

The Burden of Stability

Problem Statement

Consider the scalar ordinary differential equation (ODE):

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

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 $(x_0 = x(0), x_1, \dots, x_N)$, 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

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

starting from the forward Euler solution given by $y^{(0)} := x_n + hf(t_n, x_n)$. Iterate until $|F(y)| < 10^{-5}$. For the same values used in part (b), numerically integrate (1) using the trapezoidal method 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

c

Problem 2: Conjugacy is Independence

Problem Statement

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

Solution

Recall a sequence of vectors are linearly independent if:

$$\alpha_0 d_0 + \alpha_1 d_1 + \dots + \alpha_k d_k = 0$$

where not all scalars $\alpha_i = 0$

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

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

So $\alpha_i = 0 \forall i = 0, \dots, n-1$. □

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:

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

Solution

Code

```

1 function HTilde = modifyHessian(H)
2     n = size(H, 1);
3
4     % Problem 4.1: Write your code here to compute D
5     % using your answer in part (c).
6     h_ii = diag(H);
7     D_ii = zeros(size(h_ii));
8     off_diag = H-diag(h_ii);
9
10
11     for i = 1:n
12         r = sum(abs(off_diag(i,:)));
13         min_eigenvalue_estimate = h_ii(i) - r;
14         if min_eigenvalue_estimate < (1/2)
15             D_ii(i) = 1/2 + r - h_ii(i);
16         end
17     end
18
19     D = diag(D_ii);
20     HTilde = H + D;
21 end

1 % Code to run a generic Newton method.
2 clear all
3 close all
4 clc; history -c
5
6 MAX_ITER = 100;
7
8 % Function and its gradient and hessian.
9
10 f = @(x, y) (cos(x.^2 - 3*y) + sin(x.^2 + y.^2));
11
12 gradf = @(x, y) ([ 2*x*cos(x^2 + y^2) - 2*x*sin(x^2 - 3*y); ...
13                  3*sin(x^2 - 3*y) + 2*y*cos(x^2 + y^2) ...
14                  ]);
15
16 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), ...
17                6*x*cos(x^2 - 3*y) - 4*x*y*sin(x^2 + y^2); ...
18                6*x*cos(x^2 - 3*y) - 4*x*y*sin(x^2 + y^2), ...
19                2*cos(x^2 + y^2) - 9*cos(x^2 - 3*y) - 4*y^2*sin(x^2 + y^2)];
20
21 % Plot the function and its contour plot to
22 % appreciate how this function looks like
23 [Xgrid, Ygrid] = meshgrid(-1.5:0.1:1.5, -1.5:0.1:1.5);
24 Z = f(Xgrid, Ygrid);
25
26 subplot(2,1,1), contour(Xgrid, Ygrid, Z, 40,'Linewidth',2),
27 grid on, xlabel('$x_1$', 'Interpreter','Latex', 'FontSize', 20),
28 ylabel('$x_2$', 'Interpreter','Latex', 'FontSize', 20),
29
30 subplot(2,1,2), surf(Xgrid, Ygrid, Z), grid on,
31 xlabel('$x_1$', 'Interpreter','Latex', 'FontSize', 20),
32 ylabel('$x_2$', 'Interpreter','Latex', 'FontSize', 20),
33 zlabel('$f(x_1, x_2)$', 'Interpreter','Latex', 'FontSize', 20),
34
35
36
37 % Initialize at (1.2, 0.5).
38 xk = [1.2; 0.5];
39
40 % Tolerance for the gradient value.
41 tolerance = 1e-6;
42
43 % Variables required for the iteration.
44 shouldIterate = true;
45 iterationK = 1;
46 errorGF = 0
47 while (shouldIterate)
48
49     % Display the current iteration.
50     display(strcat('Iteration # ', num2str(iterationK)))
51     display(strcat( ...
52         'Current values of (x,y) = [', ...
53         num2str(xk), ', ...
54         ']' ...
55     ))
56     % Compute the norm of the gradient.
57     errorGF = norm(gradf(xk(1), xk(2)), 2);
58
59     display(strcat( ...
60         'Current norm of gradient = ', ...
61         num2str(errorGF) ...
62     ))
63     % Added by ESilk to display if the Hessian is PD!
64     if(iterationK == 1)
65         Hk = hessf(xk(1), xk(2));
66         output_str = {"False", "True"};

```

```

67     H_is_PD = "False";
68     if (all(eig(Hk)>0))
69         H_is_PD = "True";
70     end
71     printf('H > 0: %s', H_is_PD)
72     disp('')
73 end
74 %pause
75 % Uncomment the previous line if you want to
76 % look at each new iterate.
77
78 if (errorGF > tolerance)
79
80     % Compute the current Hessian
81     Hk = hessf(xk(1), xk(2));
82     Hk = modifyHessian(Hk);
83     if (iterationK == 1)
84         display('Modified H0:')
85         display(num2str(Hk))
86     end
87     display('')
88     assert(all(eig(Hk) > 0))
89     % Problem 4.1: Uncomment the two lines above and include your function
90     % to modify the Hessian as per your answer to part (b).
91
92
93     % Compute the Newton direction.
94     sk = - inv(Hk) * gradf(xk(1), xk(2));
95
96     alphak = 1;
97
98     % alphak = lineSearch(f, gradf, xk, sk, 0.0001);
99     % if alphak == -1
100     %     break
101     % end
102     % Problem 4.2: Uncomment the above lines for implementing your own
103     % line search function. It also enforces that the line
104     % search in fact converges.
105
106     % Compute the next iterate.
107     xk = xk + alphak * sk;
108
109     iterationK = iterationK + 1;
110 else
111     % Error is within tolerance
112     shouldIterate = false;
113 end
114
115 if iterationK == MAX_ITER
116     display('Did not converge within 100 iterations')
117     break
118 end
119 end
120
121 display(strcat('...
122     'Function value at last iterate = ', ...
123     num2str(f(xk(1), xk(2))) ...
124 ))
125 Hk = hessf(xk(1), xk(2));
126
127 H_is_PD = "False";
128 if (all(eig(Hk)>=0))
129     H_is_PD = "True";
130 end
131
132 display(strcat('Iterate is a local minimum:', ...
133     H_is_PD
134 ))

```

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>.