ECSE 543: Numerical Methods

Assignment 1

Wenjie Wei
260685967

October 3, 2018

Contents

Introduction

All programs in this assignment are written and compiled with Python 3.6. This report is structured so that the individual problems are answered in respective sections. The python codes used to solve the assignment problems are attached in the appendices, with the file names labeled at the top of the code segments.

Choleski Decomposition

Choleski Implementation

The implementation of Choleski decomposition is shown in Listing 2. There are two methods defined in `choleski.py`: `check_choleski(A, b, x)` and `choleski_decomposition(A, b)`. The latter method takes two matrices `A` and `b` as arguments, and returns `x` as the computational result of the decomposition. The first method takes these three matrices as arguments, and performs matrix production to check the result of

$$Ax = b$$

The precision of the equality is set to 0.001, as the program may end up with results with uncertainties with a quantity level of 10^{-8}.

Simple Tester Matrices

To examine the functionality of the implementation, some tester matrices are constructed. The first tester matrix has randomly chosen entries, under the condition that the matrix is a non-singular, symmetric, positive definite matrix:

$$\begin{bmatrix} 15 & -5 & 0 & -5 \\ -5 & 12 & -2 & 0 \\ 0 & -2 & 6 & -2 \\ -5 & 0 & -2 & 9 \end{bmatrix} x = \begin{bmatrix} 115 \\ 22 \\ -51 \\ 13 \end{bmatrix}$$

To ensure non-singularity and positiveness, the entries on the primary diagonal must be chosen to be positive, otherwise the program with raise errors, meaning that the matrix does not meet the requirement. If the Choleski Decomposition succeeds, the matrix is proven to be positive definite.

Figure 1 shows the result of the test of this certain tester matrix. This result is found to be correct by checking the dot product (which is implemented in file `matrix.py`) of matrix A and vector x. This result is also verified by MATLAB using the back slash operator.

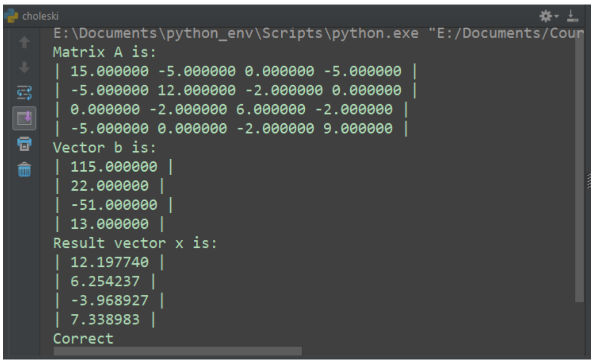

Figure 1: Result of the First Choleski Decomposition Test

Code Listings

Listing 1: Custom matrix package (`matrix.py`).

```python
import math


class matrix(object):
    def __init__(self, vec, rows, cols):
        self._vec = vec
        self._rows = rows
        self._cols = cols

    def is_square(self):
        return self._rows == self._cols

    def is_symmetric(self):
        if not self.is_square():
            return False

        else:
            for i in range(self.rows):
                for j in range(self.cols):
                    if self[i][j] != self.T[i][j]:
                        return False

        return True

    def transpose(self):
        vec_trans = [[None for _ in range(self.rows)] for _ in range(self.cols)]
        for x in range(self.cols):
            for y in range(self.rows):
                vec_trans[x][y] = self.vec[y][x]

        transposed_matrix = matrix(vec_trans, self.cols, self.rows)
        return transposed_matrix

    def dot_product(self, other):
        if self.cols != other.rows:
            raise ValueError("Incorrect dimension for vector multiplication.")

        result_vec = [[None for _ in range(other.cols)] for _ in range(self.rows)]
        result = matrix(result_vec, self.rows, other.cols)

        for i in range(self.rows):
            for j in range(other.cols):
                temp_sum = 0
                for k in range(other.rows):
                    temp_sum += self[i][k] * other[k][j]
                result[i][j] = temp_sum

        return result

    def __getitem__(self, item_number):
        if isinstance(item_number, int):
            return self._vec[item_number]

        if isinstance(item_number, tuple):
            x, y = item_number
            # use some "dummy entries" as a buffer to decrease the possibility of occurring out of boundary.
            if x < 0 or x >= self.rows or y < 0 or y >= self.cols:
                return 0
            else:
                return self._vec[x][y]

    def clone(self):
        cloned_matrix = matrix(self.vec, self.rows, self.cols)
        return cloned_matrix
```

```python

    def print_matrix(self):
        for i in range(self.rows):
            print("|", end=" ")
            for j in range(self.cols):
                print("%f" % self[i][j], end=" ")
            print("|")

    @property
    def vec(self):
        return self._vec

    @property
    def rows(self):
        return self._rows

    @property
    def cols(self):
        return self._cols

    @property
    def T(self):
        return self.transpose()
```

Listing 2: Choleski decomposition (`choleski.py`).

```python
import math
from matrix import matrix


def check_choleski(A, b, x):
    """
    This method checks if the result of the choleski decomposition is correct.
    Precision is set to 0.001.

    :param A: n by n matrix A
    :param b: result vector, n by 1
    :param x: x vector, n by 1

    :return: True if the result is correct, other wise False
    """
    temp_result = A.dot_product(x)
    print("Matrix A is:")
    A.print_matrix()
    print("Vector b is:")
    b.print_matrix()
    print("Result vector x is:")
    x.print_matrix()

    for i in range(temp_result.rows):
        for j in range(temp_result.cols):
            if abs(temp_result[i][j] - b[i][j]) >= 0.001:
                return False
    return True


def choleski_decomposition(A, b):
    """
    This is the method implemented for solving the problem Ax = b,
    using Choleski Decomposition.

    Arguments:
        A: the matrix A, a real, S.P.D. (Symmetric positive definite) n * n matrix.
        b: Column vector with n rows.

    Returns:
        Column vector x with n rows.
    """
    if not A.is_symmetric():
```

```python
            raise ValueError("Matrix must be symmetric to perform Choleski Decomposition.\n")

    n = A.rows
    sparse_matrix = [[0 for _ in range(n)] for _ in range(n)]
    L = matrix(sparse_matrix, n, n)

    for j in range(n):
        if A[j][j] <= 0:
            raise ValueError("Matrix is not positive definite.\n")

        temp_sum = 0
        for k in range(-1, j):
            temp_sum += math.pow(L[j][k], 2)
        if (A[j][j] - temp_sum) < 0:
            raise ValueError("Operand under square root is not positive. Matrix is not positive definite,
    ↪  exiting.")
        L[j][j] = math.sqrt(A[j][j] - temp_sum)

        temp_sum = 0
        for i in range(j + 1, n):
            for k in range(-1, j):
                temp_sum += L[i][k] * L[j][k]
            L[i][j] = (A[i][j] - temp_sum) / L[j][j]
    # Now L and LT are all obtained, we can move to forward elimination

    y_vec = [[None for _ in range(1)] for _ in range(n)]
    y = matrix(y_vec, n, 1)
    for i in range(y.rows):
        temp_sum = 0
        if i > 0:
            for j in range(i):
                temp_sum += L[i][j] * y[j][0]
            y[i][0] = (b[i][0] - temp_sum) / L[i][i]
        else:
            y[i][0] = b[i][0] / L[i][i]

    # Now perform back substitution to find x.
    x_vec = [[None for _ in range(1)] for _ in range(n)]
    x = matrix(x_vec, n, 1)

    for i in range(n - 1, -1, -1):
        temp_sum = 0
        for j in range(i + 1, n):
            temp_sum += L[j][i] * x[j][0]
        x[i][0] = (y[i][0] - temp_sum) / L[i][i]

    return x

"""
if __name__ == "__main__":
    a_vec = [[15, -5, 0, -5], [-5, 12, -2, 0], [0, -2, 6, -2], [-5, 0, -2, 9]]
    b_vec = [[115], [22], [-51], [13]]

    A = matrix(a_vec, 4, 4)
    b = matrix(b_vec, 4, 1)

    x = choleski_decomposition(A, b)
    if check_choleski(A, b, x):
        print("Correct")
    else:
        print("Incorrect")
"""
```

Listing 3: Linear resistive networks (`linear_networks.py`).

```python
from matrix import matrix
from choleski import choleski_decomposition, check_choleski
import os

```

```python

class linearResistiveNetwork():
    def __init__(self):
        pass

if __name__ == "__main__":
    pass
```