Assignment 6 — 2.086 Fall 2015

Due: *Tuesday, November 10 at 5 PM.*

Upload your solution to Stellar as a zip file "`YOURNAME_ASSIGNMENT_6.zip`" which includes the script for each question in the proper format *as well as* any MATLAB functions (of your own creation) your scripts may call. Remember to verify your upload before the assignment due time by downloading your solution folder yourself and re-running `grade_o_matic`. Full instructions for assignment preparation are given in the document "Preparing assignments for `grade_o_matic`" available on the 2.086 website.
In this assignment, the student mode of `grade_o_matic` will not provide partial credit for questions 4 and 5. For question 4, please submit your answer in pdf format. For question 5, the deliverables include a written (pdf) part as well as the script `A6Q5` and the function `Earth_Motion`. Your written responses may be typed in your favorite software, but scans of handwritten material will not be accepted.

Questions

1. (15 points). Write a function with signature

   ```
   function [x1_out,x2_out] = quadratic_solver(a,b,c)
   ```

   that finds the roots of the quadratic equation $ax^2 + bx + c = 0$ for any given finite real coefficients $a \neq 0$, b and c. The inputs to your function are the (corresponding) coefficients `a`, `b` and `c`. The output of your function is the two *real* numbers `x1_out` and `x2_out` defined as follows:

   - If the two roots are real, `x1_out` and `x2_out` are the two roots arranged such that `x2_out>x1_out`.
   - If the two roots are complex, `x1_out` is the real part of the roots and `x2_out` is the *magnitude* of the imaginary part of the roots. In other words, for the complex pair $x \pm i|y|$, `x1_out`$=x$ and `x2_out`$=|y|$.

   For this problem, we provide:

   - a template for the script `A6Q1_Template`. This script calls the MATLAB function `quadratic_solver`. The inputs to the script are the coefficients $a \neq 0$, b and c; these inputs should be defined outside the script. The output of the script is the variables `f1_out` and `f2_out` that correspond to the function outputs `x1_out` and `x2_out`.
   - a template for the function `quadratic_solver_Template` which should be your starting point for creating the MATLAB function `quadratic_solver`.

2. (30 points) We consider the scalar first-order ODE IVP

$$\begin{cases} \dfrac{du}{dt} = \lambda u + f(t), \quad 0 < t \leq t_f \\ u(t=0) = u_0 \end{cases}, \tag{1}$$

for parameter $\lambda < 0$ and $f(t)$ a prescribed source function. Let the numerical solution of this problem at timestep $j = 0, 1, ..., J$ be denoted by $\tilde{u}(j\Delta t)$, where $\Delta t = t_f/J$ is the integration timestep. We ask you to implement the Euler Backward method in a Matlab function with signature

```
function [u_vec] = Euler_Backward(u_0,lambda,f_source,t_final,J)
```

which provides the numerical solution `u_vec(j)` $= \tilde{u}((\texttt{j-1})\Delta t)$, $1 \leq$ `j` $\leq J + 1$.

For this problem, we provide:

- a template for the script `A6Q2_Template`. This script calls the MATLAB function `Euler_Backward`. The script takes the following five inputs: the initial condition u_0 which corresponds to the MATLAB variable `u_0`, the parameter $\lambda < 0$ which corresponds to `lambda`, the source function $f(t)$ which corresponds to the MATLAB function `f_source`, the final time t_f which corresponds to `t_final` and the number of steps to be used for the time integration that corresponds to the variable `J` in your script. These inputs should be defined outside the script. The script's single output is the approximate solution to our ODE IVP $\tilde{u}(j\Delta t)$, $0 \leq j \leq J$, formatted as the column vector `u_vec` of length `J+1`.
- a template for the function `Euler_Backward_Template` which should be your starting point for creating the MATLAB function `Euler_Backward`.

We emphasize that your function `Euler_Backward` should perform correctly for any set of inputs (and hence any admissible function `f_source`). You should yourself devise several test cases for which you can anticipate the correct answers and hence test `Euler_Backward`. The only deliverable for this problem is your `Euler_Backward` function (do not modify the script `A6Q2`). In particular, any "`f_source`" functions which you create to test your `Euler_Backward` code are for your own purposes and should not be uploaded in `YOURNAME_ASSIGNMENT_6`; `grade_o_matic` will create its own instances of "`f_source`" to test your `Euler_Backward` code. Recall that your `Euler_Backward` function may be run from the command line when testing with your own source functions.

3. (5 points) Modify your function of question 2 to create a function with signature

```
function [u_vec] = Crank_Nicolson(u_0,lambda,f_source,t_final,J)
```

which solves equation (1) using the Crank-Nicolson scheme. Input and output notation and sizing should be identical to question 2. A new template is provided for this question under `Crank_Nicolson_Template`. The deliverables for this problem are your script `A6Q3` and your `Crank_Nicolson` function.

4. (20 points) Use your two functions of questions 2 and 3 to compare the convergence order of the Backward Euler and Crank-Nicolson schemes. Display your results for both schemes on a single log-log plot of the (suitably defined) error versus timestep. (Hint: pick a convenient source function for which you know the analytical solution to (1).)

   Your solution will not be graded by `grade_o_matic`; as a result, there will be no student instances or partial credit. Please submit your *clearly labelled* log-log plot embedded in a pdf document named `A6Q4.pdf` with a discussion of your findings (e.g. what is the observed order of each scheme? does it agree with the theoretical result? which scheme do you prefer? did anything surprise you? etc).

5. (30 points) Neglecting the effect of other planets and rotation about their own axis, the motion of the earth around the sun (assumed stationary) is given by

$$\frac{d^2\vec{r}}{dt^2} = -\frac{GM}{r^3}\vec{r} \tag{2}$$

where $\vec{r}$ denotes the position vector of the earth relative to the center of the sun, $G = 6.67408 \times 10^{-11}\text{m}^3/(\text{Kgs}^2)$ is the gravitational constant and $M = 1.989 \times 10^{30}\text{Kg}$ is the mass of the sun. Assuming the motion to be planar and using a cylindrical coordinate system (r, θ), we can rewrite the above equation as

$$\frac{d^2r}{dt^2} - r\left(\frac{d\theta}{dt}\right)^2 = -\frac{GM}{r^2}$$
$$r\frac{d^2\theta}{dt^2} + 2\frac{dr}{dt}\frac{d\theta}{dt} = 0$$

Noting that the second equation corresponds to conservation of angular momentum (central force field) we can rewrite this system as

$$\frac{d^2r}{dt^2} - r\left(\frac{d\theta}{dt}\right)^2 = -\frac{GM}{r^2} \tag{3}$$
$$r^2\frac{d\theta}{dt} = A \tag{4}$$

where A is the constant angular momentum. Write a function with signature

```
function [r,drdt,theta] = Earth_Motion(A,r_0,drdt_0,theta_0,t_f,deltaT)
```

to integrate the system of equations (3) and (4) for the time period $0 < t \leq t_f$ using an Euler (forward) scheme, where $t = 0$ corresponds to the perihelion location (location of closest approach to the sun).

For this problem, we provide:

- a template for the script `A6Q5_Template`. This script calls the MATLAB function `Earth_Motion`. The inputs to the script are: the constant A (MATLAB variable `A`); the initial condition $r(t = 0)$ (MATLAB variable `r_0`); the initial condition $\frac{dr}{dt}(t = 0)$ (MATLAB variable `drdt_0`); the initial condition $\theta(t = 0)$ (MATLAB variable `theta_0`); the final time t_f (MATLAB variable `t_f`); and the timestep Δt (MATLAB variable `deltaT`). The constant A as well as the initial conditions $r(t = 0)$, $\frac{dr}{dt}(t = 0)$ and $\theta(t = 0)$ at the perihelion are provided in the problem template. Note that the variables t_f and Δt have been defined inside the script. The output of the script is the `(n+1)`×3 array $\left[r(j\Delta t), \frac{dr}{dt}(j\Delta t), \theta(j\Delta t)\right]$ (where $0 \leq j \leq n$ and $n = \text{ceil}(t_f/\Delta t)$) that corresponds to the MATLAB array `[r,drdt,theta]`.
- a template for the function `Earth_Motion_Template` which should be your starting point for creating the MATLAB function `Earth_Motion`.

We emphasize that your function should perform correctly for any set of inputs. You should yourself devise several test cases for which you can anticipate the correct answers.

Using your `Earth_Motion` function, we ask you to:

- Calculate the trajectory of the earth for 5 years using the given initial conditions (at the perihelium). Plot your result on a $x-y$ (or r-θ) plot. Explain your choice of integration timestep. Show (on a separate plot) the total energy of the system per unit earth mass, $E/m = 0.5[(dr/dt)^2 + r^2(d\theta/dt)^2] - GM/r$, as a function of time.
- Investigate your solution accuracy as the timestep size changes for a *fixed total integration time* t_f (e.g. t_f=5 years). You can do that by plotting the difference $|E/m(t = t_f) - E/m(t = 0)|$ as a function of the timestep on a log-log plot. Explain your results.

The deliverables for this question are your script `A6Q5`, your function `Earth_Motion` as well as a pdf document named `A6Q5.pdf` containing your answers (and figures) to the two questions above. Please note that even though `grade_o_matic` will not provide partial credit in student mode, your script and function will be graded along with the written part of your solution.