

Assignment 4 of Computational Astrophysics in NTHU

Wei-Hsiang Yu 游惟翔

March 31, 2021

1 Written Assignments

Q1 : Vector Norms.

For the vector $x = [-1.6, 1.2]^T$ From the lecture we know that:

$$\text{First norm : } \|x\|_1 = \sum_{i=1}^n |x_i| \quad (\text{sum of each element})=2.8$$

$$\text{Second norm : } \|x\|_2 = \left(\sum_{i=1}^n |x_i|^2\right)^{\frac{1}{2}} \quad (\text{square root the sum of each element does square})=2$$

$$\text{Infinity norm : } \|x\|_\infty = \max_{1 \leq i \leq n} |x_i| \quad (\text{maximum element})=1.6$$

Q2 : Matrix Norms.

For the matrix A

$$A = \begin{bmatrix} 7 & -3 & 2 \\ 1 & 1 & 5 \\ 2 & -2 & 1 \end{bmatrix}$$

From the lecture we know that:

$$\text{First norm : } \|A\|_1 = \max_j \sum_{i=1}^n |a_{ij}| \quad (\text{the maximum absolute \textbf{column} sum})=10$$

$$\text{Infinity norm : } \|A\|_\infty = \max_i \sum_{j=1}^n |a_{ij}| \quad (\text{the maximum absolute \textbf{row} sum})=12$$

Q3 : Matrix calculation.

$$Ax = \begin{bmatrix} 1 & 2 & 2 \\ 4 & 6 & 8 \\ 4 & 8 & 10 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 4 \\ 6 \\ 10 \end{bmatrix} = b$$

3a. Gaussian elimination

Eliminate matrix A, we can use $4 \times \text{first row}$ to subtract the *third row*(step1), and also subtract the *second row* by $4 \times \text{first row}$ (step2).

$$\begin{bmatrix} 1 & 2 & 2 \\ 4 & 6 & 8 \\ 4 & 8 & 10 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 2 & 2 \\ 4 & 6 & 8 \\ 0 & 0 & 2 \end{bmatrix}_{\text{step1}} \Rightarrow \begin{bmatrix} 1 & 2 & 2 \\ 0 & -2 & 0 \\ 0 & 0 & 2 \end{bmatrix}_{\text{step2}} = U$$

Finally, we can solve this equation.

$$\begin{bmatrix} 1 & 2 & 2 \\ 0 & -2 & 0 \\ 0 & 0 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 4 \\ 6 \\ 10 \end{bmatrix} \Rightarrow \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ -3 \\ 5 \end{bmatrix}$$

And we get $\det(A) = \epsilon^2$ $\det(A^{-1}) = 1$, so the condition number of this system is ϵ^2 . For my computer, the value of ϵ can't show when it's equal to $1e-12$.

We are interested in $\sqrt{\epsilon_{mach}}$, I choose $1e-6$ be the value of ϵ , and get the relative error of solution x in component 1 and $\epsilon(\text{Fig.1(c)})$.

We know:

$$\frac{\|\Delta x\|}{\|x\|} \leq \text{cond}(A)\epsilon_{mach} \quad (2)$$

but here the relative error about ϵ is not obey Eq.2. I guess the max ϵ_{mach} is not equal value $1e-12$ (maybe $5e-11...$), so my computer can't get good result.

Q2 : Cholesky factorization.

In Fig.2(a)., the pseudo code supplied by the lecture is translated into the fortran code. Noticed that the output variable L don't need to be assigned at the second do loop, because it doesn't finish the process yet when program go to the second loop, the first loop or the moment when we calculate k,k position on the matrix will then get the output result of Cholesky factorization.

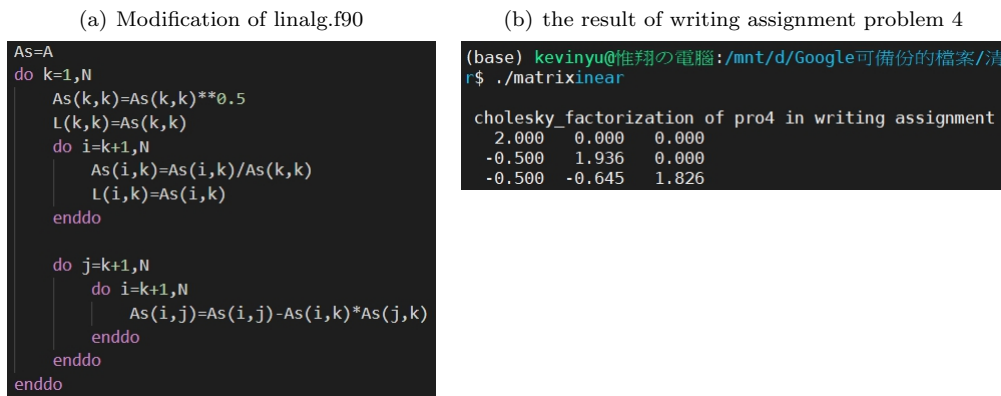


Figure 2: Cholesky factorization exercise

The test of my program is showed in Fig.2(b). The output is the same as the result we get in the written assignments problem 4.

Q3 : Banded matrix.

The package of *scipy.linalg.solve_banded* need to change the banded matrix into the diagonal banded form. The following is the example of transforming a 7×7 banded matrix into the diagonal banded form:

$$\begin{bmatrix} 9 & -4 & 1 & 0 & 0 & 0 & 0 \\ -4 & 6 & -4 & 1 & 0 & 0 & 0 \\ 1 & -4 & 6 & -4 & 1 & 0 & 0 \\ 0 & 1 & -4 & 6 & -4 & 1 & 0 \\ 0 & 0 & 1 & -4 & 6 & -4 & 1 \\ 0 & 0 & 0 & 1 & -4 & 5 & -2 \\ 0 & 0 & 0 & 0 & 1 & -2 & 1 \end{bmatrix} \Rightarrow \begin{bmatrix} * & * & 1 & 1 & 1 & 1 \\ * & -4 & -4 & -4 & -4 & -2 \\ 9 & 6 & 6 & 6 & 5 & 1 \\ -4 & -4 & -4 & -4 & -2 & * \\ 1 & 1 & 1 & 1 & * & * \end{bmatrix}$$

In this example, there are 2 nonzero diagonal below and above the main diagonal, so we can set the variables l & u which package need to 2 (Fig.3(a). & Fig.3(b).).

And I also decide two methods to assign the diagonal banded form matrix. Fig.3(a). judge loop number every time to decide whether the variable is need to be assigned. Fig.3(b). will continue to execute next loop without doing another if-else judgement.

The time consumption of those three method is showed in Fig.4.

Q4 : UL factorization.

Verify UL factorization

In Fig.5., we first given a matrix R , and transpose it into matrix R^T , then we inner product these two matrix ,and judge whether it is equal to matrix A in writing assignment problem 4.

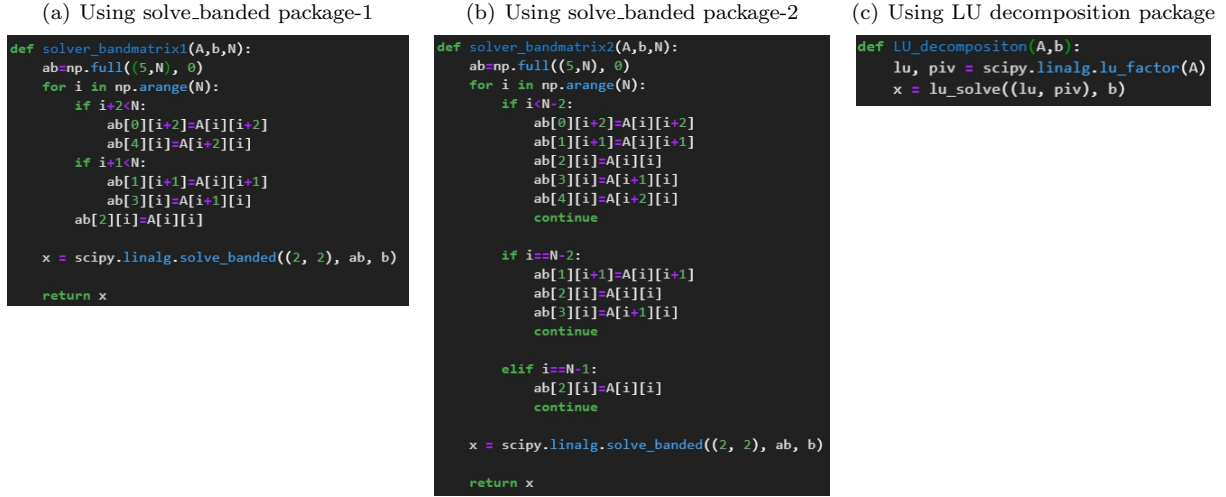


Figure 3: The three method to solve Banded matrix

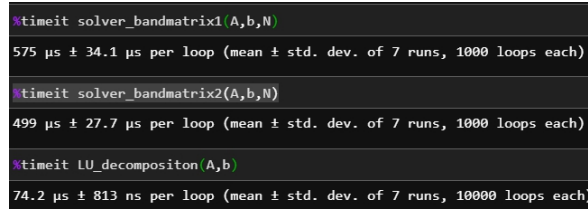


Figure 4: Time spend by different package in python.

Solving the system

In this part, I use the method in problem3.(solver.bandmatrix2 Fig.2(b).) be the solution of using banded system solver.

And add the comparison: Cholesky factorization was applied by the package in scipy called *linalg.cho_factor*

The algorithm of my code is used by the code in fortran to transform, but noticed that: **the UL factorization has different order to solve compared to LU factorization!**

$$\begin{aligned}
 Ax &= RR^T x = ULx = b \\
 Uy &= b \\
 Lx &= y
 \end{aligned}
 \tag{3}$$

The error I derive by the following steps:

- 1.Inner product the matrix A with the solution x to get the experiment b.
- 2.Compare to expected b, get the gap between expected and experiment.
- 3.Finally, sum those error to get the displacement of the calculative error.

And we can see that the error generates by UL factorization has even no error.(Fig.6) but spend much more time(Fig.7.) The option of using package or self-coding seem a kind of trade-off.

Condition number of matrix A

I use the instruction of numpy called *linalg.cond* to find out the condition number. The output is 1.29e12, it means that this matrix can get pretty accurate solution.(Since larger value of condition number represents nearly singular.)

```

R=matrixR(N)

RT=R.transpose()

A4=R.dot(RT)
print(np.array_equal(A4,A))

True

```

Figure 5: Verify UL factorization.

```

X1=UL_factorization(R,RT,b,N)
b=np.full(N, 1)
X2=cho_factorization(A,b)
b=np.full(N, 1)
X3=solver_bandmatrix2(A,b,N)

acc1=np.dot(A, X1)
acc2=np.dot(A, X2)
acc3=np.dot(A, X3)

error1=np.abs(acc1-1)
error2=np.abs(acc2-1)
error3=np.abs(acc3-1)

accurate1=error1.sum()
accurate2=error2.sum()
accurate3=error3.sum()

print ("The error of using UL_factorization: %f" %accurate1)
print ("The error of using cho_factorization: %f" %accurate2)
print ("The error of using solver_bandmatrix2: %f" %accurate3)

The error of using UL_factorization: 0.000000
The error of using cho_factorization: 0.019947
The error of using solver_bandmatrix2: 0.015274

```

Figure 6: The accuracy of doing UL factorization by different package.

```

%timeit UL_factorization(R,RT,b,N)

/home/kevinyu/miniconda3/lib/python3.7/site-packages/ipykernel_launcher
r.py:18: RuntimeWarning: overflow encountered in long_scalars
/home/kevinyu/miniconda3/lib/python3.7/site-packages/ipykernel_launcher
r.py:10: RuntimeWarning: overflow encountered in long_scalars
# Remove the CMD from sys.path while we load stuff.
750 ms ± 14.9 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)

%timeit cho_factorization(A,b)

9.93 ms ± 112 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)

%timeit solver_bandmatrix2(A,b,N)

4.65 ms ± 200 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)

```

Figure 7: The time consumption of each algorithm.

```

np.linalg.cond(A)

1295526023618.2551

```

Figure 8: condition number of matrix A.