MATHS 2104 Numerical Methods
Assignment 4:
Shooting ODEs with Newton's method
Due: Tuesday 24 October 2017 at 1:00pm

CHECKLIST

☐: Have you shown all of your working for every mathematical question?

☐: Have you included all Matlab code and plots to support your answers in programming questions?

☐: Have you made sure that all plots have labelled axes, and legend where appropriate?

☐: Have you submitted all code required to Cody Coursework?

☐: If before the deadline, have you submitted your assignment via the online submission on Canvas?

☐: Is your submission a single pdf file - correctly orientated, easy to read? If not, penalties apply.

☐: Penalty for submitting more than one document: 10% of final mark, per extra document. Note: you may resubmit and your final version is marked, but final document must be a single file.

☐: If after the deadline, but within 24 hours, have you contacted us via the enquiry page on Canvas (if applying for an extension) and submitted your assignment online via Canvas?

☐: Penalties for late submission: within 24 hours, 40% of final mark. After 24 hours, assignment cannot be submitted and you get zero.

☐: Assignments emailed instead of submitted via Canvas will not be marked and will receive zero.

☐: Have you checked that the assignment submitted is the correct one, as we cannot accept other submissions after the due date?

Preamble

The aims of this assignment are to:

- Program the shooting method for BVPs using RK4 and Newton's method;
- Explore a nonlinear oscillator system exhibiting chaotic dynamics.

Contents

1 Solving a nonlinear boundary-value problem by shooting

The Van der Pol oscillator is a famous equation exhibiting nonlinear dynamics:

$$y'' - \mu(1 - y^2)y' + y = 0 \tag{1}$$

This equation comes from a classic experimental electric device; y is related to the voltage across the oscillator, and μ is a free parameter.

We will try to solve (1) on the domain $0 \le t \le T$ subject to the boundary conditions

$$y(0) = 0.5; \quad y(T) = 1. \tag{2}$$

We solved a linear boundary value problem in Assignment 4, but this one is nonlinear. In this assignment we'll implement a 'shooting' method to solve this boundary value problem, and do a little bit of exploration of *chaos* in this system. In doing so, we will mash up two parts of the course: solving nonlinear equations (Newton iteration) and initial value problems (Runge–Kutta methods).

1.1. Let $y_1 = y$ and $y_2 = y'$. Write the second-order nonlinear ODE (1) as a system of first-order ODEs $\boldsymbol{y}' = \boldsymbol{f}(t, \boldsymbol{y})$.

1.2. Write a MATLAB function that calculates $\boldsymbol{f}(t, \boldsymbol{y})$. The function must have the interface

```
function f = vanderpol(t,y,Mu)
```

where `f` is the column vector $\boldsymbol{f}(t, \boldsymbol{y})$, `t` is the independent variable, `y` is the column vector $\boldsymbol{y}$, and `Mu` is the parameter μ. Save the function as `vanderpol.m`.

Make sure to use `Mu`, not `mu`!

(You'll need to use `t` explicitly later in the assignment.)

1.3. Write a MATLAB function that uses the fourth-order Runge–Kutta scheme

$$\boldsymbol{y}_{k+1} = \boldsymbol{y}_k + \frac{1}{6}\left[\boldsymbol{F}_1 + 2\boldsymbol{F}_2 + 2\boldsymbol{F}_3 + \boldsymbol{F}_4\right],$$

where

$$\begin{aligned}
\boldsymbol{F}_1 &= h\boldsymbol{f}(t_k, \boldsymbol{y}_k),\\
\boldsymbol{F}_2 &= h\boldsymbol{f}(t_k + \tfrac{1}{2}h, \boldsymbol{y}_k + \tfrac{1}{2}\boldsymbol{F}_1),\\
\boldsymbol{F}_3 &= h\boldsymbol{f}(t_k + \tfrac{1}{2}h, \boldsymbol{y}_k + \tfrac{1}{2}\boldsymbol{F}_2),\\
\boldsymbol{F}_4 &= h\boldsymbol{f}(t_k + h, \boldsymbol{y}_k + \boldsymbol{F}_3),
\end{aligned}$$

to solve $\boldsymbol{y}' = \boldsymbol{f}(t, \boldsymbol{y})$ subject to $\boldsymbol{y}(0) = \boldsymbol{y}_0$. The function must have the interface

```
function y = rk4(func,t,y0)
```

where `y` is a matrix containing the solution, `func` is the user-supplied function $\boldsymbol{f}(t, \boldsymbol{y})$, `t` is a row vector of values of the independent variable in ascending order and `y0` is a column vector of initial conditions. Save the function as `rk4.m`.

The solution at each point in the vector `t` must be stored in the columns of `y`, that is, `y(:,k)` should be the solution at `t(k)`. You should assume that `func(t,y)` returns a column vector given a scalar `t` and a column vector `y`. Your function should work for vectors of any length. Do not assume that `t` is equispaced.

Note that `rk4` should be a straightforward extension of the function `ieuler.m` that you developed in item 2.1 of Practical 5. It is a good idea to test `rk4` before proceeding, perhaps using a script similar to that given in item 2.2 of Practical 5.

The system of first-order ODEs derived in part 1.1 must be solved subject to the boundary conditions (2). From these boundary conditions, we know that $y_1(0) = 0.5$. If we knew $y_2(0)$, then we would have an initial value problem that could be solved using the Runge-Kutta method. But we don't know $y_2(0)$. Instead, we need to satisfy $y_1(T) = 1$. The idea behind 'shooting' is to guess a value for $y_2(0)$, say x, then integrate from $t = 0$ to $t = T$, hoping that the computed value of $y_1(T) = 1$. If $y_1(T) \neq 1$, then we adjust x and try again. We will use Newton iteration to do this, but first we need to create a function that calculates the residual $y_1(T) - 1$.

1.4. Write a MATLAB function that uses `rk4` to integrate the system of ODEs found in part 1.1 subject to the initial conditions

$$y_1(0) = 0.5, \quad y_2(0) = x,$$

and returns the value of $y_1(T) - 1$. Your function must have the interface

```
function res = vanderpolresidual(x,t,Mu)
```

where `res` is the (scalar) value of $y_1(T) - 1$, `x` is the guessed value of the initial condition $y_2(0)$, `t` is a row vector containing values of the independent variable in ascending order, and `Mu` is μ. Save the function as `vanderpolresidual.m`

This and `vanderpol.m` are very short scripts. Still, please don't forget to document!

This function will need to call

```
y = rk4(@(tt, yy) vanderpol(tt, yy, Mu), t, [0.5; x])
```

to obtain the solution.

Note that the `@` symbol creates a handle or pointer to the function `vanderpol`. If `n = length(t)`, then you may assume that $y_1(T) \equiv$ `y(1,n)`.

As a check, the function call

```
vanderpolresidual(1.5,linspace(0,5,100),0.5)
```

should return a scalar value of -2.4350.

1.5. The value of x is to be found using Newton iteration. We want to solve

$$f(x) = 0, \tag{3}$$

where $f(x) \equiv$ `vanderpolresidual(x,t)` for some vector `t` that is specified by the user.

You should have already developed a function called `newton` that implements Newton iteration in Practical 5. If so, then you should use that function. If not, then you will need to develop this function according to the specifications set out in item 1.1 of Practical 5. Save your function as `newton.m`.

Some freebie Cody marks here if you've done the prac!

1.6. Write a script that solves the boundary value problem given by (1) and boundary conditions (2). Save your script as `solvevanderpol.m`.

Your script must:

(a) Set up a vector `t` of points in the interval $0 \le t \le 100$, with step-size $h = 0.01$.

(b) Loop over all values of μ from 0 to 4 in steps of 0.5, and:

i. Find the solution of (3) by calling

```
s = newton(@(x) vanderpolresidual(x, t, Mu), 1, 1e-8)
```

Note that the syntax `@(x) vanderpolresidual(x,t,Mu)` creates a handle to an anonymous function with one variable, as is required by the function `newton`. This anonymous function is the same as `vanderpolresidual` but with `t` and `Mu` fixed to whatever is set in part (a). We used a similar trick in item 2.9 from Practical 3.

ii. Use `rk4` to solve the system of ODEs found in part 1.1 subject to the initial conditions

$$y_1(0) = 0.5, \quad y_2(0) = \texttt{s},$$

where `s` is obtained from part (b).

iii. Plot the solution y_1 versus y_2 for $10 \le t \le 100$. Make sure your plot axes are appropriately labelled, use

No need to include a legend here.

`axis equal`, and overlay the curves for each value of μ.

Include your code and graph with your report. [1] Take a look at some plots of y_1 and y_2 versus t for different values of μ as well to get a feel for what these nonlinear oscillations look like.

1.7. In this problem, we do not have an exact solution with which to compare. What other evidence could you provide to show that your numerical solution is reliable?

Time to change the model up a bit.

The *forced* Van der Pol oscillator equation is

$$y'' - \mu(1-y^2)y' + y = F_0\cos(\omega t) \qquad (4)$$

where we'll take $F_0 = 1.2$ and $\omega = \frac{2\pi}{10}$.

1.8. Change one line in one of your functions such that you can now solve the forced Van der Pol equation. I'm happy for you to do this by commenting out and replacing the relevant line (to avoid writing new `vanderpolforced`, `vanderpolforcedresidual` etc functions), but make sure you make the change you made clear in your assignment submission.

1.9. Solve the BVP (1)-(2) as before with $h = 0.01$, $T = 100$, $\mu = 8$. Plot y_1, y_2 versus t. Do the same thing, but this time solving the forced Van der Pol equation (4) with BCs (2). What happens?

The reason this occurs is because the forced Van der Pol oscillator exhibits *deterministic chaos*: small changes in initial condition $y(0)$ lead to big differences in $y(T)$, in an unpredictable way. Therefore, shooting cannot work in chaotic systems – we can't solve the BVP!

We can however explore the chaotic regions a little bit.

1.10. Add code to `solvevanderpol.m` which plots solutions (y_1 and y_2 versus t) of the forced Van der Pol oscillator for $\mu = 8.25$, 8.5, and 8.75. How many periods does the oscillation have for each value of μ? (You don't need to solve the BVP; just solve forwards from whatever initial condition you like.)

Count the local maxima of y_2 until it repeats.

1.11. Change μ very slightly to $\mu = 8.53$. What happens to the period now? Plot y_1 versus y_2 for $10 \leq t \leq 2000$ to get a good feel for what's going on here.

What you're observing here is chaotic dynamics in the forced Van der Pol oscillator, the appearance of which are controlled by the parameter μ. Figure 6 of the Scholarpedia page on the Van der Pol oscillator summarises when chaos appears in the system. The lower panel shows the maximal Lyapunov exponent, which characterises the degree of

[1] These curves are called *limit cycles*, and the spikier ones are examples of *relaxation oscillations*.

unpredictability in the system, as a function of μ. Surprisingly, chaos appears and disappears very suddenly in the system!

The rabbit hole goes very deep here, and you could do worse things with your life than studying this stuff further. An important point is that none of this fascinating science was known until after the advent of 20th century computational mathematics (for weather prediction actually), because the dynamics can't be observed without numerical simulations. A great starting place is James Gleick's book *Chaos: Making a new science*, and a number of lecturers (myself included) in the School of Mathematical Sciences at the University of Adelaide!

2 Assignment requirements

You must submit the final versions of your functions `newton` and `rk4` to Cody Coursework, as follows:

2.1. Login to Cody Coursework.

2.2. Click on "Assignment 4" and then on "Newton solver" / "Runge-Kutta solver".

2.3. To submit your code, click on "Solve" and copy and paste your functions `newton`/`rk4` into the text boxes provided.

2.4. Test your function by clicking on "Test". When you are happy, click "Submit" to save your function. You will receive one mark for each test that is passed.

2.5. You can revise and resubmit your function as often as you like, up until the deadline. Your best solution will be used to assign your execution mark.

2.6. For general help, consult the Mathworks Cody Coursework documentation.

2.7. You do not have to submit your other scripts to Cody Coursework. (But include everything in your submission to MyUni!)