Math 337 Homework 05

Andy Reagan

February 22, 2014

1. In the three examples below, write the given higher-oder ODE(s) as a system of first-order ODEs.

   (a) The equation for the charge on the capacitor in an electric circuit:

   $$L\frac{\mathrm{d}^2Q}{\mathrm{d}t^2} + R\frac{\mathrm{d}^2Q}{\mathrm{d}t^2} + CQ = \epsilon;$$

   here Q is the charge, and L, R and C are the inductange, resistance, and capacitance of the circuit.

   (b) The equation for the deflection of a loaded beam from the horizontal axis:

   $$E\frac{y''}{(1+(y')^2)^{3/2}} = M(x);$$

   here E is the beam's elacticity modeulus, Y is the deflection, and M is the bending moment.

   (c) The equations of the Kepler two-body problem (5.30):

   $$q'' = -\frac{q}{(q^2+r^2)^{3/2}} \quad , \qquad r'' = -\frac{r}{(q^2+r^2)^{3/2}};$$

   where q and r are the cartesian coordinates of a certain radius vector relative to the center of mass of the bodies.

Solution:

(a) Set $S = \mathrm{d}Q/\mathrm{d}t$ and therefore we have the system of equations

$$L\frac{\mathrm{d}S}{\mathrm{d}t} + R\frac{\mathrm{d}S}{\mathrm{d}t} + CQ = \epsilon \tag{1}$$

$$S = \frac{\mathrm{d}Q}{\mathrm{d}t}. \tag{2}$$

This can be written

$$\frac{\mathrm{d}S}{\mathrm{d}t} = \frac{\epsilon - CQ}{L+R} \tag{3}$$

$$\frac{\mathrm{d}Q}{\mathrm{d}t} = S. \tag{4}$$

(b) Set $y' = z$, such that $z' = y''$. We then have the system of ODEs

$$\begin{aligned} y' &= z \\ z' &= \frac{M(x)}{E}(1+z^2)^{3/2} \end{aligned}$$

(c) Set $q' = z$ and $r' = w$. Then we have the following system of four ODE's:

$$\begin{aligned} z' &= -\frac{q}{(q^2+r^2)^{3/2}} \\ w' &= -\frac{r}{(q^2+r^2)^{3/2}} \\ q' &= z \\ r' &= w. \end{aligned}$$

2. Write down the explicit (i.e., component by component) equations for the (a) Midpoint method and (b) the cRK method for a system of two ODEs $\vec{\mathbf{y}}' = \vec{\mathbf{f}}(x, \vec{\mathbf{y}})$.

Solution: (a) I extend the Midpoint method in a straightforward way:

$$\begin{aligned}
Y_{n+1}^{(1)} &= Y_n^{(1)} + hf^{(1)}\left(x_n, \left\{Y_n^{(1)} + hf^{(1)}\left(x_n, \left\{Y_n^{(1)}, Y_n^{(2)}\right\}\right), Y_n^{(2)} + hf^{(2)}\left(x_n, \left\{Y_n^{(1)}, Y_n^{(2)}\right\}\right\}\right)\right) \\
Y_{n+1}^{(2)} &= Y_n^{(2)} + hf^{(2)}\left(x_n, \left\{Y_n^{(1)} + hf^{(1)}\left(x_n, \left\{Y_n^{(1)}, Y_n^{(2)}\right\}\right), Y_n^{(2)} + hf^{(2)}\left(x_n, \left\{Y_n^{(1)}, Y_n^{(2)}\right\}\right\}\right)\right)
\end{aligned}$$

(b) The cRK coefficients become

$$\begin{aligned}
k_1^{(1)} &= hf^{(1)}\left(x_n, \left\{Y_n^{(1)}, Y_n^{(2)}\right\}\right) \\
k_1^{(2)} &= hf^{(2)}\left(x_n, \left\{Y_n^{(1)}, Y_n^{(2)}\right\}\right) \\
k_2^{(1)} &= hf^{(1)}\left(x_n + h/2, \left\{Y_n^{(1)} + k_1^{(1)}/2, Y_n^{(2)} + k_1^{(2)}/2\right\}\right) \\
k_2^{(2)} &= hf^{(2)}\left(x_n + h/2, \left\{Y_n^{(1)} + k_1^{(1)}/2, Y_n^{(2)} + k_1^{(2)}/2\right\}\right) \\
k_3^{(1)} &= hf^{(1)}\left(x_n + h/2, \left\{Y_n^{(1)} + k_2^{(1)}/2, Y_n^{(2)} + k_2^{(2)}/2\right\}\right) \\
k_3^{(2)} &= hf^{(2)}\left(x_n + h/2, \left\{Y_n^{(1)} + k_2^{(1)}/2, Y_n^{(2)} + k_2^{(2)}/2\right\}\right) \\
k_4^{(1)} &= hf^{(1)}\left(x_n + h, \left\{Y_n^{(1)} + k_3^{(1)}, Y_n^{(2)} + k_3^{(2)}\right\}\right) \\
k_4^{(2)} &= hf^{(2)}\left(x_n + h, \left\{Y_n^{(1)} + k_3^{(1)}, Y_n^{(2)} + k_3^{(2)}\right\}\right).
\end{aligned}$$

And the next step is

$$\begin{aligned}
Y_{n+1}^{(1)} &= Y_n^{(1)} + \frac{1}{6}\left(k_1^{(1)} + 2k_2^{(1)} + 2k_3^{(1)} + k_4^{(1)}\right) \\
Y_{n+1}^{(2)} &= Y_n^{(2)} + \frac{1}{6}\left(k_1^{(2)} + 2k_2^{(2)} + 2k_3^{(2)} + k_4^{(2)}\right)
\end{aligned}$$

3. (a) Show that the local truncation error of the simple-central-difference method (5.12) equals

$$\frac{h^4}{12}\frac{\mathrm{d}^4 y(x_n)}{\mathrm{d}x^4} + O(h^5).$$

Note that in the calculation of the local truncation error at the (n+1)st node that error at all previous nodes must be set equal to zero.

(b) Show that the local truncation error of Numerov's method (5.18) equals

$$-\frac{h^6}{240}\frac{\mathrm{d}^6 y(x_n)}{\mathrm{d}x^6} + O(h^7).$$

Hint: Note that $f_{n\pm1} = f(x_n \pm h, y(x_n \pm h)) \equiv f[x_n \pm h]$, where $f[x] \equiv f(x, y(x))$. Now use the Taylor expansion for $f_{n\pm1}$ written in the above form. Now recall that $f = y''$. What can you then say about $\mathrm{d}f/\mathrm{d}x$, etc?

Solution:

(a) The simple central difference formula is

$$Y_{n+1} - 2Y_n + Y_{n-1} = h^2 f_n.$$

Since we're solving for the error, we consider Y_n to be exact, hence $Y_n = y_n$. Therefore we use Taylor expansions for the above terms as follows (bringing the $h^2 f_n$ to the LHS):

$$\begin{aligned} Y_{n+1}: &\quad y_n + hy_n' + \frac{h^2}{2}y_n'' + \frac{h^3}{6}y_n''' + \frac{h^4}{24}y_n^{(4)} + O(h^5) \\ -2Y_n: &\quad -2y_n \\ Y_{n-1}: &\quad y_n - hy_n' + \frac{h^2}{2}y_n'' - \frac{h^3}{6}y_n''' + \frac{h^4}{24}y_n^{(4)} + O(h^5) \\ -h^2 f_n: &\quad -h^2 y_n'' \end{aligned}$$

Inspecting the above terms, we notice that when we add them, all of the y_n, hy_n, h^2y_n and h^3y_n terms cancel. Therefore, we have the local truncation error as the remaining terms:

$$\frac{h^4}{12}y_n^{(4)} + O(h^5).$$

(b) Numerov's formula is given similarly by

$$Y_{n+1} - 2Y_n + Y_{n-1} = \frac{h^2}{12}\left(f_{n+1} + 10f_n + f_{n-1}\right).$$

Bringing everything to the LHS, we have

$$Y_{n+1} - 2Y_n + Y_{n-1} - \frac{h^2}{12}f_{n+1} - \frac{5h^2}{6}f_n - \frac{h^2}{12}f_{n-1} = 0.$$

Since we're solving for the error, we consider Y_n to be exact, hence $Y_n = y_n$. Therefore we use Taylor expansions for the above terms as follows (bringing the $h^2 f_n$ to the LHS):

$$\begin{aligned}
Y_{n+1}: &\quad y_n + hy_n' + \frac{h^2}{2}y_n'' + \frac{h^3}{6}y_n''' + \frac{h^4}{24}y_n^{(4)} + \frac{h^5}{120}y_n^{(5)} + \frac{h^6}{720}y_n^{(6)} + O(h^7) \\
-2Y_n: &\quad -2y_n \\
Y_{n-1}: &\quad y_n - hy_n' + \frac{h^2}{2}y_n'' - \frac{h^3}{6}y_n''' + \frac{h^4}{24}y_n^{(4)} - \frac{h^5}{120}y_n^{(5)} + \frac{h^6}{720}y_n^{(6)} + O(h^7) \\
-\frac{h^2}{12}f_{n+1}: &\quad -\frac{h^2}{12}y_{n+1}'' = -\frac{h^2}{12}y_n'' - \frac{h^3}{12}y_n''' - \frac{h^4}{24}y_n^{(4)} - \frac{h^5}{72}y_n^{(5)} - \frac{h^6}{288}y_n^{(6)} + O(h^7) \\
-\frac{5h^2}{6}f_n: &\quad -\frac{5h^2}{6}y_n'' \\
-\frac{h^2}{12}f_{n-1}: &\quad -\frac{h^2}{12}y_{n-1}'' = -\frac{h^2}{12}y_n'' + \frac{h^3}{12}y_n''' - \frac{h^4}{24}y_n^{(4)} + \frac{h^5}{72}y_n^{(5)} - \frac{h^6}{288}y_n^{(6)} + O(h^7)
\end{aligned}$$

Since there are more terms than before, I group them by orders of h:

$$\begin{aligned}
O(h^0): &\quad y_n - 2y_n + y_n = 0. \\
O(h^1): &\quad hy_n' - hy_n' = 0. \\
O(h^2): &\quad \frac{h^2}{2}y_n'' + \frac{h^2}{2}y_n'' - \frac{h^2}{12}y_n'' - \frac{5h^2}{6}y_n'' - \frac{h^2}{12}y_n'' = 0. \\
O(h^3): &\quad \frac{h^3}{6}y_n''' - \frac{h^3}{6}y_n''' - \frac{h^3}{12}y_n''' + \frac{h^3}{12}y_n''' = 0. \\
O(h^4): &\quad \frac{h^4}{24}y_n^{(4)} + \frac{h^4}{24}y_n^{(4)} - \frac{h^4}{24}y_n^{(4)} - \frac{h^4}{24}y_n^{(4)} = 0. \\
O(h^5): &\quad \frac{h^5}{120}y_n^{(5)} - \frac{h^5}{120}y_n^{(5)} - \frac{h^5}{72}y_n^{(5)} + \frac{h^5}{72}y_n^{(5)} = 0 \\
O(h^6): &\quad \frac{h^6}{720}y_n^{(6)} + \frac{h^6}{720}y_n^{(6)} - \frac{h^6}{288}y_n^{(6)} - \frac{h^6}{288}y_n^{(6)} = -\frac{h^6}{240}y_n^{(6)}
\end{aligned}$$

Therefore, we have the local truncation error as the remaining terms:

$$\frac{h^6}{40}y_n^{(6)} + O(h^7).$$

4. Show that the Verlet method (5.27) is a second-order method. Follow the steps of a similar derivation for the Modified Euler method, presented in Sec. 5.1.

   Solution: The Verlet method is given by

$$\begin{aligned} Y_{n+1} &= Y_n + hV_n + \frac{h^2}{2} f(Y_n) \\ V_{n+1} &= V_n + \frac{h}{2}\left(f(Y_n) + f(Y_{n+1})\right) \end{aligned}$$

   Expanding the Taylor series of the numerical method we have

$$\begin{aligned} Y_{n+1} &= Y_n + hY_n' + \frac{h^2}{2} Y_n'' \\ V_{n+1} &= V_n + \frac{h}{2}\left(V_n' + f(Y_n + hY_n' + \frac{h^2}{2} Y_n'')\right) \\ &= V_n + \frac{h}{2}\left(V_n' + Y_n'' + hY_n''' + \frac{h^2}{2} Y_n^{(4)})\right) \\ &= V_n + \frac{h}{2}\left(V_n' + V_n' + hV_n'' + \frac{h^2}{2} V_n^{(3)})\right) \\ &= V_n + hV_n' + \frac{h^2}{2} V_n'' + \frac{h^3}{4} V_n^{(3)}) \end{aligned}$$

   making the assumption that $V_n = Y_n'$ at the previous time step, which is standard when computing the local error, because we assume $Y_n = y_n$ (hence $V_n = v_n$).

   The analytical solution for these is

$$\begin{aligned} y_{n+1} &= y_n + hy_n' + \frac{h^2}{2} y_n'' + O(h^3) \\ v_{n+1} &= v_n + hv_n' + \frac{h^2}{2} v_n'' + O(h^3) \end{aligned}$$

   Comparing the above analytical and numerical solutions, we see that in both Y and V the Verlet method has local truncation error $O(h^3)$ making the method globally second order ($O(h^2)$).

5. In the notes, we derived the Verlet method by first going over the half-step $[x_n, x_{n+1/2}]$ with method (5.4) and then going over the remaining half $[x_{n+1/2}, x_{n+1}]$ with method (5.2). Let us refer to the resulting Verlet method (5.27) as Verlet-1. Use the same ideas to derive a method by reversing the order of (5.3) and (5.4) (i.e. apply (5.3) over $[x_n, x_{n+1/2}]$ and then (5.4) over $[x_{n+1/2}, x_{n+1}]$). We will refer to this method as Verlet-2.

   Solution: Reversing the order of application of the symplectic Euler steps, we have

$$\begin{aligned}
x_n \quad \to \quad x_{n+1/2}, \text{using (5.3):} \quad & Y_{n+1/2} = Y_n + (h/2)V_n \\
& V_{n+1/2} = V_n + (h/2)f(Y_{n+1/2}) \\
x_{n+1/2} \quad \to \quad x_{n+1}, \text{using (5.4):} \quad & V_{n+1} = V_{n+1/2} + (h/2)f(Y_{n+1/2}) \\
& Y_{n+1} = Y_{n+1/2} + (h/2)V_{n+1}
\end{aligned}$$

   Adding these equations, we have

$$\begin{aligned}
V_{n+1} &= V_n + hf(Y_{n+1/2}) \\
&= V_n + hf(Y_n + (h/2)V_n) \\
Y_{n+1} &= Y_n + (h/2) \cdot (V_n + V_{n+1})
\end{aligned}$$

6. Consider the equations of a simple harmonic oscillator (5.25), i.e.

$$y'' = -y, \qquad y(0) = 0, \qquad y'(0) = 1.$$

   (a) Write out its analytical solution (consult any ODE or Physics-I textbook).
   Solve this equation numerically using the following methods:
   (b) Verlet-1 (or Verlet-2, the results will be similar)
   (c) Modified Euler (5.5)
   (d) cRK
   (e) simple central-difference (5.12) with (5.17)
   (f) Matlab's `ode45`
   In all cases, run the simulations until $t = 1000$. To have a fair comparison among the methods, use $h = 0.2$ for the second-order methods and $h = 0.5$ for the fourth-order ones. For the `ode45`, obtain *two* solutions with different values of the absolute tolerance: 0.002 and 0.003.
   Your ouput should contain:
   -phase plots of the numerical solutions (one per method);
   -plots of the error in the Hamiltonian vs time;
   -a table summarizing which of the methods nearly conserve the Hamiltonian and which methods do not.

Solution:

(a) The analytical soluton (from ODE textbook...and an obvious guess for $y'' = -y$):

$$y(t) = \sin(t).$$

The code that does all of the things:

```
methodcell = {'verlet1','ME','cRK','SCD','ode45␣tol␣.002','ode45␣tol␣.003'};
resultscell = cell(1,length(methodcell));
analcell = cell(1,length(methodcell));

t0 = 0; tmax = 1000;

y0 = 0; dy0 = 1;

% for second order methods
h2 = 0.2;
tvec2 = t0:h2:tmax;
% for fourth order methods
h4 = 0.5;
tvec4 = t0:h4:tmax;

% tolerances for ode45
absTol1 = 0.002;
absTol2 = 1.5*absTol1;

% solve with verlet-1
i = 1; h = h2; tvec = tvec2;
methodh = str2func(sprintf('andy_%s',methodcell{i}));
```

```
resultscell{i} = methodh(@andy_SHO_1,tvec,[y0;dy0],h,[]);
analcell{i} = [sin(tvec);cos(tvec)]; % analytical

% solve with modified Euler
i = 2; h = h2; tvec = tvec2;
methodh = str2func(sprintf('andy_%s',methodcell{i}));
resultscell{i} = methodh(@andy_SHO_2,tvec,[y0;dy0],h,[]);
analcell{i} = [sin(tvec);cos(tvec)]; % analytical

% solve with cRK
i = 3; h = h4; tvec = tvec4;
methodh = str2func(sprintf('andy_%s',methodcell{i}));
resultscell{i} = methodh(@andy_SHO_2,tvec,[y0;dy0],h,[]);
analcell{i} = [sin(tvec);cos(tvec)]; % analytical

% solve with SCD
i = 4; tvec = tvec2; h = h2;
methodh = str2func(sprintf('andy_%s',methodcell{i}));
soln = zeros(2,length(tvec));
soln(1,:) = methodh(@andy_SHO_1,tvec,[y0;dy0],h2,[]);
% compute v
soln(2,2:end-1) = (soln(1,3:end)-soln(1,1:end-2))./(2*h);
soln(2,1) = dy0;
soln(2,end) = (soln(1,end)-soln(1,end-1))/h+(h/2)*(-soln(1,end));
resultscell{i} = soln;
% analytical
analcell{i} = [sin(tvec);cos(tvec)];

% solve with ode45
i = 5;
% methodh = str2func(sprintf('%s',methodcell{i}));
for abstol=[0.002,0.003]
    options = odeset('AbsTol',abstol);
    % [tvec,soln] = methodh(@andy_SHO_2,[t0,tmax],[y0;dy0],options);
    [tvec,soln] = ode45(@andy_SHO_2,[t0,tmax],[y0;dy0],options);
    resultscell{i} = soln';
    % analytical
    analcell{i} = [sin(tvec);cos(tvec)];
    i=i+1;
end

% plot all
for i=1:length(methodcell)
    soln = resultscell{i};
    anal = analcell{i};
    figure;
    subplot(121); % phase plot
    plot(anal(1,:),anal(2,:),'r')
    hold on;
    plot(soln(1,:),soln(2,:),'b')
    legend({'analytical',methodcell{i}});
    xlabel('y','FontSize',20)
    ylabel('v','FontSize',20)
    subplot(122); % error in hamiltonian
    plot(1:length(soln(1,:)),abs(1/2.*(anal(1,:).^2+anal(2,:).^2)-1/2.*(soln(1,:)
        .^2+soln(2,:).^2)));
    % xlim([0,1000])
    xlabel('t','FontSize',20)
    ylabel('Hamiltonian␣error','FontSize',20)
    set(gcf, 'units', 'inches', 'position', [1 1 10 4])
    set(gcf,'PaperPositionMode','auto')
    print('-depsc2','-zbuffer','-r200',sprintf('andy_hw05_prb06_%02g.eps',i))
end

fprintf('now␣run\n')
fprintf('echo␣$(ls␣andy_hw05_prb06_0{1..6}*)␣|␣xargs␣-n␣1␣epstopdf\n')
```

(b) Verlet-1

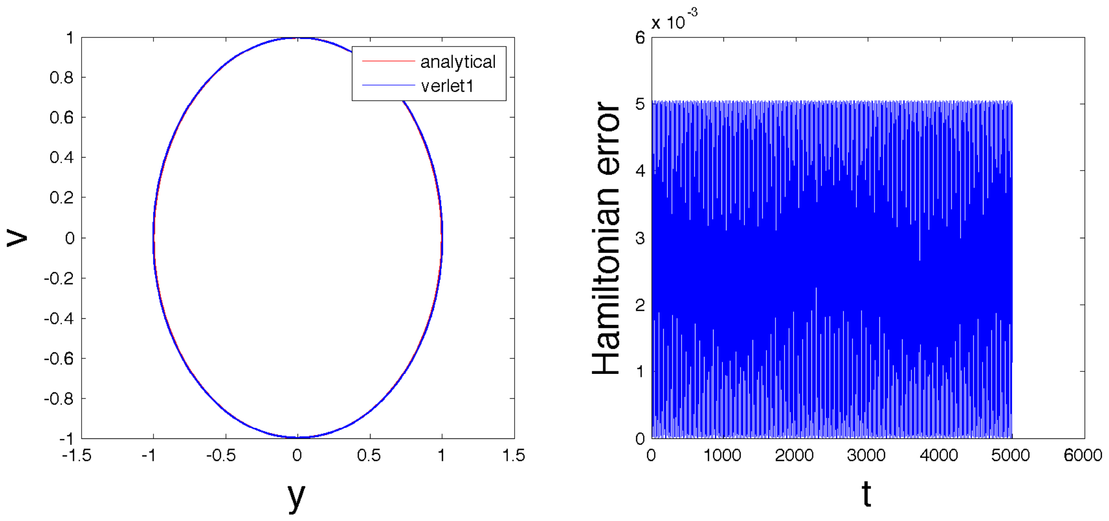

Figure 1: Harmonic oscillator solved by Verlet method.

(c) Modified Euler (5.5)

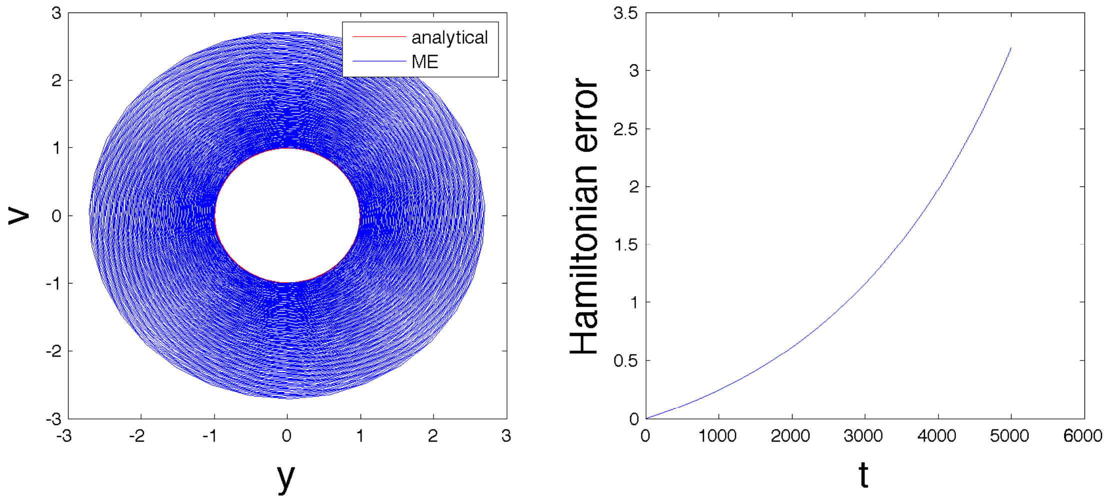

Figure 2: Harmonic oscillator solved by Modified Euler method.

(d) cRK

(e) simple central-difference (5.12) with (5.17)

(f) Matlab's `ode45`

Table summarizing which of the methods nearly conserve the Hamiltonian and which methods do not:

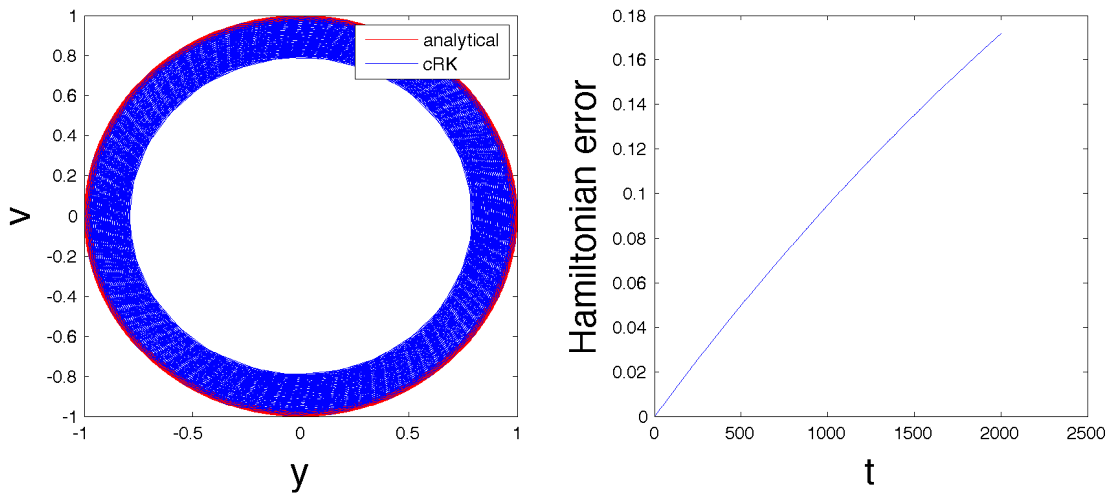

Figure 3: Harmonic oscillator solved by cRK method.

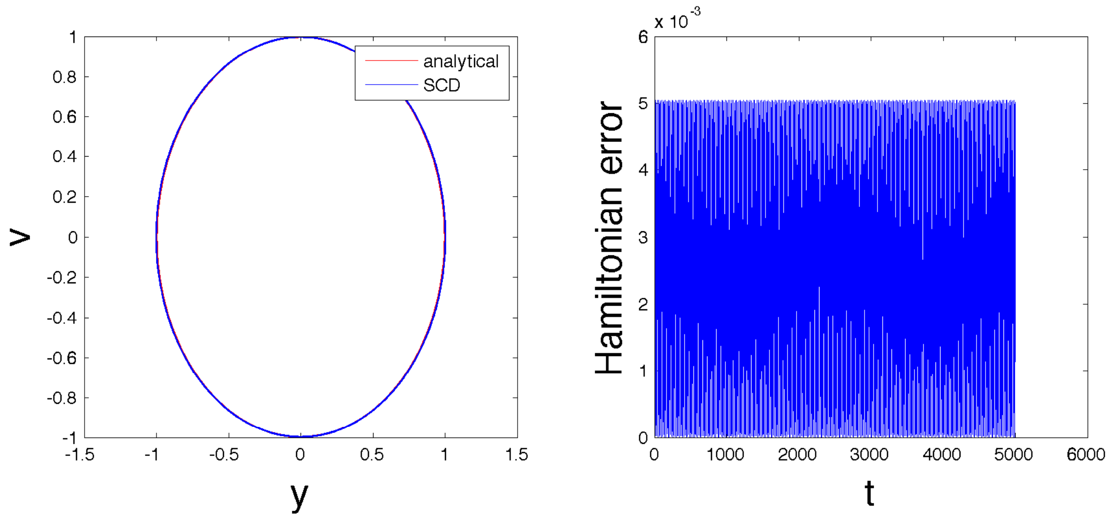

Figure 4: Harmonic oscillator solved by simple-central-difference method.

method which DO	methods which DO NOT
Verlet method	Modified Euler
simple-central-difference	cRK
`ode45` with tolerance .002	`ode45` with tolerance .003

Table 1: Table summarizing which of the methods nearly conserve the Hamiltonian and which methods do not.

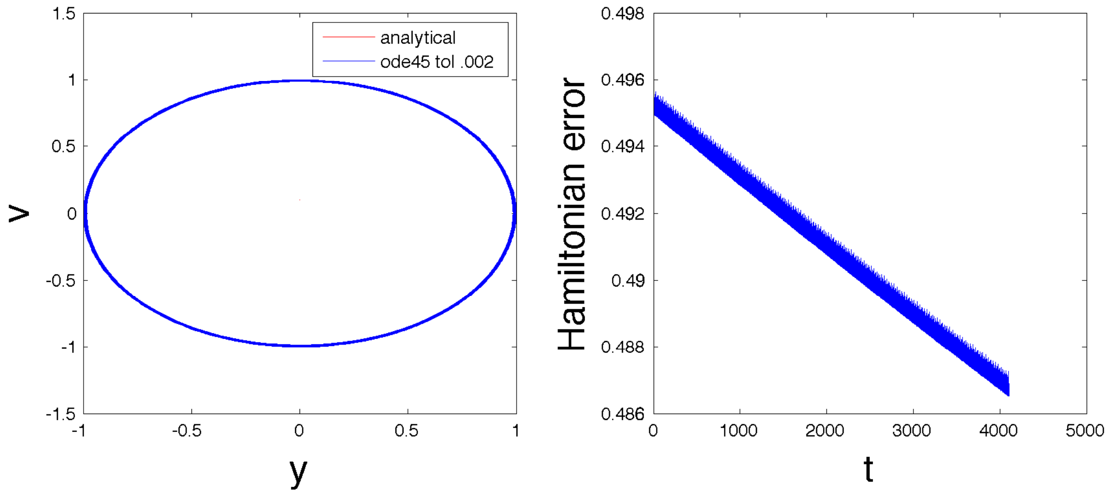

Figure 5: Harmonic oscillator solved by ode45 with tolerance .002.

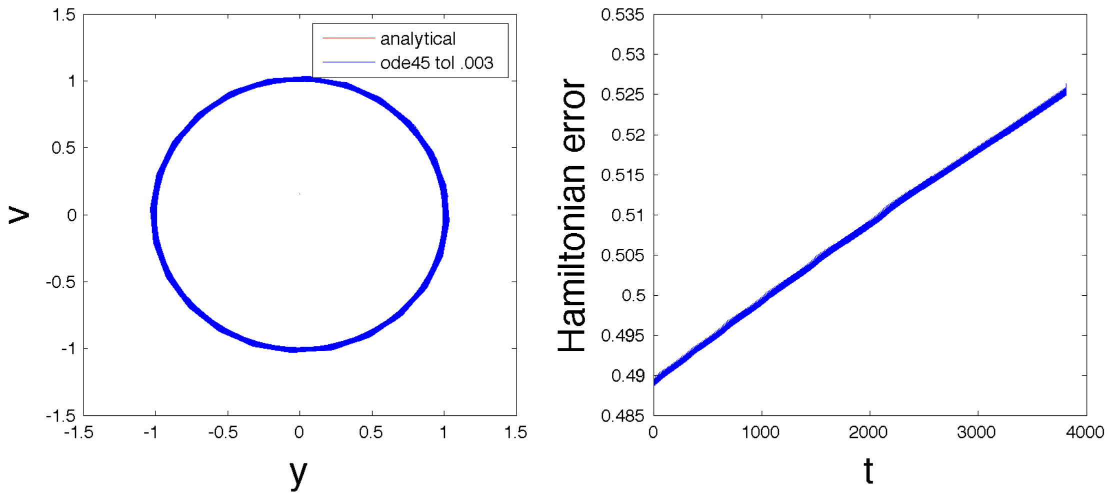

Figure 6: Harmonic oscillator solved by ode45 with tolerance .003.

7. Find an *estimate* for the growth of the Hamiltonian, $H = (1/2)(v^2 + y^2)$, of the numerical solution of the harmonic oscillator model obtained with the regular Euler method. Follow the steps described below.

   (a) For concreteness, consider the IVP of Problem 6, for which you obtained the analytical solution. Use the reult of Problem 6(a) to find the exact value of the Hamiltonian.

   (b) Write down the formulae for the corresponding numerical solutions obtained with the regular Euler method. These formulae are described, but not explicitly stated, in the paragraph (including the footnote) after (5.54). Then, with these solutions, calculate an estimate for the numerically computed Hamiltonian.

   (c) Finally, verify that your answer for $H_{\text{computed}} - H_{\text{exact}}$ agrees with that found from a figure in Sec. 5.3 (for the largest value of x).

Solution:

(a) From problem 6(a), we have the solution $y(t) = \sin(t)$. Thus we have $v(t) = \cos(t)$ and can compute the Hamiltonian as

$$H = 1/2(\sin^2 t + \cos^2 t) = 1/2.$$

(b) With the ODE written as a system, we have $y' = v$ and $v' = -y$. Regular Euler computes

$$\begin{aligned} Y_{i+1} &= Y_i + hV_i \\ V_{i+1} &= V_i - hY_i \end{aligned}$$

Creativity being discouraged, I find the following to be a very straight forward approach. Thus, explicitly computing that $H_0 = 1/2 \cdot (1^2 + 0^2) = 1/2$, and setting $x_0 = 0$:

$$\begin{aligned} H_{n+1} &= \frac{1}{2}\left(Y_{n+1}^2 + V_{n+1}^2\right) \\ &= \frac{1}{2}\left((Y_n + hV_n)^2 + (V_n - hY_n)^2\right) \\ &= \frac{1}{2}\left((1+h^2)Y_n^2 + (1+h^2)V_n^2\right) \\ &= \frac{1}{2}(1+h^2)\left(Y_n^2 + V_n^2\right) \\ &= \frac{1}{2}(1+h^2)2H_n \\ &= (1+h^2)H_n \\ &= (1+h^2)^n H_0 \\ &= \frac{1}{2}(1+h^2)^{(x-x_0)/h} \\ &= \frac{1}{2}e^{xh} \end{aligned}$$

(c) The figure in Sec. 5.3 has $H_{\text{computed}} - H_{\text{exact}} \approx 0.25$ for $x = 20$. Assuming a step size of 0.02 as in the notes, we have

$$\begin{aligned} H_{\text{computed}} &= \frac{1}{2} e^{20 \cdot .02} \\ &= 0.74 \end{aligned}$$

which agress with the error of about 0.25. This is verified in MATLAB with
`h = .02; 0.5*(h^2+1)^(20/h)`.

8. In the notes, we showed that the symplectic Euler methods (5.3) and (5.4) approximate very well the energy of a system without friction-like forces. Now, investigate the question of whether these methods still produce accurate approximations to the energy when a small friction *is* present.

   Consider a slightly damped harmonic oscillator

$$y'' = -2\gamma y' - \omega_0^2 y, \qquad y(0) = 0, \qquad y'(0) = 1.$$

   Its exact solution is $y = \frac{1}{\omega} e^{-\gamma t} \sin(\omega t)$, where $\omega = \sqrt{\omega_0^2 - \gamma^2}$. Note that the energy of a damped oscillator is defined by the same expression as in the undamped case:

$$E = \frac{1}{2}\left((y')^2 + \omega_0^2 y^2\right).$$

   One can obtain the using exact solution

$$E_{\text{exact}} = \frac{e^{-2\gamma t}}{2\omega^2}\left(\omega_0^2 - \gamma^2\cos(2\omega t) - \omega\gamma\sin(2\omega t)\right) \quad \approx \quad \frac{1}{2}e^{-2\gamma t}.$$

   Use the regular Euler and one of the two symplectic Euler methods to simulate the damped oscillator with $\omega_0 = 1$ and $\gamma = 0.015$ up to $t = 50$. For each of the methods, use three different value of step size: $h \in [0.01, 0.03, 0.05]$. For each of the above values of h, you need to present three plots. Plot 1 should contian the phase plane plots of the exact solution and the solution obtained by the regular Euler method. Plot 2 should contain the phase plane plots of the exact solution and the solution obtained by the symplectic Euler method. Plot 3 should contain the the error in the total energy for both methods.

   Using an analysis similar to that you did in Problem 7, *explain quantitatively* why the regular Euler solution behaves, relative to the exact solution, in the observed way. That is, give a quantitative estimate of the error in the oscillator's total energy for each given h and compare it with the value you obtain from you code. *Also*, you must explain why the regular Euler solution changes from decaying to growing as h is varied.

   Conclude whether is it reasonable to use symplectic methods for problems with *small* friction (or any other source of damping). Is it a good idea to use general-purpose methods (like the regular Euler) for the same task?

   Solution: The code and plots follow this discussion.

   I derive why regular Euler solution behaves the way it does, and how that depends on h. With the analytical solution given as in the problem statement, we compute the derivative for the analytical v and we have

$$y = \frac{1}{\omega}e^{-\gamma t}\sin(\omega t) \qquad \Rightarrow y' = v = \frac{-\gamma}{\omega}e^{-\gamma t}\sin(\omega t) + e^{-\gamma t}\cos(\omega t).$$

   From the notes, we know that the regular Euler solution behaves as $ye^{th/2}$ in y and $ve^{th/2}$ in v. Therefore, we compute the numerical total energy as (noting that $\omega_0/\omega \approx 1$ since γ is small):

$$
\begin{aligned}
E_{\text{computed}} &= \frac{1}{2}\left((ye^{th/2})^2\omega_0^2 + (ve^{th/2})^2\right) \\
&= \frac{1}{2}\left((\frac{1}{\omega}e^{(h/2-\gamma)t}\sin(\omega t))^2\omega_0^2 + (\frac{-\gamma}{\omega}e^{(h/2-\gamma)t}\sin(\omega t) + e^{(h/2-\gamma)t}\cos(\omega t))^2\right) \\
&\simeq \frac{1}{2}\left(e^{(h-2\gamma)t}\sin^2(\omega t) - \frac{\gamma^2}{\omega^2}e^{(h-2\gamma)t}\sin^2(\omega t) + e^{(h-2\gamma)t}\cos^2(\omega t) - \frac{2\gamma}{\omega}e^{(h-2\gamma)t}\sin^2(\omega t)\cos^2(\omega t)\right) \\
&= \frac{1}{2}e^{(h-2\gamma)t}\left(\sin^2(\omega t) - \frac{\gamma^2}{\omega^2}\sin^2(\omega t) + \cos^2(\omega t) - \frac{2\gamma}{\omega}\sin^2(\omega t)\cos^2(\omega t)\right) \\
&\approx \frac{1}{2}e^{(h-2\gamma)t}
\end{aligned}
$$

It can be seen from the problem statement that the exact solution's total energy decays as t increases. However, we observe that the numerical solution from simple Euler decays is $h < 2\gamma$, is approximately constant for $h = 2\gamma$, and grows for $h > 2\gamma$. Since we set $\gamma = 0.015$, we have $2\gamma = 0.03$. So for $h = 0.01$, the computed energy decays, for $h = 0.03$ the computed energy is constant, and for $h = 0.05$ the computed energy grows. This is exactly what we observe.

It is reasonable to use symplectic methods for small friction. However, the general purpose methods are not appropriate for this task.

```
methodcell = {'SE','symE3'};
resultscell = cell(1,length(methodcell));
analcell = cell(1,length(methodcell));
colorcell = {'r','b'};

t0 = 0; tmax = 50;
y0 = 0; dy0 = 1;

gamma = 0.015;
w0 = 1;
params = {gamma,w0}; % gamma, omega_0

for h=.01:.02:.05
    tvec = t0:h:tmax;
    figure;
    for i=1:2
        methodh = str2func(sprintf('andy_%s',methodcell{i}));
        resultscell{i} = methodh(@andy_SHO_damped,tvec,[y0;dy0],h,params);
        analcell{i} = [(1/w0)*exp(-gamma.*tvec).*sin(w0.*tvec);...
            exp(-gamma.*tvec).*cos(w0.*tvec)-...
            (gamma/w0)*exp(-gamma.*tvec).*sin(w0.*tvec)]; % analytical
        subplot(130+i); % phase plot of SE
        soln = resultscell{i};
        anal = analcell{i};
        plot(anal(1,:),anal(2,:),'r')
        hold on;
        plot(soln(1,:),soln(2,:),'b')
        legend({'analytical',methodcell{i}});
        xlabel('y','FontSize',20)
        ylabel('v','FontSize',20)
    end
    subplot(133); % error in total energy of each
    energy_anal = (exp(-2*gamma.*tvec))./(2*w0^2).*(w0^2-gamma^2.*cos(2*w0.*tvec)-w0*
        gamma.*sin(2*w0.*tvec));
    for i=1:2
        soln = resultscell{i};
        energy_soln = 0.5.*(soln(2,:).^2+w0^2.*soln(1,:).^2);
        plot(tvec,abs(energy_soln-energy_anal),colorcell{i});
        hold on;
    end
    legend(methodcell);
    xlabel('t','FontSize',20)
    ylabel('Total␣Energy␣error','FontSize',20)
    set(gcf, 'units', 'inches', 'position', [1 1 10 4])
    set(gcf,'PaperPositionMode','auto')
    print('-depsc2','-zbuffer','-r200',sprintf('andy_hw05_prb08_%0.02d.eps',100*h))
end
```

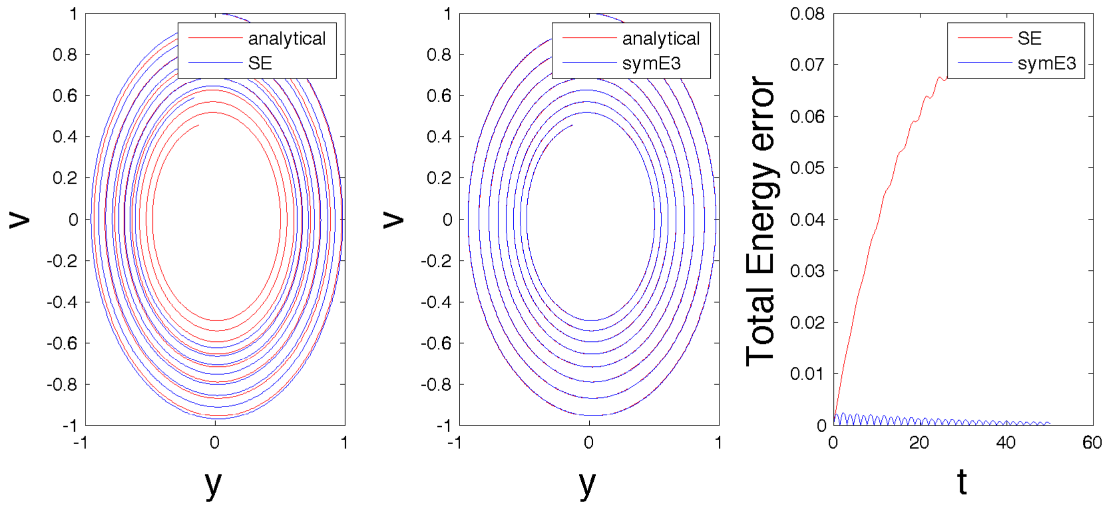

Figure 7: The simple Euler (SE) and symplectic Euler (symE3) solutions for a damped oscillator, along with the error in the total energy. The time step h is set to 0.01.

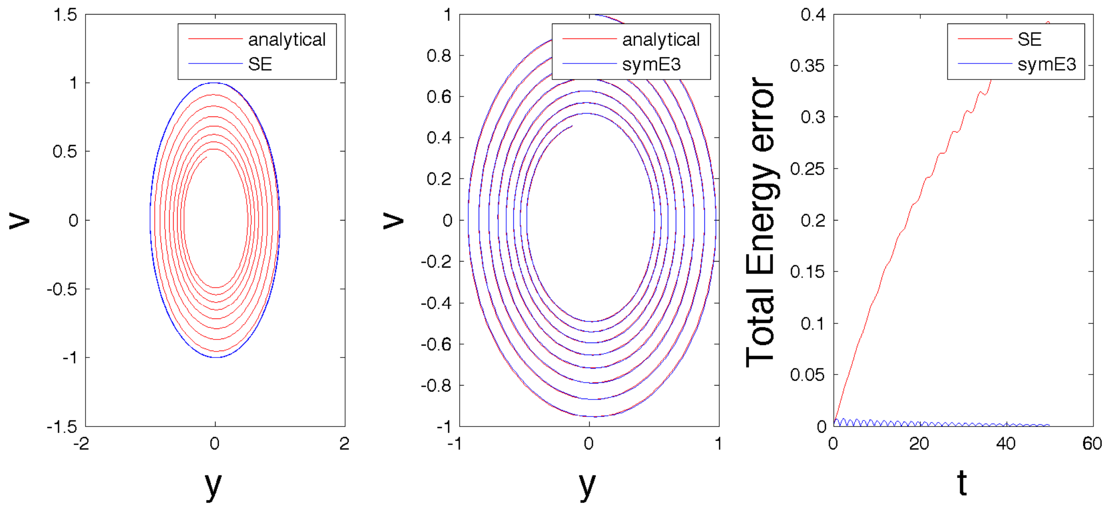

Figure 8: The simple Euler (SE) and symplectic Euler (symE3) solutions for a damped oscillator, along with the error in the total energy. The time step h is set to 0.03.

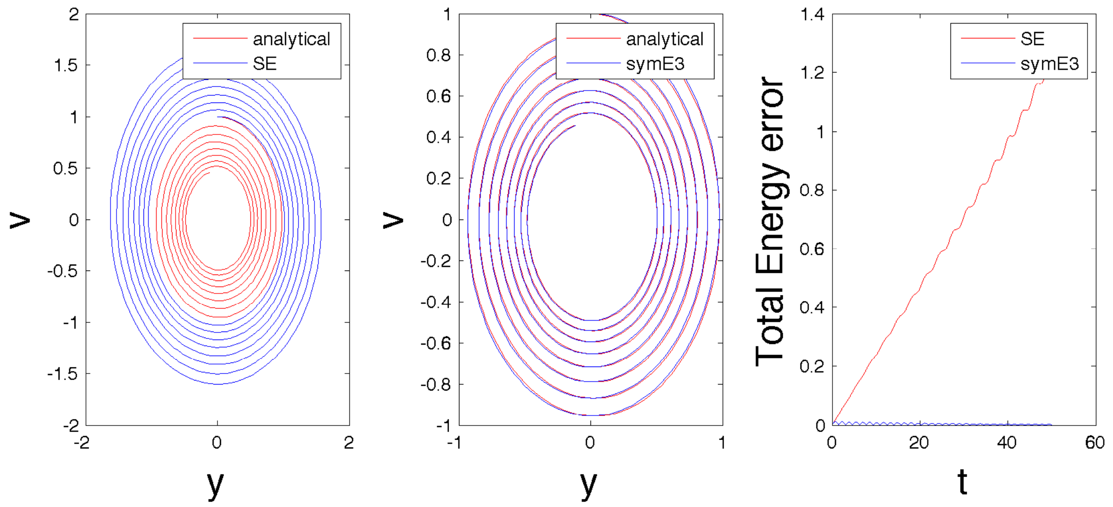

Figure 9: The simple Euler (SE) and symplectic Euler (symE3) solutions for a damped oscillator, along with the error in the total energy. The time step h is set to 0.05.

9. Follow the lines of the stability analyses for the regular and symplectic Euler methods (see Sec 5.4) and obtain the stability regions for the Modified Euler (5.5) and Verlet Method (5.27). Use (5.25a) as the model problem for both methods, where you should assume that ω is complex (as you did earlier for λ).

   Strictly speaking, you should be able to give the answers for both methods without doing any calculations, because these answers can be found in Lectures 4 and 5. However, I would like you to do those calculations in order to get some practice with stability analysis of a system of ODEs.

   Solution: For the Modified Euler method, a general purpose method, we consider the 1-D case:
   $$y' = \lambda y, \qquad \text{with} \qquad \lambda = \pm i\omega.$$
   Plugging this into the Modified Euler method, we have
   $$Y_{n+1} = Y_n \left[1 + h\lambda + \frac{1}{2}(h\lambda)^2 \right]$$
   where $\lambda = \lambda_R + i\lambda_I$ as a complex number. For this method to be stable, we require that the magnitude of the bracketed RHS. This becomes
   $$\begin{aligned} &\left| 1 + h\lambda + \frac{1}{2}(h\lambda)^2 \right| \\ =&\left| 1 + h\lambda_R + ih\lambda_I + \frac{1}{2}(h\lambda_R + ih\lambda_I)^2 \right| \\ =&\sqrt{(h\lambda_I)^2 + \left(1 - \frac{1}{2}h^2\lambda_I^2\right)} \\ =&\sqrt{1 + \frac{1}{4}h^4} \\ \approx&1 + \frac{1}{8}h^4 \end{aligned}$$
   Since this last approximation is the error growth at each timestep, we see that for any h it is greater than 1. Therefore any error will grow exponentially and the Modified Euler method is *unstable* for any h.

   For the Verlet method:
   $$\begin{aligned} Y_{n+1} &= Y_n + hV_n + \frac{h^2}{2} f(Y_n) \\ V_{n+1} &= V_n + \frac{h}{2}\left(f(Y_n) + f(Y_{n+1}) \right) \end{aligned}$$
   we use the model problem
   $$\begin{pmatrix} \omega y \\ v \end{pmatrix}' = \begin{pmatrix} 0 & \omega \\ -\omega & 0 \end{pmatrix} \begin{pmatrix} \omega y \\ v \end{pmatrix}.$$

Applying this problem to the Verlet method, we have the folowing system:

$$\begin{pmatrix} 1 & 0 \\ \frac{\omega^2 h}{2} & 1 \end{pmatrix} \begin{pmatrix} \omega Y \\ V \end{pmatrix}'_{n+1} = \begin{pmatrix} 1-\frac{\omega^2 h^2}{2} & h \\ -\frac{h\omega^2}{2} & 1 \end{pmatrix} \begin{pmatrix} \omega Y \\ V \end{pmatrix}_n .$$

To solve this system, we multiply on the left by the inverse of the matrix on the LHS:

$$\begin{pmatrix} \omega Y \\ V \end{pmatrix}'_{n+1} = \begin{pmatrix} 1 & 0 \\ \frac{\omega^2 h}{2} & 1 \end{pmatrix}^{-1} \begin{pmatrix} 1-\frac{\omega^2 h^2}{2} & h \\ -\frac{h\omega^2}{2} & 1 \end{pmatrix} \begin{pmatrix} \omega Y \\ V \end{pmatrix}_n .$$

We have the inverse on the RHS as

$$\begin{pmatrix} 1 & 0 \\ \frac{\omega^2 h}{2} & 1 \end{pmatrix}^{-1} = \begin{pmatrix} 1 & 0 \\ -\frac{\omega^2 h}{2} & 1 \end{pmatrix} .$$

And so the system becomes

$$\begin{pmatrix} \omega Y \\ V \end{pmatrix}'_{n+1} = \begin{pmatrix} 1-\frac{\omega^2 h^2}{2} & h \\ \frac{\omega^4 h^3}{4} - \omega^2 h & 1-\frac{\omega^2 h^2}{2} \end{pmatrix} \begin{pmatrix} \omega Y \\ V \end{pmatrix}_n .$$

We solve for the Eigenvalues of the matrix on the RHS as

$$\begin{aligned} &\left(1-\frac{\omega^2 h^2}{2}-\lambda\right)\left(-\frac{\omega^2 h^2}{2}+1-\lambda\right)-(h)\left(-\omega^2 h+\frac{\omega^4 h^3}{4}\right) \\ &= \lambda^2 + (\omega^2 h^2 - 2)\lambda + 1 \end{aligned}$$

We obtain

$$\lambda_{1,2} = \frac{1}{2}\left[2-\omega^2 h^2 \pm \omega h\sqrt{\omega^2 h^2 - 4}\right]$$

and setting $z = h\omega$

$$\lambda_{1,2} = \frac{1}{2}\left[2-z^2 \pm z\sqrt{z^2-4}\right] .$$

I plot the boundary of this stability region in MATLAB:

```
% make a contour plot of the stability region of Verlet

% number of points in x and y
numz = 101;
numr = 3; % bound of x,y
x = linspace(-numr,numr,numz); % real
y = linspace(-numr,numr,numz); % imag
[xz,yz] = meshgrid(x,y); % make matrices
h = 1; % h just scales the axes
z = xz+1i*yz; % use this for complex math
stable_region1 = abs(0.5.*(2-z.^2-z.*sqrt(z.^2-4)));
stable_region2 = abs(0.5.*(2-z.^2+z.*sqrt(z.^2-4)));

% non-vectorized search for the boundary
% could use 'find' in a smart way to vectorize
% but this works
stable_region = zeros(numz);
for i=1:numz
    for j=1:numz
```

```
        % check if on the boundary of both
        if stable_region1(i,j) <= 1 && stable_region2(i,j) <= 1
            stable_region(i,j) = 1;
        end
    end
end

% plot the first figure
% looks like that in notes
figure;
contour(xz,yz,stable_region1,[1,1],'k');
hold on;
contour(xz,yz,stable_region2,[1,1],'k');
% set(gca,'XTick',1:(numz/12):numz)
% set(gca,'XTickLabel',x(1:(numz/12):end))
grid on;
xlabel('h␣\omega␣_R','FontSize',24);
ylabel('h␣\omega␣_I','FontSize',24);
saveas(gcf,'andy_hw05_prb09_01.png')
% close(hf);

% plot the second figure
% cleaner boundary from search
figure;
contour(xz,yz,stable_region,[1,1],'k');
grid on;
xlabel('h␣\omega␣_R','FontSize',24);
ylabel('h␣\omega␣_I','FontSize',24);
saveas(gcf,'andy_hw05_prb09_02.png')
% close(gf);
```

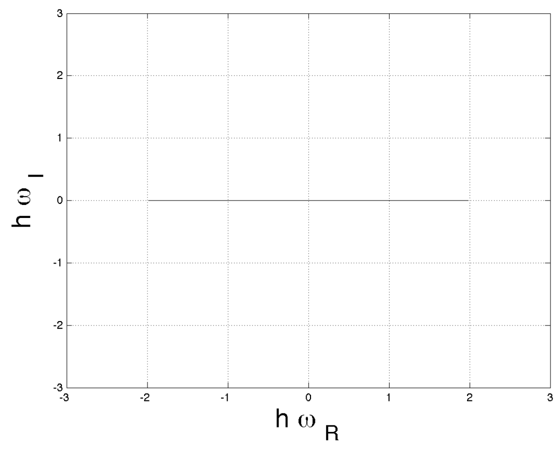

Figure 10: The stability region of the Verlet method.

10. Consider a system of two linear ODEs:

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = A \begin{pmatrix} x(t) \\ y(t) \end{pmatrix}, \qquad\qquad A = \begin{pmatrix} 100 & 99 \\ 99 & 100 \end{pmatrix}.$$

(a) Would you call this system numerically stiff?

(b) Will you answer change if both entries of A with 100 are replaced with -100? Please explain.

Solution:

(a) No. I compute the eigenvalues of A:

$$\begin{aligned} \begin{vmatrix} 100-\lambda & 99 \\ 99 & 100-\lambda \end{vmatrix} &= \lambda^2 - 200\lambda + 100\cdot 100 - 99\cdot 99 \\ &= \lambda^2 - 200\lambda + 199 = (\lambda-1)(\lambda-199) \end{aligned}$$

which are thus $1, 199$. The ratio of these is large (satisfies criterion I), but the larger eigenvalue has positive real part (fails criterion II).

(b) Yes. Again I compute the eigenvalues

$$\begin{aligned} \begin{vmatrix} -100-\lambda & 99 \\ 99 & -100-\lambda \end{vmatrix} &= \lambda^2 + 200\lambda + 100\cdot 100 - 99\cdot 99 \\ &= \lambda^2 + 200\lambda + 199 = (\lambda+1)(\lambda+199) \end{aligned}$$

which are thus $-1, -199$. The ratio of these is large (satisfies criterion I), and the larger eigenvalue has negative real part (satisfies criterion II).

Bonus-1 Explain why the solutions of `ode45` with rather similar tolerances behave *qualitatively* differently.

Hint: Look at the plot of its stability region. You can obtain it as in HW 4, while using the fact that `ode45` employs a 5th-order accurate Runge-Kutta method to compute its solution. (To see how the last piece of information determines the stability region, review what equation(s) determined the stability regions of other Runge-Kutta type methods.)

You explanation must be sufficiently detailed and coherent to recieve credit. If you are not sure what "sufficient" is in this case, ask the instructor.

Solution: Looking back at the results of Problem 6, note that `ode45` used $h \simeq 0.24$ for `AbsTol` $= .002$ and $h \simeq 0.26$ for `AbsTol` $= .003$. The stable region for the 5th-order RK method, as for the 4-th order in HW 4 and in the notes (remark after Eq 4.21), we expect to be the 5-th degree polynomial approximating the exact solution. Since RK is a general-purpose method, we study the stable region as in the 1D case, setting $\lambda = \pm i\omega$. Specifically the 5-th order method is stable when

$$\left| \sum_{k=0}^{5} \frac{(h\lambda)^k}{k!} \right| \leq 1. \qquad (5)$$

I wasn't able to to explain why the different tolerance led to such different solutions, since it seems to me that both values of h are within the stability region of the 5th order method. Here is the code, and plot, anyway:

```
% make a contour plot of the stability region of RK 5th order

% number of points in x and y
numz = 1000;
numr = 4; % bound of x,y
x = linspace(-numr,numr,numz); % real
y = linspace(-numr,numr,numz); % imag
[xz,yz] = meshgrid(x,y); % make matrices
h = 1; % h just scales the axes
z = xz+1i*yz; % use this for complex math
y = zeros(1,100);
% build the sum
% non vectorized but works
stable_region1 = ones(size(z));
for k=1:5
    stable_region1 = stable_region1+h^k.*z.^k./factorial(k);
end
stable_region1 = abs(stable_region1);


% plot the first figure
figure;
contour(xz,yz,stable_region1,[1,1],'k');
grid on;
xlabel('h␣\lambda␣_R','FontSize',24);
ylabel('h␣\lambda␣_I','FontSize',24);
saveas(gcf,'andy_hw05_prb11_01.png')
```

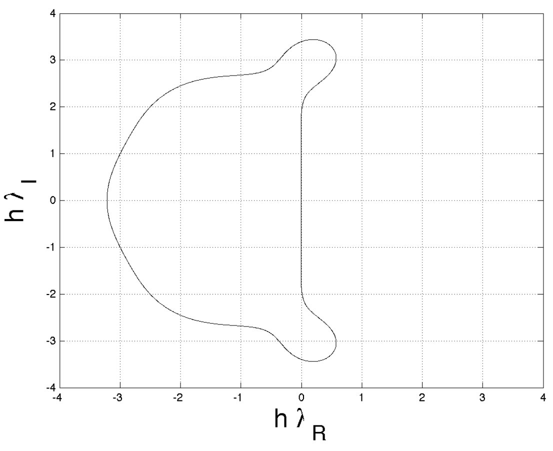

Figure 11: The stability region of the 5th order RK method.

Bonus-2 Apply the Verlet-1 and -2 methods to the Kepler two-body problem (5.30) (see also Problem 1(c) above). Use $h = 0.1$ and $t_{\max} = 500$. As the initial condition, use

$$q(0) = 1 - \text{ecc}, \quad r(0) = 0, \quad Q(0) = 0, \quad R(0) = \sqrt{\frac{1+\text{ecc}}{1-\text{ecc}}} \quad \text{for ecc} = 0.6,$$

which corresponds to the exact solution being an elipse with eccentricity 0.6. Compute the conserved quantities: the Hamiltonian (5.31), the angular momentum (5.32), and the components of the Runge-Lenz vector (5.33), and the plot them versus time.

Now plot the trajectory defined by Eqs. (5.30). What effect does the nonconservation of the Runge-Lenz vector appear to have on the numerical solution of (5.30)? If you are not sure, reduce the step size (but keep $t_{\max}$ the same) and see what this does to your numerical solution.

Solution: I present my code and the requested plots below. Non-conservation of the Runge-Lenz vector causes the solution's orbit to lose orientation, in such a way that the orientation of the elipse moves entirely around the center by $t = 1000$.

```
% keep in mind variable map
% [q' r' z' w'] where z' = q'' and w' = r''

methodcell = {'verlet1_2','verlet2_2'};
resultscell = cell(1,length(methodcell));
analcell = cell(1,length(methodcell));

t0 = 0; tmax = 1000;

ecc = 0.6;
y0 = [1-ecc;0;0;-sqrt((1+ecc)/(1-ecc))];

h = 0.1;
tvec = t0:h:tmax;

% solve with verlet-1
for i=1:2
methodh = str2func(sprintf('andy_%s',methodcell{i}));
resultscell{i} = methodh(@andy_kepler2,tvec,y0,h,[]);
% analcell{i} = [sin(tvec);cos(tvec)]; % analytical

[q,r,Q,R]=deal(resultscell{1}(1,:),resultscell{1}(2,:),resultscell{1}(3,:),resultscell
    {1}(4,:));

figure;
subplot(224);
plot(q,r)
xlabel('q','FontSize',20)
ylabel('r','FontSize',20)

subplot(221);
H = 0.5.*(Q.^2+R.^2)-1./(sqrt(q.^2+r.^2));
plot(tvec,H);
xlabel('t','FontSize',20)
ylabel('H','FontSize',20)

subplot(222);
A = q.*R-r.*Q;
plot(tvec,A);
xlabel('t','FontSize',20)
```

```
ylabel('A','FontSize',20)

subplot(223);

Li = R.*(q.*R-r.*Q)-(q)./sqrt(q.^2+r.^2);
Lj = -Q.*(q.*R-r.*Q)-(r)./sqrt(q.^2+r.^2);
plot(tvec,Li,'b');
hold on;
plot(tvec,Lj,'r');
hold off;
legend({'RL␣i','RL␣j'});
xlabel('t','FontSize',20);
ylabel('L','FontSize',20);

set(gcf, 'units', 'inches', 'position', [1 1 10 10])
set(gcf,'PaperPositionMode','auto')
print('-depsc2','-zbuffer','-r200',sprintf('andy_hw05_prb12_%02g.eps',i))
end
```

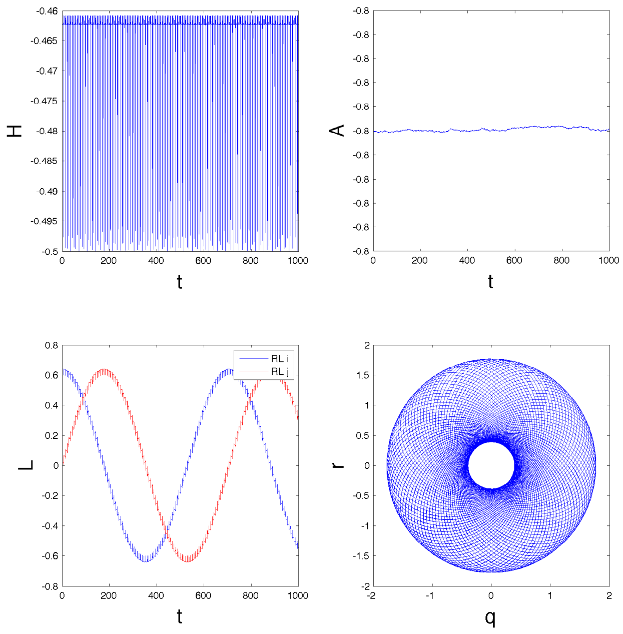

Figure 12: The Kepler two-body problem solved by Verlet-1 with $h = 0.1$. Non conservation of the LRL vector causes the orbit to lose orientation.

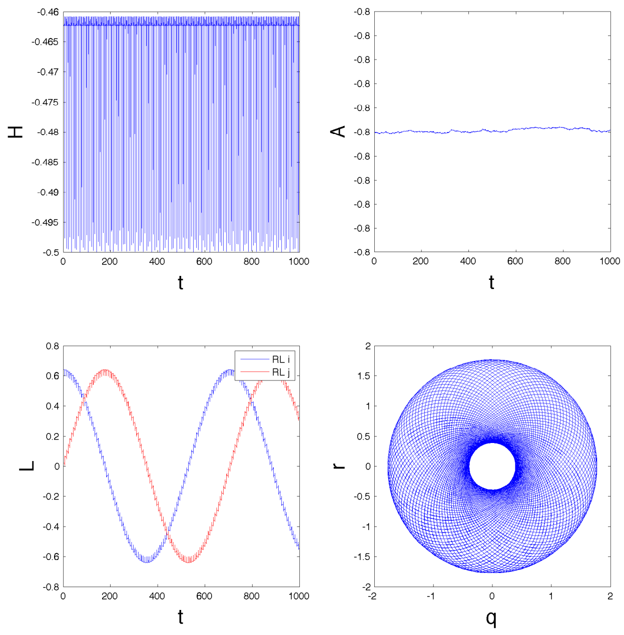

Figure 13: The Kepler two-body problem solved by Verlet-2 with $h = 0.1$. Similarly, non conservation of the LRL vector causes the orbit to lose orientation.

Appendix 1: ODE Functions

```
function dy = andy_SHO_1(t,y,params)
% simple harmonic oscillator
% y'' = -y
%
% as one equation

dy = -y;
```

```
function dy = andy_SHO_2(t,y,params)
% simple harmonic oscillator
% y'' = -y
%
% as a system of two equations:
% v' = -y
% y' = v
%
% return [y';v']

dy = [y(2);-y(1)];
```

```
function dy = andy_SHO_damped(t,y,params)
% simple harmonic oscillator
% y'' = -2\gamma y' -\omega_0^2 y
%
% as a system of two equations:
% v' = -2\gamma v -\omega_0^2 y
% y' = v
%
% return [y';v']

[g,w0] = deal(params{1},params{2});

dy = [y(2);-2*g*y(2)-w0^2*y(1)];
```

```
function dy = andy_kepler2(t,y,params)
% kepler 2-body problem in 2D
% as a system of 2 second order eq's
%
% keep in mind the variable map for 2D
% [q' r' z' w'] where z' = q'' and w' = r''
% but i only care about z', w' here

dy = [-(y(1))/(y(1)^2+y(2)^2)^(3/2);-(y(2))/(y(1)^2+y(2)^2)^(3/2)];
```

```
function dy = andy_kepler4(t,y,params)
% kepler 2-body problem in 2D
% as a system of 4 ODE's
%
% keep in mind the variable map for 2D
% [q' r' z' w'] where z' = q'' and w' = r''

dy = [y(3);y(4);-(y(1))/(y(1)^2+y(2)^2)^(3/2);-(y(2))/(y(1)^2+y(2)^2)^(3/2)];
```

Appendix 2: Numerical Methods

```
function yvec = andy_ME(func,tspan,y0,h,params)
% modified Euler

yvec = [];
yvec = [yvec y0];
t = tspan(1);
for i=2:length(tspan)
    k1 = func(t,yvec(:,i-1),params);
    k2 = func(t+h,yvec(:,i-1)+h*k1,params);
    yvec = [yvec yvec(:,i-1)+h/2*(k1+k2)];
    t=t+h;
end
```

```
function yvec = andy_SCD(func,tspan,y0,h,params)
% implements simple-central difference

yvec = [];
% start with (5.17)
yvec = [yvec y0(1) y0(1)+h*y0(2)+func(0,y0(1),params)];
disp(yvec);
t = tspan(1);
for i=3:length(tspan)
    % equation (5.12)
    y = 2*yvec(i-1)-yvec(i-2)+h^2*func(t,yvec(i-1),params);
    yvec = [yvec y];
    t=t+h;
end
```

```
function yvec = andy_SE(func,tspan,y0,h,params)
% simple Euler
%
% not much else to say

t = tspan(1);

yvec =  [];
yvec = [yvec y0];
disp(yvec);
for i=2:length(tspan)
    yvec = [yvec yvec(:,i-1)+h*func(t,yvec(:,i-1),params)];
    t = t+h;
end
```

```
function yvec = andy_cRK(func,tspan,y0,h,params)
% classical Runge-Kutta 4-th order

% set the coefficients
[a11,a21,a22,a31,a32,a33] = deal(0.5,0.0,0.5,0.0,0.0,1.0);
[b1,b2,b3,b4] = deal(1/6,1/3,1/3,1/6);
[c1,c2,c3] = deal(0.5,0.5,1);
t = tspan(1);

yvec =  []; %linspace(tspan[0],tspan[-1],num=floor((tspan[-1]-tspan[0])/h))
yvec = [yvec y0]; %[0] = y0
for i=2:length(tspan)
    k1 = h*func(t,yvec(:,i-1),params);
    k2 = h*func(t+c1*h,yvec(:,i-1)+a11*k1,params);
    k3 = h*func(t+c2*h,yvec(:,i-1)+a21*k1+a22*k2,params);
    k4 = h*func(t+c3*h,yvec(:,i-1)+a31*k1+a32*k2+a33*k3,params);
    yvec = [yvec yvec(:,i-1)+b1*k1+b2*k2+b3*k3+b4*k4]; %[i] = yvec[i-1] + b1*k1 + b2*k2
        + b3*k3 + b4*k4
    t = t+h;
end
```

```
function yvec = andy_symE1(func,tspan,y0,h,params)
% symplectic Euler 1
%
% updates the equations in forward order
%
% evaluates the function func for the whole new yvec, but
% really only needed the jth update (this is
% unnecessary function evaluation)

t = tspan(1);

% don't preallocate
yvec =  [];
yvec = [yvec y0];

for i=2:length(tspan)
    % guess
    ynew = yvec(i-1);
    for j=1:length(y0)
        % update y
        tmp = yvec(i-1)+h*func(t,ynew,params);
        % set
        ynew(j) = tmp(j);
    end
    % savep
    yvec = [yvec ynew];
    t = t+h;
end
```

```
function yvec = andy_symE2(func,tspan,y0,h,params)
% symplectic Euler 2
%
% updates the equations in reverse order
%
% evaluates the function func for the whole new yvec, but
% really only needed the jth update (this is
% unnecessary function evaluation)

t = tspan(1);

% don't preallocate
yvec =  [];
yvec = [yvec y0];

for i=2:length(tspan)
    % guess
    ynew = yvec(i-1);
    for j=fliplr(1:length(y0))
        % update y
        tmp = yvec(i-1)+h*func(t,ynew,params);
        % set
        ynew(j) = tmp(j);
    end
    % savep
    yvec = [yvec ynew];
    t = t+h;
end
```

```
function yvec = andy_symE3(func,tspan,y0,h,params)
% symplectic Euler 3
%
% updates the equations in random order
%
% evaluates the function func for the whole new yvec, but
% really only needed the jth update (this is
% unnecessary function evaluation)

t = tspan(1);

% random permutation (keep this random order)
rl = randperm(length(y0));
fprintf('using␣the␣random␣permutation:\n');
disp(rl);

% don't preallocate
yvec =  [];
yvec = [yvec y0];

for i=2:length(tspan)
    % guess
    ynew = yvec(:,i-1);
    for j=rl
        % update y
        tmp = yvec(:,i-1)+h*func(t,ynew,params);
        % set
        ynew(j) = tmp(j);
    end
    % savep
    yvec = [yvec ynew];
    t = t+h;
end
```

```
function yvec = andy_verlet1(func,tspan,y0,h,params)
% implements Verlet-1
%
% not a particularly fast implementation
% but it works

yvec = [];
yvec = [yvec y0];
disp(yvec);
t = tspan(1);
for i=2:length(tspan)
    tmp = func(t,yvec(1,i-1),params);
    yup = yvec(1,i-1)+h*yvec(2,i-1)+h^2/2*tmp;
    vup = yvec(2,i-1)+h/2*(tmp+func(t,yup,params));
    yvec = [yvec [yup;vup]];
    t=t+h;
end
```

```
function yvec = andy_verlet1_2(func,tspan,y0,h,params)
% implements Verlet-1 for 2D
%
% not a particularly fast implementation
% but it works
%
% [q' r' z' w'] where z' = q'' and w' = r''
%
% the function handle returns [z',w'] from [q,r]
```

```
yvec = [];
yvec = [yvec y0];

t = tspan(1);
for i=2:length(tspan)
    tmp = func(t,yvec(1:2,i-1),params);
    % qup = yvec(1,i-1)+h*yvec(3,i-1)+h^2/2*tmp(1); % q
    % rup = yvec(2,i-1)+h*yvec(4,i-1)+h^2/2*tmp(2); % r
    qrup = yvec(1:2,i-1)+h*yvec(3:4,i-1)+h^2/2*tmp; %[q,r]
    % zup = yvec(3,i-1)+h/2*(tmp(1)+func(t,[qup;rup],params)); % z'
    zwup = yvec(3:4,i-1)+h/2*(tmp+func(t,[qrup],params)); % z',w'
    yvec = [yvec [qrup;zwup]];
    t=t+h;
end
```

```
function yvec = andy_verlet2(func,tspan,y0,h,params)
% implements Verlet-2
%
% not a particularly fast implementation
% but it works

yvec = [];
yvec = [yvec y0];
t = tspan(1);
for i=2:length(tspan)
    vup = yvec(2,i-1)+h*func(t,yvec(1,i-1)+h/2*yvec(2,i-1),params);
    yup = yvec(1,i-1)+(h/2)*(yvec(2,i-1)+vup);
    yvec = [yvec [yup;vup]];
    t=t+h;
end
```

```
function yvec = andy_verlet2_2(func,tspan,y0,h,params)
% implements Verlet-2 for 2D
%
% not a particularly fast implementation
% but it works
%
% [q' r' z' w'] where z' = q'' and w' = r''
%
% the function handle returns [z',w'] from [q,r]


yvec = [];
yvec = [yvec y0];

t = tspan(1);
for i=2:length(tspan)
    zwup = yvec(3:4,i-1)+h*func(t,yvec(1:2,i-1)+h/2*yvec(3:4,i-1),params);
    qrup = yvec(1:2,i-1)+h/2*(yvec(3:4,i-1)+zwup);
    yvec = [yvec [qrup;zwup]];
    t=t+h;
end
```