Homework 2

Dingcheng Yue

Problem 1: (10 points)

> (a) What is the LU factorization of the following matrix?
>
> $$\mathrm{A} = \begin{bmatrix} 1 & a \\ b & c \end{bmatrix}$$
>
> (b) Given the LU factorization of A, under what condition is the matrix singular?

(a)

$$\mathrm{A} = \mathrm{LU} = \begin{bmatrix} 1 & 0 \\ -b & 1 \end{bmatrix} \begin{bmatrix} 1 & a \\ 0 & c - ab \end{bmatrix}$$

(b) The matrix is singular if there is a pivot is zero, thus when $c - ab = 0$, the matrix is singluar.

Problem 2:(10 points)

Show that the Woodbury formula

$$(A - UV^T)^{-1} = A^{-1} + A^{-1}U(I - V^T A^{-1} U)^{-1} V^T A^{-1}$$

given in Section 2.4.9 is correct. (*Hint: Multiply both sides by* $(A - UV^T)$.)

$$\begin{aligned}
&(A - UV^T)RHS \\
&= I - UV^T A^{-1} + U(I - A^T A^{-1} U)^{-1} V^T A^{-1} - UVA^{-1}(I - A^T A^{-1} U)^{-1} V^T A^{-1} \\
&= I - U(-I + (I - A^T A^{-1} U)^{-1} V^T A^{-1} - UV^T A^{-1}(I - A^T A^{-1} U)^{-1}) V^T A^{-1} \\
&= I - U(-I + (I - A^T A^{-1} U)(I - A^T A^{-1} U)^{-1}) V^T A^{-1} \\
&= I - U(-I + I) V^T A^{-1} \\
&= I
\end{aligned}$$

$(A - UV^T)LHS = I$

Thus, $(A - UV^T)^{-1} = A^{-1} + A^{-1}U(I - V^TA^{-1}U)^{-1}V^TA^{-1}$

Problem3:(10 points)

Let A be a symmetric positive definite matrix. Show that the function

$$||\vec{x}||_A := (\vec{x}^TA\vec{x})^{\frac{1}{2}}$$

satisfies the three properties of a vector norm given near the end of Section 2.3.1. This vector norm is said to be induced by the matrix A. (It is often referred to as the A-norm of $\vec{x}$.)

- Since A is symmetric positive definite matrix, if $\vec{x} \neq \vec{0}$, $(\vec{x}^TA\vec{x}) > 0$, thus, $(\vec{x}^TA\vec{x})^{\frac{1}{2}} > 0$
- $||\gamma x|| = ((\gamma x)^TA(\gamma x))^{\frac{1}{2}} = (\gamma^2\vec{x}^TA\vec{x})^{\frac{1}{2}} = \gamma||x||$
- According to Cauchy–Schwarz inequality

$$\begin{aligned}
x^TAy = y^TAx &= \langle x, y\rangle_A \\
||x||_A^2 + ||y||_A^2 + 2\langle x, y\rangle_A &= ||x+y||_A^2 \\
\langle x, y\rangle_A &\leq ||x||_A||y||_A \\
||x+y||_A^2 &\leq 2||x||_A||y||_A + ||x||_A^2 + ||y||_A^2 \\
||x+y||_A^2 &\leq (||x||_A + ||y||_A)^2
\end{aligned}$$

Problem 4: (5 points)

For the matrix norm of your choice, find a 2 × 2 matrix A that demonstrates $||A^{-1}|| \neq ||A||^{-1}$

Let us set

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

Thus, $||A^{-1}||_1 = \frac{7}{2}$ and $||A||_1^{-1} = \frac{1}{7}$

Problem 5: (5 points)

Suppose that B is nonsingular. Show that $A := B^T B$ is symmetric positive definite.

$z^T Az = z^T B^T Bz = (Bz)^T (Bz) = ||Bz||_2^2 > 0$ if $z \neq 0$

Problem 6: (10 points)

Suppose that the symmetric matrix

$$B = \begin{bmatrix} A & \vec{a} \\ \vec{a}^T & \alpha \end{bmatrix}$$

is Positive definite

(a) Show that $\alpha > 0$. (Hint: Find a vector $\vec{x}$ of length $n+1$ that isolates the effect of α when computing $||\vec{x}||A$.

We denote $z = \begin{bmatrix} x \\ \beta \end{bmatrix}$, so that

$$z^T Bz = \begin{bmatrix} x^T & \beta \end{bmatrix} \begin{bmatrix} A & a \\ a^T & \alpha \end{bmatrix} \begin{bmatrix} x \\ \beta \end{bmatrix}$$

Thus $z^T Bz = x^T Ax + \beta(a^T x + x^T a) + \alpha\beta^2$
Since B is positive definite, $z^T Bz > 0$ for $Bz \neq 0$, if $x = \vec{0}$, $\alpha\beta^2 > 0$, Thus $\alpha > 0$

(b) Since B is positive definite, $z^T Bz > 0$ for $Bz \neq 0$, if $\beta = 0$, $x^T Ax > 0$, Thus, A is positive definite.

Problem 7: (30 points)

In this problem you'll solve a linear system using the **LU** factorization of a given matrix that has
been modified by a rank-one update after the factorization.
1. Write a function that computes the **LU** factorization of a matrix. Your function should return two matrices **L** and **U**.
2. Write a function that solves an upper triangular system using back-substitution and

another function that solves a lower triangular system using forward substitution. Your functions should receive as inputs a matrix A and a vector b and return as output the vector *x*.

3. Write a function that takes as inputs **L**, **U**, and b and solves the linear system $Ax = LUx = b$. You must use your back and forward substitution functions in this function.

4. Solve the system **Ax = b** with:

$$A = \begin{bmatrix} 2 & 4 & -2 \\ 4 & 9 & -3 \\ -2 & -1 & 7 \end{bmatrix} b = \begin{bmatrix} 2 \\ 8 \\ 7 \end{bmatrix}$$

5. Now suppose that the matrix **A** in the previous bullet changes so that $a_{1,2} = 2$. Use the Sherman-Morrison updating technique to compute the new solution x without refactoring the matrix, using the original right-hand-side vector **b**.

6. Finally, solve the system again with a factorization of the updated matrix and the original right-hand-side vector **b**. Are the solutions the same using the Sherman-Morrison formula and the factorization of the updated matrix? What is the advantage or disadvantage of using
Sherman-Morrison?

The code for LU factorisation

```
import numpy as np

def LU(M):
    size = M.shape[0]
    L = np.eye(size)
    U = np.copy(M)
    for i in range(size-1):
        # eliminating
        coeff = (U[i+1:,i]/U[i,i]).reshape(size-i-1,1)
        U[i+1:,i:] -= coeff.dot(U[i,i:].reshape((1,size-i)))
        L[i+1:,i] = coeff.reshape((size-i-1,))
    return L, U
```

The code for substitution: `forward_sub` for forward substitution and `backward_sub` for backward substitution

```
def backward_sub(U, b):
    size = U.shape[0]
    x = np.zeros(size)
```

```
    for i in range(size):
        k = size-i-1
        x[k] = (b[k]-U[k,k+1:].dot(x[k+1:]))/U[k,k]
    return x

def forward_sub(L, b):
    size = L.shape[0]
    x = np.zeros(size)
    for i in range(size):
        x[i] = (b[i]-L[i,:i].dot(x[:i]))/L[i,i]
    return x
```

The code for solve function, either use `solve` for Ax=b or `solveLU` for LUx=b

```
def solve(A, b):
    L, U = LU(A)
    Ux = forward_sub(L, b)
    x = backward_sub(U, Ux)
    return x

def solveLU(L, U, b):
    Ux = forward_sub(L, b)
    x = backward_sub(U, Ux)
    return x

def solveUpdate(L, U, x, u, v):
    z = solveLU(L, U, u)
    y = x+v.dot(x)/(1-v.dot(z))*z
    return y
```

The code for solves for the system and its verification

```
A = np.array([[2, 4, -2], [4, 9, -3], [-2, -1, 7]],
dtype=np.float)
b = np.array([2, 8, 10], dtype=np.float)
L, U = LU(A)
x = solveLU(L, U, b)
print x # it is python2 and the reason to factorise is for the
next question
print np.linalg.solve(A, b)
```

The code for solves updated system and in this case $u = [1, 0, 0]$ and $v = [0, 2, 0]$ for updating

```
u = np.array([1, 0, 0], dtype=np.float)
v = np.array([0, 2, 0], dtype=np.float)
y = solveUpdate(L, U, x, u, v)
A_prim = np.array([[2, 2, -2], [4, 9, -3], [-2, -1, 7]],
dtype=np.float)
print y
print np.linalg.solve(A_prim, b)
```

Advantage: The algorithm is way faster for small change $O(n^2)$ in this case comparing to $O(n^3)$ which resolves the equation.
Disadvantage: More steps and might leads to inaccuracy in some cases.

Problem 8

An $n \times n$ Hilbert matrix H has entries $h_{ij} = 1/(i+j-1)$,so it has the form:

$$\begin{bmatrix} 1 & 1/2 & 1/3 & \cdots \\ 1/2 & 1/3 & 1/4 & \cdots \\ 1/3 & 1/4 & 1/5 & \cdots \\ \dots & \dots & \dots & \dots \end{bmatrix}$$

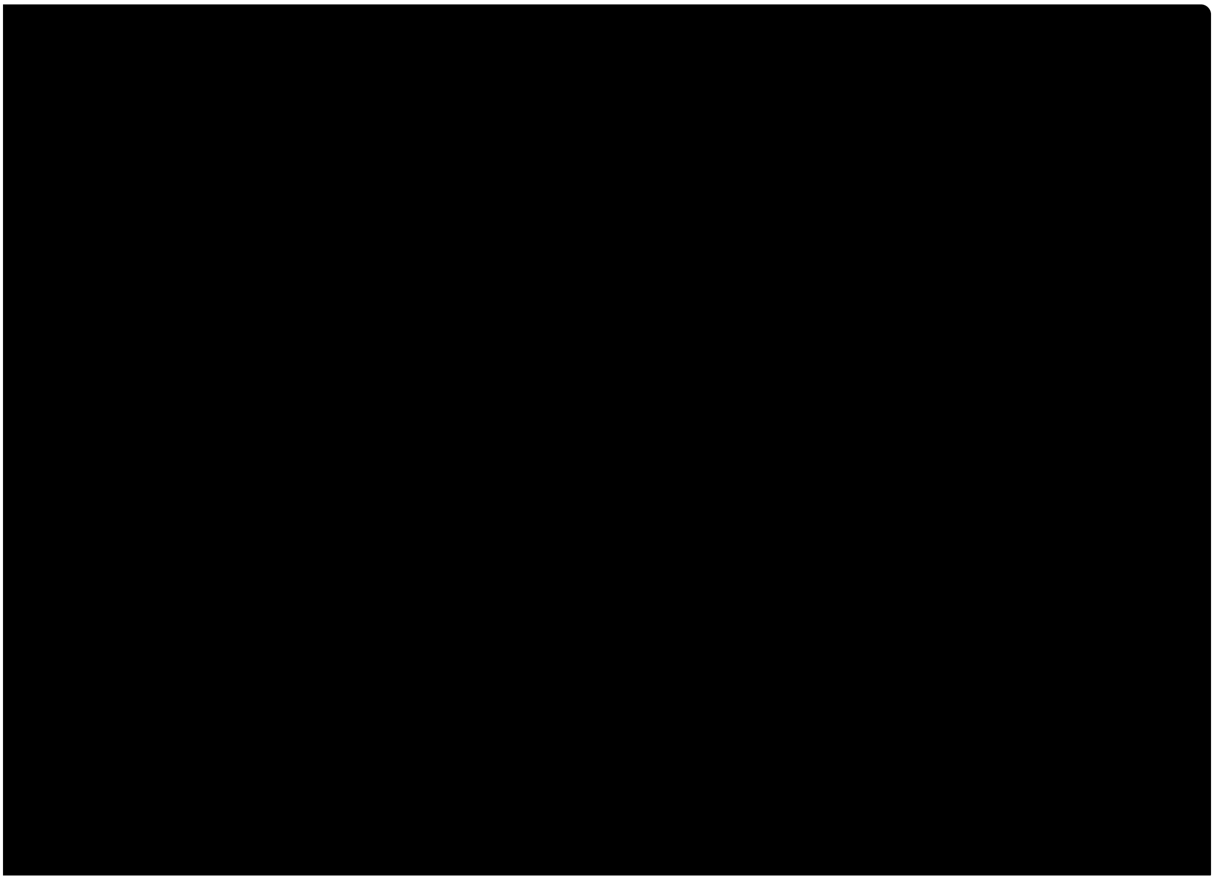
For n = 2, 3, ... , generate the Hilbert Matrix of order n, and also generate the n-vector b = Hx, where x is the n-vector with all of its components equal to 1. Use a library routine for Gaussian elimination (or Cholesky factorization, since the Hilbert Matrix is symmetric and positive definite) to solve the resulting linear system $Hx = b$, obtaining an approximate solution $\hat{x}$. Compute the ∞-norm of the residual $r = b - H\hat{x}$ and of the error $\Delta x = x - \hat{x}$ where x is the vector of all ones. How large can you take n before the error is 100 percent (i.e. there are no significant digits in the solution)? Also use a condition estimator to obtain **cond (H)** for each value of n. Try to characterize the condition number as a function of n.

When n equals 13, all the significant bits are all lost.

```
import numpy as np
def H(n):
    return 1.0/np.array([[i+j+1 for i in range(n)] for j in
range(n)])

def x(n):
```

```
        return np.ones((n,1))

def b(n):
    return H(n).dot(x(n))

def x_hat(n):
    return np.linalg.solve(H(n), b(n))

def r(n):
    return b(n)-H(n).dot(x_hat(n))

def error(n):
    return x(n)-x_hat(n)

def norm(v):
    return np.max(np.abs(v))

def k(n):
    return str(norm(error(n)))+" "+str(norm(r(n)))+" "+str(n)

print "\n".join(map(k, range(1,20)))
```

result:



The code below shows the growth of conditional number

```
def cond(n):
    return np.linalg.cond(H(n), np.inf)

map(lambda n: cond(n+1)/cond(n), range(1, 10))
```

result:

```
[27.000000000000011,
27.703703703703795,
37.934491978605998,
33.256599118943456,
30.806012999379739,
33.890107818373167,
34.381817418000253,
32.464139065511134,
32.149950139241625]
```

So, my guess is the conditional number is roughly $cond(H_n) \approx 30^n$