

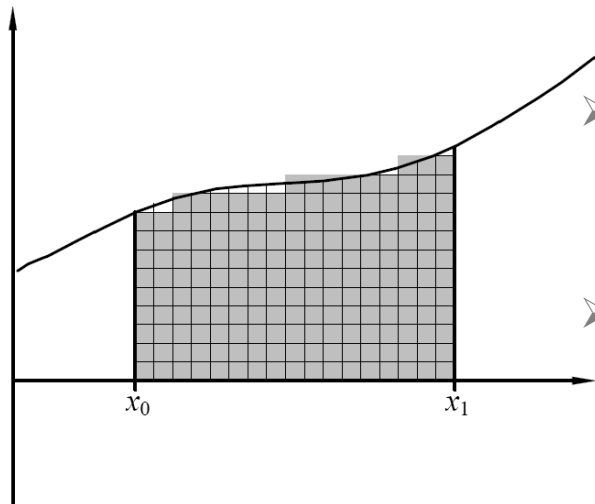
Αριθμητική ολοκλήρωση

□ 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

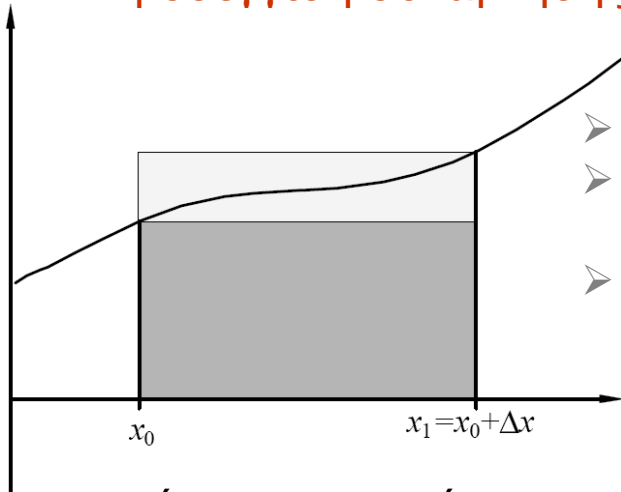
❖ Ολοκλήρωση με το “χέρι”



- 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:

$$\begin{aligned} \int_{x_0}^{x_0+\Delta x} f(x)dx &= \int_{x_0}^{x_0+\Delta x} \left[f(x_0) + f'(x_0)(x-x_0) + \frac{1}{2}f''(x_0)(x-x_0)^2 + \dots \right] dx \\ &= f(x_0)\Delta x + \frac{1}{2}f'(x_0)(x-x_0)^2 + \frac{1}{6}f''(x_0)(x-x_0)^3 + \dots = \underbrace{f(x_0)}_{\text{σταθερά}} \Delta x + O(\Delta x^2) \end{aligned}$$

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 $x_0 + \Delta x$

$$\begin{aligned} \int_{x_0}^{x_0+\Delta x} f(x) dx &= \int_{x_0}^{x_0+\Delta x} \left[f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2} f''(x_0)(x - x_0)^2 + \dots \right] dx \\ &= f(x_0)\Delta x + \frac{1}{2} f'(x_0)\Delta x^2 + \frac{1}{6} f''(x_0)\Delta x^3 + \dots \\ &= \left\{ \frac{1}{2} f(x_0) + \frac{1}{2} \left[f(x_0) + f'(x_0)\Delta x + \frac{1}{2} f''(x_0)\Delta x^2 + \dots \right] - \frac{1}{12} f''(x_0)\Delta x^2 + \dots \right\} \Delta x \end{aligned}$$

$$= \frac{1}{2} [f(x_0) + f(x_0 + \Delta x)] \Delta x + O(\Delta x^3)$$

Κανόνας του τραπεζίου
Σφάλμα προσέγγισης



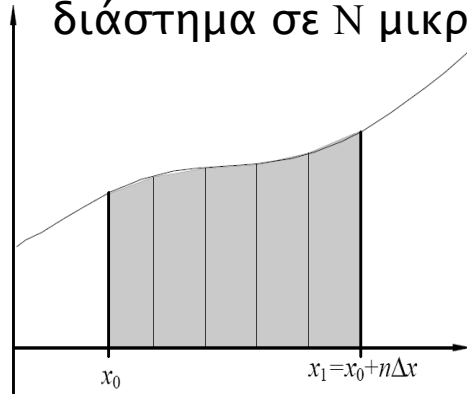
Αφού το σφάλμα ελαττώνεται κατά Δx^3 κάνοντας το διάστημα μισό το σφάλμα θα μικραίνει κατά 8

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

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

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

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

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

Αριθμητική ολοκλήρωση – Κανόνας μέσου σημείου

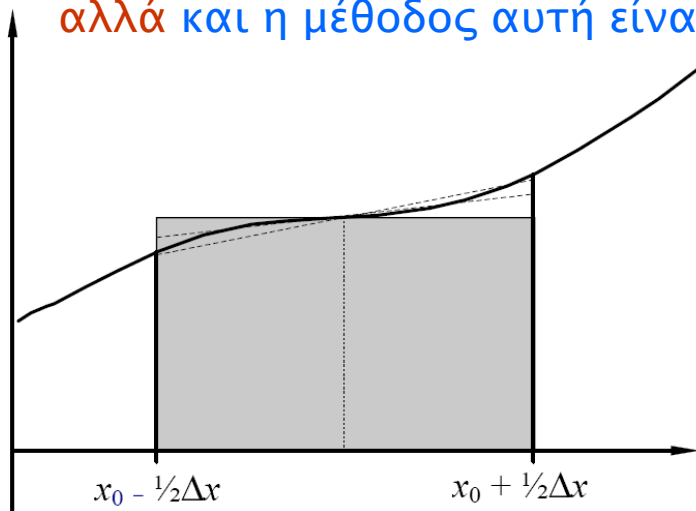
Μια παραλλαγή του κανόνα του τραπεζίου είναι ο κανόνας του μέσου σημείου

Η ολοκλήρωση του αναπτύγματος Taylor γίνεται από $x_0 - \Delta x/2$ σε $x_0 + \Delta x/2$ οπότε:

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

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

Υπολογίζοντας τη συνάρτηση στο **μέσο κάθε διαστήματος** το σφάλμα μπορεί να ελαττωθεί κάπως μια και ο παράγοντας μπροστά από τον όρο της 2^{ης} 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:

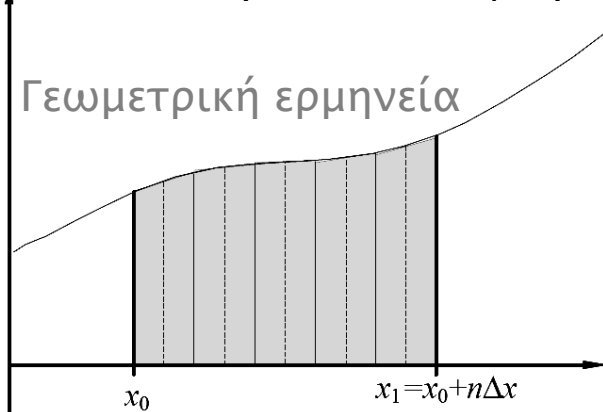
$$\int_{x_0}^{x_1} f(x) dx \approx \Delta x \sum_{i=0}^{n-1} f\left(x_0 + \left(i + \frac{1}{2}\right)\Delta x\right)$$

Αριθμητική ολοκλήρωση – Κανόνας μέσου σημείου

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

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

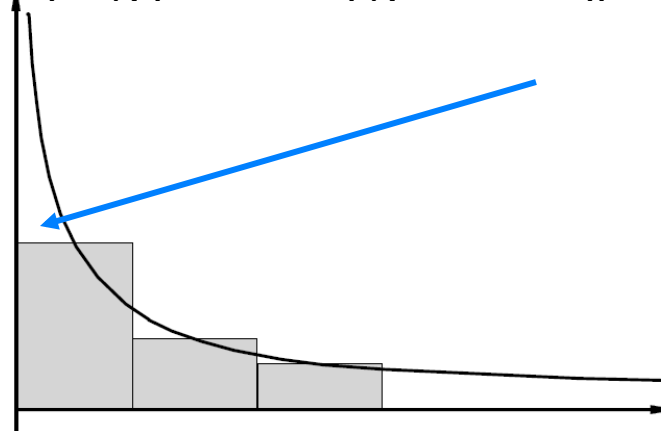
Η διαφορά τους είναι μόνο στη “φάση” των σημείων και της περιοχής υπολογισμού, και στον τρόπο υπολογισμού του πρώτου και τελευταίου διαστήματος



Ωστόσο υπάρχουν **δύο πλεονεκτήματα**

της μεθόδου του ενδιάμεσου σημείου σε σχέση με την μέθοδο του τραπεζίου:

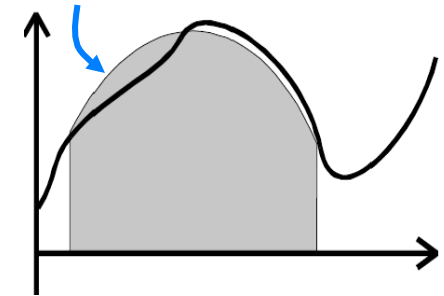
- A) Χρειάζεται ένα υπολογισμό της $f(x)$ λιγότερο
- B) Μπορεί να χρησιμοποιηθεί πιο αποτελεσματικά για τον υπολογισμό του ολοκληρώματος κοντά σε μια περιοχή που υπάρχει ολοκληρώσιμο ιδιάζων σημείο



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**.

2^{ου} βαθμού προσέγγιση της $f(x)$



Ολοκληρώνοντας το ανάπτυγμα Taylor σε ένα διάστημα $2\Delta x$

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

$$\begin{aligned} \int_{x_0}^{x_0+2\Delta x} f(x)dx &= \frac{\Delta x}{3} \left[f(x_0) + \right. \\ &\quad \left. + 4 \left(f(x_0) + f'(x_0)\Delta x + \frac{1}{2}f''(x_0)\Delta x^2 + \frac{1}{6}f'''(x_0)\Delta x^3 + \frac{1}{24}f^{iv}(x_0)\Delta x^4 + \dots \right) \right. \\ &\quad \left. + \left(f(x_0) + 2f'(x_0)\Delta x + 2f''(x_0)\Delta x^2 + \frac{4}{3}f'''(x_0)\Delta x^3 + \frac{2}{3}f^{iv}(x_0)\Delta x^4 + \dots \right) - \right. \\ &\quad \left. - \frac{17}{30}f^{iv}(x_0)\Delta x^4 \dots \right] \Rightarrow \end{aligned}$$

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

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:

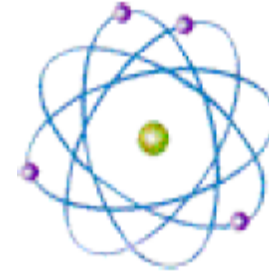
$$\int_{x_0}^{x_1} f(x)dx \approx \frac{\Delta x}{3} \sum_{i=0}^{m-1} \left[f(x_0 + 2i\Delta x) + 4f(x_0 + (2i+1)\Delta x) + f(x_0 + (2i+2)\Delta x) \right]$$

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

Ολοκλήρωση - Πολλαπλά ολοκληρώματα

Για παράδειγμα, έστω τα ηλεκτρόνια του ατόμου του ${}^4\text{Be}$:

$$I = \int_0^1 dx_1 \int_0^1 dx_2 \int_0^1 dx_3 \cdots \int_0^1 dx_{12} f(x_1, x_2, \dots, x_{12})$$



Έχουμε 3 διαστάσεις για κάθε ηλεκτρόνιο x 4 ηλεκτρόνια = 12 διαστάσεις

Αλλά για 100 σημεία για κάθε ολοκλήρωση τότε θα χρειαστούμε συνολικά

$$100^{12} = 10^{24} \text{ υπολογισμούς}$$

Για τα PCs υποθέτοντας 1Giga υπολογισμούς/sec θα χρειαστούμε 10^7 χρόνια

Ολοκλήρωση - Μέθοδος Monte Carlo

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 $[0, \text{RAND_MAX}]$

Δύο βασικές μέθοδοι χρησιμοποιούνται για την επίλυση ολοκληρωμάτων

Μέθοδος επιλογής ή δειγματοληψίας

Μέθοδος μέσης τιμής

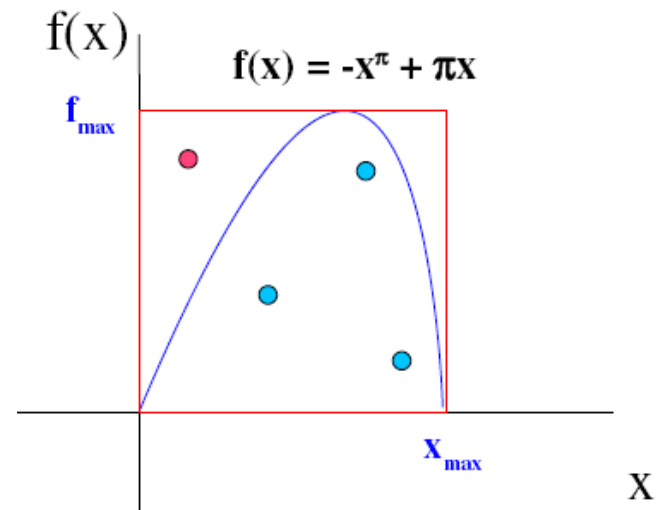
Monte Carlo μέθοδος δειγματοληψίας

- 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

$$E_{f(x)} = E_{\text{ορθογ.}} \times \frac{N_{f(x)}}{N_{\text{ορθογ.}}}$$

Όπου $N_{f(x)}$ = αριθμός ●

$N_{\text{ορθογ.}}$ = αριθμός ●

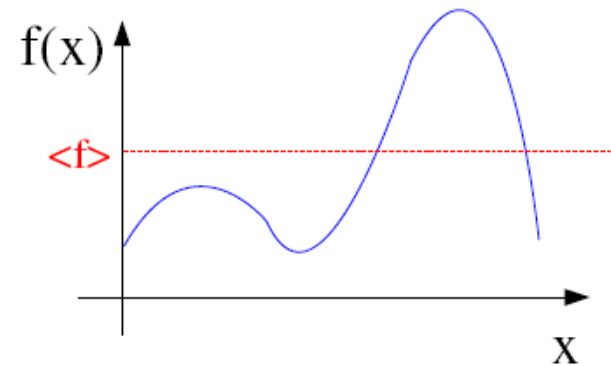


Monte Carlo μέθοδος μέσης τιμής

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

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

$$\langle f(x) \rangle \approx \frac{1}{N} \sum_{i=0}^N f(x_i)$$



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

Monte Carlo 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 $\frac{1}{\sqrt{N}}$

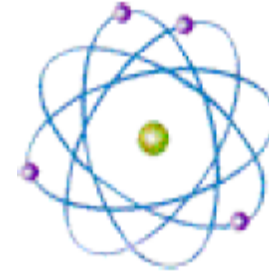
Για 2 διαστάσεις:

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

Ολοκλήρωση - Πολλαπλά ολοκληρώματα

Για παράδειγμα, έστω τα ηλεκτρόνια του ατόμου του ${}^4\text{Be}$:

$$I = \int_0^1 dx_1 \int_0^1 dx_2 \int_0^1 dx_3 \cdots \int_0^1 dx_{12} f(x_1, x_2, \dots, x_{12})$$



Έχουμε 3 διαστάσεις για κάθε ηλεκτρόνιο x 4 ηλεκτρόνια = 12 διαστάσεις

$$I = (1-0)^{12} \times \frac{1}{N} \sum f(x_1^i, x_2^i, \dots, x_{12}^i)$$

Αλλά για $N=10^6$ σημεία στη ολοκλήρωση Monte Carlo έχουμε 10^6 υπολογισμούς

Για τα PCs υποθέτοντας 1 Giga υπολογισμούς/sec θα χρειαστούμε 10^{-3} secs !!

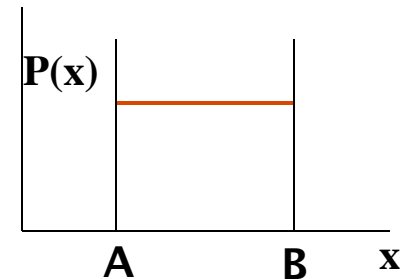
Τυχαίες κατανομές

Ομοιόμορφη (επίπεδη) Κατανομή

Αν το R είναι ένα τυχαίος πραγματικός αριθμός μεταξύ $[0,1)$ και αν τα A και B είναι πραγματικοί αριθμοί, και M, N είναι ακέραιοι αριθμοί τότε η τιμή:

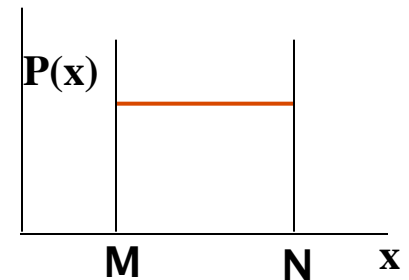
$$A + (B - A) * (rand() / RAND_MAX)$$

θα είναι ένα τυχαίος πραγματικός αριθμός στο διάστημα $[A,B)$



Η τιμή της $M + \text{int}\left(N * rand() / RAND_MAX\right)$

θα είναι ένα τυχαίος ακέραιος αριθμός στο διάστημα $[M,N]$



Για παράδειγμα, για να δημιουργήσουμε την ομοιόμορφη τυχαία κατανομή που αντιστοιχεί στο ρίξιμο των ζαριών (τιμές μεταξύ $[1,6]$) θα γράφαμε:

$$zari = 1 + \text{int}\left(6 * rand() / RAND_MAX\right)$$

Τυχαίες κατανομές – Probability Distribution Function (PDF)

□ Πως μπορούμε να βρούμε μια γενική Συνάρτηση Κατανομής Πιθανότητας (PDF)

Στις προσομοιώσεις μιας τυχαίας διεργασίας συχνά ζητάμε μια **μη** ομοιόμορφη (ισοπίθανη) κατανομή τυχαίων αριθμών.

Για παράδειγμα η ραδιενεργός διάσπαση χαρακτηρίζεται από κατανομή Poisson:

$$P(n;v) = \frac{v^n e^{-v}}{n!}$$

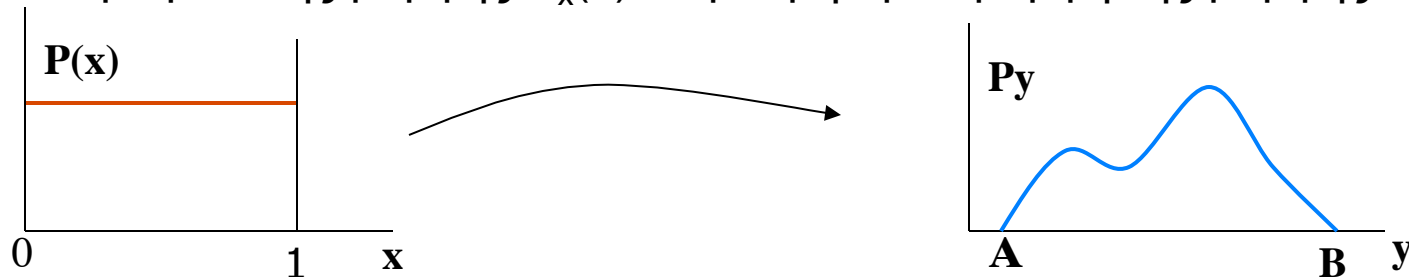
(πιθανότητα να βρούμε ακριβώς n γεγονότα όταν τα γεγονότα συμβαίνουν ανεξάρτητα το ένα από το άλλο και την ανεξάρτητη μεταβλητή x και με μέσο ρυθμό v στο διάστημα x)

➤ Χρησιμοποιούμε δυο μεθόδους (συνήθως)

□ Μέθοδος του μετασχηματισμού ([Transformation Method](#))

□ Μέθοδος της απόρριψης ([Rejection Method](#))

Σκοπός και των δυο μεθόδων είναι να μετατρέψουν μια ομοιόμορφη κατανομή τυχαίων αριθμών της μορφής $P_x(x)$ σε μια μη ομοιόμορφη της μορφής $P_y(y)$

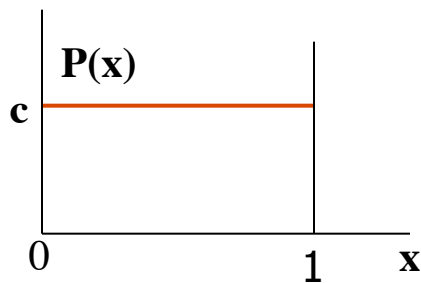


Τυχαίες κατανομές – Η μέθοδος του μετασχηματισμού

Θεωρήστε μια συλλογή από μεταβλητές $\{x_1, x_2, \dots, x_n\}$ που κατανέμονται σύμφωνα με μια συνάρτηση $P_x(x)$, τότε η πιθανότητα να βρούμε μια τιμή στο διάστημα x και $x+dx$ είναι $P_x(x)dx$

Αν y είναι κάποια συνάρτηση του x τότε μπορούμε να γράψουμε: $|P_x(x)dx| = |P_y(y)dy|$

Όπου $P_y(y)$ είναι η πυκνότητα πιθανότητας που περιγράφει την συλλογή $\{y_1, y_2, \dots, y_n\}$



$$P_x(x) = c \quad \text{Ομοιόμορφη κατανομή}$$

$$\text{και επομένως: } P_y(y) = c \left| \frac{dx}{dy} \right|$$

Για να βρούμε μια ακολουθία που χαρακτηρίζεται από την $P_y(y)$ θα πρέπει να βρούμε τη συνάρτηση μετασχηματισμού $y=f(x)$ που ικανοποιεί την εξίσωση:

$$\left| \frac{dx}{dy} \right| = P_y(y)$$

Η σταθερά C μπορεί να αγνοηθεί αφού απλά πολλαπλασιάζει την συνάρτηση

Επομένως:

□ Θέλουμε την $P_y(y)$

➤ Βρίσκουμε την $y=f(x)$ (συνάρτηση μετασχηματισμού) ώστε $|dx/dy| = P_y(y)$

Τυχαίες κατανομές - Η μέθοδος του μετασχηματισμού

Παραδείγματα τέτοιων συναρτήσεων μετασχηματισμού είναι τα ακόλουθα:

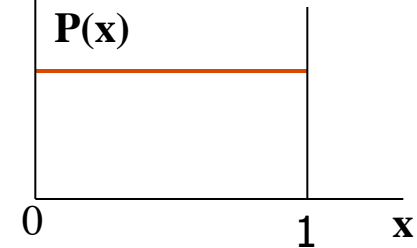
Επιθυμητή κατανομή $P_y(y)$	Συνάρτηση μετασχηματισμού $f(x)$
k	x/k
y	$\sqrt{2x}$
y^n	$[(n+1)x]^{1/(n+1)}$
$1/y$	e^x
e^y	$-\ln(x)$
$\cos y$	$\arcsin x$
\sqrt{y}	$[3/2x]^{2/3}$

Τυχαίες κατανομές - Η μέθοδος του μετασχηματισμού

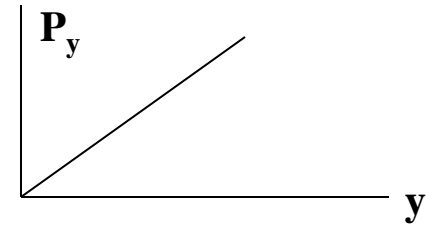
Παράδειγμα: Θεωρήστε ότι θέλετε την κατανομή $P_y(y) = y$

Η συνάρτηση μετασχηματισμού είναι τότε:

$$\left| \frac{dx}{dy} \right| = P_y(y) = y \Rightarrow x = \int y dy$$



Λύνοντας ως προς y έχουμε: $\frac{1}{2} y^2 = x \Rightarrow y = \sqrt{2x}$



Όπου τα x είναι τυχαίοι αριθμοί στο διάστημα $[0,1)$

Τυχαίες κατανομές – Η μέθοδος της απόρριψης

Η μέθοδος του μετασχηματισμού είναι χρήσιμη όταν η συνάρτηση $f(x)$ μπορεί να υπολογιστεί

Ωστόσο υπάρχουν περιπτώσεις που η επιθυμητή συνάρτηση μπορεί να μην είναι γνωστή σε αναλυτική μορφή.

Για παράδειγμα, θεωρήστε ότι θέλουμε μια Gaussian κατανομή: $P_y(y) = e^{-y^2}$

Για την περίπτωση αυτή δεν βρίσκουμε την συνάρτηση $y=f(x)$ γιατί:

$$\frac{dx}{dy} = P_y(y) = e^{-y^2} \Rightarrow x = \int e^{-y^2} dy \quad \text{Το ολοκλήρωμα αυτό δεν υπολογίζεται αναλυτικά}$$

Για τέτοιου είδους προβλήματα χρησιμοποιούμε τη μέθοδο της απόρριψης η οποία μπορεί να δημιουργήσει την επιθυμητή κατανομή για οποιαδήποτε συνάρτηση.

Με τη μέθοδο αυτή, η ακολουθία των τυχαίων αριθμών $\{y_1, y_2, \dots, y_n\}$ δημιουργείται με μια ομοιόμορφη κατανομή στο διάστημα $[y_{\min}, y_{\max}]$ που ενδιαφερόμαστε.

Υποθέστε ότι ο σκοπός μας είναι να δημιουργήσουμε μια ακολουθία αριθμών κατανεμημένων σύμφωνα με τη συνάρτηση $P_y(y)$



Προχωρούμε με την ακολουθία των αριθμών $\{y_1, y_2, \dots, y_n\}$ και δεχόμαστε τιμές που είναι πολλαπλάσια της $P_y(y)$

Τυχαίες κατανομές – Η μέθοδος της απόρριψης

➤ Ο αλγόριθμος που ακολουθούμε είναι ο ακόλουθος:

- Θέτουμε $m = 0$ (μετρητής)
- Προσδιορίζουμε την μέγιστη τιμή της επιθυμητής κατανομής $P_y(y)$ και το διάστημα $[y_{\min}, y_{\max}]$ της επιθυμητής κατανομής
- Δημιουργούμε τυχαίο ζευγάρι $zran$, $yran$ από μια ομοιόμορφη κατανομή
- Για κάθε τιμή του y θέτουμε: $y = y_{\min} + (y_{\max} - y_{\min}) * yran$
 - ❑ Δημιουργούμε ένα τυχαίο αριθμό p_{test} ομοιόμορφα κατανεμημένο στο $[0, P_y^{\max}]$

$$Py_test = Py_max * zran$$

❑ Ελέγχουμε αν η τιμή της κατανομής $P(y) > Py_test$

Αν ναι: Αυξάνουμε τον μετρητή $m = m+1$

Κρατάμε τη τιμή y μια και δίνει επιτρεπτή τιμή: θέτουμε $z(m) = y$

Αν όχι: Απορρίπτουμε την τιμή y

- Επαναλαμβάνουμε την διαδικασία N φορές:

Οι αριθμοί $z(m)$ που απομένουν στην ακολουθία κατανέμονται επομένως σύμφωνα με την συνάρτηση $P_y(y)$

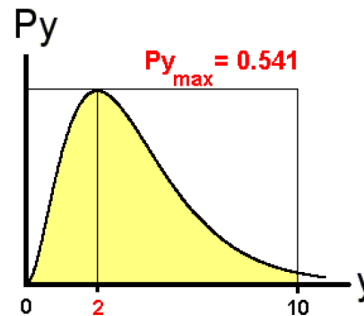
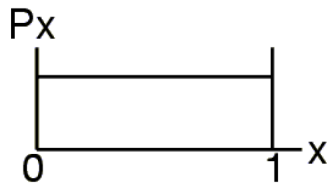
Τυχαίες κατανομές – Η μέθοδος της απόρριψης

Παράδειγμα: Έστω ότι θέλουμε την κατανομή $P_y(y) = e^{-y}y^2$ στο διάστημα $y=[0,10]$

Η μέγιστη τιμή της συνάρτησης P_y βρίσκεται ζητώντας $\frac{dP_y}{dy} = 0$

Η συνάρτηση έχει μέγιστο στο σημείο $y=2$ το οποίο αντιστοιχεί σε $P_{\max}=0.541$

Επομένως οι τιμές του P_{y_test} θα δημιουργηθούν στο διάστημα $[0,0.541]$



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 $f_{\min} = 9E9$ (πολύ μεγάλη τιμή)
- Επανάληψη της ακόλουθης διεργασίας N φορές
 - ☐ Δημιουργία ενός τυχαίου αριθμού x στο [A,B]
 - ☐ Έλεγχος αν $F(x) < f_{\min}$
 - Αν ναι Βρήκαμε νέο ελάχιστο και κρατάμε τη τιμή του x

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)

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 διωνυμική κατανομή:

Η διωνυμική 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 ($r=0,1,2,\dots,N$) σε N ανεξάρτητες προσπάθειες χωρίς να παίζει ρόλο η σειρά με την οποία παίρνουμε επιτυχία ή αποτυχία, δίνεται από τη διωνυμική 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 διωνυμική 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) $r \geq 4$ για $N=6$ ενώ $p=1/2$ ($q=1-p$) και επομένως θα έχουμε σαν ολική πιθανότητα το άθροισμα για $P(r=4) + P(r=5) + P(r=6)$

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

Κλάσεις στη C++

- ❑ Ο χρήστης μπορεί να ορίσει νέους τύπους αντικειμένων, η πρόσβαση στις ιδιότητες (στοιχεία) των οποίων γίνεται μέσω συγκεκριμένων συναρτήσεων
- ❑ Τύπος είναι η αναπαράσταση ενός «αντικειμένου» όπως `float` με συναρτήσεις `+`, `-`, `*`, `/`, ... είναι η αναπαράσταση των αριθμών κινητής υποδιαστολής
- ❑ Θα πρέπει να υπάρχει διαχωρισμός των λεπτομερειών υλοποίησης του τύπου από τις ιδιότητες / λειτουργίες του τύπου
- ❑ Η πρόσβαση στις ιδιότητες των τύπων γίνεται μέσω συναρτήσεων που ονομάζονται **member functions** της κλάσης. (Μπορεί να γίνει και με φιλικών προς την κλάση (**friends**))
- ❑ Όταν δημιουργούνται τα αντικείμενα της κλάσης, καλούνται συγκεκριμένες συναρτήσεις που δηλώνονται στην κλάση και λέγονται συναρτήσεις κατασκευής (**constructors**)
- ❑ Κατ'αναλογία υπάρχουν και συναρτήσεις που αποδομούν το χώρο που χρησιμοποιήθηκε για τα αντικείμενα της κλάσης (**destructors**)
- ❑ Το αντικείμενο μιας κλάσης μπορεί να δημιουργηθεί σαν **static** είτε σαν **automatic** ή **new**

Η ιδέα των κλάσεων

- ❑ Οι κλάσεις μοιάζουν σε ιδέα με τις δομές struct
- ❑ Η ιδέα είναι:
 - να βάλουμε μαζί διαφορετικού τύπου δεδομένα σε ένα αντικείμενο
 - τους τρόπους πρόσβασης του χρήστη στη λειτουργικότητα και ιδιότητες των τύπων
 - τις συναρτήσεις που υπαγορεύουν πως αλληλεπιδρά το αντικείμενο που ορίζεται με άλλα αντικείμενα και πως χρησιμοποιείται από τον χρήστη
- ❑ Η βάση για να το κάνουμε αυτό ονομάζεται **class**

```
class name_of_type
{
    public:
        // δηλώσεις συναρτήσεων
    private:
        // δηλώσεις μελών
};
```

- ❑ Οι λέξεις κλειδιά είναι:
 - class** **public** **private**
- ❑ Η περιοχή private, ορίζει το τμήμα της κλάσης τα μέλη του οποίου δεν είναι προσβάσιμα έξω από την κλάση
 - Δεν μπορούν να κληθούν ή να τροποποιηθούν από άλλη συνάρτηση ή υποπρόγραμμα, όπως π.χ. τη συνάρτηση main
 - Μπορούν να τροποποιηθούν ή να κληθούν από συναρτήσεις που βρίσκονται στο public τμήμα της κλάσης.
- ❑ Στο τμήμα public, υπάρχουν όλες οι συναρτήσεις επικοινωνίας του χρήστη με τις δεδομένα της κλάσης του αντικειμένου, τους τρόπους αλληλεπίδρασης και τροποποίησης. Υπάρχουν συναρτήσεις που μπορεί να χρησιμοποιούν συναρτήσεις που βρίσκονται στο private τμήμα

Κλάση

```
class name_of_type
{
    public:
        // δηλώσεις συναρτήσεων
    private:
        // δεδομένα μελών
};
```

```
class Fraction
{
    public:
        void readin();
        void print();
        Fraction reciprocal();
        void unreduced(const int m);
    private:
        int m_Numerator;
        int m_Denominator;
};
```

Το public τμήμα της κλάσης περιέχει 4 συναρτήσεις και δίνεται μόνο η δήλωση των συναρτήσεων αυτών

Το private τμήμα περιέχει τα δεδομένα της κλάσης που είναι ο αριθμητής και ο παρονομαστής που προσδιορίζουν το τύπο Fraction

Κλάση

```
class Fraction
{
    int m_Numerator;
    int m_Denominator;
public:
    void readin();
    void print();
    Fraction reciprocal();
    void unreduced(const int m);
};
```

```
class Fraction
{
public:
    void readin();
    void print();
    Fraction reciprocal();
    void unreduced(const int m);
private:
    int m_Numerator;
    int m_Denominator;
};
```

Οτιδήποτε υπάρχει μέσα στο {...} και δεν έχει δηλωθεί ακριβώς σαν public, τότε θεωρείται ότι είναι private

Είναι καλή πρακτική να ξεκαθαρίζεται επ'ακριβώς τι ανήκει στο private τμήμα και τι στο public

Ορισμός της κλάσης και χρήση

Πως χρησιμοποιούμε τα προηγούμενα:

Ο ορισμός της κλάσης (ή διαφορετικά η δήλωση της κλάσης) είναι το τμήμα που περιέχεται από το όνομα `class class_name_type {...};`

Ο ορισμός της κλάσης είναι καλό να γίνεται σε ένα header file (.h) το όνομα του οποίου είναι το όνομα της κλάσης

Δεν βάζουμε τίποτα άλλο στο header file εκτός από σταθερές που ωστόσο σχετίζονται με την class

```
//fraction.h
#ifndef FRACTION_H
#define FRACTION_H
class Fraction
{
public:
    void readin();
    void print();
    Fraction reciprocal();
    void unreduced(const int m);
private:
    int m_Numerator;
    int m_Denominator;
};
#endif
```

Η ιδέα είναι ότι θα μπορούμε να μεταφέρουμε το αντικείμενο που ορίζει η κλάση από πρόγραμμα σε πρόγραμμα

Θα πρέπει να ορίσουμε και ένα .C file το οποίο περιέχει τον κώδικα του τι κάνουν οι συναρτήσεις που έχουν δηλωθεί

Μια απλή χρήση της κλάσης φαίνεται παρακάτω μέσω του main

```
#include "fraction.h" ← header file
int main()
{
```

Fraction f, g; ← 2 αντικείμενα τύπου fraction

f.m_Numerator = 7; ← **λάθος οδηγία! Δεν μπορούμε να αλλάξουμε private δεδομένο**

f.readin();
f.print();
f.unreduce(5); } ← κλήση των μελών συναρτήσεων μέσω του καλόντος αντικειμένου **f**

return 0;

όπως και στις struct χρησιμοποιούμε τον «.» τελεστή για πρόσβαση μελών

}

Ορισμοί των συναρτήσεων μελών της κλάσης

Οι ορισμοί των συναρτήσεων γίνονται σε ένα file το οποίο ονομάζεται implementation file

```
//fraction.h
#ifndef FRACTION_H
#define FRACTION_H
class Fraction
{
public:
    void readin();
    void print();
    Fraction reciprocal();
    void unreduce(const int m);
private:
    int m_Numerator;
    int m_Denominator;
};
#endif

//fraction.cpp
#include "fraction.h"
#include <iostream>
using std::cout;
using std::cin;

void Fraction::readin() {
    cout<<"enter numerator: ";
    cin>>m_Numerator;
    cout<<"enter denominator: ";
    cin>>m_Denominator;
    return;}

void Fraction::print() {
    cout<<"("<<m_Numerator
        <<"/"<<m_Denominator<<")";
    return;}

Fraction Fraction::reciprocal() {
    Fraction returnable;
    returnable.m_Numerator =
        m_Denominator;
    returnable.m_Denominator =
        m_Numerator;
    return returnable;}

void Fraction::unreduce(const int m)
{
    m_Numerator*=m;
    m_Denominator*=m;
    return;}


```

Χρήση της κλάσης

```
//fraction.h
#ifndef FRACTION_H
#define FRACTION_H

class Fraction
{
public:
    void readin();
    void print();
    Fraction reciprocal();
    void unreduce(const int m);
private:
    int m_Numerator;
    int m_Denominator;
};

#endif
```



```
#include "fraction.h"

int main()
{
    Fraction f1, f2, f3;
    f1.readin();
    f1.print();
    f2.readin();
    f2.print();
    f2.unreduce(2);
    f2.print();
    f3 = f1.reciprocal();
    f3 = f1 + f2;
    ...
}
```


Χρήση της κλάσης

```
void Fraction::readin() {
    cout<<"enter numerator: ";
    cin>>m_Numerator;
    cout<<"enter denominator: ";
    cin>>m_Denominator;
    return;}

```

Οι συναρτήσεις που είναι μέλη της κλάσης προσδιορίζονται με το όνομα της κλάσης και τον τελεστή :: πριν το όνομα της συνάρτησης

Προσοχή: τα αντικείμενα f1,f2,f3, όλα έχουν **κοινές** συναρτήσεις μέλη της κλάσης και όχι αντίγραφα των συναρτήσεων

Ωστόσο: τα αντικείμενα f1,f2,f3, όλα έχουν **δικά τους και μοναδικά αντίγραφα** των δυο ακεραίων (στο παράδειγμά μας m_numerator και m_denominator

Η μέθοδος, f1.readin() εισάγει 2 ακεραίους οι οποίοι είναι μοναδικοί για το αντικείμενο f1 και δεν μοιράζεται με οποιοδήποτε άλλο αντικείμενο. Ένα αντικείμενο f2 θα αποκτήσει τα δικά του δεδομένα μεταβλητές μέλη με αντίστοιχη κλίση της f2.readin()

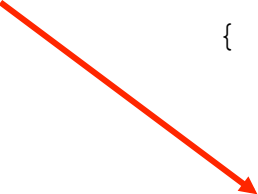
```
#include "fraction.h"

int main()
{
    Fraction f1, f2, f3;
    f1.readin();
    f1.print();
    f2.readin();
    f2.print();
    f2.unreduce(2);
    f2.print();
    f3 = f1.reciprocal();
    f3 = f1 + f2;
    ...
}
```

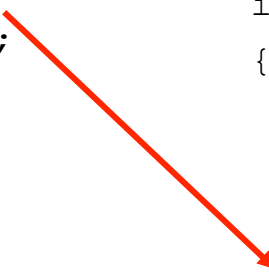
Χρήση της κλάσης

```
void Fraction::print() {  
    cout<<" "<<m_Numerator  
        <<"/"<<m_Denominator<<")";  
    return;  
}
```

```
#include "fraction.h"  
  
int main()  
{  
    Fraction f1, f2, f3;  
    f1.readin();  
    f1.print();  
    f2.readin();  
    f2.print();  
    f2.unreduce(2);  
    f2.print();  
    f3 = f1.reciprocal();  
    f3 = f1 + f2;  
    ...  
}
```



Χρήση της κλάσης

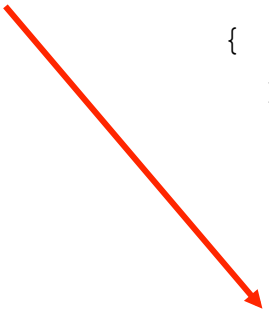
```
void Fraction::readin() {  
    cout<<"enter numerator: ";  
    cin>>m_Numerator;  
    cout<<"enter denominator: ";  
    cin>>m_Denominator;  
    return;}  

```

```
#include "fraction.h"  
  
int main()  
{  
    Fraction f1, f2, f3;  
    f1.readin();  
    f1.print();  
    f2.readin();  
    f2.print();  
    f2.unreduce(2);  
    f2.print();  
    f3 = f1.reciprocal();  
    f3 = f1 + f2;  
    ...  
}
```

Χρήση της κλάσης

```
void Fraction::print() {  
    cout<<" "<<m_Numerator  
        <<"/"<<m_Denominator<<" "<<endl;  
    return;  
}
```

```
#include "fraction.h"  
  
int main()  
{  
    Fraction f1, f2, f3;  
    f1.readin();  
    f1.print();  
    f2.readin();  
    f2.print();  
    f2.unreduce(2);  
    f2.print();  
    f3 = f1.reciprocal();  
    f3 = f1 + f2;  
    ...  
}
```



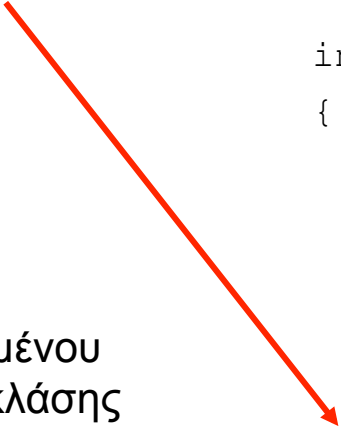
Χρήση της κλάσης

```
void Fraction::unreduce(const int m)
{
    m_Numerator*=m;
    m_Denominator*=m;
    return;
}
```

Θα αλλάξει τις τιμές του αντικειμένου
που καλεί την συνάρτηση της κλάσης

```
#include "fraction.h"

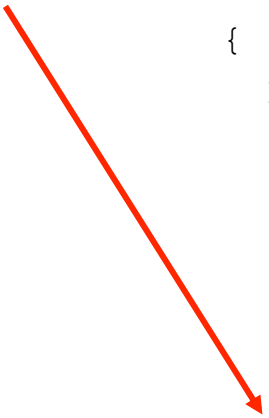
int main()
{
    Fraction f1, f2, f3;
    f1.readin();
    f1.print();
    f2.readin();
    f2.print();
    f2.unreduce(2);
    f2.print();
    f3 = f1.reciprocal();
    f3 = f1 + f2;
    ...
}
```



Χρήση της κλάσης

```
void Fraction::print() {  
    cout<<" "<<m_Numerator  
        <<"/"<<m_Denominator<<" "<<"<br>    return;  
}
```

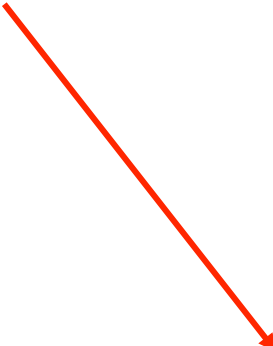
```
#include "fraction.h"  
  
int main()  
{  
    Fraction f1, f2, f3;  
    f1.readin();  
    f1.print();  
    f2.readin();  
    f2.print();  
    f2.unreduce(2);  
    f2.print();  
    f3 = f1.reciprocal();  
    f3 = f1 + f2;  
    ...  
}
```



Χρήση της κλάσης

```
Fraction Fraction::reciprocal() {  
    Fraction returnable;  
    returnable.m_Numerator =  
m_Denominator;  
    returnable.m_Denominator =  
m_Numerator;  
    return returnable;}  
}
```

```
#include "fraction.h"  
  
int main()  
{  
    Fraction f1, f2, f3;  
    f1.readin();  
    f1.print();  
    f2.readin();  
    f2.print();  
    f2.unreduce(2);  
    f2.print();  
    f3 = f1.reciprocal();  
    f3 = f1 + f2;  
    ...  
}
```

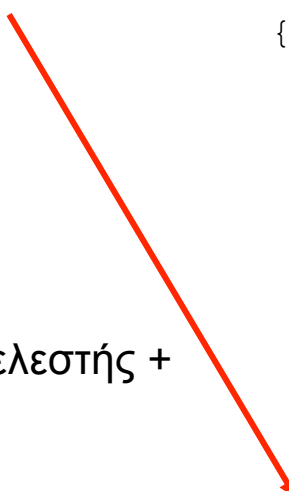


Χρήση της κλάσης

```
??? Fraction::operator+(???)  
{  
    ???  
}
```

Δεν έχει οριστεί για την κλάση ο τελεστής +

```
#include "fraction.h"  
  
int main()  
{  
    Fraction f1, f2, f3;  
    f1.readin();  
    f1.print();  
    f2.readin();  
    f2.print();  
    f2.unreduce(2);  
    f2.print();  
    f3 = f1.reciprocal();  
    f3 = f1 + f2;  
    ...  
}
```



Χρήση της κλάσης

```
??? Fraction::operator=(???)  
{  
    ???  
}
```

Δεν έχει οριστεί για την κλάση ο τελεστής =
αλλά στη C++ υπάρχουν ορισμένοι τελεστές
που ισχύουν σε αντικείμενα που ορίζει ο
χρήστης

Αυτό που θα πραγματοποιηθεί είναι ότι θα
αντιγραφούν οι τιμές των μεταβλητών μελών
από το αρχικό αντικείμενο (f1) στο
αντικείμενο προορισμού (f3)

```
#include "fraction.h"  
  
int main()  
{  
    Fraction f1, f2, f3;  
    f1.readin();  
    f1.print();  
    f2.readin();  
    f2.print();  
    f2.unreduce(2);  
    f2.print();  
    f3 = f1.reciprocal();  
    f3 = f1 + f2;  
    ...  
}
```

