Οπτικοποίηση δεδομένων

Ιστογράμματα

Τα ιστογράμματα αποτελούν έναν εύχρηστο οπτικό τρόπο για να εξαγάγουμε την κατανομή που ακολουθούν μια σειρά από μεγάλο αριθμό μετρήσεων ενός μεγέθους αλλά παράλληλα δίνουν την δυνατότητα για εύκολη στατιστική ανάλυση μετρήσεων.

Αν έχουμε μια σειρά από μεγάλο αριθμό μετρήσεων ενός μεγέθους, τότε μπορούμε να ταξινομήσουμε τις μετρήσεις και να βρούμε τη συχνότητα εμφάνισης κάθε τιμής.

Δεν κρατάμε κάθε μέτρηση ξεχωριστά αλλά χωρίζουμε το εύρος των τιμών σε κατάλληλα υποδιαστήματα και αθροίζουμε τις τιμές που πέφτουν σε κάθε υποδιάστημα. Με τον τρόπο αυτό παίρνουμε την κατανομή της συχνότητας εμφάνισης κάθε τιμής.

Για παράδειγμα, έστω ότι μετρήσαμε κάποιο μέγεθος 30 φορές και βρήκαμε:

8.16	8.10	8.12	8.12	8.16	8.16
8.14	8.18	8.12	8.10	8.21	8.14
8.12	8.18	8.17	8.14	8.14	8.17
8.16	8.18	8.06	8.09	8.16	8.18
8.18	8.24	8.10	8.16	8.13	8.21

Δηλαδή κάθε τιμή x_i εμφανίζεται n_i φορές

Η μέση τιμή μπορεί να γραφεί ως:

$$\overline{x} = \frac{\sum x_i}{N} = \frac{\sum x_k n_k}{\sum n_k}$$

Οι τιμές των μετρήσεων έχουν συχνότητα:

Τιμή	#	Τιμή	#	Τιμή	#
8.06	1	8.13	1	8.20	0
8.07	0	8.14	4	8.21	2
8.08	0	8.15	0	8.22	0
8.09	1	8.16	6	8.23	0
8.10	3	8.17	2	8.24	1
8.11	0	8.18	5		
8.12	4	8.19	0		

Ιστογράμματα

Ομαδοποιώντας τις μετρήσεις, βρίσκουμε ευκολότερα πόσο συνεισφέρει κάθε τιμή στο άθροισμα της μέσης τιμής: $\sum x_k n_k$

Στο παράδειγμα, όλες οι τιμές βρίσκονται στο διάστημα 8.00 - 8.30 και άρα η πραγματική τιμή θα βρίσκεται στο διάστημα αυτό. Εφόσον οι τιμές 8.12 - 8.18 είναι οι πιο συχνές περιμένουμε ότι και η πραγματική τιμή θα είναι στο υποδιάστημα αυτό.

Όντως η μέση τιμή είναι 8.15.

Συνήθως χωρίζουμε το διάστημα σε ίσα υποδιαστήματα κατάλληλου εύρους ώστε στο καθένα να συμπεριλαμβάνεται μεγάλος αριθμός μετρήσεων και βρίσκουμε το ποσοστό των μετρήσεων σε κάθε υποδιάστημα,

$F_k = \frac{n_k}{N}$ **Συχνότητα εμφάνισης** μιας μέτρησης x_k.

$$\overline{x} = \sum_k F_k x_k$$

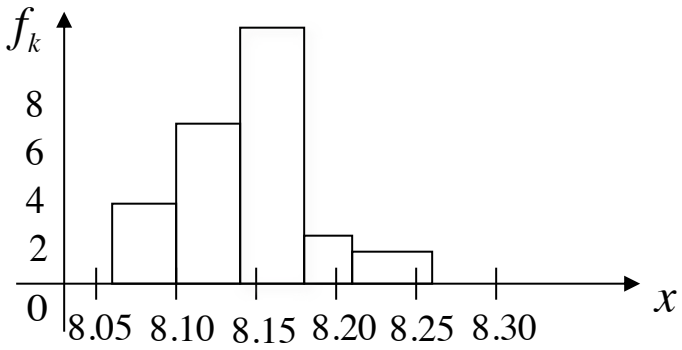
$f_k = \frac{F_k}{\Delta x_k}$ **Πυκνότητα Πιθανότητας** (probability density) μιας μεταβλητής x να βρεθεί στο *k*-διάστημα Δx

το ποσοστό δηλαδή των μετρήσεων στο διάστημα Δx_k

Ιστογράμματα

Το ιστόγραμμα αποτελεί τη γραφική παράσταση της f_k ως προς το x

Η ολική επιφάνεια των ορθογωνίων θα είναι ίση με την μονάδα: $\sum F_k = 1 = \sum f_k \Delta x_k$



Στην περίπτωση αυτή χρησιμοποιούμε Δx = 0.04 cm

Αν ο αριθμός των μετρήσεων γίνει αρκετά μεγάλος τότε το ιστόγραμμα παίρνει τη μορφή της κατανομής από την οποία προέρχονται οι μετρήσεις.

Η καμπύλη αποτελεί την οριακή κατανομή.

Κατασκευή ιστογράμματος με matplotlib

Για να κατασκευάσουμε ένα ιστόγραμμα στην python με την matplotlib, χρειάζεται να δώσουμε την εντολή:

***content*, *binvalues*, *interm* = *plt.hist*(x, bins=τιμή, range=(τιμές))**

- **Τα ορίσματα της συνάρτησης *hist*:**

***x*:** *array* [*data*] ή *list* από *arrays* [*data1, data2,...*] με τις τιμές της μεταβλητής που θέλουμε να κάνουμε το γράφημα (όπως και στην *plt.plot*)

***bins*:** ακέραιος αριθμός που δηλώνει τον αριθμό των υποδιαστημάτων ίσου εύρους στα οποία είναι χωρισμένος ο *x*-άξονας

***range** [optional]*: *tuple* που περιέχει την τιμή του *xmin* και *xmax.*

Αν δεν προσδιοριστεί, τότε αυτόματα ως *range*=(*x.min*(), *x.max*()) λαμβάνεται η ελάχιστη και μέγιστη τιμή των στοιχείων του array *x.*

Κατασκευή ιστογράμματος με matplotlib

***content*, *binvalues*, *interm* = *plt.hist*(x, bins=τιμή, range=(τιμές))**

- **Τι επιστρέφει η συνάρτηση *hist*:** ένα tuple με τα ακόλουθα στοιχεία

content: array μεγέθους N=bins με στοιχεία τη συχνότητα εμφάνισης των δεδομένων που δόθηκαν για κάθε bin. Αν δόθηκε *list* με *arrays* τότε το *content* περιέχει μια *list* με *arrays* με τη συχνότητα εμφάνισης για κάθε ξεχωριστό *array*.

binvalues: array μεγέθους N=bins+1 με τις τιμές του *x*-άξονα που αντιστοιχούν στα υποδιαστήματα που ορίστηκαν. Ο πίνακας περιέχει την αριστερή τιμή του κάθε υποδιαστήματος και την δεξιά τιμή του τελευταίου υποδιαστήματος.

interm: Μια λίστα από ενδιάμεσους υπολογισμούς που χρησιμοποιηθήκαν στην κατασκευή του ιστογράμματος

Ιστογράμματα – Επιπλέον επιλογές για ορίσματα εισόδου

facecolor *[optional]* = *'b'* το χρώμα του ιστογράμματος (bleu)

density *[optional]* = *True/None[default]*

True: επιστρέφει την πυκνότητα πιθανότητας για κάθε bin: $\frac{N_i}{N_{tot}\Delta x}$.

Το εμβαδό της επιφάνειας κάτω από το ιστόγραμμα (το ολοκλήρωμα δηλαδή) θα ισούται με 1

Η επιλογή *Density* είναι παρόμοια με την επιλογή ***Normed*** η οποία επιστρέφει απλά ως αποτέλεσμα N_i/N_{tot}

histtype: {*'bar'*,*'barstacked'*,*'step'*,*'stepfilled'*} *[optional]*

bar: ραβδόμορφο ιστόγραμμα

barstacked: ραβδόμορφα ιστογράμματα το ένα πάνω στο άλλο
αν δοθεί list από data arrays ως x

step: δημιουργεί γράφημα ως καμπύλη και η περιοχή στο εσωτερικό δεν χρωματίζεται

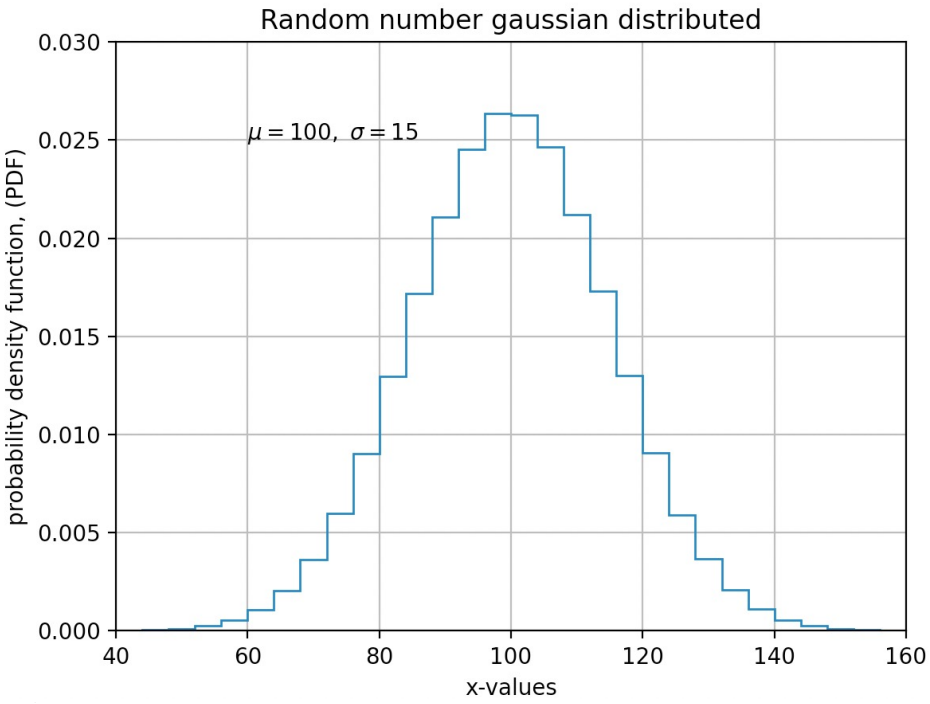
stepfilled: δημιουργεί γράφημα ως καμπύλη και η περιοχή στο εσωτερικό χρωματίζεται

Παράδειγμα

```
import numpy as np
import matplotlib.pyplot as plt
from random import random, seed, gauss
seed(12345678)
mu, sigma = 100, 15
x=[]
‘’’
```

Δημιουργία 1Μ τυχαίων αριθμών κατανεμημένους σύμφωνα με την Gauss κατανομή με μέση τιμή μ = 100 και σ = 15

```
’’’
for i in range(1000000): #
    x.append( gauss(mu,sigma ))
# Δημιουργία ιστογράμματος με 50 bins στο x [0-200].
# Δx = 4 = (xmax-xmin)/bins
cont,binval,intr=plt.hist(x, bins=50, range=(0.,200.),
                          density=True, histtype=‘step’)
plt.xlabel(‘x-values’)
plt.ylabel(‘probability density function, (PDF)’)
plt.title(‘Random number gaussian distributed”)
plt.text(60, 0.025, r’$\mu=100, \ \sigma = 15$’)
plt.xlim(40,160)
plt.ylim(0., 0.03)
plt.grid(True)
plt.show()
```



Ιστογράμματα - Οριακή κατανομή

Η οριακή κατανομή είναι μια θεωρητική καμπύλη που δεν μπορεί να προκύψει ποτέ από τις μετρήσεις.

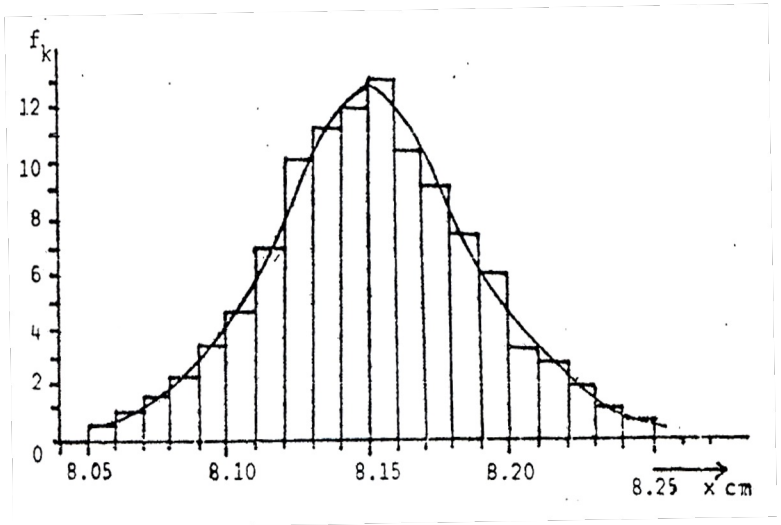
Το εμβαδό κάθε παρ/μου του ιστογράμματος είναι ισοδύναμο με τον αριθμό μετρήσεων που περιλαμβάνονται στο διάστημα αυτό.

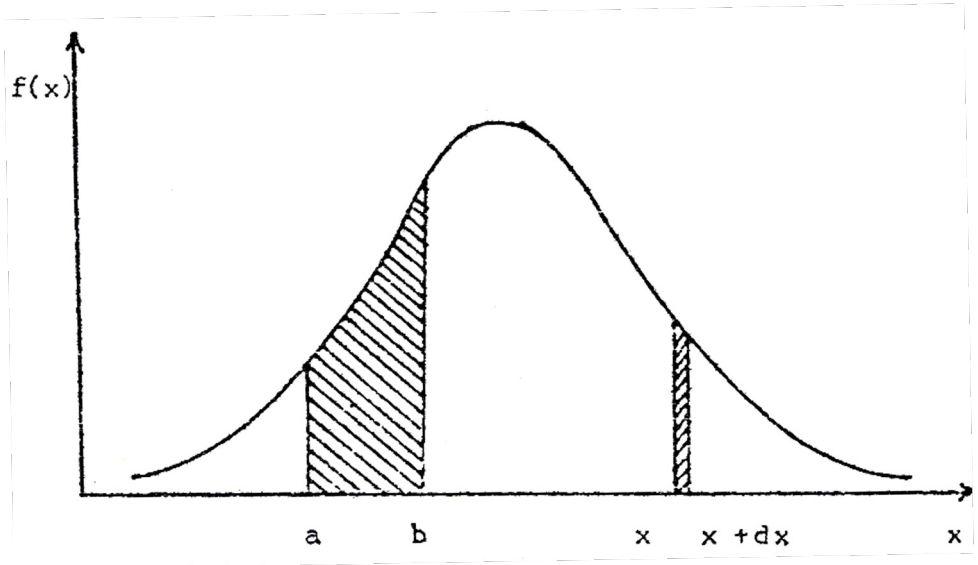
$f(x)dx$ πιθανότητα μια μέτρηση να περιλαμβάνεται μεταξύ x και x+Δx

$\int_a^b f(x)dx$ ποσοστό μετρήσεων μεταξύ x=α και x=b, η πιθανότητα δηλαδή μια μέτρηση να είναι στο διάστημα α,b

$\int_{-\infty}^{+\infty} f(x)dx = 1$ Η πιθανότητα μια μέτρηση να βρίσκεται μεταξύ -∞ και +∞

Λέμε τότε ότι η $f(x)$ είναι κανονικοποιημένη





Ιστογράμματα - Οριακή κατανομή πολλών μετρήσεων

Αν ξέρουμε την οριακή κατανομή μπορούμε να υπολογίσουμε τη μέση τιμή

$$\bar{x} = \sum x_k F_k \xrightarrow[\lim \Delta x_k \to 0]{} \bar{x} = \int_{-\infty}^{+\infty} x f(x)\, dx \quad \text{και ανάλογα} \quad \sigma_x^2 = \int_{-\infty}^{+\infty} (x - \bar{x})^2 f(x)\, dx$$

Αν οι μέτρησεις επηρεάζονται από πολλές πηγές μικρών τυχαίων σφαλμάτων τότε η οριακή κατανομή προσεγγιζει αυτή της κατανομής Gauss

Για πολύ μεγάλο αριθμό μετρήσεων και για τυχαία σφάλματα οι τιμές που αποκλίνουν πολύ από τη μέση τιμή κατανέμονται συμμετρικά ως προς τη μέση τιμή, τότε στην άθροιση της εύρεσης της μέσης τιμής, αυτές οι τιμές αλληλοαναιρούνται και η τελική τιμή πλησιάζει την πιστή τιμή του μεγέθους

$$f(x) = G(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2} \Rightarrow \int_{-\infty}^{+\infty} G(x)\, dx = 1$$

Η πιθανότητα να πάρουμε μια τιμή x στο διάστημα [x_1, x_1+dx) είναι: $P(x_1) \propto \sigma^{-1} e^{-(x_1-\mu)^2/2\sigma^2}$

Η πιθανότητα να πάρουμε μια τιμή x στο διάστημα [x_2,x_2+dx) είναι: $P(x_2) \propto \sigma^{-1} e^{-(x_2-\mu)^2/2\sigma^2}$

Αν συνεχίσουμε για όλες τις τιμές x_N τότε η πιθανότητα για την x_N: $P(x_N) \propto \sigma^{-1} e^{-(x_N-\mu)^2/2\sigma^2}$

Μέση τιμή ως η καλύτερη εκτίμηση της πραγματικής τιμής

Η πιθανότητα να παρατηρήσουμε το συγκεκριμένο δείγμα των N μετρήσεων είναι το γινόμενο των πιθανοτήτων:

$$P(x_1,x_2,\cdots,x_N)=P(x_1)\times P(x_1)\times\cdots\times P(x_N)\ \Rightarrow P(x_1,x_2,\cdots,x_N)\propto\frac{1}{\sigma^{\mathrm{N}}}e^{-\sum(x_i-\mu)^2/2\sigma^2}$$

Δεδομένων N παρατηρούμενων τιμών x_1, x_2, …, x_N η καλύτερη εκτίμηση της πραγματικής τιμής είναι αυτή που παίρνουμε αν μεγιστοποιήσουμε την πιθανότητα

Αυτό θα συμβεί όταν το άθροισμα στον εκθέτη είναι ελάχιστο $\sum(x_i-\mu)^2/2\sigma^2$

Επομένως θα πρέπει: $$\frac{d\left(\sum(x_i-\mu)^2/2\sigma^2\right)}{d\mu}=0\Rightarrow\sum(x_i-\mu)/\sigma^2=0\Rightarrow\mu=\frac{\sum_{i=1}^{N}x_i}{N}$$

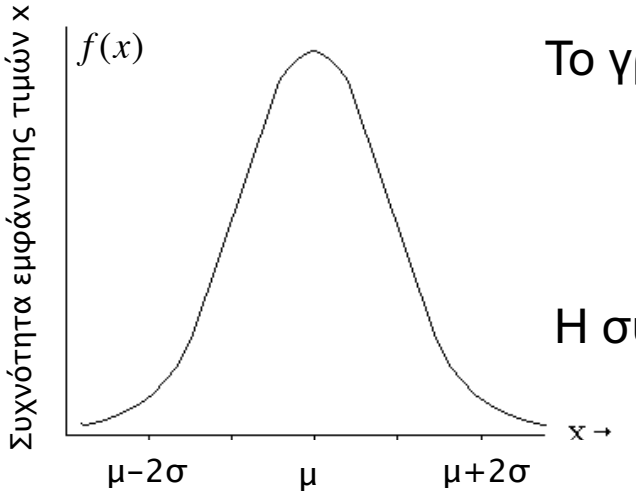
Ανάλογα ο καλύτερος υπολογισμός της τυπικής απόκλισης βρίσκεται ότι είναι:

$$\frac{dP(x_1,x_2,\cdots,x_N)}{d\sigma}=0\Rightarrow\sigma=\sqrt{\frac{1}{\mathrm{N}}\sum_{i=1}^{N}\left(x_i-\bar{x}\right)^2}$$

Gaussian κατανομή

Τυχαία σφάλματα υπάρχουν ακόμα και όταν όλες οι πηγές άλλων σφαλμάτων έχουν εξαλειφθεί. Συχνά, χρειάζεται να επαναλάβουμε μετρήσεις αρκετές φορές ώστε να εκτιμήσουμε την πιθανότητα κάποιος που επαναλαμβάνει μια πανομοιότυπη μέτρηση θα βρει διαφορετικό αποτέλεσμα.

Οι εκτιμήσεις αυτές στηρίζονται σε κάποιο μαθηματικό μοντέλο το οποίο και περιλαμβάνει τις περισσότερες περιπτώσεις. Το μοντέλο αυτό περιγράφεται από την Gaussian ή κανονική (normal) κατανομή.



Το γράφημα του σχήματος δίνεται από τη συνάρτηση:

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma}\, e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

Η συνάρτηση περιγράφεται από δύο παραμέτρους, μ και σ

Η μεταβλητή μ περιγράφει που βρίσκεται η κορυφή της καμπύλης στον $x-$ άξονα.

Η μεταβλητή σ περιγράφει τη διασπορά των τιμών του x, δηλαδή το εύρος της καμπύλης.

Ουσιαστικά η καμπύλη αυτή δείχνει τη συχνότητα με την οποία θα εμφανίζονταν διάφορες τιμές του x αν επαναλαμβάναμε μια μέτρηση πολλές φορές.

Κανονική κατανομή

Η μέση τιμή, μ, δίνει ουσιαστικά την περισσότερη πιθανή τιμή των x και αυτή είναι η μέση τιμή. Παραπλήσιες τιμές έχουν μια μικρότερη αλλά αρκετά μεγάλη πιθανότητα.

Η κατανομή εκτείνεται σε άπειρες τιμές του x (με μικρή ωστόσο πιθανότητα). Αποδεικνύεται ότι το εμβαδό της καμπύλης μεταξύ μ-σ και μ+σ ισούται με το 68.3% του συνολικού εμβαδού της καμπύλης [-∞,+∞].

Το εμβαδό της καμπύλης που περικλύεται μεταξύ μ-2σ και μ+2σ ισούται με το 95.4% του εμβαδού της καμπύλης. Ενώ για $\pm 3\sigma$ το ποσοστό είναι 99.7%

Όπως γράψαμε την παραπάνω εξίσωση το ολοκλήρωμά της από -∞,+∞

$$\int_{-\infty}^{+\infty} G(x)\,dx = \frac{1}{\sqrt{2\pi}\sigma}\int_{-\infty}^{+\infty} e^{-\left(\frac{x-\mu}{2\sigma}\right)^2} dx = 1$$

$$\hat{\mu} = \bar{x} = \frac{x_1 + x_2 + \cdots + x_N}{N}$$

$$\hat{\sigma}^2 = \frac{1}{\mathrm{N}-1}\sum_{i=1}^{N}\left(\hat{\mu} - x_i\right)^2$$

Για Ν μεγάλο $\hat{\mu} = \mu$ $\hat{\sigma} = \sigma$

Πολλές φορές χρησιμοποιούμε αντί για την ποσότητα σ, το πλήρες εύρος της κατανομής στο μέσο του μέγιστου ύψους της (FWHM) που συμβολίζεται με Γ.

Από τον ορισμό: $G\left(\mu \pm \frac{\Gamma}{2}\right) = \frac{1}{2}G(\mu)$ έχουμε ότι $\Gamma = 2\sigma\sqrt{2\ln 2} \approx 2.35\sigma$

Αριθμητικές λύσεις εξισώσεων

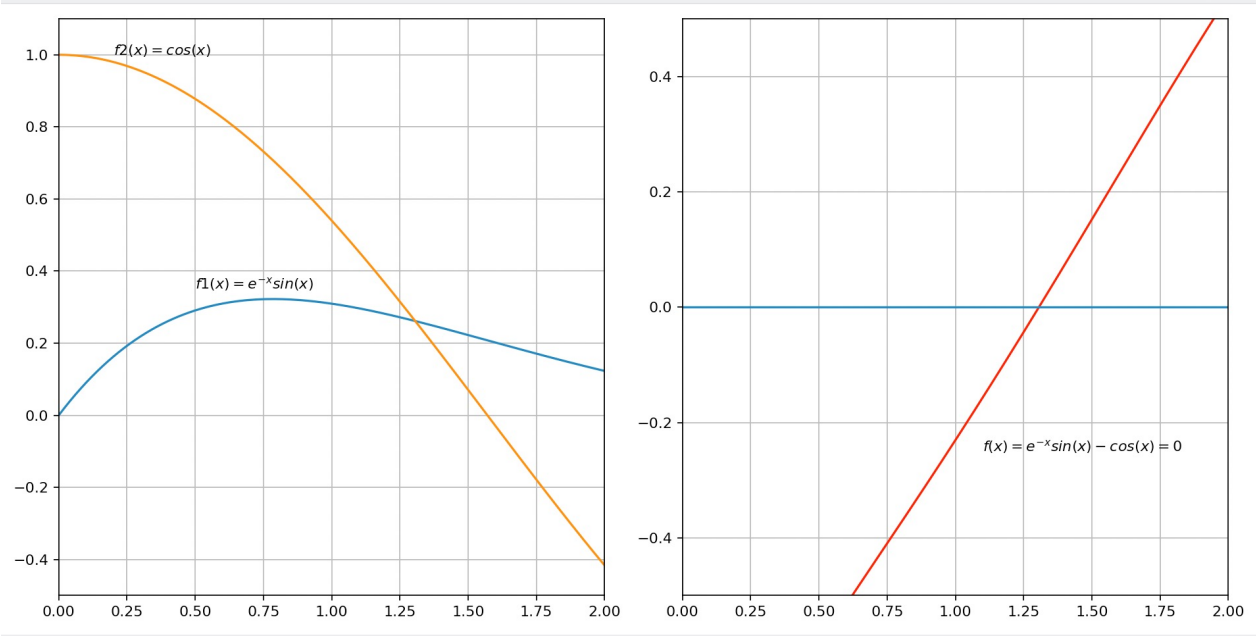
Επίλυση μή γραμμικών εξισώσεων

Στα περισσότερα προβλήματα το αποτέλεσμα στο οποίο καταλήγουμε δεν είναι κάποια δευτεροβάθμια ή τριτοβάθμια εξίσωση την οποία μπορούμε να λύσουμε εύκολα αναλυτικά

Στις περιπτώσεις αυτές αν αλλάξουμε την παραδοσιακή προσέγγιση εύρεσης ακριβούς λύσης με αυτή μιας προσεγγιστικής λύσης, τότε ανοίγονται ποικιλία δυνατοτήτων για εύρεση λύσης εξίσωσης

Για παράδειγμα, έστω ότι έχουμε την εξίσωση της μορφής: $\cos(x) = e^{-x}\sin(x)$

Μεταφέρουμε όλους τους όρους στην δεξιά πλευρά και έχουμε: $f(x) = e^{-x}\sin(x) - \cos(x)$

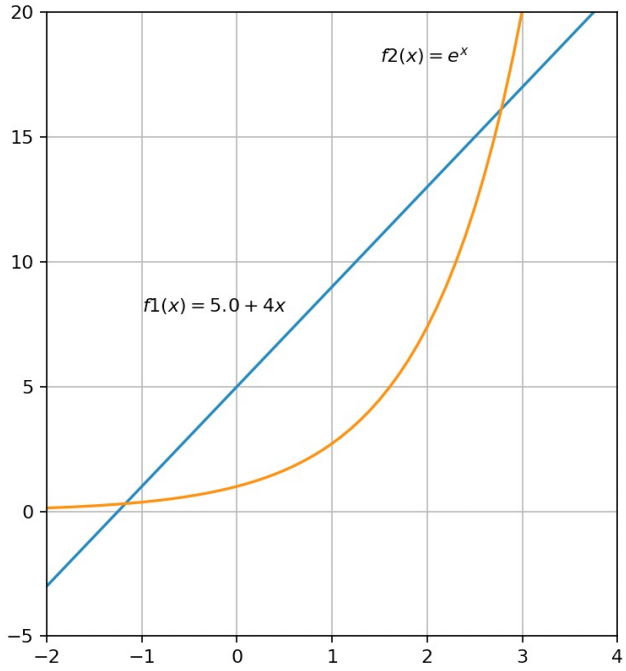


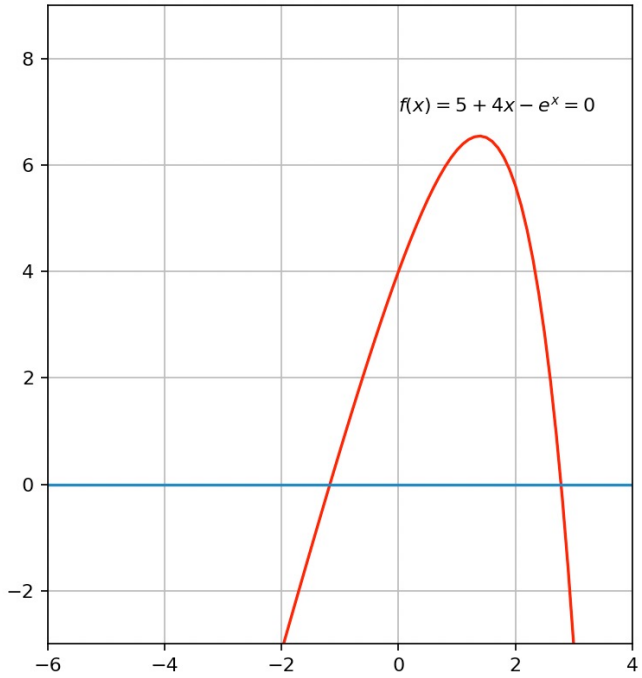
Πότε χρειάζεται εύρεση μιας τέτοιας λύσης: όταν χρησιμοποιούνται αριθμητικές μέθοδοι για λύση μιας διαφορικής εξίσωσης και όταν προσπαθούμε να βελτιστοποιήσουμε μια μέθοδο, π.χ. όταν προσπαθούμε να βρούμε μέγιστο ή ελάχιστο μιας συνάρτησης f, οπότε $f' = 0$

Επίλυση μή γραμμικών εξισώσεων

Για παράδειγμα, έστω ότι έχουμε μια εξίσωση της μορφής: $5 + 4x = e^x$

Η εξίσωση αυτή δεν μπορεί να λυθεί αναλυτικά





Μπορείτε να βρείτε το py file που κάνει τα δύο γραφήματα στο:
http://ptohos.github.io/phy140/Lectures/lect14_graph.py

Προσεγγιστικές μέθοδοι εύρεσης ριζών μιας εξίσωσης

Μέθοδος πολλαπλών προσπαθειών – Brute force

Η αναπαράσταση μιας μαθηματικής συνάρτησης σε έναν υπολογιστή παίρνει δύο μορφές. Η μία αποτελεί μια Python συνάρτηση η οποία επιστρέφει τιμές της συνάρτησης δεδομένης κάποιας τιμής του ορίσματος και η δεύτερη είναι μια συλλογή σημείων $(x, f(x))$ κατά μήκος της καμπύλης της συνάρτησης.

Η δεύτερη μορφή είναι αυτή που χρησιμοποιούμε για να κάνουμε το γράφημα της συνάρτησης υποθέτοντας γραμμική μεταβολή μεταξύ των σημείων που χρησιμοποιούνται

Η μορφή αυτή είναι επίσης κατάλληλη για επίλυση εξίσωσης αλλά και για βελτιστοποίηση:

- ακολουθούμε την καμπύλη της συνάρτησης και σε κάποιο σημείο θα περάσει τον x-άξονα
- ή για βελτιστοποίηση, χρειάζεται να βρούμε τοπικό μέγιστο ή ελάχιστο.

Η μέθοδος είναι εξαιρετικά απλή αλλά χρειάζεται να εξετάσουμε ένα πολύ μεγάλο αριθμό σημείων

Θέλουμε να λύσουμε την $f(x) = 0$, να βρούμε δηλαδή τα σημεία *x* στα οποία η *f* περνά από τον *x*-άξονα. Η μέθοδος πολλαπλών προσπαθειών έγγυται στον έλεγχο πολλών σημείων και την εύρεση των διαδοχικών σημείων για τα οποία η τιμή της συνάρτησης είναι πάνω και κάτω από τον *x*-άξονα ή αντίστροφα. Αν ισχύει η συνθήκη αυτή τότε ξέρουμε ότι η συνάρτηση $f(x)$ μηδενίζεται ανάμεσα στα δύο σημεία

Μέθοδος πολλαπλών προσπαθειών – Brute force

Έχουμε ένα σύνολο από *n*+1 σημεία (x_i, y_i), όπου $y_i = f(x_i), i = 0,1,2,\cdots,n$ και $x_0 < \cdots < x_n$

Ελέγχουμε αν $y_i < 0$ και $y_{i+1} > 0$ ή το αντίστροφο ($y_i > 0$ και $y_{i+1} < 0$)

Ο περισσότερο ενδεδειγμένος έλεγχος ωστόσο είναι: $y_i * y_{i+1} < 0$

Αν ισχύει, τότε η ρίζα της $f(x) = 0$ βρίσκεται στο διάστημα $[x_i, x_{i+1}]$

Θεωρώντας γραμμική μεταβολή της $f(x)$ στο διάστημα $[x_i, x_{i+1}]$, μπορούμε να θεωρήσουμε την προσέγγιση: $f(x) \approx \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}(x - x_i) + f(x_i) = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}(x - x_i) + y_i$

Θέτοντας $f(x) = 0$ έχουμε από την προηγούμενη εξίσωση τη λύση: $x = x_i - \frac{x_{i+1} - x_i}{y_{i+1} - y_i} y_i$

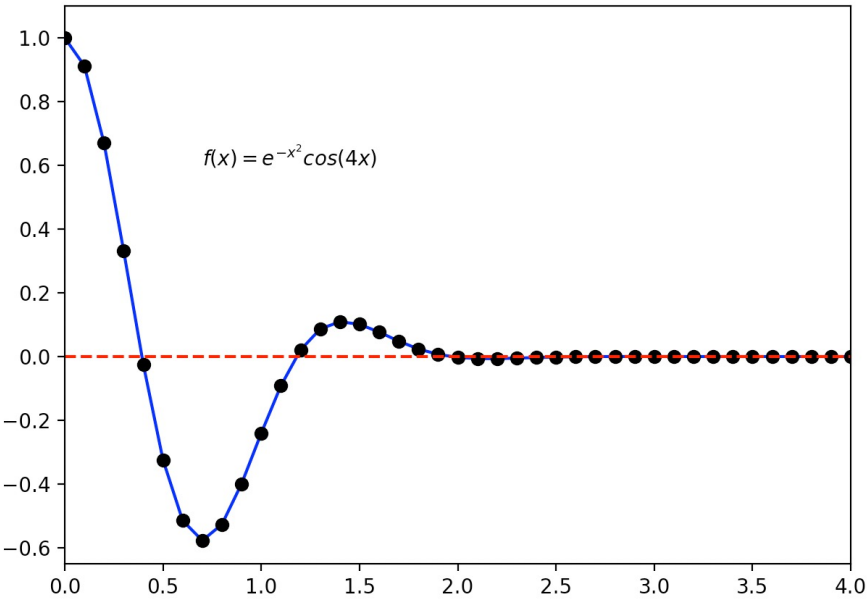
```
import numpy as np
x=[x*0.004 for x in range(1001)]
y=f(x)              # tyxaia synartisi
root = None
for i in range(len(x)-1):
    if y[i]*y[i+1]< 0 :
       root = x[i]-(x[i+1]-x[i])/(y[i+1]-y[i])*y[i]
       break       # eksw apo to loop
if root is None:
   print('No root is found in [%g,%g]'%(x[0],x[-1]))
else:
   print('The first root is x =%g'%root)
```

Εφαρμογή της μεθόδου

Μέθοδος πολλαπλών προσπαθειών – Brute force

Εφαρμογή: $f(x) = e^{-x^2}\cos(4x)$

Η ρίζα της εξίσωσης αυτής είναι $x = \frac{\pi}{8}$.



Το προηγούμενο πρόγραμμα επιστρέφει ως ρίζα την τιμή x =0.392701 που αντιστοιχεί σε σφάλμα 1.84×10^{-6}.

Μετατροπή του αλγόριθμου σε Python συνάρτηση για τη χρήση της σε μεγάλο αριθμό προβλημάτων. Η συνάρτηση θα πρέπει να δέχεται την συνάρτηση *f* και το διάστημα [a,b] σαν παραμέτρους εισδοχής καθώς επίσης τον αριθμό των υποδιαστημάτων *n*.

Η συνάρτηση επιστρέφει μια λίστα με τις ρίζες της εξίσωσης στο διάστημα [a,b].

```
def brute_force_root_finder(f, a, b, n):
    x = [a+x*(b-a)/n for x in range(n+1)]
    y = list(map(f,x))
    roots = []
    for i in range(n-1):
        if y[i]*y[i+1] < 0:
            root = x[i] - (x[i+1] - x[i])/(y[i+1] - y[i])*y[i]
            roots.append(root)
    return roots
```

Μέθοδος πολλαπλών προσπαθειών – Brute force

Εφαρμογή: $f(x) = e^{-x^2}\cos(4x)$

```
def f(x):
     return exp(-x**2)*cos(4*x)

def demo():
     roots = brute_force_root_finder(f, 0, 4, 1001)
     if roots:
         print(roots)
     else:
         print('Could not find any roots')
```

Μπορείτε να κατεβάσετε το πρόγραμμα αυτό από:
https://ptohos.github.io/phy140/Lectures/brute_force_finder.py

Βελτιστοποίηση – Brute force method

Η χρήση της brute force μεθόδου για την εύρεση ακρότατου μιας συνάρτησης, σημαίνει ότι βρίσκουμε μέγιστο ή ελάχιστο από ένα σύνολο σημείων κατά μήκος της καμπύλης της συνάρτησης.

Ένα σημείο x_i θα αντιστοιχεί σε μέγιστο της συνάρτησης αν $y_{i-1} < y_i > y_{i+1}$.

Παρόμοια, ένα σημείο x_i θα αντιστοιχεί σε ελάχιστο της συνάρτησης αν $y_{i-1} > y_i < y_{i+1}$.

Εφαρμόζουμε τον έλεγχο αυτό για όλα τα εσωτερικά σημεία $i = 1,2,\cdots,n-1$ για την εύρεση τοπικού ελάχιστου ή τοπικού μέγιστου. Χρειαζόμαστε ένα ακραίο σημείο, $i = 0$ ή $i = n$ για να ελέγξουμε αν το x_i είναι ολικό μέγιστο ή ελάχιστο

Βελτιστοποίηση – Brute force method

```
def brute_force_optimizer(f, a, b, n):
    x = [a+x*(b-a)/n for x in range(n+1)]
    y = list(map(f,x))
    minima = []
    maxima = []
    for i in range(n-1):
        if y[i-1] < y[i] > y[i+1]:
            maxima.append(i)
        if y[i-1] > y[i] < y[i+1]:
            minima.append(i)
   # What about the end points?
    y_max_inner = max([y[i] for i in maxima])
    y_min_inner = min([y[i] for i in minima])
    if y[0] > y_max_inner:
        maxima.append(0)
    if y[len(x)-1] > y_max_inner:
        maxima.append(len(x)-1)
    if y[0] < y_min_inner:
        minima.append(0)
    if y[len(x)-1] < y_min_inner:
        minima.append(len(x)-1)
    # Return x and y values
        return [(x[i], y[i]) for i in minima], \
               [(x[i], y[i]) for i in maxima]
```

Μπορείτε να κατεβάσετε το πρόγραμμα αυτό από:
https://ptohos.github.io/phy140/Lectures/brute_force_optimizer.py

Συμβολικοί υπολογισμοί στην Python - Sympy

Symbolic computations

Όπως αναφέρει η λέξη, μπορούμε να χρησιμοποιήσουμε διεργασίες μεταξύ συμβόλων (ουσιαστικά αναλυτικές ή ακριβείς διεργασίες). Στις περιπτώσεις αυτές οι υπολογισμοί γίνονται μεταξύ συμβόλων αντί των αριθμητικών σταθερών που αντιπροσωπεύουν

```
x = 2
y = 3
z = x * y
print(z)
```

Αριθμητικοί υπολογισμοί

output
6

```
from sympy import *
x,y = symbols('x y')
z = x * y
print(z)
```

Υπολογισμός με σύμβολα

output
x*y

Υπολογισμοί με σύμβολα γίνονται με τη χρήση του module **sympy** της Python

*** Αν δεν υπάρχει στο centos περιβάλλον του virtual box μπορείτε να το κάνετε εύκολα install δίνοντας την εντολή: *pip3 install sympy* (θα δώσετε το root password)

Κάθε σύμβολο αναπαρίσταται με κάποια μεταβλητή, αλλά το όνομα του συμβόλου θα πρέπει να δηλωθεί με **Symbol ('name') (κεφαλαίο S)** για την περίπτωση ενός συμβόλου ή **symbols('name1 name2 … nameN') (μικρό s) για N symbols**

Symbolic computations

Μερικές από τις βασικές πράξεις μεταξύ συμβόλων δίνονται στο παρακάτω παράδειγμα:

```
from sympy import *
x = Symbol('x')
y = Symbol('y')
print(2*x - 3*x – y)                    # Αλγεβρικοί υπολογισμοί
-x - y
print(diff(x**2,x))                     # Παραγώγιση ως προς x
2*x
print(integrate(cos(x),x))              # Ολοκλήρωση ως προς x
sin(x)
print(simplify((x**2 + x**3)/x**2))     # Παραγοντοποίηση
x+1
print(limit(sin(x)/x, x, 0))            # Εύρεση ορίου
1
print(solve(5*x – 15, x))               # Λύση εξίσωσης
[3]
```

Μπορούμε να έχουμε υπολογισμούς όπως ανάπτυγμα Taylor, γραμμική άλγεβρα (πράξεις με πίνακες ή διανύσματα) αλλά και μερικές περιπτώσεις επίλυσης διαφορικών εξισώσεων

Symbolic computations

Παράδειγμα αναπτύγματος Taylor:

series(f, x, x0=value, n=N)

διαφορετικά: *f = expression(x)*

f.series(x,x0=value, n=N)

***f**: η συνάρτηση που θέλουμε να αναπτύξουμε*

***x**: η μεταβλητή ως προς την οποία αναπτύσσεται*

***x0**: η τιμή για τον υπολογισμό του αναπτύγματος*

***n**: Ο αριθμός των όρων του αναπτύγματος*

```
from sympy import cos, lambdify
from sympy.abc import x, y    #χρήση όλων των λατινικών και
                               #ελληνικών γραμμάτων ως symbols
cos(x).series(n=4)             #  4 πρώτοι όροι του αναπτύγματος
 1-x**2/2+x**4/24+O(x**6)      #  Taylor του cos(x) ως προς x0=0

e=cos(x)
e.series(x,x0=1,n=2)           #  Ανάπτυξη ως προς x0=1, για 2 όρους
 cos(1)-(x-1)*sin(1)+O((x-1)**2,(x,1))

z=e.series(n=4).removeO()      #  Ανάθεση του αναπτύγματος στη z
                               #  αφού αφαιρεθεί το τμήμα O(x**6)
f=lambdify(x,z, 'numpy')       #  μετατροπή σε κανονική συνάρτηση
```