1η Υποχρεωτική Εργασία
Στο Μάθημα της Αριθμητικής Ανάλυσης:
Άσκηση 3

Όνοματεπώνυμο: Μπαρακλιλής Ιωάννης
ΑΕΜ: 3685

16 Δεκεμβρίου 2020

Ζητούμενο 1: Λύση του συστήματος $\mathbf{Ax} = \mathbf{b}$

Ζητείται ο προγραμματισμός μιας συνάρτησης η οποία θα παίρνει σαν είσοδο τον πίνακα $\mathbf{A}$ και το διάνυσμα $\mathbf{b}$ του γραμμικού συστήματος $\mathbf{Ax} = \mathbf{b}$ και θα επιστρέφει σαν έξοδο το διάνυσμα των αγνώστων $\mathbf{x}$, χρησιμοποιώντας την μέθοδο επίλυσης γραμμικών συστημάτων $\mathbf{PA} = \mathbf{LU}$.
Αυτό υλοποιείται στην γλώσσα python (3.7) στο αρχείο a_PA_LU.py το οποίο φαίνεται παρακάτω:

```
def swap_rows(matrix, row1, row2):
    temp = matrix[row1]
    matrix[row1] = matrix[row2]
    matrix[row2] = temp


def pivot_matrix(matrix, pivot, P):
    max_row_element_pos = pivot
    for i in range(pivot + 1, len(matrix)):
        if abs(matrix[i][pivot]) > abs(matrix[
            max_row_element_pos][pivot]):
            max_row_element_pos = i

    swap_rows(matrix, pivot, max_row_element_pos)
    swap_rows(P, pivot, max_row_element_pos)


def matrix_vector_multiplication(matrix, vector):
    result = [0 for i in range(len(vector))]
```

```
    for i in range(len(matrix)):
        for j in range(len(matrix)):
            result[i] += matrix[i][j]*vector[j]

    return result


def PLU(A):
    P = []
    U = []
    for rowNumber, row in enumerate(A):
        U.append([])
        P.append([])
        for element in row:
            U[rowNumber].append(element)
            P[rowNumber].append(0)

    for r in range(len(U)):
        P[r][r] = 1

    for row_number in range(len(U) - 1):
        pivot_matrix(U, row_number, P)
        for other_row_num in range(row_number + 1, len(U)):
            U[other_row_num][row_number] = U[other_row_num][
                row_number] / U[row_number][row_number]
            for col_number in range(row_number + 1, len(U)):
                U[other_row_num][col_number] = U[other_row_num][
                    col_number] - (U[other_row_num][row_number])
                     * \
                                                U[row_number][
                                                    col_number]

    L = [[0 for i in range(len(U))] for j in range(len(U))]
    for i in range(len(U)):
        L[i][i] = 1
    for i in range(1, len(U)):
        for j in range(i // 2 + 1):
            L[i][j] = U[i][j]
            U[i][j] = 0

    return P, L, U


def solve_system(A, b):
```

```
    P, L, U = PLU(A)

    Pb = matrix_vector_multiplication(P, b)

    y = [0.0 for i in range(len(Pb))]

    for i in range(len(L)):
        y[i] = Pb[i]
        for j in range(len(L)):
            if j == i:
                continue

            y[i] -= L[i][j] * y[j]
        y[i] = y[i] / L[i][i]

    x = [0.0 for i in range(len(y))]

    for i in range(len(U) - 1, -1, -1):
        x[i] = y[i]
        for j in range(len(U)):
            if j == i:
                continue

            x[i] -= U[i][j] * x[j]
        x[i] = x[i] / U[i][i]

    return x
```

Στον παραπάνω κώδικα:
Αρχικά ορίζω τις βοηθητικές συναρτήσεις swap_rows που δέχεται ως παραμέτρους έναν πίνακα και δύο ακέραιους και ως λειτουργία έχει να αντιμεταθέτει άμεσα (in place) τις γραμμές του πίνακα τις οποίες προσδιορίζουν οι παράμετροι.
Επίσης, ορίζεται η συνάρτηση pivot_matrix η οποία δέχεται ως παραμέτρους έναν πίνακα, έναν ακέραιο που προσδιορίζει την θέση ενός στοιχείου της διαγωνίου και έναν ακόμη πίνακα P που προσδιορίζει τον πίνακα αντιμεταθέσεων και ως λειτουργία του έχει να βρίσκει το (απολύτως) μέγιστο στοιχείο της στήλης της διαγωνίου (απο την διαγώνιο και κάτω) και να αντιμεταθέτει την γραμμή με το μέγιστο με την γραμμή του στοιχείου της διαγωνίου ενώ παράλληλα ενημερώνει τον πίνακα αντιμεταθέσεων που δέχτηκε ως παράμετρο.
Στην συνέχεια, ορίζω την βοηθητική συνάρτηση matrix_vector_multiplication που δέχεται ως παραμέτρους έναν πίνακα και ένα διάνυσμα (πίνακα στήλη) και επιστρέφει το γινόμενο του πίνακα με το διάνυσμα.

Μετά, ορίζω την συνάρτηση PLU η οποία δέχεται ως παράμετρο έναν πίνακα A και επιστρέφει τους πίνακες P, L, U που αποτελούν την παραγοντοποίηση PA = LU του πίνακα A.

Αναλυτικά:

Ορίζω τους πίνακες P, U και στην συνέχεια αντιγράφω τα περιεχόμενα του πίνακα A στον U ενώ παράλληλα εισάγω 0 σε κάθε στοιχείο του πίνακα P (ο οποίος θα έχει τελικά ίδιες διαστάσεις με τον πίνακα A). Ακολούθως, εισάγω 1 στα διαγώνια στοιχεία του πίνακα P.

Στην συνέχεια, για κάθε γραμμή του πίνακα U (εκτός της τελευταίας που παραλείπεται γιατί δεν έχει στοιχεία "άπο κάτω"): αντιμεταθέτω την γραμμή με εκείνη της οποίας το διαγώνιο (οδηγό) στοιχείο είναι απολύτως μεγαλύτερο απο αυτά που βρίσκονται απο κάτω του στοιχείο (ταυτόχρονα ενημερώνεται και ο πίνακας αντιμεταθέσεων P). Μετά, για κάθε γραμμή που ακολουθεί αποθηκεύω τον συντελεστή, με τον οποίο η αφαίρεση της γραμμής του οδηγού στοιχείου πολλαπλασιασμένη με τον συντελεστή με την τρέχουσα γραμμή θα "άφηνε" το στοιχείο της τρέχουσας γραμμής και στήλης οδηγού στοιχείου ως 0, στο στοιχείο που θα γίνονταν 0 (ώστε να μην σπαταληθεί χώρος στο να αποθηκευτούν οι συντελεστές) και αφαιρώ την υπόλοιπη γραμμή με το γινόμενο του συντελεστή αυτού επι της γραμμής του οδηγού στοιχείου. Μετά απο το τέλος αυτής της διαδικασίας, η παραγοντοποίηση PA = LU έχει τελειώσει και απομένει να οριστούν ξεκάθαρα οι επιμέρους πίνακες L, U.

Ακολούθως, δημιουργώ έναν πίνακα L ίδου μεγέθους με τον πίνακα U με κάθε στοιχείο του 0 και εισάγω 1 σε κάθε στοιχείο της διαγωνίου του.

Τελικά, εισάγω κάθε στοιχείο του πίνακα U κάτω της διαγωνίου στην αντίστοιχη θέση του πίνακα L και θέτω αυτό το στοιχείο του U σε 0.

Η διαδικασία έχει τελειώσει, επιστρέφω τους αντίστοιχους πίνακες P, L, U.

Τέλος, ορίζω την συνάρτηση solve_system που δέχεται ως παράμετρο έναν πίνακα συντελεστών A, και ένα διάνυσμα σταθερών b και επιστρέφει διάνυσμα λύσεων του συστήματος χρησιμοποιώντας την μέθοδο PA = LU.

Αναλυτικά:

Χρησιμοποιώ την συνάρτηση PLA που ορίστηκε παραπάνω και αποθηκεύω τα αποτελέσματα στους αντίστοιχους πίνακες, ενώ ορίζω τον πίνακα Pb ο οποίος αποθηκεύει το γινόμενο του πίνακα P με το διάνυσμα (πίνακα στήλη) b.

Μετά, ορίζω το διάνυσμα y και λύνω το σύστημα Ly = Pb, του οποίου τα αποτελέσματα αποθηκεύονται στο διάνυσμα y.

Τελικά, ορίζω το διάνυσμα x και λύνω το σύστημα Ux = y του οποίου τα αποτελέσματα αποθηκεύονται στο διάνυσμα x, το οποίο επιστρέφω έχοντας μόλις λύσει το σύστημα Ax = b.

Για την λύση ενός συστήματος αρκεί κάποιος να εκτελέσει την συνάρτηση solve_system με τις κατάλληλες παραμέτρους.

Ζητούμενο 2: Αποσύνθεση Cholesky

Ζητείται ο προγραμματισμός μιας συνάρτησης που δέχεται σαν είσοδο έναν συμμετρικό και θετικά ορισμένο πίνακα A και επιστρέφει έναν κάτω τριγωνικό πίνακα L που αποτελεί την αποσύνθεση Cholesky του πίνακα A.

Αυτό υλοποιείται στην γλώσσα python στο αρχείο b_Cholesky.py το οποίο φαίνεται παρακάτω:

```
import math


def matrix_multiplication(lhs_matrix, rhs_matrix):
    result = [[0.0 for j in range(len(rhs_matrix[0]))] for i in
        range(len(lhs_matrix))]

    for i in range(len(lhs_matrix)):
        for k in range(len(rhs_matrix[0])):
            for j in range(len(rhs_matrix)):
                result[i][k] += lhs_matrix[i][j] * rhs_matrix[j
                    ][k]

    return result


def cholesky_update_submatrix(matrix, prev_diagonal_position):
    rest_of_column = []
    rest_of_column_transpose = [[]]
    for row in range(prev_diagonal_position + 1, len(matrix)):
        rest_of_column.append([matrix[row][
            prev_diagonal_position]])
        rest_of_column_transpose[0].append(matrix[row][
            prev_diagonal_position])

    submatrix_to_remove = matrix_multiplication(rest_of_column,
        rest_of_column_transpose)

    for i in range(0, len(submatrix_to_remove)):
        for j in range(0, len(submatrix_to_remove)):
            matrix[i + prev_diagonal_position + 1][j +
                prev_diagonal_position + 1] -=
                submatrix_to_remove[i][j]


def cholesky_core(matrix, diagonal_position):
```

```
    if (diagonal_position >= len(matrix)):
        return matrix

    matrix[diagonal_position][diagonal_position] = math.sqrt(
        matrix[diagonal_position][diagonal_position])
    for i in range(diagonal_position + 1, len(matrix)):
        matrix[i][diagonal_position] = matrix[i][
            diagonal_position] / matrix[diagonal_position][
            diagonal_position]
        matrix[diagonal_position][i] = 0

    cholesky_update_submatrix(matrix, diagonal_position)

    return cholesky_core(matrix, diagonal_position + 1)


def cholesky(matrix):
    return cholesky_core(matrix, 0)
```

Στον παραπάνω κώδικα:
Αρχικά, ορίζεται η βοηθητική συνάρτηση matrix_multiplication που δέχεται ως παραμέτρους δύο πίνακες και επιστρέφει το γινόμενο τους.
Μετά, ορίζεται η συνάρτηση cholesky_update_submatrix η οποία δέχεται ως παραμέτρους έναν πίνακα και έναν ακέραιο που αντιστοιχεί στην θέση ενος διαγωνίου στοιχείου και έχει ως λειτουργία να "προετοιμάσει" τον πίνακα για το επόμενο βήμα του αλγορίθμου. Η αναλυτική εξήγηση της λειτουργίας του θα γίνει παρακάτω, μαζί με την εξήγηση της συνάρτησης cholesky_core που την χρησιμοποιεί.
Ακολούθως ορίζεται η αναδρομική συνάρτηση cholesky_core η οποία δέχεται ως παραμέτρους έναν πίνακα και έναν ακέραιο που αντιστοιχεί στην θέση ενος διαγωνίου στοιχείου.

Αναλυτικά:
Αρχικά, ελέγχεται άν η θέση του διαγώνιου στοιχείου του ορίσματος είναι "έκτός" του πίνακα (ο ακέραιος του ορίσματος είναι μεγαλύτερος απο το μέγεθος του πίνακα) και αν ναί, επιστρέφεται ο πίνακας του ορίσματος καθώς έχει τελειώσει η διαδικασία της αποσύνθεσης και έτσι τερματίζεται η συνάρτηση.
Στην συνέχεια το στοιχείο της διαγωνίου (αυτό με θέση του ορίσματος) αντικαθίσταται με την ρίζα του στοιχείου που βρίσκονταν πρίν σε αυτή την θέση και κάθε ένα απο τα υπόλοιπα στοιχεία της στήλης κάτω απο το στοιχείο της διαγωνίου διαιρείται με το νέο στοιχείο της διαγωνίου. Παράλληλα, κάθε στοιχείο της υπόλοιπης γραμμής δεξιά του στοιχείου της διαγωνίου τίθεται σε 0.
Στην συνέχεια ο πίνακας ενημερώνεται για το επόμενο βήμα του αλγορίθμου καλώντας την συνάρτηση cholesky_update_submatrix με παραμέτρους τον

πίνακα του ορίσματος και το τρέχον στοιχείο διαγωνίου. Έτσι, θα ενημερωθεί ο υποπίνακας ο οποίος έχει ως πρώτο στοιχείο το επόμενο στοιχείο της διαγωνίου.
Τελικά, επιστρέφεται ο πίνακας που θα επιστρέψει η αναδρομική κλήση της τρέχουσας συνάρτησης με παραμέτρους τον πίνακα του ορίσματος και το επόμενο στοιχείο της διαγωνίου. Έτσι, ο αλγόριθμος θα τελειώσει όταν γίνει επεξεργασία κάθε διαγώνιου στοιχείου, στήλης και υποπίνακα.

Εδώ είναι το σημείο όπου θα εξηγηθεί αναλυτικά η λειτουργία της συνάρτησης cholesky_update_submatrix:
Αρχικά, ορίζονται οι δύο πίνακες rest_of_column και rest_of_column_transpose (είναι πίνακας στήλη και πίνακας γραμμή αντίστοιχα) που ο πρώτος αποθηκεύει το μέρος της στήλης του πίνακα του ορίσματος κάτω απο το στοιχείο της διαγωνίου και ο δεύτερος αποθηκεύει το ανάστροφο του πρώτου.
Στην συνέχεια, αποθηκεύω τον πίνακα που προκύπτει απο τον πολλαπλασιασμό αυτών των δύο πινάκων και εκτελώ αφαίρεση αυτού του πίνακα με τον υποπίνακα του πίνακα ορίσματος που έχει ως πρώτο στοιχείο το επόμενο στοιχείο της διαγωνίου (τα στοιχεία του βρίσκονται "κάτω" απο την γραμμή του στοιχείου διαγωνίου και "δεξιά" απο την στήλη αυτού).

Τέλος, ορίζεται η συνάρτηση cholesky, η οποία δέχεται ως παράμετρο έναν συμμετρικό και θετικά ορισμένο πίνακα και επιστρέφει έναν κάτω τριγωνικό πίνακα L που αποτελεί την αποσύνθεση Cholesky του πίνακα A. Η λειτουργία του είναι να εκκινεί την συνάρτηση cholesky_core για το πρώτο (διαγώνιο) στοιχείο του πίνακα. Για να "πάρει" κάποιος την αποσύνθεση cholesky ενός συμμετρικού και θετικά ορισμένου πίνακα αρκεί κάποιος να εκτελέσει την συνάρτηση cholesky_core με παράμετρο τον πίνακα.

Ζητούμενο 3: Επίλυση συστήματος με μέθοδο Gauss-Seidel

Ζητείται ο προγραμματισμός της μεθόδου Gauss-Seidel και η χρησιμοποίηση της για επίλυση με ακρίβεια 4 δεκαδικών ψηφίων (ως προς την άπειρη νόρμα) το $n \times n$ αραιό σύστημα $\mathbf{Ax} = \mathbf{b}$ για $n = 10$ και $n = 10000$, με $A(i,i) = 5, A(i+1,i) = A(i,i+1) = -2$ και $\mathbf{b} = [3,1,1,...,1,1,3]^\top$.

Αυτό υλοποιείται στην γλώσσα python στο αρχείο c_Gauss_Seidel.py το οποίο φαίνεται παρακάτω:

```
def vector_subtraction(lhs_vector, rhs_vector):
    lhs_copy = []
    for i in range(len(lhs_vector)):
        lhs_copy.append(lhs_vector[i])

    for i in range(len(lhs_copy)):
```

```
        lhs_copy[i] -= rhs_vector[i]

    return lhs_copy


def max_norm(vector):
    max_element = abs(vector[0])

    for i in vector:
        if abs(i) > max_element:
            max_element = abs(i)

    return max_element


def gauss_seidel(A, b, initial_guess, digits_of_precision):
    x_old = [i for i in initial_guess]
    while True:
        x = x_old.copy()

        for i in range(len(x)):
            x[i] = b[i]

            for j in range(i):
                x[i] -= A[i][j] * x[j]

            for j in range(i + 1, len(x)):
                x[i] -= A[i][j] * x[j]

            x[i] = x[i] / A[i][i]

        if max_norm(vector_subtraction(x, x_old)) < 0.5 * (10
            ** (-1 * digits_of_precision)):
            return x
        else:
            x_old = x


if __name__ == '__main__':
    n = 10

    A = [[0 for it in range (n)] for itj in range (n)]

    b = [3]
```

```
for i in range(n-2):
    b.append(1)
b.append(3)

for i in range(n-1):
    A[i][i] = 5
    A[i + 1][i] = A[i][i + 1] = -2
A[n - 1][n - 1] = 5

begin_x = [1.5 for i in range(len(A))]

solution = gauss_seidel(A, b, begin_x, 4)

print("Solution␣for␣n␣=", n)
for position, i in enumerate(solution):
    print("x[", position + 1, "]=␣", i, sep="")

n = 10000

A = [[0 for it in range (n)] for itj in range (n)]

b = [3]
for i in range(n-2):
    b.append(1)
b.append(3)

for i in range(n-1):
    A[i][i] = 5
    A[i + 1][i] = A[i][i + 1] = -2
A[n - 1][n - 1] = 5

begin_x = [1.5 for i in range(len(A))]

solution = gauss_seidel(A, b, begin_x, 4)

print("Solution␣for␣n␣=", n)
for position, i in enumerate(solution):
    print("x[", position + 1, "]=␣", i, sep="")
```

Στον παραπάνω κώδικα:
Αρχικά, ορίζω την βοηθητική συνάρτηση vector_subtraction η οποία δέχεται ως παραμέτρους δύο διανύσματα και επιστρέφει την διαφορά του πρώτου με το δεύτερο.
Μετά, ορίζω την βοηθητική συνάρτηση max_norm η οποία δέχεται ως όρισμα

ένα διάνυσμα και επιστρέφει την νόρμα μεγίστου (άπειρη νόρμα) αυτού.

Ακολούθως, ορίζεται η συνάρτηση gauss_siedel η οποία δέχεται ως παραμέτρους τον πίνακα συντελεστών A, τον πίνακα σταθερών b, μία αρχική εκτίμηση για το διάνυσμα λύσης και τα απαιτούμενα ψηφία ακρίβειας.
Στην συνέχεια, αντιγράφονται τα περιεχόμενα του διανύσματος της αρχικής εκτίμησης στο διάνυσμα x_old και αρχίζει ένας ατέρμονος βρόχος που θα τερματιστεί όταν "πετύχω" την επιθυμητή ακρίβεια:

Αντιγράφω τα περιεχόμενα του διανύσματος (πίνακα) εκτίμησης λύσης στην μεταβλητή x που (θα) αποθηκεύει την νέα εκτίμηση της λύσης.
Για κάθε στοιχείο του διανύσματος (που βρίσκεται έστω στην θέση i) εκτίμησης λύσης θέτω ως τιμή του την σταθερά που βρίσκεται στην αντίστοιχη θέση (i) του πίνακα σταθερών και για κάθε στοιχείο του διανύσματος εκτίμησης λύσης εκτός του τρέχοντος αφαιρώ απο το τρέχον στοιχείο το γινόμενο του άλλου στοιχείου (που βρίσκεται έστω στην θέση j) με το στοιχείο του πίνακα συντελεστών που βρίσκεται σε γραμμή με ίδια θέση με την θέση του τρέχοντος στοιχείου στο διάνυσμα εκτίμησης λύσης (i) και στήλη με ίδια θέση με την θέση του άλλου στοιχείου στο διάνυσμα (j). Μετά, διαιρώ το τρέχον στοιχείο με το στοιχείο του πίνακα συντελεστών με γραμμή και στήλη με ίδια τιμή με την θέση του τρέχοντος στοιχείου στο διάνυσμα εκτίμησης λύσης (A[i][i]).
Ουσιαστικα, με το παραπάνω "λύνω" κάθε εξίσωση του συστήματος ως προς την μεταβλητή με ίδια θέση ως προς γραμμή και στήλη υλοποιώντας το $x_i^{(m+1)} = \frac{1}{a_{ii}}(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(m+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(m)})$ της θεωρίας.
Το παραπάνω εκτελεί τον αλγόριθμο του Gauss-Seidel γιατί μετά απο κάθε επανάληψη (υπολογισμός του επόμενου "τρέχοντος στοιχείου") το διάνυσμα αλλάζει εφόσον έχουμε υπολογίσει το προηγούμενο στοιχείο του διανύσματος και χρησιμοποιούμε το ίδιο διάνυσμα (πίνακα) ως αποθετήριο νέας λύσης και ως σημείο πρόσβασης παλαιότερων λύσεων.
Τελικα, ελέγχεται άν η νόρμα μεγίστου (απείρου) της διαφοράς του νέου και του παλιού διανύσματος είναι μικρότερη απο την ζητούμενη ακρίβεια και άν είναι επιστρέφεται η τρέχουσα εκτίμηση της λύσης. Διαφορετικά, το διάνυσμα x_old παίρνει την τιμή του x και συνεχίζεται η εκτέλεση του ατέρμονου βρόχου.

Ακολουθεί ο εκτελέσιμος κώδικας ο οποίος θα εκτελεστεί αν εκτελέσουμε το παραπάνω αρχείο ο οποίος λύνει το ζητούμενο σύστημα για n = 10 και n = 10000 ως εξής:
Αρχικά ορίζει την σταθερά n σε 10 και δημιουργεί τους πίνακες A και b σύμφωνα με την εκφώνηση .
Μετά, για αρχική εκτίμηση λύσης ορίζει το διάνυσμα με 1.5 σε όλα του τα στοιχεία (πλήθους n) και εκτελεί την συνάρτηση gauss_siedel για τα παραπάνω και τυπώνει τα αποτελέσματα στην οθόνη.
Τέλος, ακολουθεί διαδικασία ανάλογη με το πάνω για n = 10000.

Άν εκτελέσουμε τον παραπάνω κώδικα θα εμφανιστεί αρχικά η λύση του συστήματος για n = 10:

```
Solution for n = 10
x[1]= 1.0000359006561093
x[2]= 1.0000532232928523
x[3]= 1.0000574957318902
x[4]= 1.0000534609441445
x[5]= 1.0000449039516446
x[6]= 1.0000346227023011
x[7]= 1.0000245026410677
x[8]= 1.000015658125082
x[9]= 1.0000086060774946
x[10]= 1.0000034424309978
```

Στο παραπάνω στιγμιότυπο εξόδου μπορούμε να δούμε ότι η λύση του συστήματος για n = 10 είναι (με αποκοπή στα 6 δεκαδικά ψηφία) το $\mathbf{x} = [1.000035, 1.000053, 1.000057, 1.000053, 1.000044, 1.000034, 1.000024, 1.000015, 1.000008, 1.000003]^\top$

Ακολουθώντας το προηγούμενο, εμφανίζεται η λύση του συστήματος για n = 10000:

```
Solution for n = 10000
x[1]= 1.0000229551224915
x[2]= 1.0000381641625455
x[3]= 1.0000481997101471
x[4]= 1.000054790415278
x[5]= 1.000059096057985
x[6]= 1.000061892823282
x[7]= 1.0000636984029412
x[8]= 1.0000648566039527
x[9]= 1.0000655945988892
x[10]= 1.0000660616336123
x[11]= 1.000066355142335
x[12]= 1.0000665383066347
x[13]= 1.0000666518075958
x[14]= 1.000066721647562
x[15]= 1.0000667643226453
x[16]= 1.0000667902191704
x[17]= 1.0000668058270858
x[18]= 1.0000668151710452
x[19]= 1.0000668207281838
x[20]= 1.000066824011867
x[21]= 1.0000668259399224
x[22]= 1.0000668270650097
x[23]= 1.0000668277175735
x[24]= 1.0000668280938367
x[25]= 1.0000668283095393
x[26]= 1.0000668284325023
x[27]= 1.000066828502215
x[28]= 1.0000668285415268
x[29]= 1.0000668285635803
x[30]= 1.0000668285758896
x[31]= 1.0000668285827259
x[32]= 1.0000668285865044
x[33]= 1.0000668285885832
x[34]= 1.0000668285897216
x[35]= 1.0000668285903422
x[36]= 1.000066828590679
x[37]= 1.0000668285908612
x[38]= 1.0000668285909593
x[39]= 1.000066828591012
x[40]= 1.0000668285910401
```

Στο παραπάνω στιγμιότυπο εξόδου (φαίνονται τα πρώτα 40 στοιχεία της λύσης όπου τα υπόλοιπα έχουν ανάλογες τιμές) μπορούμε να δούμε ότι η λύση του συστήματος για n = 10000 είναι (με αποκοπή στα 6 δεκαδικά ψηφία) το $\mathbf{x} = [1.000022, 1.000038, ..., 1.0000025, 1.000001]^\top$