

PHYSICS 410

**THE MATLAB `ode45` INTEGRATOR
AND
APPLICATIONS**

- As with Runge-Kutta-Fehlberg, uses two RK methods, one $O(h^4)$, one $O(h^5)$; specifically, the Dormand-Prince pair, which minimizes the $O(h^5)$ solution error, whereas the Fehlberg technique minimizes the $O(h^4)$ error
- Recommended “first choice” integrator: if solution with this proves problematic, MATLAB has many more algorithms available
- Calling sequences (partial list):

```
[tout, yout] = ode45(odefun, tspan, y0)
```

```
[tout, yout] = ode45(odefun, tspan, y0, options)
```

`[TOUT,YOUT] = ODE45(ODEFUN,TSPAN,Y0)` with `TSPAN = [T0 TFINAL]` integrates the system of differential equations $y' = f(t,y)$ from time `T0` to `TFINAL` with initial conditions `Y0`. `ODEFUN` is a function handle. For a scalar `T` and a vector `Y`, `ODEFUN(T,Y)` must return a column vector corresponding to $f(t,y)$. Each row in the solution array `YOUT` corresponds to a time returned in the column vector `TOUT`. To obtain solutions at specific times `T0,T1,...,TFINAL` (all increasing or all decreasing), use `TSPAN = [T0 T1 ... TFINAL]`.

[TOUT,YOUT] = ODE45(ODEFUN,TSPAN,Y0,OPTIONS) solves as above with default integration properties replaced by values in OPTIONS, an argument created with the ODESET function. See ODESET for details. Commonly used options are scalar relative error tolerance 'RelTol' (1e-3 by default) and vector of absolute error tolerances 'AbsTol' (all components 1e-6 by default). If certain components of the solution must be non-negative, use ODESET to set the 'NonNegative' property to the indices of these components.

- Options example (for ODE system with two components):

```
>> options = odeset('AbsTol', [1.0e-5, 1.0e-6], ...  
                   'RelTol', [1.0e-5])
```

Example: Simple Harmonic Motion

- Governing differential equation (unit angular frequency)

$$\frac{d^2y(t)}{dt^2} = -y$$

- Cast in canonical first-order form by defining dependent variables

$$y_1(t) \equiv y(t) \qquad y_2(t) \equiv \frac{dy_1}{dt}$$

- Then equation of motion becomes the system

$$\frac{dy_1}{dt} = y_2 \qquad \frac{dy_2}{dt} = -y_1$$

subject to initial conditions

$$y_1(0) = y(0) \qquad y_2(0) = \frac{dy}{dt}(0)$$

Function fcn_sho

```
function dydt = fcn_sho(t, y)
% fcn_sho: Derivatives function for simple harmonic motion with unit
% angular frequency. Note that the function must return a COLUMN
% vector.
    dydt = zeros(2,1);
    dydt(1) = y(2);
    dydt(2) = -y(1);
end
```

Script ode45_sho.m

```
% ode45_sho: Integrates equations of motion for simple harmonic oscillator
% using ODE45

% Integrate on the domain 0 <= t <= 3 pi, with initial conditions
%
% y_1(0) = x(0) = 0
% y_2(1) = v(0) = 1
%
% corresponding to the exact solution
%
% y(t) = sin(t)

% Integrate with default parameters ...
[tout yout] = ode45(@fcn_sho, [0.0 3.0*pi], [0.0 1.0]);

figure(1);
plot(tout, yout(:,1));
xlabel('t');
ylabel('y');
print('sho_y.jpg', '-djpeg');
```

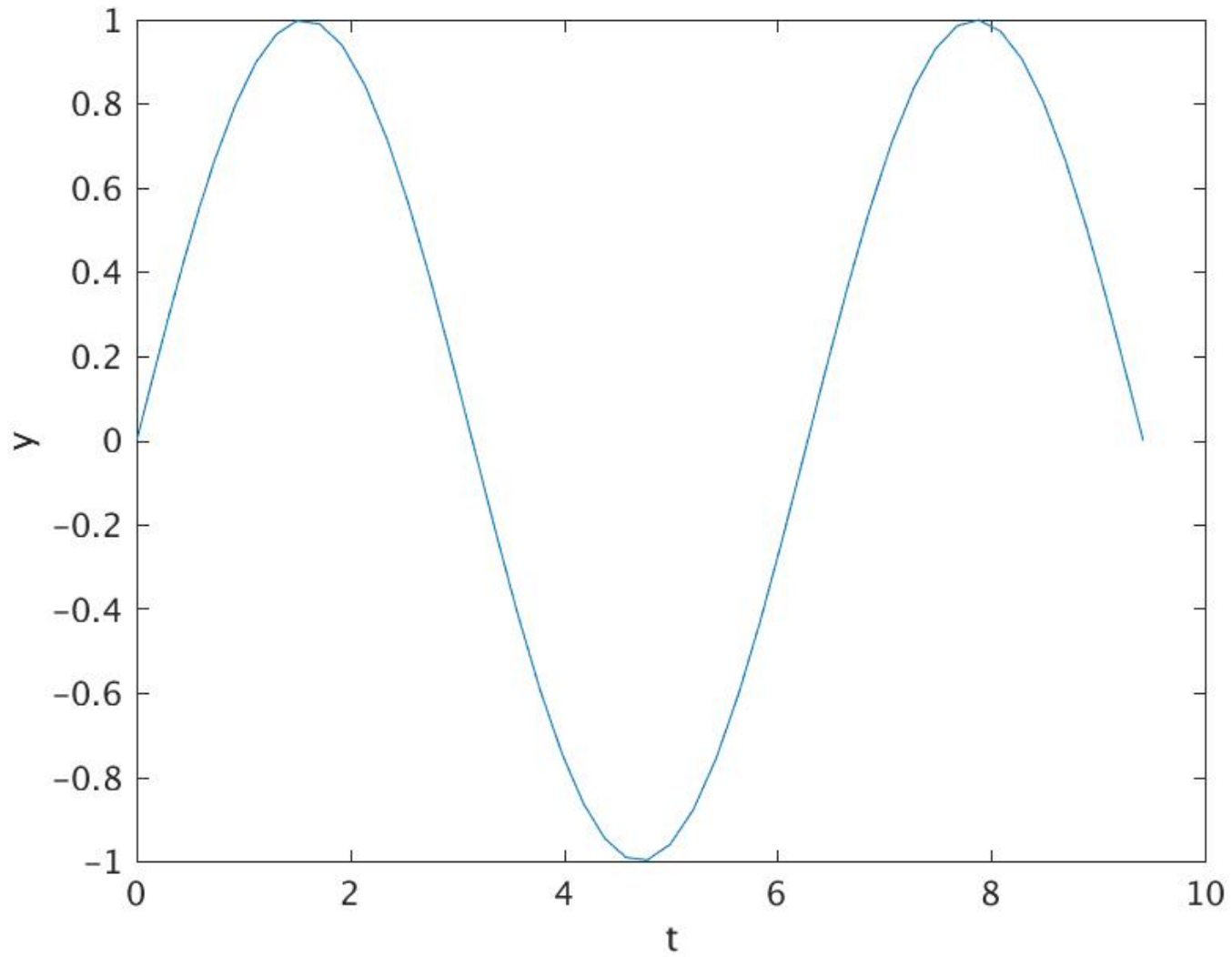
```

figure(2);
yxct = sin(tout);
plot(tout, yout(:,1) - yxct);
xlabel('t');
ylabel('y - y_{\rm exact}');
print('sho_error.jpg', '-djpeg');

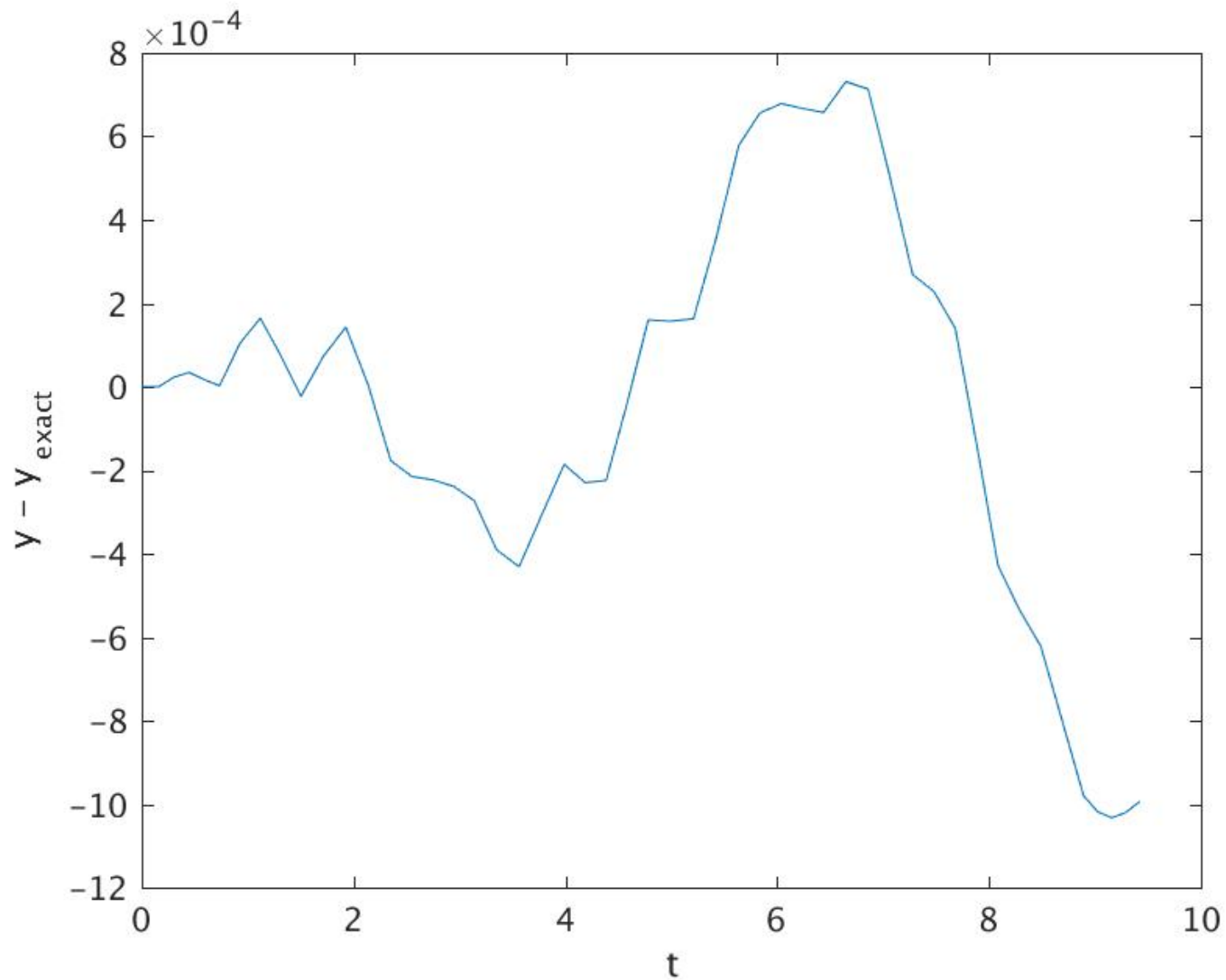
figure(3);
iplot = 1;
% Integrate with a variety of error tolerances ...
for tol = [1.0e-3, 1.0e-5, 1.0e-7, 1.0e-9]
    options = odeset('AbsTol', [tol, tol], 'RelTol', tol);
    [tout yout] = ode45(@fcn_sho, [0.0 3.0*pi], [0.0 1.0], options);
    yxct = sin(tout);
    subplot(2,2,iplot);
    plot(tout, yout(:,1) - yxct);
    xlabel('t');
    ylabel('Error');
    label = sprintf('Tolerance = %g', tol);
    legend(label,'Location','South');
    iplot = iplot + 1;
end
print('sho_var_tolerance.jpg', '-djpeg');

```

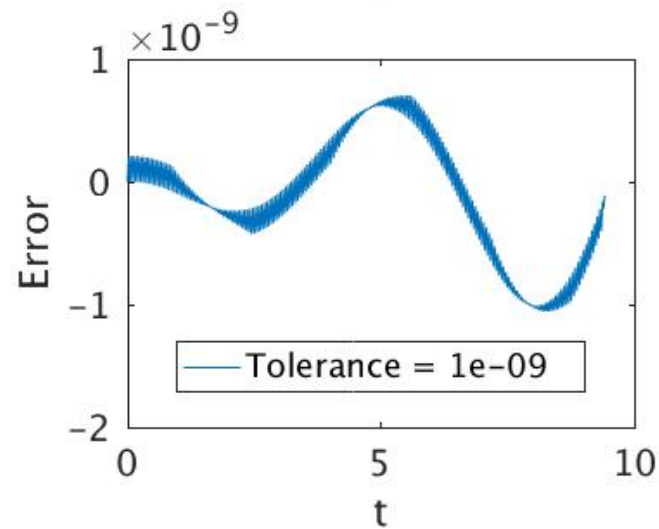
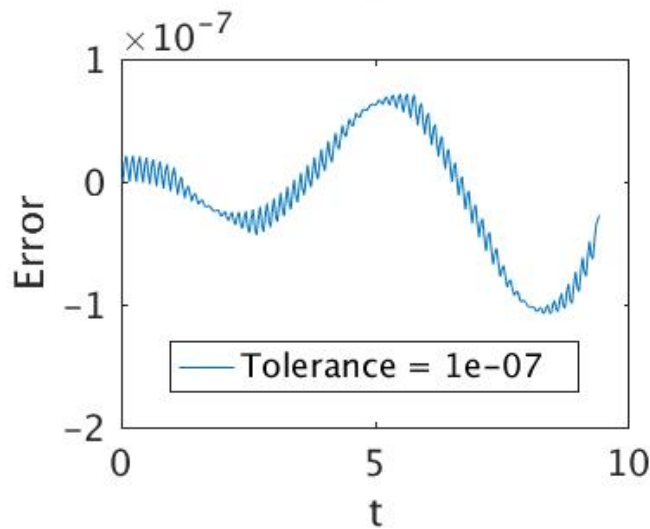
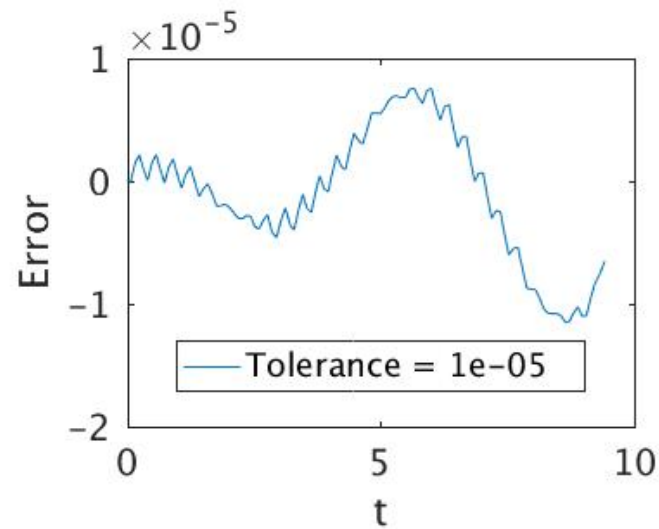
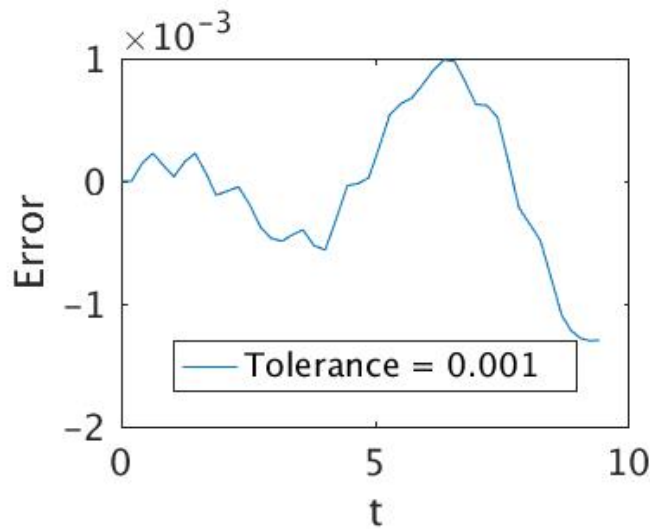
- Plot of approximate solution versus time



- Plot of error versus time (using default ODE45 parameters: AbsTol = 1.0e-6, RelTol = 1.0e-3)



- Plot of error versus time using various tolerances: AbsTol = RelTol = Tolerance)



Independent residual evaluation

- First, rewrite ODE in form

$$\frac{d^2y(t)}{dt^2} + y(t) = 0 \quad (1)$$

- Next, using ODE45, generate solution $\hat{u}(t^h, \epsilon)$ on a level- ℓ uniform mesh:

$$t_n^h = 0, h, 2h, \dots, t_{\max}$$

with (assuming $t_{\min} = 0$)

$$h = \frac{t_{\max}}{2^\ell}$$

- Then, apply $O(h^2)$ finite-difference discretization of (1) to \hat{y} to compute *residual* R_n :

$$R_n \equiv \frac{\hat{y}_{n+1} - 2\hat{y}_n + \hat{y}_{n-1}}{h^2} + \hat{y}_n \quad n = 1, 2, \dots, 2^\ell - 1$$

- Should find that RMS value (ℓ_2 norm) of R_n is an $O(h^2)$ quantity:

$$\left[\frac{\sum_n |R_n|^2}{2^\ell - 1} \right]^{\frac{1}{2}} \equiv \|\mathbf{R}\|_2 = O(h^2)$$

- In particular, consider computing R_n at three separate levels of 2:1-related discretization, $h_1 = h_\ell, h_2 = 2h_1 = h_{\ell-1}, h_3 = 4h_1 = h_{\ell-2}$. From a single level ℓ computation in MATLAB, these can be generated by taking every element, every second element and every fourth element of the output vector from the ODE integrator.
- Then, by plotting

$$16R^\ell, 4R^{\ell-1}, R^{\ell-2}$$

on a single graph, convergence of the independent residual evaluator will appear as near-coincidence of the 3 curves.

Independent residual evaluation: script file ode45_sho_ir.m

```
% ode45_sho_ir: Integrates equations of motion for simple harmonic
% oscillator using ODE45 and evaluates independent residuals using
% a finite difference approximation

% Integrate on the domain 0 <= t <= 3 pi, with initial conditions
%
% y_1(0) = x(0) = 0
% y_2(1) = v(0) = 1
%
% corresponding to the exact solution
%
% y(t) = sin(t)

% Integrate with default parameters, except use an uniform mesh of
% output times of length 2^10 + 1 and a stringent error tolerance
tol = 1.0e-10;
options = odeset('AbsTol', [tol, tol], 'RelTol', tol);
[tout yout] = ode45(@fcn_sho, linspace(0.0,3.0*pi,1025), [0.0 1.0], ...
    options);
```

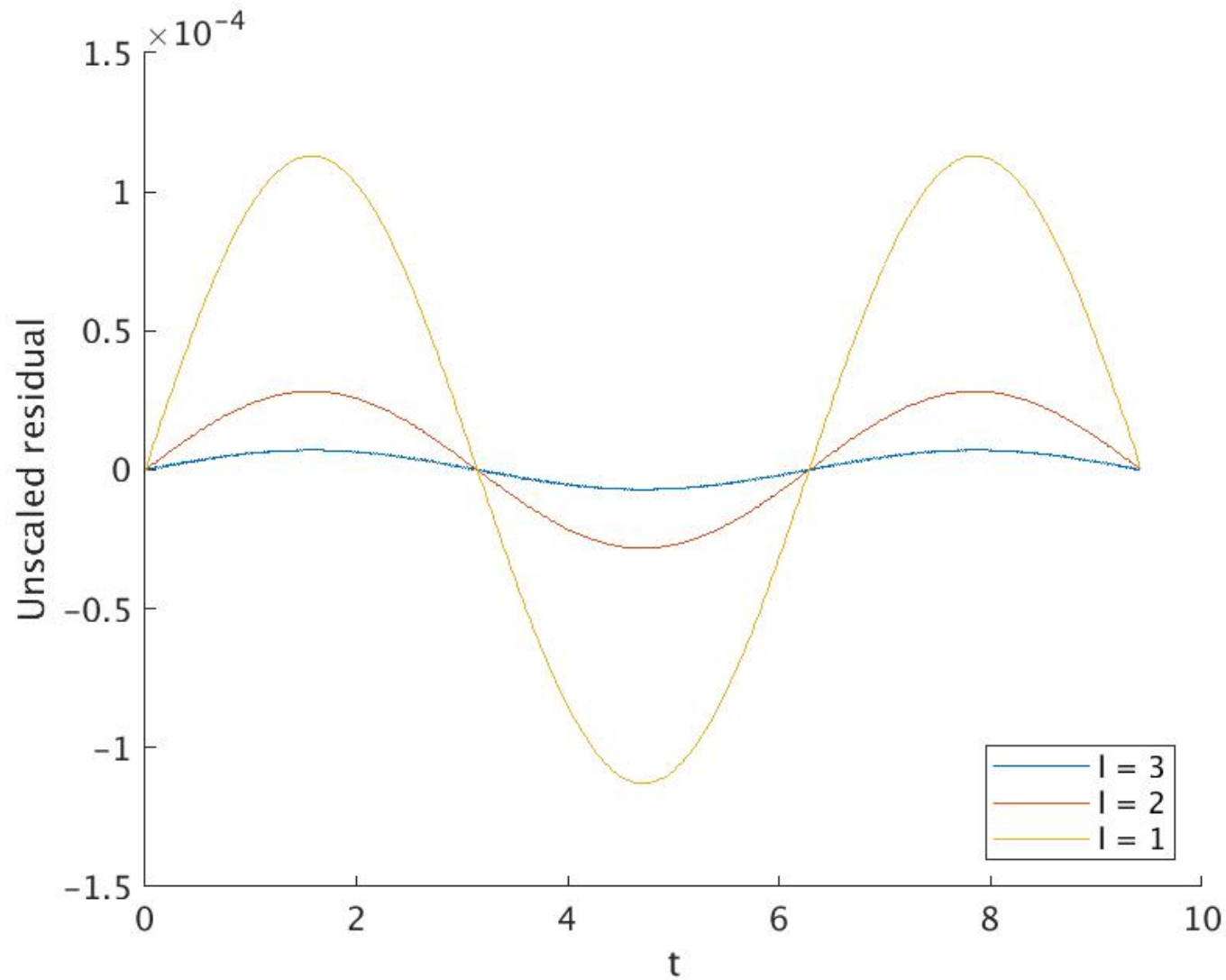
```

% Compute and plot three levels of scaled independent residuals
figure(1); clf; figure(2); clf
yir = yout(:,1);
tir = tout;
for ll = 1 : 3
    dt = tir(2) - tir(1);
    ir = zeros(1,length(yir));
    for j = 2 : length(yir) - 1
        ir(j) = (yir(j-1) - 2.0 * yir(j) + yir(j+1)) / dt^2 + yir(j);
    end
    % Plot unscaled residuals
    figure(1);
    hold on;
    plot(tir, ir);
    % Plot scaled residuals
    figure(2);
    hold on;
    plot(tir, 4.0^(3 - ll) * ir);
    yir = yir(1:2:end);
    tir = tir(1:2:end);
end;

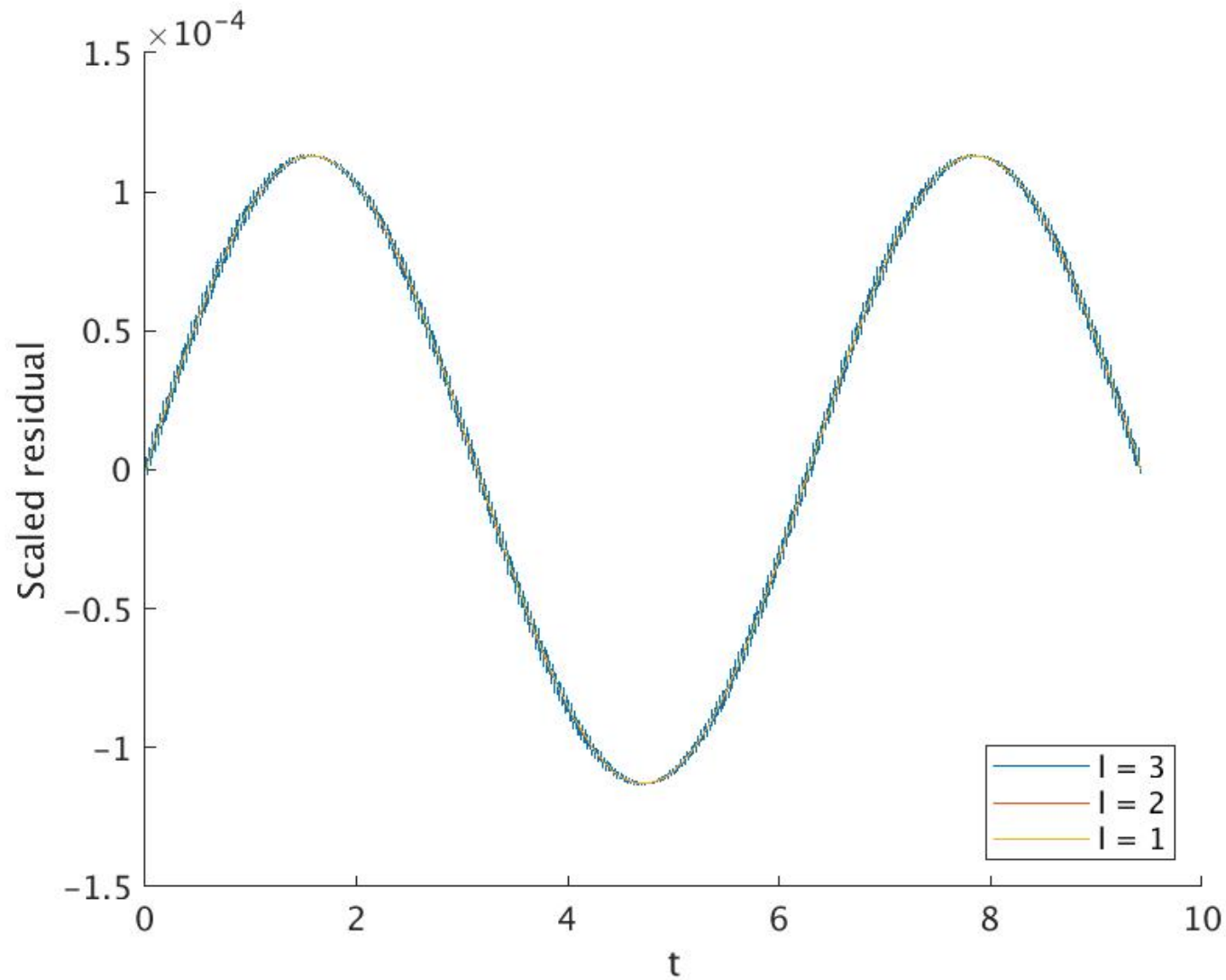
```

```
figure(1);
xlabel('t');
ylabel('Unscaled residual');
legend('l = 3', 'l = 2', 'l = 1', 'Location', 'SouthEast');
print('sho_ir_unscaled.jpg', '-djpeg');
figure(2);
xlabel('t');
ylabel('Scaled residual');
legend('l = 3', 'l = 2', 'l = 1', 'Location', 'SouthEast');
print('sho_ir_scaled.jpg', '-djpeg');
```

- Plot of unscaled independent residuals (AbsTol = $1.0e-10$, RelTol = $1.0e-10$)



- Plot of scaled independent residuals (AbsTol = $1.0e-10$, RelTol = $1.0e-10$)



Example: Orbiting Dumbbell (Model for Hyperion)

Function: fcn_dumb

```
function dydt = fcn_dumb(t, y)
% Function fcn_dumb evaluates derivatives for orbiting dumbbell
% problem.
    global mu d;

    dydt = zeros(6,1);

    xc = y(1);
    yc = y(3);
    th = y(5);
    om = y(6);

    d1 = mu * d;
    d2 = (1.0 - mu) * d;

    x1 = xc + d1 * cos(th);
    y1 = yc + d1 * sin(th);
    x2 = xc - d2 * cos(th);
    y2 = yc - d2 * sin(th);
```

```
r1m3 = 1.0 / (x1^2 + y1^2)^1.5;
```

```
r2m3 = 1.0 / (x2^2 + y2^2)^1.5;
```

```
c1 = -1.0;
```

```
c2 = 1.0 / d;
```

```
dydt(1) = y(2);
```

```
dydt(2) = c1 * ((1.0 - mu) * x1 * r1m3 + ...  
              mu * x2 * r2m3);
```

```
dydt(3) = y(4);
```

```
dydt(4) = c1 * ((1.0 - mu) * y1 * r1m3 + ...  
              mu * y2 * r2m3);
```

```
dydt(5) = y(6);
```

```
dydt(6) = c2 * (r1m3 - r2m3) * ...  
            (sin(th) * xc - cos(th) * yc);
```

```
end
```

Function: energy_dumb

```
function [t_trans, t_rot, v_grav, e_tot] = energy_dumb(y, m1, m2, d)
    % energy_dumb: Computes energy quantities from solution y.
    xc = y(:,1); vxc = y(:,2); yc = y(:,3);
    vyc = y(:,4); th = y(:,5); om = y(:,6);

    t_trans = 0.5 * (m1 + m2) * (vxc.^2 + vyc.^2);

    t_rot = 0.5 * (m1 * m2) / (m1 + m2) * d^2 * om.^2;

    mu = m2 / (m1 + m2);
    d1 = mu * d;
    d2 = (1 - mu) * d;
    x1 = xc + d1 * cos(th);
    y1 = yc + d1 * sin(th);
    x2 = xc - d2 * cos(th);
    y2 = yc - d2 * sin(th);
    r1 = sqrt(x1.^2 + y1.^2);
    r2 = sqrt(x2.^2 + y2.^2);
    v_grav = -(m1 ./ r1 + m2 ./ r2);

    e_tot = t_trans + t_rot + v_grav;
end
```

Script: ode45_dumb

```
% ode45_dumb: Uses ODE45 to integrate equations for orbiting
% dumbbell problem
global mu d;

m1bym2 = 2.0;
mu = 1.0 / (1.0 + m1bym2);
d = 0.3;

m1 = 1.0;
m2 = m1 / m1bym2;

tmax = 60;
tol = 1.0e-12;
options = odeset('AbsTol', tol * ones(1,6), 'RelTol', tol);

% Circular orbit ...
[tout yout] = ode45(@fcn_dumb, [0.0 tmax], ...
                   [1.0 0.0 0.0 1.0 0.0 0.0], options);

clf;
plot(tout, yout(:,6));
xlabel('t');
ylabel('\omega');
title('Circular Orbit -- Tolerance=1.0e-12');
```

```

print('dumb_omega_circular.jpg', '-djpeg');

[t_trans, t_rot, v_grav, e_tot] = energy_dumb(yout, m1, m2, d);

clf;
plot(tout, e_tot);
xlabel('t');
ylabel('E');
title('Circular Orbit -- Tolerance=1.0e-12');
print('dumb_etot_circular.jpg', '-djpeg');

% Elliptical orbit ...
tmax = 200;
[tout yout] = ode45(@fcn_dumb, [0.0 tmax], ...
                    [1.0 0.0 0.0 1.2 0.0 0.0], options);

clf;
plot(tout, yout(:,6));
xlabel('t');
ylabel('\omega');
title('Elliptical Orbit -- Tolerance=1.0e-12');
print('dumb_omega_elliptical.jpg', '-djpeg');

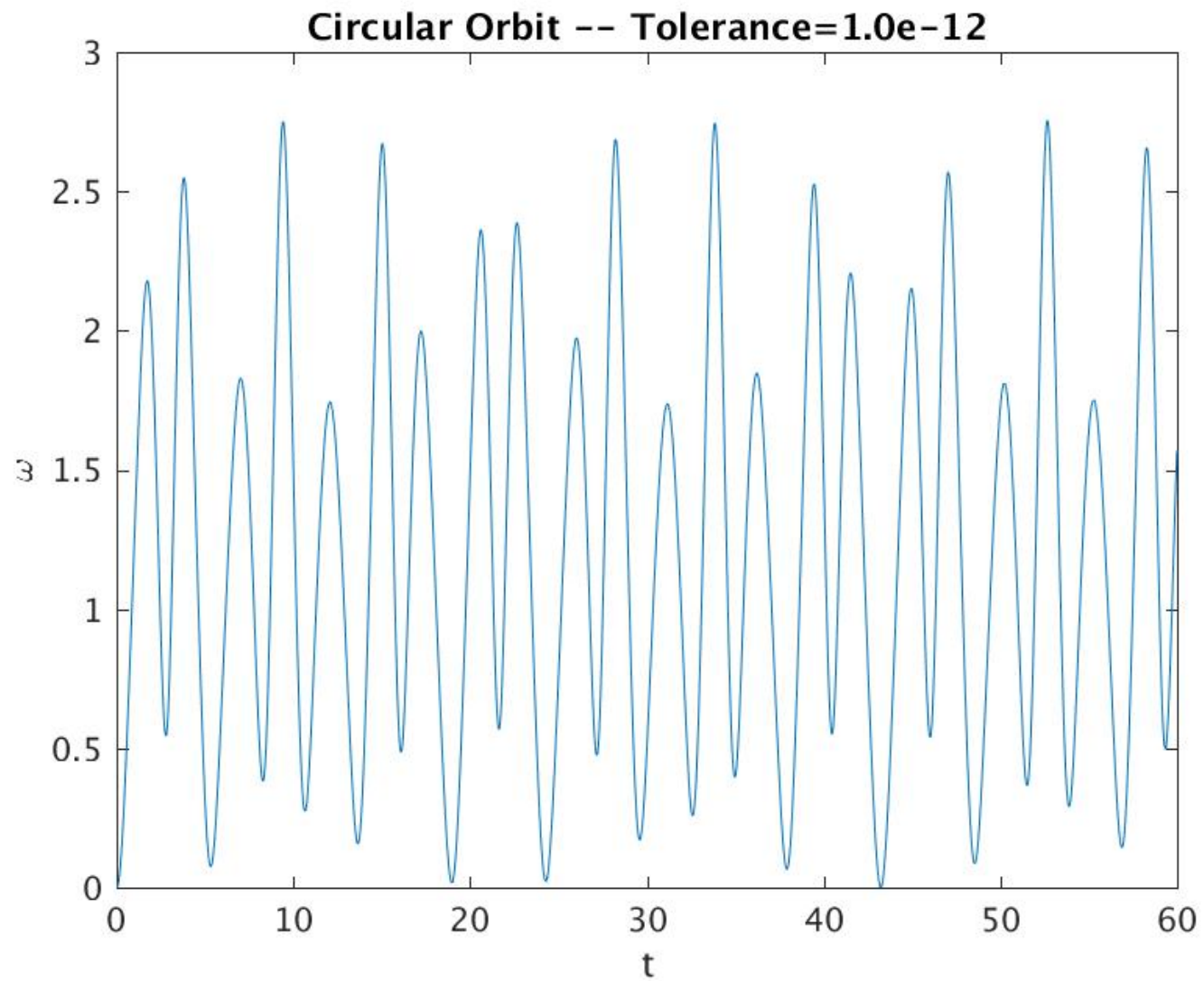
[t_trans, t_rot, v_grav, e_tot] = energy_dumb(yout, m1, m2, d);

```

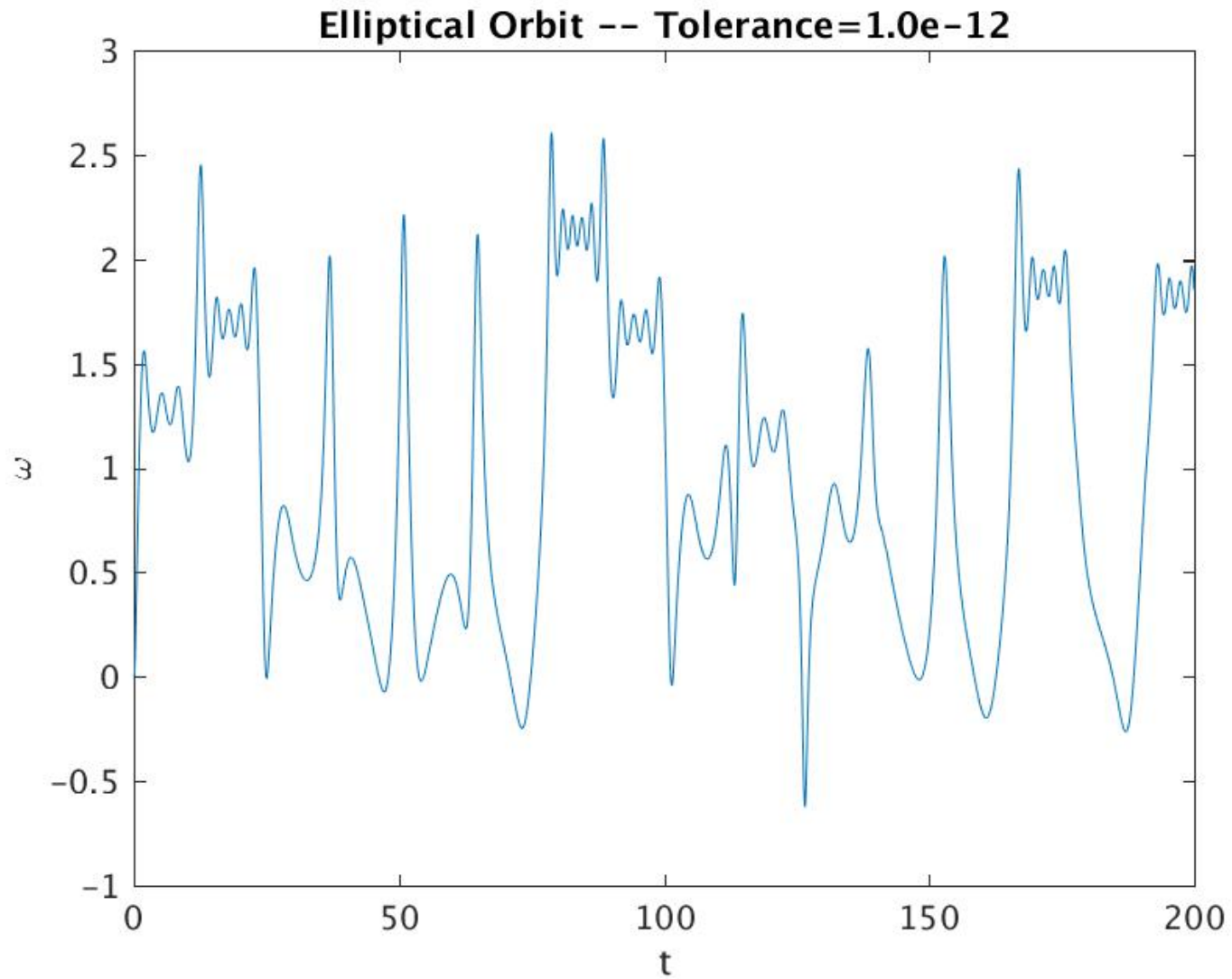


```
clf;
plot(tout, yout(:,6));
xlabel('t');
ylabel('\omega');
title('Elliptical Orbit -- Tolerance=1.0e-12');
print('dumb_omega_elliptical_long.jpg', '-djpeg');
```

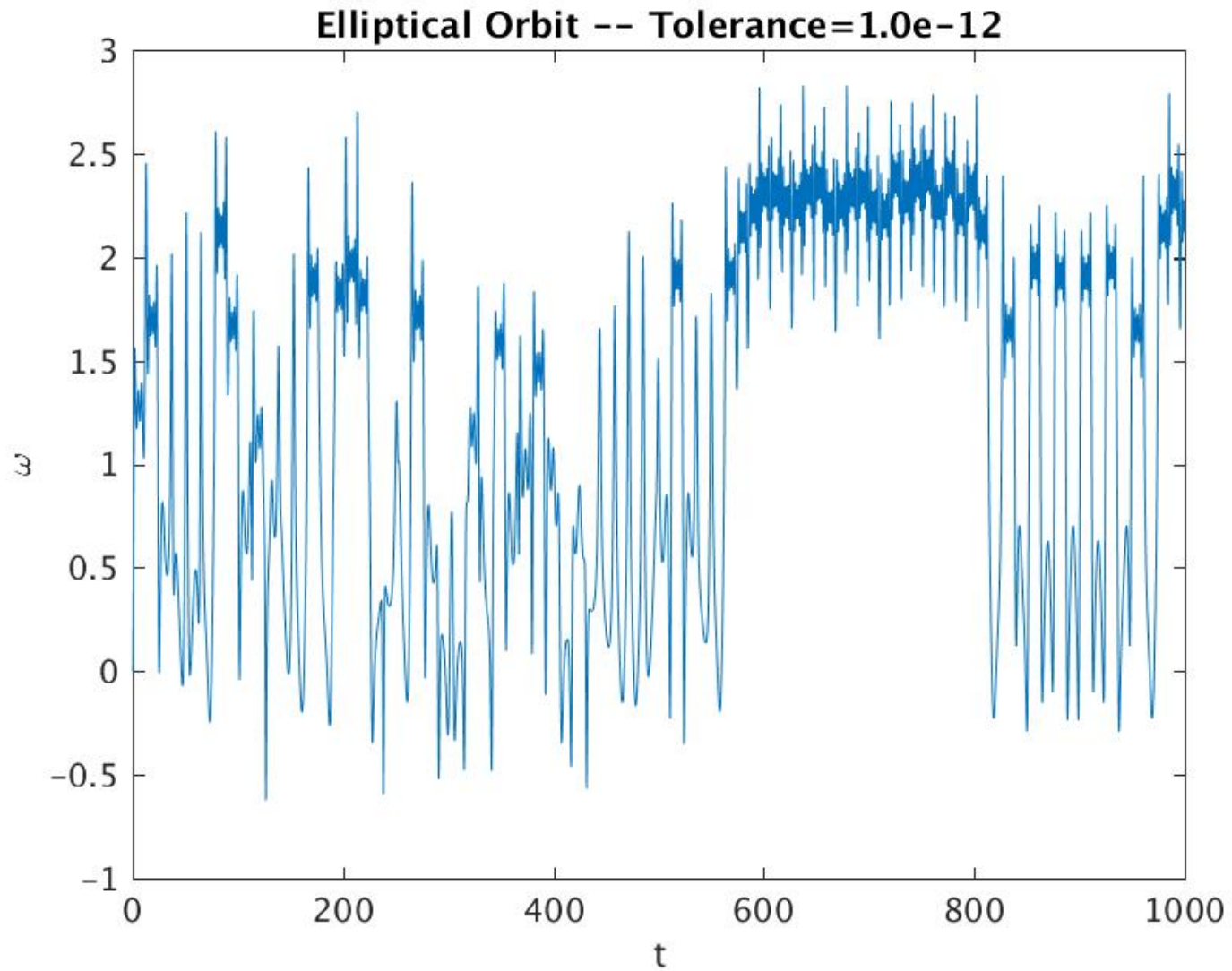

- Plot of $\omega(t)$ for circular orbit



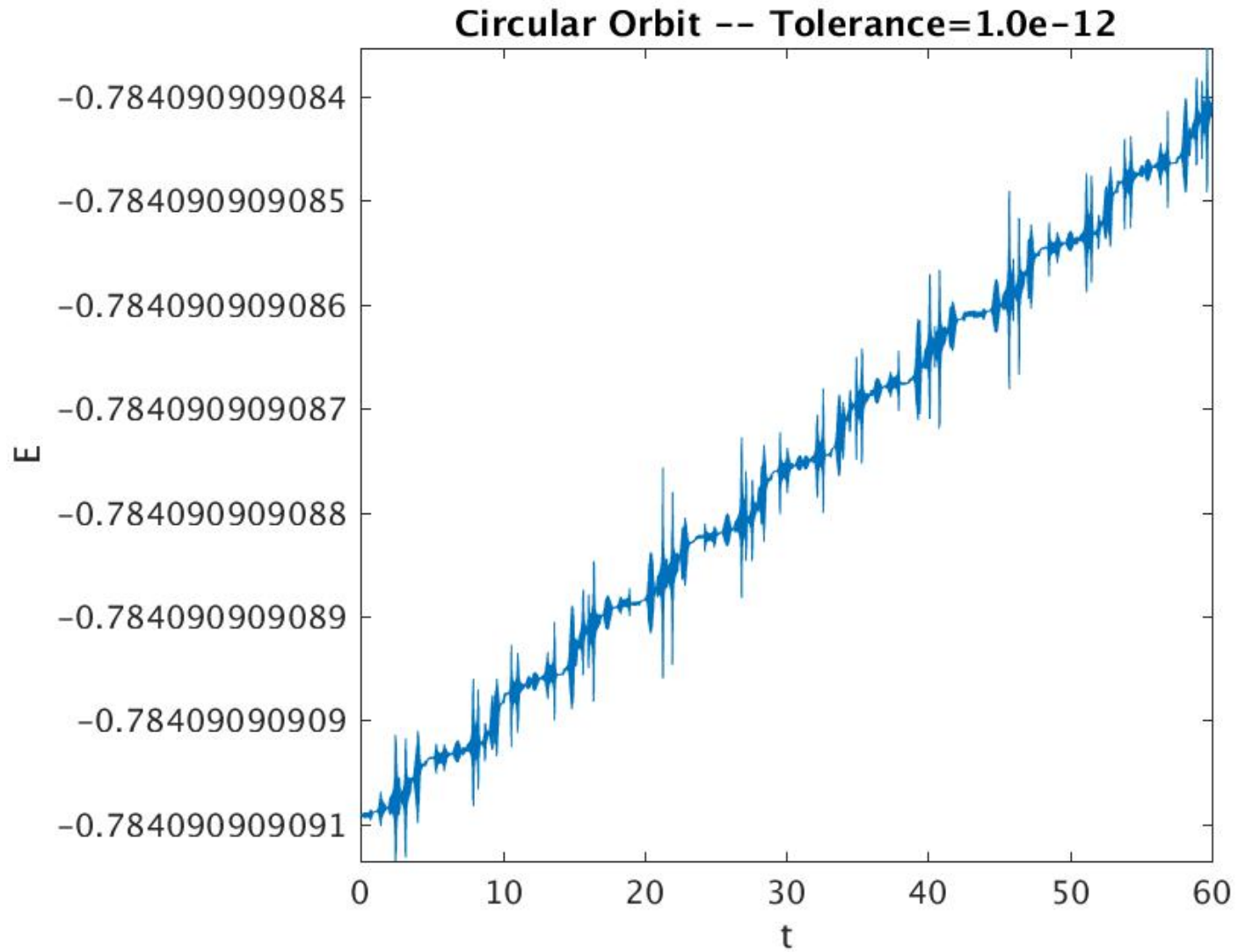
- Plot of $\omega(t)$ for elliptical orbit



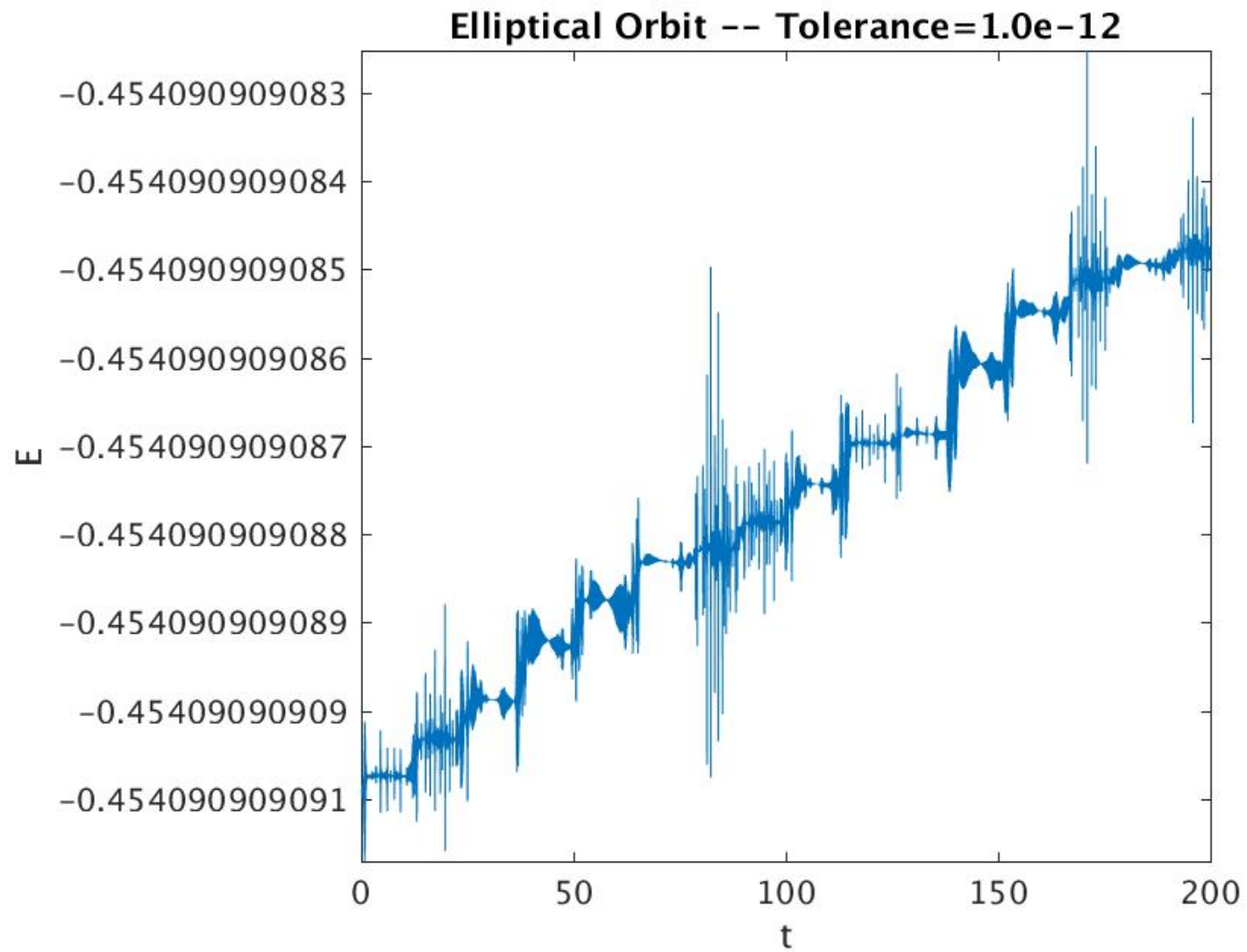
- Plot of $\omega(t)$ for elliptical orbit, longer integration time



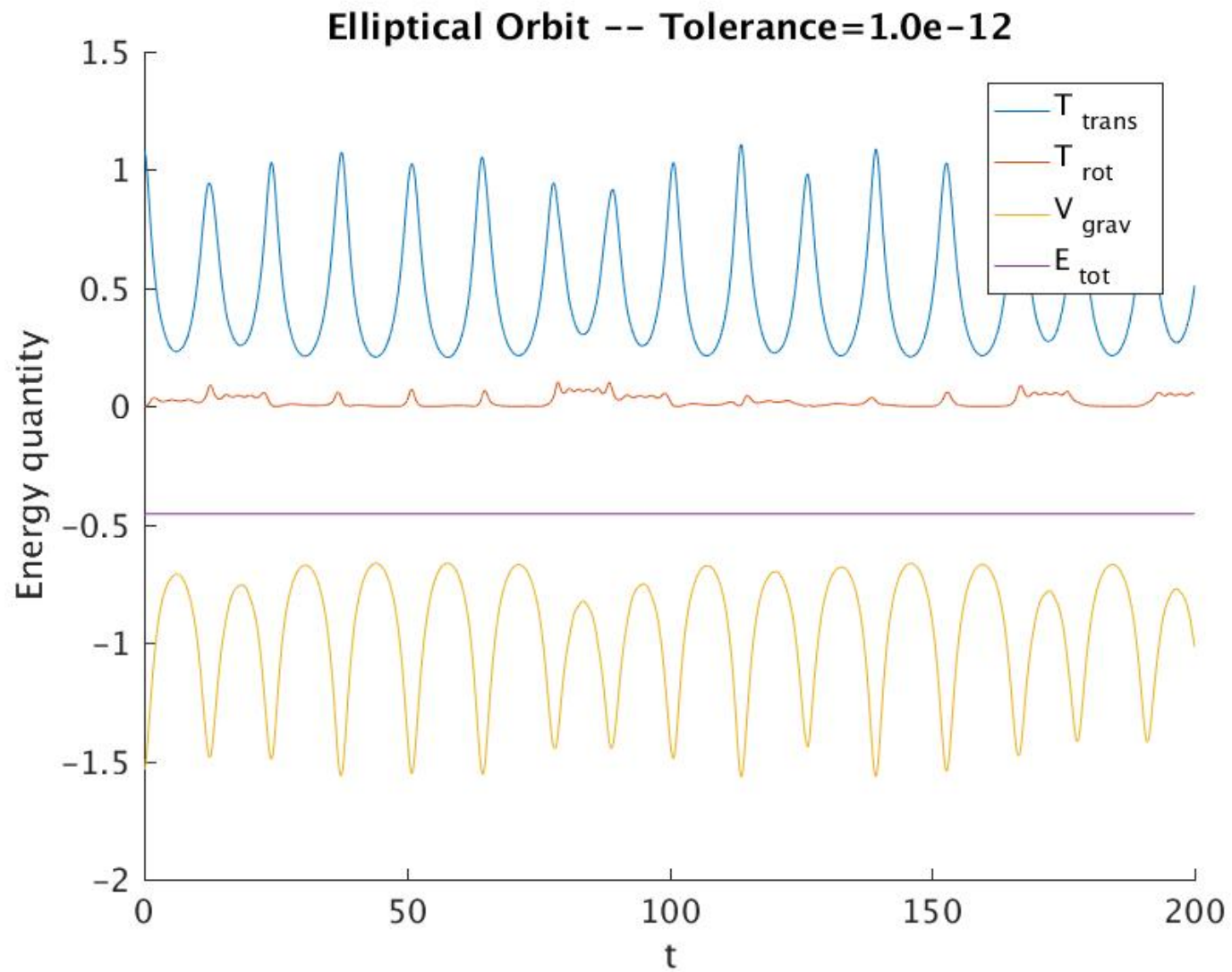
- Total energy, circular orbit



- Total energy, elliptical orbit



- Various energy quantities, elliptical orbit



Example: Toda Lattice

Function: fcn_toda

```
function dydt = fcn_toda(t, y)
% Function dydt evaluates derivatives for 1D Toda lattice.
%
% y is a vector of length 2*ns, where ns is the number of sites.
% y(1:ns)      = generalized coordinates
% y(ns+1:2*ns) = generalized momenta
%
% Static boundary conditions are imposed on the momenta.

ns = length(y) / 2;
dydt = zeros(length(y), 1);

% Time derivatives of coordinates ...
dydt(1:ns, 1) = y(ns+1:2*ns);
% Time derivatives of momenta ...
dydt(ns+2:2*ns-1, 1) = exp(-(y(2:ns-1) - y(1:ns-2))) - ...
    exp(-(y(3:ns) - y(2:ns-1)));
% Impose static boundary conditions on momenta ...
dydt(ns+1, 1) = 0;
dydt(2*ns, 1) = 0;
end
```

Script: ode45_toda1

```
% ode45_toda1: Uses ODE45 to integrate equations for 1D Toda lattice
% following
% https://www.mat.univie.ac.at/~gerald/ftp/book-jac/toda.html
%
% Single soliton setup.

% Set to 1 to enable .avi creation
avienable = 1;
avisecs = 20;
aviframerate = 25;
aviframes = avisecs * aviframerate;
avifilename = 'toda1.avi';

% Minimum and maximum time ...
tmin = 0.0;
tmax = 200.0;

% Sites and number of sites ...
n = 1:401;
ns = length(n);

% Tolerance parameters ...
abstol = 1.0e-6;
reltol = 1.0e-6;
```



```
options = odeset('AbsTol', abstol * ones(1,2*ns), 'RelTol', reltol);

% Set initial data ...

% yinit = solitons(n, n0, qp, gamma, kappa, sigma);
yinit = solitons(n, ...
    [50.0], ...
    [1.0], ...
    [1.0], ...
    [0.05], ...
    [1.0]);

% Integrate system using ode45 ...
[tout yout] = ode45(@fcn_toda, [tmin tmax], yinit, options);

% Open .avi file if video generation enabled ...
if avienable
    avifreq = round(length(tout) / aviframes);
    aviobj = VideoWriter(avifilename);
    open(aviobj);
end

% Loop over time steps ...
for it = 1 : length(tout)
    if avienable
```

```

    if (it == 1) | (it == length(tout))
        framecount = 5 * aviframerate;
    elseif rem(it,avifreq) == 0
        framecount = 1;
    else
        framecount = 0;
    end
end
% Plot displacements ...
if ~avienable | (avienable & framecount)
    clf;
    hold on;
    axis square;
    box on;
    xlim([0, ns]);
    ylim([min(yinit(1:ns)), max(yinit(1:ns))]);
    xlabel('n');
    ylabel('q(n)');
    plot(n, yout(it,1:ns), 'Marker', 'o', 'Markersize', 2.0, ...
        'MarkerEdgeColor', 'r', 'MarkerFaceColor', 'r');
    titlestr = sprintf('Step: %d    Time: %.1f', it, tout(it));
    title(titlestr, 'FontSize', 16, 'FontWeight', 'bold', ...
        'Color', [0.25, 0.42, 0.31]);
    drawnow;
end

```

```
% Output JPEG of first timestep ...
if it == 1
    print('toda1.jpg', '-djpeg');
end
% avi recording, if enabled ...
if avienable
    for iframe = 1 : framecount
        writeVideo(aviobj, getframe(gcf));
    end
end
end

% Close avi file ...
if avienable
    close(aviobj);
    fprintf('Created video file: %s\n', avifilename);
end
```

Function: solitons

```
function yinit = solitons(n, n0, qp, gamma, kappa, sigma)
% solitons defines initial values corresponding to multi-soliton
% Toda lattice.

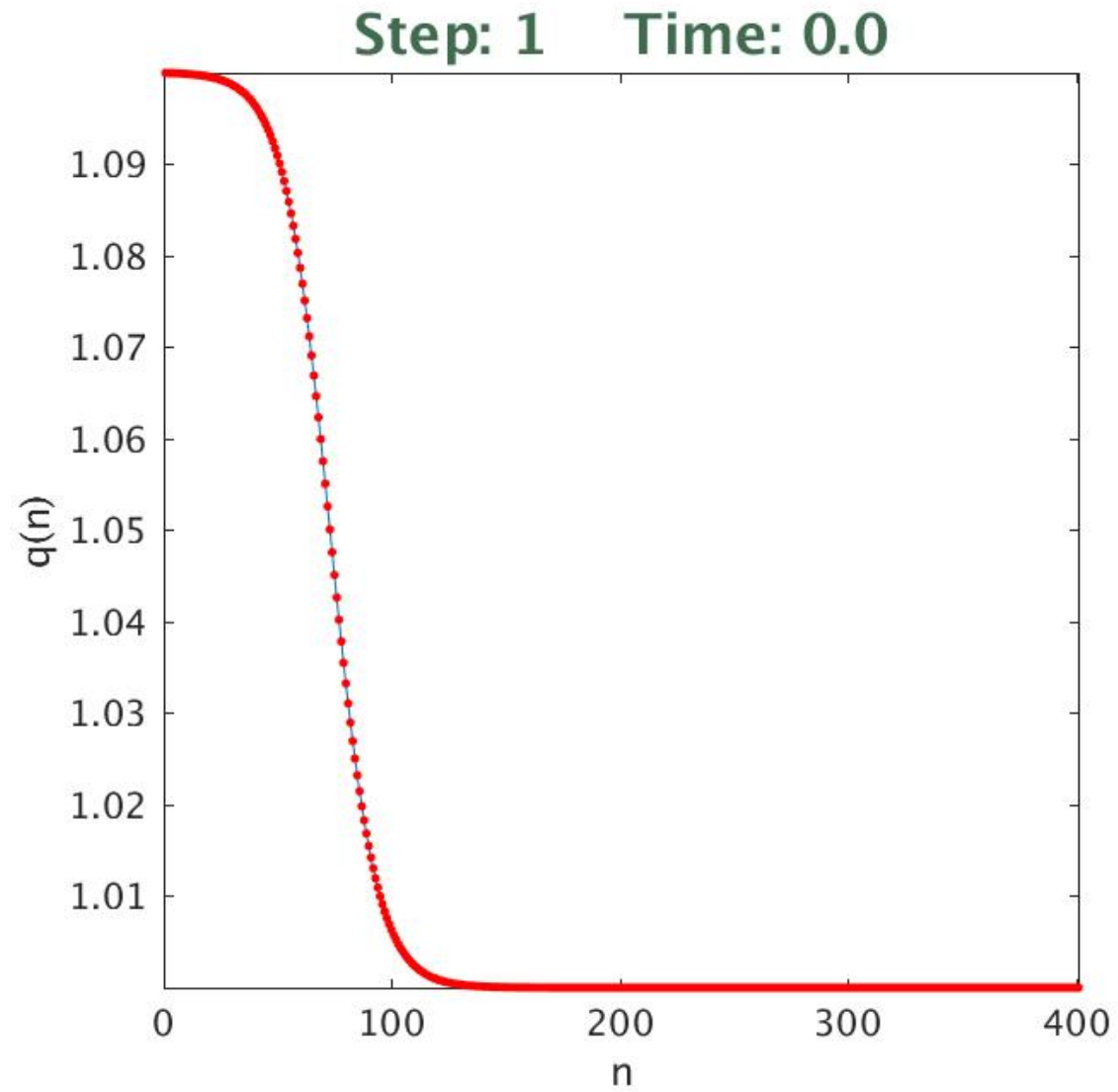
yinit = zeros(2 * length(n),1);
for is = 1 : length(n0)
    t0 = n0(is) * kappa(is) / sinh(kappa(is));
    for ii = 1 : length(n)
        yinit(ii) = yinit(ii) + ...
            q1(n(ii), t0, qp(is), gamma(is), kappa(is), sigma(is));
        yinit(length(n) + ii) = yinit(length(n) + ii) + ...
            dq1dt(n(ii), t0, qp(is), gamma(is), kappa(is), sigma(is));
    end
end
end
end
```

Functions: q1 and dq1dt

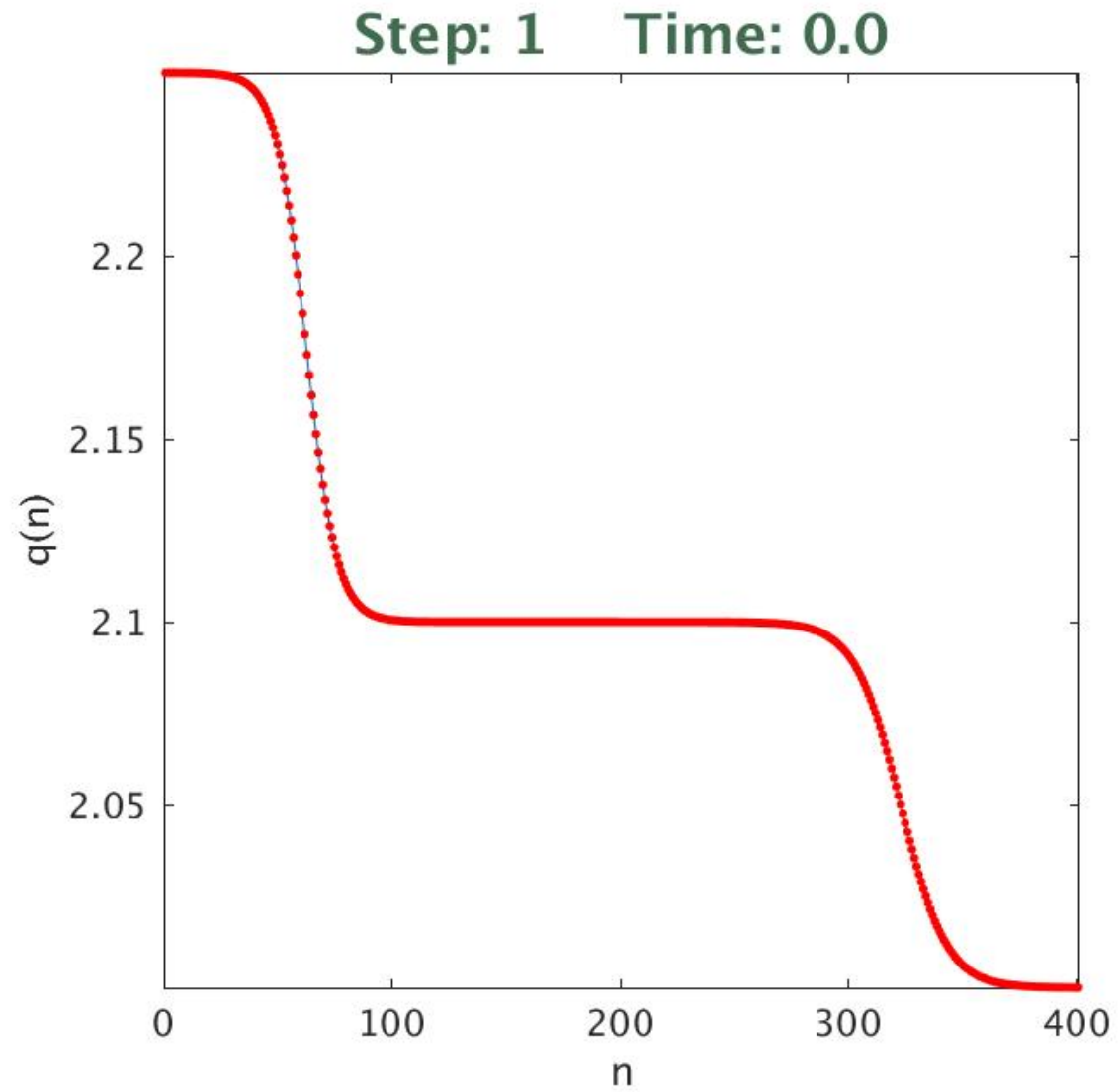
```
function q1nt = q1(n,t,qplus,gamma,kappa,sigma)
% q1: Defines single soliton solution for Toda lattice following
% equation (3) of
% https://www.mat.univie.ac.at/~gerald/ftp/book-jac/toda.html
q1nt = qplus + log( ...
    (1 + (gamma / (1 - exp(-2*kappa)))) * ...
    exp(-2*kappa*n + 2*sigma*sinh(kappa)*t)) / ...
    (1 + (gamma / (1 - exp(-2*kappa)))) * ...
    exp(-2*kappa*(n + 1) + 2*sigma*sinh(kappa)*t)) ...
);
end

function dq1dnt = dq1dt(n,t,qplus,gamma,kappa,sigma)
% dq1dt: Defines time derivative of single soliton solution for Toda
% lattice following equation (3) of
% https://www.mat.univie.ac.at/~gerald/ftp/book-jac/toda.html
t1 = exp(-2*kappa*n + 2*sigma*sinh(kappa)*t);
t2 = exp(-2*kappa*(n + 1) + 2*sigma*sinh(kappa)*t);
t3 = 1 - exp(-2*kappa);
t4 = (1 + gamma*t2 / t3);
dq1dnt = (2*gamma*sigma*sinh(kappa)*t1 / (t3 * t4) - ...
    2*(1 + gamma*t1/t3)*gamma*sigma*sinh(kappa) * t2 / ...
    (t4^2 * t3)) * t4 / (1 + gamma*t1 / t3);
end
```

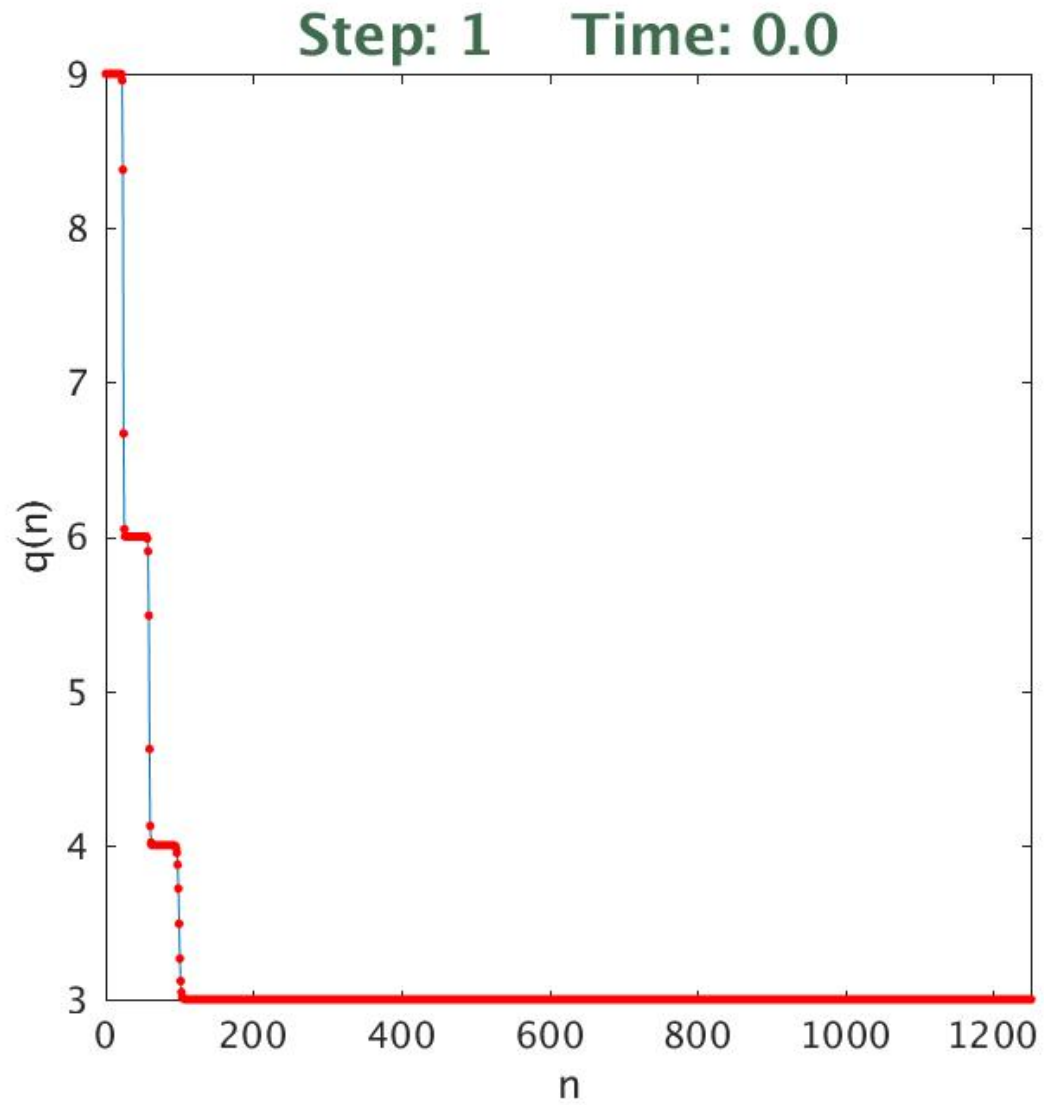
- Initial conditions: one soliton



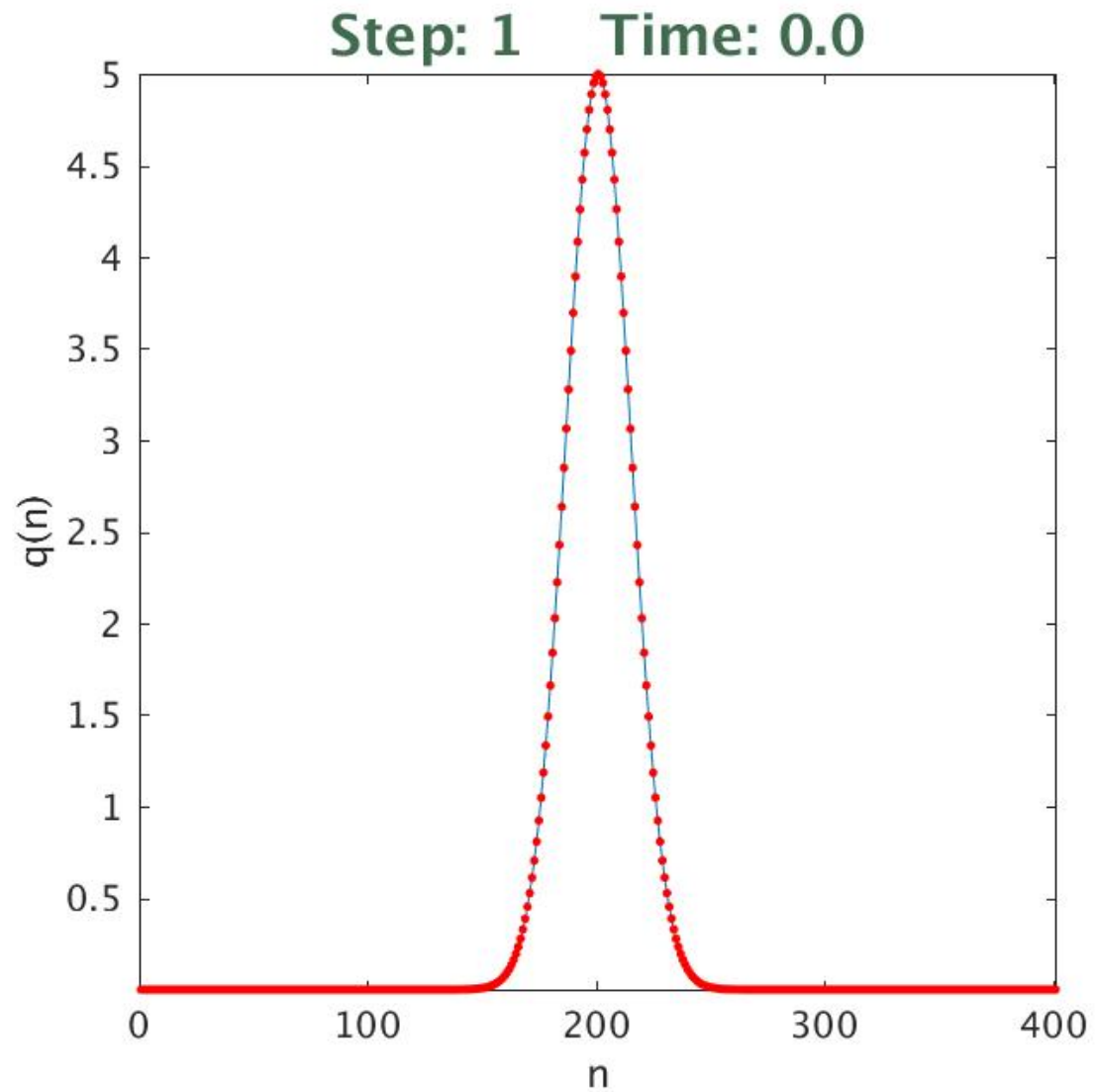
- Initial conditions: two solitons



- Initial conditions: three solitons



- Initial conditions: Gaussian pulse, $p(n, 0) = 0$



Example: van der Pol Oscillator

Function: fcn_vdp

```
function dydt = fcn_vdp(t, y)
% Function dydt evaluates derivatives for van der Pol oscillator
% following Tsatsos:
% https://arxiv.org/pdf/0803.1658.pdf
    global a b omega;

    dydt = ones(2,1);
    dydt(1) = y(2);
    dydt(2) = -y(1) - a * (y(1)^2 - 1) * y(2) + b * sin(omega * t);
end
```

Function: ode45_vdp_b

```
% ode45_vdp: Uses ODE45 to integrate equations for Van de Pol
% oscillator following Tsatsos:
% https://arxiv.org/pdf/0803.1658.pdf
%
% This script generates bifurcation diagrams.
global a b omega;

% Tolerance parameters ...
abstol = 1.0e-6;
reltol = 1.0e-6;
options = odeset('AbsTol', abstol * [1 1]', 'RelTol', reltol);

% Control parameters ...
a = 5.0;
b = 5.0;
vomega = 3.3695:0.000005:3.370;
nomega = length(vomega);

% Number of points in Poincare section ...
np = 500;

% Define space for bifurcation plot variables ...
xb = zeros(np,nomega);
```

```

% Initial data ...
yinit = [2.0, 0.0];

for iomega = 1 : nomega
    iomega
    omega = vomega(iomega);
    % Output times ...
    tspan = [0:2000] * (2.0 * pi / omega);
    % Integrate system using ode45 ...
    [tout yout] = ode45(@fcn_vdp, tspan, yinit, options);

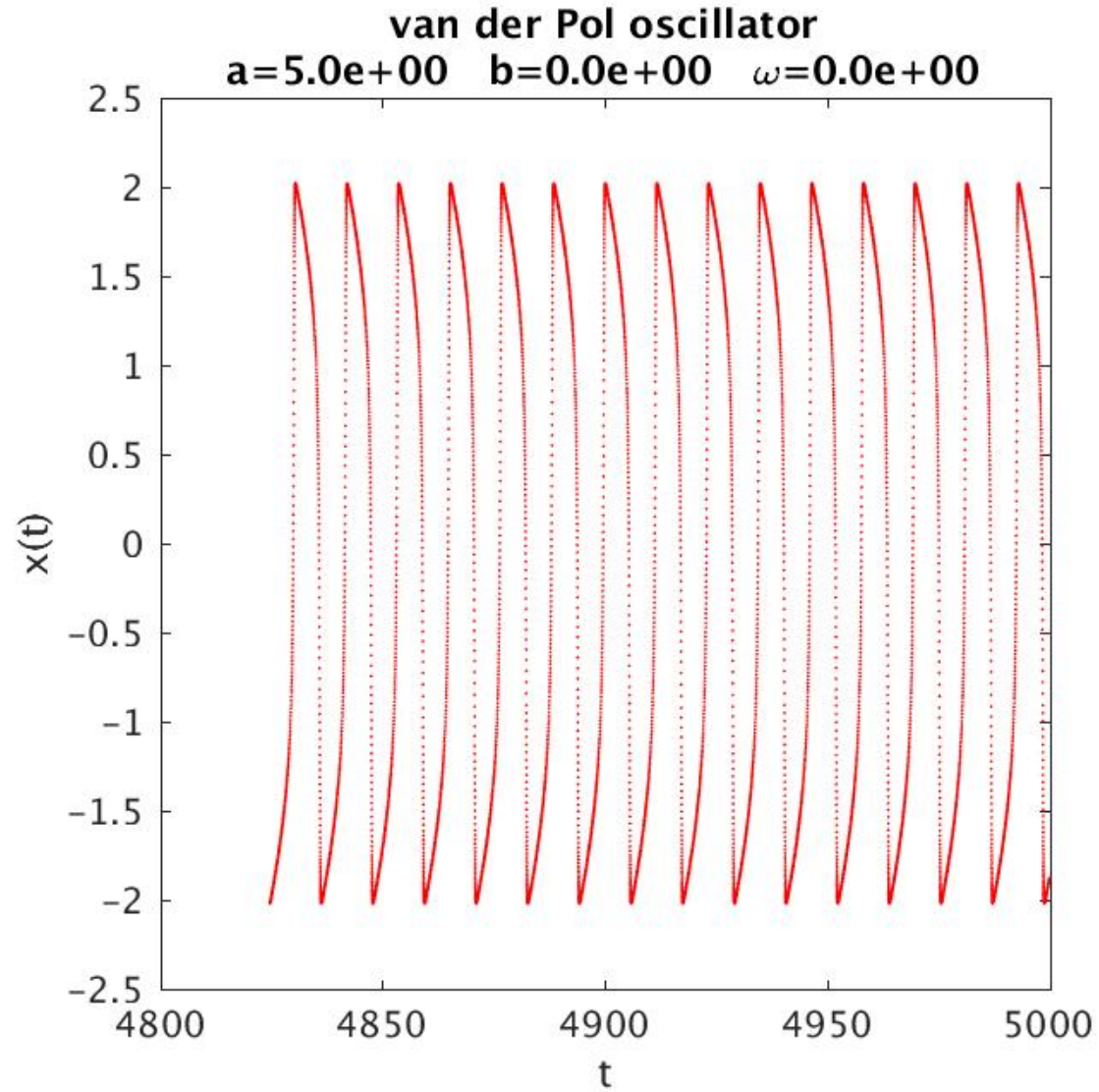
    % Save Poincare points ...
    xb(:,iomega) = yout(end-np+1:end,1);

    % Make bifurcation plot ...
    figure(1);
    clf;
    hold on;
    axis square;
    box on;
    xlabel('\omega');
    ylabel('x_p(t)');
    titlestr = ...
        sprintf('van der Pol oscillator\ na=%.1f    b=%.1f', a, b);
    title(titlestr);

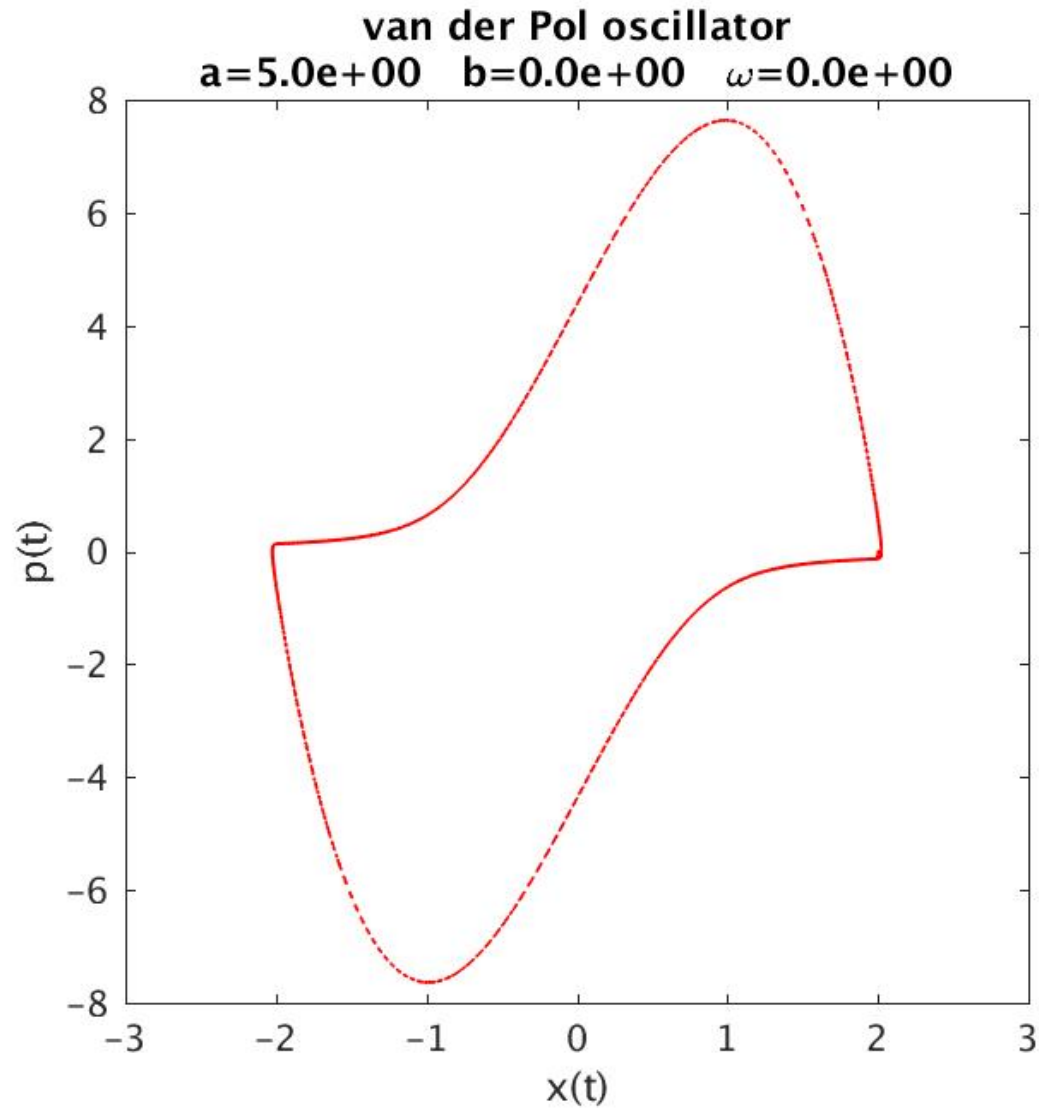
```

```
for i = 1 : iomega
    plot(repmat(vomega(i),1,np), xb(:,i), ...
        'ro', 'Markersize', 0.5);
end
drawnow
end
print('vdp_bifurcation.jpg', '-djpeg');
```

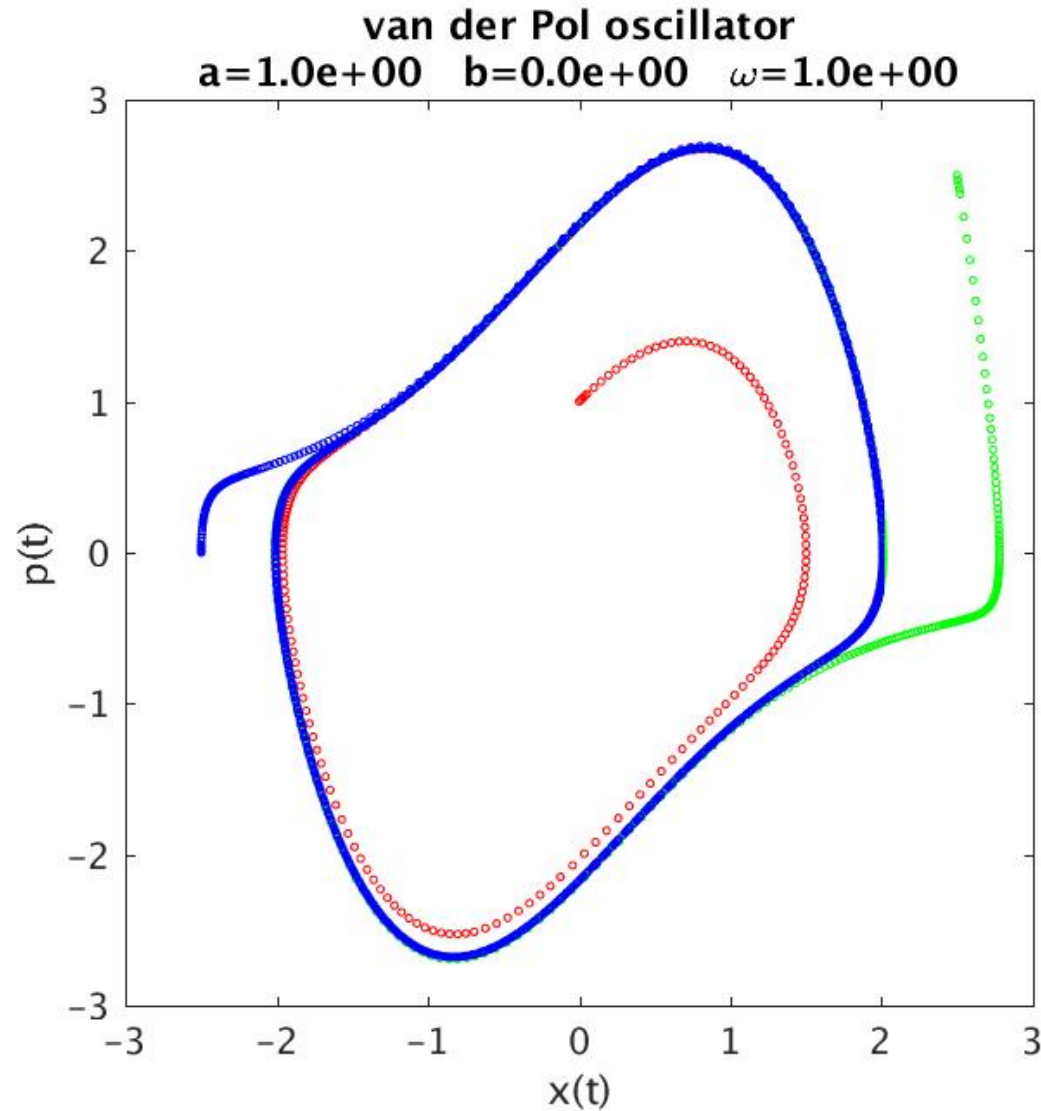
- Oscillator displacement for typical evolution with no driving ($b = 0$)



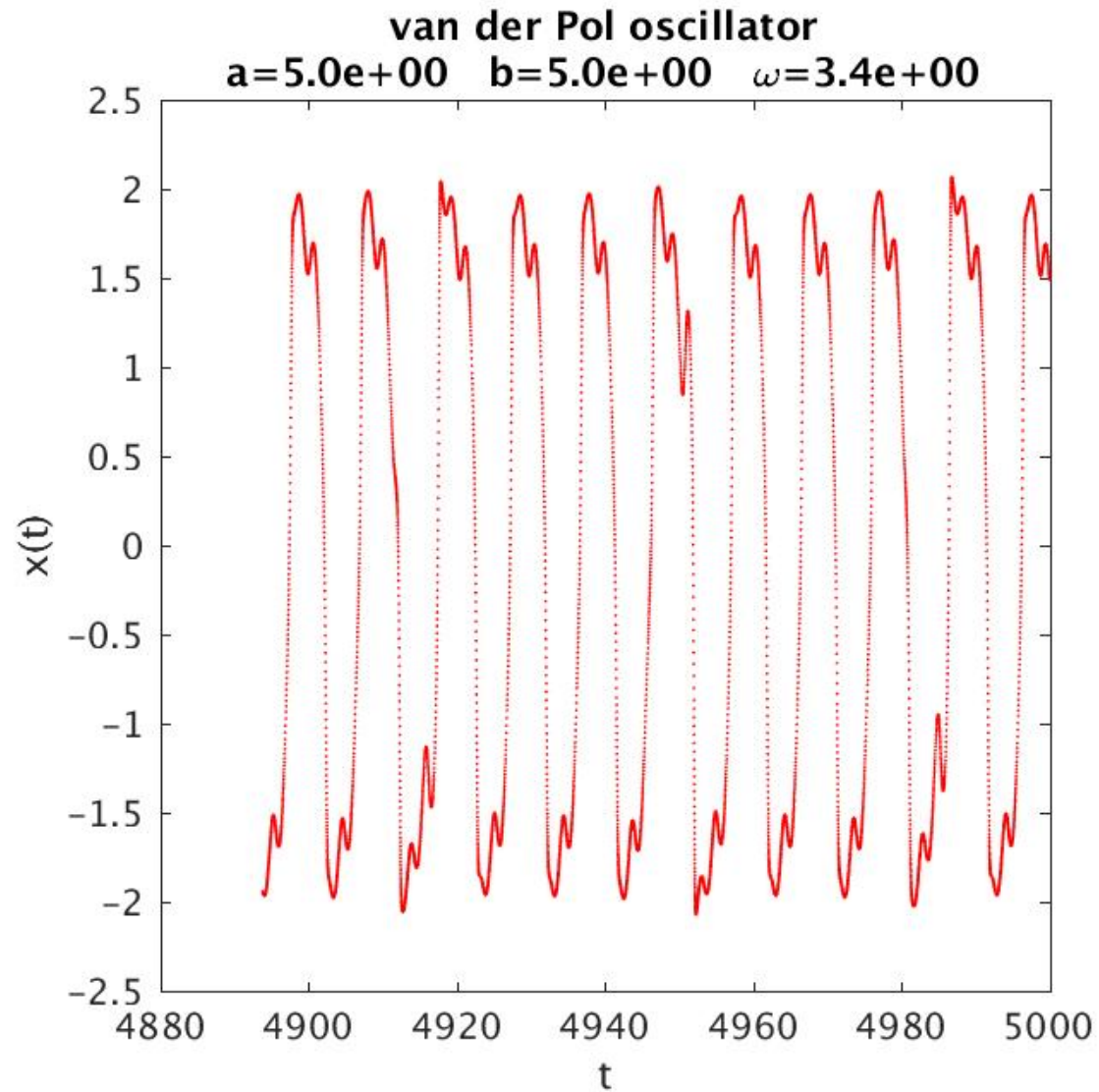
- Phase-space trajectory for typical evolution with no driving ($b = 0$)



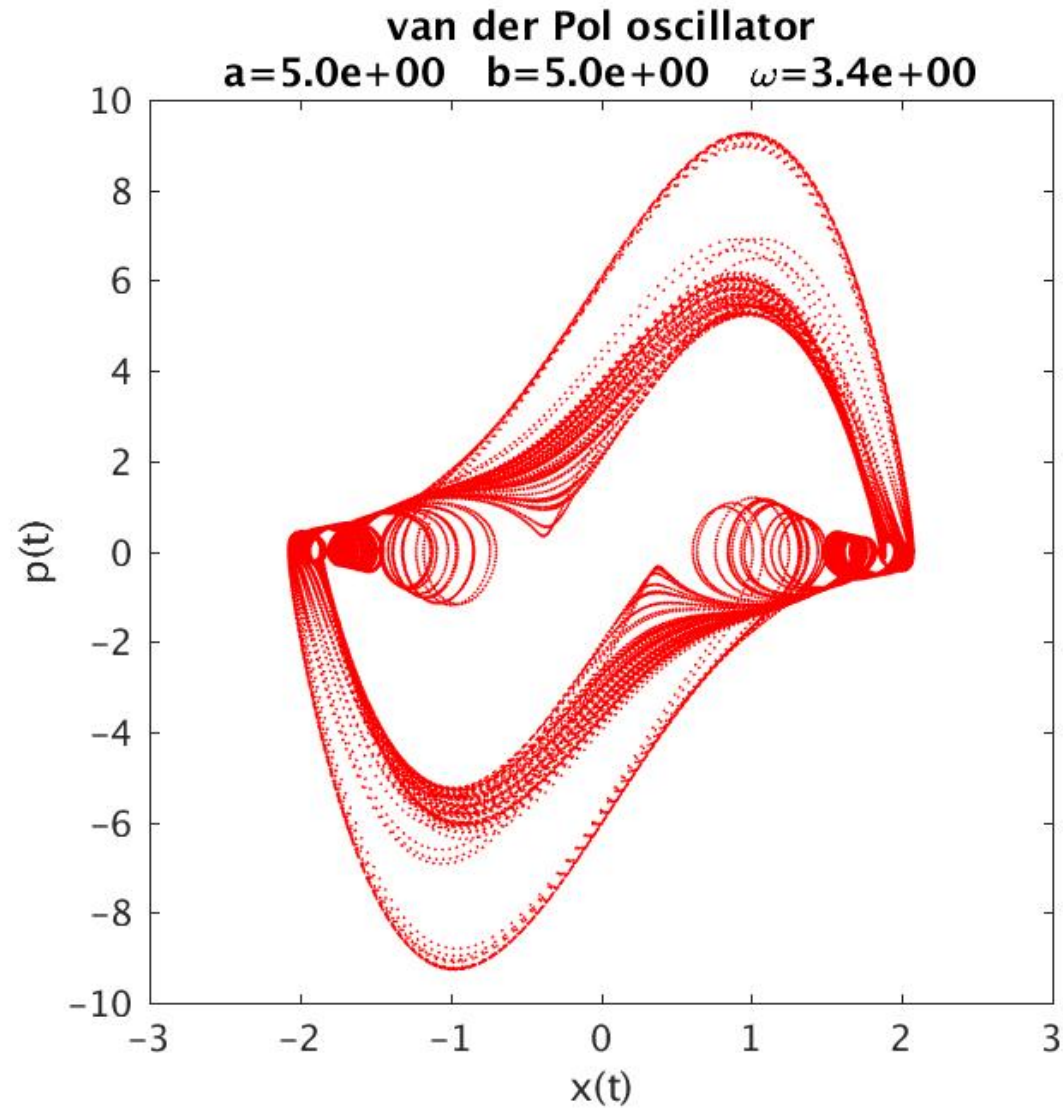
- Phase-space evolution of 3 distinct initial conditions with $b = 0$ (no driving force). Illustrates how dynamics tend to limit cycle (depends only on a).



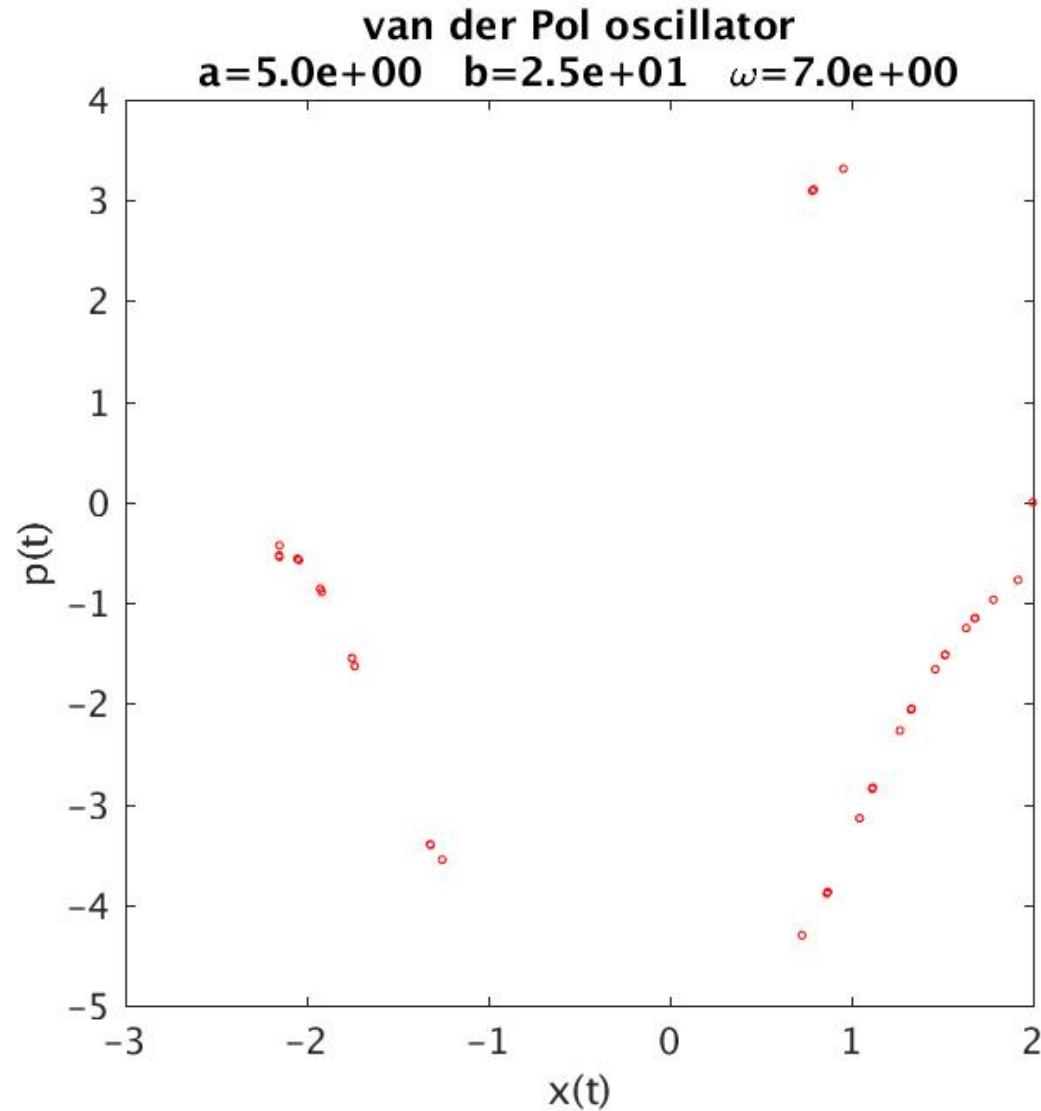
- (Driven) oscillator displacement in a chaotic regime.



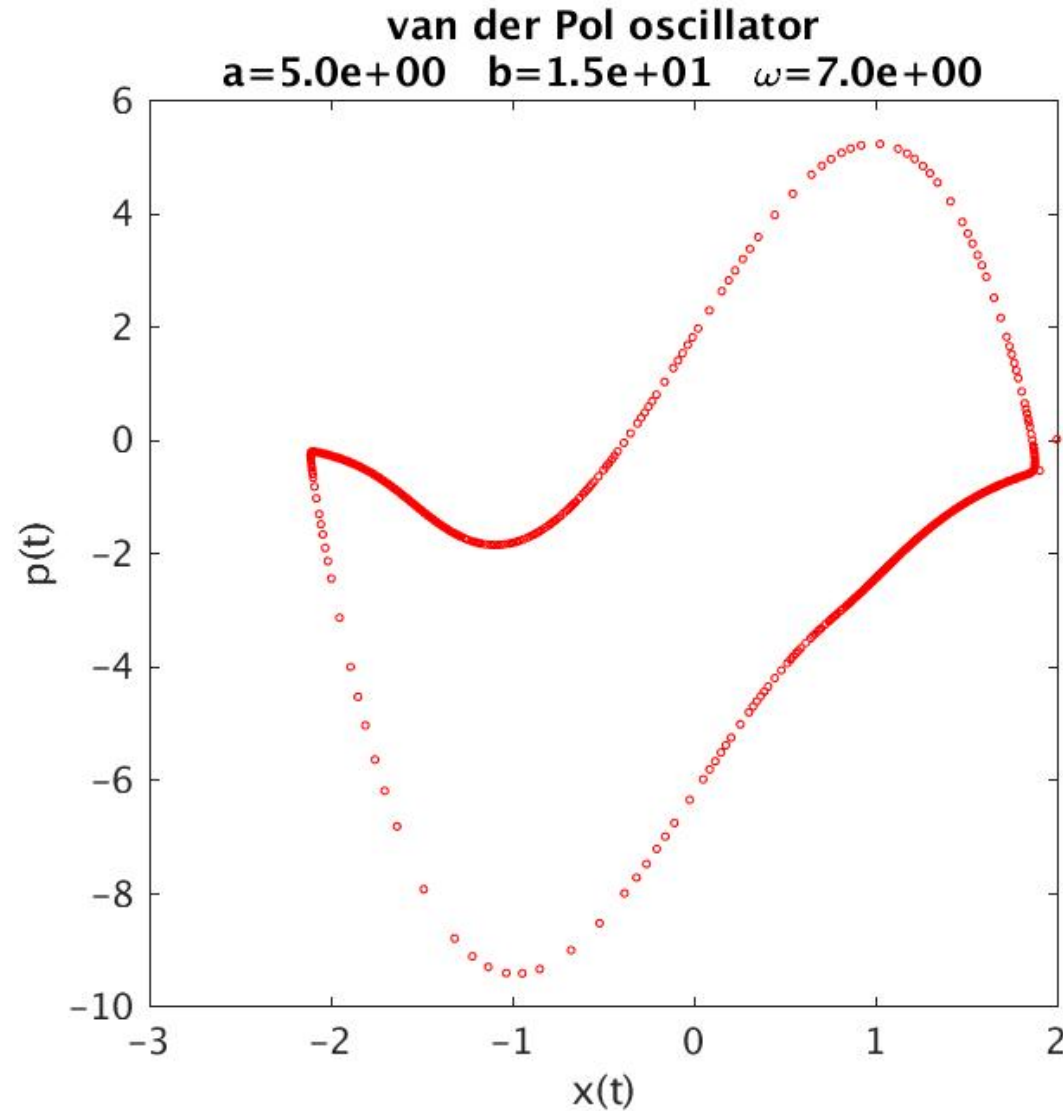
- Phase-space evolution for driven oscillator in a chaotic regime.



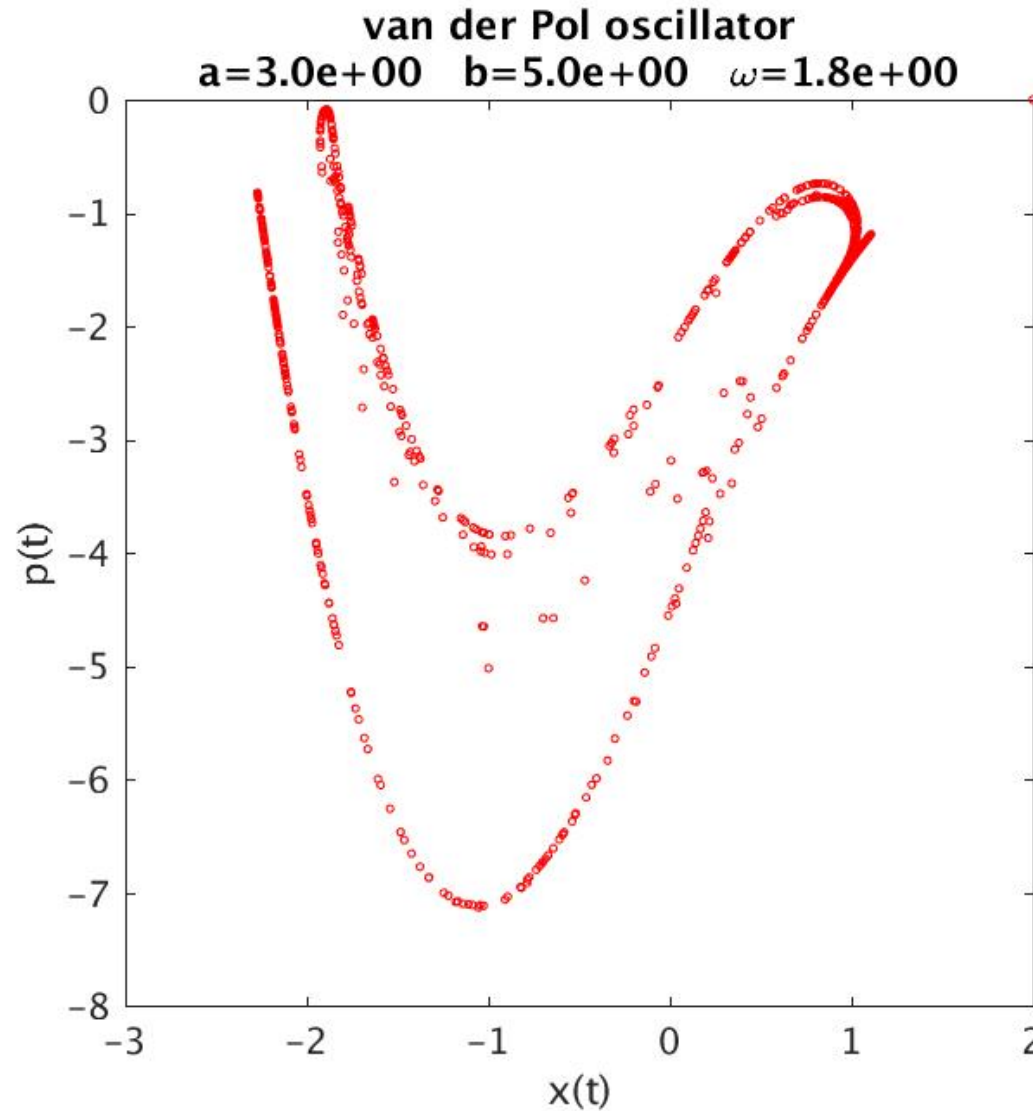
- Poincaré section for driven oscillator showing periodic trajectory. Output times are $2\pi n/\omega$, $n = 0, \dots, 1000$.



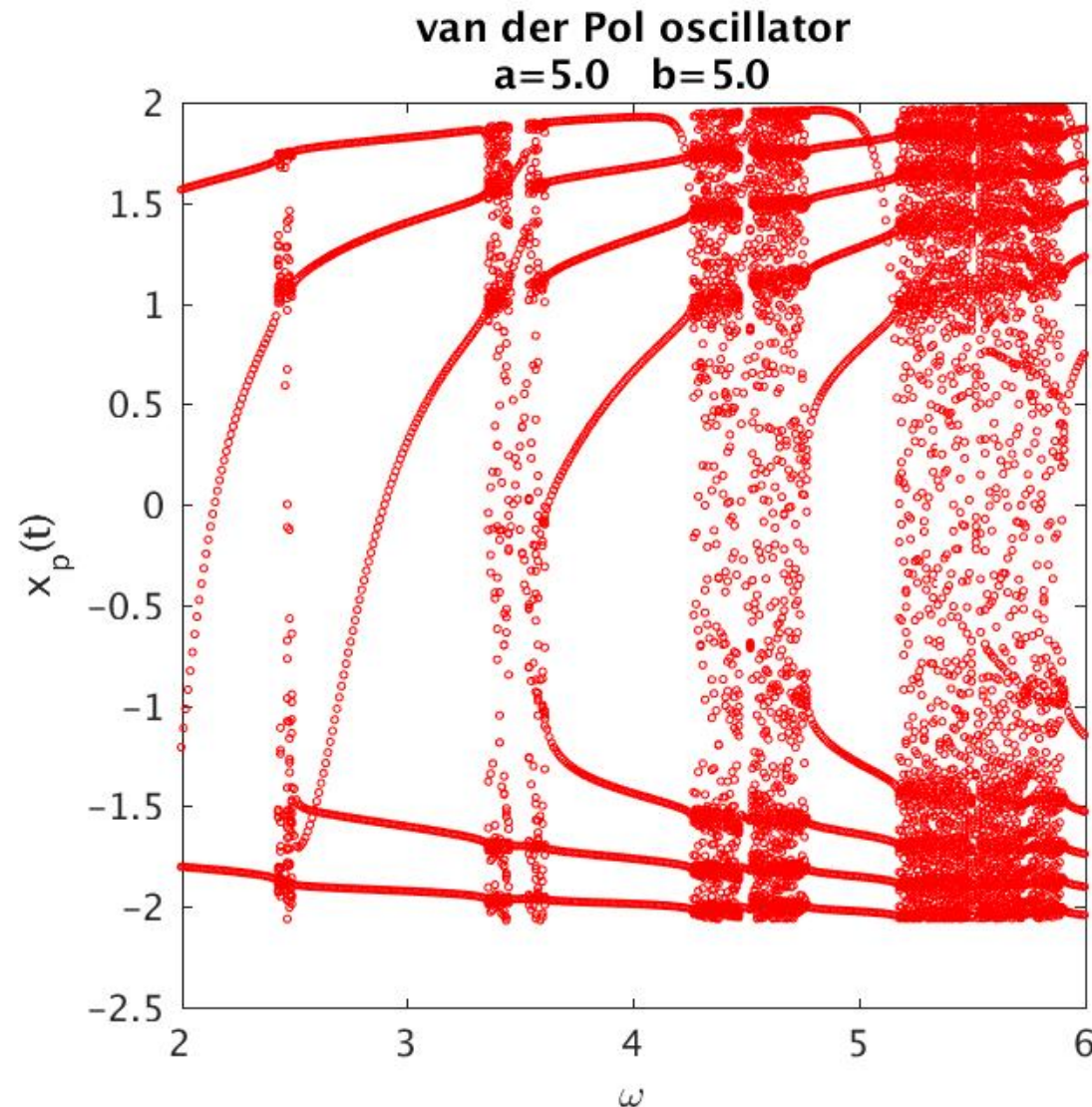
- Poincaré section for driven oscillator showing almost periodic trajectory. Output times are $2\pi n/\omega$, $n = 0, \dots, 1000$.



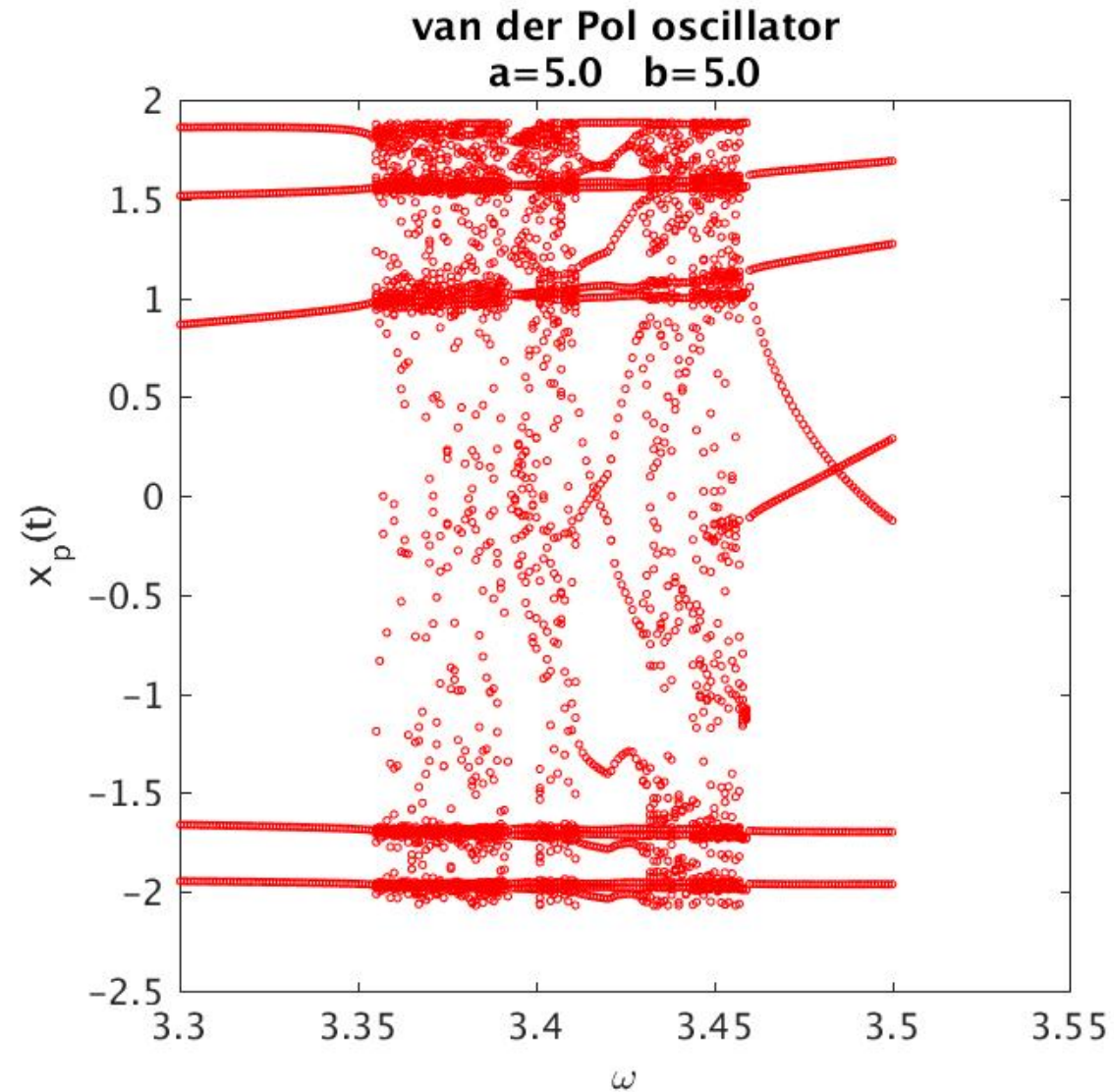
- Poincaré section for driven oscillator showing chaotic trajectory. Output times are $2\pi n/\omega$, $n = 0, \dots, 1000$.



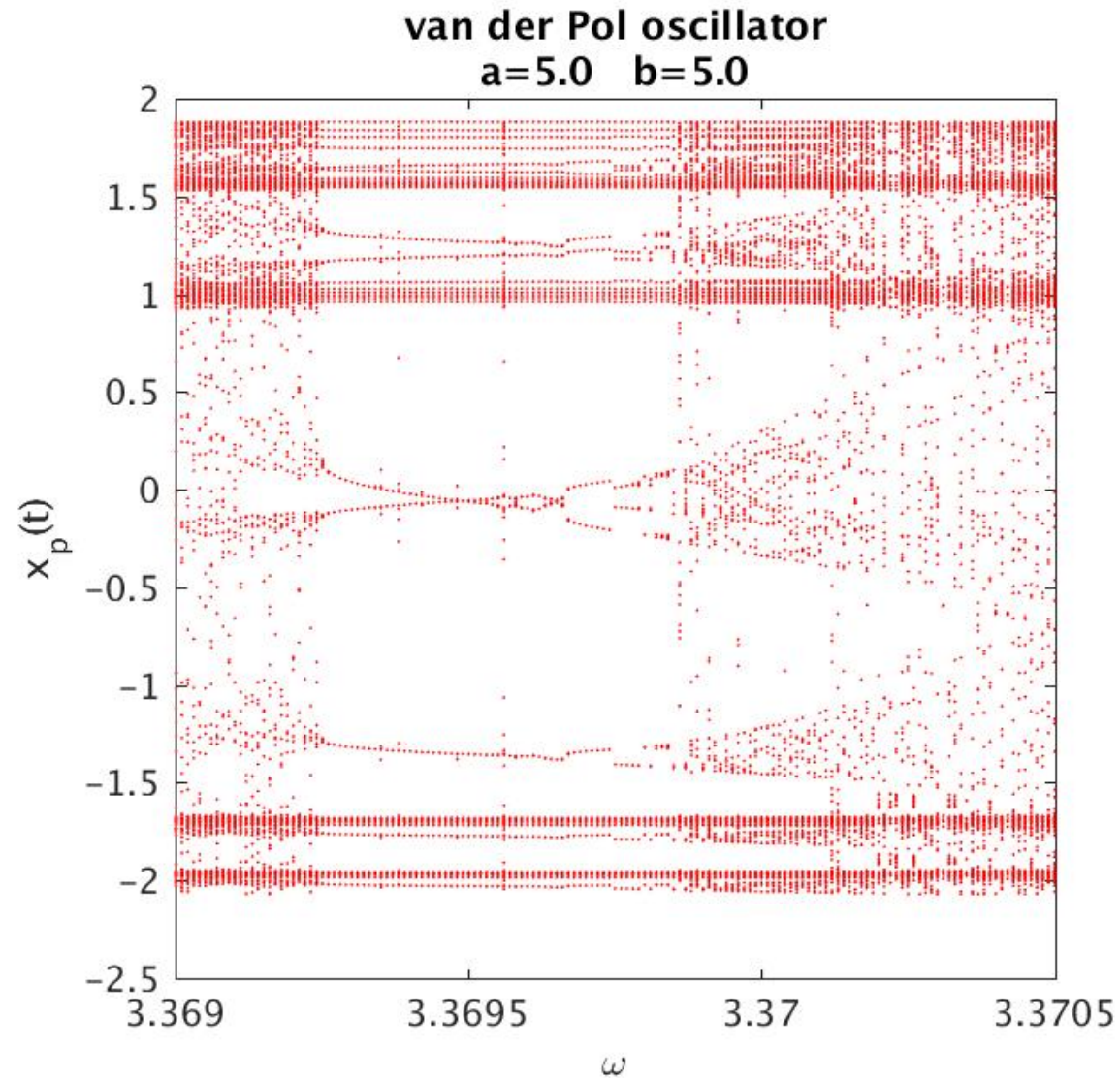
- Bifurcation diagram (1 of 4). Driving frequency ω is the control parameter. Each vertical slice in the plot represents a single evolution with output times $t_p = 2\pi n/\omega$, $n = 1500, \dots, 2000$, and the $x_p(t)$ are the oscillator displacements at those times. Regions where the motion is periodic as well as chaotic are clearly seen.



- Bifurcation diagram (2 of 4). Zoomed-in view of previous diagram.



- Bifurcation diagram (3 of 4). Zoomed-in view of previous diagram.



- Bifurcation diagram (4 of 4). Zoomed-in view of previous diagram. Central region (vertically) shows “classic” period doubling to complete chaotic behaviour

