# Coursework 1: MATH 3018/6141 - Numerical methods

## Due: 19 November 2014

In this coursework you will implement given numerical algorithms. The assessment will be based on

- correct implementation of the algorithms – **4 marks**
- correct use of the algorithms in tests – **3 marks**
- figures to illustrate the tests – **2 marks**
- documentation of code – **3 marks**
- unit testing, robustness and error checking of code – **3 marks**.

The deadline is noon, Wednesday 23 November 2016 (week 8). For late submissions there is a penalty of 10% of the total marks for the assignment per day after the assignment is due, for up to 5 days. No marks will be obtained for submissions that are later than 5 days.

Your work must be submitted electronically via Blackboard. Only the Python files needed to produce the output specified in the tasks below is required.

## 1 Stiff ODEs and implicit methods

Many interesting physical systems can be written as Initial Value Problems (IVPs), which we write in the form

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{q} = \mathbf{f}(t, \mathbf{q}). \tag{1}$$

The behaviour depends on the scales (length or time) involved in the problem. When the scales differ by many orders of magnitude the system is called *stiff* and can be very difficult to simulate. In stiff problems tiny errors in the large scale behaviour can swamp the correct transient, small scale behaviour, or vice versa. This obvious problem for numerical approximations is usually avoided by using *implicit* solvers.

Here we will work with the Modified Prothero-Robinson Problem as given in the paper of Constantinescu and Sandu (Journal of Scientific Computing, **56** 28-44 [2013]). This system is here studied in terms of the variables $\mathbf{q}(t) = (x(t), y(t))^T$, which satisfies

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{q} = \frac{\mathrm{d}}{\mathrm{d}t}\begin{pmatrix} x(t) \\ y(t) \end{pmatrix} &= \mathbf{f}(t, \mathbf{q}) \\ &= \begin{pmatrix} \Gamma & \varepsilon \\ \varepsilon & -1 \end{pmatrix}\begin{pmatrix} \left(-1 + x^2 - \cos(t)\right)/(2x) \\ \left(-2 + y^2 - \cos(\omega t)\right)/(2y) \end{pmatrix} - \begin{pmatrix} \sin(t)/(2x) \\ \omega\sin(\omega t)/(2y) \end{pmatrix}\end{aligned} \tag{2}$$

with initial conditions $\mathbf{q}(0) = (\sqrt{2}, \sqrt{3})^T$. The exact solution is

$$\mathbf{q}(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = \begin{pmatrix} \sqrt{1 + \cos(t)} \\ \sqrt{2 + \cos(\omega t)} \end{pmatrix}. \tag{3}$$

Only the constant $\omega$ is important in defining the physical scales of the problem; the additional constants $\Gamma, \varepsilon$ are important in determing the scales on which the numerical algorithm is stable. The two particular cases that we will later be interested in simulating are shown in figure 1.

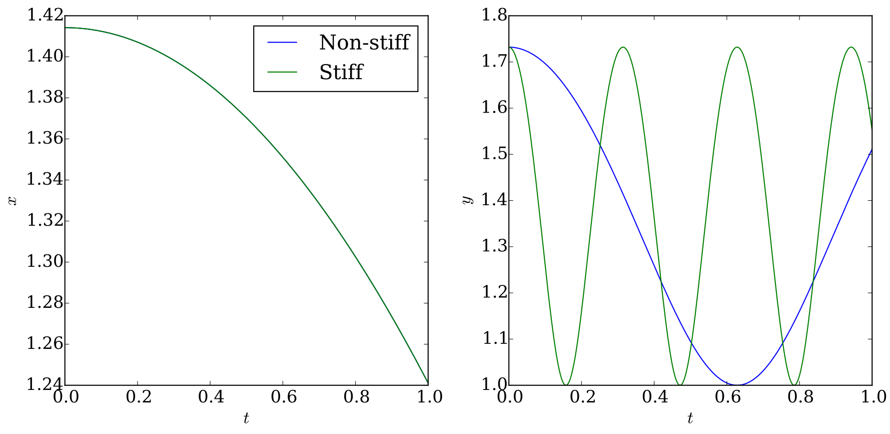

Figure 1: The exact solution for both variables in two cases: the "non-stiff" case ($\Gamma = -2, \omega = 5, \varepsilon = 0.05$), and the "stiff" case ($\Gamma = -2 \times 10^5, \omega = 20, \varepsilon = 0.5$).

## 2 Algorithms

### 2.1 Explicit algorithm

The standard explict third order Runge-Kutta method, written RK3, is to be implemented here. Given initial data $\mathbf{q}_n$ for the ODE problem described by equation (1) at location $t_n$, using an evenly spaced grid with spacing $\Delta t$ (so that $t_{n+1} = t_n + \Delta t$), the algorithm can be written

$$\mathbf{k}^{(1)} = \mathbf{f}(t_n, \mathbf{q}_n), \tag{4a}$$

$$\mathbf{k}^{(2)} = \mathbf{f}\left(t_n + \frac{1}{2}\Delta t, \mathbf{q}_n + \frac{\Delta t}{2}\mathbf{k}^{(1)}\right), \tag{4b}$$

$$\mathbf{k}^{(3)} = \mathbf{f}\left(t_n + \Delta t, \mathbf{q}_n + \Delta t\left[-\mathbf{k}^{(1)} + 2\mathbf{k}^{(2)}\right]\right), \tag{4c}$$

$$\mathbf{q}_{n+1} = \mathbf{q}_n + \frac{\Delta t}{6}\left(\mathbf{k}^{(1)} + 4\mathbf{k}^{(2)} + \mathbf{k}^{(3)}\right). \tag{4d}$$

### 2.2 Implicit algorithm

The implicit algorithm to be implemented is the third order accurate Gauss-Radau Runge-Kutta method, written GRRK3. Given initial data $\mathbf{q}_n$ for the ODE problem described by equation (1) at location $t_n$, using an evenly spaced grid with spacing $\Delta t$ (so that $t_{n+1} = t_n + \Delta t$), the algorithm can be written

$$\mathbf{k}^{(1)} = \mathbf{f}\left(t_n + \frac{1}{3}\Delta t, \mathbf{q}_n + \frac{\Delta t}{12}\left[5\mathbf{k}^{(1)} - \mathbf{k}^{(2)}\right]\right), \tag{5a}$$

$$\mathbf{k}^{(2)} = \mathbf{f}\left(t_n + \Delta t, \mathbf{q}_n + \frac{\Delta t}{4}\left[3\mathbf{k}^{(1)} + \mathbf{k}^{(2)}\right]\right), \tag{5b}$$

$$\mathbf{q}_{n+1} = \mathbf{q}_n + \frac{\Delta t}{4}\left(3\mathbf{k}^{(1)} + \mathbf{k}^{(2)}\right). \tag{5c}$$

We note that the definition of $\mathbf{k}^{(i)}$ is *implicit*, so finding the update terms will involve solving a nonlinear algebraic equation. The implementation must solve the nonlinear algebraic system defined by equations (5a-5b): with this solution the update defined in equation (5c) can be computed explicitly. To solve the implicit problem above we define

$$\mathbf{K} = \begin{pmatrix} \mathbf{k}^{(1)} \\ \mathbf{k}^{(2)} \end{pmatrix}. \tag{6}$$

We then write the problem to be solved as $\mathbf{F}(\mathbf{K}) = \mathbf{0}$, where

$$\mathbf{F} = \mathbf{K} - \begin{pmatrix} \mathbf{f}\left(t_n + \frac{1}{3}\Delta t, \mathbf{q}_n + \frac{\Delta t}{12}\left[5\mathbf{k}^{(1)} - \mathbf{k}^{(2)}\right]\right) \\ \mathbf{f}\left(t_n + \Delta t, \mathbf{q}_n + \frac{\Delta t}{4}\left[3\mathbf{k}^{(1)} + \mathbf{k}^{(2)}\right]\right) \end{pmatrix}. \tag{7}$$

Given some initial guess for $\mathbf{K}$, which corresponds to initial guesses for $\mathbf{k}^{(1,2)}$, this can be solved via the `fsolve` or `root` function from the `scipy` library. A simple initial guess for $\mathbf{K}$ sets $\mathbf{k}^{(1)} = \mathbf{f}(t_n + \Delta t/3, \mathbf{q}_n)$ and $\mathbf{k}^{(2)} = \mathbf{f}(t_n + \Delta t, \mathbf{q}_n)$.

## 3 Task

1. Implement the explicit RK3 method for a single step given in equation (4) as a function of the form

   ```
   qnew = MyRK3_step(f, t, qn, dt, options)
   ```

   The input arguments are a function `f` that defines the ODE, a number `t` giving the time $t_n$, a vector `qn` giving the data $\mathbf{q}_n$, a number `dt` giving the timestep $\Delta t$, and the `options` argument that may contain additional options (e.g. the parameters $\Gamma, \omega, \varepsilon$ defining the right hand side). The function `f` should take the form

   ```
   rhs = f(t, q, options)
   ```

   where the input is the time $t$ (which may vary inside the RK step), the data $\mathbf{q}$ at time $t$, and any additional options. The output should be the right hand side function $\mathbf{f}$ as defined in (2).

   For the function `MyRK3_step` the output is the approximate solution $\mathbf{q}_{n+1}$.

   The input to the function should be carefully checked, and the function fully documented.

2. Implement the implicit Gauss-Radau algorithm given in equation (5) as a function of the form

   ```
   qnew = MyGRRK3_step(f, t, qn, dt, options)
   ```

   The input and output arguments follow the same form as for the RK3 algorithm. Again the function should carefully check its input and should be fully documented. When solving the nonlinear systems in equations (5a–5b) use the inbuilt `fsolve` function or similar. This may send warnings to the screen even when correctly implemented – these warnings can be suppressed using optional arguments to `fsolve`.

3. Apply your RK3 and GRRK3 algorithms to the system of equations (2) with the "non-stiff" values $\Gamma = -2, \omega = 5, \varepsilon = 0.05$. Use $\Delta t = 0.05$, find the solution on $t \in [0, 1]$. Plot the result of both algorithms, along with the exact solution, in two subplots of the same figure window (as in figure 1).

4. Show evidence of the convergence of your algorithms by calculating solutions for $\Delta t = \frac{0.1}{2^j}$ for $j = 0, \ldots, 7$. Given your solution, compute the 1-norm error *only* for $y(t)$ as

   $$\|\text{Error}\|_1 = \Delta t \sum_{j=1}^{N} \left| y_j - (y_{\text{exact}})_j \right| . \tag{8}$$

   Plot the error against $\Delta t$, using an appropriate scale. By fitting an appropriate curve to the data using `polyfit`, which should also be plotted, give evidence to show that the algorithm is converging at third order.

5. Apply your RK3 algorithm to the system (2) using the "stiff" values $\Gamma = -2 \times 10^5, \omega = 20, \varepsilon = 0.5$ using a timestep of $\Delta t = 0.001$. Plot the numerical and exact solutions as functions of $t$, on a scale such that the features of the exact solution can be seen, showing that the RK3 is unstable for this timestep on this problem.

6. Apply your GRRK3 algorithm to the "stiff" system above, using a timestep of $\Delta t = 0.005$. Plot the numerical and exact solutions as functions of $t$, showing that the GRRK3 algorithm is stable and accurate.

7. Show evidence of the convergence of your GRRK3 algorithm by calculating solutions for $\Delta t = \frac{0.05}{2^j}$ for $j = 0, \dots, 7$, producing a similar plot to the non-stiff case above.

## 3.1 Summary and assessment criteria

You should submit the code electronically as noted above.

### 3.1.1 Python

As Python allows for many function definitions in one file, you may submit as many Python files as necessary to solve the problem. However, one file should clearly be the master script that when run (eg from the console inside spyder), produces all the plots required.

### 3.1.2 Plots

The code you submit is expected to produce 5 (five) figures:

1. For the non-stiff case, one figure with two subplots showing the exact and numerical solutions for each component as a function of $t$;

2. For the non-stiff case, one figure showing evidence of the convergence rate for both algorithms;

3. For the stiff case, one figure with two subplots showing that RK3 is unstable with the specified timestep;

4. For the stiff case, one figure with two subplots showing that GRRK3 is stable with the specified timestep;

5. For the stiff case, one figure showing evidence of the convergence rate for GRRK3.

Plots should be displayed to the screen.

#### 3.1.3 Assessment criteria

The primary assessment criteria will be the correct implementation of the RK3 and GRRK3 algorithms. The correctness of the algorithms must be clear, and must be demonstrated by completing the tests shown.

The secondary assessment criteria will be the clarity and robustness of your code. Your functions and scripts must be well documented with appropriate internal comments describing inputs, outputs, what the function does and how it does it. The code should also be well structured to make it easy to follow. Input must be checked and sensible error messages given when problems are encountered. Unit tests of individual functions may also be used to show the correctness of the code. The clarity of the output (such as plots or results printed to the command window) is also important.

Code efficiency is not important unless the algorithm takes an exceptional amount of time to run. As an indication, the model implementation runs all tasks in a maximum of 2 minutes.