Coursework 1: MATH 3018/6141 - Numerical methods

Due: 22 November 2017

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 22 November 2017 (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 transients

The ability to solve Initial Value Problems (IVPs), which we write in the form

$$\mathbf{y}' = \mathbf{f}(x, \mathbf{y}), \tag{1}$$

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.

An example is the system

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix}' = \begin{pmatrix} -1000y_1 \\ 1000y_1 - y_2 \end{pmatrix}, \tag{2}$$

for which one solution is

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} e^{-1000x} \\ \frac{1000}{999}\left(e^{-x} - e^{-1000x}\right) \end{pmatrix}. \tag{3}$$

As shown by figure 1, initially y_2 jumps very rapidly from 0 to 1 before decaying very slowly. Standard explicit algorithms need to restrict to very small step-lengths to be stable when dealing with the initial behaviour, but this is a waste of resources for the slow decay.

Here we will simplify the problem to equations of the form

$$\mathbf{y}' = A\mathbf{y} + \mathbf{b}(x), \tag{4}$$

where the $n \times n$ matrix A has constant coefficients and the vector $\mathbf{b}$ does not depend on the solution $\mathbf{y}$, but *does* depend on the independent variable x. This means that the implicit steps in the algorithm require the solution of a linear system of equations instead of a nonlinear system.

The example above, when written in this form, is defined by

$$A = \begin{pmatrix} -a_1 & 0 \\ a_1 & -a_2 \end{pmatrix}, \qquad \mathbf{b} = \mathbf{0}. \tag{5}$$

With the initial data

$$\mathbf{y}_0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \tag{6}$$

this has the solution

$$\mathbf{y} = \begin{pmatrix} e^{-a_1 x} \\ \frac{a_1}{a_1 - a_2}\left(e^{-a_2 x} - e^{-a_1 x}\right) \end{pmatrix}. \tag{7}$$

Stiff behaviour occurs when $a_1 \gg a_2 > 0$.

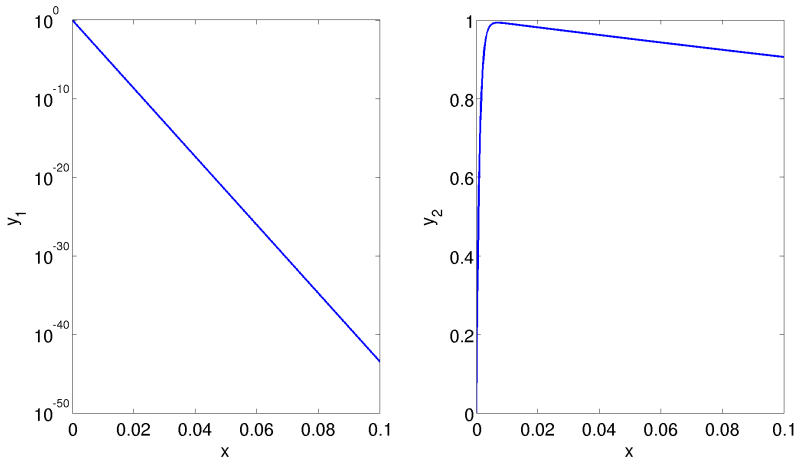

Figure 1: The solution for the system of equation (2) given by equation (3). Note the different scales required to show the rapid decay of y_1 and the slow decay of y_2. Note also the initial rapid transient behaviour of y_2.

2 Algorithms

2.1 Explicit algorithm

The standard explict third order Runge-Kutta method, written RK3, is to be implemented here. Given initial data $\mathbf{y}_n$ for the ODE problem described by equation (4) at location x_n, using an evenly spaced grid with spacing h (so that $x_{n+1} = x_n + h$), the algorithm can be written

$$\mathbf{y}^{(1)} = \mathbf{y}_n + h\left[A\mathbf{y}_n + \mathbf{b}(x_n)\right], \tag{8a}$$

$$\mathbf{y}^{(2)} = \tfrac{3}{4}\mathbf{y}_n + \tfrac{1}{4}\mathbf{y}^{(1)} + \tfrac{1}{4}h\left[A\mathbf{y}^{(1)} + \mathbf{b}(x_n + h)\right], \tag{8b}$$

$$\mathbf{y}_{n+1} = \tfrac{1}{3}\mathbf{y}_n + \tfrac{2}{3}\mathbf{y}^{(2)} + \tfrac{2}{3}h\left[A\mathbf{y}^{(2)} + \mathbf{b}(x_n + h)\right]. \tag{8c}$$

2.2 Implicit algorithm

The algorithm to be implemented is the optimal two stage third order accurate Diagonally Implicit Runge-Kutta method, written DIRK3. Given initial data $\mathbf{y}_n$ for the ODE problem described by equation (4) at location x_n, using an evenly spaced grid with spacing h (so that $x_{n+1} = x_n + h$), the algorithm can be written

$$\left[I - h\mu A\right]\mathbf{y}^{(1)} = \mathbf{y}_n + h\mu\mathbf{b}(x_n + h\mu), \tag{9a}$$

$$\left[I - h\mu A\right]\mathbf{y}^{(2)} = \mathbf{y}^{(1)} + h\nu\left[A\mathbf{y}^{(1)} + \mathbf{b}(x_n + h\mu)\right] + h\mu\mathbf{b}(x_n + h\nu + 2h\mu), \tag{9b}$$

$$\mathbf{y}_{n+1} = (1-\lambda)\mathbf{y}_n + \lambda\mathbf{y}^{(2)} + h\gamma\left[A\mathbf{y}^{(2)} + \mathbf{b}(x_n + h\nu + 2h\mu)\right], \qquad (9c)$$

where the required coefficients are

$$\mu = \tfrac{1}{2}\left(1 - \frac{1}{\sqrt{3}}\right), \qquad (10a)$$

$$\nu = \tfrac{1}{2}\left(\sqrt{3} - 1\right), \qquad (10b)$$

$$\gamma = \frac{3}{2\left(3+\sqrt{3}\right)}, \qquad (10c)$$

$$\lambda = \frac{3\left(1+\sqrt{3}\right)}{2\left(3+\sqrt{3}\right)}. \qquad (10d)$$

3 Task

1. Implement the explicit RK3 method given in equation (8) as a Python function

   `x, y = rk3(A, bvector, y0, interval, N)`

   The input arguments are a matrix `A`, a function `bvector`, a list `interval` giving the start and end values of $x = [x_0, x_{\text{end}}]$ on which the solution is given, the n-vector of initial data $\mathbf{y}_0 \equiv \mathbf{y}(x_0)$, and the number of steps N that the algorithm should take (so $h = (x_{\text{end}} - x_0)/N$). The function `bvector` should take the form

   `b = bvector(x)`

   where the input is the location x (which may vary inside the RK step), and the output is the vector $\mathbf{b}$ that is needed, together with the matrix `A` to define the system, as given in equation (4).

   The first output argument `x` is the locations x_j at which the solution is evaluated; this should be a real vector of length $N+1$ covering the required interval. The second output argument `y` should be the numerical solution approximated at the locations x_j, which will be an array of size $n \times (N+1)$.

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

2. Implement the implicit DIRK3 algorithm given in equation (9) as a Python function

   `x, y = dirk3(A, bvector, y0, interval, N)`

   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 linear systems in equations (9a–9b) use the in-built `numpy` solvers.

3. Apply your RK3 and DIRK3 algorithms to the system of equations (5) with the explicit values $a_1 = 1000, a_2 = 1$ as in figure 1, using the initial data of

equation (6) over the interval $x \in [0, 0.1]$. This will require defining a function to provide the (trivial) values of $\mathbf{b}$. Set N equal to $40k$ with $k = 1, \ldots, 10$. Compute the 1-norm of the relative error in the component y_2 by comparing to the exact solution in equation (7) (for $x \neq 0$) as

$$\|\text{Error}\|_1 = h \sum_{j=2}^{N} \left| \frac{(y_2)_j - ((y_2)_{\text{exact}})_j}{((y_2)_{\text{exact}})_j} \right|. \tag{11}$$

Plot the error against h, using an appropriate scale. By fitting an appropriate curve to the data using `numpy.polyfit`, which should also be plotted, give evidence to show that the algorithm is converging at third order. In addition plot the solution computed with the highest resolution and the exact solution against x on appropriate scales (in one figure for each algorithm, but different variables in subplots).

4. Define a new system using the matrix

$$A = \begin{pmatrix} -1 & 0 & 0 \\ -99 & -100 & 0 \\ -10\,098 & 9\,900 & -10\,000 \end{pmatrix} \tag{12}$$

and with the function $\mathbf{b}$ taking the values

$$\mathbf{b} = \begin{pmatrix} \cos(10x) - 10\sin(10x) \\ 199\cos(10x) - 10\sin(10x) \\ 208\cos(10x) + 10\,000\sin(10x) \end{pmatrix}. \tag{13}$$

With the initial data

$$\mathbf{y}_0 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \tag{14}$$

this system has the exact solution

$$\mathbf{y} = \begin{pmatrix} \cos(10x) - e^{-x} \\ \cos(10x) + e^{-x} - e^{-100x} \\ \sin(10x) + 2e^{-x} - e^{-100x} - e^{-10\,000x} \end{pmatrix}. \tag{15}$$

Then apply both the RK3 and DIRK3 algorithms to this system over the interval $x \in [0, 1]$ using $N = 200k$ with $k = 4, \ldots, 16$. Again plot the error against h, showing evidence of the convergence rate for DIRK3 only, and plot the solutions for both algorithms computed at the highest resolution. All components of $\mathbf{y}$ should be plotted against the exact solution, none on logarithmic scales. When computing the error, only the y_3 component should be used.

3.1 Summary and assessment criteria

You should submit the Python code electronically as noted above.

You are expected to submit all the Python code needed to produce the required output. Ideally there should be a top-level `Coursework1.py` script that, when run, produces all output. It is possible to complete the coursework by submitting a single file, but if you wish to use more files that is fine, provided all are submitted.

The script should produce 7 (seven) plots: for each system (the moderately stiff case in task 3 and the stiff case in task 4), and for each algorithm (RK3 and DIRK3), two plots are required. The first plot should show the behaviour of the errors against h and should be annotated to show the convergence rate where appropriate (this plot is not required for the RK3 algorithm in the stiff case). The second plot should show the exact and numerical solutions for each component on appropriate scales in individual subplots (e.g., as in figure 1). All plots should be suitably labelled and clear.

The primary assessment criteria will be the correct implementation of the RK3 and DIRK3 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 docstrings and 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. 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.

For those interested in applications of these techniques to current problems, the review of Gottlieb et al. includes a broad range of examples, and the paper of Pareschi and Russo gives a specific example where DIRK methods are important.