שאלה 1

נמספר את איברי המטריצה לפי העמודות, באופן הבא:

B באותו אופן.  
לכן:   
ולכן

מהגדרת מכפלת מטריצות:   
מכאן:   
כנדרש.

שאלה 4

נחשב פירוק LU . תחילה נחשב את U באמצעות הפיכת A למשולשית עליונה באמצעות מטריצות פרמוטציה:

לכן:

נציב . נפתור:

נפתור נקבל את הוקטור   
ב. למדנו כי *חישוב במחשב נתן את אותה התוצאה.*

*ג. תהי A מטריצה משולשית עליונה.   
נסמן את ההופכית שלה כוקטור עמודות: .  
נבחין כי כל עמודה הינה מטריצה בגודל . לכן .  
מהגדרת מטריצת היחידה, עבור כל k, הינו וקטור בו כל האיברים (מלבד האיבר ה-k) הם אפסים.  
בפרט כל האיברים אחרי k. מאחר ו-A מטריצה משולשית עליונה, כל האיברים בשורה ה-k לפני k מתאפסים, ולכן k האיברים הראשונים ב מתאפסים בסכום. עם זאת, אין הבטחה דומה עבור האיברים האחרונים, ולכן הם צריכים להיות 0 בוקטור עצמו.  
  
הראנו כי בכל עמודה k במטריצה ההופכית, כל האיברים אחרי k הראשונים בהכרח אפסים. לכן היא מטריצה משולשית עליונה כנדרש.*

*ד. תהיינה A, B מטריצות משולשיות עליונות. תהי C מטריצה כך ש- .  
נתבונן באיברים מתחת לאלכסון הראשי של C. כל איבר כזה מוגדר על ידי זוג אינדקסים .  
לכן:   
נבחין כי A משולשית עליונה. לכן בשורה ה-i, כל האיברים לפני העמודה i יהיו 0. לכן ערך המכפלה יהיה:  
באותו אופן, B משולשית עליונה ולכן בעמודה ה-j כל האיברים אחרי השורה ה-j יהיו אפסים. נבחין כי ולכן הטענה נכונה גם כלפי כל האיברים אחרי השורה ה-i. לכן תוצאת המכפלה:*

*לכן כל האיברים מתחת לאלכסון הראשי של C הינם אפסים, כלומר C משולשית עליונה.  
  
  
שאלה 5*

*בתרגול ראינו כי עבור , שיטת יעקובי תתכנס עבור כל ניחוש התחלתי אם   
 עבור נורמה כלשהי.   
נתון . לכן:  
כנדרש.*

*ב.* import numpy as np  
import math  
import matplotlib.pyplot as plt  
plt.style.use('seaborn-whitegrid')  
  
def get\_lud(matrix):  
 lu = np.copy(matrix)  
 d = np.zeros(matrix.shape)  
 dInverse = np.zeros(matrix.shape)  
  
 for rowIndex, row in enumerate(matrix):  
 lu[rowIndex][rowIndex] = 0  
 d[rowIndex][rowIndex] = row[rowIndex]  
 dInverse[rowIndex][rowIndex] = 1/row[rowIndex]  
  
 return lu, d, dInverse  
  
  
def jacobi\_step(a\_matrix, dInv, b\_vector, previous\_step):  
 ax\_prev = np.dot(a\_matrix, previous\_step)  
 b\_minus\_ax = np.subtract(b\_vector, ax\_prev)  
  
 return np.add(previous\_step, np.dot(dInv, b\_minus\_ax))  
  
def l2\_norm(vector):  
 return math.sqrt(sum([math.pow(item, 2) for item in vector]))  
  
def get\_residue\_norm(a\_matrix, b\_vector, step\_result):  
 ax = np.dot(a\_matrix, step\_result)  
 ax\_minus\_b = np.subtract(ax, b\_vector)  
  
 return l2\_norm(ax\_minus\_b)  
  
  
  
# Solves Ax = b  
def jacobi(a\_matrix, b\_vector, initial\_guess, max\_residue):  
 lu, d, dInv = get\_lud(a\_matrix)  
  
 step\_results = [initial\_guess]  
 residues = [get\_residue\_norm(a\_matrix, b\_vector, initial\_guess)]  
  
 while residues[-1] > max\_residue:  
 nextIteration = jacobi\_step(a\_matrix, dInv, b\_vector, step\_results[-1])  
  
 step\_results.append(nextIteration)  
 residues.append(get\_residue\_norm(a\_matrix, b\_vector, nextIteration))  
  
 return step\_results, residues  
  
def get\_error\_norms(a\_matrix, b\_vector, results\_steps):  
 actual\_results = np.linalg.solve(a\_matrix, b\_vector)  
 error\_values = [np.subtract(result\_step, actual\_results)  
 for result\_step in results\_steps]  
  
 return [l2\_norm(error) for error in error\_values]  
  
  
def plot(step\_results, residues, errors):  
 f, axarr = plt.subplots(3, 1)  
  
 axarr[0].set\_title("Residue norms")  
 axarr[0].semilogy(residues)  
  
 axarr[1].set\_title("Error norms")  
 axarr[1].semilogy(errors)  
  
 axarr[2].set\_title("Results")  
 axarr[2].plot([tuple[0] for tuple in results], color='red', label="x1")  
 axarr[2].plot([tuple[1] for tuple in results], color='yellow', label="x2")  
 axarr[2].plot([tuple[2] for tuple in results], color='green', label="x3")  
 axarr[2].plot([tuple[3] for tuple in results], color='blue', label="x4")  
 axarr[2].plot([tuple[4] for tuple in results], color='purple', label="x5")  
 plt.show()  
  
a = np.array([[-5, 0.2, 0.2, 0.2, 0.2],  
 [0.2, -5, 0.2, 0.2, 0.2],  
 [0.2, 0.2, -5, 0.2, 0.2],  
 [0.2, 0.2, 0.2, -5, 0.2],  
 [0.2, 0.2, 0.2, 0.2, -5]  
 ])  
b = np.array([1, 2, 3, 4, 5])  
initial\_guess = np.array([1, 1, 1, 1, 1])  
lu, d, dInv = get\_lud(a)  
  
results, residues = jacobi(a, b, initial\_guess, 0.0000001)  
errors = get\_error\_norms(a, b, results)  
print(results[-1])  
print(len(results))  
plot(results, residues, errors)

*לאחר 12 איטרציות קיבלנו* 