Επαναληπτικές Μέθοδοι

Θεωρούμε τη διάσπαση του $A = M - N$. Μια επαναληπτική μέθοδος δίνεται από

$$\left\{\begin{array}{l} x_0 \in \mathbb{R}^n, \\ Mx_{k+1} = Nx_k + b, \quad k \geq 0 \end{array}\right. \tag{1}$$

Αλγόριθμος επαναληπτικής μεθόδου

```
Επιλέγουμε αρχικά $x_{old} = 0$ και ένα επιθυμητό σφάλμα $\epsilon$
Υπολογίζουμε $r = b - Ax_{old}$
Εφόσον $\|r\| > \epsilon$
    $Mx_{new}$=$Nx_{old} + b$ #Λύνουμε το γραμμικό σύστημα
    $x_{old} = x_{new}$
    $r = b - Ax_{old}$
Τέλος εφόσον
```

Επιπλέον προσθέτουμε και έναν μετρητή για το πόσες επαναλήψεις πραγματοποιούνται. Σε περίπτωση που ο αριθμός των επαναλήψεων ξεπεράσει έναν δοσμένο αριθμό MAX, σταματάμε την εκτέλεση του αλγορίθμου. Οπότε ο αλγόριθμος διαμορφώνετε ως εξής:

```
Επιλέγουμε αρχικά $x_{old} = 0$ και ένα επιθυμητό σφάλμα $\epsilon$ και μέγιστο αριθμό επαναλήψεων $\mathrm{MAX}$
Υπολογίζουμε $r = b - Ax_{old}$
$k = 0$ # Μετρητής επαναλήψεων
Εφόσον $\|r\| > \epsilon$ και $k < MAX$
    $Mx_{new}$=$Nx_{old} + b$ #Λύνουμε το γραμμικό σύστημα
    $x_{old} = x_{new}$
    $r = b - Ax_{old}$
    $k = k + 1$ #Αυξάνουμε τον μετρητή κατά 1
Τέλος εφόσον
```

Μέθοδοι Jacobi και Gauss-Seidel

$$\underbrace{\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{12} & a_{22} & \dots & \\ \vdots & & \ddots & \vdots \\ a_{n1} & & \dots & a_{nn} \end{pmatrix}}_{A} = \underbrace{\begin{pmatrix} a_{11} & 0 & \dots & 0 \\ 0 & a_{22} & \dots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \dots & a_{nn} \end{pmatrix}}_{D} - \underbrace{\begin{pmatrix} 0 & 0 & \dots & 0 \\ -a_{21} & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ -a_{n1} & \dots & -a_{n(n-1)} & 0 \end{pmatrix}}_{L} - \underbrace{\begin{pmatrix} 0 & -a_{12} & \dots & -a_{1n} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & -a_{(n-1)n} \\ 0 & 0 & \dots & 0 \end{pmatrix}}_{U}$$

Θεωρούμε τη διάσπαση του $A = D - L - U$,

όπου $D = \mathrm{diag}(a_{ii})$ τα διαγώνια στοιχεία του πίνακα A,

L κάτω τριγωνικός πίνακας με τα στοιχεία του $-A$ κάτω από τη διαγώνιο (χωρίς τη διαγώνιο)

και U άνω τριγωνικός πίνακας με τα στοιχεία του $-A$ πάνω από τη διαγώνιο (χωρίς τη διαγώνιο).

η επαναληπτική μέθοδος Jacobi δίνεται:

$$x^{(0)} \in \mathbb{R}^m, \quad Dx^{(k)} = (L + U)x^{(k-1)} + b, \quad k = 1, 2, \dots$$

η επαναληπτική μέθοδος Gauss-Seidel δίνεται:

$$x^{(0)} \in \mathbb{R}^m, \quad (D - L)x^{(k)} = Ux^{(k-1)} + b, \quad k = 1, 2, \dots$$

Παράδειγμα εφαρμογής επαναληπτικών μεθόδων

Θα πραγματοποιήσουμε **ένα βήμα** για παράδειγμα.

Ας θεωρήσουμε τον πίνακα

$$A = \begin{pmatrix} 6 & 1 & 1 & 1 \\ 1 & 7 & 1 & 1 \\ 1 & 1 & 8 & 1 \\ 1 & 1 & 1 & 9 \end{pmatrix}$$

και το διάνυσμα b τέτοιο ώστε το $x = (1, 1, 1, 1)^T$ να είναι η ακριβής λύση του $Ax = b$.

In [1]:

```python
import numpy as np
A=np.ones((4,4)) # Δημιουργούμε έναν πίνακα 4x4 με μονάδες
###### Θετουμε τα διαγώνια στοιχεια
A[0,0]=6
A[1,1]=7
A[2,2]=8
A[3,3]=9
#############
print('A=',A)
x_sol=np.ones((4))
print('x_sol=',x_sol)
b=np.dot(A,x_sol)
print('b=',b)
```

```
A= [[6. 1. 1. 1.]
 [1. 7. 1. 1.]
 [1. 1. 8. 1.]
 [1. 1. 1. 9.]]
x_sol= [1. 1. 1. 1.]
b= [ 9. 10. 11. 12.]
```

Για να δημιουργήσουμε τους πίνακες D, L και U των μεθόδων Jacobi και Gauss-Seidel, μπορούμε να χρησιμοποιήσουμε την εντολή `diag` για να δημιουργήσουμε ένα πίνακα με τα διαγώνια στοιχεία του A και την εντολή `triu` για να δημιουργήσουμε ένα πίνακα με τα στοιχεία του A που βρίσκονται πάνω από την διαγώνιο. Με την εντολή `tril` δημιουργούμε ένα πίνακα με τα στοιχεία του A που βρίσκονται κάτω από την διαγώνιο.

In [2]:

```python
#### Δημιουργούμε τον πίνακα με τα διαγώνια στοιχεία του A
D=np.diag(np.diag(A))
#
#### Δημιουργούμε τον άνω τριγωνικό πίνακα με τα στοιχεία του A, πάνω από τη διαγώνιο
U=np.triu(-A,1)
#
#### Δημιουργούμε τον κάτω τριγωνικό πίνακα με τα στοιχεία του A, κάτω από τη διαγώνιο
L=np.tril(-A,-1)
print('D=',D)
print('L=',L)
print('U=',U)
```

```
D= [[6. 0. 0. 0.]
 [0. 7. 0. 0.]
 [0. 0. 8. 0.]
 [0. 0. 0. 9.]]
L= [[ 0.  0.  0.  0.]
 [-1.  0.  0.  0.]
 [-1. -1.  0.  0.]
 [-1. -1. -1.  0.]]
U= [[ 0. -1. -1. -1.]
 [ 0.  0. -1. -1.]
 [ 0.  0.  0. -1.]
 [ 0.  0.  0.  0.]]
```

Μέθοδος Jacobi

Αν θέσουμε $x^{(0)} = (0, 0, 0, 0)^T$ τότε $x^{(1)}$ είναι η λύση του γραμμικού συστήματος

$$Dx^{(1)} = (L + U)x^{(0)} + b$$

In [3]:

```python
x0=np.zeros(4)  # προσεγγιση για να ξεκινήσει η Jacobi. θέτουμε συνήθως το μηδενικό διάνυσμα
y=np.dot(L+U,x0)+b
x=np.linalg.solve(D,y)   # επόμενη προσεγγιση

print('Η επόμενη προσέγγιση με τη μέθοδο Jacobi, x=',x)
```

```
Η επόμενη προσέγγιση με τη μέθοδο Jacobi, x= [1.5        1.42857143 1.375      1.33333333]
```

Για να ελέγξουμε πόσο απέχουμε από την πραγματική λύση μπορούμε να χρησιμοποιήσουμε την εντολή `norm` για να υπολογίσουμε τη διαφορά των 2 διανυσμάτων

In [4]:

```python
err_total=np.linalg.norm(x_sol-x)
print('Σφάλμα από την ακριβή λύση:',err_total)
err=np.linalg.norm(x0-x)
print('Διαφορά 2 συνεχόμενων προσεγγίσεων:',err)
r=b-np.dot(A,x) #υπολογισμός υπολοίπου
err_r=np.linalg.norm(r)
print('Σφάλμα υπολοίπου:',err_r)
```

```
Σφάλμα από την ακριβή λύση: 0.8278946675144527
Διαφορά 2 συνεχόμενων προσεγγίσεων: 2.8212088019691826
Σφάλμα υπολοίπου: 8.45627634919044
```

Μέθοδος Gauss-Seidel

Αν θέσουμε $x^{(0)} = (0, 0, 0, 0)^T$ τότε $x^{(1)}$ είναι η λύση του γραμμικού συστήματος

$$(D - L)x^{(1)} = Ux^{(0)} + b$$

Πραγματοποιούμε δηλαδή ένα βήμα με τη μέθοδο Gauss-Seidel.

In [5]:

```python
x0=np.zeros(4)  # προσεγγιση για να ξεκινήσει η Gauss-Seidel. θέτουμε συνήθως το μηδενικό διάνυσμα
y=np.dot(U,x0)+b
x=np.linalg.solve(D-L,y)   # επόμενη προσεγγιση

print('Η επόμενη προσέγγιση με τη μέθοδο Gauss-Seidel, x=',x)

err_total=np.linalg.norm(x_sol-x)
print('Σφάλμα από την ακριβή λύση:',err_total)
err=np.linalg.norm(x0-x)
print('Διαφορά 2 συνεχόμενων προσεγγίσεων:',err)
```

```
Η επόμενη προσέγγιση με τη μέθοδο Gauss-Seidel, x= [1.5        1.21428571 1.03571429 0.91666667]
Σφάλμα από την ακριβή λύση: 0.5514873724714509
Διαφορά 2 συνεχόμενων προσεγγίσεων: 2.374336045156371
```

Μέθοδος κλίσεων

Θεωρούμε τη διάσπαση του $A = (\frac{1}{a}I) - (\frac{1}{a}I - A)$, για κάποιο θετικό αριθμό $a > 0$

η επαναληπτική μέθοδος των κλίσεων/ δίνεται:

$$x^{(0)} \in \mathbb{R}^m, \quad \left\{\begin{array}{l} r = b - Ax^{(k-1)} \\ x^{(k)} = x^{(k-1)} + ar, \quad k = 1, 2, \dots \end{array}\right.$$

Αλγόριθμος Μεθόδου κλίσεων

```
Επιλέγουμε αρχικά $x_{old} = 0$, $a > 0$ και ένα επιθυμητό σφάλμα $\epsilon$ και ένα μέγιστο αριθμό επαναλήψεων $\mathrm{MAX}$
Υπολογίζουμε $r = b - Ax_{old}$
$k = 0$ # Μετρητής επαναλήψεων
Εφόσον $\|r\| > \epsilon$ και $k < MAX$
    $x_{new}$=$x_{old} + ar$ #υπολογίζουμε τη νεα προσέγγιση
    $r = b - Ax_{new}$ #υπολογίζουμε το νεο υπόλοιπο
    $x_{old} = x_{new}$
    $k = k + 1$ #Αυξάνουμε τον μετρητή κατά 1
Τέλος εφόσον
```

Θεωρούμε τον πίνακα A που ειχαμε παραπάνω

In [6]:

```python
import numpy as np
A=np.ones((4,4)) # Δημιουργούμε έναν πίνακα 4x4 με μονάδες
###### Θετουμε τα διαγώνια στοιχεια
A[0,0]=6
A[1,1]=7
A[2,2]=8
A[3,3]=9
#############
print('A=',A)
x_sol=np.ones((4))
print('x_sol=',x_sol)
b=np.dot(A,x_sol)
print('b=',b)
```

```
A= [[6. 1. 1. 1.]
 [1. 7. 1. 1.]
 [1. 1. 8. 1.]
 [1. 1. 1. 9.]]
x_sol= [1. 1. 1. 1.]
b= [ 9. 10. 11. 12.]
```

Αν θέσουμε $x^{(0)} = (0, 0, 0, 0)^T$ και επιλέξουμε μια παράμετρο a

$$r^{(1)} = b - Ax^{(0)}$$

$$x^{(1)} = x^{(0)} + ar^{(1)}$$

Πραγματοποιούμε δηλαδή ένα βήμα με τη μέθοδο των κλίσεων.

In [7]:

```python
x0=np.zeros(4)  # προσεγγιση για να ξεκινήσει η μεθοδος. θέτουμε συνήθως το μηδενικό διάνυσμα
a=.1 # μια δοσμένη παράμετρος για τη μεθοδο των κλισεων.
r=b-np.dot(A,x0)  # Υπόλοιπο
x=x0+a*r  # επόμενη προσεγγιση

print('Η επόμενη προσέγγιση με τη μέθοδο των κλίσεων, x=',x)
```

```
Η επόμενη προσέγγιση με τη μέθοδο των κλίσεων, x= [0.9 1.  1.1 1.2]
```

In [8]:

```python
err_total=np.linalg.norm(x_sol-x)
print('Σφάλμα από την ακριβή λύση:',err_total)
err=np.linalg.norm(x0-x)
print('Διαφορά 2 συνεχόμενων προσεγγίσεων:',err)
r=b-np.dot(A,x) #υπολογισμός υπολοίπου
err_r=np.linalg.norm(r)
print('Σφάλμα υπολοίπου:',err_r)
```

```
Σφάλμα από την ακριβή λύση: 0.24494897427831797
Διαφορά 2 συνεχόμενων προσεγγίσεων: 2.111871208194288
Σφάλμα υπολοίπου: 2.044504830026088
```

Μέθοδος μεγίστων κλίσεων (απότομης καθόδου)

η επαναληπτική μέθοδος των μεγίστων κλίσεων δίνεται:

$$x^{(0)} \in \mathbb{R}^m, \quad \left\{\begin{array}{l} r = b - Ax^{(k-1)} \\ a = (r, r)/(Ar, r) \\ x^{(k)} = x^{(k-1)} + ar, \quad k = 1, 2, \dots \end{array}\right.$$

Αλγόριθμος μεθόδου μεγίστων κλίσεων (απότομης καθόδου)

```
Επιλέγουμε αρχικά $x_{old} = 0$, $a > 0$ και ένα επιθυμητό σφάλμα $\epsilon$ και ένα μέγιστο αριθμό επαναλήψεων $\mathrm{MAX}$
Υπολογίζουμε $r = b - Ax_{old}$, $a = (r, r)/(Ar, r)$
$k = 0$ # Μετρητής επαναλήψεων
Εφόσον $\|r\| > \epsilon$ και $k < MAX$
    $x_{new}$=$x_{old} + ar$ #υπολογίζουμε τη νεα προσέγγιση
    $r = b - Ax_{new}$ #υπολογίζουμε το νεο υπόλοιπο
    $a = (r, r)/(Ar, r)$ #υπολογίζουμε το νεο συντελεστή
    $x_{old} = x_{new}$
    $k = k + 1$ #Αυξάνουμε τον μετρητή κατά 1
Τέλος εφόσον
```

Αν θέσουμε $x^{(0)} = (0, 0, 0, 0)^T$ και η παράμετρος a προκύπτει από την ελαχιστοποιηση της συνάρτησης f στην κατευθυνση r

$$r^{(1)} = b - Ax^{(0)}$$

$$a = (r, r)/(Ar, r)$$

$$x^{(1)} = x^{(0)} + ar^{(1)}$$

Πραγματοποιούμε δηλαδή ένα βήμα με τη μέθοδο των μεγίστων κλίσεων.

In [9]:

```python
x0=np.zeros(4)  # προσεγγιση για να ξεκινήσει η μεθοδος. θέτουμε συνήθως το μηδενικό διάνυσμα

r=b-np.dot(A,x0)  # Υπόλοιπο
c=np.dot(A,r) #βοηθητική παραμετρος για να υπολογισουμε το Ar
a=np.linalg.norm(r)/np.dot(c,r) # η παράμετρος για τη μεθοδο μεγιστης κλίσης
x=x0+a*r  # επόμενη προσεγγιση

print('Η επόμενη προσέγγιση με τη μέθοδο των μεγίστων κλίσεων, x=',x)
```

```
Η επόμενη προσέγγιση με τη μέθοδο των μεγίστων κλίσεων, x= [0.03986334 0.0442926  0.04872186 0.05315112]
```

In [10]:

```python
err_total=np.linalg.norm(x_sol-x)
print('Σφάλμα από την ακριβή λύση:',err_total)
err=np.linalg.norm(x0-x)
print('Διαφορά 2 συνεχόμενων προσεγγίσεων:',err)
r=b-np.dot(A,x) #υπολογισμός υπολοίπου
err_r=np.linalg.norm(r)
print('Σφάλμα υπολοίπου:',err_r)
```

```
Σφάλμα από την ακριβή λύση: 1.9070112571093236
Διαφορά 2 συνεχόμενων προσεγγίσεων: 0.09354026845637584
Σφάλμα υπολοίπου: 20.118812178931403
```

Μέθοδος συζυγών κλίσεων

η επαναληπτική μέθοδος των συζυγών κλίσεων δίνεται:

$$x^{(0)}, p^{(0)} = r^{(0)} = b - Ax^{(0)} \in \mathbb{R}^m, \quad \left\{\begin{array}{l} a = \|r^{(k-1)}\|^2/(Ap^{(k-1)}, p^{(k-1)}) \\ x^{(k)} = x^{(k-1)} + ap^{(k-1)} \\ r^{(k)} = r^{(k-1)} - aAp^{(k-1)} \\ \beta = \|r^{(k)}\|^2/\|r^{(k-1)}\|^2 \\ p^{(k)} = r^{(k-1)} + \beta p^{(k-1)}, \quad k = 1, 2, \dots \end{array}\right.$$

Αλγόριθμος Μεθόδου συζυγών κλίσεων

```
Επιλέγουμε αρχικά $x_{old} = 0$, και ένα επιθυμητό σφάλμα $\epsilon$ και ένα μέγιστο αριθμό επαναλήψεων $\mathrm{MAX}$
Υπολογίζουμε $p_{old} = r_{old} = b - Ax_{old}$
$k = 0$ # Μετρητής επαναλήψεων
Εφόσον $\|r_{old}\| > \epsilon$ και $k < MAX$
    $a = (r_{old}, r_{old})/(Ap_{old}, p_{old})$ #υπολογίζουμε το νεο συντελεστή a
    $x_{new}$=$x_{old} + ap_{old}$ #υπολογίζουμε τη νεα προσέγγιση
    $r_{new} = r_{old} - aAp_{old}$ #υπολογίζουμε το νεο υπόλοιπο
    $\beta = \|r^{(k)}\|^2/\|r^{(k-1)}\|^2$ #υπολογίζουμε το νεο συντελεστή β
    $p_{new}$=$r_{new} + \beta p_{old}$ #υπολογίζουμε τη νεα κατευθυνση
    $x_{old} = x_{new}$
    $r_{old} = r_{new}$
    $p_{old} = p_{new}$
    $k = k + 1$ #Αυξάνουμε τον μετρητή κατά 1
Τέλος εφόσον
```

In [11]:

```python
x0=np.zeros(4)  # προσεγγιση για να ξεκινήσει η μεθοδος. θέτουμε συνήθως το μηδενικό διάνυσμα

r0=b
p0=b

c=np.dot(A,p0) #βοηθητική παραμετρος, υπολογιζουμε το Ap
a=np.linalg.norm(r0)**2/np.dot(c,p0) # η παράμετρος α για τη μεθοδο συζυγών κλίσεων

x=x0+a*p0  # επόμενη προσεγγιση

#### Προετοιμασια για το επόμενο βήμα
r=r0-a*c  # Υπόλοιπο
beta=np.linalg.norm(r)**2/np.linalg.norm(r0)**2 # η παράμετρος β για τη μεθοδο συζυγών κλίσεων
p=r+beta*p0  # επόμενη κατευθυνση


print('Η επόμενη προσέγγιση με τη μέθοδο των συζυγών κλίσεων, x=',x)
```

```
Η επόμενη προσέγγιση με τη μέθοδο των συζυγών κλίσεων, x= [0.84186242 0.93540268 1.02894295 1.12248322]
```

In [12]:

```python
err_total=np.linalg.norm(x_sol-x)
print('Σφάλμα από την ακριβή λύση:',err_total)
err=np.linalg.norm(x0-x)
print('Διαφορά 2 συνεχόμενων προσεγγίσεων:',err)
r=b-np.dot(A,x) #υπολογισμός υπολοίπου
err_r=np.linalg.norm(r)
print('Σφάλμα υπολοίπου:',err_r)
```

```
Σφάλμα από την ακριβή λύση: 0.2121795057816693
Διαφορά 2 συνεχόμενων προσεγγίσεων: 1.9754499975978441
Σφάλμα υπολοίπου: 1.3402739018232337
```