp(1) = aa*(y(2) - y(1));
        yp(2) = y(1)*(bb - y(3)) - y(2);
        yp(3) = y(1)*y(2) - cc*y(3) + g*wdrive*sin(wdrive*t);

        yd = [yp(1);yp(2);yp(3)];

    end     % end f2

    function [m,b] = linfit(x,y)

        meanx = mean(x);
        meany = mean(y);
        meanxy = mean(x.*y);
        meanx2 = mean(x.^2);

        m = (meanxy - meanx*meany)/(meanx2 - meanx^2);
        b = meany - m*meanx;
```

```
        f = m*x + b;
    end

end % end vandpol
```

couplelorenz.m

Coupled Lorenz oscillators.

```
% Companion to IMD
% couplelorenz
% 2 coupled Lorenz systems

function couplelorenz

a = 10.;        % 10
b = 50;         % 50
c = 8./3;       % 8/3
g = 1;            % 0.85  0.88

y0 = [20 1 1 -20 1 1];
%y0 = [12.8 10.955 53.455];
%y0 = [-4.3426   -9.3985 21.389];
tspan = [0 30];

options = odeset('OutputFcn',@odephas3);

[ttmp,ytmp] = ode45(@f5,tspan,y0,options);
[sy,dum] = size(ytmp);
y0 = ytmp(sy,:);

tspan = [0 100];
[t,y] = ode45(@f5,tspan,y0,options);

figure(2)
colormap(jet)
plot(t,y(:,1),'k',t,y(:,2),'r',t,y(:,6),'b')
legend('x','y','w')

figure(3)
plot(y(:,1),y(:,5))
xlabel('x-value')
ylabel('v-value')

u = sqrt(y(:,1).^2 + y(:,2).^2);
w = y(:,6);

figure(4)
plot(u,w)
xlabel('(x,y)-value')
ylabel('w-value')


PowS = 1;
```

```
% Power Spectrum
if PowS == 1        % Turn this off by setting PowS = 0
    [st dum] = size(t);
    tshort = t(400:st,1);
    %yshort = y(400:st,1);
    yshort = u(400:st,1);
    [st dum] = size(tshort);
    dt = (max(tshort)-min(tshort))/st;
    df = 1/st/dt;
    fmax = 1/dt;
    freq = df:df:fmax;

    yf = yshort(:,1);
    F = fft(yf);
    Pow = F.*conj(F);

    mid = round(st/4);
    figure(6)
    semilogx(freq(2:mid),Pow(2:mid))
    title('Power Spectrum')
    xlabel('Frequency (Hz)')


    [Pmax I] = max(Pow);
    f0 = freq(I)
    T0 = 1/f0
end

d = y(:,1);
e = y(:,2);
f = y(:,3);
Printfile4('lout',t',d',e',f')

    function yd = f5(t,y)

        yp(1) = a*(y(2) - y(1));
        yp(2) = y(1)*(b - y(3)) - y(2);
        yp(3) = y(1)*y(2) - c*y(3) - g*(y(3) - y(6));

        yp(4) = a*(y(5) - y(4));
        yp(5) = y(4)*(b - y(6)) - y(5);
        yp(6) = y(4)*y(5) - c*y(6)- g*(y(6) - y(3));

        yd = [yp(1);yp(2);yp(3);yp(4);yp(5);yp(6)];

    end     % end f5

end % end
```

sinecircle1.m

Sine-circle map

```
clear

omega = 1.618033988749895;   %exp(1)/2     pi/2     golden

testcase = 2;           % 1 = iterate at a single g     2= scan g

if testcase == 1
    gval = 80.;         % 43 76
    glooplo = gval;
    gloophi = gval;
elseif testcase == 2   % scan g
    figure(3)
    close
    glooplo = 0;
    gloophi = 2;
end


delg = 0.005;
for gloop = glooplo:delg:gloophi

    g =gloop;

    if testcase == 1
        disp(strcat('g = ',num2str(g)))
    end

    thet0 = rand;
    thet1last = thet0;
    N = 100;
    for loop = 1:N

        thet1emp = mod(thet1last + omega + g*sin(2*pi*thet1last),1);
        thet1last = mod(thet1emp,1);

    end

    for loop = 1:N

        thet1(loop) = mod(thet1last + omega + g*sin(2*pi*thet1last),1);
        thet1last = mod(thet1(loop),1);

    end

    if testcase == 1
        figure(1)
        plot(thet1,thet2,'o')
        axis([0 1 0 1])

        figure(2)
        plot(thet1,(thet2-thet1),'o')
        axis([0 1 0 1])
    elseif testcase == 2
        x = g*ones(1,N);
        y = thet1;
```

```
        hold on
        figure(3)
        plot(x,y,'.k')
        axis([0 gloophi 0 1])
        hold off
        set(gcf,'Color','White')
    end

end
```

Chapter 5 Dynamic Networks

makeER.m
Erdos.m
makeLB.m
makeSF.m
makeSW.m
addlink.m
sublink.m
addnode.m
subnode.m
removenode.m
coupleN.driver
node2distance.m
adjacency.m
clusternum.m
clusterstats.m
clustercoef.m
avgeigs.m
diffusionmat.m
SIRS.m

makeER.m

Make Erdös-Renyi graph

```
% function node = makeER(N,p)
% Creates an Erdos-Renyi graph of N nodes and link probability p
% node structure is ...
% node(1).element = node number;
% node(1).numlink = number_of_links;
% node(1).link = [set of linked node numbers];


function node = makeER(N,p)

% Set node structure
node(1).element = 1;
node(1).numlink = 0;
node(1).link = [];

% Create ER graph
[A,degree,Lap] = Erdos(N,p);

% set links
for rowloop = 2:N
```

```
    node = addnode(rowloop,node);

    for coloop = 1:rowloop-1
        if A(rowloop, coloop) ==1
            node = addlink(rowloop,coloop,node);
        end



    end     % end coloop
end     % end rowloop
```

Erdos.m

Function called by makeER

```
% function [A,degree,Lap] = Erdos(N,p)
% Generates an Erdos-Renyi random graph of N nodes with edge probability p
% A is the adjacency matrix
% degree is the degree of the node
% Lap is the Lapacian matrix

function [A,degree,Lap] = Erdos(N,p)

e = round(p*N*(N-1)/2);

A = zeros(N,N);         %Adjacency matrix

loop = 0;
while loop ~=e

    x = round(rand*(N-1))+1;
    y = round(rand*(N-1))+1;

    flag = A(y,x);
    if (x~=y)&(flag==0)
        A(x,y) = 1;
        A(y,x) = 1;
        loop = loop +1;
    end


end

degree = sum(A);

Lap = -A;
for loop = 1:N
    Lap(loop,loop) = degree(loop);
end
```

makeLB.m

Make Linear graph

```
% function node = makeLB(N,m,)
% Creates a linear graph (cycle) of N nodes and 2*m neighbor links per new
node
% node structure is ...
% node(1).element = node number;
% node(1).numlink = number_of_links;
% node(1).link = [set of linked node numbers];


function node = makeLB(N,m)

A = zeros(N,N);

node(1).element = 1;
node(1).numlink = 2*m;
node(1).link = [];
for loop = 2:N
    node = addnode(loop,node);
    node(loop).numlink = 2*m;
end

for loop = 1:N
    nlinks = 0;
        for neighloop = 1:m
            nlinks = nlinks + 1;
            if (loop+neighloop) <=N
                A(loop,loop+neighloop) = 1;
                node(loop).link(nlinks) = loop+neighloop;
            else
                A(loop,loop+neighloop-N) = 1;
                node(loop).link(nlinks) = loop + neighloop-N;
            end
            nlinks = nlinks+1;
            if (loop-neighloop) >=1
                A(loop,loop-neighloop) = 1;
                node(loop).link(nlinks) = loop-neighloop;
            else
                A(loop,N + loop - neighloop) = 1;
                node(loop).link(nlinks) = N + loop - neighloop;
            end
        end
end
```

makeSF.m

Make Scale-Free graph

```
% function node = makeSF(N,m)
% Creates a Scale-Free graph of N nodes and m linksper new node
% node structure is ...
% node(1).element = node number;
% node(1).numlink = number_of_links;
```

```
% node(1).link = [set of linked node numbers];


function node = makeSF(N,m)

% Set node initial structure
node = makeER(2*m,1);
numnodes = 2*m;

for addloop = 2*m+1:N   % add nodes up to N

    node = addnode(addloop,node);

    sumdegree = 0;
    for dloop = 1:numnodes
        sumdegree = sumdegree + node(dloop).numlink;
    end     % end dloop
    for dloop = 1:numnodes
        nodeprob(dloop) = node(dloop).numlink/sumdegree;
    end     % end dloop


    for mloop = 1:m     % add m links preferentially

        flag = 1;
        while flag == 1
            temp = nonuniformrand(numnodes,nodeprob);
            Ar = ismember(node(addloop).link,temp);
            if sum(Ar) == 0
                 nodetolink = temp;
                 flag = 0;
            end
        end

        node = addlink(addloop,nodetolink,node);



    end     % end mloop


numnodes = numnodes + 1;

end     % end node addition loop
```

makeSW.m

Make Small-World graph

```
% function node = makeSW(N,m, p)
% Creates a Small-world graph of N nodes and m links per new node
% p is the re-wiring probability
% node structure is ...
% node(1).element = node number;
```

```
% node(1).numlink = number_of_links;
% node(1).link = [set of linked node numbers];
% Calls:
%  makeLB
%  sublink
%  addlink


function node = makeSW(N,m,p)

% Set node initial structure
node = makeLB(N,m);             % linear cycle with 2*m neighbors

for loop = 1:N
    %loop
    nlink = node(loop).numlink;
    linknum = node(loop).link;

    for linkloop = 1:nlink
        oldtarget = linknum(linkloop);

        if oldtarget > loop

            test = rand;
            if test < p

                node = sublink(loop,oldtarget,node);

                flag = 1;
                while flag == 1
                    ind = round(rand*(N-1)) + 1;
                    if ind ~= loop
                        flag = 0;
                        node = addlink(loop,ind,node);
                    end
                end     % end while

%                 drawnet(node)
%                 keyboard

            end     % end test

        end     % end if oldtarget

    end     % end linkloop

end     % end loop
```

addlink.m

```
function node = addlink(node1,node2,node);

ind = node(node1).numlink;
```

```
node(node1).link(ind+1) = node2;
node(node1).numlink = ind + 1;

ind = node(node2).numlink;
node(node2).link(ind+1) = node1;
node(node2).numlink =ind+1;

end
```

sublink.m

```
%function node = sublink(node1,node2,node)

function newnode = sublink(node1,node2,node)

templink1 = node(node1).link;
templink2 = node(node2).link;

tempnode = node;
clear node

% templink1
% node2
% templink2
% node1
w1 = where(templink1,node2);
w2 = where(templink2,node1);

if (w1(1) == -1)|(w2(1) == -1)
    disp('Error in sublink: node missing')
    newnode = tempnode;
    return
end

nlink1 = tempnode(node1).numlink;
if nlink1 == 1                          % only one linke in set
    tempnode(node1).numlink = 0;
    tempnode(node1).link = [];
elseif w1 == nlink1             % target at end of set
    newnlink = nlink1 - 1;
    temp1 = templink1(1:newnlink);
    tempnode(node1).numlink = newnlink;
    tempnode(node1).link = temp1;
    tempnode(node1).numlink = newnlink;
else
    newnlink = nlink1 - 1;
    tempnode(node1).link(w1) = tempnode(node1).link(nlink1);
    temp1 =  tempnode(node1).link(1:newnlink);
    tempnode(node1).link = temp1;
    tempnode(node1).numlink = newnlink;
end

nlink2 = tempnode(node2).numlink;
if nlink2 == 1                          % only one linke in set
    tempnode(node2).numlink = 0;
    tempnode(node2).link = [];
```

```
elseif w2 == nlink2             % target at end of set
    newnlink = nlink2 - 1;
    temp2 = templink2(1:newnlink);
    tempnode(node2).numlink = newnlink;
    tempnode(node2).link = temp2;
    tempnode(node2).numlink = newnlink;
else
    newnlink = nlink2 - 1;
    tempnode(node2).link(w2) = tempnode(node2).link(nlink2);
    temp2 = tempnode(node2).link(1:newnlink);
    tempnode(node2).link = temp2;
    tempnode(node2).numlink = newnlink;
end


newnode = tempnode;
```

addnode.m

```
%function node = addnode(newnodenum,node)

function node = addnode(newnodenum,node)

[dum,sz] = size(node);

node(sz+1).element = newnodenum;
node(sz+1).numlink = 0;
node(sz+1).link = [];

end
```

subnode.m

```
%function newnode = subnode(nodenum,node)

function newnode = subnode(nodenum,node)

[dum sz] = size(node);

tempnode = node;

for nodeloop = 1:sz
    nlink = node(nodeloop).numlink;
    for linkloop = 1:nlink
        linknum = node(nodeloop).link(linkloop);
        if (linknum == nodenum)
            gonode.label(1) = nodenum;
            gonode.label(2) = node(nodeloop).element;
            tempnode = sublink(gonode.label(1),gonode.label(2),tempnode);
        end
    end
end
```

```
newnode = tempnode;
```

removenode.m

```
% function newnode = removenode(nodenum,node)
% Removes node from graph
% See also subnode (which keeps the node but deletes all links)


function newnode = removenode(nodenum,node)

[dum sz] = size(node);

ind = 0;
for nodeloop = 1:sz
    if nodeloop ~= nodenum
        ind = ind + 1;
        graph(ind).element = ind;

        nlink = node(nodeloop).numlink;
        lind = 0;
        for linkloop = 1:nlink
            linknum = node(nodeloop).link(linkloop);
            if linknum < nodenum
                lind = lind +1;
                graph(ind).link(lind) = linknum;
            elseif linknum > nodenum
                lind = lind + 1;
                graph(ind).link(lind) = linknum-1;
            end
        end
        graph(ind).numlink = lind;
    end

end

newnode = graph;
```

sublink.m

```
%function node = sublink(node1,node2,node)

function newnode = sublink(node1,node2,node)

templink1 = node(node1).link;
templink2 = node(node2).link;

tempnode = node;
clear node

% templink1
% node2
```

```
% templink2
% node1
w1 = where(templink1,node2);
w2 = where(templink2,node1);

if (w1(1) == -1)|(w2(1) == -1)
    disp('Error in sublink: node missing')
    newnode = tempnode;
    return
end

nlink1 = tempnode(node1).numlink;
if nlink1 == 1                          % only one linke in set
    tempnode(node1).numlink = 0;
    tempnode(node1).link = [];
elseif w1 == nlink1                 % target at end of set
    newnlink = nlink1 - 1;
    temp1 = templink1(1:newnlink);
    tempnode(node1).numlink = newnlink;
    tempnode(node1).link = temp1;
    tempnode(node1).numlink = newnlink;
else
    newnlink = nlink1 - 1;
    tempnode(node1).link(w1) = tempnode(node1).link(nlink1);
    temp1 =  tempnode(node1).link(1:newnlink);
    tempnode(node1).link = temp1;
    tempnode(node1).numlink = newnlink;
end

nlink2 = tempnode(node2).numlink;
if nlink2 == 1                          % only one linke in set
    tempnode(node2).numlink = 0;
    tempnode(node2).link = [];
elseif w2 == nlink2                 % target at end of set
    newnlink = nlink2 - 1;
    temp2 = templink2(1:newnlink);
    tempnode(node2).numlink = newnlink;
    tempnode(node2).link = temp2;
    tempnode(node2).numlink = newnlink;
else
    newnlink = nlink2 - 1;
    tempnode(node2).link(w2) = tempnode(node2).link(nlink2);
    temp2 = tempnode(node2).link(1:newnlink);
    tempnode(node2).link = temp2;
    tempnode(node2).numlink = newnlink;
end


newnode = tempnode;
```

coupleNdriver.m

This is a workhorse program that couples a large network of phase oscillators.

```
%coupleNdriver.m
% Calls: makeER, makerSF, makeSW, coupleN, coupleN0
%            makeglobal, make2Dlattice, makecycle, maketree, histfix

clear
gcf = figure(7);
close(gcf)

writeavi = 0;
example = 3;         % 1 = global     2 = distributed    3 = square lattice
4 = 1D lattice   5 = tree  6 = ER 7 = Scale-free 8 = small-world
inexcoupling = 1;        % 1 = intercoupling    2 = external coupling
Nfac = 30;

displine('Example = ',example);
displine('Coupling = ',inexcoupling);
displine('Nfac = ',Nfac);

if writeavi
    moviename = strcat('Couple.avi');
    aviobj = avifile(moviename,'fps',4);
end

ch = colormap(lines);
%ch = colormap(gray);
z = 1:32;
h = zeros(64,3);
h(1:32,1) = z/32;
h(33:64,2) = (33-z)/32;
h(32,:) = [0 0 0];
h(33,:) = [0 0 0];

%h = colormap(gray);

colormap(h)

if example == 1     % Global coupling
    N = 256;        %100
    width = 0.2;

    omegatemp = width*(rand(1,N)-1);
    meanomega = mean(omegatemp);
    omega = omegatemp - meanomega;
    sto = std(omega);

    node = makeglobal(N);

    [synch,l2,lmax] = eigenlap(node);
    displine('synch = ',synch)
    [cluscoef,eicoef,clus,ei] = clustercoef(node);
     displine('cluscoef = ',cluscoef)
```

```
    nodecouple = node;

    for loop = 1:N
        nodecouple(loop).element = omega(loop);
        lnk(loop) = nodecouple(loop).numlink;
    end

    avgdegree = mean(lnk)


elseif example == 2;        % distributed example
    N = 11;
    width = 0.31;

    omegat(1) = 0.16;
    omegat(2) = 0.24;
    omegat(3) = 0.28;
    omegat(4) = 0.30;
    omegat(5) = 0.31;
    omegat(6) = -0.16;
    omegat(7) = -0.24;
    omegat(8) = -0.28;
    omegat(9) = -0.30;
    omegat(10) = -0.31;
    omegat(11) = 0;

    for yloop = 1:N
        for xloop = 1:N
            g0(yloop,xloop) = 1;
        end
    end

    node = makeglobal(N);
    nodecouple = node;

    for loop = 1:N
        nodecouple(loop).element = omegat(loop);
        lnk(loop) = nodecouple(loop).numlink;
    end

    avgdegree = mean(lnk);
    displine('avgdegree =',avgdegree)
    sto = std(omegat);


elseif example ==3      % square lattice

    Row = 16;       %10
    Col = 16;           % 10
    N = Row*Col;
    width = 0.2;

    omegatemp = width*(rand(1,N)-1);
    meanomega = mean(omegatemp);
```

```
    omega = omegatemp - meanomega;
    sto = std(omega);

    node = make2Dlattice(Row,Col);

    [synch,l2,lmax] = eigenlap(node);
    displine('synch = ',synch)
    [cluscoef,eicoef,clus,ei] = clustercoef(node);
     displine('cluscoef = ',cluscoef)

    nodecouple = node;

    for loop = 1:N
        nodecouple(loop).element = omega(loop);
        lnk(loop) = nodecouple(loop).numlink;
    end

    avgdegree = mean(lnk);
    displine('avgdegree =',avgdegree)


elseif example == 4      % linear cycle
    N = 100;
    width = 0.2;

    omegatemp = width*(rand(1,N)-1);
    meanomega = mean(omegatemp);
    omega = omegatemp - meanomega;
    sto = std(omega);

    node = makecycle(N);
    nodecouple = node;

    for loop = 1:N
        nodecouple(loop).element = omega(loop);
        lnk(loop) = nodecouple(loop).numlink;
    end

    avgdegree = mean(lnk);
    displine('avgdegree =',avgdegree)

elseif example == 5     % make a tree
    degree = 2;
    depth = 7;
    width = 0.2;

    node = maketree(degree, depth);

    [dum,N] = size(node);

    omegatemp = width*(rand(1,N)-1);
    meanomega = mean(omegatemp);
    omega = omegatemp - meanomega;
    sto = std(omega);
```

```
    nodecouple = node;

    for loop = 1:N
        nodecouple(loop).element = omega(loop);
        lnk(loop) = nodecouple(loop).numlink;
    end

    avgdegree = mean(lnk);
    displine('avgdegree =',avgdegree)


elseif example == 6     % Erdos graph
    N = 100;
    p = 0.06;
     width = 0.2;

    omegatemp = width*(rand(1,N)-1);
    meanomega = mean(omegatemp);
    omega = omegatemp - meanomega;
    sto = std(omega);

    node = makeER(N,p);
    [synch,l2,lmax] = eigenlap(node);
    displine('synch = ',synch)
    [cluscoef,eicoef,clus,ei] = clustercoef(node);
     displine('cluscoef = ',cluscoef)

     nodecouple = node;

    for loop = 1:N
        nodecouple(loop).element = omega(loop);
        lnk(loop) = nodecouple(loop).numlink;
    end

    avgdegree = mean(lnk);
    displine('avgdegree =',avgdegree)



elseif example == 7     % scale-free graph
    N = 100;
    m = 3;
     width = 0.2;

    omegatemp = width*(rand(1,N)-1);
    meanomega = mean(omegatemp);
    omega = omegatemp - meanomega;
    sto = std(omega);

    node = makeSF(N,m);
    [synch,l2,lmax] = eigenlap(node);
     displine('synch = ',synch)
     [cluscoef,eicoef,clus,ei] = clustercoef(node);
     displine('cluscoef = ',cluscoef)
```

```
     nodecouple = node;

    for loop = 1:N
        nodecouple(loop).element = omega(loop);
        lnk(loop) = nodecouple(loop).numlink;
    end

    avgdegree = mean(lnk);
    displine('avgdegree =',avgdegree)


    elseif example == 8     % small-world graph
    N = 100;
    m = 3;
    p = 0.75;
    width = 0.2;

    omegatemp = width*(rand(1,N)-1);
    meanomega = mean(omegatemp);
    omega = omegatemp - meanomega;
    sto = std(omega);

    node = makeSW(N,m,p);
     [synch,l2,lmax] = eigenlap(node);
     displine('synch = ',synch)
     [cluscoef,eicoef,clus,ei] = clustercoef(node);
     displine('cluscoef = ',cluscoef)

    nodecouple = node;

    for loop = 1:N
        nodecouple(loop).element = omega(loop);
        lnk(loop) = nodecouple(loop).numlink;
    end

    avgdegree = mean(lnk);
    displine('avgdegree =',avgdegree)

end         % end if example

mnomega = 1;

if writeavi == 1
    figure(2)
    clf
    colormap(h)
    caxis([-0.1 0.1])
    colorbar
end

for facloop = 1:Nfac
    facloop
    tic

    if example == 1     % global
```

```
        facoef = 0.2;

    elseif example == 2  % distributed example
        facoef = 0.3;

    elseif example == 3     % Square lattice
        facoef = 0.3;

    elseif example == 4     % Linear cycle
        facoef = 1.3;

    elseif example == 5     % Tree
        facoef = 1;

    elseif example == 6     % ER graph
        facoef = 0.5;

     elseif example == 7     % scale-free graph
        facoef = 0.5;

     elseif example ==8     % small-world graph
        facoef = 0.5;

    end

    fac = facoef*(16*facloop/(Nfac))*(1/avgdegree)*sto/mnomega;
    [dum,nodesz] = size(nodecouple);
    for nodeloop = 1:nodesz
        [dum,linksz] = size(nodecouple(nodeloop).link);
        for linkloop = 1:linksz
            nodecouple(nodeloop).coupling(linkloop) = fac;
        end
    end

    if facloop == 1     % Print the g-matrix
        for omloop = 1:N
            linksz = node(omloop).numlink;
            for cloop = 1:linksz
                ix = nodecouple(omloop).link(cloop);
                gtemp(omloop,ix) = 1;
            end
        end

        figure(100)
        imagesc(gtemp)
        colormap(jet)
        yyy = 1;
    end

    facval(facloop) = fac*avgdegree;

    if inexcoupling == 1        % internal coupling

        omegout = coupleN(nodecouple,0);                                % Here is
the subfunction call for the flow
```

```
    elseif inexcoupling == 2        % external coupling
        g = 1;
        fac = 0.05*1.16^(30*facloop/Nfac);
        gext = fac*(1/N)*sto/mnomega;

        omegout = coupleN0(omega,zeros(N,N),gext);       % Here is the
subfunction call for the flow

    end     % end if coupling

    if example == 30         % square lattice
        ind = 0;
        for yloop = 1:Row
            for xloop = 1:Col
                ind = ind+1;
                A(yloop,xloop) = omegout(ind);
            end
        end

        figure(2)
        imagesc(A)
        colormap(h)
        caxis([-width/2 width/2])
        if writeavi
            frame = getframe(gcf);
            aviobj = addframe(aviobj,frame);
        end
    end

    stdev(facloop) = std(omegout);

    [y,x] = histfix(omegout,11,-0.1,0.1);
    %    figure(3)
    %    plot(x,y)
    mx(facloop) = max(y);

    xx(N*(facloop-1)+1:facloop*N) = ones(1,N)*facval(facloop);
    yy(N*(facloop-1)+1:facloop*N) = omegout;

    for omloop = 1:N
        yyy(facloop,omloop) = omegout(omloop);
    end
    xxx(facloop) = facval(facloop);

    tictoc(facloop) = toc;

    S = whos;
    [Ssz,dum] = size(S);
    stemp = 0;
    for sloop = 1:Ssz
        stemp = stemp + S(sloop).bytes;
    end
    bytesize(facloop) = stemp;

end     % end facloop
```

```
duration = sum(tictoc)

figure(4)
plot(bytesize)
title('Byte Size')

figure(5)
plot(tictoc)
title('Durations')

figure(6)
plot(xx,yy,'.')

figure(7)
hold on
for omloop = 1:N
    plot(xxx,yyy(:,omloop),'.-','Color',ch(round(34-
omloop*32/N),:),'LineWidth',1.1)
end
hold off
set(gcf, 'color', 'white')
%axes('LineWidth',1.1,'FontSize',14)

xp = 1:Nfac;

mins = min(stdev);
maxs = max(stdev);
%Responst = (1 - (stdev-mins)/(maxs-mins)).^2;
Responst = (1 - stdev/maxs);

minm = min(mx);
maxm = max(mx);
%Responmx = (mx-minm)/(maxm-minm);
Responmx = (mx-minm)/(1-minm);
figure(8)
plot(facval,Responst,'r',facval,Responmx,'b')
title('Respon')
legend('std','max')

Printfile3('cNdout',facval,Responst,Responmx)


if writeavi
    aviobj = close(aviobj);
end
```

node2distance.m

Convert a network to a distance matrix.

```
% function dist = node2distance(node)
% node is the input network
```

```
% dist is the distance matrix
% disconnected cluster distance is set to -1


function [dist maxdis] = node2distance(node, maxset, varargin)

[sy,N] = size(node);

dist = zeros(N,N);
for vertex = 1:N
    list = node(vertex).link;

    b = zeros(N,1);
    vec = zeros(N,1);
    vec(vertex) = 1;
    b = vec;

    A = adjacency(node);

    endtest = 1;
    bold = b;
    index = 0;
    while (endtest ~= -1)&(index < 20)
        index = index + 1;
        bnew = bold + A*bold;

        dif = bnew - bold;
        for loop = 1:N
            if (bnew(loop) >0)&(bold(loop) == 0)
                dist(vertex,loop) = index;
            end

        end

        endtest = where(bnew,0);
        bold = bnew;

    end

end

maxdis = max(max(dist));
if nargin == 2
    setmax = maxset;
else
    setmax = maxdis+1;
end
for yloop = 1:N-1
    for xloop = yloop+1:N
        if dist(yloop,xloop) == 0
            %dist(yloop,xloop) = -1;
            %dist(xloop,yloop) = -1;
            dist(yloop,xloop) = setmax;
            dist(xloop,yloop) = setmax;
```

```
        end
    end
end
```

adjacency.m

Convert a network to an adjacency matrix

```
% function [A,degree,Lap] = adjacency(node)
% Extract adjacency matrix, node degree and Laplacian of a graph
% node structure is ...
% node(1).element = node number;
% node(1).numlink = number_of_links;
% node(1).link = [set of linked node numbers];
function [A,degree,Lap] = adjacency(node)

[dum,N] = size(node);
A = zeros(N,N);
Lap = zeros(N,N);

for Nloop = 1:N

    [dum,L] = size(node(Nloop).link);

    for Lloop = 1:L

        A(Nloop,node(Nloop).link(Lloop)) = 1;
        A(node(Nloop).link(Lloop), Nloop) = 1;

    end     % end Lloop

end     % end Nloop

degree = sum(A);

Lap = -A;
for loop = 1:N
    Lap(loop,loop) = degree(loop);
end
```

clusternum.m

Calculate number of disconnected clusters.

```
% function numclus = clusternum(node)
% Number of disconnected clusters

function numclus = clusternum(node)

[dum,N] = size(node);
[A,degree,Lap] = adjacency(node);
LamL = eig(Lap);
```

```
% Calculate number of disconnected clusters
dflag = 0; loop = 0; cnt = 0;
while dflag == 0
    loop = loop + 1;
    if LamL(loop) < 1e-3
        cnt = cnt + 1;
    else
        dflag = 1;
    end
end
numclus = cnt;
```

clusterstats.m

Statistics of a network.

```
% %
function[N,e,avgdegree,maxdegree,mindegree,numclus,meanclus,Lmax,L2,LmaxL2,me
andistance,diam] = clusterstats(node)
% Generates statistics on selected  graphs
% N = size of network
% e = number of edges
% avgdegree = average degree
% maxdegree = maximum degree
% mindegree = minimum degree
% numclus = number of clusters
% meanclus = cluster coef
% Lmax = maximum eigenvalue
% L2 = second eigenvalue
% LmaxL2 = ratio of Lmax to L2
% meandistance = mean of the distances

function[N,e,avgdegree,maxdegree,mindegree,numclus,meanclus,Lmax,L2,LmaxL2,me
andistance,diam] = clusterstats(node)

[dum,N] = size(node);
[A,degree,Lap] = adjacency(node);

e = sum(degree)/2;  % number of edges
avgdegree = mean(degree);
maxdegree = max(degree);
mindegree = min(degree);
loop2 = trace(A^2);

[cluscoef,eicoef,clus,ei] = clustercoef(node);
meanclus = cluscoef;
dis = node2distance(node);
meandistance = mean(mean(dis))*(N^2/(N*(N-1)));
diam = max(max(dis));

Lam = eig(A);

LamL = eig(Lap);
Lmax = LamL(N);

% Calculate number of disconnected clusters
```

```
dflag = 0; loop = 0; cnt = 0;
while dflag == 0
    loop = loop + 1;
    if loop <=N
        if LamL(loop) < 0.1/N
            cnt = cnt + 1;
        else
            dflag = 1;
        end
    else
        dflag = 1;
    end
end
numclus = cnt;

if numclus == 1
    L2 = LamL(2);
    LmaxL2 = Lmax/L2;
else
    L2 = 0;
    LmaxL2 = 0;
end
```

clustercoef.m

Calculate cluster coefficient of a network.

```
%function  [cluscoef,eicoef,clus,ei] = clustercoef(node)
% cluscoef = average of 2E/k(k-1)
% eicoef = average number of shared edges
% clus = clustering of node
% ei = shared edges of node

function [cluscoef,eicoef,clus,ei] = clustercoef(node)

[Adjac,degree,Lap] = adjacency(node);

[dum,N] = size(Adjac);

for iloop = 1:N
    temp = 0;
    for rowloop = 1:N
        for coloop = 1:N
            temp = temp +
0.5*Adjac(iloop,rowloop)*Adjac(rowloop,coloop)*Adjac(coloop,iloop);
        end
    end

    ei(iloop) = temp;
    ki = node(iloop).numlink;

    if ki > 1
        clus(iloop) = 2*ei(iloop)/ki/(ki-1);
    else
        clus(iloop) = 0;
    end
```

```
end

cluscoef = mean(clus);
eicoef = mean(ei);
```

avgeigs.m

Eigenvalues of the Laplacian for several networks.

```
% Companion to IMD
N = 50;

Nrep = 100;
eigvalER = zeros(1,N);
eigvalSF = zeros(1,N);
eigvalSW = zeros(1,N);
eigER = zeros(1,N);
eigSF = zeros(1,N);
eigSW = zeros(1,N);
for loop = 1:Nrep

    nodeER = makeER(N,0.08);
    nodeSF = makeSF(N,2);
    nodeSW = makeSW(N,2,0.1);

    [AER,degree,LapER] = adjacency(nodeER);
    [VER,DER] = eig(LapER);
    [vER,dER] = eig(AER);
    [ASF,degree,LapSF] = adjacency(nodeSF);
    [VSF,DSF] = eig(LapSF);
    [sSF,dSF] = eig(ASF);
    [ASW,degree,LapSW] = adjacency(nodeSW);
    [VSW,DSW] = eig(LapSW);
    [vSW,dSW] = eig(ASW);
    for nloop = 1:N
        eigvalER(nloop) = eigvalER(nloop) + DER(nloop,nloop)/Nrep;
        eigvalSF(nloop) = eigvalSF(nloop) + DSF(nloop,nloop)/Nrep;
        eigvalSW(nloop) = eigvalSW(nloop) + DSW(nloop,nloop)/Nrep;
        eigER(nloop) = eigER(nloop) + dER(nloop,nloop)/Nrep;
        eigSF(nloop) = eigSF(nloop) + dSF(nloop,nloop)/Nrep;
        eigSW(nloop) = eigSW(nloop) + dSW(nloop,nloop)/Nrep;
    end

end

x = 1:N;
figure(1)
plot(x,eigvalER,x,eigvalSF,x,eigvalSW)
legend('ER','SF','SW')
title('Graph Laplacian')
```

diffusiondriver.m

Diffusion on a nework.

```
% diffusiondriver.m
% 3-24-12

clear

close all

N = 128;
p = 0.05;
m = 4;
beta = 0.01;


%node = makeER(N,0.06);
%node = makeSF(N,m);
node = makeSW(N,m,0.1);


% A = adjacency(node);
% A = ham2adj(N);
% node = adj2node(A);

[N,e,avgdegree,maxdegree,mindegree,numclus,meanclus,Lmax,L2,LmaxL2] =
clusterstats(node);

disp(' ')
displine('Number of nodes = ',N)
disp(strcat('Number of edges = ',num2str(e)))
disp(strcat('Mean degree = ',num2str(avgdegree)))
displine('Maximum degree = ',maxdegree)
disp(strcat('Number of clusters = ',num2str(numclus)))
disp(strcat('mean cluster coefficient = ',num2str(meanclus)))
disp(' ')
disp(strcat('Lmax = ',num2str(Lmax)))
disp(strcat('L2 = ',num2str(L2)))
disp(strcat('Lmax/L2 = ',num2str(LmaxL2)))
disp(' ')

 [A,degree,Lap] = adjacency(node);

 [V,D] = eig(Lap);

 for loop = 1:N
     eigval(loop) = D(loop,loop);
 end

 figure(1)
 plot(eigval)
 title('Eigenvalues')
```

```
% initial values
c = zeros(N,1);
c(1) = 1;

% eigvec decomposition

for eigloop = 1:N
    Vtemp = V(:,eigloop);
    v(eigloop) = sum(c.*Vtemp);
end

% time loop
Ntime = 100;
for timeloop = 1:Ntime       % 200

    for nodeloop = 1:N

        temp = 0;
        for eigloop = 1:N

            temp = temp + V(nodeloop,eigloop)*v(eigloop)*exp(-
eigval(eigloop)*beta*(timeloop-1));

        end    % end eigloop

                        concentration(timeloop,nodeloop) = temp;

    end    % endnodeloop


end % end timeloop

figure(2)
imagesc(real(log(concentration)))
colormap(jet)
colorbar
caxis([-10 0])
title('Log Concentrations vs. time')
xlabel('Node Number')

figure(3)
plot(concentration(100,:))
title('Ending Concentrations')
xlabel('Node Number')


 x = 0:Ntime-1;
h = colormap(jet);
figure(4)
for nodeloop = 1:N
    rn = round(rand*63 + 1);
    y =  concentration(:,nodeloop)+0.001;
    semilogy(x,y,'Color',h(rn,:))
     hold on
end
```

```
 hold off
title('Concentrations vs. time')


 x = 0:Ntime-1;
 h = colormap(jet);
 figure(5)
 for nodeloop = 1:10
     rn = round(rand*63 + 1);
     y =  concentration(:,nodeloop*10)+0.001;
     %semilogy(x,y,'Color',h(rn,:),'LineWidth',1.1)
     semilogy(x,y,'k','LineWidth',1.2)
      hold on
 end
 hold off
 set(gcf,'Color','White')
 title('Selected Nodes: Continuous time')


 % Now try the discrete-time-map approach

 c0 = c;
 dt = 1;     % 5
 M = eye(N,N) - beta*Lap*dt;

 for timeloop = 1:200   %40

     c = (M^timeloop)*c0;

     Con(timeloop,:) = c';
 end

 x = 0:1:199;   % 0:5:199
 h = colormap(jet);
 figure(6)
 for nodeloop = 1:10
     rn = round(rand*63 + 1);
     y =  Con(:,nodeloop*10)+0.001;
     %semilogy(x,y,'Color',h(rn,:),'LineWidth',1.1)
     semilogy(x,y,'k','LineWidth',1.2)
      hold on
 end
 hold off
 set(gcf,'Color','White')
 title('Selected Nodes: Discrete time')
```

SIRS.m

Susecpt-Infect-Remove-Suscept model.

```
% SIRS.m
% converted from diffusionmat on 1/4/16
```

```
% Susecpt-Infect-Remove-Suscept  plus an innoculation of highest-degree node

clear

T1 = 50;
T2 = 100;
N = 50;          % 50
beta = 0.2;      % infection rate  0.2
mu = 0.6;           % recovery rate 0.45

p = 0.1;        %
m = 2;

node = makeSF(N,m);      % N = 50 m = 2  beta = 0.2  mu = 0.7
%node = makeER(N,p);       % N = 50 p = 0.05 beta = 0.2 mu = 0.4
%node = makeSW(N,m,p);     % N = 50 m = 2 p = 0.05 beta = 0.2 mu = 0.4

[A,degree,Lap] = adjacency(node);
[V,D] = eig(Lap);

% initial values
a = zeros(N,1);
[Y,I] = max(degree);
a(I) = 1;

dt = 1;
R = eye(N,N);
c = a;

for timeloop = 1:T1

    M = eye(N,N) + beta*A*dt.*randbin2(N,N,1-beta).*(ones(N,N)-eye(N,N));

    ctmp = M*c;

    ctmp2 = ceil(ctmp);
    c = maskbilevel(ctmp2,0,1.01,0,1);

    ctmp3 = R*c;
    ctmp4 = floor(ctmp3);
    c = maskbilevel(ctmp4,-0.01,1.01,0,1);

    Con(timeloop,:) = c';

    for nodeloop = 1:N
        node(nodeloop).value = c(nodeloop);
        Rtmp(nodeloop,nodeloop) = c(nodeloop);
    end

    R = eye(N,N) - (Rtmp.*mu*dt.*eye(N,N).*randbin2(N,N,1-mu));

end

% function y = randbin2(M,N,thresh)
% random binary matrix of length M by N
```

```
% thresh between 0 and 1
% p(1) = 1-thresh
% p(0) = thresh

function y = randbin2(M,N,thresh)

temp = rand(M,N);
y = zeros(M,N);
for mloop = 1:M
    for nloop = 1:N
        if temp(mloop,nloop) > thresh
            y(mloop,nloop) = 1;
        end
    end
end

% maskbilevel.m
% function [y mask] = masklevel(A,lowcut,hicut,vallo,valhi)
% Assigns values vallo and valhi to outside of cuts
% mask is binary 1's where inside cutvalues

function [y mask] = maskbilevel(A,lowcut,hicut,vallo,valhi)

% slope = 0.00001*(hicut-lowcut);
%mtemp = clip(A,lowcut,hicut,slope);
mtemp = heaviside0(A-lowcut).*heaviside1(hicut-A);
y = A.*mtemp + valhi*heaviside1(A-hicut).*(1-mtemp) +
vallo*heaviside0(lowcut-A).*(1-mtemp);
mask = mtemp;


% function y = heaviside1(x)
% y = 1 at x = 0

function y = heaviside1(x)
[sy, sx] = size(x);
for yloop = 1:sy
    for xloop = 1:sx

        if x(yloop,xloop)<0
            y(yloop,xloop) = 0;
        elseif x(yloop,xloop) == 0
            y(yloop,xloop) = 1;
        elseif x(yloop,xloop) > 0
            y(yloop,xloop) = 1;
        end
    end
end

% function y = heaviside0(x)
% y = 0 at x = 0

function y = heaviside0(x)
[sy, sx] = size(x);
for yloop = 1:sy
    for xloop = 1:sx
```

```
        if x(yloop,xloop)<0
            y(yloop,xloop) = 0;
        elseif x(yloop,xloop) == 0
            y(yloop,xloop) = 0;
        elseif x(yloop,xloop) > 0
            y(yloop,xloop) = 1;
        end
    end
end
```

Chapter 6 Neural Networks

NaK.m
threelayer.m
logistic.m

NaK model of a single neuron.

NaK.m

```
function NaK

% I = 12, EL = -78, Vn = -45, tau = 1              pg. 117  supercritical
Hopf bifurcation
% I = 4.53, EL = -80, Vn = -25, tau = 1            pg. 113  Homoclinic
% I = 3, EL = -80, Vn = -25, tau = 0.152           pg. 110  Bistability
y0=[-20,0.4],[-70,0.1]
% I=43-50,gL=1,gNa=gK=4,Vm=-30,km=7,EL=-78,Vn=-45,tau=1   pg. 176
subcritical Hopf

I = 50;      % 10
gL = 8;      % 8
EL = -78;    % -80   -78
gNa = 20;     % 20
ENa = 60;    % 60
gK = 10;      % 10
EK = -90;    %-90
C = 1;       % 1
Vn = -45;    % -25   -45
kn = 5;      % 5
Vm = -20;    % -20
km = 15;      % 15
tau = 1;     % 1  0.152


%y0 = [-20  0.4];
%y0 = [-70  0.1];
%y0 = [-50  0.25];
y0 = [-61,0.001];
tspan = [0 100];

figure(1)
options = odeset('OutputFcn',@odephas2);
[t,y] = ode45(@f5,tspan,y0,options);

figure(2)
plot(t,y(:,1),t,y(:,2))


figure(3)
```

```
plot(y(:,1),y(:,2))
set(gcf, 'color', 'white')

for loop = 1:100
    xp(loop) = -80 + 100*loop/100;
    ninf = 1./(1 + exp((Vn - xp(loop))/kn));
    minf = 1./(1 + exp((Vm - xp(loop))/km));
    yp(loop) = ninf;
    yyp(loop) = (I - gL*(xp(loop) - EL) - gNa*minf*(xp(loop) -
ENa))/(gK*(xp(loop) - EK));
end

hold on
plot(xp,yp,'r',xp,yyp,'k')
hold off

printfile = 0;

if printfile == 1
    [sz,dum] = size(t);
    ty1 = y(1:300,1)';
    ty2 = y(1:300,2)';
    tyV = y(1:sz,1);
    Printfile3('Nullclines.txt',xp,yp,yyp);
    Printfile2('Trajectory.txt',ty1,ty2);
    Printfile2('TimeSeries.txt',t',tyV');
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    function yd = f5(t,y)

        ninf = 1./(1 + exp((Vn - y(1))/kn));
        minf = 1./(1 + exp((Vm - y(1))/km));

        yp(1) = I/C - gL*(y(1) - EL)/C - gNa*minf*(y(1) - ENa)/C -
gK*y(2).*(y(1) - EK)/C;
        yp(2) = (ninf - y(2))./tau;



        yd = [yp(1);yp(2)];

    end     % end f5

end
```

Three-layer Perceptron.

threelayer.m

```
% threelayer.m
```

```
clear

format compact
disp('%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%')

alpha = 0.1;
gain = 1;

ex = 3;

if ex == 1
    M = 4;  % number of inputs
    L = 7;  % number of hidden neurons
    N = 1;  % number of outputs

    mu = 8; % number of training cases


    X(:,1) = [1; -1; 1; -1];
    X(:,2) = [1; 1; -1; -1];
    X(:,3) = [1; -1; -1; 1];
    X(:,4) = [1; 1; 1; -1];
    X(:,5) = [-1; -1; 1; -1];
    X(:,6) = [-1; -1; 1; 1];
    X(:,7) = [-1; -1; -1; -1];
    X(:,8) = [1; 1; 1; 1];

    ZT(:,1) = 1;
    ZT(:,2) = 1;
    ZT(:,3) = 1;
    ZT(:,4) = -1;
    ZT(:,5) = -1;
    ZT(:,6) = 1;
    ZT(:,7) = -1;
    ZT(:,8) = -1;

    w = 5;

    W1 = w*randn(L,M);
    b1 = w*randn(L,1);
    W2 = w*randn(N,L);
    b2 = w*randn(N,1);


elseif ex == 2  % XOR

    M = 2;  % number of inputs
    L = 2;  % number of hidden neurons
    N = 1;  % number of outputs

    mu = 4; % number of training cases


    X(:,1) = [-1; -1];
    X(:,2) = [-1; 1];
```

```
    X(:,3) = [1; -1];
    X(:,4) = [1; 1];

    ZT(:,1) = -1;
    ZT(:,2) = 1;
    ZT(:,3) = 1;
    ZT(:,4) = -1;

    w = 5;

    W1 = [w w;w w];
    b1 = [w;-w];
    W2 = [-2*w w];
    b2 = 2*w;


elseif ex == 3  % AND

    M = 4;  % number of inputs
    L = 5;  % number of hidden neurons
    N = 1;  % number of outputs

    mu = 8; % number of training cases


    X(:,1) = [1;1;1;1];
    X(:,2) = [1;1;1;-1];
    X(:,3) = [1;1;-1;1];
    X(:,4) = [-1;-1;-1;-1];
    X(:,5) = [-1;-1;-1;1];
    X(:,6) = [-1;1;1;1];
    X(:,7) = [1;-1;-1;1];
    X(:,8) = [-1;1;-1;-1];

    ZT(:,1) = 1;
    ZT(:,2) = -1;
    ZT(:,3) = -1;
    ZT(:,4) = -1;
    ZT(:,5) = -1;
    ZT(:,6) = -1;
    ZT(:,7) = -1;
    ZT(:,8) = -1;

    w = 5;

    W1 = w*randn(L,M);
    b1 = w*randn(L,1);
    W2 = w*randn(N,L);
    b2 = w*randn(N,1);

elseif ex == 4  % OR

    M = 4;  % number of inputs
    L = 5;  % number of hidden neurons
    N = 1;  % number of outputs
```

```
    mu = 8; % number of training cases


    X(:,1) = [1;1;1;1];
    X(:,2) = [1;1;1;-1];
    X(:,3) = [1;1;-1;1];
    X(:,4) = [-1;-1;-1;-1];
    X(:,5) = [-1;-1;-1;1];
    X(:,6) = [-1;1;1;1];
    X(:,7) = [1;-1;-1;1];
    X(:,8) = [-1;1;-1;-1];

    ZT(:,1) = 1;
    ZT(:,2) = 1;
    ZT(:,3) = 1;
    ZT(:,4) = -1;
    ZT(:,5) = 1;
    ZT(:,6) = 1;
    ZT(:,7) = 1;
    ZT(:,8) = 1;

    w = 5;

    W1 = w*randn(L,M);
    b1 = w*randn(L,1);
    W2 = w*randn(N,L);
    b2 = w*randn(N,1);


end

dev = 0.5*w;

W1 = W1 + dev*randn(L,M); % initial weight matrix : input to hidden
W2 = W2 + dev*randn(N,L); % hidden to output

b1 = b1 + dev*randn(L,1);
b2 = b2 + dev*randn(N,1);
%b2 = zeros(N,1);



for muloop = 1:mu

    In = X(:,muloop);
    vj = W1*In - b1;
    Y = logistic(vj,1);
    vk = W2*Y - b2;
    Z = logistic(vk,1);
    displine('Z = ',Z);

end

itnum = 0;
```

```
eps = 1;
mx = 800;
while (eps > 0.001)&(itnum < mx)
    itnum = itnum + 1;

    %rz = randintexc(mu,mu); % randomize
    rz = 1:mu;

    eptest = 0;
    Djsum = zeros(1,L);
    for muloop = 1:mu   % loop through one epoch



        % Forward propagation

        In = X(:,rz(muloop));
        vj = W1*In - b1;
        Y = logistic(vj,1);
        vk = W2*Y - b2;
        Z = logistic(vk,1);

        for kloop = 1:N
            Er(kloop) = 0.5*(ZT(kloop,rz(muloop)) -
logistic(gain*vk(kloop),1)).^2;
            eptest = eptest + Er(kloop);
        end



        % Error back-propagation

        for kloop = 1:N
            Deltak(kloop) = (ZT(kloop,rz(muloop)) -
logistic(gain*vk(kloop),1))*dlogistic(gain*vk(kloop),1);
            b2(kloop) = b2(kloop) - alpha*Deltak(kloop);
            for jloop = 1:L
                W2temp(kloop,jloop) = W2(kloop,jloop) +
alpha*Deltak(kloop)*Y(jloop);
            end
        end



        for jloop = 1:L

            sumtemp = 0;
            for kloop = 1:N
                sumtemp = sumtemp + Deltak(kloop)*W2(kloop,jloop);
            end

            Deltaj = sumtemp*dlogistic(gain*vj(jloop),1);
            b1(jloop) = b1(jloop) - alpha*Deltaj;
            for iloop = 1:M
                W1(jloop,iloop) = W1(jloop,iloop) + alpha*Deltaj*In(iloop);
            end
```

```
            Djsum(jloop) = Djsum(jloop) + Deltaj;
        end

        W2 = W2temp;

        Dk(muloop) = sum(abs(Deltak));


    end     % end muloop

    Dj(:,itnum) = Djsum';

    Dkt(itnum,:) = Dk;

    eps = eptest/mu;

    ep(itnum) = eps;
    %pause(0.25)

end

figure(1)
plot(ep)

[dum,sx] = size(ep);
xx = 1:sx;
h = colormap(jet);
figure(2)
hold on
for jloop = 1:L
    yy = Dj(jloop,:);
    plot(xx,yy,'Color',h(jloop*10,:))
end
hold off



for muloop = 1:mu
    In = X(:,muloop);
    vj = W1*In - b1;
    Y = logistic(vj,1);
    vk = W2*Y - b2;
    Z = logistic(vk,1);
    displine('Zfinal = ',Z);
end
disp(' ')

for loop1 = 1:2
    for loop2 = 1:2
        for loop3 = 1:2
            for loop4 = 1:2
                In = [-1+2*(loop1-1);-1+2*(loop2-1);-1+2*(loop3-1);-
1+2*(loop4-1)];
                vj = W1*In - b1;
                Y = logistic(vj,1);
```

```
                vk = W2*Y - b2;
                Z = logistic(vk,1);
                displine('Zprobe = ',Z);
            end
        end
    end
end

displine('ep = ',ep(sx))
```

Logistic sigmoid function (not to be confused with logistic map.)

logistic.m

```
% function y = logistic(x,z,varargin)
% z = 0   y = [0, 1]
% z = 1   y = [-1, 1]

function y = logistic(x,z,varargin)

if (nargin == 1)|(z == 0)

    y = 1./(1 + exp(-x));

else

    y = 2*(1./(1 + exp(-x)) - 0.5);

end
```

Chapter 7 Evolutionary Dynamics

PredPrey.m
repeq.m
quasiSpec.m
quasiHam.m
hamming.m
stochasticmatrix.m
randunitary.m
repmut.m
NK.m

Predator-Prey model.

PredPrey.m

```
% PredPrey.m
% 9-3-11
%

function PredPrey

alpha = 3;
beta = 2;
gamma = 2;
delta = 2.5;

xinf = gamma/delta;
yinf = alpha/beta;
eps = .1;

options = odeset('RelTol',1e-6,'AbsTol',1e-7);


%y0 = [(xinf + eps) (yinf + 0.0)];

y0 = [2 2];

tspan = [0 200];

[t,y] = ode45(@f5,tspan,y0);

figure(1)
plot(t,y(:,1),t,y(:,2))

figure(2)
plot(y(:,1),y(:,2))
axis([0 3 0 4])
```

```
    function yd = f5(t,y)

        yp(1) = y(1)*(alpha - beta*y(2));
        yp(2) = -y(2) *(gamma - delta*y(1));

        yd = [yp(1);yp(2)];

    end     % end f5

end
```

Replicator Equation.

repeq.m

```
% replicator equation
% 3-10-12

function repeq

N = 7;
asymm = 2;      % 1 = zero diag   2 = zero trace
phi0 = 0.05;            % average fitness (positive number) damps
oscillations

displine('N = ',N)
displine('asymm = ',asymm)
displine('phi0 = ',phi0)

h = newcolormap('fluorodark');

tempx = rand(1,N);
x0 = tempx/sum(tempx);% initial populations

node = makeER(N,0.99);
%node = makeSW(N,N/32,0.25);
%node = makeSF(N,N/32);
Adj = adjacency(node);
numclus = clusternum(node);
[N,e,n] = clusterstats(node);
displine('number of clusters = ',numclus)
displine('average degree = ',n)

if asymm == 1
    % Asymmetric payoff matrix with zero diagonals
    A = zeros(N,N);
    for yloop = 1:N
        for xloop = yloop + 1:N
            A(yloop,xloop) = 2*(0.5 - rand);
            A(xloop,yloop) = -A(yloop,xloop);
        end
    end

else
```

```
    % Asymmetric payoff matrix with zero trace
    Atemp = zeros(N,N);
    for yloop = 1:N
        for xloop = yloop +1:N
            Atemp(yloop,xloop) = 2*(0.5 - rand);
            Atemp(xloop,yloop) = -Atemp(yloop,xloop);
        end
        Atemp(yloop,yloop) = 2*(0.5 - rand);
    end
    tr = trace(Atemp);
    A = Atemp;
    for yloop = 1:N
        A(yloop,yloop) = Atemp(yloop,yloop) - tr/N;
    end

end % end if asymm




A = A.*Adj;

figure(1)
imagesc(A);
colormap(h)
colorbar

tspan = [1 1600];
[t,x] = ode45(@f5,tspan,x0);

[sy,sx] = size(x);

pop = sum(x,2);
av = mean(mean(x));
displine('population = ',mean(pop))
displine('av pop =', mean(mean(x)));

% How many non-zero?
cnt = 0;
for loop = 1:N
    if mean(x(sy-100:sy,loop),1) > av/20
        cnt = cnt + 1;
    end
end
displine('cnt = ',cnt)

% Average max slope?
% for loop = 1:N
%     mx = max(x(200:sy,loop));
%     deriv(:,loop) = diff(x(200:sy,loop));
%     freq(loop) = mean(abs(deriv(:,loop)))/mx;
% end
% mnfreq = mean(freq);
% displine('mean frequency = ',mnfreq)

figure(2)
```

```
for loop = 1:N
    plot(t,x(:,loop),'Color',h(round(loop*64/N),:),'LineWidth',1.25)
    %plot(t,x(:,loop),'k','LineWidth',1.25)
    hold on
end
hold off

figure(3)
plot(x(:,1),x(:,N))

figure(4)
plot(t,x(:,1),t,x(:,round(N/2)),t,x(:,N))



    function yd = f5(t,y)

        for iloop = 1:N
            ftemp = 0;
            for jloop = 1:N
                ftemp = ftemp + A(iloop,jloop)*y(jloop);
            end     % end jloop
            f(iloop) = ftemp;
        end     % end iloop

        phitemp = phi0;          % Can adjust this from 0 to 1 to stabilize
(but Nth population is no longer independent)
        for loop = 1:N
            phitemp = phitemp + f(loop)*y(loop);
        end
        phi = phitemp;

        for loop = 1:N-1
            yd(loop) = y(loop)*(f(loop) - phi);
        end

        if abs(phi0) < 0.01             % average fitness maintained at zero
            yd(N) = y(N)*(f(N)-phi);
        else                                        % non-zero average fitness
            ydtemp = 0;
            for loop = 1:N-1
                ydtemp = ydtemp - yd(loop);
            end
            yd(N) = ydtemp;
        end

        yd = yd';


    end     % end f5


end     % end repeq
```

Quasi-species equation.

quasiSpec.m

```
% Quasispecies simulation.  Includes mutation with Hamming distance.

function quasiSpec

%close all
h = colormap(lines);

randpop = 1;     % 0) = spike population; 1) = random population
mutype = 0;      % 0) = Hamming; 1) = rand
fitype = 3;      % 0) = Hamming; 1) = 2-peak; 2) = rand+gauss;  3) freq-dep

B = 7;
N = 2^B;          % size of mutation space (64)
lam = 1;         % Hamming fitness only
gamma = 1;     % freq-dep fitness (payoff matrix only)
relran = 0.025;       % relative random contrib to fitness
time_expand = 50;

ep = 0.0290;       % average mutation rate: 0.1 to 0.01 typical  (0.4835)


%%%%% Set original population
if randpop == 1
    rng(0);
    x0temp = rand(1,N);     % Initial population
    sx = sum(x0temp);
    x0 = x0temp/sx;
else
    x0 = zeros(1,N);
    x0(1) = 0.667; x0(2) = 0.333;
end

Pop0 = sum(x0);


%%%%%% Set Hamming distance
for yloop = 1:N
    for xloop = 1:N
        H(yloop,xloop) = hamming(yloop-1,xloop-1);
    end
end



%%%%%%% Set Mutation matrix
if mutype == 0
    Qtemp = 1./(1+H/ep);     %Mutation matrix on Hamming
    %Qtemp = exp(-H/(ep*50));
    Qsum = sum(Qtemp,2);
```

```
    % Normalize mutation among species
    for yloop = 1:N
        for xloop = 1:N
            Q(yloop,xloop) = Qtemp(yloop,xloop)/Qsum(xloop);
        end
    end

end
if mutype == 1
    rng(0);
    S = stochasticmatrix(N);
    Stemp = S - diag(diag(S));
    Qtemp = ep*Stemp;
    sm = sum(Qtemp,2)';
    Q = Qtemp + diag(ones(1,N) - sm);
end


%keyboard


%%%%%%% Set fitness landscape

if fitype == 0      % Hamming
    x = 1:N;
    alpha = 84;
    ftemp = exp(-lam*H(alpha,:));   % Fitness landscape
    sf = sum(ftemp);
    f = ftemp/sf;
end
if fitype == 1      % double peak and rand
    rng(1);
    f = rand(1,N);
    x = 1:N;
    delg = 20;
    sig1 = 1;
    sig2 = 4;
    g1 = gaussprob(x,(N/2 - delg),sig1);
    g2 = 3*gaussprob(x,(N/2 + delg),sig2);
    ftemp = relran*f + g1 + g2;
    f = ftemp/sum(ftemp);
end
if fitype == 2      % rand + Gauss
    rng(0);
    f = rand(1,N);
    x = 1:N;
    ftemp = relran*f + gauss((x-N/2)/2);   % Fitness landscape
    f = ftemp/sum(ftemp);
end
if fitype == 3      % frequency-dependent Hamming
    avgdis = mean(mean(H));
    %payoff = exp(-gamma*(H - avgdis));    % payoff matrix
    %payoff = H.^2;
    %payoff = ones(size(H));
    payoff = exp(-gamma*H);
end
```

```
%keyboard

% Run time evolution
tspan = [0 1000];
[t,x] = ode45(@quasispec,tspan,x0);

Pop0
[sz,dum] = size(t);
Popend = sum(x(sz,:))

phistar = sum(f.*x(sz,:))       % final average fitness


figure(1)
plot(f,'-')
hold on
figure(1)
plot(x(sz,:),'r')
hold off


figure(2)
for loop = 1:N
    semilogx(t,x(:,loop),'Color',h(round(loop*64/N),:),'LineWidth',1.25)
    hold on
end
hold off
set(gcf,'Color','white')
xlabel('Time','FontSize',14)
ylabel('Population','FontSize',14)
hh = gca;
set(hh,'FontSize',14)


figure(3)
for loop = 1:N
    plot(t,x(:,loop),'Color',h(round(loop*64/N),:))
    hold on
end
hold off

figure(4)
for loop = 1:N
    loglog(t,x(:,loop),'Color',h(round(loop*64/N),:))
    hold on
end
hold off

figure(5)
for loop = 1:N
    semilogy(t,x(:,loop),'Color',h(round(loop*64/N),:))
    hold on
end
hold off
```

```
% Eigenvalues

[V,D] = eig(W);

max(D(:,1))

figure(6)
%semilogy(abs(V(:,1)))
plot((V(:,1)))

disp(' ')


if fitype == 1
    xlo = N/2 - delg - 2*sig1;
    xhi = N/2 - delg + 2*sig1;
    fit44 = sum(f(xlo:xhi))
    pop44 = sum(x(sz,xlo:xhi))/Popend
    xlo = N/2 + delg - 2*sig2;
    xhi = N/2 + delg + 2*sig2;
    fit84 = sum(f(xlo:xhi))
    pop84 = sum(x(sz,xlo:xhi))/Popend

end

function yd = quasispec(~,y)

        if fitype == 3      % frequency-dependent Hamming
            for loop = 1:N
                ftemp(loop) = sum(payoff(:,loop).*y);
            end
            f = time_expand*ftemp/sum(ftemp);
        end

        % Transition matrix
        for yloop = 1:N
            for xloop = 1:N
                W(yloop,xloop) = f(yloop)*Q(yloop,xloop);
            end
        end

        phi = sum(f'.*y);   % Average fitness of population

        yd = W*y - phi*y;

end     % end quasispec

end     % end quasiSpec
```

Quasi-species with Hamming distance.

quasiHam.m

```
% Quasispecies simulation (similar to quasiSpec.m).
% Use this to explore fitness landscapes

function quasiHam

%close all
h = colormap(lines);

randpop = 1;

B = 7;
N = 127;          % size of mutation space (32)
ep = 0.5;       % average mutation rate: 0.1 to 0.01 typical
lam = 1;

if randpop == 1
    x0temp = rand(1,N);    % Initial population
    sx = sum(x0temp);
    x0 = x0temp/sx;
else
    x0 = zeros(1,N);
    x0(1) = 0.667; x0(2) = 0.333;
end

Pop0 = sum(x0);

for yloop = 1:127
    for xloop = 1:127
        H(yloop,xloop) = hamming(yloop,xloop);
    end
end

keyboard


Qtemp = 1./(1+ep*H);    %Mutation matrix
Qsum = sum(Qtemp,2);

% Normalize along species
for yloop = 1:N
    for xloop = 1:N
        Q(yloop,xloop) = Qtemp(yloop,xloop)/Qsum(yloop);
    end
end

x = 1:N;
alpha = 64;
ftemp = exp(-lam*H(alpha,:));   % Fitness landscape
sf = sum(ftemp);
f = ftemp/sf;

% Transition matrix
for yloop = 1:N
    for xloop = 1:N
```

```
        W(yloop,xloop) = f(yloop)*Q(yloop,xloop);
    end
end

tspan = [0 500];


[t,x] = ode45(@quasispec,tspan,x0);

Pop0
[sz,dum] = size(t);
Popend = sum(x(sz,:))

phistar = sum(f.*x(sz,:))


figure(1)
plot(f,'o-')
hold on
figure(1)
plot(x(sz,:),'o-r')
hold off

figure(2)
for loop = 1:N
    semilogx(t,x(:,loop),'Color',h(round(loop*64/N),:))
    hold on
end
hold off

figure(3)
for loop = 1:N
    plot(t,x(:,loop),'Color',h(round(loop*64/N),:),'LineWidth',1.25)
    hold on
end
hold off
set(gcf,'Color','white')
xlabel('Time','FontSize',14)
ylabel('Population','FontSize',14)
hh = gca;
set(hh,'FontSize',14)

figure(4)
for loop = 1:N
    loglog(t,x(:,loop),'Color',h(round(loop*64/N),:))
    hold on
end
hold off

figure(5)
for loop = 1:N
    semilogy(t,x(:,loop),'Color',h(round(loop*64/N),:),'LineWidth',1.25)
    hold on
end
hold off
set(gcf,'Color','white')
```

```
xlabel('Time','FontSize',14)
ylabel('Population','FontSize',14)
hh = gca;
set(hh,'FontSize',14)

% Eigenvalues
 [V,D] = eig(W);
max(D(:,1))
figure(6)
plot(V(:,1))
disp(' ')

    function yd = quasispec(t,y)

        phi = sum(f'.*y);   % Average fitness of population

        yd = W*y - phi*y;

      end     % end quasispec

end     % end quasiHam
```

Hamming Distance.

hamming.m

```
%function y = hamming(x,y)
%Hamming binary distance between non-negative decimal integers x and y

function hsum = hamming(x,y)

sx = ceil(log2(x+1));
sy = ceil(log2(y+1));
sz = max(sx,sy);

xb = dec2bin(x);
yb = dec2bin(y);

z = bitxor(x,y);

sum = 0;
for loop = 1:sz
    sum = sum + bitget(z,loop);
end

hsum = sum;
```

Generate a stochastic matrix.

stochasticmatrix.m

```
% function y = stochasticmatrix(N)
% creates NxN stochastic matrix
% mean value of off-diag elements equals 1/N

function y = stochasticmatrix(N)
```

```
U = randunitary(N);
U2 = abs(U).^2;

y = 0.5*(U2 + U2');
```

Generate a random unitary matrix

randunitary.m

```
% function U = randunitary(N)
% Generates NxN random unitary matrix

function U = randunitary(N)

A = rand(N,N) + i*rand(N,N);

H = 0.5*(A + A');

U = expm(i*H);
```

Replicator mutator model.

repmut.m

```
%function repmut
% 12/08/14

function repmut

clear
format compact

N = 64;
p = 1/sqrt(N);
time_expand = N;

mutype = 0;      % 0 = Hamming   1 = rand
pay = 0;         % 0 = Hamming   1 = p
ep = 0.1;       % average mutation rate: 0.1 to 0.01 typical  (0.4835)

%%%%% Set original population
x0temp = rand(1,N);    % Initial population
sx = sum(x0temp);
y0 = x0temp/sx;
Pop0 = sum(y0);


%%%%% Set Adjacency
node = makeER(N,0.1);
%node = makeSF(N,4);
%node = makeSW(N,4,0.5);
[Adj,degree,Lap] = adjacency(node);
```

```
%%%%%% Set Hamming distance
for yloop = 1:N
    for xloop = 1:N
        H(yloop,xloop) = hamming(yloop-1,xloop-1);
    end
end

%%%%%%% Set Mutation matrix
if mutype == 0
    Qtemp = 1./(1+H/ep);    %Mutation matrix on Hamming
    %Qtemp = exp(-H/(ep*50));
    Qsum = sum(Qtemp,2);

    % Normalize mutation among species
    for yloop = 1:N
        for xloop = 1:N
            Q(yloop,xloop) = Qtemp(yloop,xloop)/Qsum(xloop);
        end
    end

elseif mutype == 1
    S = stochasticmatrix(N);
    Stemp = S - diag(diag(S));
    Qtemp = ep*Stemp;
    sm = sum(Qtemp,2)';
    Q = Qtemp + diag(ones(1,N) - sm);
end

figure(1)
imagesc(Q)
title('Mutation Matrix')

%%%%%%% Set payoff matrix
if pay == 1
    payoff = zeros(N,N);
    for yloop = 1:N
        payoff(yloop,yloop) = 1;
        for xloop = yloop + 1:N
            payoff(yloop,xloop) = p;
            payoff(xloop,yloop) = p;
            %payoff(yloop,xloop) = p*2*(0.5 - randbin(1,0.5));
            %payoff(xloop,yloop) = payoff(yloop,xloop);
            %payoff(xloop,yloop) = -payoff(yloop,xloop);
        end
    end
else
    payoff = exp(-1*H);
end


figure(2)
imagesc(payoff)
title('Payoff Matrix')

% Run time evolution
tspan = [0 1000];
[t,x] = ode45(@quasispec,tspan,y0);
```

```
Pop0
[sz,dum] = size(t);
Popend = sum(x(sz,:))

phistar = sum(f.*x(sz,:))       % final average fitness

 figure(3)
clf
h = colormap(lines);
for loop = 1:N
    plot(t,x(:,loop),'Color',h(round(loop*64/N),:))
    hold on
end
hold off

figure(4)
clf
for loop = 1:N
    semilogx(t,x(:,loop),'Color',h(round(loop*64/N),:))
    hold on
end
hold off

figure(5)
clf
for loop = 1:N
    semilogy(t,x(:,loop),'Color',h(round(loop*64/N),:))
    hold on
end
hold off

%keyboard

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    function yd = quasispec(~,y)

        for loop = 1:N
            ftemp(loop) = sum(payoff(:,loop).*y);
        end
        f = time_expand*ftemp/sum(ftemp);

% Transition matrix
        for yloop = 1:N
            for xloop = 1:N
                W(yloop,xloop) = f(yloop)*(Adj(yloop,xloop)*Q(yloop,xloop));
            end
        end

        phi = sum(f'.*y);   % Average fitness of population

        yd = W*y - phi*y;
```

```
    end     % end quasispec

end
```

Kauffmann's NK epistatic fitness model.
NK.m

```
%Companion to IMD

clear
format compact

drawnet = 1;

N = 7;
K = 2;   % must be 0 <= K < N
P = 0.05;  % Discreteness (set to 0.0001 to approximate continuous values)

numnode = 2^N;

node = makehypercube(N);

% Generate fitness value matrix
numepi = 2^(K+1);
for epiloop = 1:numepi
    for locloop = 1:N
        fitnessmatrix(epiloop,locloop) = rand;
    end
end

figure(1)
imagesc(fitnessmatrix)
xlabel('loci')
ylabel('permutations')
title('Weight Matrix')

%Generate epi circuits
for locloop = 1:N
        epi(locloop,:) = randintexc(K,N,locloop);
end

% Calculate fitness values
maxfit = 0;
for nodeloop = 1:numnode
    fitsum = 0;
    for locloop = 1:N
        v = node(nodeloop).val(locloop);

        for kloop = 1:K
```

```
            v = strcat(v,node(nodeloop).val(epi(locloop,kloop)));
        end

        index = bin2dec(v) + 1;

        fitsum = fitsum + fitnessmatrix(index,locloop);

    end
    temp = P*floor(fitsum/P/N);
    node(nodeloop).fitness = temp;
    if temp > maxfit
        maxfit = temp;
        maxnode = nodeloop;
    end
end

% Find number of local maxima in landscape
% Start a walker

start = randint(1,numnode);
loc = start;

displine('start = ',start)
displine('startfit = ',node(start).fitness)
disp(' ')

ind = 0; cnt = 0;
flag = 1;
while flag == 1
    cnt = cnt + 1;

    oldloc = loc;
    nlink = node(loc).numlink;
    val = node(loc).fitness;
    newloc = loc;
    neutralflag = 0; opt = 0; clear choice
    for loop = 1:nlink
        neighbor = node(loc).link(loop);
        if node(neighbor).fitness > val
            val = node(neighbor).fitness;
            newloc = neighbor;
        elseif node(neighbor).fitness == val;
            neutralflag = neutralflag + 1;
            choice(neutralflag) = neighbor;
        end
        if neutralflag > 0
            opt = length(choice);
            chnc = randintexc(1,opt);
            newloc = choice(chnc);
            val = node(newloc).fitness;
        end
    end

    if (newloc == oldloc)||(cnt > 2*N)||(opt == 1)
        flag = 0;
    else
```

```
        loc = newloc;
        ind = ind + 1;
        fitness(cnt) = node(newloc).fitness;
        locat(cnt) = newloc;
        displine('newloc = ',newloc)
        displine('newfit = ',node(newloc).fitness)
        locpath(ind).index = newloc;
        locpath(ind).fitness = node(newloc).fitness;

    end


end

finaldist = hamming(bin2dec(node(start).val),bin2dec(node(newloc).val));

disp(' ')
displine('numsteps = ',ind)
displine('finaldist = ',finaldist)
if ind ~=0
    displine('endratio = ',locpath(ind).fitness/maxfit)
    displine('endfit = ',locpath(ind).fitness)
end
displine('maxfit = ',maxfit)

if ind > 0
    figure(2)
    plot(fitness)
    title('Fitness')

    figure(3)
    plot(locat)
    title('Location')
end

if drawnet == 1

     for loop = 1:numnode
         node(loop).numval = node(loop).fitness;
     end

     DynamicDrawNet(node)

end
```

Chapter 8 Econophysics

Sh182.m
SDlogistic.m
mutrep.m
Sh398.m

Van der Pol-like economics model.

Sh182.m

```
function y = Sh182

a = 50;         % 50
b = 9;          % 9
c = 0.8;        % 0.8
d = 0.02;       % 0.02
q0 = -0.0;      % 0.87   2.61
q1 = 0.5;       % 0.5
alpha = 1;      % 1
beta = 2;       % 2

y0 = [25 45];      % [1 1]
tspan = [1 50];
[t,y] = ode45(@f5,tspan,y0);

figure(2)
plot(t,y(:,1),t,y(:,2))


figure(3)
plot(y(:,1),y(:,2))
title('x-z')
set(gcf, 'color', 'white')
hold on

xx = 0:0.1:25;
yy1 = (a - b*xx + c*xx.^2 - d*xx.^3);
plot(xx,yy1,'r')
yy2 = (xx - q0)/q1;
plot(xx,yy2,'k')

Printfile3('null.txt',xx,yy1,yy2)

e = y(:,1);
f = y(:,2);
Printfile3('182out.txt',t',e',f');

y0 = [20 1];      % [1 1]
tspan = [1 50];
[t,y] = ode45(@f5,tspan,y0);
[sz dum] = size(y);
plot(y(:,1),y(:,2))
```

```
y0 = [1 1];      % [1 1]
tspan = [1 50];
[t,y] = ode45(@f5,tspan,y0);
[sz dum] = size(y);
plot(y(:,1),y(:,2))
hold off

    function yd = f5(t,y)
        yp(1) = alpha*(a - b*y(1) + c*y(1).^2 - d*y(1).^3-y(2));
        yp(2) = beta*(y(1) - q0 - q1*y(2));
        yd = [yp(1);yp(2)];
     end     % end f5


end % end ltest
```

Supply-and-Demand logistic map.

SDlogistic.m

```
% logistic.m

clear

N = 2000;
A = zeros(N,N);

xold = 0.231;
gmax = 2000;         % 6000 for good coverage (pause off)
maxg = +1.5;
ming = -1.5;
maxy = 2;

lam = 0.305;     % 0.305
b = 0.25;        % 0.25
r = 4.5;         % 4.5

g = 0.0;
x = -2:0.01:2;
f = (1-lam)*x + lam*g/b - (lam/b)*atan(r*x);
xlin = -2:2;
ylin = xlin;
ylin2 = -xlin;
figure(1)
plot(x,f,xlin,ylin,xlin,ylin2)
axis([-2 2 -2 2])
line([0 0], [-2 2],'color','k')
line([-2 2], [0 0],'color','k')


for gainloop = 1:gmax

    g=(maxg-ming)*gainloop/gmax + ming;
```

```
    for sloop = 1:100            %settle-down loop
        xnew = (1-lam)*xold + lam*g/b - (lam/b)*atan(r*xold);
        xold = xnew;
    end                       % end settle-down loop

    rep = 10000;         % 10000
    ind = 0;
    for rloop = 1:rep
        xnew = (1-lam)*xold + lam*g/b - (lam/b)*atan(r*xold);
        xold = xnew;

        ind = ind + 1;
        x(ind) = g + (maxg-ming)*(rloop-1)/(gmax*rep);
        y(ind) = xnew;

    xx = floor(N*0.999*(g-ming)/(maxg-ming) - ming) + 1;
    yy = floor(N*(maxy-xnew)/(2*maxy)) + 1;

    %keyboard

    if xx > 2000
        keyboard
    end
    if yy > 2000
        keyboard
    end

    A(yy,xx) = A(yy,xx) + 1;

    end     % end rloop



end     % end gainloop


figure(2)
imagesc(A)
h = newcolormap('graycolor');
caxis([0 60])
colormap(h)
%colorbar
```

Mutator replicator model.

mutrep.m

```
% Mutator-Replicator simulation.

function mutrep

%close all
h = colormap(lines);
```

```
N = 32;          % size of mutation space (32)
ep = 0.03;       % average mutation rate: 0.1 to 0.01 typical

%x0 = rand(1,N);    % Initial population
x0 = zeros(1,N);
x0(1) = 1; x0(2) = 0.5;

Pop0 = sum(x0);

Qtemp = ep*rand(N) + eye(N);    %Mutation matrix
Qsum = sum(Qtemp,2);

% Normalize along species
for yloop = 1:N
    for xloop = 1:N
        Q(yloop,xloop) = Qtemp(yloop,xloop)/Qsum(yloop);
    end
end

ftemp = rand(1,N);
xx = 1:N;
g = Gauss((xx-N/2)/2);
fp = ftemp + 2*g;   % Fitness landscape


tspan = [0 64*(64/N)];

[t,x] = ode45(@quasispec,tspan,x0);

Pop0
[sz,dum] = size(t);
Popend = sum(x(sz,:))


figure(1)
plot(x(sz,:),'o-r')


figure(2)
for loop = 1:N
    loglog(t,x(:,loop),'Color',h(loop*64/N,:))
    hold on
end
hold off

    function yd = quasispec(t,y)

        f = fp/max(fp) - 0.49*y';  % Fitness includes feedback from
population

        % Transition matrix
        for yloop = 1:N
            for xloop = 1:N
                W(yloop,xloop) = f(yloop)*Q(yloop,xloop);
```

```
            end
        end

      phi = sum(f'.*y);   % Average fitness of population

        yd = W*y - phi*y;
      end     % end quasispec
end     % end quasi
```

Two competing companies.

Sh398.m

```
% 2 competing companies

function Shone398

sNorm = 0;

pmax = 9;
m1 = 4;  % manufacturing cost
m2 = 3;
k1 = 0.75;  % adjustment rate
k2 = 0.5;

xmax = pmax - m2;
ymax = pmax - m1;

% Isoprofit phase space
figure(1)
for piloop = 1:15
    if piloop ==1
        pi1 = 0; pi2 = 0;
    else
        pi1 = piloop - 0.225;
        pi2 = piloop + 0.44;
    end

    q1 = 0.1:0.1:xmax;
    q2 = pmax - pi1./q1 -m1 - q1;

    plot(q1,q2)
    axis([0 xmax 0 ymax])

    hold on

    clear q1 q2
    q2 = 0.1:0.1:ymax;
    q1 = pmax - pi2./q2 -m2 - q2;
    plot(q1,q2)

end
hold off

x1 = 0;
y1 = (pmax - m1) - 2*x1;
```

```
x2 = xmax;
y2 = (pmax - m1) - 2*x2;
X = [x1 x2];
Y = [y1 y2];
line(X,Y,'color','black')

x1 = 0;
y1 = 0.5*(pmax - m2) - 0.5*x1;
x2 = xmax;
y2 = 0.5*(pmax - m2) - 0.5*x2;
X = [x1 x2];
Y = [y1 y2];
line(X,Y,'color','black')

set(gcf,'Color','white')

clear q1 q2
for xloop = 1:100
    for yloop = 1:100-xloop+1

        q1 = (xloop-1)*xmax/100;
        q2 = (yloop-1)*ymax/100;

        p1 = (pmax -q1 -q2)*q1 - m1*q1;
        p2 = (pmax -q1 -q2)*q2 - m2*q2;

        XX(xloop) = q1;
        YY(yloop) = q2;
        P1(yloop,xloop) = p1;
        P2(yloop,xloop) = p2;
        pp(yloop,xloop) = sqrt(heaviside(p1*p2)*(p1*p2));
        ps(yloop,xloop) = p1 + p2;


    end
end

figure(2)
pcolor(XX,YY,P1)
shading interp
colormap(jet)
title('Profit 1')

figure(3)
pcolor(XX,YY,P2)
shading interp
colormap(jet)
title('Profit 2')

figure(4)
pcolor(XX,YY,pp)
shading interp
colormap(jet)
title('Geometric mean')
```

```
figure(5)
pcolor(XX,YY,ps)
shading interp
colormap(jet)
title('Profit sum')

% Dynamics

[X,Y] = meshgrid(0:0.1:10, 0:0.1:10);

[x,y] = f5(X,Y);

figure(6)
clf
N = 10;
for xloop = 1:N
    xs = 0.01 + xmax*(xloop-1)/N;
    for yloop = 1:N
        ys = 0.01 + ymax*(yloop-1)/N;

        streamline(X,Y,x,y,xs,ys)

    end
end

M = 10;
hold on
[XQ,YQ] = meshgrid(0:xmax/M:xmax,0:ymax/M:ymax);
[xq,yq] = f5(XQ,YQ);
quiver(XQ,YQ,xq,yq,2,'r')
hold off
set(gcf,'color','white')

axis([0 xmax 0 ymax])

x1 = 0;
y1 = (pmax - m1) - 2*x1;
x2 = xmax;
y2 = (pmax - m1) - 2*x2;
X = [x1 x2];
Y = [y1 y2];
line(X,Y,'color','black')

x1 = 0;
y1 = 0.5*(pmax - m2) - 0.5*x1;
x2 = xmax;
y2 = 0.5*(pmax - m2) - 0.5*x2;
X = [x1 x2];
Y = [y1 y2];
line(X,Y,'color','black')

keyboard

    function [x,y] = f5(X,Y)

        f = 0.5*(pmax-m1)*k1-k1*X-0.5*k1*Y;
```

```
        g = 0.5*(pmax-m2)*k2-k2*Y-0.5*k2*X;

        s = sqrt(f.^2 + g.^2);

        if sNorm == 1
            x = f./s;
            y = g./s;
        else
            x = f;
            y = g;
        end

    end     % end f5

end % end Sh398
```

Chapter 9 Metric Spaces

Launch a light ray in warped space-time.

raysimple.m

```
function raysimple

clear

% Refractive index Landscape

delx = 40/100; dely = 40/100;
for yloop = 1:100;
    for xloop = 1:100;
        x(xloop) = -20 + xloop*delx;
        y(yloop) = -20 + yloop*dely;
        ref(yloop,xloop) = refindex(x(xloop),y(yloop));
    end
end

figure(1)
imagesc(x,y,ref)
colormap(gray)
axis square
colorbar
title('Refractive index')
set(gcf,'Color','white')

v1 = 0.707;
v2 = sqrt(1-v1^2);
y0 = [12 v1 0 v2];
tspan = [0 1470];
%tspan = [0 1550];

[t,y] = ode45(@f5,tspan,y0);

figure(2)
plot(t,y(:,1),t,y(:,2))

figure(3)
plot(y(:,1),y(:,2),'LineWidth',1.2)
axis([-20 20 -20 20])
axis square
title('Light Orbits')
set(gcf,'Color','white')

    function yd = f5(t,y)

        [n nx ny] = refindex(y(1),y(2));

        yp(1) = y(3)./n;
        yp(2) = y(4)/n;
        yp(3) = nx;
        yp(4) = ny;
```

```
        yd = [yp(1);yp(2);yp(3);yp(4)];

    end     % end f5

    function [n nx ny] = refindex(x,y)

        selection = 2;

        if selection == 1

            sig = 10;

            n = 1+ exp(-(x.^2 + y.^2)/2/sig^2);
            nx = (-2*x/2/sig^2).*exp(-(x.^2 + y.^2)/2/sig^2);
            ny = (-2*y/2/sig^2).*exp(-(x.^2 + y.^2)/2/sig^2);

        elseif selection == 2

            sig = 10;
            r2 = (x.^2 + y.^2);
            r1 = sqrt(r2);
            expon = exp(-r2/2/sig^2);

            n = 1+0.3*r1.*expon;
            nx = 0.3*r1*(-2*x/2/sig^2)*expon + 0.3*expon*2*x/r1;
            ny = 0.3*r1*(-2*y/2/sig^2)*expon + 0.3*expon*2*y/r1;

        end


    end

end
```