PHYS 410: Computational Physics Fall 2023
Homework 2
Due: Friday, November 3, 11:59 PM
PLEASE report all bugs, comments, gripes etc. to Matt: `choptuik@physics.ubc.ca`

Problem 1

Code a `MATLAB` function with the header

```
function yout = rk4step(fcn, t0, dt, y0)
```

that takes a single fourth-order Runge-Kutta step. The input and output arguments are defined as follows:

```
%  Inputs
%           fcn:     Function handle for right hand sides of ODEs (returns
%                    length-n column vector).
%           t0:      Initial value of independent variable.
%           dt:      Time step.
%           y0:      Initial values (length-n column vector).
%
%  Output
%           yout:    Final values (length-n column vector)
```

Using a system of ODEs of your own choosing that has a known solution, verify that the local (per step) error incurred by using `rk4step` is $O(\Delta t^5)$. Document your findings in your writeup, and ensure that your results can be reproduced by running a script called `trk4step`.

Problem 2

Use your implementation of `rk4step` to code another function with header:

```
function [tout yout] = rk4(fcn, tspan, y0)
```

Let n_{out} be the length of the vector `tspan`, whose elements can be assumed to be in strictly increasing order. Then `rk4` integrates the system of ODEs defined by `fcn` and outputs the approximate values of the unknowns at the n_{out} values of `tspan`. The input and output arguments to `rk4` are defined as follows:

```
%  Inputs
%        fcn:     Function handle for right hand sides of ODEs (returns
%                 length-n column vector)
%        tspan:   Vector of output times (length nout).
%        y0:      Initial values (length-n column vector).
%
%  Outputs
%        tout:    Vector of output times.
%        yout:    Output values (nout x n array. The ith column of yout
%                 contains the nout values of the ith dependent variable).
```

The step sizes that `rk4` is to use in the integration are given by the differences between the adjacent elements of `tspan`. That is, `rk4` is *not* to take any substeps to times that are not defined in `tspan`.

Use your implementation of `rk4` to solve the simple harmonic motion system:

$$\frac{d^2x(t)}{dt^2} = -x(t)$$
$$x(0) = 0$$
$$\frac{dx}{dt}(0) = 1$$

on the domain $0 \leq t \leq 3\pi$ and with Δt corresponding to levels $\ell = 6, 7$ and 8. Plot the scaled errors, with scaling factors corresponding to the anticipated reduction in error relative to level 6, on a single plot which is to be included in your write up. Ensure that the calculations and plotting can be reproduced by running a script called `trk4_sho`.

Now use `rk4` to integrate the equations for the unforced Van der Pol oscillator (see Tutorial 4) with $a = 5$ on the domain $0 \leq t \leq 100$ at level 12 with initial conditions

$$x(0) = 1$$

$$\frac{dx}{dt}(0) = -6$$

Make plots of (1) the oscillator's position and (2) the phase-space evolution and include them in your write up. Ensure that the calculations and plotting can be reproduced by running a script called `trk4_vdp`.

Problem 3

In this problem you will implement an *adaptive* RK4 integrator—that is, your function will vary the step size in response to the estimated local solution error.

Error estimation

You will base the error estimation on the following analysis. (Do *not* implement the adaptive stepping using some approach of your own invention. Ask for help if you do not understand what follows.)

Let t_0 be the current value of the independent variable during an integration, and let Δt be the current step size. Then for our fourth order Runge-Kutta method we have (C denotes "coarse")

$$y_C(t_0 + \Delta t) \approx y_{\text{exact}}(t_0 + \Delta t) + k(t_0)\Delta t^5$$

where $k(t)$ is some function of time. Note that we deduce from this that the local error, $e_C = y_C - y_{\text{exact}}$, is $\approx k(t_0)\Delta t^5$. Now consider advancing from t_0 to $t_0 + \Delta t$ using two time steps of size $\Delta t/2$. After the first time step we have (F denotes "fine")

$$y_F(t_0 + \Delta t/2) \approx y_{\text{exact}}(t_0 + \Delta t/2) + k(t_0)(\Delta t/2)^5$$

Because the error is additive for a small number of steps (why?), after the second step we will have

$$\begin{aligned} y_F(t_0 + \Delta t) &\approx y_{\text{exact}}(t_0 + \Delta t) + k(t_0)(\Delta t/2)^5 + k(t_0 + \Delta t/2)(\Delta t/2)^5 \\ &\approx y_{\text{exact}}(t_0 + \Delta t) + 2k(t_0)(\Delta t/2)^5 \end{aligned}$$

where the last step follows from the fact that $k(t_0 + \Delta t/2) = k(t_0) + O(\Delta t)$. Therefore, if we subtract y_C and y_F at the advanced time, we get

$$y_C(t_0 + \Delta t) - y_F(t_0 + \Delta t) \approx \frac{15}{16}k(t_0)\Delta t^5 \approx \frac{15}{16}e_C.$$

In summary, by taking one step of size Δt, and then, starting from the same initial condition, two steps of size $\Delta t/2$, a subtraction of the resulting values will yield an estimate of the local solution error (up to the constant factor 15/16).

Implementation

Implement an adaptive RK4 integrator as a `MATLAB` function with header

```
function [tout yout] = rk4ad(fcn, tspan, reltol, y0)
```

The inputs and outputs for the function are defined by

```
%  Inputs
%        fcn:     Function handle for right hand sides of ODEs (returns
%                 length-n column vector)
%        tspan:   Vector of output times (length nout vector).
%        reltol:  Relative tolerance parameter.
%        y0:      Initial values (length-n column vector).
%
%  Outputs
%        tout:    Output times (length-nout column vector, elements
%                 identical to tspan).
%        yout:    Output values (nout x n array. The ith column of yout
%                 contains the nout values of the ith dependent variable).
```

As previously, the elements of `tspan` can be assumed to be in strictly increasing order.

`rk4ad` should integrate the set of ODEs defined by `fcn` using the error estimation procedure described above. That is, when taking any step of size Δt, the function should also take two steps of size $\Delta t/2$, compute the error estimate, then adjust the step size accordingly so that the tolerance given by `reltol` is achieved. It is up to you to figure out precisely how the step size will be changed, and note that Δt can either decrease, increase or stay the same at any given step.

You must also ensure that the function calculates approximate solution values at *precisely* those times defined by the `tspan` array. In general, this will necessitate some additional adjustments of Δt, and in this regard, note that it doesn't matter if a particular step size turns out to be less than it strictly needs to be to achieve the error tolerance.

Your implementation of `rk4ad` must "self-initialize": that is you cannot assume that a particular value of Δt will be sufficient to meet the error criteria at the initial time. Rather, your function must compute an appropriate Δt so that the relative error tolerance is achieved initially.

Additionally, implement a "floor" of `1.0e-4` for the time step: do not allow your integrator to use a step size which is less than this.

Hint: If it is not obvious, `rk4step` should be used here as well.

Results

Once you are satisfied that your implementation of `rk4ad` is working, use it to integrate the simple harmonic oscillator system given in Problem 2 using `tspan` defined by

```
tspan = linspace(0.0, 3.0 * pi, 65);
```

and values for `reltol` of

```
1.0e-5, 1.0e-7, 1.0e-9, 1.0e-11
```

Make plots of the resulting errors from the four calculations and include them in your writeup. Ensure that the calculations and plotting can be reproduced by running a script called `trk4ad_sho`.

Finally, use `rk4ad` to integrate the unforced Van der Pol equation with $a = 5.0$, `tspan` defined by

```
tspan = linspace(0.0, 100, 4097);
```

`reltol = 1.0e-10`, and the same initial conditions as in Problem 2. Make plots of the oscillator displacement and the phase space evolution and include them in your writeup. Ensure that the calculations and plotting can be reproduced by running a script called `trk4ad_vdp`.