Set 2

Σεϊτανίδου Δήμητρα

2 Φεβρουαρίου 2020

Άσκηση 1

Θέλουμε να λύσουμε ένα γραμμικό σύστημα εξισώσεων $Ax = b$ χρησιμοποιώντας την μέθοδο απαλοιφής Gauss.Η μέθοδος αυτή μετατρέπει το σύστημα των εξισώσεων σε άνω-τριγωνικό, έτσι ώστε η λύση του να γίνει πολύ εύκολη. Άνω-τριγωνικός είναι ένας πίνακας που όλα τα στοιχεία του κάτω από τη διαγώνιο είναι μηδέν. Επομένως πρώτο μας βήμα είναι να μετατρέψουμε τους πίνακες συντελεστών A και b σε άνω-τριγωνικούς. Για να μην κάνουμε αυτή τη διαδικασία δύο φορές ορίζουμε ένα επαυξημένο πίνακα AA που είναι ο πίνακας A μαζί με τον b στην τελευταία του στήλη.

Στον κώδικα για να κατασκευάσουμε τον άνω-τριγωνικό πίνακα χρησιμοποιούμε ένα βρόχο στον οποίο για κάθε γραμμή του πίνακα AA θέτουμε το πρώτο μη μηδενικό της στοιχείο με μηδέν και αφαιρούμε από το κάθε επόμενο στοιχείο της το πηλίκο του πρώτου στοιχείου της γραμμής με το πρώτο στοιχείο της προηγούμενης γραμμής επί το εκάστοτε στοιχείο.

Υπάρχει όμως η περίπτωση να οδηγηθούμε σε μεγάλες ανακρίβειες αν το πρώτο στοιχείο της προηγούμενης γραμμής (δηλαδή αυτό με το οποίο διαιρούμε) είναι πολύ κοντά στο μηδέν. Σε αυτή την περίπτωση πρέπει να χρησιμοποιήσουμε μια μέθοδο που ονομάζεται οδήγηση (pivoting). Σύμφωνα με αυτή την μέθοδο πρέπει να φροντίσουμε το πρώτο στοιχείο της πρώτης γραμμής να είναι το απολύτως μεγαλύτερο.

Στον κώδικα κάνουμε pivoting πριν φτιάξουμε τον άνω-τριγωνικό πίνακα, ψάχνοντας στην πρώτη στήλη το μέγιστο κατ΄ απόλυτη τιμή στοιχείο και αλλάζοντας τις γραμμές έτσι ώστε αυτό το στοιχείο να είναι στην πρώτη γραμμή. Ο άνω-τριγωνικός επαυξημένος πίνακας στον οποίο καταλήξαμε είναι ο:

$$AA = \begin{bmatrix} 2 & -2 & 3 & -3 & -20 \\ 0 & 2 & 1 & 0 & -2 \\ 0 & 0 & 4 & 3 & 4 \\ 0 & 0 & 0 & -2 & -8 \end{bmatrix} \tag{1}$$

Τώρα αφού έχουμε φτιάξει τον άνω-τριγωνικό πίνακα AA προχωράμε στην λύση του συστήματος. Η λύση αυτή δίνεται από τις παρακάτω σχέσεις:

$$x_n = \frac{b_n^{'}}{\alpha_{nn}^{'}} \tag{2}$$

$$x_i = \frac{b_i^{'} - \sum_{j=i+1}^{n} \alpha_{ij}^{'} x_j}{\alpha_{nn}^{'}} \tag{3}$$

όπου $\alpha_{ii}^{'}$ είναι τα στοιχεία των πρώτων τεσσάρων στηλών του πίνακα AA και $b_i^{'}$ είναι τα στοιχεία της τελευταίας στήλης του AA. Στον κώδικα έχουμε ορίζει την βοηθητική συνάρτηση S που υπολογίζει το άθροισμα της σχέσης (3). Τελικά η λύση του συστήματος είναι το διάνυσμα:

$$\mathbf{x} = [\mathbf{-1}, \mathbf{0}, \mathbf{-2}, \mathbf{4}]^{\mathbf{T}}$$

Παρακάτω δίνεται ο κώδικας.

```
import numpy as np

A = np.array([[1,-1,2,-1],[2,-2,3,-3],[1,1,1,0],[1,-1,4,3]])
b = np.array([[-8],[-20],[-2],[4]])
AA = np.hstack((A,b)) #stacks arrays horizondally
n = len(b)

def S(AA,x,i,n):
    sol = 0
    for j in range(i+1,n):
        sol += AA[i][j]*x[j]
    return sol


for i in range(n):

    #pivoting
    max = abs(AA[i][i])
    maxind = i
    for j in range(i+1,n):
        if abs(AA[j][i])>max:
            max = abs(AA[j][i])
            maxind = j

    for k in range(i,n+1):
        temp = AA[maxind][k]
        AA[maxind][k] = AA[i][k]
        AA[i][k] = temp

    #creating the upper triangular matrix
    for j in range(i+1,n):
        for k in range(i+1,n+1):
            AA[j][k] += (-AA[j][i]/AA[i][i])*AA[i][k]
            AA[j][i] = 0

print(AA)
#solving the system with backsubstitution
x = np.zeros(n)
x[n-1] = AA[n-1][n]/AA[n-1][n-1]
for i in range(n-2,-1,-1):
        x[i] = 1/AA[i][i]*(AA[i][n]-S(AA,x,i,n))

print(x)
```

Άσκηση 2

Σε αυτή την άσκηση καλούμαστε να συμπληρώσουμε τον πίνακα που δίνεται στην εκφώνηση, υπολογίζοντας την αριθμητική παράγωγο της εξίσωσης $f(x) = xe^x$ για δοσμένες τιμές του x και να τις συγκρίνουμε με τις πραγματικές τιμές. Η άσκηση ζητάει ακρίβεια $O(h^2)$ οπότε θα χρησιμοποιήσουμε κεντρικές διαφορές που δίνονται από τον τύπο:

$$f^{'}(x_i) = \frac{f(x_{i+1}) - f(x_{i-1})}{2h} \tag{4}$$

όπου h η απόσταση των σημείων x_i και x_{i-1}.

Στον κώδικα χρησιμοποιώντας την σχέση (4) εύκολα βρίσκουμε την αριθμητική παράγωγο και συμπληρώνουμε τον πίνακα:

x	$f(x)$	$f^{'}(x)$
1.8	10.889365	16.987444
1.9	12.703199	19.443734
2.0	14.778112	22.228787
2.1	17.148957	25.384588
2.2	19.855030	28.958314

Για να συγκρίνουμε τις παραπάνω τιμές με τις πραγματικές υπολογίσαμε τις αντίστοιχες τιμές για την παράγωγο της συνάρτησης $f^{'}(x) = e^x(1 + x)$ και τις αφαιρέσαμε από τις παραπάνω τιμές. Τα σφάλματα που βρήκαμε δίνονται στον παρακάτω πίνακα:

x	errors
1.8	0.04843147
1.9	0.05463993
2.0	0.06161858
2.1	0.06946078
2.2	0.07827095

Από τα παραπάνω σφάλματα παρατηρούμε ότι έχουμε ακρίβεια μέχρι το πρώτο δεκαδικό ψηφίο.

Τέλος καλούμαστε να υπολογίσουμε αριθμητικά το ολοκλήρωμα της $f(x)$ στο διάστημα $x \in (1.8, 2.2)$ χρησιμοποιώντας την μέθοδο Simpson 1/3. Ο τύπος που δίνει την μέθοδο είναι:

$$\int_{\alpha=x_0}^{b=x_n} f(x)dx = \frac{h}{3}(f_1 + 4f_2 + 2f_3 + 4f_4 + ... + 2f_{n-2} + 4f_{n-1} + fn) \tag{5}$$

όπου $h = (b - \alpha)/n$ και n το πλήθος των σημείων. Στον κώδικα εφαρμόζουμε τον τύπο (5) όπου πολλαπλασιάζουμε τα στοιχεία με άρτιο δείκτη με 4 και τα στοιχεία με περιττό δείκτη με 2. Το αποτέλεσμα είναι ίσο με:

$$\mathbf{I = 4.792247}$$

Παρακάτω δίνεται ο κώδικας.

```
import numpy as np

f = lambda x: x*np.exp(x)
df = lambda x: np.exp(x)*(1+x)
x = np.array([1.7,1.8,1.9,2.0,2.1,2.2,2.3])
n = len(x)
fx = np.zeros(n)
dfx = np.zeros(n)
ndfx = np.zeros(n)
err = np.zeros(n)

#real values of f(x) and f'(x)
for i in range(n):
    fx[i] = f(x[i])
    dfx[i] = df(x[i])

#central differnces
for j in range(1,n-1):
    ndfx[j] = (f(x[j+1])-f(x[j-1]))/(2*(x[j]-x[j-1]))
    err[j] = dfx[j]-ndfx[j]

print(dfx)
print(ndfx)
print(err)

#Simpson's 1/3 integration

h = (x[n-2]-x[1])/(n-2)
integr = f(x[1])+f(x[n-2])
for k in range(2,n-2):
    if k%2 == 0:
        integr += 4*f(x[k])
    else:
        integr += 2*f(x[k])
integr = integr*(h/3)

print(integr)
```

Άσκηση 3

Στην τελευταία άσκηση ζητούμενο είναι να βρούμε την μεγαλύτερη ιδιοτιμή και το αντίστοιχο ιδιοδιάνυσμα ενός πίνακα. Για να το κάνουμε αυτό χρησιμοποιήσαμε την μέθοδο των δυνάμεων. Η μέθοδος αυτή δουλεύει ως εξής.

Αν x είναι τυχαίο διάνυσμα τότε μπορεί να γραφτεί ως γραμμικός συνδυασμός των ιδιοδιανυσμάτων u:

$$x = \alpha_1 u^{(1)} + \alpha_2 u^{(2)} + ... + \alpha_n u^{(n)} \tag{6}$$

και εφόσον $Au = \lambda u$ έχουμε:

$$x^{(1)} = Ax = \alpha_1 \lambda_1 u^{(1)} + \alpha_2 \lambda_2 u^{(2)} + ... + \alpha_n \lambda_n u^{(n)} \tag{7}$$

Μετά από k επαναλήψεις:

$$x^{(k)} = A^k x = \alpha_1 \lambda_1^k u^{(1)} + \alpha_2 \lambda_2^k u^{(2)} + ... + \alpha_n \lambda_n^k u^{(n)} \tag{8}$$

Από τη στιγμή που το λ_1 είναι η μεγαλύτερη κατ' απόλυτη τιμή ιδιοτιμή μπορούμε να πούμε ότι για $k \to \infty$:

$$x^{(k)} = A^k x \approx \alpha_1 \lambda_1^k u^{(1)} \tag{9}$$

όπου $u^{(1)}$ είναι το μεγαλύτερο ιδιοδιάνυσμα.

Στον κώδικα τώρα, αφού δηλώσουμε τον πίνακα A και ορίσουμε ένα τυχαίο διάνυσμα x υπολογίζουμε το $x^{(k)}$ για k πολλαπλασιάζοντας σε κάθε επανάληψη τα A και x. Επειδή όμως επιθυμούμε μεγάλο αριθμό επαναλήψεων, για να έχουμε όσο το δυνατό μεγαλύτερη ακρίβεια, και μετά από κάθε επανάληψη η τιμή του $x^{(k)}$ αυξάνεται, πρέπει να το κανονικοποιείσουμε, για να αποφύγουμε σφάλματα στον κώδικα. Κάνοντας αυτήν την κανονικοποίηση παίρνουμε το ιδιοδιάνυσμα $u^{(1)}$ της μεγαλύτερης ιδιοτιμής.

Για να υπολογίσουμε την ίδια την ιδιοτιμή τώρα χρησιμοποιούμε την σχέση:

$$\lambda = \frac{x^T A^T x}{x^T x} \tag{10}$$

που είναι γνωστή και ως Rayleigh quotient. Να σημειωθεί ότι στον κώδικα δεν χρειάζεται να διαιρέσουμε με το $x^T x$, επειδή αυτό έχει γίνει ήδη στην κανονικοποίηση.

Για 40 επαναλήψεις βρήκαμε ότι:

- $\lambda_1 = 8$
- $u^{(1)} = (0.408, 0.816, 0.408, 0)$

Παρακάτω δίνεται ο κώδικας.

```
import numpy as np

A = np.array([[4.0,1,2,1],[1,7,1,0],[2,1,4,-1],[1,0,-1,3]])
x = np.random.rand(4)

for i in range(40):
    xk = np.dot(A,x)
    xk_norm = np.linalg.norm(xk) #normalization
    x = xk/xk_norm #eigenvector
    l = np.dot(x,np.dot(A,x)) #eigenvalue

print(x)
print(l)
```