HW_2_code

February 27, 2022

[1]:
```python
import numpy as np
```

0.1 Question 3b

[2]:
```python
def choleskyfact(A):
    """
    Input: A: n*n matrix, 2d array
    Output: R n*n matrix, 2d array
    """

    # Check if the input is valid

    if A.shape[0] != A.shape[1]:
        raise ValueError("Input should be square.")

    if not np.allclose(A, A.T):
        raise ValueError("Input should be symmetric.")


    # Initialize the output R matrix

    n = A.shape[0]
    R = np.zeros((n, n))

    # Implement the iteration accordingly as the given instruction

    for j in range(n):
        temp = A[j][j] - sum([ (R[j][k])**2 for k in range(0, j) ])
        if temp <= 0:
            raise ValueError("The quantity under the square root is negative")

        R[j][j] = temp**(1/2)

        for i in range(j+1, n):
            R[i][j] = (1/R[j][j])*(A[i][j] - sum([ R[i][k]*R[j][k] for k in␣
↪range(0, j)]))
```

```python
    return R
```

[3]:
```python
# Apply the code to the matrix given

A = np.array([
    [2, 1, 1/2, 1/4],
    [1, 4, 1, 1/2],
    [1/2, 1, 4, 1],
    [1/4, 1/2, 1, 2]
])

print("The R solved by the self-defined function is: \n", choleskyfact(A))
print("\nCompared with the solution generated by numpy method: \n", np.linalg.
 ↪cholesky(A))
```

```
The R solved by the self-defined function is:
 [[1.41421356 0.         0.         0.        ]
 [0.70710678 1.87082869 0.         0.        ]
 [0.35355339 0.40089186 1.92724822 0.        ]
 [0.1767767  0.20044593 0.44474959 1.31558703]]

Compared with the solution generated by numpy method:
 [[1.41421356 0.         0.         0.        ]
 [0.70710678 1.87082869 0.         0.        ]
 [0.35355339 0.40089186 1.92724822 0.        ]
 [0.1767767  0.20044593 0.44474959 1.31558703]]
```

0.2 Question 4

[4]:
```python
def backward(U, b):
    """
    Input: U, n*n upper matrix, 2Darray, b: n*1 array
    Output: x, n*1 array
    """
    b = b.squeeze() # in case input is an column vactor

    # Check if the input is valid

    if U.shape[0] != U.shape[1]:
        raise ValueError("U should be square.")

    if not np.allclose(U, np.triu(U)):
        raise ValueError("U should be upper triangular.")

    if b.shape[0] != U.shape[0]:
```

```
        raise ValueError("Input dimension not compatible.")

    n = U.shape[0]

    # Initialize the output array
    x = np.zeros(n)

    # Implement the iteration according to the instructions given
    for i in range(1, n+1):
        x[-i] = (1 / U[-i][-i])*(b[-i] - sum([U[-i][-j]*x[-j] for j in range(1,␣
↪i)]))

    return x
```

[5]:
```
# Apply the program on the matrix given.

U = np.array([
    [1, 2, 6, -1],
    [0, 3, 1, 0],
    [0, 0, 4, -1],
    [0, 0, 0, 2],
])

b = np.array([-1, -3, -2, 4])

print("The result is: \n", backward(U, b))
print("\nCompare with the result from taking inverse: \n", np.linalg.inv(U)@b.T)
```

```
The result is:
 [ 3. -1.  0.  2.]

Compare with the result from taking inverse:
 [ 3. -1.  0.  2.]
```