The Burden of Stability

Problem Statement

Consider the scalar ordinary differential equation (ODE):

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

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

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

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

lot 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

$$x_{n+1} \coloneqq x_n + hf(t_{n+1}, x_{n+1})$$

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

$$F(y) \coloneqq y - x_n - hf(t_{n+1}, y)$$

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 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

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

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

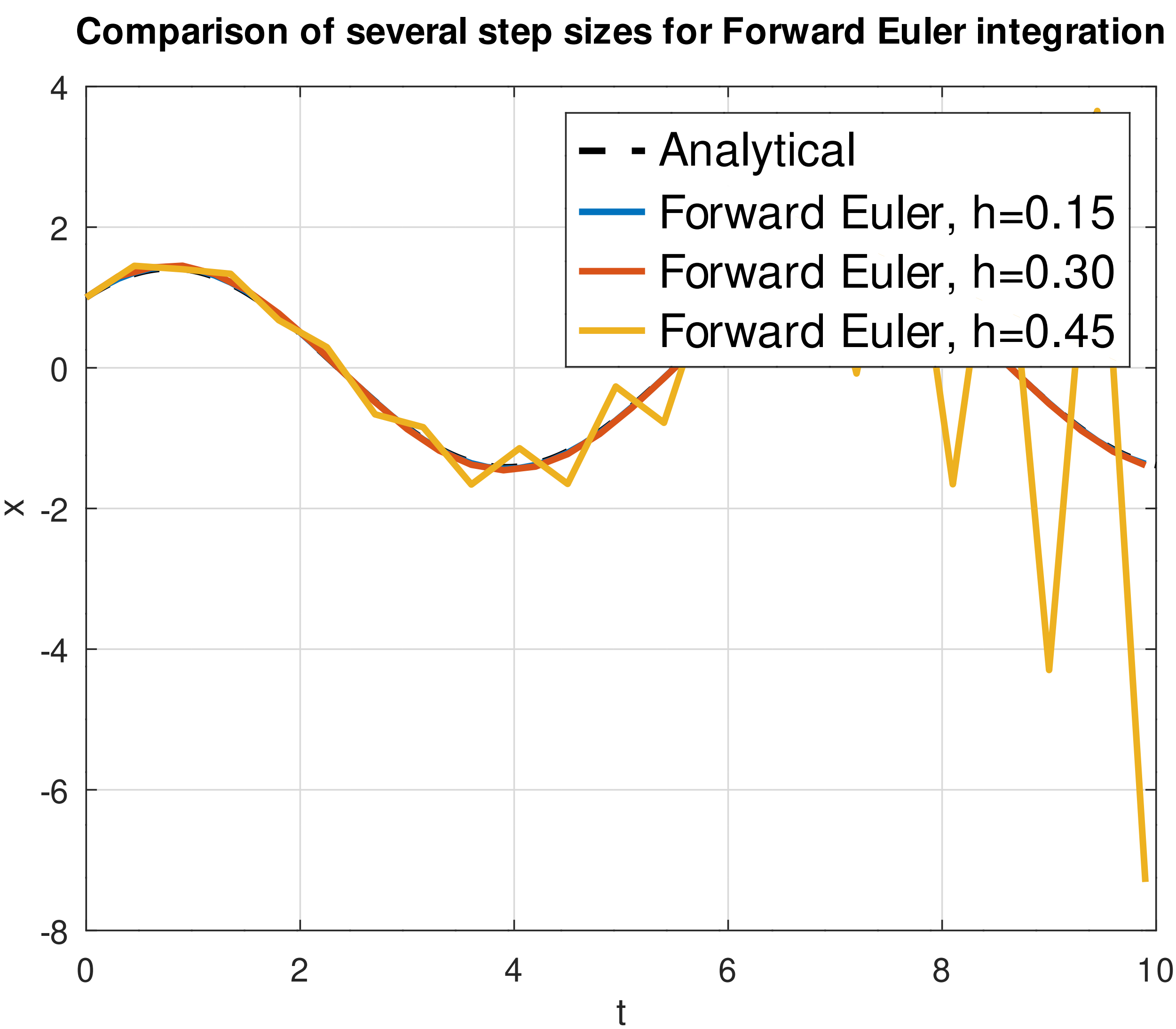
$$F(y) \coloneqq y - x_n - \frac{h}{2}[f(t_n, x_n) + f(t_{n+1}, y)]$$

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

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.

c, d

```
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
```

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 not all scalars $\alpha_i = 0$

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

Code

```
function HTilde = modifyHessian(H)
    n = size(H, 1);

    % Problem 4.1: Write your code here to compute D
    % using your answer in part (c).
    h_ii = diag(H);
    D_ii = zeros(size(h_ii));
    off_diag = H-diag(h_ii);

    for i = 1:n
        r = sum(abs(off_diag(i,:)));
        min_eigenvalue_estimate = h_ii(i) - r;
        if min_eigenvalue_estimate < (1/2)
            D_ii(i) = 1/2 + r - h_ii(i);
        end
    end

    D = diag(D_ii);
    HTilde = H + D;
end
```

```
% Code to run a generic Newton method.
clear all
close all
clc; history -c

MAX_ITER = 100;

% Function and its gradient and hessian.

f = @(x, y) (cos(x.^2 - 3*y)  + sin(x.^2 + y.^2));

gradf = @(x, y) ([ 2*x*cos(x^2 + y^2) - 2*x*sin(x^2 - 3*y); ...
                   3*sin(x^2 - 3*y) + 2*y*cos(x^2 + y^2) ...
                ]);

hessf = @(x,y) [ 2*cos(x^2 + y^2) - 2*sin(x^2 - 3*y) - 4*x^2*cos(x^2 - 3*y) - 4*x^2*sin(x^2 + y^2), ...
                   6*x*cos(x^2 - 3*y) - 4*x*y*sin(x^2 + y^2); ...
                 6*x*cos(x^2 - 3*y) - 4*x*y*sin(x^2 + y^2), ...
                   2*cos(x^2 + y^2) - 9*cos(x^2 - 3*y) - 4*y^2*sin(x^2 + y^2)];

% Plot the function and its contour plot to
% appreciate how this function looks like
[Xgrid, Ygrid] = meshgrid(-1.5:0.1:1.5, -1.5:0.1:1.5);
Z = f(Xgrid, Ygrid);

subplot(2,1,1), contour(Xgrid, Ygrid, Z, 40,'Linewidth',2),
grid on, xlabel('$x_1$', 'Interpreter','Latex', 'Fontsize', 20),
ylabel('$x_2$', 'Interpreter','Latex', 'Fontsize', 20),

subplot(2,1,2), surf(Xgrid, Ygrid, Z), grid on,
xlabel('$x_1$', 'Interpreter','Latex', 'Fontsize', 20),
ylabel('$x_2$', 'Interpreter','Latex', 'Fontsize', 20),
zlabel('$f(x_1, x_2)$', 'Interpreter','Latex', 'Fontsize', 20),



% Initialize at (1.2, 0.5).
xk = [1.2; 0.5];

% Tolerance for the gradient value.
tolerance = 1e-6;

% Variables required for the iteration.
shouldIterate = true;
iterationK = 1;
errorGF = 0
while (shouldIterate)

    % Display the current iteration.
    display(strcat('Iteration # ', num2str(iterationK)))
    display(strcat( ...
        'Current values of (x,y) = [', ...
        num2str(xk'), ...
        ']'...
    ))
    % Compute the norm of the gradient.
    errorGF = norm(gradf(xk(1), xk(2)), 2);

    display(strcat( ...
        'Current norm of gradient = ', ...
        num2str(errorGF) ...
    ))
    % Added by ESilk to display if the Hessian is PD!
    if(iterationK == 1)
        Hk = hessf(xk(1), xk(2));
        output_str = {"False", "True"};
```

```
        H_is_PD = "False";
        if(all(eig(Hk)>0))
          H_is_PD = "True";
        end
        printf('H > 0: %s', H_is_PD)
        disp('')
    end
    %pause
        % Uncomment the previous line if you want to
        % look at each new iterate.

    if (errorGF > tolerance)

        % Compute the current Hessian
        Hk = hessf(xk(1), xk(2));
        Hk = modifyHessian(Hk);
        if(iterationK == 1)
            display('Modified H0:')
            display(num2str(Hk))
        end
        display('')
        assert (all(eig(Hk) > 0))
            % Problem 4.1: Uncomment the two lines above and include your function
            % to modify the Hessian as per your answer to part (b).


        % Compute the Newton direction.
        sk = - inv(Hk) * gradf(xk(1), xk(2));

        alphak = 1;

        % alphak = lineSearch(f, gradf, xk, sk, 0.0001);
        % if alphak == -1
        %     break
        % end
            % Probelm 4.2: Uncomment the above lines for implementing your own
            % line search function. It also enforces that the line
            % search in fact converges.

        % Compute the next iterate.
        xk = xk + alphak * sk;

        iterationK = iterationK + 1;
    else
        % Error is within tolerance
        shouldIterate = false;
    end

    if iterationK == MAX_ITER
        display('Did not converge within 100 iterations')
        break
    end
end

display(strcat( ...
    'Function value at last iterate = ', ...
    num2str(f(xk(1), xk(2))) ...
))
Hk = hessf(xk(1), xk(2));

H_is_PD = "False";
if(all(eig(Hk)>=0))
  H_is_PD = "True";
end

display(strcat('Iterate is a local minimum:', ...
  H_is_PD
 ))
```

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.