Numerical Optimization — HW1

Deadline: October 8, 2025

1. (10 points) Let $A \in \mathbb{R}^{n\times n}$ be an invertible matrix and $B \in \mathbb{R}^{n\times n}$. If

$$\|A^{-1}(B-A)\|_2 < 1,$$

prove that B is also invertible, and moreover

$$\|B^{-1}\|_2 \le \frac{\|A^{-1}\|_2}{1-\|A^{-1}(A-B)\|_2}.$$

Solution:

1. Prove B is invertible

$B = A + (B - A)$.

$B = A\,(I + A^{-1}(B - A))$.

We have known that $\|A^{-1}(B-A)\|_2 = \|A^{-1}(A-B)\|_2 < 1$.

According to the properties of matrix inverse, we know if a matrix norm $\|X\| < 1$, then the matrix $(I - X)$ is invertible. This is because its inverse can be expressed as a convergent Neumann series: $(I-X)^{-1} = \sum_{k=0}^{\infty} X^k = I + X + X^2 + \dots$

And also: A matrix $\|X\| < 1$ implies all eigenvalues λ of X satisfy $|\lambda| < 1$. This means all eigenvalues of $(I - X)$ are non-zero, making $(I - X)$ invertible.

So $I + A^{-1}(B - A)$ is invertible, as the product of two invertible matrices is also invertible, so the matrix $B = A\,(I + A^{-1}(B - A))$ must also be invertible.

2. Prove the inequality

Let $X = A^{-1}(A - B)$

$B^{-1} = (A(I - X))^{-1} = (I - X)^{-1}A^{-1}$.

Because for two matrix $\|MN\| \le \|M\|\|N\|$,

So $\|B^{-1}\|_2 = \|(I-X)^{-1}A^{-1}\|_2 \le \|(I-X)^{-1}\|_2\|A^{-1}\|_2$.

and: $(I-X)^{-1} = I + X + X^2 + X^3 + \dots$

and because: $\|X^k\|_2 \le \|X\|_2^k$:

So: $\|(I-X)^{-1}\|_2 = \|\sum_{k=0}^{\infty} X^k\|_2 \le \sum_{k=0}^{\infty}\|X^k\|_2 \le \sum_{k=0}^{\infty}\|X\|_2^k = \frac{1}{1-\|X\|_2}$.

i.e. $\|(I-X)^{-1}\|_2 \le \frac{1}{1-\|X\|_2}$.

So $\|B^{-1}\|_2 \le \frac{1}{1-\|X\|_2}\|A^{-1}\|_2 = \frac{\|A^{-1}\|_2}{1-\|X\|_2}$.

Substituting this back into the final inequality :

$$\|B^{-1}\|_2 \le \frac{\|A^{-1}\|_2}{1-\|A^{-1}(A-B)\|_2}.$$

2. (15 points) Let $G : \mathbb{R}^n \to \mathbb{R}^m$ be a differentiable function, and suppose that its Jacobian ∇G is L-Lipschitz under the operator norm, i.e., for any $u, v \in \mathbb{R}^n$,

$$\|\nabla G(u) - \nabla G(v)\|_{op} \leq L\|u - v\|.$$

Prove that for any $x, y \in \mathbb{R}^n$, the inequality holds:

$$\|G(x) - G(y) - \nabla G(y)(x - y)\| \leq \tfrac{1}{2}L\|x - y\|^2.$$

Solution

Let $f(t) = G(y + t(x - y))$ for $t \in [0, 1]$

$$f'(t) = \nabla G(y + t(x - y))(x - y)$$

$$G(x) - G(y) = f(1) - f(0) = \int_0^1 f'(t)dt = \int_0^1 \nabla G(y + t(x - y))(x - y)dt$$

So

$$G(x) - G(y) - \nabla G(y)(x - y) = \int_0^1 \nabla G(y + t(x - y))(x - y)dt - \nabla G(y)(x - y)$$

And because

$$\nabla G(y)(x - y) = \int_0^1 \nabla G(y)(x - y)dt$$

Substituting this back into the equation:

$$G(x) - G(y) - \nabla G(y)(x - y) = \int_0^1 \left(\nabla G(y + t(x - y)) - \nabla G(y)\right)(x - y)dt$$

Take the norm of both sides :

$$\|G(x) - G(y) - \nabla G(y)(x - y)\| = \left\| \int_0^1 \left(\nabla G(y + t(x - y)) - \nabla G(y)\right)(x - y)dt \right\|$$

And using the inequality of integral:

$$\|G(x) - G(y) - \nabla G(y)(x - y)\| \leq \int_0^1 \left\|\left(\nabla G(y + t(x - y)) - \nabla G(y)\right)(x - y)\right\| dt$$

Using the inequality: $\|A\mathbf{v}\| \leq \|A\|_{op}\|\mathbf{v}\|$ for any matrix A and vector $\mathbf{v}$:

$$\left\|\left(\nabla G(y + t(x - y)) - \nabla G(y)\right)(x - y)\right\| \leq \|\nabla G(y + t(x - y)) - \nabla G(y)\|_{op}\|x - y\|$$

And becuase:

$$\|\nabla G(u) - \nabla G(v)\|_{op} \leq L\|u - v\|$$

So let $u = y + t(x - y)$ and $v = y$. Then:

$$\|\nabla G(y + t(x - y)) - \nabla G(y)\|_{op} \leq L\|(y + t(x - y)) - y\| = L\|t(x - y)\| = L|t|\|x - y\|$$

Since $t \in [0,1]$, $|t| = t$. Therefore:

$$\|\nabla G(y + t(x-y)) - \nabla G(y)\|_{op} \leq Lt\|x-y\|$$

Substituting this back:

$$\|G(x) - G(y) - \nabla G(y)(x-y)\| \leq \int_0^1 Lt\|x-y\| \cdot \|x-y\| dt$$

$$\|G(x) - G(y) - \nabla G(y)(x-y)\| \leq \int_0^1 Lt\|x-y\|^2 dt$$

$$\|G(x) - G(y) - \nabla G(y)(x-y)\| \leq L\|x-y\|^2 \int_0^1 t dt$$

$$\int_0^1 t dt = \frac{1}{2}$$

So:

$$\|G(x) - G(y) - \nabla G(y)(x-y)\| \leq \tfrac{1}{2}L\|x-y\|^2$$

3. (15 points) Suppose a sequence $\{x_k\}$ in $\mathbb{R}^n$ satisfies $\lim_{k\to\infty} \|x_k - x_{k+1}\|_2 = 0$.

(1) Does $\{x_k\}$ have limit points? Why?

(2) Does $\{x_k\}$ converge? If not, give an example. If yes, provide a proof.

(3) If $\sum_{k=1}^{\infty} \|x_k - x_{k-1}\|^2 < \infty$, does $\{x_k\}$ converge? If not, give a counterexample. If yes, provide a proof.

Solution

(1)

May not.

Because the sequence $\{x_k\}$ may not be bounded.

When $x_k = \sqrt{k}$,

$\|x_k - x_{k+1}\|_2 = \|\sqrt{k} - \sqrt{k+1}\|_2 = \|\frac{\sqrt{k}+\sqrt{k+1}}{k-(k+1)}\|_2 = \|\sqrt{k} + \sqrt{k+1}\|_2$

$\lim_{k\to\infty} \|x_k - x_{k+1}\|_2 = \lim_{k\to\infty} \|\sqrt{k} + \sqrt{k+1}\|_2 \to \infty$.

So $\{x_k\}$ may be unbounded, so it may not have limit points.

(2)

No.

When $x_k = \sqrt{k}$, according to (1), when $k \to \infty$, $x_k \to \infty$ so it is not converged.

(3)

For any $m > n$, we have:

$$\|x_m - x_n\|_2 = \left\| \sum_{j=n+1}^{m} (x_j - x_{j-1}) \right\|_2 \tag{1}$$

Using the Cauchy–Schwarz inequality :

$$\|x_m - x_n\|_2^2 \leq \sum_{j=n+1}^{m} \|x_j - x_{j-1}\|_2^2 \tag{2}$$

Since $\sum_{k=1}^{\infty} \|x_k - x_{k-1}\|^2 < \infty$, for any $\epsilon > 0$, there exists an N such that for all $n \geq N$:

$$\sum_{j=n+1}^{\infty} \|x_j - x_{j-1}\|_2^2 < \epsilon^2 \tag{3}$$

Therefore, for any $m > n \geq N$:

$$\|x_m - x_n\|_2^2 \leq \sum_{j=n+1}^{m} \|x_j - x_{j-1}\|_2^2 \tag{4}$$

$$< \epsilon^2 \tag{5}$$

This implies:

$$\|x_m - x_n\|_2 < \epsilon \tag{6}$$

Since $\mathbb{R}^n$ is complete, the sequence converges.

4. (30 points) **Oil Refinery Problem (Lecture 0, page 30).**

   Consider the oil refinery optimization problem given in Lecture 0 (page 30). Your task is to implement this problem in AMPL and solve it either by:

   - Submitting the AMPL model to the NEOS server, or
   - Using a solver available in other optimization software (e.g., CPLEX, Gurobi, IPOPT).

   You may choose your own initial values if required by the solver.

   In your report, you should clearly provide:

   (a) The complete AMPL model (or the code in the solver/software you used).

   (b) The solution obtained (values of all decision variables).

   (c) The optimal value of the objective function.

   (d) The solver's exit flag or termination status.

   (e) A brief explanation (in English) of how you set up the problem, selected the initial values, and interpreted the results.

   Solution

   AMPL model file

   Listing 1: AMPL Model File

```
set CRUDE_OILS;
set PRODUCTS;

param Sulfur{CRUDE_OILS};
param Price{PRODUCTS};
param Cost{CRUDE_OILS};
param Demand{PRODUCTS};
param SulfurLimit{PRODUCTS};
param UB >= 0;

var x >= 0, <= Demand['x'];
var y >= 0, <= Demand['y'];
var A >= 0, <= UB;
var B >= 0, <= UB;
var C_x >= 0, <= UB;
var C_y >= 0, <= UB;
var P_x >= 0, <= UB;
var P_y >= 0, <= UB;

var p >= 0, <= 100;

maximize Profit:
    Price['x'] * x + Price['y'] * y
  - Cost['A'] * A - Cost['B'] * B - Cost['C'] * (C_x + C_y);
```

```
subject to PoolFlowBalance:
    P_x + P_y = A + B;

subject to ProductXFlowBalance:
    x = P_x + C_x;

subject to ProductYFlowBalance:
    y = P_y + C_y;

subject to PoolSulfurBalance:
    Sulfur['A'] * A + Sulfur['B'] * B = p * (P_x + P_y);

subject to ProductXSulfurContent:
    p * P_x + Sulfur['C'] * C_x <= SulfurLimit['x'] * x;

subject to ProductYSulfurContent:
    p * P_y + Sulfur['C'] * C_y <= SulfurLimit['y'] * y;
```

Data file

Listing 2: Data File

```
set CRUDE_OILS := A B C;
set PRODUCTS   := x y;
param Sulfur :=
  A 3
  B 1
  C 2 ;
param Price :=
  x 9
  y 15 ;
param Cost :=
  A 6
  B 8
  C 10 ;
param Demand :=
  x 200
  y 200 ;
param SulfurLimit :=
  x 2.5
  y 1.5 ;
param UB := 500;
```

Command file

Listing 3: Command file

```
option solver baron;
option baron_options 'maxtime=300 epsr=1e-6 epsa=1e-8 outlev=1';
solve;
display x, y, A, B, P_x, P_y, C_x, C_y, p;
display "Objective Value:", _obj;
```

The solution obtained is:

$$
\begin{aligned}
x &= 200, \\
y &= 200, \\
A &= 100, \\
B &= 300, \\
P_x &= 200, \\
P_y &= 200, \\
C_x &= -8.83437e-11 \approx 0, \\
C_y &= -4.61011e-12 \approx 0, \\
p &= 1.5
\end{aligned}
$$

The optimal value of the objective function is 1800

The exit flag is the solver successfully found a globally optimal solution that satisfies all problem constraints and objective function within the standard numerical tolerances. The logs are as below:

Listing 4: Exit flag and termination status

```
BARON 25.8.5 (2025.08.05): threads=4
maxtime=300
epsr=1e-6
epsa=1e-8
outlev=1

===========================================================================
 Preprocessing found feasible solution with value 0.00000
 Doing local search
 Preprocessing found feasible solution with value 500.000
 Solving bounding LP
 Starting multi-start local search
 Preprocessing found feasible solution with value 1800.00
 Done with local search
===========================================================================
  Iteration      Time (s)      Mem    Lower bound      Upper bound
Progress
          1          0.04     12MB      1800.00          1800.00
100.00%

                        *** Normal completion ***

 Wall clock time:                     0.04
 Total CPU time used:                 0.04

 Total no. of BaR iterations:       1
```

```
Best solution found at node:       1
Max. no. of nodes in memory:       1

All done
```

```
BARON 25.8.5 (2025.08.05): 1 iterations, optimal within tolerances.
```

Model Setup

Variables and Parameters

Sets: CRUDE_OILS $= \{A, B, C\}$, PRODUCTS $= \{x, y\}$.

Parameters: Sulfur contents (Sulfur_i); product prices (Price_j); crude costs (Cost_i); demands (Demand_j); sulfur limits (SulfurLimit_j); capacity bound (UB).

Decision Variables: Product outputs (x, y); crude-to-pool flows (A, B); direct crude C splits (C_x, C_y); pool-to-product flows (P_x, P_y); pool sulfur fraction (p).

Objective Function

The objective is to maximize the net profit:

$$\max\ \text{Price}_x x + \text{Price}_y y - \text{Cost}_A A - \text{Cost}_B B - \text{Cost}_C(C_x + C_y).$$

Constraints

The system is subject to the following mass balance and quality constraints:

$$\begin{aligned}
&\text{Pool Mass:} && P_x + P_y = A + B, \\
&\text{Mixer } x: && x = P_x + C_x, \qquad \text{Mixer } y: \qquad y = P_y + C_y, \\
&\text{Pool Sulfur:} && \text{Sulfur}_A A + \text{Sulfur}_B B = p(P_x + P_y), \\
&\text{Spec } x: && pP_x + \text{Sulfur}_C C_x \le \text{SulfurLimit}_x\, x, \\
&\text{Spec } y: && pP_y + \text{Sulfur}_C C_y \le \text{SulfurLimit}_y\, y.
\end{aligned}$$

The problem's **nonconvexity** arises from bilinear terms like pP_x, pP_y, and $p(P_x + P_y)$. Due to this, the global optimization solver **BARON** (Global NLP) is required.

Initial Values

Default AMPL zeros are generally sufficient, as BARON performs rigorous bound tightening before starting the main algorithm.

Result Interpretation

- **Optimal Production (x, y):** These are the profit-maximizing production levels, checked against demand bounds.
- **Optimal Flows $(A, B, C_x, C_y, P_x, P_y)$:** These define the optimal crude allocation and blending strategy.
- **Pool Sulfur (p):** The value of p indicates the sulfur level of the intermediate pool. Binding sulfur specifications (Spec x or Spec y) suggest that product quality, rather than cost or demand, is the limiting factor.
- **Objective Value:** The final value represents the global maximum profit. The reported dual feasibility and the small optimality gap (within specified epsr, epsa) certify global optimality.

5. (30 points) **Multivariable Newton's Method: Inverse Kinematics of a Robot Arm**

Application background: Consider a planar two-link robotic arm:

- The first link has length $L_1 = 2$ m.
- The second link has length $L_2 = 1$ m.

The end-effector position (x, y) is related to the two joint angles θ_1, θ_2 through the forward kinematics equations:

$$x = L_1 \cos(\theta_1) + L_2 \cos(\theta_1 + \theta_2), \quad y = L_1 \sin(\theta_1) + L_2 \sin(\theta_1 + \theta_2).$$

Inverse kinematics problem: Given a desired target position $(x_a, y_a) = (2.0\text{m}, 1.0\text{m})$ for the end effector, use the *multivariable Newton's method* to solve for the joint angles θ_1 and θ_2.

Your program should:

(a) Implement the multivariable Newton's method.

(b) Start from an initial guess for (θ_1, θ_2) and iterate until convergence.

(c) Output the computed joint angles (θ_1, θ_2), the number of iterations, and the final residual error **similar to Lecture 2, page 28**.

Solution

$\mathbf{F}(\boldsymbol{\theta}) = \mathbf{0}$, where $\boldsymbol{\theta} = \begin{pmatrix} \theta_1 \\ \theta_2 \end{pmatrix}$ and $\mathbf{F}(\boldsymbol{\theta})$:

$$\mathbf{F}(\theta_1, \theta_2) = \begin{pmatrix} f_1(\theta_1, \theta_2) \\ f_2(\theta_1, \theta_2) \end{pmatrix} = \begin{pmatrix} x - x_a \\ y - y_a \end{pmatrix}$$

$$f_1(\theta_1, \theta_2) = 2\cos(\theta_1) + \cos(\theta_1 + \theta_2) - 2$$
$$f_2(\theta_1, \theta_2) = 2\sin(\theta_1) + \sin(\theta_1 + \theta_2) - 1$$

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \mathbf{J}^{-1}(\boldsymbol{\theta}^{(k)})\mathbf{F}(\boldsymbol{\theta}^{(k)})$$

$J_{ij} = \frac{\partial f_i}{\partial \theta_j}$: So

$$\mathbf{J}(\theta_1, \theta_2) = \begin{pmatrix} \frac{\partial f_1}{\partial \theta_1} & \frac{\partial f_1}{\partial \theta_2} \\ \frac{\partial f_2}{\partial \theta_1} & \frac{\partial f_2}{\partial \theta_2} \end{pmatrix}$$

$$\mathbf{J}(\theta_1, \theta_2) = \begin{pmatrix} -2\sin(\theta_1) - \sin(\theta_1 + \theta_2) & -\sin(\theta_1 + \theta_2) \\ 2\cos(\theta_1) + \cos(\theta_1 + \theta_2) & \cos(\theta_1 + \theta_2) \end{pmatrix}$$

Then we use python to do all steps(we use $\boldsymbol{\theta_0} = \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}$ as the initial value):

```
import numpy as np

def forward_kinematics(theta, L1=2.0, L2=1.0):
    theta1, theta2 = theta
    x = L1 * np.cos(theta1) + L2 * np.cos(theta1 + theta2)
    y = L1 * np.sin(theta1) + L2 * np.sin(theta1 + theta2)
    return np.array([x, y])

def residual(theta, target, L1=2.0, L2=1.0):
    """F(theta) = f(theta) - target"""
    return forward_kinematics(theta, L1, L2) - target

def jacobian(theta, L1=2.0, L2=1.0):
    """compute J"""
    theta1, theta2 = theta
    J=np.array([
        [-L1*np.sin(theta1) - L2*np.sin(theta1 + theta2),
            -L2*np.sin(theta1 + theta2)],
        [ L1*np.cos(theta1) + L2*np.cos(theta1 + theta2),
            L2*np.cos(theta1 + theta2)]
    ])
    return J

def multivariable_newton(target=np.array([2.0, 1.0]),
                         theta0=np.array([0.5, 0.5]),
                         tol=1e-8, max_iter=50):
    theta = theta0.copy()
    history = []
    for k in range(1, max_iter + 1):
        F = residual(theta, target)
        J = jacobian(theta)
        delta = np.linalg.solve(J, F)
        theta_next = theta - delta
        err = np.linalg.norm(delta)
        history.append((k, theta[0], theta[1], F[0], F[1], err))
        theta = theta_next
        if err < tol:
            break
    return theta, history
def print_table(history):
    rows = ""
    rows+=f"k & theta_1 & theta_2 & f_1 & f_2 & ||Delta x|| \n"
    for (k, t1, t2, f1, f2, err) in history:
        rows+=f"{k} & {t1:.6f} & {t2:.6f} & {f1:.6f} & {f2:.6f} & {err:.6e} \n"
    return rows
theta_sol, history = multivariable_newton()
print("theta:", theta_sol)
print("Iterate times:", len(history))
```

```
print(print_table(history))
```

Then we get the solution:

$$\boldsymbol{\theta} = \begin{pmatrix} 1.26201044e-18 \\ 1.57079633e+00 \end{pmatrix}$$

i.e.

$$\theta_1 = 1.26201044e-18, \theta_2 = 1.57079633e+00$$

k	θ_1	θ_2	f_1	f_2	$\|\Delta\theta\|$
1	0.500000	0.500000	0.295467	0.800322	2.374667e+00
2	-0.368841	2.710013	-0.830913	-1.003421	1.169309e+00
3	-0.199764	1.552993	0.176082	-0.420450	1.989272e-01
4	-0.013469	1.622752	-0.038659	-0.027678	5.307944e-02
5	-0.000366	1.571315	-0.000153	-0.000733	6.353004e-04
6	-0.000000	1.570796	-0.000000	-0.000000	1.402003e-07
7	-0.000000	1.570796	-0.000000	-0.000000	6.517775e-15