Computional Homework13 Report

Student: 纪浩正, jihz2023@mail.sustech.edu.cn

1 1D Storage of a Varying-Bandwidth Matrix

This section implements a compact one–dimensional storage scheme for symmetric matrices with varying bandwidth.

Implement: Encoding and Decoding

```
def decode(storge,dia):
    d=len(dia)
    M=np.zeros((d,d))
    M[0,0]=storge[0]
    for ii in range(d-1):
        n=dia[ii+1]-dia[ii]
        M[ii+1,ii+2-n:ii+2]=storge[dia[ii]:dia[ii+1]]
        M[ii+2-n:ii+2,ii+1]=storge[dia[ii]:dia[ii+1]]
    return M
```

```
def encode(M):
    d=len(M)
    storge=[]
    dia=[]
    for ii in range(d):
        idx=M[ii].nonzero()[0]
        storge.extend(M[ii,idx[0]:ii+1])
        dia.append(len(storge))
    return storge,dia
```

1.1 Test Matrices

We test the encoding/decoding procedure using the two matrices

$$\mathbf{M}_1 = \begin{pmatrix} 2 & 2 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 3 & 9 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 3 & 0 & 5 & 5 & 0 \\ 0 & 9 & 0 & 5 & 7 & 3 \\ 0 & 0 & 0 & 0 & 3 & 8 \end{pmatrix}, \qquad \mathbf{M}_2 = \begin{pmatrix} 2 & 2 & 0 & 3 & 0 & 0 \\ 2 & 1 & 4 & 0 & 0 & 0 \\ 0 & 4 & 1 & 0 & 9 & 0 \\ 3 & 0 & 0 & 5 & 5 & 0 \\ 0 & 0 & 9 & 5 & 7 & 3 \\ 0 & 0 & 0 & 0 & 3 & 8 \end{pmatrix}.$$

1.2 Result

Table 1 summarizes the encoded 1D storage array and the corresponding diagonal index for both test matrices.

Table 1 Encoding results for $\mathbf{M}_1$ and $\mathbf{M}_2$.

Matrix	1D Storage	Diagonal Index
$\mathbf{M}_1$	[2, 2, 1, 1, 3, 0, 5, 9, 0, 5, 7, 3, 8]	[1, 3, 4, 7, 11, 13]
$\mathbf{M}_2$	[2, 2, 1, 4, 1, 3, 0, 0, 5, 9, 5, 7, 3, 8]	[1, 3, 5, 9, 12, 14]

Figure 1 shows the terminal output after encoding and decoding each matrix.

```
PS C:\Users\20369\Documents\SUSTech\05Programming\Computati
aconda3/envs/py310/python.exe "c:/Users/20369/Documents/SUS
omerwork/homework13/StorageSparseMatrix.py"
Encode: Matrix 1
Storage with 1D=[2, 2, 1, 1, 3, 0, 5, 9, 0, 5, 7, 3, 8]
Diagonal index=[1, 3, 4, 7, 11, 13]
Test: Use storage and diagonal index to decode=
[[2. 2. 0. 0. 0. 0.]
 [2. 1. 0. 3. 9. 0.]
 [0. 0. 1. 0. 0. 0.]
 [0. 3. 0. 5. 5. 0.]
 [0. 9. 0. 5. 7. 3.]
 [0. 0. 0. 0. 3. 8.]]

Encode: Matrix 2
Storage with 1D=[2, 2, 1, 4, 1, 3, 0, 0, 5, 9, 5, 7, 3, 8]
Diagonal index=[1, 3, 5, 9, 12, 14]
Test: Use storage and diagonal index to decode=
[[2. 2. 0. 3. 0. 0.]
 [2. 1. 4. 0. 0. 0.]
 [0. 4. 1. 0. 9. 0.]
 [3. 0. 0. 5. 5. 0.]
 [0. 0. 9. 5. 7. 3.]
 [0. 0. 0. 0. 3. 8.]]
```

Figure 1 Encoding and decoding results for $\mathbf{M}_1$ and $\mathbf{M}_2$.

```
M1=np.array([
    [2,2,0,0,0,0],
    [2,1,0,3,9,0],
    [0,0,1,0,0,0],
    [0,3,0,5,5,0],
    [0,9,0,5,7,3],
    [0,0,0,0,3,8]])

M2=np.array([
    [2,2,0,3,0,0],
    [2,1,4,0,0,0],
    [0,4,1,0,9,0],
    [3,0,0,5,5,0],
    [0,0,9,5,7,3],
```

```
    [0,0,0,0,3,8]])

s1, d1 = encode(M1)
print(f"Encode: Matrix 1\nStorage with 1D={s1}")
print(f"Diagonal index={d1}")
m1 = decode(s1, d1)
print(f"Test: Use storage and diagonal index to decode=\n{m1}\n")

s2, d2 = encode(M2)
print(f"Encode: Matrix 2\nStorage with 1D={s2}")
print(f"Diagonal index={d2}")
m2 = decode(s2, d2)
print(f"Test: Use storage and diagonal index to decode=\n{m2}")
```

2 Solve the Linear System Using the Gauss–Seidel Iterative Formula

We consider the following linear system:

$$\begin{aligned} 2x_1 - x_2 &= 1, \\ -x_1 + 2x_2 - x_3 &= 0, \\ -x_2 + 2x_3 - x_4 &= 1, \\ -x_3 + 2x_4 &= 0. \end{aligned}$$

Rewriting each equation so that it fits the fixed-point iteration form, we obtain

$$\begin{aligned} x_1 &= 0\,x_1 + \tfrac{1}{2}\,x_2 + 0\,x_3 + 0\,x_4 + \tfrac{1}{2} \\ x_2 &= \tfrac{1}{2}\,x_1 + 0\,x_2 + \tfrac{1}{2}\,x_3 + 0\,x_4 + 0 \\ x_3 &= 0\,x_1 + \tfrac{1}{2}\,x_2 + 0\,x_3 + \tfrac{1}{2}\,x_4 + \tfrac{1}{2} \\ x_4 &= 0\,x_1 + 0\,x_2 + \tfrac{1}{2}\,x_3 + 0\,x_4 + 0 \end{aligned}$$

Using the fixed-point method, we update $x_1^k, x_2^k, x_3^k, x_4^k$ via

$$\begin{aligned} x_1^{k+1} &= 0\,x_1^k + \tfrac{1}{2}\,x_2^k + 0\,x_3^k + 0\,x_4^k + \tfrac{1}{2} \\ x_2^{k+1} &= \tfrac{1}{2}\,x_1^k + 0\,x_2^k + \tfrac{1}{2}\,x_3^k + 0\,x_4^k + 0 \\ x_3^{k+1} &= 0\,x_1^k + \tfrac{1}{2}\,x_2^k + 0\,x_3^k + \tfrac{1}{2}\,x_4^k + \tfrac{1}{2} \\ x_4^{k+1} &= 0\,x_1^k + 0\,x_2^k + \tfrac{1}{2}\,x_3^k + 0\,x_4^k + 0 \end{aligned}$$

We then accelerate the iteration using the Gauss–Seidel method.

In this method,

- x_1^{k+1} is updated using $(x_1^k, x_2^k, x_3^k, x_4^k)$,
- x_2^{k+1} is updated using $(x_1^{k+1}, x_2^k, x_3^k, x_4^k)$,
- x_3^{k+1} is updated using $(x_1^{k+1}, x_2^{k+1}, x_3^k, x_4^k)$,
- x_4^{k+1} is updated using $(x_1^{k+1}, x_2^{k+1}, x_3^{k+1}, x_4^k)$.

Thus the iteration becomes

$$\begin{aligned}
x_1^{k+1} &= 0\,x_1^k \quad + \tfrac{1}{2}\,x_2^k \quad + 0\,x_3^k \quad + 0\,x_4^k + \tfrac{1}{2} \\
x_2^{k+1} &= \tfrac{1}{2}\,x_1^{k+1} + 0\,x_2^k \quad + \tfrac{1}{2}\,x_3^k \quad + 0\,x_4^k + 0 \\
x_3^{k+1} &= 0\,x_1^{k+1} + \tfrac{1}{2}\,x_2^{k+1} + 0\,x_3^k \quad + \tfrac{1}{2}\,x_4^k + \tfrac{1}{2} \\
x_4^{k+1} &= 0\,x_1^{k+1} + 0\,x_2^{k+1} + \tfrac{1}{2}\,x_3^{k+1} + 0\,x_4^k + 0
\end{aligned}$$

The iteration can be written as:

$$x^{k+1} = B_1 x^{k+1} + B_2 x^k + g,$$

where

$$B_1 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 \end{bmatrix}, \quad B_2 = \begin{bmatrix} 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad g = \begin{bmatrix} \frac{1}{2} \\ 0 \\ \frac{1}{2} \\ 0 \end{bmatrix}.$$

Then

$$(I - B_1)x^{k+1} = B_2 x^k + g,$$

and the explicit iteration formula becomes:

$$x^{k+1} = (I - B_1)^{-1} B_2 x^k + (I - B_1)^{-1} g$$

$$x^{k+1} = Ax^k + b$$

We have

$$I - B_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -\frac{1}{2} & 1 & 0 & 0 \\ 0 & -\frac{1}{2} & 1 & 0 \\ 0 & 0 & -\frac{1}{2} & 1 \end{bmatrix},$$

and its inverse:

$$(I - B_1)^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ \frac{1}{2} & 1 & 0 & 0 \\ \frac{1}{4} & \frac{1}{2} & 1 & 0 \\ \frac{1}{8} & \frac{1}{4} & \frac{1}{2} & 1 \end{bmatrix}.$$

Thus,

$$A = (I - B_1)^{-1} B_2 = \begin{bmatrix} 0 & \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{4} & \frac{1}{2} & 0 \\ 0 & \frac{1}{8} & \frac{1}{4} & \frac{1}{2} \\ 0 & \frac{1}{16} & \frac{1}{8} & \frac{1}{4} \end{bmatrix},$$

```
A=np.array([
   [0,0.5,0,0],
   [0,1/4,1/2,0],
   [0,1/8,1/4,1/2],
   [0,1/16,1/8,1/4]
])
```

Similarly,

$$b = (I - B_1)^{-1} g = \begin{bmatrix} 1/2 \\ 1/4 \\ 5/8 \\ 5/16 \end{bmatrix},$$

which corresponds to:

```
b=np.array([
   1/2,1/4,5/8,5/16
])
```

Thus the complete iterative formula is:

$$x^{k+1} = Ax^k + b.$$

The exact solution of the system is:

$$x_{\text{exact}} = \left[\frac{6}{5}, \frac{7}{5}, \frac{8}{5}, \frac{4}{5}\right] = [1.2,\ 1.4,\ 1.6,\ 0.8].$$

We use an initial guess:

$$x^{(0)} = [0, 0, 0, 0].$$

The Python implementation computes the iteration and the L_2 error:

```
x=np.array([
   0,0,0,0
])
x_prec = np.array([1.2, 1.4, 1.6, 0.8], dtype=float)

for ii in range(35):
   x = A @ x + b

   l2_norm = np.linalg.norm(x - x_prec)

   print(
      f"Iter {ii:2d}: x1={x[0]:.6f}, x2={x[1]:.6f}, x3={x[2]:.6f}, x4={x[3]:.6f} "
      f"| L2 error={l2_norm:.6f}"
   )
```

```
Iter  0: x1=0.500000, x2=0.250000, x3=0.625000, x4=0.312500 | L2 error=1.732276
Iter  1: x1=0.625000, x2=0.625000, x3=0.968750, x4=0.484375 | L2 error=1.195552
Iter  2: x1=0.812500, x2=0.890625, x3=1.187500, x4=0.593750 | L2 error=0.788869
Iter  3: x1=0.945312, x2=1.066406, x3=1.330078, x4=0.665039 | L2 error=0.516936
Iter  4: x1=1.033203, x2=1.181641, x3=1.423340, x4=0.711670 | L2 error=0.338398
Iter  5: x1=1.090820, x2=1.257080, x3=1.484375, x4=0.742188 | L2 error=0.221490
Iter  6: x1=1.128540, x2=1.306458, x3=1.524323, x4=0.762161 | L2 error=0.144967
Iter  7: x1=1.153229, x2=1.338776, x3=1.550468, x4=0.775234 | L2 error=0.094883
Iter  8: x1=1.169388, x2=1.359928, x3=1.567581, x4=0.783791 | L2 error=0.062101
Iter  9: x1=1.179964, x2=1.373773, x3=1.578782, x4=0.789391 | L2 error=0.040646
Iter 10: x1=1.186886, x2=1.382834, x3=1.586112, x4=0.793056 | L2 error=0.026603
```

```
Iter 15: x1=1.198425, x2=1.397938, x3=1.598332, x4=0.799166 | L2 error=0.003195
Iter 20: x1=1.199634, x2=1.399451, x3=1.599565, x4=0.799783 | L2 error=0.000383
Iter 25: x1=1.199916, x2=1.399894, x3=1.599936, x4=0.799968 | L2 error=0.000046
Iter 30: x1=1.199983, x2=1.399966, x3=1.599983, x4=0.799992 | L2 error=0.000006
Iter 35: x1=1.199997, x2=1.399995, x3=1.599997, x4=0.799998 | L2 error=0.000001
```

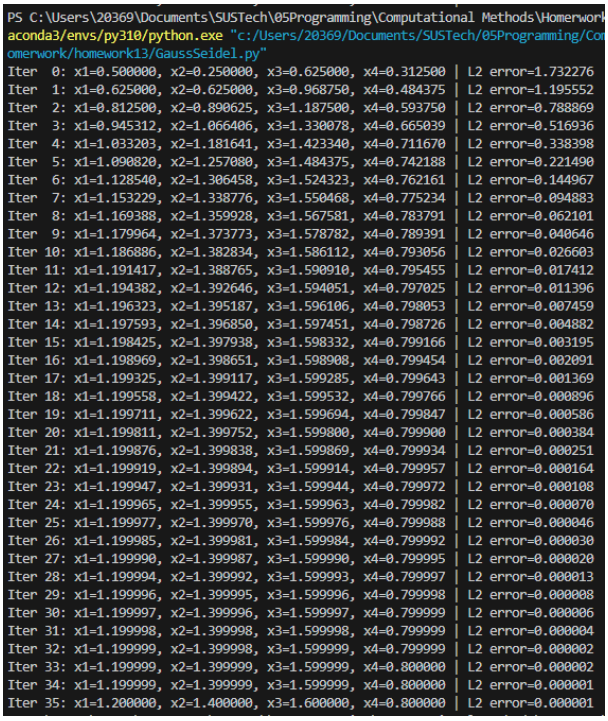

Figure 2　Computed values and L2 errors at selected Gauss–Seidel iteration steps