Τυχαίοι Αριθμοί

και Monte Carlo

Monte Carlo και τυχαίοι αριθμοί

Τα προσδιορισμένα συστήματα (deterministic systems) περιγράφονται εν γένει από κάποιο μαθηματικό κανόνα

Κάποια συστήματα ωστόσο δεν είναι προσδιορισμένα Τυχαία ή στοχαστικά

Οποιαδήποτε διεργασία ή αλγόριθμος χρησιμοποιεί τυχαίους αριθμούς και αντιτίθεται σε προσδιορισμένους αλγόριθμους ονομάζεται Monte Carlo

Η μέθοδος Monte Carlo χρησιμοποιείται ευρέως στις επιστήμες:

Φυσική: προσομοίωση φυσικών διεργασιών

Μαθηματικά: αριθμητική ανάλυση

Βιολογία: προσομοίωση κυττάρων

Οικονομικά: εκτίμηση της διακύμανσης του χρηματιστηρίου αξιών

Μηχανική: προσομοίωση πειραματικών διατάξεων

Σημασία των μεθόδων Monte Carlo

Οι μέθοδοι Monte Carlo αποτελούν ένα από τα σημαντικότερα εργαλεία στη Φυσική

- Ανάλυση δεδομένων
- Προσομοίωση φυσικών γεγονότων που στηρίζονται σε τυχαίες διεργασίες - πιθανότητες
- Σχεδιασμό ανιχνευτών, βελτιστοποίηση και προσομοίωση

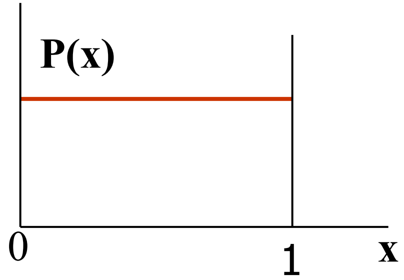
Επομένως ας μάθουμε μερικές από τις βασικές αρχές

- Σκοπός των γεννητόρων/προγραμμάτων Monte Carlo
- Γεννήτορες τυχαίων αριθμών
- Ολοκλήρωση
- Μερικά ιδιαίτερα δημοφιλή Monte Carlo προγράμματα

Τυχαίοι αριθμοί

Τυχαίος αριθμός είναι ένας αριθμός επιλεγμένος σαν να ήταν καθαρά τυχαία από μια συγκεκριμένη κατανομή

Σε μια ομοιόμορφη κατανομή τυχαίων αριθμών στο διάστημα [0,1), κάθε αριθμός έχει την ίδια τύχη να επιλεχθεί



Για παράδειγμα: Όταν ρίχνετε ένα ζάρι οι αριθμοί που μπορείτε να πάρετε είναι ομοιόμορφα κατανεμημένοι μεταξύ 1 και 6.

Κάθε αριθμός έχει την ίδια πιθανότητα να ''βγεί''

Γεννήτορες τυχαίων αριθμών

Ο καλύτερος τρόπος για να πάρουμε τυχαίους αριθμούς είναι να χρησιμοποιήσουμε μια διεργασία που συμβαίνει στη φύση.

- Ρίξιμο ενός ζαριού ή ενός νομίσματος
- Λότто
- Τα αποτελέσματα του ποδοσφαίρου
- Η ραδιενεργός διάσπαση των πυρήνων

Φυσικά ο τρόπος αυτός για να διαλέξουμε τυχαίους αριθμούς δεν είναι ιδιαίτερα αποδοτικός

- Υπολογιστικές μέθοδοι αναπτύχθηκαν που κάνουν την ίδια διαδικασία

Πως μπορούμε όμως να κάνουμε κάποιο πρόγραμμα να υπολογίζει κάτι τυχαία;

- Με ένα γεννήτορα τυχαίων αριθμών

Γεννήτορες τυχαίων αριθμών

Γεννήτορας τυχαίων αριθμών είναι μία συνάρτηση η οποία δημιουργεί μια ακολουθία τυχαίων αριθμών

Όλοι οι υπολογιστές σήμερα περιέχουν στην βιβλιοθήκη τους ένα μηχανισμό για την δημιουργία ακολουθίας τυχαίων αριθμών οι οποίοι είναι ομοιόμορφα κατανεμημένοι στο διάστημα [0,1)

Η ακολουθία των αριθμών αυτών μπορεί να θεωρηθεί σαν ψευδο-τυχαία ακολουθία αφού για κάθε εκτέλεση του προγράμματος, θα πάρουμε και πάλι την ίδια ακολουθία τυχαίων αριθμών για την ίδια αρχική τιμή του ‘‘σπόρου’’ (seed) της ακολουθίας

Στην πραγματικότητα οι συναρτήσεις που καλούμε στον υπολογιστή χρησιμοποιούν μια μέθοδο (Lehmer 1948) στηριγμένη σε 32-bit ακεραίους και επομένως έχουν περίοδο το πολύ 2^{31}~10^9.

Αυτό είναι το πλήθος των τυχαίων αριθμών που μπορούν να δημιουργηθούν μέσα σε λίγα δευτερόλεπτα σε ένα μοντέρνο υπολογιστή

Η μέθοδος που ακολουθείται χρησιμοποιεί μια εξίσωση της μορφής:

$$x_{n+1} = \mathrm{mod}\left[\left(ax_n + b\right), m\right]$$

όπου mod είναι το modulo. Οι σταθερές α,b και m διαλέγονται προσεκτικά ώστε η ακολουθία των αριθμών να γίνεται χαοτική και ομοιόμορφα κατανεμημένη

Γεννήτορες τυχαίων αριθμών

$$x_{n+1} = \mathrm{mod}\left[\left(ax_n + b\right), m\right]$$

Κανόνες

- Η πρώτη αρχική τιμή, x_0, (seed) επιλέγεται
- Η τιμή του m> x_0 και $a, b \geq 0$
- Το εύρος των τιμών είναι μεταξύ 0 και m
  (διαιρώντας με m μετατρέπεται μεταξύ 0 και 1)
- Η περίοδος του γεννήτορα αυτού είναι m-1

  Το m πρέπει να είναι αρκετά μεγάλο αφού η περίοδος δεν μπορεί ποτέ να γίνει μεγαλύτερη από m.

Για παράδειγμα:

Ας διαλέξουμε α=b=x_0=7 και m = 10 και από την σχέση: x_i =mod(a*x_{i-1}+b,m)

η ακολουθία ψευδο-τυχαίων αριθμών που θα πάρουμε θα είναι:

7, 6, 9, 0 7, 6, 9, 0 7, 6, 9, 0

Και διαιρώντας με 10 θα έχουμε την ακολουθία

0.7, 0.6, 0.9, 0.0 0.7, 0.6, 0.9, 0.0 0.7, 0.6, 0.9, 0.0 Πολύ κακή ακολουθία

Γεννήτορες τυχαίων αριθμών

Από τους πλέον δημοφιλής γεννήτορες είναι ο **RANDU** ο οποίος αναπτύχθηκε από την IBM το 1960 με τον ακόλουθο αλγόριθμο:

$$x_{n+1} = \mathrm{mod}\left(65069 x_n, 2^{31} - 1\right)$$

και αργότερα οι Park και Miller πρότειναν πως η εξίσωση $x_{n+1} = \mathrm{mod}\left(16807 x_n, 2^{31} - 1\right)$ δίνει την ελάχιστη συνθήκη για ένα ικανοποιητικό γεννήτορα

Τυχαίοι αριθμοί στην Python

Στην PYTHON μπορούμε να πάρουμε τυχαίους αριθμούς από την βιβλιοθήκη **random**

from random import randrange, seed, random

```
seed(42)
for i in range(4):
    print(randrange(10))
```

print(randrange(10))	Τυπώνει ένα τυχαίο ακέραιο αριθμό από το 0, 9
random()	Τυπώνει έναν τυχαίο δεκαδικό τυχαίο αριθμό sto [0,1)
randrange(n)	Τυπώνει ένα τυχαίο ακέραιο αριθμό από το 0, n-1
randrange(m,n)	Τυπώνει ένα τυχαίο ακέραιο αριθμό από το m, n-1
randrange(m,n,k)	Τυπώνει ένα τυχαίο ακέραιο αριθμό από το m, n-1 με βηματα k-1

Οι συναρτήσεις αυτές δίνουν έναν νέο τυχαίο αριθμό κάθε φορά που καλούνται

MONTE CARLO ΒΕΛΤΙΣΤΟΠΟΙΗΣΗ

ΕΥΡΕΣΗ ΑΚΡΟΤΑΤΩΝ ΣΥΝΑΡΤΗΣΗΣ

Monte Carlo βελτιστοποίηση

Μπορούμε να χρησιμοποιήσουμε τυχαίους αριθμούς για να βρούμε τη μέγιστη ή ελάχιστη τιμή μιας συνάρτησης πολλών μεταβλητών

Τυχαία αναζήτηση

Η μέθοδος αυτή υπολογίζει την συνάρτηση πολλές φορές σε τυχαία επιλεγμένες τιμές των ανεξάρτητων μεταβλητών.

Αν συγκεντρώσουμε ένα ικανοποιητικό αριθμό δειγμάτων τότε προφανώς θα έχουμε εντοπίσει και το ακρότατο.

Παράδειγμα: Χρησιμοποιήστε τη μέθοδο Monte Carlo για να υπολογίσετε το ελάχιστο της συνάρτησης $f(x) = x^2 - 6x + 5$ στο διάστημα x [1,5]

Η ακριβής λύση είναι f_{min}=-4.0 για x=3.0

Αλγόριθμος για ελάχιστα:

- Προσδιορισμός του πλήθους των πειραμάτων (N)
- Προσδιορισμός του διαστήματος [A,B]
- Αρχική τιμή για το ελάχιστο fmin=9E9 (πολύ μεγάλη τιμή)
- Επανάληψη της ακόλουθης διεργασίας N φορές)
  - Δημιουργία ενός τυχαίου αριθμού x στο [A,B]
  - Έλεγχος αν F(x) < fmin

    Αν ναι Βρήκαμε νέο ελάχιστο και κρατάμε τη τιμή του x

ΠΙΘΑΝΟΤΗΤΕΣ

Πιθανότητες

Αν ένα νόμισμα ριχθεί δεν είναι σίγουρο αν θα πάρουμε την πάνω όψη του. Ωστόσο αν συνεχίσουμε να επαναλαμβάνουμε το πείραμα αυτό, έστω N φορές και παίρνουμε την πάνω όψη S φορές, τότε ο λόγος S/N γίνεται σταθερός μετά από ένα μεγάλο πλήθος επανάληψης του πειράματος.

Η πιθανότητα P ενός γεγονότος A ορίζεται ως ακολούθως:

Αν το A μπορεί να συμβεί με S τρόπους από συνολικά K ισότιμους τρόπους τότε:

$$P = \frac{S}{K}$$

Παράδειγμα: Ρίχνοντας ένα νόμισμα, η πάνω όψη μπορεί να συμβεί μια φορά από τις δυνατές δύο περιπτώσεις. Επομένως η πιθανότητα είναι P = 1/2

Ο Αλγόριθμος για να λύσουμε το πρόβλημα αυτό:

- Προσδιορίζουμε το αριθμό των πειραμάτων N
- Μηδενίζουμε το μετρητή των επιτυχημένων αποτελεσμάτων
- Εκτελούμε τα N πειράματα (διαδικασία loop)
  - Δημιουργούμε ένα τυχαίο αριθμό x (ομοιόμορφη κατανομή)
  - Ελέγχουμε αν το x < P και αν ναι αυξάνουμε το μετρητή κατά 1 (επιτυχημένη προσπάθεια)

Το πρόγραμμα coin.py περιέχει το παραπάνω παράδειγμα

Ρίχνοντας ένα νόμισμα

Ένα νόμισμα ρίχνεται 6 φορές. Ποιά είναι η πιθανότητα να πάρουμε

(α) ακριβώς 4 φορές την πάνω όψη

(β) τουλάχιστον 4 φορές την πάνω όψη

Τα ακριβή αποτελέσματα δίνονται από τη διονυμική κατανομή:

Η διονυμική κατανομή: Μια τυχαία διεργασία με ακριβώς δυο πιθανά αποτελέσματα τα οποία συμβαίνουν με συγκεκριμένες πιθανότητες καλείται διεργασία Bernoulli. Αν η πιθανότητα να πάρουμε κάποιο αποτέλεσμα ("επιτυχία") σε κάθε προσπάθεια είναι p, τότε η πιθανότητα να πάρουμε ακριβώς r επιτυχίες (r=0,1,2,…,N) σε N ανεξάρτητες προσπάθειες χωρίς να παίζει ρόλο η σειρά με την οποία παίρνουμε επιτυχία ή αποτυχία, δίνεται από τη διονυμική κατανομή

$$f(r;N,p) = \frac{N!}{r!(N-r)!} p^r q^{N-r}$$

Για το πρόβλημά μας θα έχουμε επομένως:

(α) r=4 για N = 6 ενώ p=1/2 (q=1-p) και επομένως από τη διονυμική κατανομή:

$$P = \frac{6!}{4!\cdot 2!}\left(\frac{1}{2}\right)^4\left(\frac{1}{2}\right)^2 = \frac{30}{2}\frac{1}{64} = \frac{15}{64} = 0.234375$$

(β) r≥4 για N = 6 ενώ p=1/2 (q=1-p) και επομένως θα έχουμε σαν ολική πιθανότητα το άθροισμα για P(r=4) + P(r=5) + P(r=6)

$$P = \frac{6!}{4!\cdot 2!}\left(\frac{1}{2}\right)^4\left(\frac{1}{2}\right)^2 + \frac{6!}{5!\cdot 1!}\left(\frac{1}{2}\right)^5\left(\frac{1}{2}\right)^1 + \frac{6!}{6!\cdot 0!}\left(\frac{1}{2}\right)^6 = \frac{11}{32} = 0.34375$$

Το πρόγραμμα coin6.py περιέχει τη MC λύση του προβλήματος αυτού

Επίλυση ολοκληρωμάτων

Επίλυση ολοκληρωμάτων με τυχαίους αριθμούς

Χρησιμοποίηση τυχαίων αριθμών για επίλυση ολοκληρωμάτων

Η μέθοδος Monte Carlo δίνει μια διαφορετική προσέγγιση για την επίλυση ενός ολοκληρώματος

Τυχαίοι αριθμοί

Η συνάρτηση rand προσφέρει μια ακολουθία τυχαίων αριθμών ομοιόμορφα κατανεμημένων στο διάστημα [0,1)

Δύο βασικές μέθοδοι χρησιμοποιούνται για την επίλυση ολοκληρωμάτων

- Μέθοδος επιλογής ή δειγματοληψίας
- Μέθοδος μέσης τιμής

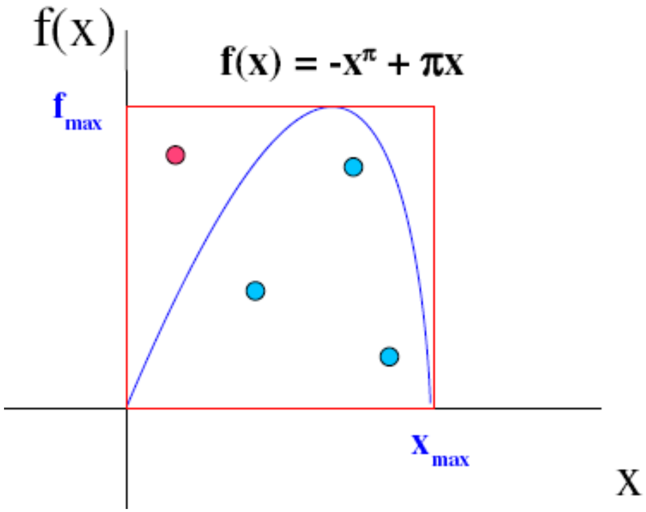
Monte Carlo – Μέθοδος δειγματοληψίας

- Περικλείουμε την συνάρτηση που θέλουμε να ολοκληρώσουμε μέσα σε ένα ορθογώνιο στο διάστημα της ολοκλήρωσης
  - Υπολογίζουμε το εμβαδό του ορθογωνίου
- Εισάγουμε τυχαία σημεία στο ορθογώνιο
  - Μετρούμε τα σημεία που βρίσκονται μέσα στο ορθογώνιο και αυτά που περικλείονται από την συνάρτηση
  - Το εμβαδό της συνάρτησης (ολοκλήρωμα) στο διάστημα ολοκλήρωσης δίνεται από

$$E_{f(x)} = E_{o\rho\theta o\gamma.} \times \frac{N_{f(x)}}{N_{o\rho\theta o\gamma.}}$$

Όπου $N_{f(\chi)}$= αριθμός

$N_{o\rho\theta o\gamma.}$ = αριθμός



Monte Carlo – Μέθοδος δειγματοληψίας/Απόρριψης

```
#!/usr/bin/python3

'''
!-----------------------------------------------------------
! Paradeigma oloklirwsis tis methodou aporripsis
! xrisimopoiontas tyxaioys arithmoys
! Efarmogi sti synartisi F(x) = x**2*exp(-x).
! Apotelesma einai -(x^2+2x+2)exp(-x)~1.99446
!-----------------------------------------------------------
'''
import numpy as np
import matplotlib.pyplot as plt
from random import random, seed

Ntries = int(input("How many tries to estimate the integral ? "))
iseed = 123456
seed(iseed)

def myfunc(x):
    return x*x * np.exp(-x)

ymax   =  0.55       # mexisti timi tis oloklirwteas sunartisis (paragwgos 0)
ymin   =  0.         # elaxisti timi tis oloklirwteas sunartisis
xlolim =  0.         # katw orio oloklirwsis
xuplim = 10.         # panw orio oloklirwsis

Npass  = 0.          # Metritis epityxeiwn
for itries in range(Ntries):
    x = random()                      # tyxaio simeio sto diastima [0,1)
    x = xlolim + (xuplim - xlolim)*x  # metatropi sto diastima [0,10)

    ftest=ymin + ymax * random()      # tuxaia metavliti ftest sto [0,ymax)
    if myfunc(x) > ftest :
       Npass = Npass+1.               # H f(x) perase to ftest

A = (ymax-ymin)*(xuplim-xlolim) # Embado tou orthog. pou perikleiei ti sunartisi
integral= A * Npass/Ntries
print("Meta apo %6d prospatheies to oloklirwma einai %6.4f:"%(Ntries,integral))
```

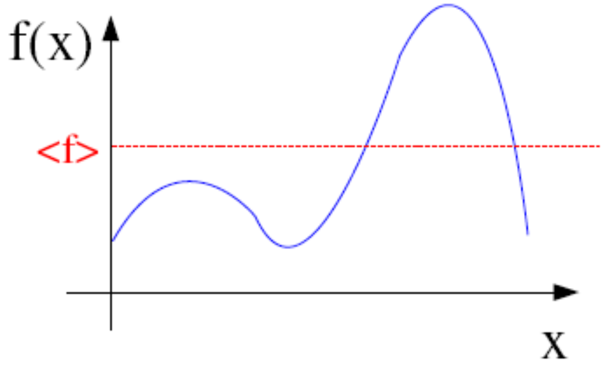
Το πρόγραμμα MCIntegrate_RejectAccept.py περιέχει τη MC λύση του προβλήματος αυτού

Monte Carlo - Μέθοδος Μέσης Τιμής

Η ολοκλήρωση με Monte Carlo γίνεται με το να πάρουμε τη μέση τιμή της συνάρτησης υπολογιζόμενη σε τυχαία επιλεγμένα σημεία μέσα στο διάστημα ολοκλήρωσης

$$I = \int_{x_a}^{x_b} f(x)dx = (b-a) < f(x) >$$

$$< f(x) > \simeq \frac{1}{N}\sum_{i=0}^{N} f(x_i)$$



Το στατιστικό σφάλμα: $\delta I = \sigma_{\bar{f}}$ όπου $\sigma_{\bar{f}} = \frac{\sigma_f}{\sqrt{N}}$

Monte Carlo - Μέθοδος Μέσης Τιμής

```
#!/usr/bin/python3
'''
!------------------------------------------------------
! Paradeigma oloklirwsis tis methodou mesis timis
! xrisimopoiontas tyxaioys arithmoys
! Efarmogi sti synartisi F(x) = x**2*exp(-x).
! Apotelesma einai -(x^2+2x+2)exp(-x)~1.99446
!------------------------------------------------------
'''
import numpy as np
import matplotlib.pyplot as plt
from random import random, seed

def integrand(x):
    y = x * x * np.exp(-x)
    return y

def monte_carlo_integration(ntr,LowLim,UpLim):
    value=0.
    for i in range(ntr):
        rnd = LowLim + (UpLim-LowLim)* random()
        rvs = integrand(rnd)
        value = value + rvs
    expected_value = (UpLim-LowLim)*(value/ntr)
    return expected_value


ntries = int(input("How many tries to evaluate the integral ? "))
lowlimit = float(input("The low limit of integration "))
hilimit = float(input("The upper limit of integration "))
iseed = 123456
seed(iseed)

result = monte_carlo_integration(ntries,lowlimit, hilimit)
print("After %6d tries, the integral is %10.5f"%(ntries,result))
```

Το πρόγραμμα MCIntegrate_MeanValue.py περιέχει τη MC λύση του προβλήματος αυτού

Monte Carlo - Ολοκλήρωση σε πολλές διαστάσεις

Εύκολο να γενικεύσουμε τη μέθοδο της μέσης τιμής σε πολλές διαστάσεις

Το σφάλμα στη μέθοδο ολοκλήρωσης με Monte Carlo είναι στατιστικό

Ελαττώνεται ως $1/\sqrt{N}$

Για 2 διαστάσεις:

$$I = \int_a^b dx_1 \int_c^d dx_2 f(x,y) \simeq (b-a)(d-c) \times \frac{1}{N} \sum_i^N f(x_i, y_i)$$

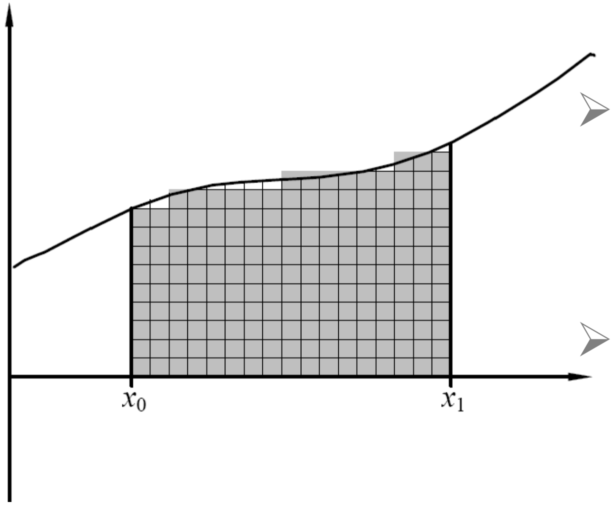
Αριθμητικές μέθοδοι ολοκλήρωσης

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

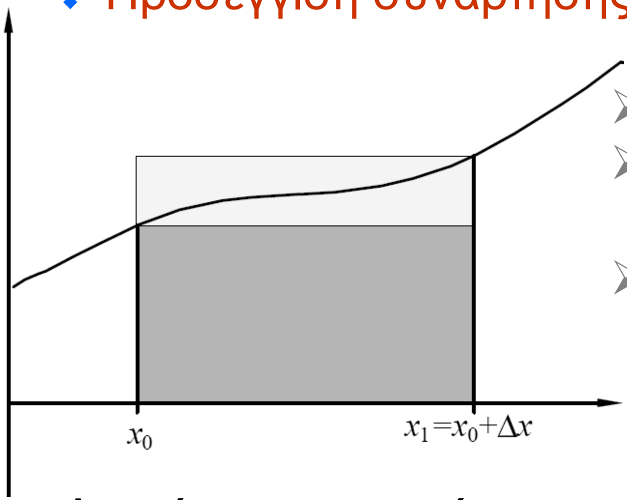
❖ Ολοκλήρωση με το "χέρι"



- 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:

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

σταθερά

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

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 1ης παραγώγου

Αριθμητική ολοκλήρωση – Κανόνας του τραπεζίου

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

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

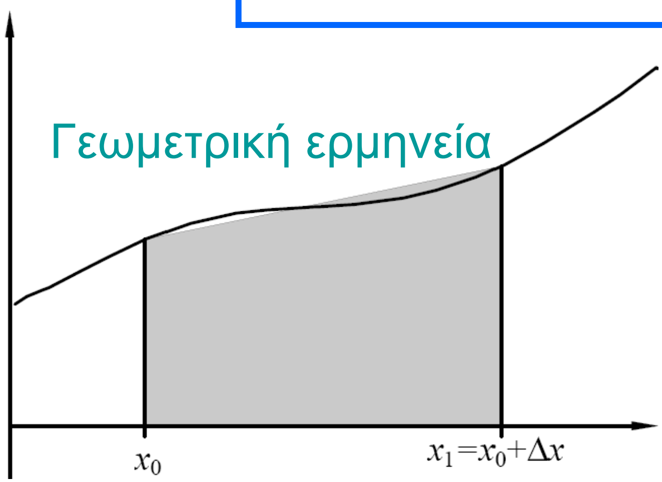
$$= f(x_0)\Delta x+\tfrac{1}{2}f'(x_0)\Delta x^2+\tfrac{1}{6}f''(x_0)\Delta x^3+\cdots$$

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

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

Κανόνας του τραπεζίου

Σφάλμα προσέγγισης

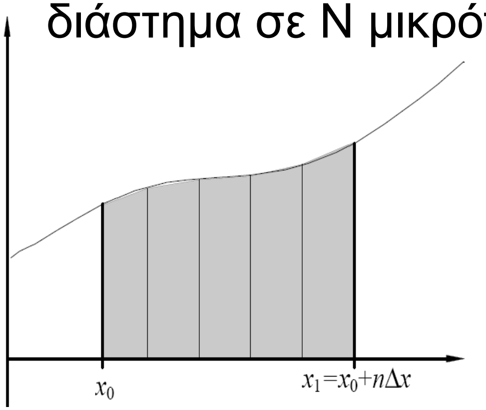


Αφού το σφάλμα ελαττώνεται κατά Δx^3 κάνοντας το διάστημα μισό, το σφάλμα θα μικραίνει κατά 8

Αλλά η περιοχή θα ελαττωθεί στο μισό και θα πρέπει να χρησιμοποιήσουμε τον 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

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

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

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

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 2η 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
```

Το πρόγραμμα Integration_trapezoidal.py περιέχει το παραπάνω πρόγραμμα

Αριθμητική ολοκλήρωση – Κανόνας μέσου σημείου

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

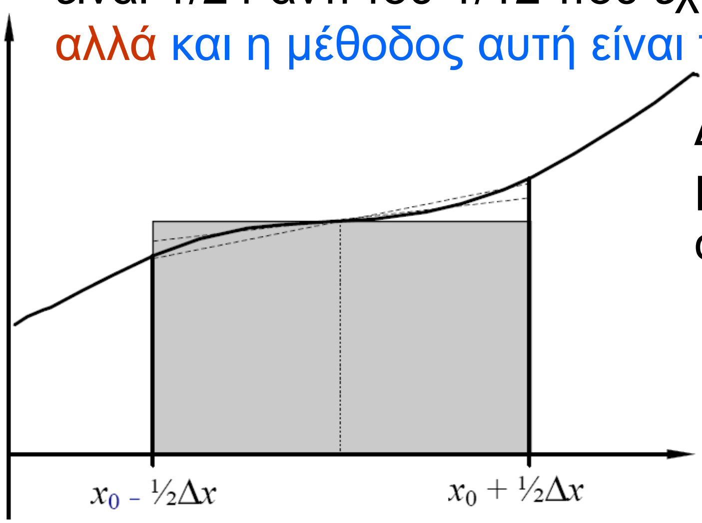
Η ολοκλήρωση του αναπτύγματος Taylor γίνεται από x_0-Δx/2 σε x_0+Δx/2 οπότε:

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

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

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:

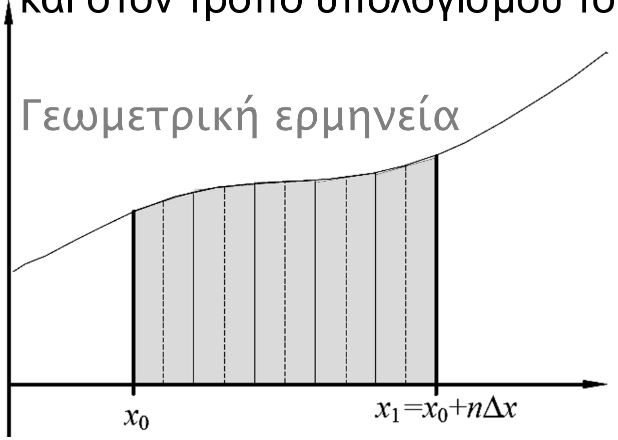
$$\int_{x_0}^{x_1} f(x)dx \approx \Delta x \sum_{i=0}^{n-1} f\left(x_0+\left(i+\tfrac{1}{2}\right)\Delta x\right)$$

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

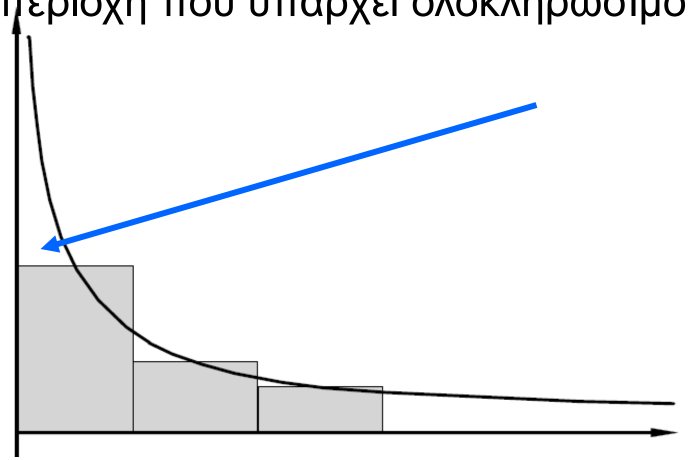
Η διαφορά τους είναι μόνο στη "φάση" των σημείων και της περιοχής υπολογισμού, και στον τρόπο υπολογισμού του πρώτου και τελευταίου διαστήματος



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 ιδιάζον 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))
```

Το πρόγραμμα Integration_midpoint.py περιέχει το παραπάνω πρόγραμμα