MATH 352, Spring 2021
Homework 1

Dr. Alessandro Veneziani

Kai Chang, Kevin Hong, Rodrigo Gonzalez

- A quick note at the very first place: we did not include the initial time or final time as arguments in any of the codes; instead, we assumed the initial time is zero and the final time T_{final} is 10 across the problem. Therefore, our arguments that manipulate time are the number of time steps (represented by n) and the size of time step (represented by Dt). We realized the inconvenience of this setting after we finished everything. However, since it does not affect the consistency for this specific problem, we did not change our codes.

- **Finding the exact solution**

  Consider the Cauchy Problem

$$\frac{d\vec{y}}{dt} = A\vec{y}$$

  with A an n by n matrix, and $\vec{y}(0) = \vec{y}_0$.

  We may assume that A is diagonalizable. That is, that there is a diagonal matrix D containing the eigenvalues of A, and an invertible matrix T containing the corresponding eigenvectors, such that $T^{-1}AT = D$.

  Then, we may multiply the equation by T^{-1}, and use the fact that $TT^{-1} = I$ to obtain:

$$\frac{dT^{-1}\vec{y}}{dt} = (T^{-1}AT)T^{-1}\vec{y}$$

  Let $\vec{z} = T^{-1}\vec{y}$ to get:

$$\frac{d\vec{z}}{dt} = D\vec{z}$$

  This system is easily solvable because that D is diagonal. In fact, the ith entry of $\vec{z}$ is given by $z_i = e^{\lambda_i t}(z_0)_i$, where λ_i is the ith entry of D. We may rewrite this in matrix notation:

$$\vec{z} = e^{Dt}\vec{z_0}$$

Here, e^{Dt} is notation for the diagonal matrix that contains $e^{\lambda_i t}$ in the *ith* diagonal entry. Then, we have:

$$\vec{y} = Te^{Dt}\vec{z_0}$$

To avoid computing T^{-1}, we recall that $\vec{z_0}$ is the solution to the system $T\vec{z_0} = \vec{y_0}$.

Code:

```
function [y] = exactSolution(A, y0)


%Convert sparse matrix to full storage
A = full(A);

%Diagonalize A
[T, D] = eig(A);

%Find z_0
z0 = T\y0;

%Build the exact solution y
y = @(t) T*diag(exp(diag(D)*t))*z0;
```

- **Analyzing the accuracy of these two methods**

  We may use *Theorem 1* to check consistency and determine the order of accuracy of the methods.

  - **AB2**

    We have $\vec{\alpha} = (1, 0)$, and $\vec{\beta} = (0, 3/2, -1/2)$. Then, we check:

$$\sum_{0}^{p} \alpha_j = 1 + 0 = 1$$

$$-\sum_{0}^{p} j\alpha_j + \sum_{-1}^{p} \beta_j = -(0)(1) - (1)(0) + 0 + 3/2 - 1/2 = 1$$

    We have consistency. We now find the order of the method.
    Let $k = 2$,

$$\sum_{0}^{p} (-j)^2 \alpha_j + (2) \sum_{-1}^{p} (-j)\beta_j = 0 + 2[(1)(0) + 0(3/2) + (-1)(-1/2)] = 1$$

    Let $k = 3$,

$$\sum_{0}^{p} (-j)^3 \alpha_j + (3) \sum_{-1}^{p} (-j)^2\beta_j = 0 + 3[(1)^2 0 + (0)^2(3/2) + (-1)^2(-1/2)] = 3/2 \neq 1$$

    Therefore, AB2 is of order 2.

    Code:

```
function [U, errors] = AB2(A, y0, y1, n, Dt, yex)
%
%   A:  the matrix
%   y0, y1: the 2 initial conditions
%   n: the # of time steps
%   Dt: time step
%   yex: the exact solution
%
%   U: matrix contaning each of the solutions as columns
%   errors: vector containing the error associated
%   to each step
%

%Create the matrix
U = zeros(length(A), n+1);
%Build an identity matrix
I = eye(length(A));
```

```
%The vector that will store the errors
err = zeros(n-1,1);
%A time variable to calculate the exact solution. We initialize t as
t = 2*Dt;

%The two initial values
U(:,1) = y0;
U(:,2) = y1;

%Matrices that are used several times in the loop
M1 = I + (Dt*3/2)*A;
M2 = (Dt/2)*A;

%Main loop to calculate the numerical solutions,
%and find the errors
for i = 2:n
    U(:,i+1) = M1*U(:,i) - M2*U(:,i-1);
    err(i-1) = max(abs(yex(t) - U(:,i+1)));
    t = t + Dt;
end

%return the errors
errors = err;
```

- **AM3**
  We have $\vec{\alpha} = (1, 0)$, and $\vec{\beta} = (5/12, 2/3, -1/12)$. Then, we check:

$$\sum_{0}^{p} \alpha_j = 0 + 1 = 1$$

$$-\sum_{0}^{p} j\alpha_j + \sum_{-1}^{p} \beta_j = -(0)(1) - (1)(0) + 5/12 + 2/3 - 1/12 = 1$$

  We have consistency. We now find the order of the method.
  Let $k = 2$,

$$\sum_{0}^{p} (-j)^2 \alpha_j + (2) \sum_{-1}^{p} (-j)\beta_j = 0 + 2[(1)(5/12) + 0(2/3) + (-1)(-1/12)] = 1$$

  Let $k = 3$,

$$\sum_{0}^{p} (-j)^3 \alpha_j + (3) \sum_{-1}^{p} (-j)^2\beta_j = 0 + 3[(1)(5/12) + 0(2/3) + (-1)^2(-1/12)] = 1$$

  Let $k = 4$,

$$\sum_{0}^{p} (-j)^4 \alpha_j + (4) \sum_{-1}^{p} (-j)^3\beta_j = 0 + 4[(1)(5/12) + 0(2/3) + (-1)^3(-1/12)] = 2 \neq 1$$

  Therefore, AM3 is of order 3.

Code:

```
function [U, errors] = AM3(A, y0, y1, n, Dt, yex)
%
%   A:  the matrix
%   y0, y1: the 2 initial conditions
%   n: the # of time steps
%   Dt: time step
%   yex: the exact solution
%
%   U: matrix contaning each of the solutions as columns
%   errors: vector containing the error associated
%           to each step
%

%Create the matrix
U = zeros(length(A), n+1);
%Build and identity matrix
```

```
I = eye(length(A));
%The vector that will stor
err = zeros(n-1,1);
%A time variable to calculate the exact solution
t = 2*Dt;

%The two initial values
U(:,1) = y0;
U(:,2) = y1;

%Matrices that are used several times in the loop
M1 = I - (Dt*5/12)*A;
M2 = (I + (Dt*2/3)*A);
M3 = (Dt/12)*A;


%Main loop to calculate the numerical solutions,
%and find the errors
for i = 2:n
    U(:,i+1) = M1\(M2*U(:,i) - M3*U(:,i-1));
    err(i-1) = max(abs(yex(t) - U(:,i+1)));
    t = t + Dt;
end

%return the error
errors = err;
```

- **Calculating with a one-step second-order method all the initial data needed to use the two methods above.**

  We used Heun to calculate the second initial value since Heun is second-order and explicit, which makes it convenient for practical use.

  Code:

```
function [U, errors] = Heun(A, y0, n, Dt, yex)
%
%   A:  the matrix
%   y0: the initial condition
%   n: the # of time steps
%   Dt: time step
%   yex: the exact solution
%
%   U: matrix contaning each of the solutions as columns
%   errors: vector containing the error associated
%   to each step
%

%Create the matrix
U = zeros(length(A), n+1);
%Build and identity matrix
I = eye(length(A));
%The vector that will stor
err = zeros(n,1);
%A time variable to calculate the exact solution
t = Dt;

%The initial values
U(:,1) = y0;

%Matrix that is used several times in the loop
M = I + Dt*A+ ((Dt^2)/2)*A*A;

%Main loop to calculate the numerical solutions,
%and find the errors
for i = 1:n
    U(:,i+1) = M*U(:,i);
    err(i) = max(abs(yex(t) - U(:,i+1)));
    t = t + Dt;
end

%return the error
errors = err;
```

- **Test the accuracy of the methods**

  For both methods, we initially used $n = 500$ steps. This meant a time step of $\Delta t = T_{final}/n = 10/500 = 0.02$. Then, n was multiplied by 2 and 4 and the respective errors were calculated.

  Let $e(\Delta t)$ be the error associated with a time step Δt. The results were as following:

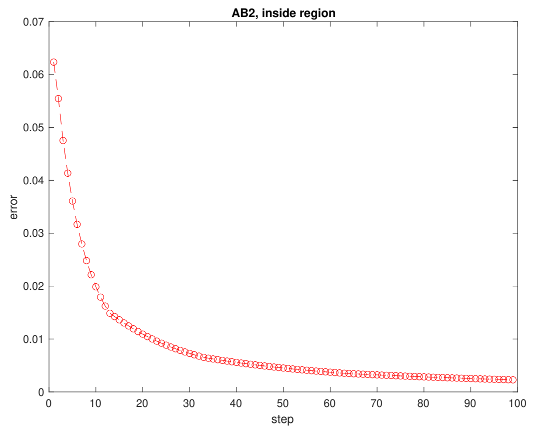
  - **AB2:**
$$\frac{e(\Delta t)}{e(\Delta t/2)} = 3.9981 \approx 2^2$$
$$\frac{e(\Delta t)}{e(\Delta t/4)} = 15.9898 \approx 4^2$$
  - **AM3:**
$$\frac{e(\Delta t)}{e(\Delta t/2)} = 7.7578 \approx 2^3$$
$$\frac{e(\Delta t)}{e(\Delta t/4)} = 61.1217 \approx 4^3$$

  The code is given in the next section.

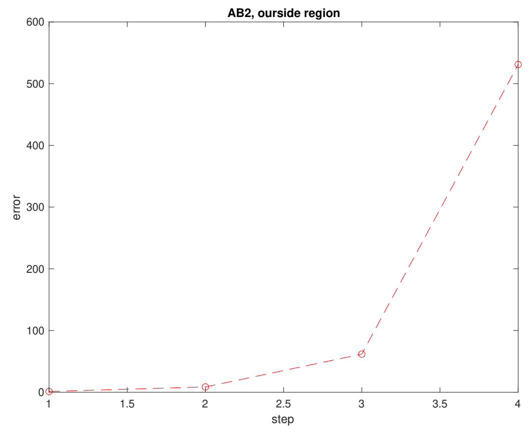
- **Stability**

  The spectral radius is $\rho = 4$, which corresponds to an eigenvalue $\lambda = -4$, with no imaginary component.

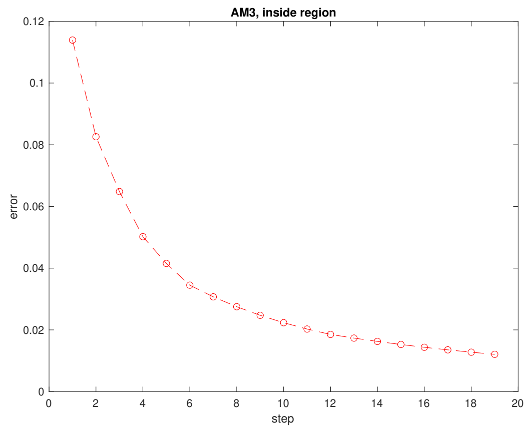
  - **AB2:**
    * Choosing a time step $\Delta t = 0.1$, we get $\lambda\Delta t = -0.4$, which is inside the region of stability. The following graph shows that the error decreases with each iteration, implying that the numerical solution approaches the exact solution.



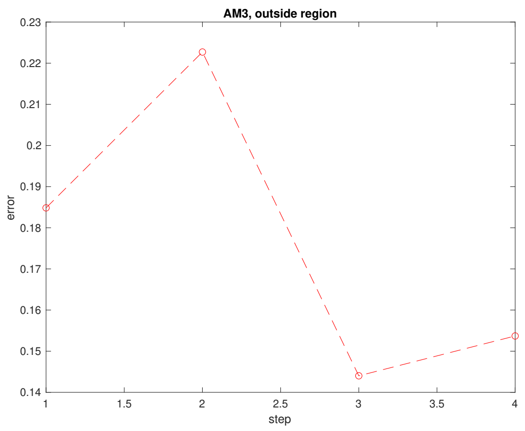
    - Choosing a time step $\Delta t = 2$, we get $\lambda \Delta t = -8$, which is outside the region of stability. The following graph shows that the error is actually boosting up as the iteration goes further. This indicates that our numerical approximation is not well-behaved under this choice of Dt.



- **AM3:**
    - Choosing a time step $\Delta t = 0.5$, we get $\lambda \Delta t = -2$, which is inside the region of stability. The following graph shows that the error decreases with each iteration. This implies that the numerical solution should converge and approach the exact solution as our iteration goes on.



    - Choosing a time step $\Delta t = 2$, we get $\lambda \Delta t = -8$, which is outside the region of stability. If Dt is a "reasonable" choice, we should expect the error behaves similarly to the previous case. Nevertheless, the following graph shows that the error's behavior does not align with our expectation.



Code:

```
%matrix setup
N = 100;
e = ones(N,1);
A = spdiags([e -2*e e],-1:1,N,N);
full(A);

%IC
y0 = e;

%interval length
T = 10;

%Initial number of steps
n = 500;

%Compute the exact solution
yex = exactSolution(A,y0);

%Vectors for storing the erros
errorAB2 = zeros(3,1);
errorAM3 = zeros(3,1);

for i = 0:2
    %adjust the # of steps
    nsteps = n*(2^i);
    %Calculate the time step size
    Dt = T/nsteps;
    %calculate the first step (Heun)
    [UH, eH] = Heun(A, y0, 1, Dt, yex);
    y1 = UH(:,2);
    %Use AB2
    [UAB2, eAB2] = AB2(A, y0, y1, nsteps, Dt, yex);
    errorAB2(i+1) = max(eAB2);

    %Use AB2
    [UAM3, eAM3] = AM3(A, y0, y1, nsteps, Dt, yex);
    errorAM3(i+1) = max(eAM3);
end


%Check we have the desired ratios
errorAB2(1)/errorAB2(2)
errorAB2(1)/errorAB2(3)

errorAM3(1)/errorAM3(2)
errorAM3(1)/errorAM3(3)

%Get the spectral radius
y = eig(A);
[rho, ind] = max(abs(y));

%Check that the eigenvalue is real and negative
%y(ind)

%then rho = -4

%AB2, Dt = 0.1 would give, Dt*rho = -0.4 is inside, that
%would require n = 100
[UAB2, eAB2] = AB2(A, y0, y1, 100, T/100, yex);

%plot
figure(1)
plot(eAB2,'--ro')
xlabel('step')
ylabel('error')
title('AB2, inside region')

%AB2, Dt = 2 would give, Dt*rho = -8 is outside, that
%would require n = 5
[UAB2, eAB2] = AB2(A, y0, y1, 5, T/5, yex);

%Plot
figure(2)
plot(eAB2,'--ro')
xlabel('step')
```

```
ylabel('error')
title('AB2, ourside region')


%AM3, Dt = 0.5 would give, Dt*rho = -2 is inside, that
%would require n = 20
[UAM3, eAM3] = AM3(A, y0, y1, 20, T/20, yex);

%Plot
figure(3)
plot(eAM3,'--ro')
xlabel('step')
ylabel('error')
title('AM3, inside region')

%AM3, Dt = 2 would give, Dt*rho = -8 is outside, that
%would require n = 5
[UAM3, eAM3] = AM3(A, y0, y1, 5, T/5, yex);

%Plot
figure(4)
plot(eAM3,'--ro')
xlabel('step')
ylabel('error')
title('AM3, outside region')
```