Code for BZ Reaction Project

David Connolly and Brad Nelson

\* headers for different codes are in RED

\* note that many of the codes do not define the function or its

parameters – this must be done beforehand

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

OSCILLATIONS AND 3D ATTRACTOR FOR OREGONATOR

% Clear previous variables

clear all;

% Define parameters

k1=1.28;

k2=2.4e6;

k3=33.6;

k4=3e3;

kc=1;

b=0.02;

a=0.06;

% Initial conditions

xxo=0.0005;

yyo=0.02;

zzo=0.02;

% Make dimensionless

eps=kc\*b/k3/a;

epsp=2\*kc\*k4\*b/k2/k3/a;

q=2\*k1\*k4\*b/k2/k3/a;

fconst=1;

% Define Oregonator

f = @(t,x) [(q.\*x(2)-x(1).\*x(2)+x(1).\*(1-x(1)))./eps; (-q.\*x(2)-x(1).\*x(2)...

+fconst.\*x(3))./epsp;x(1)-x(3)];

% Make dimensionless

xo=[2\*k4\*xxo/k3/a,k4\*yyo/k3/a,kc\*k4\*b\*zzo/k3^2/a^2];

% Solve ODE system

[ts,xs] = ode23s(f, [0 100], xo);

% Put back into normal units

x = [k3\*a\*xs(:,1)/2/k4,k3\*a\*xs(:,2)/k4,k3^2\*a^2\*xs(:,3)/kc/k4/b];

% Plot oscillations

figure(1); subplot(3,1,1); plot(ts,x(:,1)); xlabel('Time'); ylabel('HBrO\_2');

subplot(3,1,2); plot(ts,x(:,2)); xlabel('Time'); ylabel('Br^-');

subplot(3,1,3); plot(ts,x(:,3)); xlabel('Time'); ylabel('Ce^4^+');

% Plot 3-d attractor

figure(2); scatter3(x(2000:end,1),x(2000:end,2),x(2000:end,3),'.');

xlabel('log(HBrO\_2)'); ylabel('log(Br^-)'); zlabel('log(Ce^4^+)');

set(gca,'xscale','log','yscale','log','zscale','log');

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

TIME DELAY RECONSTRUCTION FOR OREGONATOR

% Solve Oregonator at 100,000 points between t = 0 and 400

t=linspace(0,400,100000);

x=deval(ode23s(f, [0 400], xo, odeset('abstol',1e-6)),t);

t=t';

xlim=x(1,1000:end);

% Plot

figure(7);

hold on;

for i=1:(length(xlim)-1);

plot(xlim(i),xlim(i+1),'-');

end

hold off;

title('Time-Delay Reconstruction of the Oregonator'); xlabel('x(t)'); ylabel('x(t+1)');

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

CORRELATION DIMENSION FOR OREGONATOR

%Solve Oregonator at 100,000 points

t=linspace(0,400,100000);

x=deval(ode23s(f, [0 400], xo, odeset('abstol',1e-6)),t);

x = x(:,20000:end);

N = length(x(1,:));

%Radii

rs=[1,.9,.8,.7,.6,.5];

%blank list of cs

cs=zeros(1,length(rs));

%find number of pairs for c(r)

for i=1:length(rs);

pairs=0;

for n=1:N;

pairs=pairs+numel(find(sum((kron(x(:,n), ones(1,N)) - x).^2, 1) < rs(i)^2));

end

cs(i)=pairs/((N+1)\*N/2); %c(r)

end

%Figure plotting C(r) vs. r on log-log plot

figure(5); loglog(rs,cs,'.-');

title('C(r) for Oregonator Attractor');

xlabel('r');

ylabel('C(r)');

%now to find slope...

logrs=log(rs);

logcs=log(cs);

line=polyfit(logrs,logcs,1);

cordim=line(1);

display(cordim);

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

LYAPUNOV EXPONENTS FOR OREGONATOR

% test lorenz time-1 map

[x, J] = lorenz\_time1map(yo)

grid on;

% Measure Lyapunov exponents...

% ----- Re-orthogonalizing version, repeated averaging

M = 6; % how many averaging loops

N = 12; % how many its per meas step

x = [2\*k4\*xxo/k3/a;k4\*yyo/k3/a;kc\*k4\*b\*zzo/k3^2/a^2];

h = zeros(3,1); % place to store averaged lyap exps

for m=1:M

J = eye(3); % Id is where Jacobean starts

for n=1:N

[x Jx] = lorenz\_time1map(x);

J = Jx\*J; % update Jacobean

[Q,R] = qr(J); % re-orthogonalize

J = Q\*diag(diag(R)); % but keep them correct lengths

end

rN = abs(diag(R)) % print out progress

h = h + log(rN)/N; % sum up lyap exps from each run

end

h = h/M % final answer: mean (over runs) of lyap exps

% you can see the middle lyap exp is v close to 0 - in fact all flows have

% a zero exponent if they are bounded and hit no equilibrium points.

NEED TIME1MAP FUNCTION...

function [x, DFx] = lorenz\_time1map(xo)

% evolve Lorenz flow for 1 time unit, including finding the jacobean matrix

% parameters

k1=1.28;

k2=2.4e6;

k3=33.6;

k4=3e3;

kc=1;

b=0.02;

a=0.06;

% make dimensionless

eps=kc\*b/k3/a;

epsp=2\*kc\*k4\*b/k2/k3/a;

q=2\*k1\*k4\*b/k2/k3/a;

fconst=1;

% function

f = @(t,x) [(q.\*x(2)-x(1).\*x(2)+x(1).\*(1-x(1)))./eps; (-q.\*x(2)-x(1).\*x(2)+fconst.\*x(3))./epsp;x(1)-x(3)];

% jacobian

Df = @(x) [(-x(2)+1-2\*x(1))/eps, (q-x(1))/eps, 0; -x(2)/epsp, (-q-x(1))/epsp, fconst/epsp; 1, 0, -1]; % Jacobean matrix as func of x

J0 = eye(3); % initial Jac matrix

% 12-component ODE flow given by 3 components of solution and 9 components

% of the J matrix (J satisfies the ODE dJ/dt = Df.J)

G = @(t,z) [f(t,z(1:3,:)); Df(z(1:3,:))\*z(4:6,:); Df(z(1:3,:))\*z(7:9,:); ...

Df(z(1:3,:))\*z(10:12,:) ];

[ts, xs] = ode23s(G, [0 1], [xo; J0(:)]); % numerically solve in t domain

x = reshape(xs(end,1:3), [3,1]); % extract the answer at the final time t=1

J = xs(end,4:end); % same for the J components

DFx = reshape(J,[3 3]); % send J out as a 3x3 matrix.

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

OSCILLATIONS AND 3D ATTRACTOR FOR GF

%BZ model found in Gyorgi and Field paper

%parameters from fig a in paper

clear;

%starting point

xxo=0.0468627;

zzo=0.89870;

vvo=0.846515;

%relatively constant concentrations (Molar)

A=0.1;

M=0.25;

H=0.26;

C=0.000833;

%adjustable parameters

alpha=666.7;

beta=0.3478;

%rate constants (SI units)

k1=4e6;

k2=2.0;

k3=3000;

k4=55.2;

k5=7000;

k6=0.09;

k7=0.23;

%variable rate constant (depends on reactor)

%inverse residence time of reactor

kf=3.9e-4;

% kf = 10e-4;

%dimension scalars

To=1/(10\*k2\*A\*H\*C);

Xo=k2\*A\*H\*H/k5;

Yo=4\*k2\*A\*H\*H/k5;

Zo=C\*A/(40\*M);

Vo=4\*A\*H\*C/(M^2);

%constants used in function

c1=-k1\*H\*Yo;

c2=k2\*A\*H^2\*Yo/Xo;

c3=-2\*k3\*Xo;

c4=0.5\*k4\*A^(.5)\*H^(1.5)\*Xo^(-.5);

c5=-0.5\*k5\*Zo;

c6=k4\*A^(0.5)\*H^(1.5)\*Xo^0.5;

c7=-k5\*Xo;

c8=-alpha\*k6\*Vo;

c9=-beta\*k7\*M;

c10=2\*k1\*H\*Xo\*Yo/Vo;

c11=k2\*A\*H^2\*Yo/Vo;

c12=k3\*Xo^2/Vo;

c13=-alpha\*k6\*Zo;

%ybar constants

yb1=alpha\*k6\*Zo\*Vo;

yb2=k1\*H\*Xo;

yb3=k2\*A\*H^2;

ybar = @(x) ((yb1.\*x(2).\*x(3)./(yb2.\*x(1)+yb3+kf))/Yo);

%function to evolve dimensionless system of equation

f = @(t,x) [To\*(c1.\*x(1).\*ybar(x)+c2.\*ybar(x)+c3\*x(1).^2+c4\*(C-Zo\*x(2)).\*(x(1).^(0.5))+c5\*x(1).\*x(2)-kf\*x(1));...

   To\*(c6\*(C/Zo-x(2)).\*(x(1).^(0.5))+c7\*x(1).\*x(2)+c8\*x(2).\*x(3)+c9\*x(2)-kf\*x(2));...

   To\*(c10.\*(x(1).\*ybar(x))+c11.\*ybar(x)+c12.\*(x(1).^2)+c13\*x(2).\*x(3)-kf\*x(3))];

%setting up ODE solver

xo = [xxo,zzo,vvo];

[ts,xs]  = ode23s(f, [0 6], xo, odeset('abstol',1e-6));

%plotting data

figure(1); subplot(3,1,1); plot(ts,xs(:,1)); xlabel('t'); ylabel('x');

subplot(3,1,2); plot(ts,xs(:,2)); xlabel('t'); ylabel('z');

subplot(3,1,3); plot(ts,xs(:,3)); xlabel('t'); ylabel('v');

figure(2); scatter3(xs(1000:end,1),xs(1000:end,2),xs(1000:end,3),'-k');

set(gca,'xscale','log','yscale','log','zscale','log');

xlabel('log x');

ylabel('log z');

zlabel('log v');

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

BIFURCATION DIAGRAM FOR GF (WITH CONSTANTS SET UP ALREADY)

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

%creation of bifurcation diagram

figure(5);

hold on;

xt=0.0468627; %for poincare plane passing through this point on x axis

for n=3.0e-4:0.5e-6:4.5e-4;

    kf=n;

    ybar = @(x) ((yb1.\*x(2).\*x(3)./(yb2.\*x(1)+yb3+kf))/Yo);

    f = @(t,x) [To\*(c1.\*x(1).\*ybar(x)+c2.\*ybar(x)+c3\*x(1).^2+c4\*(C-Zo\*x(2)).\*(x(1).^(0.5))+c5\*x(1).\*x(2)-kf\*x(1));...

   To\*(c6\*(C/Zo-x(2)).\*(x(1).^(0.5))+c7\*x(1).\*x(2)+c8\*x(2).\*x(3)+c9\*x(2)-kf\*x(2));...

   To\*(c10.\*(x(1).\*ybar(x))+c11.\*ybar(x)+c12.\*(x(1).^2)+c13\*x(2).\*x(3)-kf\*x(3))];

   ts=linspace(0,1,16000);

   xs  = deval(ode23s(f, [0 1], xo,odeset('abstol',1e-6)),ts);

   xs=xs';

   for i=3:length(xs(:,1));

       if(xs(i,1)<=xt && xs(i-1,1)>=xt);

           slope=(xs(i,2)-xs(i-1,2))/(xs(i,1)-xs(i-1,1));

           deltaz=slope\*(xt-xs(i,1));

           plot(kf,xs(i,2)+deltaz,'-');

       end

   end

end

xlabel('k\_f');

ylabel('z (on Poincare plane)');

title('Bifurcation Diagram for k\_f');

hold off;

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

TIME DELAY RECONSTRUCTION FOR GF

%investigation of limit cycle behavior

t=linspace(0,100,100000);

x=deval(ode23s(f, [0 100], xo, odeset('abstol',1e-6)),t);

t=t';

xlim=x(1,1000:end);

ys=zeros(2,length(xlim)-1);

hold on;

for i=1:(length(xlim)-1);

ys(:,i)=[xlim(i),xlim(i+1)];

end

figure(6);

plot(ys(1,:),ys(2,:),'k-');

hold off;

xlabel('x(t)');

ylabel('x(t+1)');

title('time-delay reconstuction for t=\tau /1000');

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

CORRELATION DIMENSION FOR GF 3D ATTRACTOR

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

% calculation of the correlation dimension

% first, to calculate an obscene number of data points

xo = [xxo,zzo,vvo];

[ts,xs] = ode23s(f, [0 12], xo, odeset('abstol',1e-6));

N=length(xs(1000:end,1));

x=xs(1000:end,:)';

%list of rs to calculate c(r) for

rs=[1,.5,.1,.05,.01,5e-3,1e-3,5e-4,1e-4,5e-5,1e-5];

%blank list of cs

cs=zeros(1,length(rs));

for i=1:length(rs);

pairs=0;

for n=1:N;

pairs=pairs+numel(find(sum((kron(x(:,n), ones(1,N)) - x).^2, 1) < rs(i)^2));

end

cs(i)=pairs/((N+1)\*N/2); %c(r)

end

%Figure plotting C(r) vs. r on log-log plot

figure(5); loglog(rs,cs,'.-');

title('C(r) for GF Attractor');

xlabel('r');

ylabel('C(r)');

%now to find slope...

logrs=log(rs);

logcs=log(cs);

%finds best fit linear line that passes through good points

line=polyfit(logrs(3:8),logcs(3:8),1);

cordim=line(1);

display(cordim);

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

CORRELATION DIMENSION FOR TIME DELAY ATTRACTOR FOR GF

(AFTER SOLVING THE TIME DELAY...)

%parameters

N=length(ys(1,:));

x=ys;

%list of rs to calculate c(r) for

rs=[.1,.05,.01,5e-3,1e-3,5e-4,1e-4];

%blank list of cs

cs=zeros(1,length(rs));

for i=1:length(rs);

pairs=0;

for n=1:N;

pairs=pairs+numel(find(sum((kron(x(:,n), ones(1,N)) - x).^2, 1) < rs(i)^2));

end

cs(i)=pairs/((N+1)\*N/2); %c(r)

end

%Figure plotting C(r) vs. r on log-log plot

figure(3); loglog(rs,cs,'.-');

title('C(r) for GF Attractor');

xlabel('r');

ylabel('C(r)');

%now to find slope...

logrs=log(rs);

logcs=log(cs);

%finds best fit linear line that passes through good points

line=polyfit(logrs(1:6),logcs(1:6),1);

cordim=line(1);

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

LYAPUNOV EXPONENTS FOR GF

% test lorenz time-1 map

[x, J] = lorenz\_time1map2(yo)

% grid on;

% Measure Lyapunov exponents...

% ----- Re-orthogonalizing version, repeated averaging

M = 25; % how many averaging loops

N = 500; % how many its per meas step

x = [xxo;zzo;vvo];

h = zeros(3,1); % place to store averaged lyap exps

for m=1:M

J = eye(3); % Id is where Jacobean starts

for n=1:N

[x Jx] = lorenz\_time1map2(x);

J = Jx\*J; % update Jacobean

[Q,R] = qr(J); % re-orthogonalize

J = Q\*diag(diag(R)); % but keep them correct lengths

end

rN = abs(diag(R)) % print out progress

h = h + log(rN)/N; % sum up lyap exps from each run

end

h = h/M % final answer: mean (over runs) of lyap exps

% you can see the middle lyap exp is v close to 0 - in fact all flows have

% a zero exponent if they are bounded and hit no equilibrium points.

NEED TIME1MAP FUNCTION (ACTUALLY A TIME-0.001 MAP)...

function [x, DFx] = lorenz\_time1map2(xo)

% evolve Lorenz flow for 1 time unit, including finding the jacobean matrix

% relatively constant concentrations (Molar)

A=0.1;

M=0.25;

H=0.26;

C=0.000833;

% adjustable parameters

alpha=666.7;

beta=0.3478;

% rate constants (SI units)

k1=4e6;

k2=2.0;

k3=3000;

k4=55.2;

k5=7000;

k6=0.09;

k7=0.23;

% variable rate constant (depends on reactor)

% inverse residence time of reactor

kf=3.9e-4;

% kf = 10e-4;

% dimension scalars

To=1/(10\*k2\*A\*H\*C);

Xo=k2\*A\*H\*H/k5;

Yo=4\*k2\*A\*H\*H/k5;

Zo=C\*A/(40\*M);

Vo=4\*A\*H\*C/(M^2);

% constants used in function

c1=-k1\*H\*Yo;

c2=k2\*A\*H^2\*Yo/Xo;

c3=-2\*k3\*Xo;

c4=0.5\*k4\*A^(.5)\*H^(1.5)\*Xo^(-.5);

c5=-0.5\*k5\*Zo;

c6=k4\*A^(0.5)\*H^(1.5)\*Xo^0.5;

c7=-k5\*Xo;

c8=-alpha\*k6\*Vo;

c9=-beta\*k7\*M;

c10=2\*k1\*H\*Xo\*Yo/Vo;

c11=k2\*A\*H^2\*Yo/Vo;

c12=k3\*Xo^2/Vo;

c13=-alpha\*k6\*Zo;

% ybar constants

yb1=alpha\*k6\*Zo\*Vo;

yb2=k1\*H\*Xo;

yb3=k2\*A\*H^2;

% ybar and its derivatives

ybar = @(x) ((yb1.\*x(2).\*x(3)./(yb2.\*x(1)+yb3+kf))/Yo);

ybardx = @(x) ((-yb1.\*x(2).\*x(3)./(yb2.\*x(1)+yb3+kf)^2)/Yo);

ybardz = @(x) ((yb1.\*x(3)./(yb2.\*x(1)+yb3+kf))/Yo);

ybardv = @(x) ((yb1.\*x(2)./(yb2.\*x(1)+yb3+kf))/Yo);

dxybar = @(x) (Yo\*yb2)/(yb1.\*x(2).\*x(3));

dzybar = @(x) (-Yo\*(yb2.\*x(1)+yb3+kf))/(yb1.\*x(2).^2.\*x(3));

dvybar = @(x) (-Yo\*(yb2.\*x(1)+yb3+kf))/(yb1.\*x(2).\*x(3).^2);

% function to evolve dimensionless system of equation

f = @(t,x) [To\*(c1.\*x(1).\*ybar(x)+c2.\*ybar(x)+c3\*x(1).^2+c4\*(C-Zo\*x(2)).\*(x(1).^(0.5))+c5\*x(1).\*x(2)-kf\*x(1));...

To\*(c6\*(C/Zo-x(2)).\*(x(1).^(0.5))+c7\*x(1).\*x(2)+c8\*x(2).\*x(3)+c9\*x(2)-kf\*x(2));...

To\*(c10.\*(x(1).\*ybar(x))+c11.\*ybar(x)+c12.\*(x(1).^2)+c13\*x(2).\*x(3)-kf\*x(3))];

% jacobian...(yikes!)

Df = @(x) [To\*(c1\*ybar(x)+c1\*x(1).\*ybardx(x)+c2\*dxybar(x)+2\*c3\*x(1)+.5\*c4\*(C-Zo\*x(2)).\*x(1).^(-.5)+c5\*x(2)-kf),...

To\*(c1\*x(1).\*ybardz(x)+c2\*dzybar(x)-c4\*Zo\*x(1).^(.5)+c5\*x(1)),...

To\*(c1\*x(1).\*ybardv(x)+c2\*dvybar(x));...

To\*(.5\*c6\*((C/Zo)-x(2)).\*x(1).^(-.5)+c7\*x(2)),...

To\*(-c6\*x(1).^(.5)+c7\*x(1)+c8\*x(3)+c9-kf),...

To\*(c8\*x(2));...

To\*(-c10/(x(1).^2.\*ybar(x))+c10\*dxybar(x)./x(1)+c11\*dxybar(x)-2\*c12/x(1)^3),...

To\*(c10\*dzybar(x)./x(1)+c11\*dzybar(x)+c13\*x(3)),...

To\*(c10\*dvybar(x)./x(1)+c11\*dvybar(x)+c13\*x(2)-kf)];

J0 = eye(3); % initial Jac matrix

% 12-component ODE flow given by 3 components of solution and 9 components

% of the J matrix (J satisfies the ODE dJ/dt = Df.J)

G = @(t,z) [f(t,z(1:3,:)); Df(z(1:3,:))\*z(4:6,:); Df(z(1:3,:))\*z(7:9,:); ...

Df(z(1:3,:))\*z(10:12,:) ];

[ts, xs] = ode23s(G, [0 .001], [xo; J0(:)]); % numerically solve in t domain

x = reshape(xs(end,1:3), [3,1]); % extract the answer at the final time t=1

J = xs(end,4:end); % same for the J components

DFx = reshape(J,[3 3]); % send J out as a 3x3 matrix.