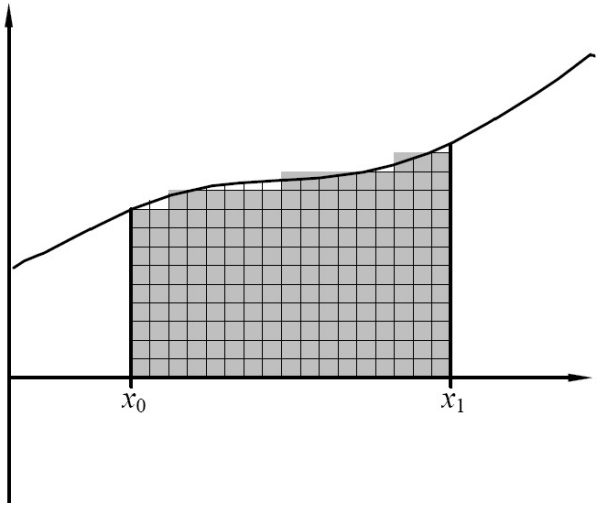
Αριθμητική Ολοκλήρωση

Αριθμητική ολοκλήρωση

- Υπάρχουν πολλοί λόγοι που κάποιος θέλει να κάνει αριθμητική ολοκλήρωση:
  - Το ολοκλήρωμα είναι δύσκολο να υπολογισθεί αναλυτικά
  - Ολοκλήρωση πίνακα δεδομένων
- Διάφοροι τρόποι ολοκλήρωσης ανάλογα με το πρόβλημα

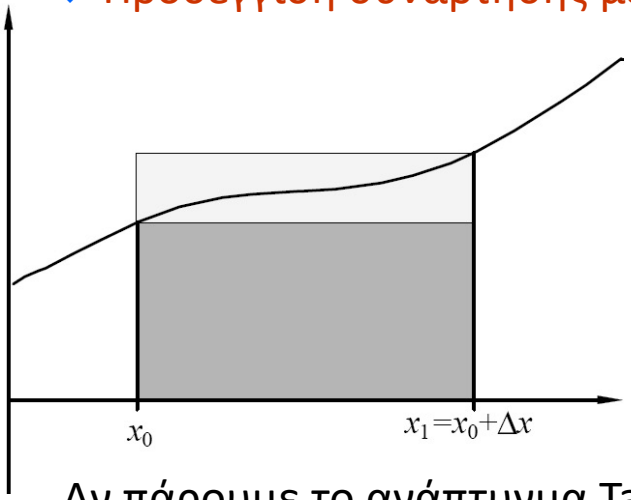
Ολοκλήρωση με το "χέρι"



- Ένας τρόπος είναι να χρησιμοποιήσουμε ένα πλέγμα πάνω στο γράφημα της συνάρτησης προς ολοκλήρωση και να μετρήσουμε τα τετράγωνα (μόνο αυτά που περιέχονται κατά 50% από τη συνάρτηση).
- Αν οι υποδιαιρέσεις του πλέγματος (τετράγωνα) είναι πολύ μικρές τότε μπορούμε να προσεγγίσουμε αρκετά καλά το ολοκλήρωμα της συνάρτησης.

Αριθμητική ολοκλήρωση

❖ Προσέγγιση συνάρτησης με σταθερά



- Ο πιο απλός τρόπος ολοκλήρωσης.
- Υποθέτουμε ότι η συνάρτηση f(x) είναι σταθερή στο διάστημα (x_0,x_1)
- Η μέθοδος δεν είναι ακριβής και οδηγεί σε αμφίβολα αποτελέσματα ανάλογα με το αν η σταθερά επιλέγεται στην αρχή ή το τέλος του διαστήματος ολοκλήρωσης.

Αν πάρουμε το ανάπτυγμα Taylor της συνάρτησης f(x) ως προς το κατώτερο όριο:

$$\int_{x_0}^{x+\Delta x} f(x)dx = \int_{x_0}^{x+\Delta x}\left[f(x_0)+f'(x_0)(x-x_0)+\frac{1}{2}f''(x_0)(x-x_0)^2+\cdots\right]dx$$

$$= f(x_0)\Delta x+\frac{1}{2}f'(x_0)(x-x_0)^2+\frac{1}{6}f''(x_0)(x-x_0)^3+\cdots \qquad = f(x_0)\Delta x+O(\Delta x^2)$$

σταθερά

Αν η σταθερά λαμβάνεται από το πάνω όριο ολοκλήρωσης θα είχαμε:

$$\int_{x_0}^{x_0+\Delta x} f(x)dx = f(x_0+\Delta x)\Delta x+O(\Delta x^2)$$

- Το σφάλμα και στις 2 περιπτώσεις είναι τάξης $O(\Delta x^2)$ με το συντελεστή να καθορίζεται από τη τιμή της $1^{ης}$ παραγώγου

Αριθμητική ολοκλήρωση – Κανόνας του τραπεζίου

Θεωρήστε το ανάπτυγμα της σειράς Taylor που ολοκληρώνεται μεταξύ x_0 και $x_0+\Delta x$

$$\int_{x_0}^{x+\Delta x} f(x)dx = \int_{x_0}^{x+\Delta x} \left[f(x_0) + f'(x_0)(x-x_0) + \tfrac{1}{2} f''(x_0)(x-x_0)^2 + \cdots \right] dx$$

$$= f(x_0)\Delta x + \tfrac{1}{2} f'(x_0)\Delta x^2 + \tfrac{1}{6} f''(x_0)\Delta x^3 + \cdots$$

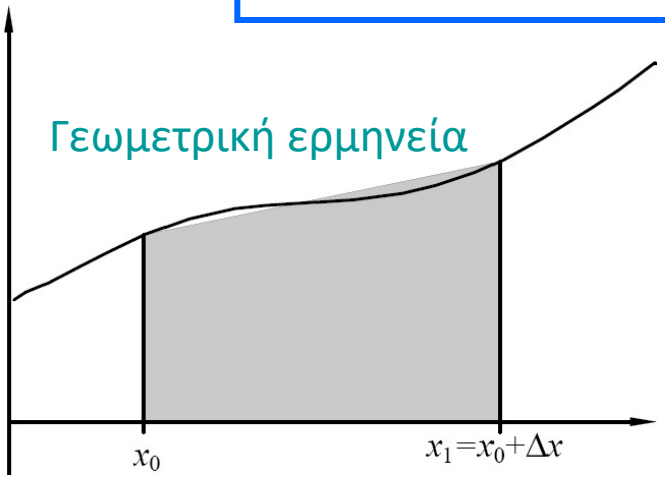
$$= \left\{ \tfrac{1}{2} f(x_0) + \tfrac{1}{2}\left[f(x_0) + f'(x_0)\Delta x + \tfrac{1}{2} f''(x_0)\Delta x^2 + \cdots \right] - \tfrac{1}{12} f''(x_0)\Delta x^2 + \cdots \right\} \Delta x$$

$$= \boxed{\tfrac{1}{2}\left[f(x_0) + f(x_0+\Delta x) \right]\Delta x} + O(\Delta x^3)$$

Κανόνας του τραπεζίου

Σφάλμα προσέγγισης

Γεωμετρική ερμηνεία

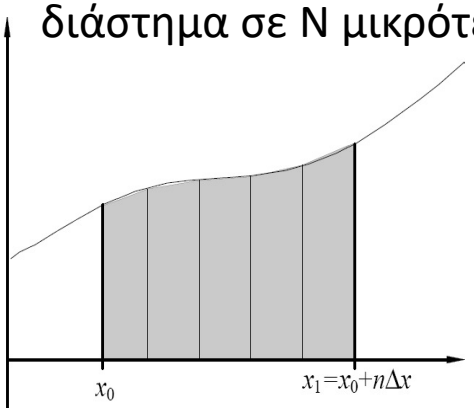


Αφού το σφάλμα ελαττώνεται κατά Δx^3 κάνοντας το διάστημα μισό το σφάλμα θα μικραίνει κατά 8

Αλλά η περιοχή θα ελαττωθεί στο μισό και θα πρέπει να χρησιμοποιήσουμε το κανόνα 2 φορές και να αθροίσουμε.
Το σφάλμα τελικά ελαττώνεται κατά 4

Αριθμητική ολοκλήρωση – Κανόνας του τραπεζίου

Συνήθως όταν θέλουμε να ολοκληρώσουμε σε ένα διάστημα x_0, x_1 χωρίζουμε το διάστημα σε N μικρότερα διαστήματα $\Delta x=(x_1-x_0)/N$



Εφαρμόζοντας το κανόνα του τραπεζίου

$$\int_{x_0}^{x_1} f(x)dx = \sum_{i=0}^{n-1} \int_{x_0+i\Delta x}^{x_0+(i+1)\Delta x} f(x)dx$$

$$\approx \frac{\Delta x}{2}\sum_{i=0}^{n-1}\left[f\left(x_0+i\Delta x\right)+f\left(x_0+(i+1)\Delta x\right)\right] \Rightarrow$$

$$\int_{x_0}^{x_1} f(x)dx \approx \frac{\Delta x}{2}\left[f\left(x_0\right)+2f\left(x_0+\Delta x\right)+2f\left(x_0+2\Delta x\right)+\cdots+2f\left(x_0+(n-1)\Delta x\right)+f\left(x_1\right)\right]$$

Ενώ το σφάλμα για κάθε βήμα είναι Δx^3 το συνολικό σφάλμα είναι αθροιστικό ως προς όλα τα βήματα (N) και επομένως θα είναι N φορές $O(\Delta x^2)\sim O(N^{-2})$

Στα παραπάνω υποθέσαμε ότι το βήμα, Δx, είναι σταθερό σε όλο το διάστημα.
Θα μπορούσε ωστόσο να μεταβάλλεται σε μια περιοχή (να ’ναι πιο μικρό) ώστε να έχουμε μικρότερο σφάλμα. π.χ. περιοχές με μεγάλη καμπύλωση της συνάρτησης
Προσοχή το σφάλμα παραμένει και πάλι της τάξης $O(\Delta x^2)$ αλλά οι υπολογισμοί θα είναι πιο ακριβείς
Αυτό το κάνουμε γιατί σε περιοχές μεγάλης καμπύλωσης η 2^{η} παράγωγος της συνάρτησης θα γίνει πολύ μεγάλη οπότε μικρότερο Δx θα κρατήσει τον όρο μικρό

Παράδειγμα για κανόνα του τραπεζίου

```
#!/usr/bin/python3
import numpy as np
import matplotlib.pyplot as plt

def Trapezoidal(f, a, b, n):
   dx = (b-a)/float(n)        #Eyros ypodiastimatos – n arithmos ypodiastimatwn
                              # a kai b katw kai panw orio oloklirwsis
   s  = 0.5*(f(a) + f(b))     #H prwti kai teleutaia pleyra
   for i in range(1,n):
      s = s + f(a + i*dx)     # H timi tis synartisis stα endiamesa diastimata
   return dx*s

def myfunc(t):
   return np.exp(-t**4)

a = -2; b = 2                 #panw kai katw orio
n = 1                         #arithmos upodiastimatwn
print("The integral of the function exp(-x**4) in the range [-2,2]")
print("\t Nsteps\t Value")
for i in range(21):
   result = Trapezoidal(myfunc, a, b, n)
   print("\t %5d \t %7.5f " %(n,result))
   n *=2
```

Αριθμητική ολοκλήρωση – Κανόνας μέσου σημείου

Μια παραλλαγή του κανόνα του τραπεζίου είναι ο κανόνας του μέσου σημείου

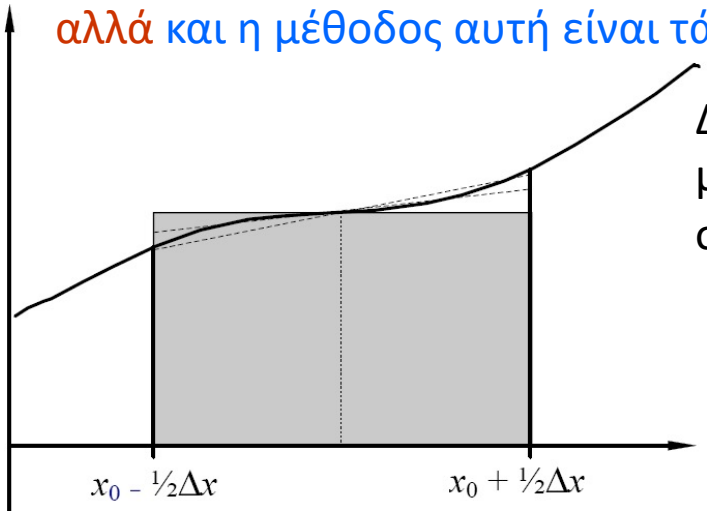
Η ολοκλήρωση του αναπτύγματος Taylor γίνεται από x_0-Δx/2 σε x_0+Δx/2 οπότε:

$$\int_{x_0-\frac{1}{2}\Delta x}^{x_0+\frac{1}{2}\Delta x} f(x)dx = \int_{x_0-\frac{1}{2}\Delta x}^{x_0+\frac{1}{2}\Delta x} \left[f(x_0) + f'(x_0)(x-x_0) + \frac{1}{2} f''(x_0)(x-x_0)^2 + \cdots \right] dx \Rightarrow$$

$$\int_{x_0-\frac{1}{2}\Delta x}^{x_0+\frac{1}{2}\Delta x} f(x)dx \approx f(x_0)\Delta x + \frac{1}{24} f''(x_0)\Delta x^3 + \cdots$$

Υπολογίζοντας τη συνάρτηση στο μέσο κάθε διαστήματος το σφάλμα μπορεί να ελαττωθεί κάπως μια και ο παράγοντας μπροστά από τον όρο της 2ης παραγώγου είναι 1/24 αντί του 1/12 που έχουμε στη μέθοδο του τραπεζίου

αλλά και η μέθοδος αυτή είναι τάξης $O(\Delta x^2)$



Διαχωρίζοντας το διάστημα σε υποδιαστήματα μπορούμε να ελαττώσουμε το σφάλμα όπως και στην περίπτωση του κανόνα του τραπεζίου:

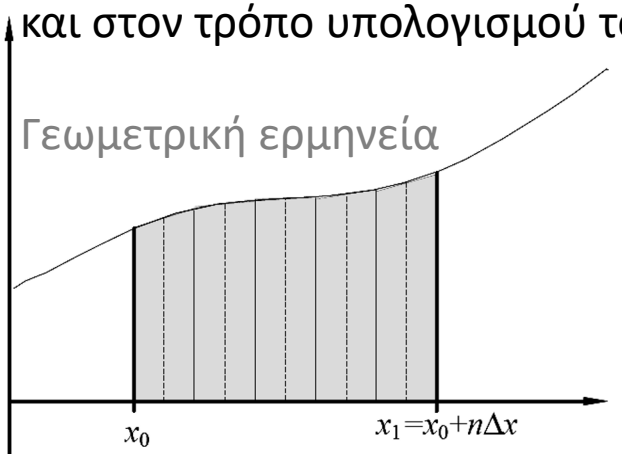
$$\int_{x_0}^{x_1} f(x)dx \approx \Delta x \sum_{i=0}^{n-1} f\left(x_0 + \left(i + \frac{1}{2}\right)\Delta x\right)$$

Αριθμητική ολοκλήρωση – Κανόνας μέσου σημείου

Κανόνας μέσου σημείου: $$\int_{x_0}^{x_1} f(x)dx \approx \Delta x \sum_{i=0}^{n-1} f\left(x_0 + \left(i + \tfrac{1}{2}\right)\Delta x\right)$$

Κανόνας τραπεζίου: $$\int_{x_0}^{x_1} f(x)dx \approx \frac{\Delta x}{2}\left[f(x_0) + 2f(x_0+\Delta x) + 2f(x_0+2\Delta x) + \cdots + 2f(x_0+(n-1)\Delta x) + f(x_1)\right]$$

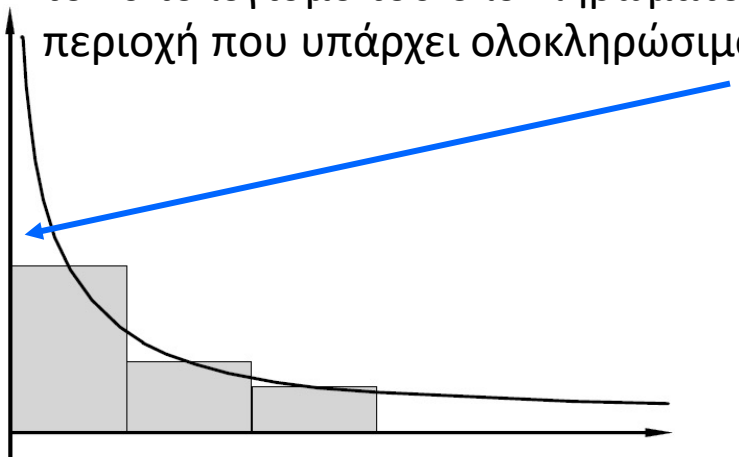
Η διαφορά τους είναι μόνο στη "φάση" των σημείων και της περιοχής υπολογισμού, και στον τρόπο υπολογισμού του πρώτου και τελευταίου διαστήματος



Ωστόσο υπάρχουν δύο πλεονεκτήματα της μεθόδου του ενδιάμεσου σημείου σε σχέση με την μέθοδο του τραπεζίου:

A) Χρειάζεται ένα υπολογισμό της f(x) λιγότερο

B) Μπορεί να χρησιμοποιηθεί πιο αποτελεσματικά για τον υπολογισμό του ολοκληρώματος κοντά σε μια περιοχή που υπάρχει ολοκληρώσιμο ιδιάζων σημείο



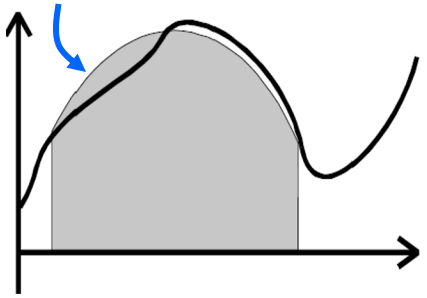
Παράδειγμα για κανόνα του ενδιάμεσου σημείου

```
#! /usr/bin/python3
import numpy as np

def midpoint(f, a, b, n):
     dx = float(b-a)/n
     result = 0
     for i in range(n):
          result += f((a + dx/2.0) + i*dx)
          result *= dx
     return result

def myfunc(t):
      return 3*(t**2)*np.exp(t**3)

n = int(input('n: ‘))
uplim = 1
lolim = 0
numerical = midpoint(myfunc, lolim, uplim, n)
Vup = np.exp(uplim**3)
Vlo = np.exp(lolim**3)
exact = Vup - Vlo
error = exact - numerical
print('n=%d: %.8f, error: %10g' % (n,
numerical, error))
```

Αριθμητική ολοκλήρωση – Κανόνας του Simpson

Ένας διαφορετικός τρόπος προσέγγισης από το να ελαττώνουμε το μέγεθος του Δx στην ολοκλήρωση για καλύτερη ακρίβεια είναι να αυξήσουμε την ακρίβεια των συναρτήσεων που χρησιμοποιούμε για να προσεγγίσουμε το ολοκλήρωμα.

2ου βαθμού προσέγγιση της f(x)



Ολοκληρώνοντας το ανάπτυγμα Taylor σε ένα διάστημα 2Δx

$$\int_{x_0}^{x_0+2\Delta x} f(x)dx = 2f(x_0)\Delta x + 2f'(x_0)\Delta x^2 + \tfrac{4}{3}f''(x_0)\Delta x^3 + \tfrac{2}{3}f'''(x_0)\Delta x^4 + \tfrac{4}{15}f^{iv}(x_0)\Delta x^5 \cdots$$

$$\int_{x_0}^{x_0+2\Delta x} f(x)dx = \frac{\Delta x}{3}\Big[f(x_0) +$$

$$+4\left(f(x_0) + f'(x_0)\Delta x + \tfrac{1}{2}f''(x_0)\Delta x^2 + \tfrac{1}{6}f'''(x_0)\Delta x^3 + \tfrac{1}{24}f^{iv}(x_0)\Delta x^4 + \cdots\right)$$

$$+\left(f(x_0) + 2f'(x_0)\Delta x + 2f''(x_0)\Delta x^2 + \tfrac{4}{3}f'''(x_0)\Delta x^3 + \tfrac{2}{3}f^{iv}(x_0)\Delta x^4 + \cdots\right) -$$

$$-\tfrac{17}{30}f^{iv}(x_0)\Delta x^4 \cdots\Big] \Rightarrow$$

$$\int_{x_0}^{x_0+2\Delta x} f(x)dx = \frac{\Delta x}{3}\Big[f(x_0) + 4f\left(x_0+\Delta x\right) + f\left(x_0+2\Delta x\right)\Big] + O\left(\Delta x^5\right)$$

Ο κανόνας του Simpson είναι 2 τάξεις περισσότερο ακριβής από αυτόν του τραπεζίου δίνοντας ακριβή ολοκλήρωση κύβων

Αριθμητική ολοκλήρωση – Βελτιστοποίηση Simpson

Χωρίζοντας το διάστημα από x_0 σε x_1 σε μικρότερα (N) διαστήματα, μπορούμε να αυξήσουμε την ακρίβεια του αλγόριθμου.
Το γεγονός ότι χρειαζόμαστε 3 σημεία για την ολοκλήρωση κάθε διαστήματος απαιτεί να υπάρχουν άρτια σε πλήθος διαστήματα.
Άρα θα πρέπει να εκφράσουμε το πλήθος των διαστημάτων σαν N = 2m.

Θα έχουμε:

$$\int_{x_0}^{x_1} f(x)dx \approx \frac{\Delta x}{3}\sum_{i=0}^{m-1}\left[f(x_0+2i\Delta x)+4f\left(x_0+(2i+1)\Delta x\right)+f\left(x_0+(2i+2)\Delta x\right)\right]$$

$$\int_{x_0}^{x_1} f(x)dx \approx \frac{\Delta x}{3}\left[f\left(x_0\right)+4f\left(x_0+\Delta x\right)+2f\left(x_0+2\Delta x\right)+\cdots+4f\left(x_0+(N-1)\Delta x\right)+f\left(x_1\right)\right]$$

Παράδειγμα για κανόνα Simpson

Χρήση του κανόνα Simpson για την προσέγγιση του ολοκληρώματος $\int_0^{\pi} sinxdx$ χωρίζοντας το διάστημα ολοκλήρωσης σε 10 υποδιαστήματα

```
#! /usr/bin/python3
import numpy as np

def simpson(f, a, b, n):          # n einai o arithos twn simeiwn
                                  # 11 stin periptwsi 10 ypodiastimatwn
     dx = float(b-a)/(n-1)
     result = 0
     for i in range(0,n-2,2):
        xa = a + i*dx
        xb = a + (i+1)*dx
        xc = a + (i+2)*dx
        result += (f(xa)+ 4*f(xb)+ f(xc))
     result *= dx/3
     return result

def myfunc(t):
   return np.sin(t)
npnts = int(input('npnts: '))
lolim = 0
uplim = np.pi
exact = 2.0
numerical = simpson(myfunc, lolim, uplim, npnts)
error = exact - numerical
print('n=%d: %.8f, error: %10g' % (npnts, numerical, error))
```

Χρήση της scipy βιβλιοθήκης

Η Python έχει μια βιβλιοθήκη η οποία περιέχει όλες τις μεθόδους ολοκλήρωσης . επίλυσης διαφορικών εξισώσεων και λύσης μη γραμμικών εξισώσεων

Η βιβλιοθήκη ονομάζεται **scipy** και θα χρειαστεί να την εγκαταστήσετε στο laptop σας ή στο λογαριασμό σας στο εργαστήριο Η/Υ του τμήματος δίνοντας την εντολή: **python3 –m pip install --user scipy** **(**2- πριν το user, 1- πριν το m**)**

```
import numpy as np
from scipy.integrate import trapz
a = 0
b = np.pi
n = 11
h = (b - a) / (n - 1)
x = np.linspace(a, b, n)
f = np.sin(x)
I_trapz = trapz(f,x)
I_trap = (h/2)*(f[0] + 2 * sum(f[1:n-1]) + f[n-1])
print(I_trapz)
print(I_trap)
```

Δημιουργία Κινούμενων Εικόνων με την Python

Κινούμενες εικόνες (animation) και Python

Μέχρι τώρα έχουμε χρησιμοποιήσει την κλάση pyplot της βιβλιοθήκης matplotlib για να κάνουμε διάφορα γραφήματα.

Η βιβλιοθήκη matplotlib δίνει την δυνατότητα για την δημιουργία κινούμενων εικόνων που βοηθά περισσότερο στην κατανόηση προβλημάτων και λύσεων που εξάγονται.

Υπάρχουν δύο βασικοί τρόποι με τους οποίους μπορούν να διασυνδεθούν τα αποτελέσματα της python με το πακέτο των κινούμενων εικόνων

TimedAnimation

FuncAnimation

Θα επικεντρωθούμε στην περίπτωση της μεθόδου FuncAnimation που δημιουργεί κινούμενη εικόνα με την επαναληπτική κλήση μιας συνάρτησης

Η εικόνα κινείται με την χρήση ενός χρονομετρητή στον οποίο αναφέρεται η εικόνα της κλάσης Animation

Μια κινούμενη εικόνα μπορεί να αποθηκευτεί στο δίσκο χρησιμοποιώντας τις μεθόδους **Animation.save** ή **Animation.to_html5_video**

Ένα απλό παράδειγμα

Τα εργαλεία για animations σχετίζονται την κλάση: **matplotlib.animation.Animation**

```
import numpy as np
from matplotlib import pyplot as plt
from matplotlib import animation
```

```
fig = plt.figure()
ax = plt.axes(xlim(0,2),ylim(-2,2)
line, = ax.plot([], [], lw=2)
```

δημιουργία της εικόνας

των αξόνων x και y με τα αντίστοιχα όρια

δημιουργία του στοιχείου του γραφήματος (line) που σκοπεύουμε να κινούμε

Προσοχή ότι το αντικείμενο line ακολουθείται από ένα **,** γιατί η μέθοδος plot επιστρέφει tuple

```
def init():
    line.set_data([],[])
    return line,
```

Συνάρτηση αρχειοποίησης: σχεδιάση του υποβάθρου κάθε στιγμιότυπου της εικόνας

Σημαντικό, η συνάρτηση αυτή να επιστρέφει το αντικείμενο line, γιατί αυτό λέει στο γεννήτορα της κίνησης που είναι το αντικείμενο στο γράφημα που θα πρέπει να ανανεωθεί

Ένα απλό παράδειγμα

```
def animate(i):
    x = np.linespace(0,2,1000)
    y = np.sin(2*np.pi*(x-0.01 * i))
    line. set_data(x,y)
    return line,
```

Η συνάρτηση η κινούμενη εικόνα της οποίας θα δημιουργηθεί με επαναληπτική κλήση της

Η συνάρτηση έχει ένα μόνο όρισμα, το i, που αντιπροσωπεύει τον αριθμό της εικόνας (frame).

Η συνάρτηση σχεδιάζει ένα ημιτονοειδές κύμα το οποίο παρουσιάζει μετατόπιση που εξαρτάται από την παράμετρο i

Η συνάρτηση επιστρέφει και πάλι το αντικείμενο που έχει ανανεωθεί ώστε να χρησιμοποιηθεί από τον γεννήτορα των κινούμενων εικόνων.

```
anim = animation.FuncAnimation(fig, animate,
       init_func=init, frames=200, interval=20, blit=True)
```

Το αντικείμενο που κινείται. Γίνεται ανάθεση σε μια μεταβλητή για να διατηρείται.

Επιλέξαμε 200 frames (εικόνες) με 20ms μεταξύ τους

Η παράμετρος blit είναι πολύ σημαντική αφού ενημερώνει για ποια στοιχεία δεν έχουν αλλάξει και ποια θα πρέπει να επανασχεδιαστούν αφού έχουν αλλάξει. Επειδή σχεδιάζονται λιγότερα αντικείμενο, η χρήση της blit=True επιταχύνει την εκτέλεση

Ένα απλό παράδειγμα

```
import numpy as np
from matplotlib import pyplot as plt
from matplotlib import animation

fig = plt.figure()
ax = plt.axes(xlim=(0, 2), ylim=(-2, 2))
line, = ax.plot([], [], lw=2)

def init():
   line.set_data([], [])
   return line,

def animate(i):
   x = np.linspace(0, 2, 1000)
   y = np.sin(2 * np.pi * (x - 0.01 * i))
   line.set_data(x, y)
   return line,

anim = animation.FuncAnimation(fig, animate, init_func=init, frames=200,
                               interval=20, blit=True)
# save the animation as an mp4. This requires ffmpeg or mencoder to be
# installed. Do it: python3 -m pip install --user ffmpeg-python
#http://matplotlib.sourceforge.net/api/animation_api.html

anim.save('basic_animation.mp4', fps=30, extra_args=['-vcodec', 'libx264'])

plt.show()
```