The Burden of Stability

Problem Statement

Consider the scalar ordinary differential equation (ODE):

$$\dot{x}(t) = f(t, x(t)) := \lambda x(t) + (1 - \lambda)\cos(t) - (1 + \lambda)sin(t),\ x(0) = 1 \tag{1}$$

The file run_ODE.m provides a framework to run various different methods for numerical integration of (1) with step-size $h > 0$ over time $t \in [0, T = 10]$. Let $N = \lfloor Th \rfloor$, where the notation $\lfloor \cdot \rfloor$ stands for the largest integer not exceeding z. Specifically, it computes a vector $(x_0 = x(0), x_1, \ldots, x_N)$, where x_n's are the proxies for $x(t_n)$ computed recursively via different methods with tn = nh. Define the average error of any numerical integration method applied to this ODE as

$$\mathcal{E} := \frac{1}{N+1} \sum_{n=0}^{N} |x(t_n) - x_n|$$

Use code or other methods to generate the plots required below and answer the questions. Please submit your code (at least for parts c and d).

a

Show that the analytical solution of the DOE is given by $x(t) = \cos(t) + \sin(t)$.

b

Plot the result of numerical integration via forward Euler method with $h = 0.15, 0.30, 0.45$ over $[0, T]$ together with the analytical solution. Comment how the average error for forward Euler $\mathcal{E}_{FE}$ varies with h. Is forward Euler method stable for all values of h you simulated?

c

To implement backward Euler method for a given step-size h ¿ 0, one needs to solve the nonlinear equation

$$x_{n+1} := x_n + hf(t_{n+1}, x_{n+1})$$

in each iteration $n \geq 0$. Implement Newton-Raphson to solve the equation $F(y) = 0$ where

$$F(y) := y - x_n - hf(t_{n+1}, y)$$

starting from the forward Euler solution, given by $y^{(0)} := x_n + hf(t_n, x_n)$. Iterate until $|F(y)| < 10^{-5}$. For the same values of h used in part (b), numerically integrate (1) using backward Euler method and plot the results together with the analytical solution on the same graph. Comment how the average error for backward Euler $\mathcal{E}_{BE}$ varies with h. Is backward Euler method stable for all values of h you simulated?

d

To implement the trapezoidal method with step-size $h > 0$, one needs to solve the nonlinear equation

$$x_{n+1} := x_n + \frac{h}{2}[f(t_n, x_n) + f(t_{n+1}, x_{n+1})]$$

in each iteration $n \geq 0$. Implement Newton-Raphson to solve the equation $F(y) = 0$ with

$$F(y) \coloneqq y - x_n - \frac{h}{2}[f(t_n, x_n) + f(t_{n+1}, y)]$$

starting from the forward Euler solution given by $y^{(0)} \coloneqq x_n + hf(t_n, x_n)$. Iterate until $|F(y)| < 10^{-5}$. For the same values used in part (b), numerically integrate (1) using the trapezoidal methond and plot the results together with the analytical solution on the same graph. Comment how the average error for this method varies with h. Is this method stable for the values of h you've simulated?

Solution

a

$$\begin{aligned}
\dot{x} &= \lambda(cos(t) + sin(t)) + (1-\lambda)\cos(t) - (1+\lambda)\sin(t) \\
&= \lambda\cos(t) + (1-\lambda)\cos(t) + \lambda\sin(t) - (1+\lambda)\sin(t) \\
&= (\lambda + 1 - \lambda)\cos(t) + (\lambda - 1 - \lambda)\sin(t) \\
&= \cos(t) - \sin(t)
\end{aligned}$$

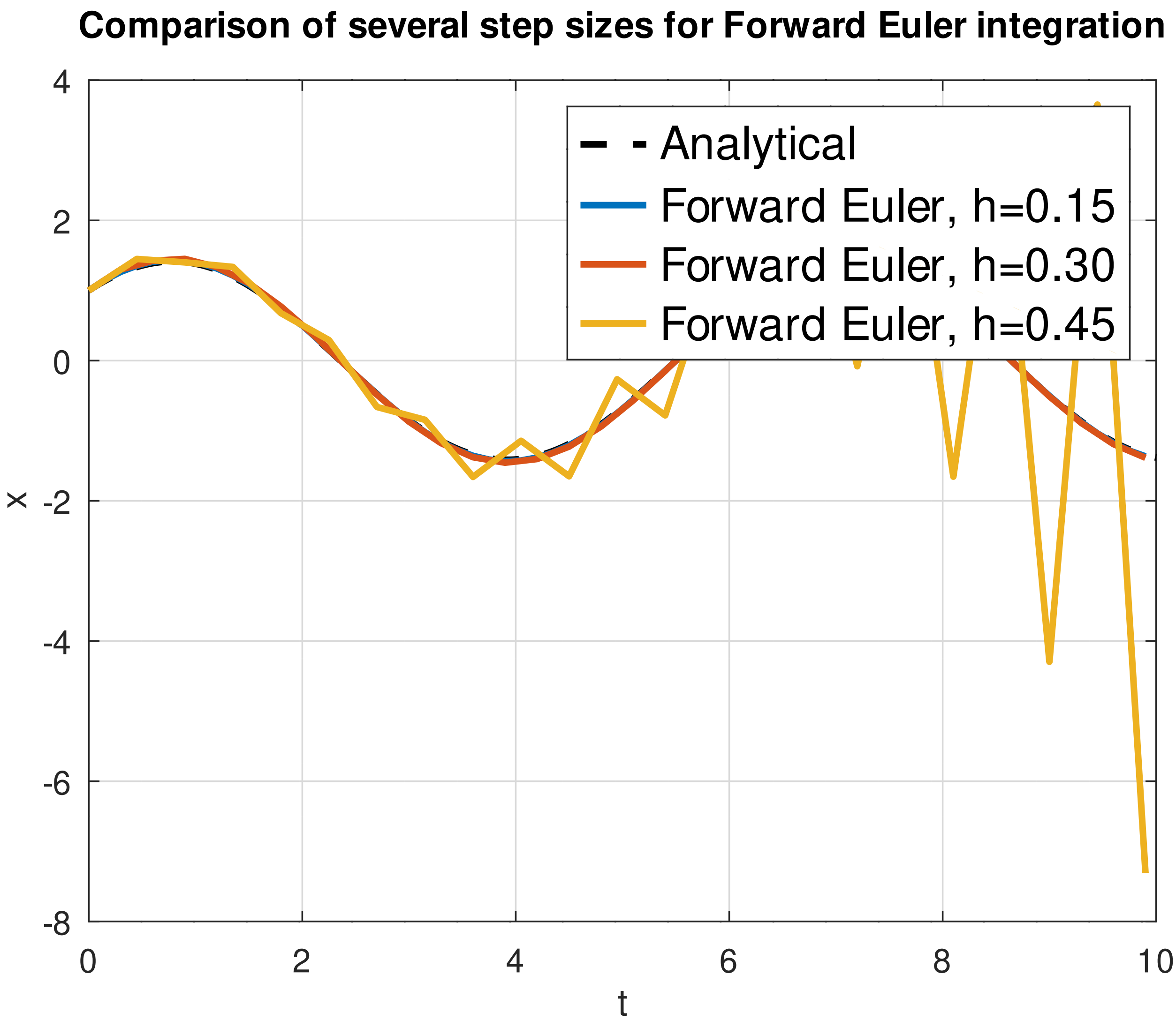
And indeed:

$$\frac{\partial}{\partial t}x(t) = \frac{\partial}{\partial t}(\cos(t) + \sin(t)) = -\sin(t) + \cos(t) = \cos(t) - \sin(t)$$

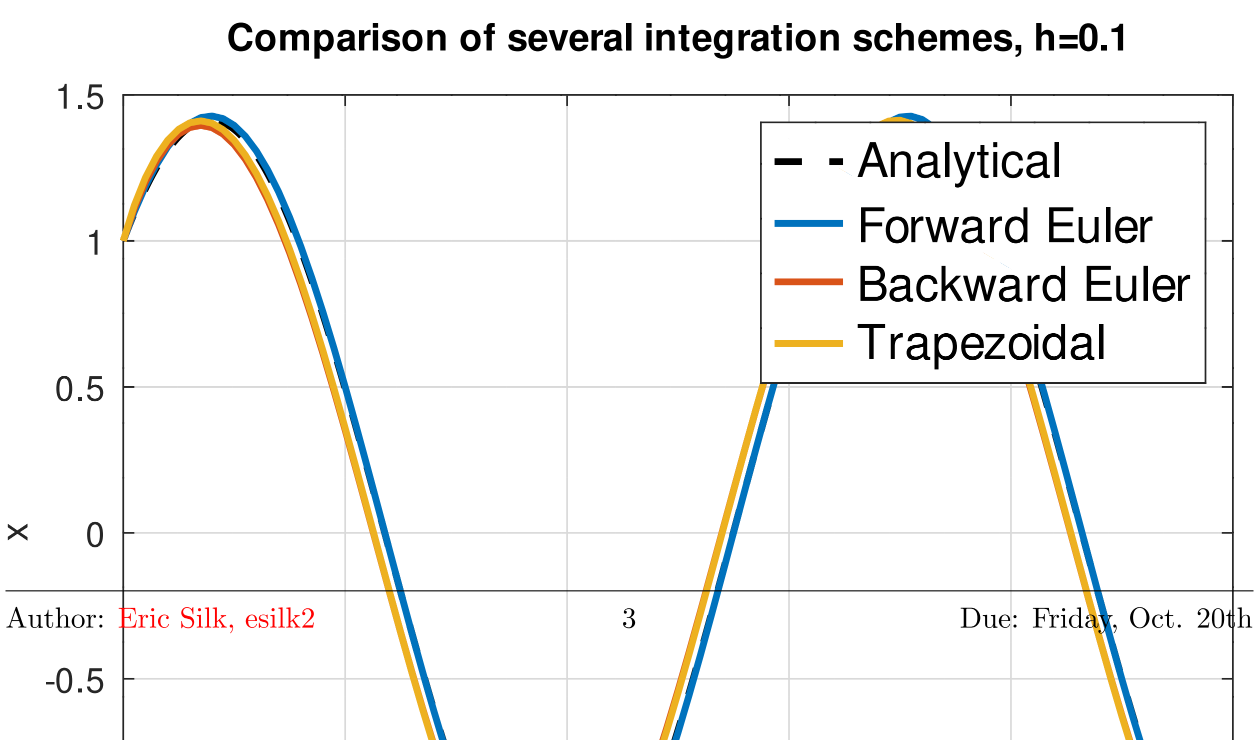
□

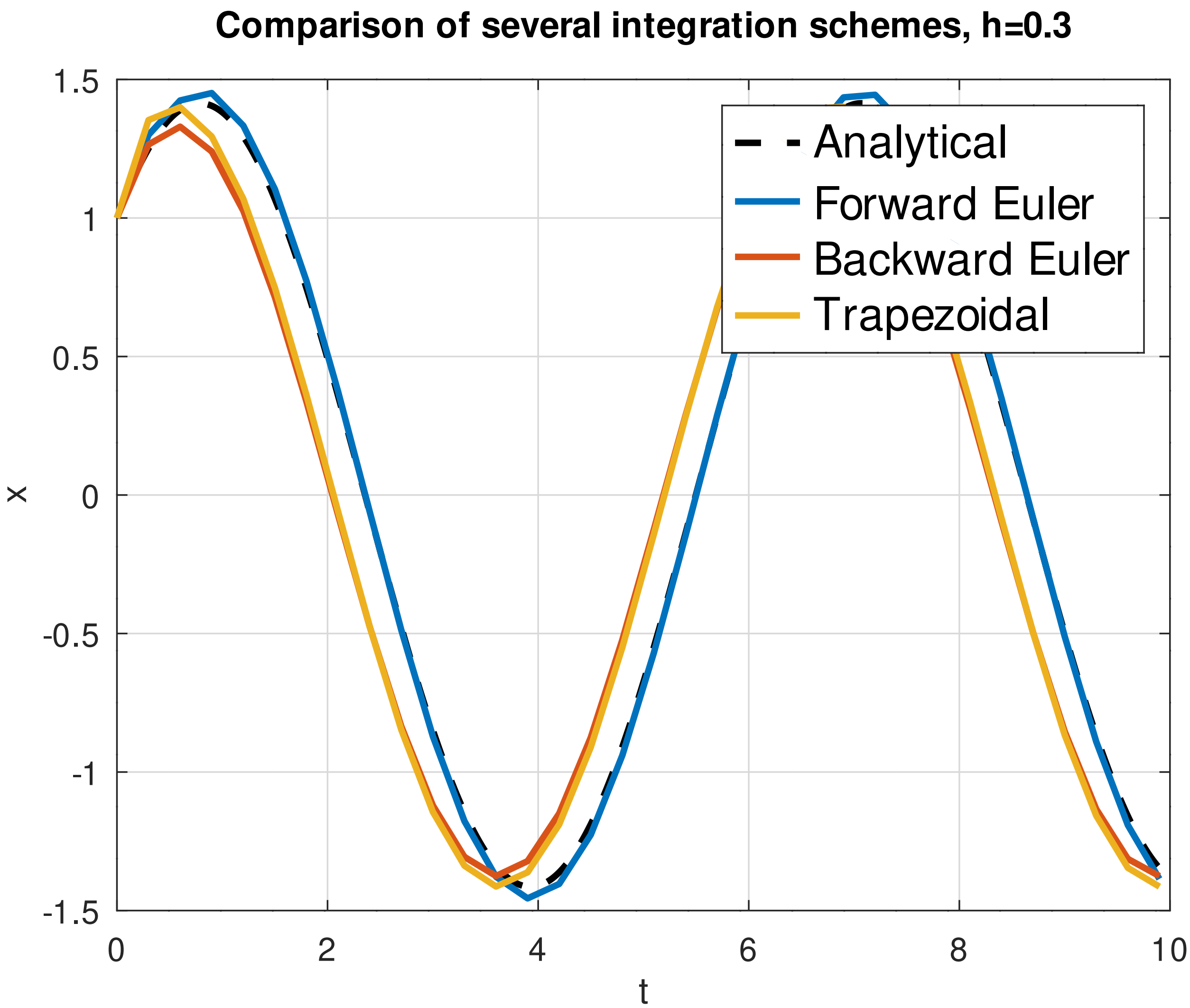
b

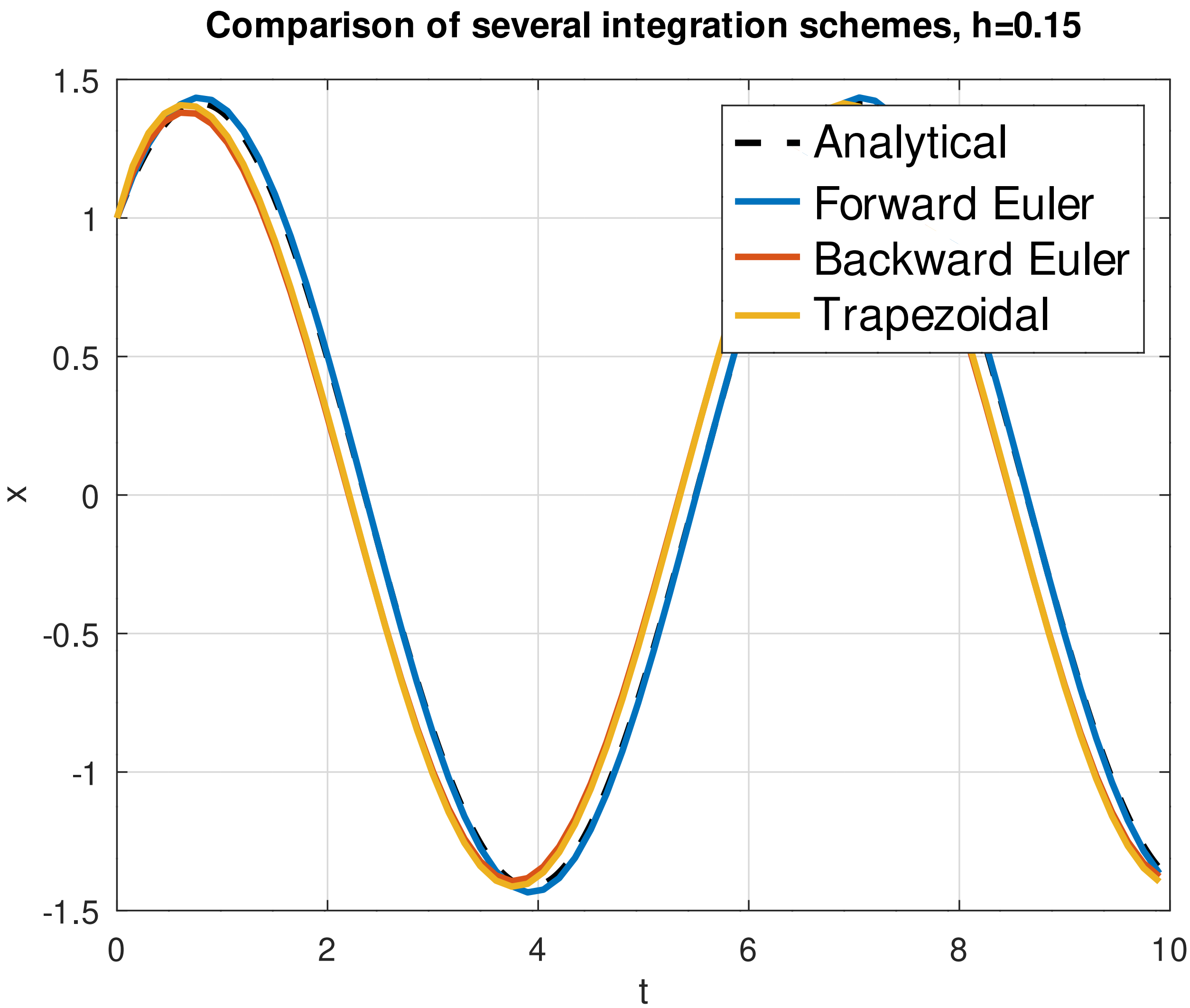
Mean error reported is 0.013344 for $h = 0.15$, 0.027145 for $h = 0.30$, and 1.2763 for $h = 0.45$. Also, it is visually apparent that the solution for $h = 0.45$ diverges, indicating numerical instability.

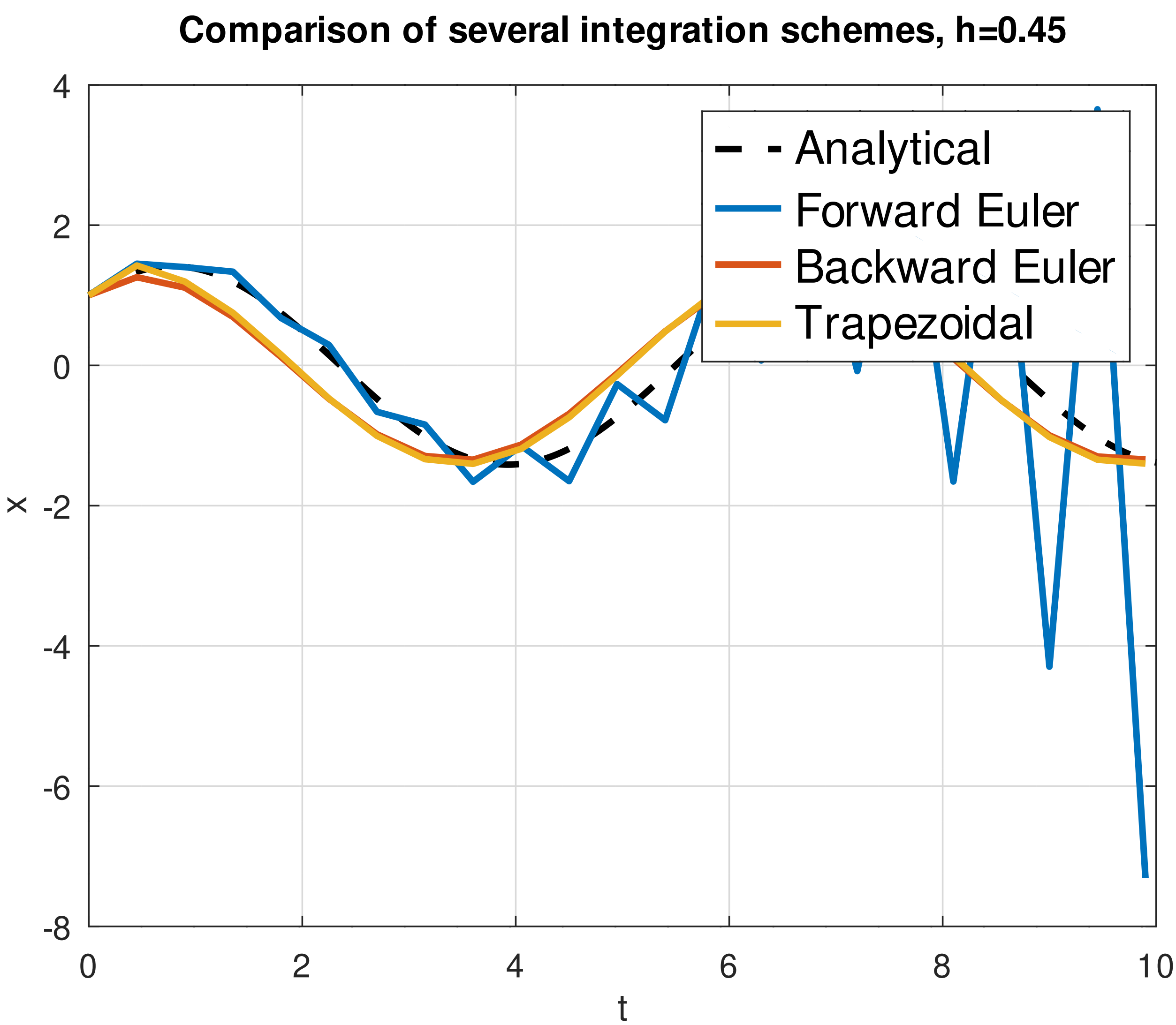

Comparison of several step sizes for Forward Euler integration
Analytical
Forward Euler, h=0.15
Forward Euler, h=0.30
Forward Euler, h=0.45
x
t

c, d


Comparison of several integration schemes, h=0.1
Analytical
Forward Euler
Backward Euler
Trapezoidal
x


Comparison of several integration schemes, h=0.3
Analytical
Forward Euler
Backward Euler
Trapezoidal
x
t
1.5
1
0.5
0
-0.5
-1
-1.5
0
2
4
6
8
10


Comparison of several integration schemes, h=0.15
1.5
1
0.5
0
-0.5
-1
-1.5
x
0
2
4
6
8
10
t
Analytical
Forward Euler
Backward Euler
Trapezoidal


Comparison of several integration schemes, h=0.45
Analytical
Forward Euler
Backward Euler
Trapezoidal
x
t
4
2
0
-2
-4
-6
-8
0
2
4
6
8
10

```
==============================================================
h=0.1
Processing analytical solution.
Processing forward Euler method.
Mean error =0.0088783
Processing backward Euler method.
Mean error =0.086862
Processing trapezoidal method.
Mean error =0.085593
==============================================================
h=0.15
Processing analytical solution.
Processing forward Euler method.
Mean error =0.013344
Processing backward Euler method.
Mean error =0.13035
Processing trapezoidal method.
Mean error =0.12856
==============================================================
h=0.3
Processing analytical solution.
Processing forward Euler method.
Mean error =0.027145
Processing backward Euler method.
Mean error =0.25538
Processing trapezoidal method.
Mean error =0.25064
==============================================================
h=0.45
Processing analytical solution.
Processing forward Euler method.
Mean error =1.2763
Processing backward Euler method.
Mean error =0.37076
Processing trapezoidal method.
Mean error =0.36708
```

Backward Euler and Trapezoidal both appear to be stable for all simulated values. Forward Euler is shown for completeness, and it blows up as expected.

Problem 2: Conjugacy is Independence

Problem Statement

If $\{d^0, \ldots, d^{n-1}\}$ are pairwise $Q-$conjugate vectors in $\mathbb{R}^n$
$\{0\}$ for $Q \succ 0$ the prove they are linearly independent.

Solution

Recall a sequence of vectors are linearly independent if:

$$\alpha_0 d_0 + \alpha_1 d_0 + \ldots + \alpha_k d_k = 0$$

where $\{\alpha_1, \ldots, \alpha_k\} \setminus 0 \neq \varnothing$ (i.e. there's at least one non-zero α).

If $\sum_{i=0}^{n-1} \alpha_i d_i$ then for $i_0 \in 0, 1, \ldots, n-1$

$$0 = d_{i_0}^T Q \sum_{i=0}^{n-1} \alpha_i d_i = \alpha_{i_0} d_{i_0}^T Q d_i$$

So $\alpha_i = 0 \forall i = 0, \ldots, n-1$. $\square$

Credit to Dr. Burke's website[1].

Problem 3: The Price of Laziness

Problem Statement

For the ODE given by $\dot{x} = f(x,t)$, investigate the absolute stability of the following numerical integration schemes:

1. Trapezoidal rule with one step fixed-point iteration from forward Euler, given by:

$$x_{n+1} = x_n + \frac{h}{2}[f(x_n, t_n) + f(x_n + hf(x_n, t_n), t_{n+1})]$$

2. Backward Euler with one step fixed-point iteration from forward Euler, given by:

$$x_{n+1} = x_n + \frac{h}{2}f(x_n + hf(x_n, tn), t_{n+1})$$

Solution

a

From the notes, let $f(x,t) \coloneqq \lambda x$. Because this system is autonomous, we can simplify and say:

$$f(x_n, t_n) = f(x_n) = \lambda x_n$$

$$f(x_n + hf(x_n, t_n), t_{n+1}) = f(x_n + hf(x_n)) = f(x_n + h(\lambda x_n)) = \lambda(x_n + h(\lambda x_n))$$

$$x_{n+1} = x_n + \frac{h}{2}[\lambda x_n + \lambda(x_n + h\lambda x_n)]$$

$$x_{n+1} = x_n(1 + \frac{h}{2}[\lambda + \lambda(1 + h\lambda)])$$

$$x_{n+1} = \frac{x_n}{2}(h^2\lambda^2 + 2h\lambda + 2)$$

$$\implies x_n = (\frac{1}{2}h^2\lambda^2 + h\lambda + 1)^n$$

Which, taking the limit as $n \to \infty$, we see that it will only go to zero for

$$-1 < \frac{1}{2}h^2\lambda^2 + h\lambda + 1 < 1$$

$$\implies -2 < \frac{1}{2}(h\lambda)^2 + h\lambda < 0$$

Considering the left term:

$$0 < \frac{1}{2}(h\lambda)^2 + h\lambda + 2$$

which holds $\forall h\lambda$. The right portion, then:

$$\frac{1}{2}(h\lambda)^2 + h\lambda < 0$$

This is true for all $-2 < h\lambda < 0$. Recall $\lambda < 0$ and $h > 0$; therefore, $h\lambda < 0$ and the above equation is only restricted by the minimum value. Again, because *lambda* is strictly negative, I'm going to relate this using the absolute value of *lambda* (otherwise the negatives make the expression weird...):

$$h < 2/|\lambda|$$

Conceptually, the faster the prototypical exponential function decays (more negative λ), the smaller our step size needs to be. Because this is not stable $\forall h > 0$, we cannot say this is absolutely stable.

b

Proceeding as before:

$$x_{n+1} = x_n + h\lambda(x_n + h\lambda x_n) = x_n(1 + h\lambda + (h\lambda)^2)$$

So for stability:

$$|1 + h\lambda + (h\lambda)^2| < 1 \implies -1 < 1 + h\lambda + (h\lambda)^2 < 1$$

The left relationship is true $\forall h\lambda \in \mathbb{R} \implies \forall h \in \mathbb{R}$. The right, however, is only true for:

$$-1 < h\lambda < 0$$

Which does not hold $\forall h \in \mathbb{R}$. Therefore, the method is not absolutely stable.

Code

"run_ODE.m"

```
clc
clear
close all

% Code for HW 5, ECE 530, Fall 2023.

% Consider the ODE \dot{x} = f(t, x), starting from x0.
% Define the function f.
L = -5;
f = @(t, x) (L*x + (1-L) * cos(t) - (1 + L) * sin(t));

% Define the derivative of 'f' with respect to 'x'.
deriv_f = @(t,x) (L);
f_prime = deriv_f;

% Initial point
x0 = 1;

% Time horizon
T = 10;

hs = [0.1, 0.15, 0.30, 0.45];

for i=1:numel(hs)
  % Step-size
  h = hs(i);

  % Number of iterations
  N = floor(T/h);
  times = (0:h:N*h)';

  % Create a vector of results in x. Notice that our implementation is
  % such that x(1) = x_0, x(2) = x_1, x(3) = x_2, etc.
  x = zeros(1 + N, 1);
  x(1) = x0;

  figure()
  %--------------------------
  % Analytical Solution of ODE
  %--------------------------
  disp("================================================================")
  disp(strcat("h=", num2str(h)))
  disp('Processing analytical solution.')

  % Define the analytical solution.
  solution_ODE = @(t) (cos(t) + sin(t));

  % Plot the analytical solution.
  times_cont = (0:0.01:T)';
  plot(times_cont, solution_ODE(times_cont), 'k--', 'Linewidth', 2)
  hold on


  %-------------
  % Forward Euler method.
  %-------------

  disp('Processing forward Euler method.')

  % Implement the method.
  for n = 1:N
      x(n+1) = x(n) + h * f((n-1) * h, x(n));
  end

  % Plot the outcome.
  plot(times, x, 'Linewidth', 2)
  hold on

  % Compute the average error.
  display(strcat(...
      'Mean error = ', ...
      num2str(mean(abs(x - solution_ODE(times))))...
  ))


  %-------------
  % Backward Euler method.
  %-------------

  disp('Processing backward Euler method.')

  y = x(1);
  for n = 1:N

      % Define F(y) such that F(y)=0 is equivalent to solving
      % the implicit equation that arises in each iteration of
```

```
      % backward Euler method. Implement a Newton-Raphson method
      % to compute x(n+1). Start the NR iteratiion from the explicit
      % Euler solution. Iterate till | F(y) | > 10^{-5}

      % Insert your code here to approximately solve F(y) = 0.
      F = @(ynp1) ynp1-y-h*f(h*(n+1), ynp1);
      F_prime = @(ynp1) 1-h*f_prime(h*(n+1), ynp1);
      y = newton_raphson(y, F, F_prime, 1e-5, 1000);
      x(n+1) = y;
  end


  % Plot the outcome.
  plot(times, x, 'Linewidth', 2)
  hold on

  % Compute the average error.
  display(strcat(...
      'Mean error = ', ...
      num2str(mean(abs(x - solution_ODE(times))))...
  ))


  %------------
  % Trapezoidal method.
  %------------

  disp('Processing trapezoidal method.')
  y = x(1);
  for n = 1:N

      % Define F(y) such that F(y)=0 is equivalent to solving
      % the implicit equation that arises in each iteration of
      % the trapezoidal method. Implement a Newton-Raphson method
      % to compute x(n+1). Start the NR iteratiion from the explicit
      % Euler solution. Iterate till | F(y) | > 10^{-5}

      % Insert your code here to approximately solve F(y) = 0.
      F = @(ynp1) ynp1-y-(h/2)*(f(h*n, y)+f(h*(n+1), ynp1));
      F_prime = @(ynp1) 1-0.5*h*f_prime(h*(n+1), ynp1);
      y = newton_raphson(y, F, F_prime, 1e-5, 1000);
      x(n+1) = y;

  end

  % Plot the outcome.
  plot(times, x, 'Linewidth', 2)
  xlabel("t")
  ylabel("x")
  title(strcat("Comparison of several integration schemes, h=", num2str(h)))
  hold on

  % Compute the average error.
  display(strcat(...
      'Mean error = ', ...
      num2str(mean(abs(x - solution_ODE(times))))...
  ))

  %------------
  % Add legends, grid to the plot.
  %------------

  legend({'Analytical', 'Forward Euler', 'Backward Euler', 'Trapezoidal'}, 'FontSize',14)
  grid on
  hold off
  print(strcat("integration_", num2str(h), '.eps'), '-deps', '-color')
endfor
```

"forward_euler_stability.m"

```
clc
clear
close all

% Code for HW 5, ECE 530, Fall 2023.

% Consider the ODE \dot{x} = f(t, x), starting from x0.
% Define the function f.
L = -5;
f = @(t, x) (L*x + (1-L) * cos(t) - (1 + L) * sin(t));

% Define the derivative of 'f' with respect to 'x'.
deriv_f = @(t,x) (L);
f_prime = deriv_f;

% Initial point
x0 = 1;

% Time horizon
T = 10;

% Step-size
hs = [0.15, 0.3, 0.45];
h=hs(1);

% Number of iterations
N = floor(T/h);
times = (0:h:N*h)';

% Create a vector of results in x. Notice that our implementation is
% such that x(1) = x_0, x(2) = x_1, x(3) = x_2, etc.
x = zeros(1 + N, 1);
x(1) = x0;


%------------------------
% Analytical Solution of ODE
%------------------------

disp('Processing analytical solution.')

% Define the analytical solution.
solution_ODE = @(t) (cos(t) + sin(t));

% Plot the analytical solution.
times_cont = (0:0.01:T)';
figure(1);
plot(times_cont, solution_ODE(times_cont), 'k--', 'Linewidth', 2)
hold on


%-------------
% Forward Euler method.
%-------------

disp('Processing forward Euler method.')
for i = 1:3
  h = hs(i);
  N = floor(T/h);
  times = (0:h:N*h)';

  % Create a vector of results in x. Notice that our implementation is
  % such that x(1) = x_0, x(2) = x_1, x(3) = x_2, etc.
  x = zeros(1 + N, 1);
  x(1) = x0;
  % Implement the method.
  for n = 1:N
      x(n+1) = x(n) + h * f((n-1) * h, x(n));
  end
  % Plot the outcome.
  plot(times, x, 'Linewidth', 2)
  xlabel("t")
  ylabel("x")
  title("Comparison of several step sizes for Forward Euler integration")
  hold on

  % Compute the average error.
  display(strcat(...
      'Mean error = ', ...
      num2str(mean(abs(x - solution_ODE(times))))...
  ))
endfor


legend({'Analytical', 'Forward Euler, h=0.15', 'Forward Euler, h=0.30', 'Forward Euler, h=0.45'}, '
    FontSize',14)
grid on
hold off
print('forward_euler.eps', '-deps', '-color')
```

"newton_raphson.m"

```
function xn = newton_raphson(x0, func, deriv, tol, max_iter)
  x = x0;
  for n = 1:max_iter
    fx = func(x);
    if (abs(fx) < tol)
      xn = x;
      return;
    endif
    fprime = deriv(x);
    dx = fx/fprime;
    if isnan(dx)
      display("NaN encountered, halting!")
      xn = x;
      return;
    endif
    tmp = x-dx;
    if (isnan(tmp))
      display("NaN would be produced, halting!")
      xn = x;
      return;
    endif
    x = x - dx;
  endfor
  xn = x0;
  disp("Failed to converge!")
end
```

Bibliography

[1] James V Burke. *Conjugate Direction Methods - University of Washington*. Feb. 2007. URL: `https://sites.math.washington.edu/~burke/crs/408f/notes/nlp/cg.pdf`.