ECSE 543 - ASSIGNMENT 1

MIDO ASSRAN - 260505216

Question 1

a. Refer to listing

Date: October 15, 2016.

```python
# ------------------------------------------------------------ #
# Cholesky Decomposition
# ------------------------------------------------------------ #
# Author: Mido Assran
# Date: 30, September, 2016
# Description: CholeskyDecomposition solves the linear system of equations:
# Ax = b by decomposing matrix A using Cholesky factorization and using
# forward and backward substitution to determine x. Matrix A must
# be symmetric, real, and positive definite.

import random
import timeit
import numpy as np
from utils import matrix_transpose

class CholeskyDecomposition(object):


    def solve(self, A, b):
        """
        :type A: np.array([float])
        :type b: np.array([float])
        :rtype: np.array([float])
        """
        start_time = timeit.default_timer()

        # If the matrix, A, is not square, exit
        if A.shape[0] != A.shape[1]:
            return None

        n = A.shape[1]


        # ------------------------------------------------------------------------------------ #
        # Simultaneous cholesky factorization of A and chol-elimination
        # ------------------------------------------------------------------------------------ #

        # Cholesky factorization & forward substitution
```

```
    for j in range(n):

        # If the matrix A is not positive definite, exit
        if A[j,j] <= 0:
            return None

        A[j,j] = A[j,j] ** 0.5     # Compute the j,j entry of chol(A)
        b[j] /= A[j,j]             # Compute the j entry of forward-sub


        for i in range(j+1, n):

            A[i,j] /= A[j,j]       # Compute the i,j entry of chol(A)
            b[i] -= A[i,j] * b[j]  # Look ahead modification of b

            if A[i,j] == 0:        # Optimization for matrix sparsity
                continue

            # Look ahead moidification of A
            for k in range(j+1, i+1):
                A[i,k] -= A[i,j] * A[k,j]
    # ----------------------------------------------------------------- #


    # ----------------------------------------------------------------- #
    # Now solve the upper traingular system
    # ----------------------------------------------------------------- #
    # Transpose(A) is the upper-tiangular matrix of chol(A)
    A[:] = matrix_transpose(A)

    # Backward substitution
    for j in range(n - 1, -1, -1):
        b[j] /= A[j,j]

        for i in range(j):
            b[i] -= A[i,j] * b[j]
    # ----------------------------------------------------------------- #

    elapsed_time = timeit.default_timer() - start_time
```

```python
        print("Execution time: ", elapsed_time, end="\n\n")

        # The solution was overwritten in the vector b
        return b

if __name__ == "__main__":
    from utils import generate_positive_semidef, matrix_dot_vector

    order = 10
    seed = 5

    print("\n", end="\n")
    print("# ------------------ TEST ------------------ #", end="\n")
    print("# ------- Cholesky Decomposition ------- #", end="\n")
    print("# ------------------------------------------ #", end ="\n\n")
    # Create a symmetric, real, positive definite matrix.
    A = generate_positive_semidef(order=order, seed=seed)
    x = np.random.randn(order)
    b = matrix_dot_vector(A=A, b=x)
    print("A:\n", A, end="\n\n")
    print("x:\n", x, end="\n\n")
    print("b (=Ax):\n", b, end="\n\n")
    chol_d = CholeskyDecomposition()
    v = chol_d.solve(A=A, b=b)
    print("chol_d.solve(A, b):\n", v, end="\n\n")
    print("2-norm error:\n", np.linalg.norm(v - x), end="\n\n")
    print("# ------------------------------------------ #", end ="\n\n")
```

Listing 1 . Cholesky.py

```python
# ------------------------------------------------------------------------ #
# Utils
# ------------------------------------------------------------------------ #
# Author: Mido Assran
# Date: 5, October, 2016
# Description: Utils provides a cornucopia of useful matrix
# and vector helper functions.

import random
import numpy as np

def matrix_transpose(A):
    """
    :type A: np.array([float])
    :rtype: np.array([floats])
    """

    # Initialize A_T(ranspose)
    A_T = np.empty([A.shape[1], A.shape[0]])

    # Set the rows of A to be the columns of A_T
    for i, row in enumerate(A):
        A_T[:, i] = row

    return A_T


def matrix_dot_matrix(A, B):
    """
    :type A: np.array([float])
    :type B: np.array([float])
    :rtype: np.array([float])
    """

    # If matrix shapes are not compatible return None
    if (A.shape[1] != B.shape[0]):
        return None

```

```python
    A_dot_B = np.empty([A.shape[0], B.shape[1]])
    A_dot_B[:] = 0  # Initialize entries of the new matrix to zero

    B_T = matrix_transpose(B)

    for i, row_A in enumerate(A):
        for j, column_B in enumerate(B_T):
            for k, v in enumerate(row_A):
                A_dot_B[i, j] += v * column_B[k]

    return A_dot_B


def matrix_dot_vector(A, b):
    """
    :type A: np.array([float])
    :type b: np.array([float])
    :rtype: np.array([float])
    """

    # If matrix shapes are not compatible return None
    if (A.shape[1] != b.shape[0]):
        return None

    A_dot_b = np.empty([A.shape[0]])
    A_dot_b[:] = 0  # Initialize entries of the new vector to zero

    for i, row_A in enumerate(A):
        for j, val_b in enumerate(b):
            A_dot_b[i] += row_A[j] * val_b

    return A_dot_b


def vector_to_diag(b):
    """
    :type b: np.array([float])
    :rtype: np.array([float])
    """
```

```python

    diag_b = np.empty([b.shape[0], b.shape[0]])
    diag_b[:] = 0    # Initialize the entries of the new diagonal matrix to zero

    for i, val in enumerate(b):
        diag_b[i, i] = val

    return diag_b

def generate_positive_semidef(order, seed=0):
    """
    :type order: int
    :type seed: int
    :rtype: np.array([float])
    """

    np.random.seed(seed)
    A = np.random.randn(order, order)
    A = matrix_dot_matrix(A, matrix_transpose(A))

    return A
```

Listing 2 . Utils.py