Homework Problems (with explicit formulas + MATLAB implementation)

Problem 1: Unconstrained minimization in $\mathbb{R}^n$ using BFGS (MATLAB-ready)

1. A concrete test function (Rosenbrock in $\mathbb{R}^2$)

We will minimize the classical Rosenbrock function

$$f(x,y) = 100\,(y - x^2)^2 + (1-x)^2, \qquad \begin{bmatrix} x \\ y \end{bmatrix} \in \mathbb{R}^2.$$

It has a global minimizer at $(x^\star, y^\star) = (1,1)$ with $f(1,1) = 0$.

Gradient (explicit). Let $z = y - x^2$. Then

$$\frac{\partial f}{\partial x} = -400x(y-x^2) - 2(1-x) = -400xz - 2(1-x), \qquad \frac{\partial f}{\partial y} = 200(y-x^2) = 200z.$$

Thus, with $u = [x; y]$,

$$g(u) = \nabla f(u) = \begin{bmatrix} -400x(y-x^2) - 2(1-x) \\ 200(y-x^2) \end{bmatrix}.$$

2. BFGS iteration (explicit formulas)

Given $x_k \in \mathbb{R}^2$ and an inverse-Hessian approximation $H_k \in \mathbb{R}^{2\times 2}$:

Step 1: Compute gradient $g_k = g(x_k)$.

Step 2: Compute search direction

$$p_k = -H_k g_k.$$

Step 3: Choose a step size $\alpha_k > 0$ (we will use a simple backtracking Armijo line search).

Step 4: Update

$$x_{k+1} = x_k + \alpha_k p_k.$$

Step 5: Define

$$s_k = x_{k+1} - x_k, \qquad y_k = g_{k+1} - g_k.$$

Step 6: Update H_k by the BFGS formula:

$$\rho_k = \frac{1}{y_k^\top s_k}, \qquad H_{k+1} = (I - \rho_k s_k y_k^\top)\, H_k\, (I - \rho_k y_k s_k^\top) + \rho_k s_k s_k^\top.$$

3. MATLAB code (complete runnable script)

```
%% BFGS on Rosenbrock function (2D)
clear; clc;

% Rosenbrock function and gradient (explicit)
f = @(x) 100*(x(2)-x(1)^2)^2 + (1-x(1))^2;
g = @(x) [ -400*x(1)*(x(2)-x(1)^2) - 2*(1-x(1));
           200*(x(2)-x(1)^2) ];
```

```
8
9  % Parameters
10 x = [-1.2; 1.0]; % initial guess
11 H = eye(2); % initial inverse Hessian approx
12 maxit = 5000;
13 tol_g = 1e-8;
14
15 % Armijo backtracking parameters
16 c1 = 1e-4;
17 beta = 0.5;
18
19 % Storage for history
20 hist = zeros(maxit,4); % [iter f normg alpha]
21
22 for k = 1:maxit
23     fk = f(x);
24     gk = g(x);
25     ng = norm(gk);
26
27     if ng < tol_g
28         hist = hist(1:k-1,:);
29         break;
30     end
31
32     % BFGS direction
33     pk = -H*gk;
34
35     % Backtracking Armijo line search:
36     % f(x+alpha p) <= f(x) + c1*alpha*g^T p
37     alpha = 1.0;
38     gTp = gk'*pk;
39     while f(x + alpha*pk) > fk + c1*alpha*gTp
40         alpha = beta*alpha;
41         if alpha < 1e-14
42             warning('Line␣search␣failed:␣alpha␣too␣small.');
43             break;
44         end
45     end
46
47     % Step
48     xnew = x + alpha*pk;
49     gnew = g(xnew);
50
51     sk = xnew - x;
52     yk = gnew - gk;
53
54     % Safeguard: ensure y^T s > 0
55     ys = yk'*sk;
56     if ys <= 1e-14
57         % If curvature condition fails, reset H (simple robust fix)
58         H = eye(2);
59     else
60         rho = 1/ys;
61         I = eye(2);
62         H = (I - rho*sk*yk')*H*(I - rho*yk*sk') + rho*(sk*sk');
63     end
64
65     x = xnew;
```

```
66
67      hist(k,:) = [k, fk, ng, alpha];
68  end
69
70  fprintf('Final␣x␣=␣[%g,␣%g],␣f␣=␣%.3e,␣||g||␣=␣%.3e,␣iters␣=␣%d\n', ...
71          x(1), x(2), f(x), norm(g(x)), size(hist,1));
72
73  % Plot convergence of f and gradient norm
74  figure;
75  semilogy(hist(:,1), hist(:,2), 'o-'); grid on;
76  xlabel('iteration'); ylabel('f(x_k)');
77
78  figure;
79  semilogy(hist(:,1), hist(:,3), 'o-'); grid on;
80  xlabel('iteration'); ylabel('||grad␣f(x_k)||');
```

What you should submit.

- Your MATLAB script implementing BFGS (you can use the above, but add your own comments).
- A plot of $f(x_k)$ vs. iteration and $\|\nabla f(x_k)\|$ vs. iteration.
- The final iterate x_k and iteration count.

Problem 2: Forward Euler and Backward Euler for a linear ODE (explicit updates + MATLAB)

Consider the linear ODE

$$y'(t) = \lambda y(t) + r(t), \quad t \in [0, T], \qquad y(0) = y_0,$$

with constant $\lambda \in \mathbb{R}$.

Let $\Delta t = T/N$, $t_n = n\Delta t$, and denote $y_n \approx y(t_n)$.

1. Forward Euler (explicit formula)

$$y_{n+1} = y_n + \Delta t\big(\lambda y_n + r(t_n)\big) = (1 + \lambda\Delta t)\, y_n + \Delta t\, r(t_n).$$

2. Backward Euler (explicit solve for linear case)

Backward Euler writes

$$y_{n+1} = y_n + \Delta t\big(\lambda y_{n+1} + r(t_{n+1})\big).$$

Because it is linear, you can solve for y_{n+1} explicitly:

$$(1 - \lambda\Delta t)\, y_{n+1} = y_n + \Delta t\, r(t_{n+1}), \qquad y_{n+1} = \frac{y_n + \Delta t\, r(t_{n+1})}{1 - \lambda\Delta t}.$$

Now use

$$\lambda = -10, \quad r(t) = \sin(t), \quad y_0 = 1, \quad T = 5.$$

What you should submit.

- MATLAB code implementing Forward Euler and Backward Euler.
- A plot comparing both methods for at least two different Δt.