APPM 5610 - Homework 6

Eappen Nelluvelil

March 16, 2022

1. Implement the trapezoidal rule to solve the initial-value problem

$$\mathbf{y}' = \mathbf{f}\left(t, \mathbf{y}\right),$$

where $\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$, $\mathbf{f}\left(t, \mathbf{y}\right) = \begin{bmatrix} f_1\left(t, \mathbf{y}\right) \\ f_2\left(t, \mathbf{y}\right) \end{bmatrix}$, and $\mathbf{y}\left(0\right) = \mathbf{y}_0$. Use repeated Richardson extrapolation to improve the results.

Use your code to solve

$$\begin{aligned} t^2 y'' + ty' + \left(t^2 - 1\right) y &= 0, \\ y\left(0\right) &= 0, \\ y'\left(0\right) &= \frac{1}{2} \end{aligned}$$

on the interval $[0, 3\pi]$. Use repeated Richardson extrapolation to compute $y\left(3\pi\right)$ to 10 accurate digits.

Hint: For constructing the first-order system, determine the first few terms of the Taylor expansion of the solution $y\left(t\right) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + \mathcal{O}\left(t^4\right)$, and then substitute $y\left(t\right) = tu\left(t\right)$ to obtain the first-order system for u. The exact solution is $J_1\left(t\right)$, the Bessel function of the first kind of order 1.

Using Mathematica, we obtain the third-order Taylor expansion of the solution about $t = 0$:

$$y\left(t\right) \sim \frac{1}{2}t - \frac{1}{16}t^3 + \mathcal{O}\left(t^4\right).$$

From the substitution $y\left(t\right) = tu\left(t\right)$, we deduce that $u\left(t\right) \sim \frac{1}{2} - \frac{1}{16}t^2 + \mathcal{O}\left(t^3\right)$.

Using the above substitution, we compute the derivatives of y:

$$\begin{aligned} y'\left(t\right) &= u\left(t\right) + tu'\left(t\right), \\ y''\left(t\right) &= 2u'\left(t\right) + tu''\left(t\right). \end{aligned}$$

We then transform the original ODE to be in terms of u:

$$\begin{aligned} t^2\left(2u'\left(t\right) + tu''\left(t\right)\right) + t\left(u\left(t\right) + tu'\left(t\right)\right) + \left(t^2 - 1\right) tu\left(t\right) = 0 &\implies t^3 u''\left(t\right) + 3t^2 u'\left(t\right) + t^3 u\left(t\right) = 0 \\ &\implies u''\left(t\right) = -\frac{3}{t}u'\left(t\right) - u\left(t\right). \end{aligned}$$

Introducing the auxiliary variable $v\left(t\right) = u'\left(t\right)$, we can write the above transformed second-order ODE as a system of first-order ODEs:

$$\begin{bmatrix} u'\left(t\right) \\ v'\left(t\right) \end{bmatrix} = \begin{bmatrix} v\left(t\right) \\ -\frac{3}{t}v\left(t\right) - u\left(t\right) \end{bmatrix}, \quad \begin{bmatrix} u\left(0\right) \\ v\left(0\right) \end{bmatrix} = \begin{bmatrix} \frac{1}{2} \\ 0 \end{bmatrix}.$$

The above first-order system poses a problem at the initial time $t_0 = 0$ due to a division by zero in the second equation. However, we can remedy this by differentiating the ODE in u, $t^3 u''(t) + 3t^2 u'(t) + t^3 u(t) = 0$, once and evaluating the resulting ODE at $t = 0$:

$$\begin{aligned} t^3 u''(t) + 3t^2 u'(t) + t^3 u(t) = 0 &\implies t u''(t) + 3u'(t) + t u(t) = 0 \\ &\implies t u^{(3)}(t) + 4u''(t) + t u'(t) + u(t) = 0 \\ &\implies 4u''(0) + u(0) = 0 \\ &\implies u''(0) = -\frac{u(0)}{4} = -\frac{1}{8}. \end{aligned}$$

We arrive at the following first-order system:

$$\begin{bmatrix} u'(t) \\ v'(t) \end{bmatrix} = \begin{bmatrix} v(t) \\ -\frac{3}{t} v(t) - u(t) \end{bmatrix}, \quad \begin{bmatrix} u(0) \\ v(0) \end{bmatrix} = \begin{bmatrix} \frac{1}{2} \\ 0 \end{bmatrix}, \quad \begin{bmatrix} u'(0) \\ v'(0) \end{bmatrix} = \begin{bmatrix} 0 \\ -\frac{1}{8} \end{bmatrix}.$$

We now wish to solve the above system using the trapezoidal method, which is an implicit method. Recall that the trapezoidal method is given by

$$\mathbf{u}_{n+1} = \mathbf{u}_n + \frac{h}{2} \left(\mathbf{f}(t_n, \mathbf{u}_n) + \mathbf{f}(t_{n+1}, \mathbf{u}_{n+1}) \right),$$

where $\mathbf{u}_n \approx \begin{bmatrix} u(t_n) \\ v(t_n) \end{bmatrix}$ and $\mathbf{f}(t_n, \mathbf{u}_n) \approx \begin{bmatrix} v(t_n) \\ -\frac{3}{t_n} v(t_n) - u(t_n) \end{bmatrix}$. Using our knowledge about the system, we rearrange the trapezoidal method to solve for $\mathbf{u}_{n+1}$:

$$\begin{bmatrix} 1 & -\frac{h}{2} \\ \frac{h}{2} & \left(1 + \frac{3h}{2t_{n+1}}\right) \end{bmatrix} \mathbf{u}_{n+1} = \mathbf{u}_n + \frac{h}{2} \mathbf{f}(t_n, \mathbf{u}_n)$$

At each time step, we solve the above system for $\mathbf{u}_{n+1}$, and apply the trapezoidal rule with $\mathbf{u}_n$ and $\mathbf{u}_{n+1}$.

To refine the accuracy of the computed solution, we apply Richardson extrapolation. Upon applying the extrapolation procedure to the numerical approximation of $u(3\pi)$, we get that $y(\pi) = 3\pi u(3\pi) \approx 0.1767252152275255$.

2. Show that the two-step method

$$\mathbf{y}_{n+1} = \frac{1}{2} \left(\mathbf{y}_n + \mathbf{y}_{n-1} \right) + \frac{h}{4} \left(4\mathbf{f}\left(t_{n+1}, \mathbf{y}_{n+1}\right) - \mathbf{f}(t_n, \mathbf{y}_n) + 3\mathbf{f}\left(t_{n-1}, \mathbf{y}_{n-1}\right) \right)$$

is second order.

To show that the above two-step method is second order, we have to show that it satisfies the conditions:

$$\begin{aligned} \sum_{m=0}^{2} a_m &= 0, \\ \sum_{m=0}^{2} m^k a_m &= k \sum_{m=0}^{2} m^{k-1} b_m, \quad k = 1, 2 \\ \sum_{m=0}^{2} m^3 a_m &\neq 3 \sum_{m=0}^{2} m^2 b_m, \end{aligned}$$

where $a_0 = -\frac{1}{2}$, $a_1 = -\frac{1}{2}$, $a_2 = 1$, $b_0 = \frac{3}{4}$, $b_1 = -\frac{1}{4}$, and $b_2 = 1$.

We see that $a_0 + a_1 + a_2 = -\frac{1}{2} - \frac{1}{2} + 1 = 0$, so the first condition is satisfied. We now check the remaining conditions:

(a) $k = 1$

We see that

$$\begin{aligned}(1)^1 a_1 + (2)^1 a_2 &= -\frac{1}{2} + 2 = \frac{3}{2},\\ 1\left((0)^0 b_0 + (1)^0 b_1 + (2)^2 b_2\right) &= \frac{3}{4} - \frac{1}{4} + 1 = \frac{3}{2}.\end{aligned}$$

(b) $k = 2$

We see that

$$\begin{aligned}(1)^2 a_1 + (2)^2 a_2 &= -\frac{1}{2} + 4 = \frac{7}{2},\\ 2\left((0)^1 b_0 + (1)^1 b_1 + (2)^1 b_2\right) &= -\frac{1}{2} + 4 = \frac{7}{2}.\end{aligned}$$

(c) $k = 3$

We see that

$$\begin{aligned}(1)^3 a_1 + (2)^3 a_2 &= -\frac{1}{2} + 8 = \frac{15}{2},\\ 3\left((0)^2 b_0 + (1)^2 b_1 + (2)^2 b_2\right) &= -\frac{3}{4} + 6 = \frac{23}{4}.\end{aligned}$$

The sufficient order conditions are satisfied, so the above method is second order.

3. Determine the order of the multistep method

$$\mathbf{y}_{n+1} = 4\mathbf{y}_n - 3\mathbf{y}_{n-1} - 2h\mathbf{f}\left(t_{n-1}, \mathbf{y}_{n-1}\right),$$

and illustrate with an example that the method is unstable.

To show that the above two-step method is second order, we have to show that it satisfies the conditions:

$$\begin{aligned}\sum_{m=0}^{2} a_m &= 0,\\ \sum_{m=0}^{2} m^k a_m &= k \sum_{m=0}^{2} m^{k-1} b_m, \quad k = 1, 2\\ \sum_{m=0}^{2} m^3 a_m &\neq 3 \sum_{m=0}^{2} m^2 b_m,\end{aligned}$$

where $a_0 = 3$, $a_1 = -4$, $a_2 = 1$, $b_0 = -2$, $b_1 = 0$, and $b_2 = 0$.

We see that $a_0 + a_1 + a_2 = 3 - 4 + 1 = 0$, so the first condition is satisfied. We now check the remaining conditions:

(a) $k = 1$

We see that

$$\begin{aligned}(1)^1 a_1 + (2)^1 a_2 &= -4 + 2 = -2,\\ 1\left((0)^0 b_0 + (1)^0 b_1 + (2)^2 b_2\right) &= -2 = -2.\end{aligned}$$

(b) $k = 2$

We see that

$$\begin{aligned}(1)^2 a_1 + (2)^2 a_2 &= -4 + 4 = 0,\\ 2\left((0)^1 b_0 + (1)^1 b_1 + (2)^1 b_2\right) &= 0 + 0 = 0.\end{aligned}$$

(c) $k = 3$

We see that

$$\begin{aligned}(1)^3 a_1 + (2)^3 a_2 &= -4 + 8 = 4,\\ 3\left((0)^2 b_0 + (1)^2 b_1 + (2)^2 b_2\right) &= 0 + 0 = 0.\end{aligned}$$

The sufficient order conditions are satisfied, so the above method is second order. To demonstrate that this method is unstable, we consider the following first-order system of ODEs:

$$\begin{aligned}\mathbf{y}' &= \Lambda \mathbf{y},\\ \mathbf{y}(0) &= \mathbf{y}_0,\end{aligned}$$

where $\mathbf{\Lambda} = \begin{bmatrix} -100 & 1 \\ 0 & -\frac{1}{10} \end{bmatrix}$ and $\mathbf{y}_0 = \begin{bmatrix} 1 \\ \frac{999}{10} \end{bmatrix}$. The diagonalization of $\mathbf{\Lambda}$ is given by $\mathbf{\Lambda} = \mathbf{V}\mathbf{D}\mathbf{V}^{-1}$, where $\mathbf{V} = \begin{bmatrix} 1 & 1 \\ 0 & \frac{999}{10} \end{bmatrix}$ and $\mathbf{D} = \operatorname{diag}\left(-100, -\frac{1}{10}\right)$, and the exact solution to the above system is given by

$$\mathbf{y}(t) = \mathbf{V} e^{t\mathbf{D}} \mathbf{V}^{-1} \mathbf{y}_0.$$

We investigate the stability of the above method by implementing it in MATLAB, using the initial condition $\mathbf{y}_0 = \begin{bmatrix} 1 \\ \frac{999}{10} \end{bmatrix}$ and evolving the solution to $t = 1$. The numerical solution, found using a step size of $h = \frac{1}{10}$, is given by $\mathbf{y}_{\text{numerical}} \sim 10^3 \times \begin{bmatrix} -0.012 \\ -1.26 \end{bmatrix}$, but the actual solution is given by $\mathbf{y}_{\text{actual}} \sim \begin{bmatrix} 0.90 \\ 90.39 \end{bmatrix}$.

4. Show that the multistep method

$$\mathbf{y}_{n+3} + a_2 \mathbf{y}_{n+2} + a_1 \mathbf{y}_{n+1} + a_0 \mathbf{y}_n = h\left(b_2 \mathbf{f}\left(t_{n+2}, \mathbf{y}_{n+2}\right) + b_1 \mathbf{f}\left(t_{n+1}, \mathbf{y}_{n+1}\right) + b_0 \mathbf{f}\left(t_n, \mathbf{y}_n\right)\right)$$

is fourth-order if and only if $a_0 + a_2 = 8$ and $a_1 = -9$. Deduce that this method cannot be both fourth-order and convergent (this is problem 2.6 in Iserles).

$(\longrightarrow)$ Suppose that the above-method is fourth-order. We wish to show that this implies that $a_0 + a_2 = 8$ and $a_1 = -9$. Per Iserles, for this method to be fourth-order, the coefficients $a_0, \ldots, a_3$ and $b_0, \ldots, b_2$ have to satisfy the following conditions:

$$\begin{aligned}\sum_{m=0}^{3} a_m &= 0,\\ \sum_{m=0}^{3} m^k a_m &= k \sum_{m=0}^{2} m^{k-1} b_m, \quad k = 1, 2, 3, 4\\ \sum_{m=0}^{3} m^5 a_m &\neq 5 \sum_{m=0}^{2} m^4 b_m.\end{aligned}$$

We can arrange the above conditions as an augmented matrix system, where the unknowns are $\mathbf{x} = \begin{bmatrix} a_0 & a_1 & a_2 & b_0 & b_1 & b_2 \end{bmatrix}^T$:

$$\left[\begin{array}{cccccc|c} 1 & 1 & 1 & 0 & 0 & 0 & -1 \\ 0 & 1 & 2 & -1 & -1 & -1 & -3 \\ 0 & 1 & 4 & 0 & -2 & -4 & -9 \\ 0 & 1 & 8 & 0 & -3 & -12 & -27 \\ 0 & 1 & 16 & 0 & -4 & -32 & -81 \end{array}\right] \sim \left[\begin{array}{cccccc|c} 1 & 0 & 0 & 0 & 0 & 3 & 17 \\ 0 & 1 & 0 & 0 & 0 & 0 & -9 \\ 0 & 0 & 1 & 0 & 0 & -3 & -9 \\ 0 & 0 & 0 & 1 & 0 & -1 & -6 \\ 0 & 0 & 0 & 0 & 1 & -4 & -18 \end{array}\right].$$

From adding the first and second rows to each other, we get that $a_0 + a_2 = 8$, and from the third equation, we get that $a_1 = -9$.

The Dahlquist first barrier states that the maximal order of a convergent s-step method is just s for explicit methods. The method at hand is a 3-step explicit method, so the maximal order of this method is 3. This means that if the above method is a 4^{th} order method, then it cannot also be convergent.

```matlab
% Problem 1
clc;
clear;

% Initial conditions
y_0 = [1/2; 0];
t_0 = 0;
t_f = 3*acos(-1);

% Testing purposes - make sure the trapezoidal method returns a numerical
% value that's close to the actual solution
% n = 10000;
% h = (t_f-t_0)/n;
% y = trapezoid_2(@odefun, t_0, t_f, h, y_0);
% fprintf("Absolute error between numerical approximation and actual solution:
 %0.16e\n", ...
%         abs(besselj(1, t_f) - t_f*y(1)));

% Use repeated Richardson extrapolation to obtain a numerical solution
% that's accurate to 10 digits of the actual answer
actual_soln = besselj(1, t_f);
max_rows = 20;
tol = 1e-11;
h = (t_f - t_0);

A = zeros(max_rows, max_rows);
y = trapezoid_2(@odefun, t_0, t_f, h, y_0);
A(1, 1) = y(1);

for i = 1:(max_rows - 1)
    h = h/2;

    y = trapezoid_2(@odefun, t_0, t_f, h, y_0);
    A(i + 1, 1) = y(1);

    for j = 1:i
        A(i + 1, j + 1) = ((4^j)*A(i + 1, j) - A(i, j))/((4^j) - 1);
    end

    if abs(A(i + 1, i + 1) - A(i, i)) < tol
        break;
    end
end

num_approx = t_f*A(max_rows, max_rows);
abs_err    = abs(num_approx - actual_soln);
rel_err    = abs_err/abs(actual_soln);

fprintf("Numerical approximation of solution: %0.16f\n", num_approx);
% fprintf("Absolute error between numerical approximation computed via
 Richardson extrapolation and actual solution: %0.16e\n", ...
%         abs_err);
```

```
% fprintf("Relative error between numerical approximation computed via
 Richardson extrapolation and actual solution: %0.16e\n", ...
%          rel_err);

function [f, J_f] = odefun(t, y)
    if norm(t) < 1e-10
        f = [0; (-1/8)];
    else
        f = [y(2); (-3./t).*y(2) - y(1)];
    end

    % Return the Jacobian of f if requested
    if nargout > 1
        J_f = [0, 1; ...
               -1, -3./t];
    end
end

function [y] = trapezoid(odefun, t_0, t_f, h, y_0)
    % Tolerance for Newton's method
    tol = 1e-10;
    y = y_0;

    while t_0 <= t_f
        % Compute the explicit part of the trapezoidal method
        E_y = y + (h/2)*odefun(t_0, y);

        % Perform one step of forward Euler to kick off Newton's method
        FE_y = y + h*odefun(t_0, y);

        % Apply Newton's method to calculate the implicit part of the
        % trapezoidal method
        while true
            [f, J_f] = odefun(t_0 + h, y);
            update = ((h/2)*J_f - eye(size(J_f)))\(E_y + (h/2)*f - FE_y);
            FE_y = FE_y - update;
            if norm(update) < tol
                break;
            end
        end

        % Compute y_{n+1} using the explicit and implicit parts of the
        % trapezoidal method
        y = E_y + (h/2)*odefun(t_0 + h, FE_y);

        t_0 = t_0 + h;
    end
end

function [y] = trapezoid_2(odefun, t_0, t_f, h, y_0)
    y = y_0;

    while t_0 <= t_f
        E_y = y + (h/2)*odefun(t_0, y);
```

```
        I_m = [1, -h/2; ...
               h/2, 1 + (3/2)*h/(t_0 + h)];
        I_y = I_m\E_y;

        y = E_y + (h/2)*odefun(t_0 + h, I_y);
        t_0 = t_0 + h;
    end
end
```

```
Numerical approximation of solution: 0.1767252152275348
```

Published with MATLAB® R2022a

```matlab
clc;
clear;

t_0 = 0;
t_f = 1;
h = (t_f - t_0)/10;
y_0 = [1; 999/10];

y = method(@odefun, t_0, t_f, h, y_0);

V = [1, 1; ...
     0, 999/10];
D = [-100, 0; ...
     0, -1/10];

y_actual = @(t, y_0) V*expm(t*D)*inv(V)*y_0;

disp(y);
disp(y_actual(t_f, y_0));

function [f] = odefun(t, y)
    f = [-100, 1; ...
         0, (-1/10)]*y;
end

function [y] = method(odefun, t_0, t_f, h, y_0)
    y0 = y_0;

    % Compute y_1 using one iteration of forward Euler
    y1 = y0 + h*odefun(t_0, y0);

    while t_0 <= t_f
        y2 = 4*y1 - 3*y0 - 2*h*odefun(t_0, y0);
        y0 = y1;
        y1 = y2;
        t_0 = t_0 + h;
    end

    y = y1;
end
```

```
   1.0e+03 *

  -0.012666247885568
  -1.265358189337219

   0.904837418035960
  90.393258061792366
```

Published with MATLAB® R2022a