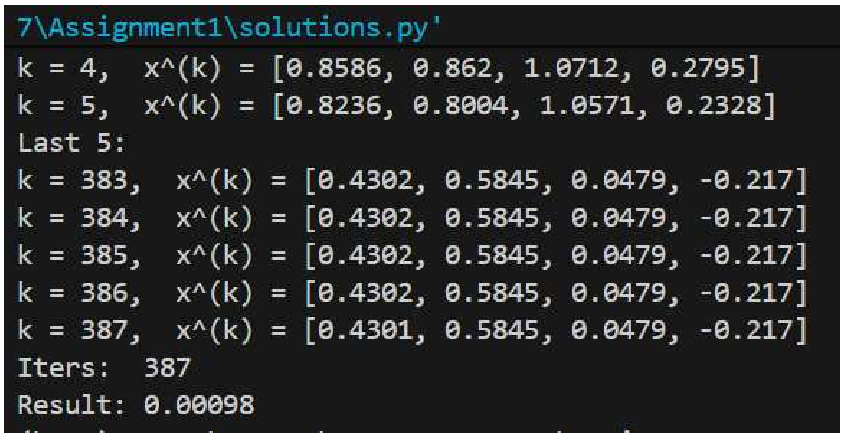
Assignment 1

COMP-9417

Tian Huang

Oct 11th 2025

Part A:



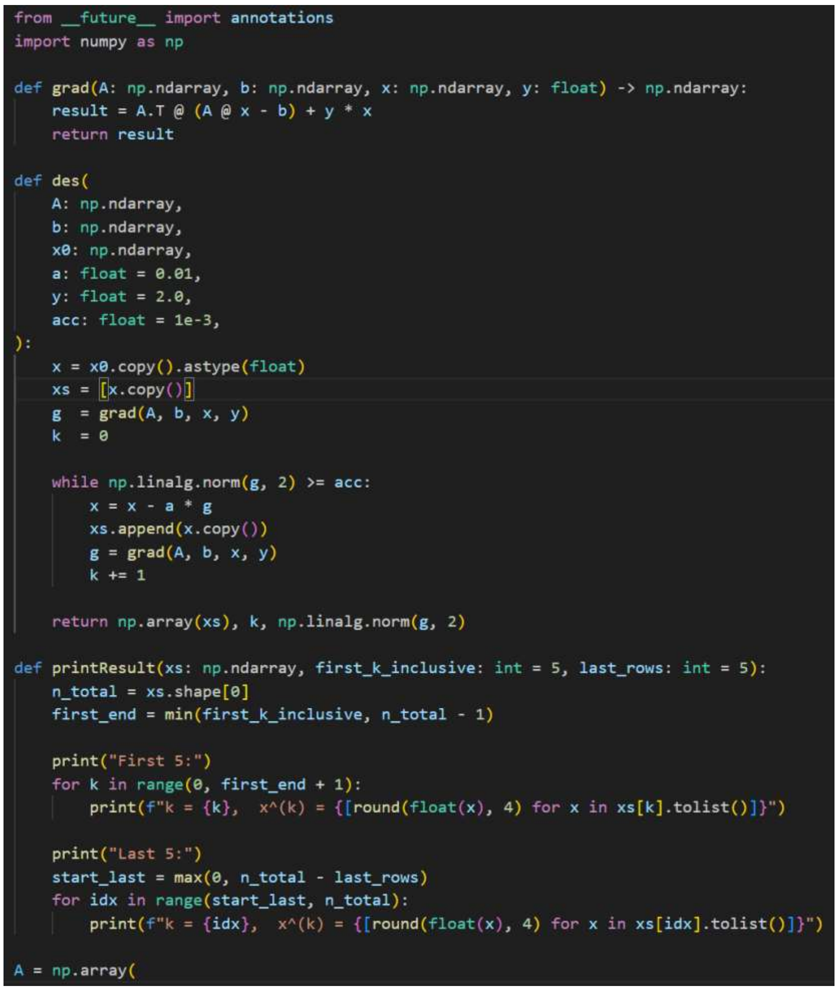
```
7\Assignment1\solutions.py'
k = 4,   x^(k) = [0.8586, 0.862, 1.0712, 0.2795]
k = 5,   x^(k) = [0.8236, 0.8004, 1.0571, 0.2328]
Last 5:
k = 383,   x^(k) = [0.4302, 0.5845, 0.0479, -0.217]
k = 384,   x^(k) = [0.4302, 0.5845, 0.0479, -0.217]
k = 385,   x^(k) = [0.4302, 0.5845, 0.0479, -0.217]
k = 386,   x^(k) = [0.4302, 0.5845, 0.0479, -0.217]
k = 387,   x^(k) = [0.4301, 0.5845, 0.0479, -0.217]
Iters:  387
Result: 0.00098
```

$$f(x) = \frac{1}{2}\|Ax-b\|_2^2 + \frac{\gamma}{2}\|x\|_2^2$$

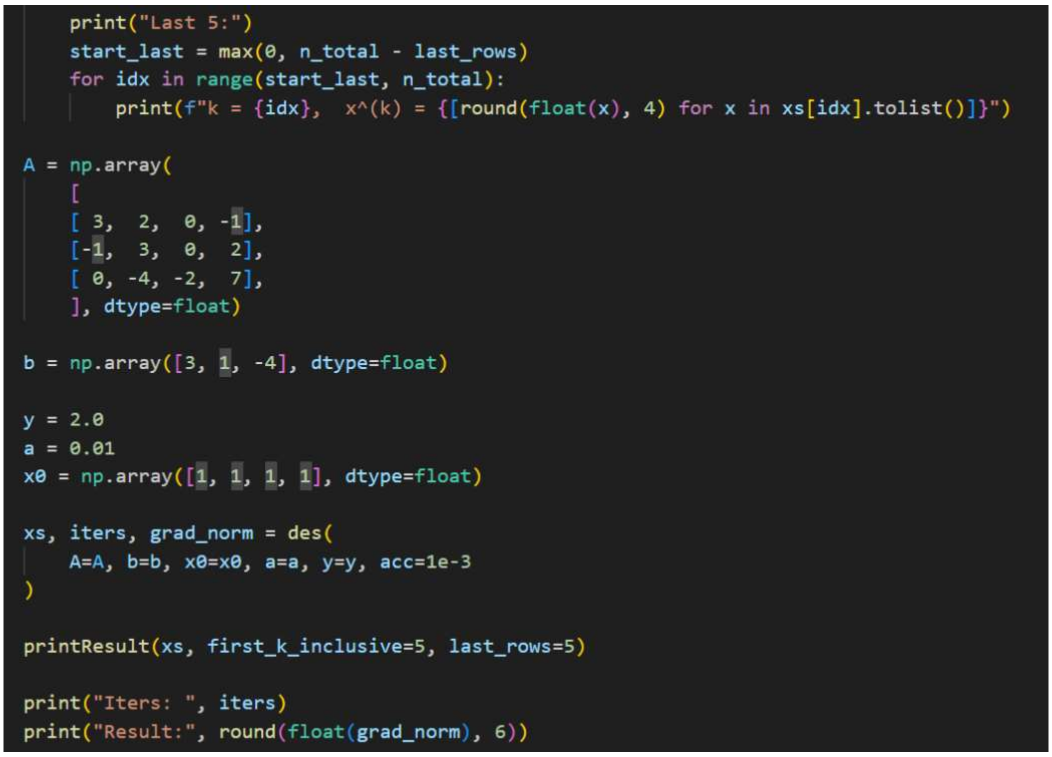
$$\nabla f(x) = A^T(Ax-b) + \gamma x$$

$$x^{(k+1)} = x^{(k)} - \alpha \nabla f(x^{(k)})$$

$$= x^{(k)} - \alpha\left[A^T(Ax^{(k)} - b) + \gamma x^{(k)}\right]$$

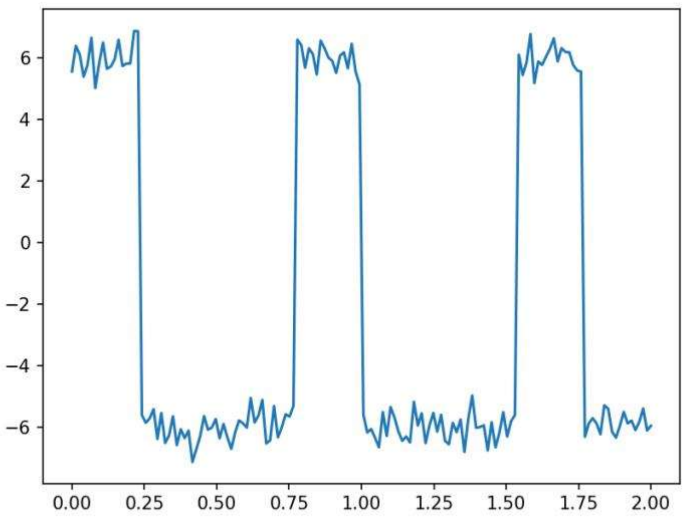


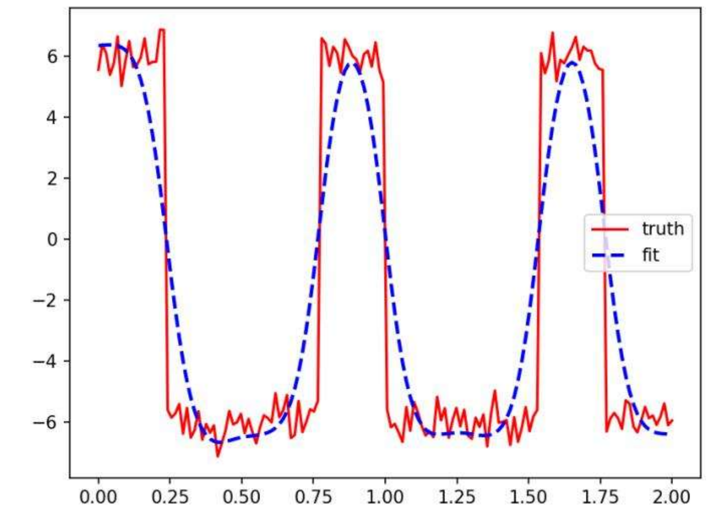
```python
from __future__ import annotations
import numpy as np

def grad(A: np.ndarray, b: np.ndarray, x: np.ndarray, y: float) -> np.ndarray:
    result = A.T @ (A @ x - b) + y * x
    return result

def des(
    A: np.ndarray,
    b: np.ndarray,
    x0: np.ndarray,
    a: float = 0.01,
    y: float = 2.0,
    acc: float = 1e-3,
):
    x = x0.copy().astype(float)
    xs = [x.copy()]
    g  = grad(A, b, x, y)
    k  = 0

    while np.linalg.norm(g, 2) >= acc:
        x = x - a * g
        xs.append(x.copy())
        g = grad(A, b, x, y)
        k += 1

    return np.array(xs), k, np.linalg.norm(g, 2)

def printResult(xs: np.ndarray, first_k_inclusive: int = 5, last_rows: int = 5):
    n_total = xs.shape[0]
    first_end = min(first_k_inclusive, n_total - 1)

    print("First 5:")
    for k in range(0, first_end + 1):
        print(f"k = {k},  x^(k) = {[round(float(x), 4) for x in xs[k].tolist()]}")

    print("Last 5:")
    start_last = max(0, n_total - last_rows)
    for idx in range(start_last, n_total):
        print(f"k = {idx},  x^(k) = {[round(float(x), 4) for x in xs[idx].tolist()]}")

A = np.array(
```



```python
    print("Last 5:")
    start_last = max(0, n_total - last_rows)
    for idx in range(start_last, n_total):
        print(f"k = {idx},  x^(k) = {[round(float(x), 4) for x in xs[idx].tolist()]}")

A = np.array(
    [
    [ 3,  2,  0, -1],
    [-1,  3,  0,  2],
    [ 0, -4, -2,  7],
    ], dtype=float)

b = np.array([3, 1, -4], dtype=float)

y = 2.0
a = 0.01
x0 = np.array([1, 1, 1, 1], dtype=float)

xs, iters, grad_norm = des(
    A=A, b=b, x0=x0, a=a, y=y, acc=1e-3
)

printResult(xs, first_k_inclusive=5, last_rows=5)

print("Iters: ", iters)
print("Result:", round(float(grad_norm), 6))
```

Part B:





$L(\beta) = \frac{1}{2p}\|y-\beta\|_2^2 + \lambda\|W\beta\|_2^2 \qquad y, \beta \in R^p, \lambda > 0$

$\nabla_\beta L(\beta) = \frac{1}{p}(\beta - y) + 2\lambda W^T W\beta$

$\frac{1}{p}(\beta - y) + 2\lambda W^T W \beta = 0$

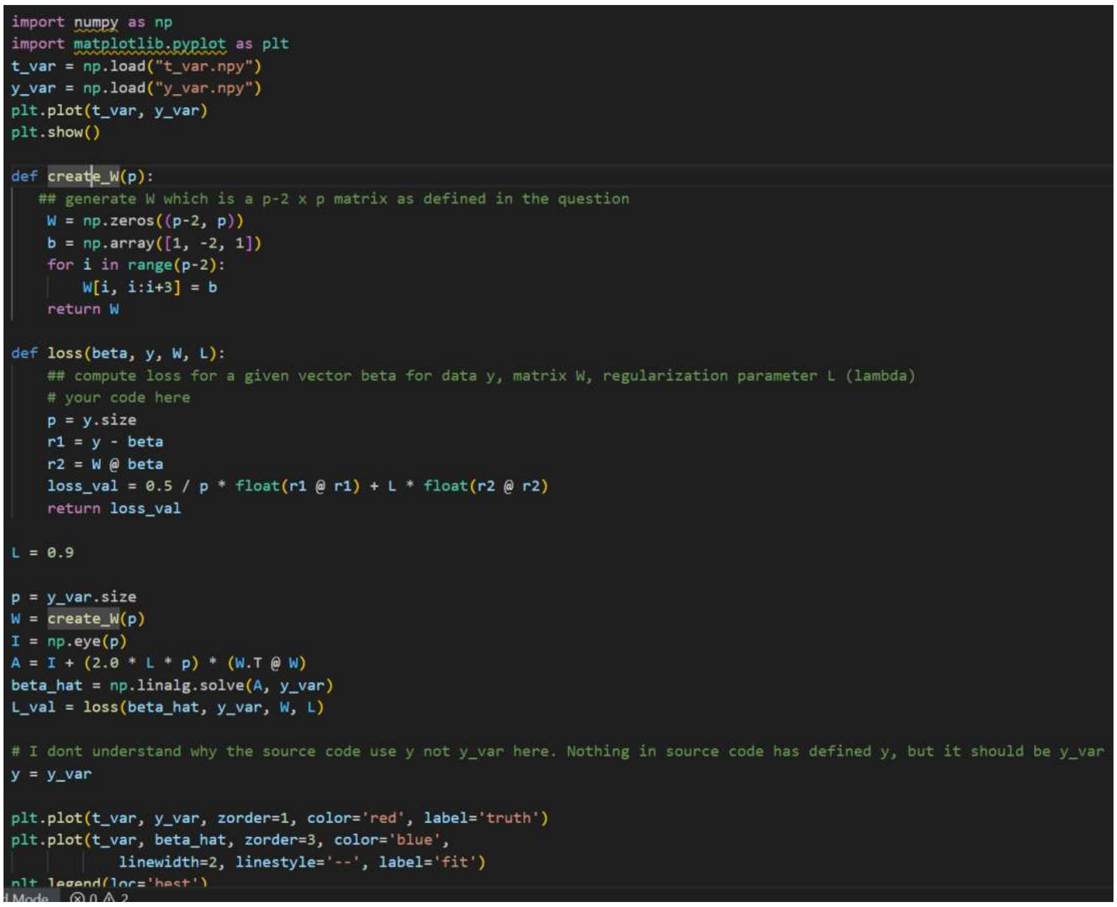
$$\Downarrow$$

$(\frac{1}{p} I + 2\lambda W^T W)\beta = \frac{1}{p} y$

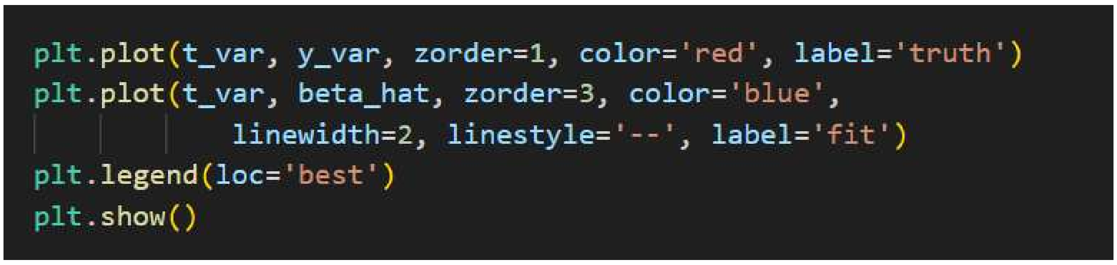
$\therefore (I + 2\lambda p W^T W)\beta = y$

$\because W^T W \geqslant 0$

$\therefore \hat{\beta} = (I + 2\lambda p W^T W)^{-1} y$



```python
import numpy as np
import matplotlib.pyplot as plt
t_var = np.load("t_var.npy")
y_var = np.load("y_var.npy")
plt.plot(t_var, y_var)
plt.show()

def create_W(p):
    ## generate W which is a p-2 x p matrix as defined in the question
    W = np.zeros((p-2, p))
    b = np.array([1, -2, 1])
    for i in range(p-2):
        W[i, i:i+3] = b
    return W

def loss(beta, y, W, L):
    ## compute loss for a given vector beta for data y, matrix W, regularization parameter L (lambda)
    # your code here
    p = y.size
    r1 = y - beta
    r2 = W @ beta
    loss_val = 0.5 / p * float(r1 @ r1) + L * float(r2 @ r2)
    return loss_val

L = 0.9

p = y_var.size
W = create_W(p)
I = np.eye(p)
A = I + (2.0 * L * p) * (W.T @ W)
beta_hat = np.linalg.solve(A, y_var)
L_val = loss(beta_hat, y_var, W, L)

# I dont understand why the source code use y not y_var here. Nothing in source code has defined y, but it should be y_var
y = y_var

plt.plot(t_var, y_var, zorder=1, color='red', label='truth')
plt.plot(t_var, beta_hat, zorder=3, color='blue',
            linewidth=2, linestyle='--', label='fit')
plt.legend(loc='best')
```



```python
plt.plot(t_var, y_var, zorder=1, color='red', label='truth')
plt.plot(t_var, beta_hat, zorder=3, color='blue',
            linewidth=2, linestyle='--', label='fit')
plt.legend(loc='best')
plt.show()
```

Part C:

Data fit:

$$\frac{1}{2p}\|y-\beta\|_2^2 = \frac{1}{2p}\sum_{i=1}^{p}(y_i-\beta_i)^2$$

It push β_i to y_i. The grad is $\frac{1}{p}(\beta-y)$

Smoothness:

$$\lambda\|W\beta\|_2^2 = \lambda\sum_{i=1}^{p-2}(\beta_i - 2\beta_{i+1} + \beta_{i+2})^2$$

Make β in range i to $i+2$ like a line, smooth,

The grad is $2\lambda W^T W\beta$

Part D:

$$L(\beta) = \frac{1}{2p} \|y-\beta\|_2^2 + \lambda \|W\beta\|_2^2$$

$$= \frac{1}{2p} \sum_{i=1}^{p} (y_i - \beta_i)^2 + \lambda \|W\beta\|_2^2$$

$$L_j(\beta) = \frac{1}{2} (y_j + \beta_j)^2 + \lambda \|W\beta\|_2^2$$

$j = 1, 2, \cdots p$

$$\frac{1}{p} \sum_{j=1}^{p} L_j(\beta) = \frac{1}{p} \left(\frac{1}{2} \sum_{j=1}^{p} (y_j - \beta_j)^2 + \sum_{j=1}^{p} \lambda \|W\beta\|_2^2 \right)$$

$$= \frac{1}{2p} \sum_{j=1}^{p} (y_j - \beta_j)^2 + \|W\beta\|_2^2 = L(\beta)$$

For $\nabla L_j(\beta)$

$$\nabla_\beta \left[\frac{1}{2} (y_j - \beta_j)^2 \right] = \frac{d}{d\beta_j} \frac{1}{2} (y_j - \beta_j)^2 = -(y_j - \beta_j)$$

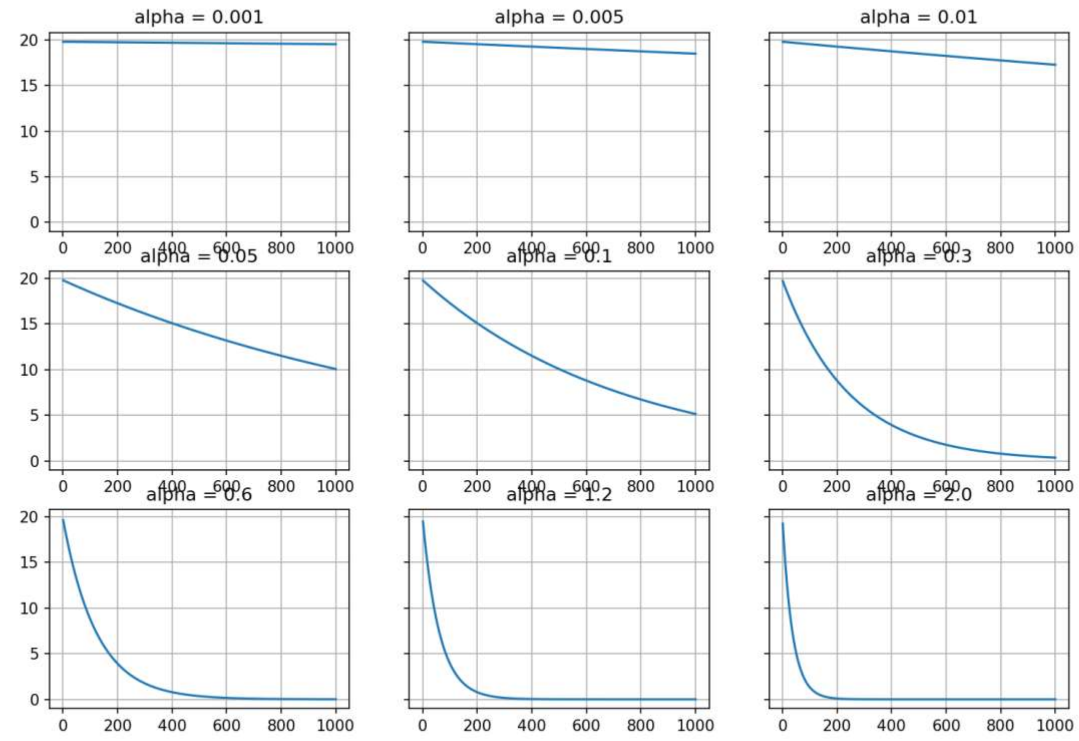
$$\nabla_\beta \left[\lambda \|W\beta\|_2^2 \right] = 2\lambda W^T W \beta$$

$$\nabla L_j(\beta) = -(y_j - \beta_j) e_j + 2\lambda W^T W \beta$$

for $j = 1 \dots p$

$$\nabla L(\beta) = \frac{1}{p} \sum_{j=1}^{p} \nabla L_j(\beta) = \frac{1}{p} (\beta - y) + 2\lambda W^T W \beta$$

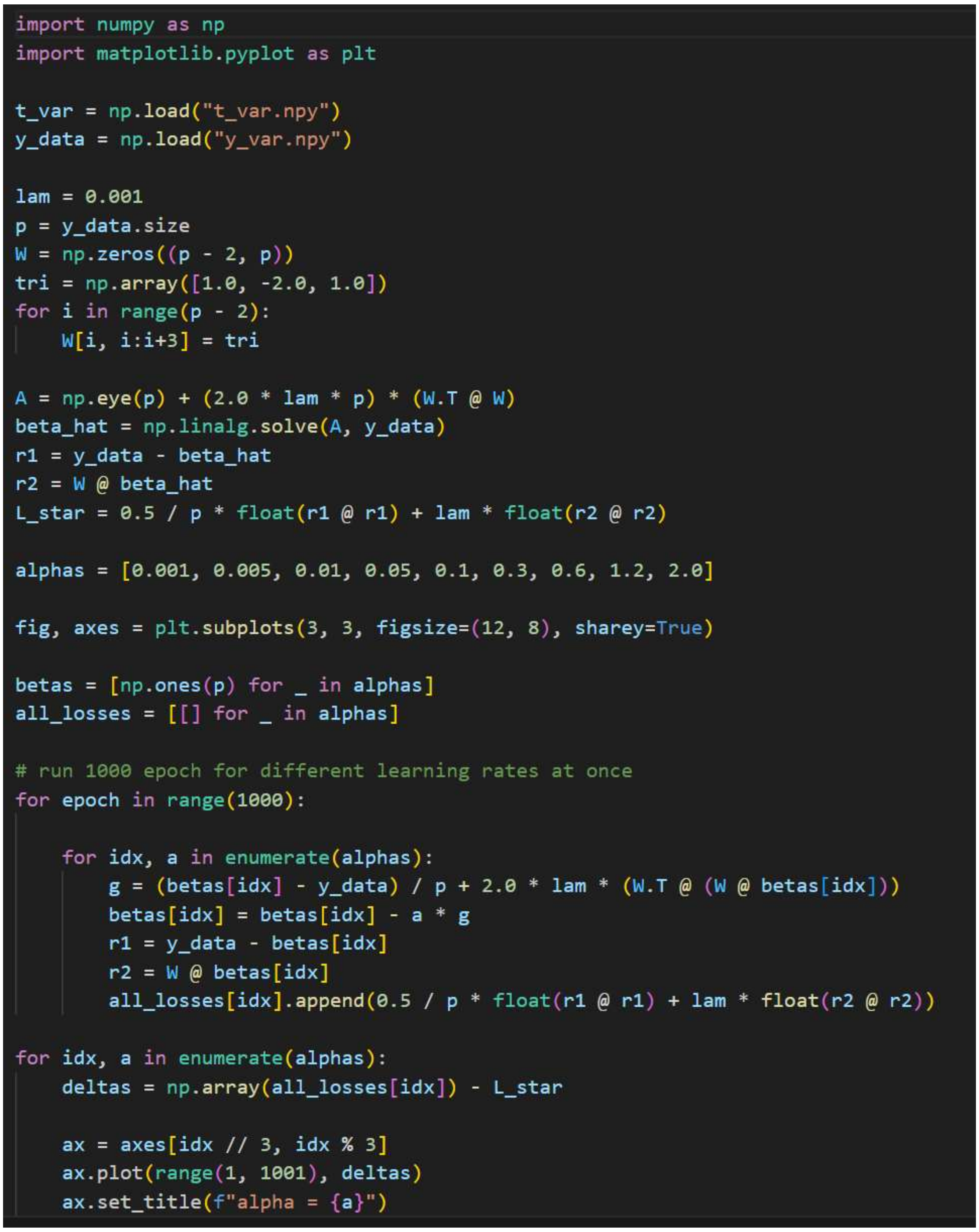
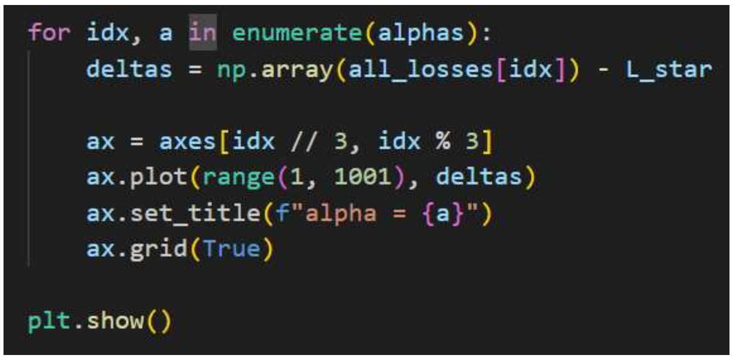
Part E:



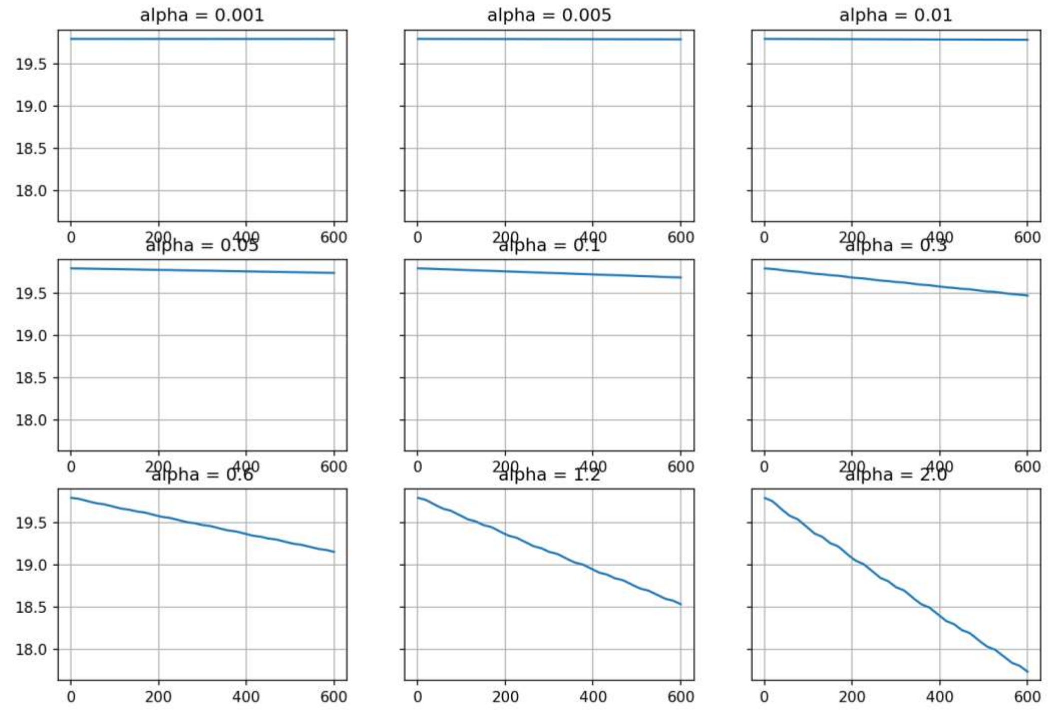
For $\alpha = 0.001, 0.005, 0.01, 0.05, 0.1$
they decent too slow that does not reach the bottom

For $\alpha \geqslant 0.3$, the reach the bottom, which means their decent speed is fast enough for 1000 epoch

So $\alpha = 0.6$ would be the best choice, stable and fast enough


```python
import numpy as np
import matplotlib.pyplot as plt

t_var = np.load("t_var.npy")
y_data = np.load("y_var.npy")

lam = 0.001
p = y_data.size
W = np.zeros((p - 2, p))
tri = np.array([1.0, -2.0, 1.0])
for i in range(p - 2):
    W[i, i:i+3] = tri

A = np.eye(p) + (2.0 * lam * p) * (W.T @ W)
beta_hat = np.linalg.solve(A, y_data)
r1 = y_data - beta_hat
r2 = W @ beta_hat
L_star = 0.5 / p * float(r1 @ r1) + lam * float(r2 @ r2)

alphas = [0.001, 0.005, 0.01, 0.05, 0.1, 0.3, 0.6, 1.2, 2.0]

fig, axes = plt.subplots(3, 3, figsize=(12, 8), sharey=True)

betas = [np.ones(p) for _ in alphas]
all_losses = [[] for _ in alphas]

# run 1000 epoch for different learning rates at once
for epoch in range(1000):

    for idx, a in enumerate(alphas):
        g = (betas[idx] - y_data) / p + 2.0 * lam * (W.T @ (W @ betas[idx]))
        betas[idx] = betas[idx] - a * g
        r1 = y_data - betas[idx]
        r2 = W @ betas[idx]
        all_losses[idx].append(0.5 / p * float(r1 @ r1) + lam * float(r2 @ r2))

for idx, a in enumerate(alphas):
    deltas = np.array(all_losses[idx]) - L_star

    ax = axes[idx // 3, idx % 3]
    ax.plot(range(1, 1001), deltas)
    ax.set_title(f"alpha = {a}")
```

```python
for idx, a in enumerate(alphas):
    deltas = np.array(all_losses[idx]) - L_star

    ax = axes[idx // 3, idx % 3]
    ax.plot(range(1, 1001), deltas)
    ax.set_title(f"alpha = {a}")
    ax.grid(True)

plt.show()
```

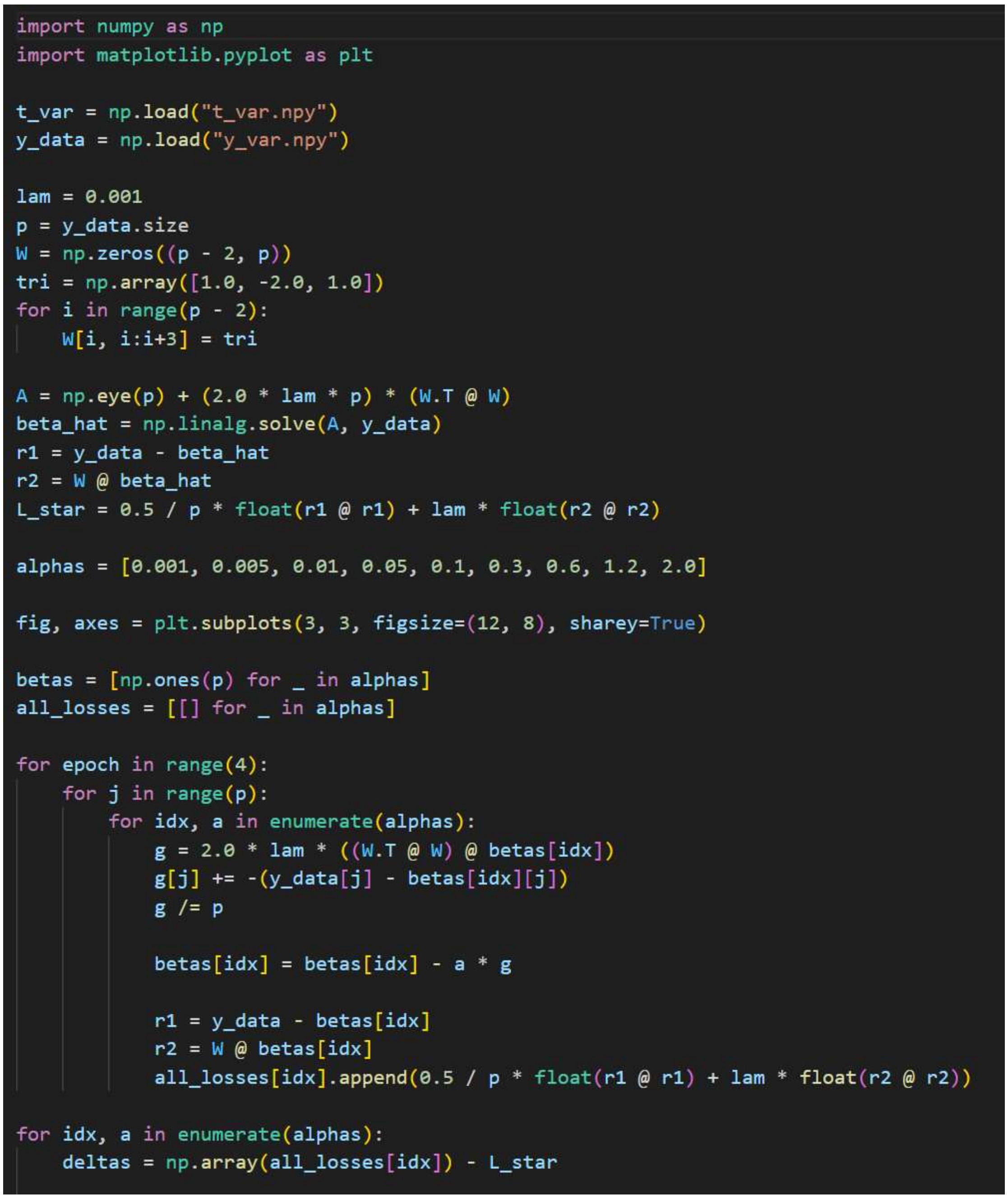
Part F:



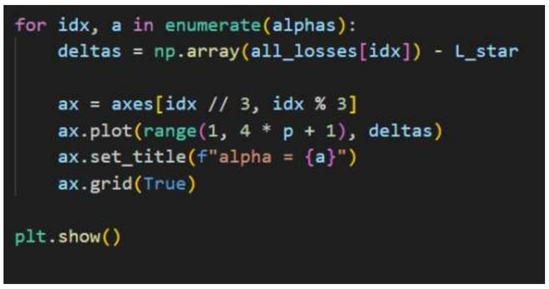
$L(\beta) \Rightarrow \frac{1}{p} L_i(\beta)$

Grad is $\frac{1}{p} \nabla L_i(\beta)$

For each single sample, it could possibily push β to different direction, it will not happend in GD, which is more stable.
The α is not very big, so the shack is not big in this case. but ~~it~~ is easy to see when α increase shack increase
I think the best α should be 0.05. the shack is not big but keep decenting. In 600 round it not reach the turning point.
but $\alpha = 2.0$ neither as well.



```python
import numpy as np
import matplotlib.pyplot as plt

t_var = np.load("t_var.npy")
y_data = np.load("y_var.npy")

lam = 0.001
p = y_data.size
W = np.zeros((p - 2, p))
tri = np.array([1.0, -2.0, 1.0])
for i in range(p - 2):
    W[i, i:i+3] = tri

A = np.eye(p) + (2.0 * lam * p) * (W.T @ W)
beta_hat = np.linalg.solve(A, y_data)
r1 = y_data - beta_hat
r2 = W @ beta_hat
L_star = 0.5 / p * float(r1 @ r1) + lam * float(r2 @ r2)

alphas = [0.001, 0.005, 0.01, 0.05, 0.1, 0.3, 0.6, 1.2, 2.0]

fig, axes = plt.subplots(3, 3, figsize=(12, 8), sharey=True)

betas = [np.ones(p) for _ in alphas]
all_losses = [[] for _ in alphas]

for epoch in range(4):
    for j in range(p):
        for idx, a in enumerate(alphas):
            g = 2.0 * lam * ((W.T @ W) @ betas[idx])
            g[j] += -(y_data[j] - betas[idx][j])
            g /= p

            betas[idx] = betas[idx] - a * g

            r1 = y_data - betas[idx]
            r2 = W @ betas[idx]
            all_losses[idx].append(0.5 / p * float(r1 @ r1) + lam * float(r2 @ r2))

for idx, a in enumerate(alphas):
    deltas = np.array(all_losses[idx]) - L_star
```



```python
for idx, a in enumerate(alphas):
    deltas = np.array(all_losses[idx]) - L_star

    ax = axes[idx // 3, idx % 3]
    ax.plot(range(1, 4 * p + 1), deltas)
    ax.set_title(f"alpha = {a}")
    ax.grid(True)

plt.show()
```

Part G:

$$\nabla_\beta L(\beta) = \frac{1}{p}(\beta - y) + 2\lambda W^T W_\beta$$

for j th as 0

$$\frac{1}{p}(\beta_j - y_j) + 2\lambda (W^T W_\beta)_j = 0$$

$$\Downarrow$$

$$\beta_j = y_j \quad -2\lambda \quad , \quad p(W^T W\beta)_j$$

For $W^T W$ for $p =$:

Each line as [1, -2, 1] in 3 lines

$M = W^T W$ for $j =$ ..... 3 $p-2$:

$$M(\beta)_j = \beta_{j-2} - 4\beta_{j-1} + 6\beta_j - 4\beta_{j+1} + \beta_{j+2}$$

$$(M\beta)_1 = \beta_1 - 2\beta_2 + \beta_3$$

$$(M\beta)_2 = -2\beta_1 + 5\beta_2 - 4\beta_3 + \beta_4$$

$$(M\beta)_{p-1} = \beta_{p-3} - 4\beta_{p-2} + 5\beta_{p-1} - 2\beta_p$$

$$(M\beta)_p = \beta_{p-2} - 2\beta_{p-1} + \beta_p$$

$$\therefore \hat{\beta}_j = \frac{y_j + 8\lambda p(\beta_{j-1} + \beta_{j+1}) - 2\lambda p(\beta_{j-2} + \beta_{j+2})}{1 + 12\lambda p}$$

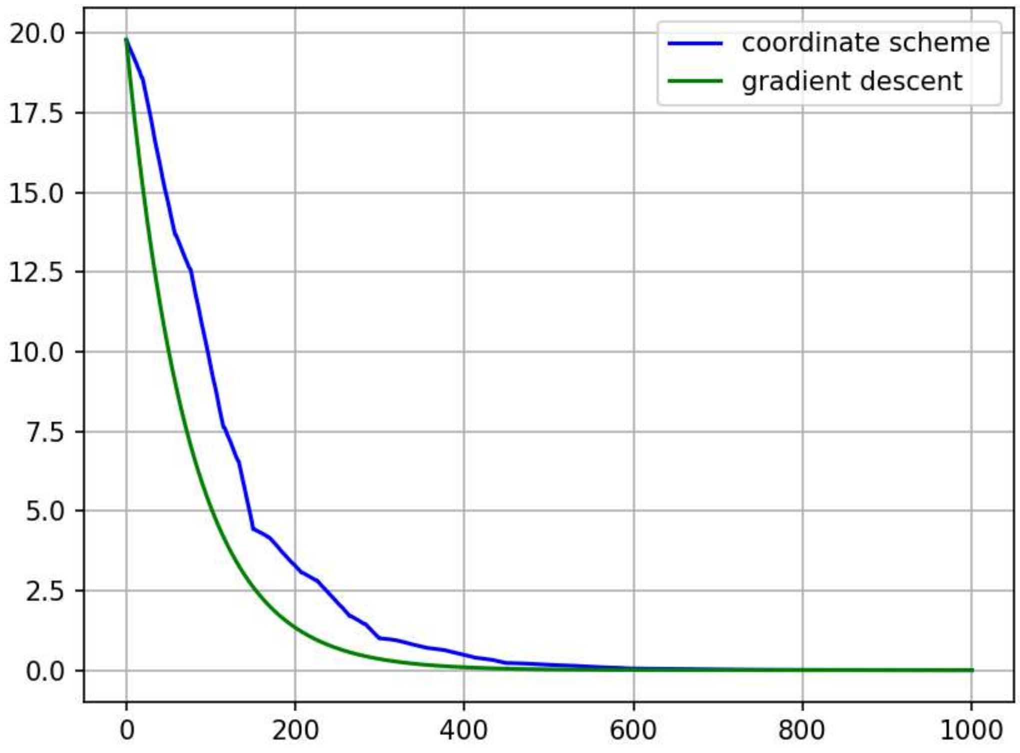
$$\hat{\beta}_1 = \frac{y_1 + 4\lambda p\beta_2 - 2\lambda\beta_3}{1 + 2\lambda p}$$

$$\hat{\beta}_2 = \frac{y_2 + 4\lambda p\beta_1 + 8\lambda p\beta_3 - 2\lambda p\beta_4}{1 + 10\lambda p}$$

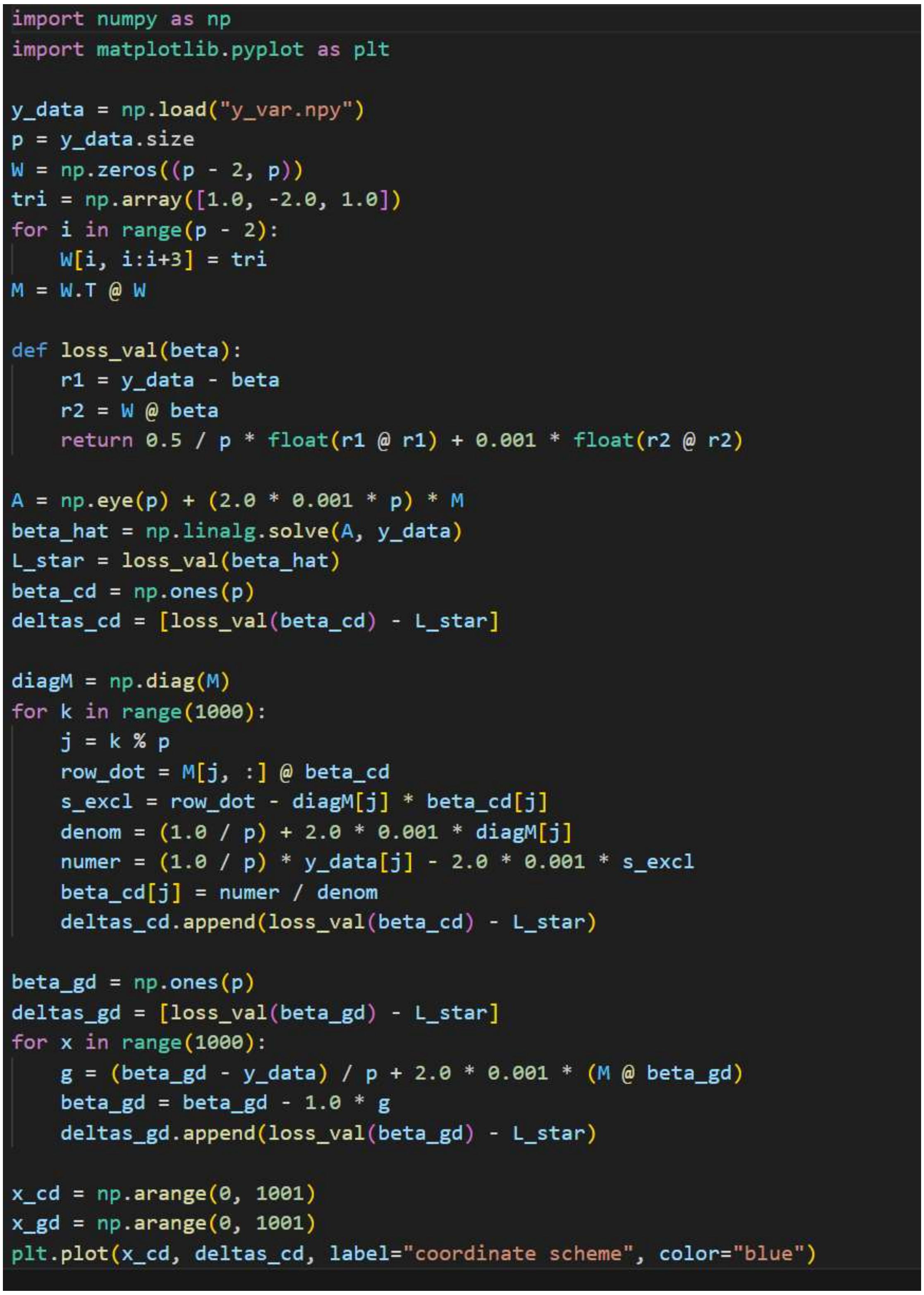
$$\hat{\beta}_{p-1} = \frac{y_{p-1} + 8\lambda\beta_{p-2} + 4\lambda p\beta_p - 2\lambda p\beta_{p-3}}{1 + 10\lambda p}$$

$$\hat{\beta}_p = \frac{y_p + 4\lambda p\beta_{p-1} - 2\lambda p\beta_{p-2}}{1 + 2\lambda p}$$

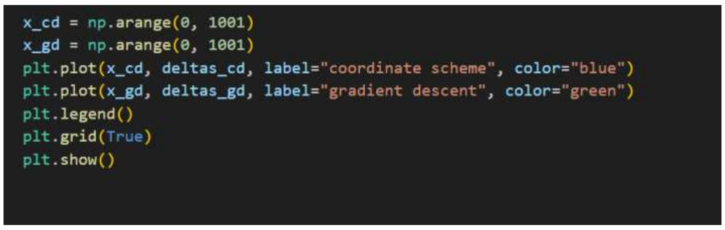
Part H:



20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0
200
400
600
800
1000
coordinate scheme
gradient descent



```python
import numpy as np
import matplotlib.pyplot as plt

y_data = np.load("y_var.npy")
p = y_data.size
W = np.zeros((p - 2, p))
tri = np.array([1.0, -2.0, 1.0])
for i in range(p - 2):
    W[i, i:i+3] = tri
M = W.T @ W

def loss_val(beta):
    r1 = y_data - beta
    r2 = W @ beta
    return 0.5 / p * float(r1 @ r1) + 0.001 * float(r2 @ r2)

A = np.eye(p) + (2.0 * 0.001 * p) * M
beta_hat = np.linalg.solve(A, y_data)
L_star = loss_val(beta_hat)
beta_cd = np.ones(p)
deltas_cd = [loss_val(beta_cd) - L_star]

diagM = np.diag(M)
for k in range(1000):
    j = k % p
    row_dot = M[j, :] @ beta_cd
    s_excl = row_dot - diagM[j] * beta_cd[j]
    denom = (1.0 / p) + 2.0 * 0.001 * diagM[j]
    numer = (1.0 / p) * y_data[j] - 2.0 * 0.001 * s_excl
    beta_cd[j] = numer / denom
    deltas_cd.append(loss_val(beta_cd) - L_star)

beta_gd = np.ones(p)
deltas_gd = [loss_val(beta_gd) - L_star]
for x in range(1000):
    g = (beta_gd - y_data) / p + 2.0 * 0.001 * (M @ beta_gd)
    beta_gd = beta_gd - 1.0 * g
    deltas_gd.append(loss_val(beta_gd) - L_star)

x_cd = np.arange(0, 1001)
x_gd = np.arange(0, 1001)
plt.plot(x_cd, deltas_cd, label="coordinate scheme", color="blue")
```



```python
x_cd = np.arange(0, 1001)
x_gd = np.arange(0, 1001)
plt.plot(x_cd, deltas_cd, label="coordinate scheme", color="blue")
plt.plot(x_gd, deltas_gd, label="gradient descent", color="green")
plt.legend()
plt.grid(True)
plt.show()
```

Part I:

$y \mid \beta \sim \mathcal{N}(\beta, \frac{1}{p} I_p)$

$\pi(\beta) \propto \exp(-\lambda \|W\beta\|_2^2)$

$p(y \mid \beta) \propto \exp(-\frac{1}{2}(y-\beta)^T (pI)(y-\beta)) = \exp(-\frac{p}{2}\|y-\beta\|_2^2)$

$\to \log p(\beta \mid y) = -\frac{p}{2}\|y-\beta\|_2^2 - \lambda \|W\beta\|_2^2 + \text{const}$

$J(\beta) = \frac{p}{2}\|y-\beta\|_2^2 + \lambda \|W\beta\|_2^2$

$\nabla_\beta J(\beta) = p(\beta - y) + 2\lambda W^T W \beta = 0$

$\therefore (pI + 2\lambda W^T W)\hat{\beta}_{map} = py$

$\hat{\beta}_{map} = (I + \frac{2\lambda}{p} W^T W)^{-1} y$

Compare with b part:

$L(\beta) = \frac{1}{2p}\|y-\beta\|_2^2 + \lambda_{det}\|W\beta\|_2^2$

$\hat{\beta} = (I + 2\lambda_{det}\, p W^T W)^{-1} y$

It has similar format. If $\lambda_{det} = \frac{\lambda}{p^2}$, they will be same