```
#
# Jacobi iterations 11/25/2020
# K. Rasku RN
# HDS 802 E-Term 2 2020
#
#
# INTRODUCTION:
# A system of equations Ax = b can be represented by:
#   a matrix A
#   a vector x of unknowns / independent variable-values,
#   and a vector b of outcomes
#
# There are several iterative ways to solve such a system.
#
# One way, which works on square, diagonally-dominant systems, is the Jacobi
method, named for German mathematician Carl Jacobi.
#
# The Jacobi method is a type of 'splitting method', where the results of one
iteration are fed back into the equation
# for the next iteration until one of the following outcomes is achieved:
#
#   1. Convergence is reached, and an x vector solution is found
#   2. A threshold difference between the k and k+1 values is reached  (ie. it is
determined that
#      guesses for the values of vector x are 'close enough' to vector x's true
solution)
#   3. A maximum number of iterations has been performed, but no acceptably
accurate solution for vector x has been found
#
# In linear equation systems, there are the the following possibilities:
#   1. the system has 1 solution
#   2. the system has infinite solutions (with 1 or more unknowns)
#   3. the system has no solution / represents an impossible claim
#
# DESCRIPTION:
# Let Ax = b be a diagonally-dominant square system of n linear equations such
that:
# The number of unknowns is equal to the number of equations (the order is nxn)
# A is a matrix of coefficient values for all equations in the system
# for each row in A, |aii| > |sum(all aij)|
# x is a vector of unknowns for each equation in the system for which the user
supplies an initial guess
# b is a vector of right sides for all equations in the system
# convergence is determined by the Euclidean norm between two subsequent solutions
||x(k-1)-x(k)||2 < a supplied tolerance level
# the user will have the opportunity to tell the iterator how many times to run
before giving up on finding a solution
# if the iterator finds the solution before the maximum (or default maximum) number
of iterations has been reached, a result will be given early
#
# for each iteration specified:
#       until convergence is reached:
#           the value of the next x-guess, x(k+1) = the Inverse(Diagonal(A)) * (b -
(Remainder(Diagonal(A)) * the value of the current x-guess, x(k)
#

import numpy as np
```

```
def jacobi(A, b, x=None, iters=50, conv_thresh=.0001):

    if x is None:
        x = np.zeros(len(A[0]))

    abs_a = np.abs(A)
    if np.all(2 * np.diag(abs_a) >= np.sum(abs_a, axis=1)) and A.shape[0] ==
A.shape[1]:
        D = np.diag(A)
        R = A - np.diagflat(D)
        reached = 0
        for i in range(iters):
            x_next = (b - np.dot(R, x))/D
            if np.linalg.norm(x_next - x) < conv_thresh or (x_next == x).all():
                reached = 1
            x = x_next
            if reached == 1:
                 print("The solution: " + str(x_next))
                 print("The solution was found in " + str(i) + " iterations.")
                 return
        if reached == 0:
            print("No solution was found in " + str(iters) + " iterations.")
            print("x is currently: " + str(x))
        return

    else:
        print("The matrix is not diagonally dominant or the matrix is not square.
Please use another method.")
        return


# Test Case 1
tA = np.array([[4, -1, -1],
               [-2, 6, 1],
               [-1, 1, 7]])
tb = np.array([3, 9, -6])
jacobi(tA, tb, None, 20)
print("\r")

# Test Case 2
jacobi(tA, tb, None, 20, 0.01)
print("\r")

# Test Case 3
tx = np.array([10, -2000, 4000])
jacobi(tA, tb, tx, 15)
print("\r")

# Test Case 4
jacobi(tA, tb, None, 100, 1e-20)
print("\r")
```