chapter 6 Direct Methods for Solving Linear Systems

Xu Feng

1 Gauss Elimination

- P357-4c

1.000000	0.500000	0.333333	0.250000	0.166667
0.000000	0.083333	0.083333	0.075000	0.059524
0.000000	0.083333	0.088889	0.083333	0.069444
0.000000	0.075000	0.083333	0.080357	0.069444

1.000000	0.500000	0.333333	0.250000	0.166667
0.000000	0.083333	0.088889	0.083333	0.069444
0.000000	0.000000	-0.005556	-0.008333	-0.009921
0.000000	0.000000	0.003333	0.005357	0.006944

1.000000	0.500000	0.333333	0.250000	0.166667
0.000000	0.083333	0.088889	0.083333	0.069444
0.000000	0.000000	-0.005556	-0.008333	-0.009921
0.000000	0.000000	0.000000	0.000357	0.000992

We get the solution $x = [\mathbf{-0.03174603, -0.03174603, -0.03174603, -0.03174603}]$.

- p357-4d

3.000000	1.000000	-4.000000	0.000000	5.000000	6.000000
0.000000	-0.333333	3.333333	-1.000000	-0.666667	0.000000
0.000000	-2.000000	-1.000000	1.000000	-1.000000	-5.000000
0.000000	0.333333	1.666667	1.000000	-6.333333	3.000000
0.000000	-1.333333	0.333333	-1.000000	-0.666667	1.000000

3.000000	1.000000	-4.000000	0.000000	5.000000	6.000000
0.000000	-2.000000	-1.000000	1.000000	-1.000000	-5.000000
0.000000	0.000000	3.500000	-1.166667	-0.500000	0.833333
0.000000	0.000000	1.500000	1.166667	-6.500000	2.166667
0.000000	0.000000	1.000000	-1.666667	0.000000	4.333333

3.000000	1.000000	-4.000000	0.000000	5.000000	6.000000
0.000000	-2.000000	-1.000000	1.000000	-1.000000	-5.000000
0.000000	0.000000	3.500000	-1.166667	-0.500000	0.833333
0.000000	0.000000	0.000000	1.666667	-6.285714	1.809524
0.000000	0.000000	0.000000	-1.333333	0.142857	4.095238

3.000000	1.000000	-4.000000	0.000000	5.000000	6.000000
0.000000	-2.000000	-1.000000	1.000000	-1.000000	-5.000000
0.000000	0.000000	3.500000	-1.166667	-0.500000	0.833333
0.000000	0.000000	0.000000	1.666667	-6.285714	1.809524
0.000000	0.000000	0.000000	0.000000	-4.885714	5.542857

We get the solution $x = [\mathbf{1.918129, 1.964912, -0.988304, -3.192982, -1.134503}]$.

2 Gauss with pivoting strategies

When we set the rounding as 3, the error could be huge, where Gauss method with pivoting strategies may be helpful.

	x1	**x2**	**x3**
ground truth	0.000000	10.000000	0.142857
gauss	0.000000	10.000000	0.142857
gauss with rounding=3	-0.001175	9.999706	0.142857
partial pivoting with rounding=3	-0.001175	9.999706	0.142857
scaled partial pivoting with rounding=3	-0.001175	9.999706	0.142857
error	-0.001175	-0.000294	0.000000

3 Matrix factorization

3.1 LU

[[2.1756,4.0231,-2.1732,5.1967],

[-4.0231,6,0,1.1973],

[-1,-5.2107,1.1111,0],

[6.0235,7,0,-4.1561]]

```
u [[ 2.1756       4.0231      -2.1732       5.1967     ]
 [ 0.          13.43948042 -4.01866194 10.80699101]
 [ 0.           0.          -0.89295239   5.09169403]
 [ 0.           0.           0.          12.03612803]]
l [[ 1.           0.           0.           0.         ]
 [-1.84919103   1.           0.           0.         ]
```

[-0.45964332 -0.25012194 1.   0.   ]
[2.76866152 -0.30794361 -5.35228302 1.   ]]

3.2 LDL

[[1.   0.   0.   0.   ]
[0.33333333 1.   0.   0.   ]
[0.16666667 0.2   1.   0.   ]
[-0.16666667 0.1   -0.24324324 1.   ]]
[6.   3.33333333 3.7   2.58108108]

3.3 LL

[[6,2,1,-1],

[2,4,1,0],

[1,1,4,-1],

[-1,0,-1,3]]

[[2.44948974 0. 0. 0.]
[0.81649658 2.   0.   0.   ]
[0.40824829 0.   2.   0.   ]
[-0.40824829 0.   0.   1.73205081]]
[[2.44948974 0.   0.   0.   ]
[0.81649658 1.82574186 0.   0.   ]
[0.40824829 0.36514837 1.92353841 0.   ]
[-0.40824829 0.18257419 -0.46788772 1.60657433]]

4 code

```
import numpy as np



def gauss(a, rounding = 32):
    n = a.shape[0]
    seq = np.arange(n)
    res = np.ones_like(seq).astype(np.float64)
    a = np.round(a, 3)

    for i in range(n-1):
        p = i
        while (a[p, i] == 0):
            p += 1
        if (p==n):
            return "Solution not unique!"
        if not (p==i):
            a[[p, i]]=a[[i, p]] # swap 2 rows
            seq[[p, i]]=seq[[i, p]]
```

```
        for j in range(i+1, n):
            a[j] -= a[j][i] / a[i][i] * a[i]

        a = np.round(a, 3)
        # print(a)


    if (a[n - 1][n - 1] == 0):
        return "Solution not unique!"

    res[n-1] = a[n - 1][n] / a[n - 1][n - 1]
    for i in range(n-1, -1, -1):
        tmp = a[i][n]
        for j in range(i+1, n):
            tmp -= a[i][j] * res[j]
        res[i] = tmp / a[i][i]

    # res2 = np.ones_like(res).astype(np.float64)
    # res2[seq] = res
    # print(res2)
    # print(res)
    return res

    # return resSorted


def gauss1(a, rounding = 32):
    n = a.shape[0]
    seq = np.arange(n)
    res = np.ones_like(seq).astype(np.float64)

    a = np.round(a, rounding)

    for i in range(n-1):

        p = np.argmax(abs(a[i:n-1,i]))+i
        if (p==n):
            return "Solution not unique!"
        if not (p==i):
            a[[p, i]]=a[[i, p]] # swap 2 rows
            seq[[p, i]]=seq[[i, p]]
        for j in range(i+1, n):
            a[j] -= a[j][i] / a[i][i] * a[i]

        a = np.round(a, rounding)
        # print(a)
```

```
    if (a[n - 1][n - 1] == 0):
        return "Solution not unique!"

    res[n-1] = a[n - 1][n] / a[n - 1][n - 1]


    for i in range(n-1, -1, -1):
        tmp = a[i][n]
        for j in range(i+1, n):
            tmp -= a[i][j] * res[j]
        res[i] = tmp / a[i][i]

    # print(res)
    # print(seq)
    # res2 = np.ones_like(res).astype(np.float64)
    # res2[seq] = res
    # print(res2)
    return res


def gauss2(a, rounding = 32):
    n = a.shape[0]
    seq = np.arange(n)
    res = np.ones_like(seq).astype(np.float64)

    a = np.round(a, rounding)

    for i in range(n-1):
        q = np.max(abs(a[i, i:n-1]))
        p = np.argmax(abs(a[i:n-1,i])/q)+i
        if (p==n):
            return "Solution not unique!"
        if not (p==i):
            a[[p, i]]=a[[i, p]] # swap 2 rows
            seq[[p, i]]=seq[[i, p]]
        for j in range(i+1, n):
            a[j] -= a[j][i] / a[i][i] * a[i]

        a = np.round(a, rounding)
        # print(a)

    if (a[n - 1][n - 1] == 0):
        return "Solution not unique!"

    res[n-1] = a[n - 1][n] / a[n - 1][n - 1]
```

```
    for i in range(n-1, -1, -1):
        tmp = a[i][n]
        for j in range(i+1, n):
            tmp -= a[i][j] * res[j]
        res[i] = tmp / a[i][i]

    # print(res)
    # print(seq)
    # res2 = np.ones_like(res).astype(np.float64)
    # res2[seq] = res
    # print(res2)
    return res

if __name__ == '__main__':
    # a = np.array([[1,1,0,3,4],
    #               [2,1,-1,1,1],
    #                [3,-1,-1,2,-3],
    #                [-1,2,3,-1,4]]).astype(np.float64)
    # print(gauss1(a))
    #
    # print(gauss(a))
    # print(gauss2(a))


    # p3574c = np.array([[1,1/2,1/3,1/4,1/6],
    #                    [1 / 2, 1 / 3, 1 / 4, 1/5,1/7],
    #                    [1 / 3, 1 / 4, 1/5, 1 / 6, 1/8],
    #                    [ 1 / 4, 1/5, 1 / 6,1/7,1/9]]).astype(np.float64)
    # res = gauss1(p3574c)
    # print(res)
    # print(np.matmul(p3574c[:,0:4],res))
    # print(p3574c[:,4])
    #
    # p3574d = np.array([[2,1,-1,1,-3,7],
    #                     [1,0,2,-1,1,2],
    #                     [0,-2,-1,1,-1,-5],
    #                     [3,1,-4,0,5,6],
    #                     [1,-1,-1,-1,1,3]]).astype(np.float64)
    # res = gauss1(p3574d)
    # print(res)
    # print(np.matmul(p3574d[:,0:5],res))
    # print(p3574d[:,5])


    a = np.array( [[3.03,-12.1,14,-119],
                  [-3.03,12.1,-7,120],
                  [6.11,-14.2,21,-139]]).astype(np.float64)
```

```python
res = gauss(a)
print(res)

print(gauss(a, 3))
# res2 = gauss1(a, 3)
# print(gauss1(a, 3))
# print(gauss2(a,3))
```