Solving Linear Systems with Triangular Matrices

1. Implement a function `solve_tri(A, b, lower=False)` which takes as input a triangular matrix A, a vector b and an optional boolean parameter `lower`, which is set to `False` by default. This function shall first check if the dimensions of the input parameters fit and if the matrix is invertible and if both is true, compute and output the solution x by applying the above derived formulas. Otherwise return a warning that there is a dimension mismatch or that the matrix is not invertible.

2. Test your routine on some examples with lower and upper triangular matrices. Find the corresponding SciPy Routine and compare.

Solution:

```python
import numpy as np


def solve_tri(A, b, lower=False):
    """
    Solves a system Ax = b where A is triangular.

    Parameters:
    -----------
    A : triangular matrix as numpy.ndarray of shape (n,n)
    b : right-hand side vector as numpy.ndarray of shape (n,)
    lower : boolean determining if lower or upper triangular

    Returns:
    --------
    x : solution of Ax = b as numpy.ndarray of shape (n,)
    """
    m, n = A.shape
    nb = len(b)
    d = A.diagonal()

    # test dimensions of A and b
    if m != nb:
        raise Exception('dimension of A and b must match. The dimension for A\
                        and b are: {} and {}'.format(np.shape(A), len(b)))

    # test if A is invertible: quadratic?
    elif n != m:
        raise Exception('A is not invertible, its shape is nonquadratic:\
                        {}'.format(np.shape(A)))

    # test if A is invertible: nonzero diagonals?
    elif np.any(d == 0):
        raise Exception('A is not invertible, it has zero diagonal entries')

    # A is invertible:
    else:
        x = np.zeros(n)
        # solve for lower triangular matrix
        if lower:
            for i in range(n):
                for j in range(i):
                    x[i] += 1. / A[i, i] * (-A[i, j] * x[j])
```

```
                x[i] += 1. / A[i, i] * b[i]

        # solve for upper triangular matrix
        else:
            for i in range(n)[::-1]:
                for j in range(i+1, n):
                    x[i] += 1. / A[i, i] * (-A[i, j] * x[j])
                x[i] += 1. / A[i, i] * b[i]
    return x


if __name__ == "__main__":
    ## Test dimension mismatch
#     A = np.array([[2, 0],
#                   [0, 1]])
#     b = np.array([6])
#     x = solve_tri(A, b, lower = True)

    ## Test nonqudratic A
#     A = np.array([[2, 0, 1],
#                   [0, 1, 1],])
#     b = np.array([6, 2])
#     x = solve_tri(A,b, lower = True)

    ## Test noninvertible A
#     A = np.array([[2, 0,],
#                   [0, 0],])
#     b = np.array([6,2])
#     x = solve_tri(A,b, lower = True)

    ## Test: ill-conditioned for small delta and large n
    n = 100
    delta = 1.0
    # draw from uniform distribution, shift with -0.5 and strengthen diagonal
    A = np.tril(np.random.rand(n,n)-0.5) + delta * np.eye(n)
    print("det(A) = {}\ncond(A) = {}\nmin(diag(A)) = {}".format(np.linalg.det(A),
                                                          np.linalg.cond(A),
                                                          np.min(abs(A.diagonal())
                                                          )))
    b = np.random.rand(n)
    x = solve_tri(A,b, lower = True)
    print("our test Ax=b:", np.allclose(A.dot(x),b))
    # in SciPy
    from scipy.linalg import solve_triangular
    x = solve_triangular(A, b, lower = True)
    print("scipy test Ax=b:", np.allclose(A.dot(x),b))
    # imshow on the matrix
#     import matplotlib.pyplot as plt
#     plt.imshow(A)
#     plt.show()

    ### Test ill-conditioned diagonal matrix ##
#     n = 10
#     D = np.diag(np.arange(1,n+1))
#     b = np.ones(n)
#     x = solve_tri(D, b, lower = True)
#     print("diag test Ax=b:", np.allclose(D.dot(x),b))
#     # imshow on the matrix
#     import matplotlib.pyplot as plt
#     plt.imshow(D)
#     plt.show()
#     # condition and determinant of the matrix
```

```
#     print("condition number of = ", np.linalg.cond(D))
#     print("determinant of A = ", np.linalg.det(D))
```