MATH7003-00: Assignment #5

Sangman Jung
e-mail: sangmanjung@khu.ac.kr
Kyung Hee University – April 29, 2020

Problem 1. For the example on slide 25 of week5-1, generate the same Table 6.14. Discuss about the result. [1].

Example. Method (6.7.27) was used to solve

$$y' = \frac{1}{1+x^2} - 2y^2 \qquad y(0) = 0$$

which has the solution $Y(x) = x/(1+x^2)$. The initial values y_1, y_2, y_3 were taken to be the true values to simplify the example. The solution values were computed with two values of h, and the resulting errors at a few node points are given in Table 6.14. The column labeled Ratio is the ratio of the error with $h = .125$ to that with $h = .0625$. Note that the values of Ratio are near 16, which is the theoretical ratio that would be expected since method (6.7.27) is fourth order.

Table 6.14 Numerical example of the Adams method

x	Error for $h = .125$	Error for $h = .0625$	Ratio
2.0	2.07E − 5	1.21E − 6	17.1
4.0	2.21E − 6	1.20E − 7	18.3
6.0	3.74E − 7	2.00E − 8	18.7
8.0	1.00E − 7	5.24E − 9	19.1
10.0	3.58E − 8	1.83E − 9	19.6

Discussion. Method (6.7.27) in the textbook is the perdictor-corrector method that using Adams-Bashforth predictor, and Adams-Moulton corrector (fourth-order). It is given by

$$y_{n+1}^{(0)} = y_n + \frac{h}{12}\left[23f(x_n, y_n) - 16f(x_{n-1}, y_{n-1}) + 5f(x_{n-2}, y_{n-2})\right] \quad \text{(predictor)},$$

$$y_{n+1}^{(j+1)} = y_n + \frac{h}{24}\left[9f(x_{n+1}, y_{n+1}^{(j)}) + 19f(x_n, y_n) - 5f(x_{n-1}, y_{n-1}) + f(x_{n-2}, y_{n-2})\right] \quad \text{(corrector)}.$$

This is similar to the trapezoidal corrector with the Euler predictor. If the relation can be the second-order Adams-Moulton formula with the first-order Adams-Bashforth formula as predictor, then the trapezoidal formula with Euler is the same as the Moulton with the Bashforth.

Likewise the trapezoidal method case, the multi-step implicit method has the better approximation and the smaller truncation error. If we apply the higher order of it, we can reduce the error significantly. The fourth order Adams-Moulton with Adams-Bashforth is the example of that. Note that the principal reason for using implicit formulas is the reduction of the truncation error, but the formulas sometimes cause the large cost of computation.

Here is the result of the problem using MATLAB.

Table 6.14 Numerical example of the Adams method

x	Error for h = 0.125	Error for h = 0.0625	Ratio
2.0	2.07e-05	1.21e-06	17.2
4.0	2.21e-06	1.20e-07	18.3
6.0	3.74e-07	2.00e-08	18.7
8.0	1.00e-07	5.24e-09	19.1
10.0	3.58e-08	1.83e-09	19.6

fx >>

Figure 1. The result of the equation in the problem 1.

Problem 2. For the example on slide 30, and 37 in week6-1, generate the same Table 6.18. and 6.17. Discuss about the result. [1].

Example 2-1. Apply Euler's method to the problem

$$y' = \lambda y + (1-\lambda)\cos(x) - (1+\lambda)\sin(x) \qquad y(0) = 1$$

whose true solution is $Y(x) = \sin(x) + \cos(x)$. We give results for several values of λ and h. For $\lambda = -1, -10, -50$, the bound $|1 + h\lambda| < 1$ implies the respective bounds on h of

$$0 < h < 2 \quad 0 < h < \frac{1}{5} = .2 \quad 0 < h < \frac{1}{25} = .04.$$

The use of larger values of h gives poor numerical results, as seen in Table 6.17.

Table 6.17 Euler's method for (6.8.51)

λ	x	Error: $h = .5$	Error: $h = .1$	Error: $h = .01$
− 1	1	−2.46E − 1	−4.32E − 2	−4.22E − 3
	2	−2.55E − 1	−4.64E − 2	−4.55E − 3
	3	−2.66E − 2	−6.78E − 3	−7.22E − 4
	4	2.27E − 1	3.91E − 2	3.78E − 3
	5	2.72E − 1	4.91E − 2	4.81E − 3
− 10	1	3.98E − 1	−6.99E − 3	−6.99E − 4
	2	6.90E + 0	−2.90E − 3	−3.08E − 4
	3	1.11E + 2	3.86E − 3	3.64E − 4
	4	1.77E + 3	7.07E − 3	7.04E − 4
	5	2.83E + 4	3.78E − 3	3.97E − 4
− 50	1	3.26E + 0	1.06E + 3	−1.39E − 4
	2	1.88E + 3	1.11E + 9	−5.16E − 5
	3	1.08E + 6	1.17E + 15	8.25E − 5
	4	6.24E + 8	1.23E + 21	1.41E − 4
	5	3.59E + 11	1.28E + 27	7.00E − 5

Example 2-2. Apply the trapezoidal method to the problem above ((6.8.51)), which was solved earlier with Euler's method. We use a stepsize of $h = .5$ for $\lambda = -1, -10, -50$. The results are given in Table 6.18. They illustrate that the trapezoidal rule does not become unstalbe as $|\lambda|$ increases, while Real(λ)< 0.

Table 6.18 Example of trapezoidal rule: $h = .5$

x	Error: $\lambda = -1$	Error: $\lambda = -10$	Error: $\lambda = -50$
2	−1.13E − 2	−2.78E − 3	−7.91E − 4
4	−1.43E − 2	−8.91E − 5	−8.91E − 5
6	2.02E − 2	2.77E − 3	4.72E − 4
8	−2.86E − 3	−2.22E − 3	−5.11E − 4
10	−1.79E − 2	−9.23E − 4	−1.56E − 4

Discussion. The advantage of the explicit method is that it requires only two function evaluations pr. time step. The advantage of the implicit method is that it is **A-stable.** The disadvantage of the explicit method is that the stability region is small. The disadvantage of the implicit method is that typically requires more than two function evaluations per time step.

Note) A-stable

For example, there is a **parasitic solution** if h is not sufficiently small, in multi-step method (In this situation, we say that this method has the **weak stability**, <ex> midpoint method). This implies that the region of absolute stability of the method we choose is small. **A-stable** means that the stability of a numerical method ensure for all $Re(h\lambda) < 0$ in the test equation $y' = \lambda y$. the A-stable method has the benefits that there is no limitation of any h.

The stability of these method are studied by examining their behavior when applied to the simple test equation $y' = \lambda y$ .When the real part of λ is strictly less than zero, then the exact solution decays to zero as x tends to infinity. (The set of all $h\lambda$ for which this is true is called the **region of absolute stability** of the method. The larger this region, the less the restriction on h in order to hae a stable numerical solution.)

The implicit method will reproduce this behavior regardless of the time step h. The explicit method will only produce this behavior for sufficient small values of h. Using the implicit method requires less knowledge about the problem, but it will frequently take more time to obtain a solution. We can often reduce the time to solve the nonlinear equation by using the explicit method to generate a good initial guess.

The given equation in this problem is **stiff differential equation**. That equation have the form

$$Y'(x) = \lambda Y(x) + g(x)$$

with λ negative but large in magnitude. This equation is characterized by the property that when it is solved by a numerical scheme without absolute stability, the stepsize h is usually forced to be excessively small in order for the scheme to generate reasonable approximate solution [3]. This problem represent the restriction of the size of h in using Euler method to solve this stiff equation. Thus, for

stiff differential equations, one must use a numerical method that is absolutely stable or has a large region of absolute stability. Usually, implicit methods are preferred to the explicit ones in solving stiff differential equations.

The result of the first example using MATLAB is as follows

Table 6.17 Euler's method for (6.8.51)

lambda	x	Error: h = 0.5	Error: h = 0.1	Error: h = 0.01
-1	1	-2.46e-01	-4.32e-02	-4.22e-03
-1	2	-2.55e-01	-4.64e-02	-4.56e-03
-1	3	-2.66e-02	-6.79e-03	-7.03e-04
-1	4	+2.27e-01	+3.91e-02	+3.80e-03
-1	5	+2.72e-01	+4.91e-02	+4.81e-03
-10	1	+3.98e-01	-6.99e-03	-6.99e-04
-10	2	+6.90e+00	-2.90e-03	-3.08e-04
-10	3	+1.11e+02	+3.86e-03	+3.66e-04
-10	4	+1.77e+03	+7.07e-03	+7.04e-04
-10	5	+2.83e+04	+3.78e-03	+3.94e-04
-50	1	+3.26e+00	+1.06e+03	-1.39e-04
-50	2	+1.88e+03	+1.11e+09	-5.15e-05
-50	3	+1.08e+06	+1.17e+15	+8.30e-05
-50	4	+6.24e+08	+1.23e+21	+1.41e-04
-50	5	+3.59e+11	+1.29e+27	+6.96e-05

fx >>

Figure 2. The result of the equation in the problem 2-1.

Table 6.18 Example of trapezoidal rule: h = 0.5

x	Error: lambda = -1	Error: lambda = -10	Error: lambda = -50
2.0	-1.13e-02	-2.78e-03	-7.91e-04
4.0	-1.43e-02	-8.91e-05	-8.91e-05
6.0	+2.02e-02	+2.77e-03	+4.72e-04
8.0	-2.86e-03	-2.22e-03	-5.11e-04
10.0	-1.79e-02	-9.23e-04	-1.56e-04

fx >>

Figure 3. The result of the equation in the problem 2-2.

In **Figure 2** and **Figure 3**, we can see the results that the trapezoidal method has the better approximation than Euler's method at $h = 0.5$.

Table_6_14_HW5.m

```
%% MATH7003-00: Assignment #5-(1), 2019310290 Sangman Jung
clear,clc

%%% fundamental setting
h = [0.125 0.0625]; % step size
f = @(x,y) (1./(1+x.^2))-2*y.^2; % differential equation
Y = @(x) x./(1+x.^2); % exact solution
x0 = 0; x_end = 10; y0 = 0; % interval and initial condition
fval = cell(1,length(h)); % save the values of y for y_0125 and y_00625
xval = cell(1,length(h)); % save the values of x for y_0125 and y_00625
num_x = zeros(1,2); % allocation of size of interval x

%%% Adams method iteration: using predictor-corrector method
for h_iter = 1:length(h)
    x = x0:h(h_iter):x_end; % interval x
    num_x(h_iter) = (x_end-x0)/h(h_iter); % interval size for each h
    y = zeros(1,num_x(h_iter)); % allocation of the solution y
    y_p = zeros(1,num_x(h_iter)); % allocation of the predictor
    % The initial values y1,y2,y3 were taken to be the true values to
    % simplify the example. <---- refer to the textbook in our class.
    y(1) = y0; y(2:3) = Y(x(2:3));
    % Adam-Bashforth predictor & Adam-Moulton corrector iteration
    for j = 3:num_x(h_iter)
        % predictor
        y(j+1) = y(j) + (h(h_iter)/12)*(23*f(x(j),y(j))-16*f(x(j-1),y(j-1))+5*f(x(j-2),y(j-2)));
        % corrector
        y(j+1) = y(j) + (h(h_iter)/24)*(9*f(x(j+1),y(j+1))+19*f(x(j),y(j))-5*f(x(j-1),y(j-1)) + f(x(j-2),y(j-2)));
    end
    % save the values
    fval(h_iter) = {y'};
    xval(h_iter) = {x'};
end

%%% print out setting
% numerical x, y for h = 0.125
x_0125 = cell2mat(xval(1));
y_0125 = cell2mat(fval(1));
% numerical x, y for h = 0.0625
x_00625 = cell2mat(xval(2));
y_00625 = cell2mat(fval(2));

% we need to represent the values in Table 6.14.
table = 2:2:10;
index_0125 = zeros(1,length(table)*2);
index_00625 = zeros(1,length(table)*2);
for k = table
    index_0125(k) = find(x_0125 == k);
    index_00625(k) = find(x_00625 == k);
end
index_0125 = nonzeros(index_0125)';
index_00625 = nonzeros(index_00625)';

% the error for each step size and the ratio of the error with two step sizes.
Error_125 = Y(x_0125(index_0125))-y_0125(index_0125);
Error_0625 = Y(x_00625(index_00625))-y_00625(index_00625);
Ratio = Error_125./Error_0625;

% print the Table 6.14
fprintf("Table 6.14  Numerical example of the Adams method\n");
fprintf("-------------------------------------------------------------------------\n");
fprintf("|   x   |   Error for h = 0.125   |   Error for h = 0.0625   |   Ratio   |\n");
fprintf("-------------------------------------------------------------------------\n");
for i = 1:length(table)
    fprintf('   %1.1f          %1.2e              %1.2e            %2.1f    \n',...
        abs([table(i) Error_125(i) Error_0625(i) Ratio(i)]));
end
fprintf("-------------------------------------------------------------------------\n");
```

Table_6_17_HW5.m

```
%% MATH7003-00: Assignment #5-(2-1), 2019310290 Sangman Jung.
clear,clc

%%% fundamental setting
h = [0.5 0.1 0.01]; % step size
lambda = [-1 -10 -50]; % parameters
% given the differential equation
f = @(x,y,lambda) lambda*y + (1-lambda)*cos(x) - (1+lambda)*sin(x);
Y = @(x) sin(x)+cos(x); % exact solution of y'

%%% Euler's method for the ODE :
% y' = lambda*y + (1-lambda)*cos(x) - (1+lambda)*sin(x);
for lam_iter = 1:3 % lambda : -1, -10, -50
    for h_iter = 1:3 % step size : 0.5, 0.1, 0.01
        y = []; % initialize
        y0 = 1; % initial condition
        x0 = 0; % initial x
        x = x0:h(h_iter):10; % variable x
        for j = 1:length(x)-1 % Euler's method loop
            y(j+1) = y0 + h(h_iter)*f(x0,y0,lambda(lam_iter)); % numerical scheme
            y0 = y(j+1); % update y
            x0 = x0 + h(h_iter); % update x per step size
        end
        fval(lam_iter,h_iter) = {y'}; % save the values of y for each lambda and step size
        xval(lam_iter,h_iter) = {x'}; % save the values of for each step size
    end
end

%%% print out setting
% load the value of x and y for each lambda and step size
x_lam1(:,[1 2 3]) = cell2mat(xval([1 2 3]));
x_lam2(:,[1 2 3]) = cell2mat(xval([4 5 6]));
x_lam3(:,[1 2 3]) = cell2mat(xval([7 8 9]));
y_lam1(:,[1 2 3]) = cell2mat(fval([1 2 3]));
y_lam2(:,[1 2 3]) = cell2mat(fval([4 5 6]));
y_lam3(:,[1 2 3]) = cell2mat(fval([7 8 9]));

% table view setting
table = 1:5;
for k = table
    index_h1(k) = find(x_lam1(:,1) == k);
    index_h2(k) = find(x_lam2(:,1) == k);
    index_h3(k) = find(x_lam3(:,1) == k);
end
index_h1 = nonzeros(index_h1)';
index_h2 = nonzeros(index_h2)';
index_h3 = nonzeros(index_h3)';

%%% output
Error_lam1 = Y(x_lam1(index_h1,:))-y_lam1(index_h1,:);
Error_lam2 = Y(x_lam2(index_h2,:))-y_lam2(index_h2,:);
Error_lam3 = Y(x_lam3(index_h3,:))-y_lam3(index_h3,:);

%%% print the Table 6.17
fprintf("Table 6.17  Euler's method for (6.8.51)\n");
fprintf("-------------------------------------------------------------------------------------\n");
fprintf("|   lambda   |   x   |   Error: h = 0.5   |   Error: h = 0.1   |   Error: h = 0.01   |\n");
fprintf("-------------------------------------------------------------------------------------\n");
for k = 1:length(table)
    fprintf('     %d        %d         %+1.2e          %+1.2e           %+1.2e\n',...
        [lambda(1) table(k) Error_lam1(k,1) Error_lam2(k,1) Error_lam3(k,1)]);
end
fprintf("-------------------------------------------------------------------------------------\n");
for k = 1:length(table)
    fprintf('     %d        %d         %+1.2e          %+1.2e           %+1.2e\n',...
        [lambda(2) table(k) Error_lam1(k,2) Error_lam2(k,2) Error_lam3(k,2)]);
end
fprintf("-------------------------------------------------------------------------------------\n");
for k = 1:length(table)
    fprintf('     %d        %d         %+1.2e          %+1.2e           %+1.2e\n',...
        [lambda(3) table(k) Error_lam1(k,3) Error_lam2(k,3) Error_lam3(k,3)]);
end
fprintf("-------------------------------------------------------------------------------------\n");
```

Table_6_18_HW5.m

```
%% MATH7003-00: Assignment #5-(2-2), 2019310290 Sangman Jung.
clear,clc

%%% fundamental setting
h = 0.5; % step size
lambda = [-1 -10 -50]; % parameters
% given the differential equation
f = @(x,y,lambda) lambda*y + (1-lambda)*cos(x) - (1+lambda)*sin(x);
Y = @(x) sin(x)+cos(x); % exact solution of y'

%%% trapezoidal method iteration
for lam_iter = 1:3 % lambda = -1, -10, -50
    y0 = 1; % initial condition
    x0 = 0; % initial x
    x = x0:h:10; % variable x
    num_x = (10-x0)/h;
    y = zeros(1,num_x);
    y(1) = y0;
    for j = 1:num_x % Trapezoidal method loop
        y(j+1) = ((1+(h/2)*lambda(lam_iter))*y(j)+h/2*((1-lambda(lam_iter))*cos(x(j))-...
            (1+lambda(lam_iter))*sin(x(j))+(1-lambda(lam_iter))*cos(x(j+1))-...
            (1+lambda(lam_iter))*sin(x(j+1))))/(1-(h/2)*lambda(lam_iter)); % explicit version
    end
    fval(lam_iter) = {y'}; % save the values of y for each lambda
    xval(lam_iter) = {x'}; % save the values of x for each lambda
end

%%% print out setting
% load the value of x and y for each lambda
x_lam1 = cell2mat(xval(1)); % x for lambda = -1
x_lam2 = cell2mat(xval(2)); % x for lambda = -10
x_lam3 = cell2mat(xval(3)); % x for lambda = -50
y_lam1 = cell2mat(fval(1)); % numerical y for lambda = -1
y_lam2 = cell2mat(fval(2)); % numerical y for lambda = -10
y_lam3 = cell2mat(fval(3)); % numerical y for lambda = -50

%%% table view setting
table = 2:2:10;
for k = table
    index_lam1(k) = find(x_lam1 == k);
    index_lam2(k) = find(x_lam2 == k);
    index_lam3(k) = find(x_lam3 == k);
end
index_lam1 = nonzeros(index_lam1)';
index_lam2 = nonzeros(index_lam2)';
index_lam3 = nonzeros(index_lam3)';

%%% output
Error_lam1 = Y(x_lam1(index_lam1))-y_lam1(index_lam1);
Error_lam2 = Y(x_lam2(index_lam2))-y_lam2(index_lam2);
Error_lam3 = Y(x_lam3(index_lam3))-y_lam3(index_lam3);

%%% print the Table 6.18
fprintf("Table 6.18  Example of trapezoidal rule: h = 0.5\n");
fprintf("-------------------------------------------------------------------------------------\n");
fprintf("|   x   |   Error: lambda = -1   |   Error: lambda = -10   |   Error: lambda = -50   |\n");
fprintf("-------------------------------------------------------------------------------------\n");
for k = 1:length(table)
    fprintf('   %1.1f          %+1.2e              %+1.2e             %+1.2e\n',...
        [table(k) Error_lam1(k) Error_lam2(k) Error_lam3(k)]);
end
fprintf("-------------------------------------------------------------------------------------\n");
```

References.

[1] Atkinson, K. E. (2008). An introduction to numerical analysis. John wiley & sons.

[2] Atkinson, K., Han, W., & Stewart, D. E. (2011). Numerical solution of ordinary differential equations (Vol. 108). John Wiley & Sons.

[3] Atkinson, K. E., & Han, W. (1985). Elementary numerical analysis (p. 17). New York et al.: Wiley.