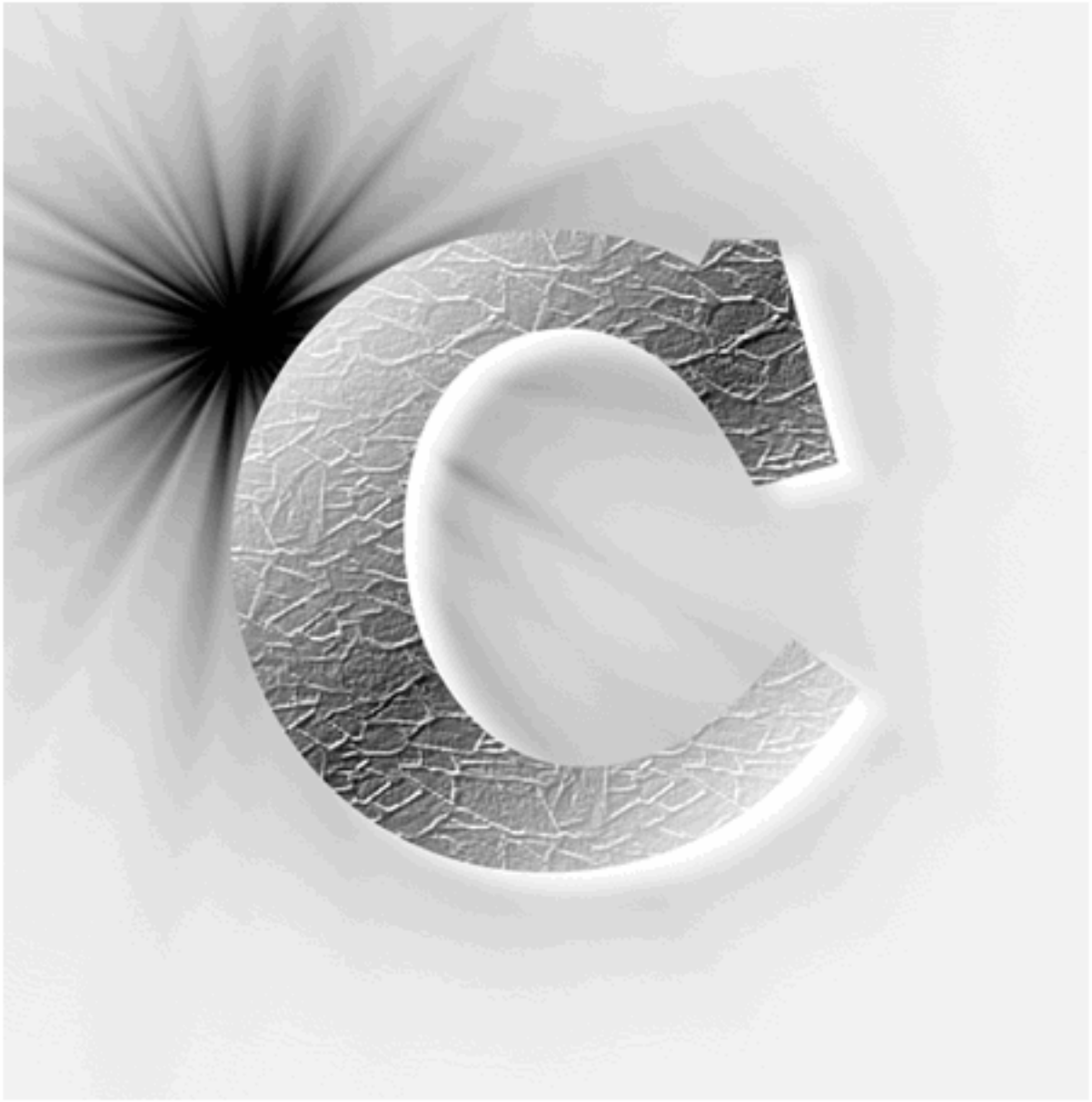


If you can't make it,
you don't know it.

Γιατί προγραμματισμό ;

- Απομυθοποίηση του υπολογιστή.
- Γνωριμία με έναν κατ' εξοχήν αναλυτικό τρόπο σκέψης και επίλυσης προβλημάτων.
- Δυνατότητα επίλυσης ιδίων προβλημάτων
  - Αντικατάσταση της ερώτησης

    «υπάρχει έτοιμο πρόγραμμα που να λύνει αυτό το πρόβλημα ;»

    από την ερώτηση

    «πως λύνεται αυτό το πρόβλημα ;».

Γιατί C ;

- Με τον τρόπο που θα χρησιμοποιηθεί στο μάθημα, ουσιαστικά ***δεν*** θα είναι C [τα χαρακτηριστικά της γλώσσας που την κάνουν προσφιλή στους προγραμματιστές (όπως η συμπαγής και περιεκτική έκφραση) δεν θα τονιστούν].
-

  είναι μια απλή γλώσσα προγραμματισμού με ελάχιστες ενσωματωμένες εντολές (αλλά πολλούς τελεστές).

Γιατί C ;

- Με τον τρόπο που θα χρησιμοποιηθεί στο μάθημα, ουσιαστικά ***δεν*** θα είναι C [τα χαρακτηριστικά της γλώσσας που την κάνουν προσφιλή στους προγραμματιστές (όπως η συμπαγής και περιεκτική έκφραση) δεν θα τονιστούν].
- Στο επίπεδο που θα χρησιμοποιηθεί θα είναι μια απλή γλώσσα προγραμματισμού με ελάχιστες ενσωματωμένες εντολές (αλλά πολλούς τελεστές).

Γιατί C ;

- Με τον τρόπο που θα χρησιμοποιηθεί στο μάθημα, ουσιαστικά ***δεν*** θα είναι C [τα χαρακτηριστικά της γλώσσας που την κάνουν προσφιλή στους προγραμματιστές (όπως η συμπαγής και περιεκτική έκφραση) δεν θα τονιστούν].
- Στο επίπεδο που θα χρησιμοποιηθεί θα είναι μια απλή γλώσσα προγραμματισμού με ελάχιστες ενσωματωμένες εντολές (αλλά πολλούς τελεστές).

Πόσο απλή ;

Γιατί C ;

Εντολές	`for`
	`while`
	`if και if-else`

Συναρτήσεις εισόδου/εξόδου	`printf()` `scanf()`

Τύποι μεταβλητών	`int`
	`float`
	`char`

Γιατί C ;

- Μεταγλωττιστές για C (και μάλιστα δωρεάν) υπάρχουν για το σύνολο των υπολογιστικών συστημάτων που κατά πάσα πιθανότητα θα συναντήσετε (υπάρχουν δωρεάν μεταγλωττιστές ακόμα και για υπολογιστικά συστήματα που ουσιαστικά στερούνται λειτουργικού συστήματος, δείτε για παράδειγμα **`http://www.mingw.org/`**).

Τι δεν θα μάθετε για τη C

- Δεν θα μάθετε τόσα ώστε να μπορείτε να γράψετε ένα πρόγραμμα 2000 γραμμών (?).
- Δεν θα μάθετε τόσα ώστε να μπορείτε να διαβάσετε οποιοδήποτε πρόγραμμα γραμμένο από έμπειρους χρήστες της γλώσσας.

Το πρώτο πρόγραμμα σε C

```
#include <stdio.h>


main()
{
    printf("Hello world.\n");
}
```

Το πρώτο πρόγραμμα σε C

```
#include <stdio.h>


main()
{
    printf("Hello world.\n");
}
```

Είναι βλακώδες.

Το πρώτο πρόγραμμα σε C

```
#include <stdio.h>

main()
{
    printf("Hello world.\n");
}
```

Είναι βλακώδες, αλλά …

Το πρώτο πρόγραμμα σε C

```
#include <stdio.h>


main()
{
    printf("Hello world.\n");
}
```

Επικεφαλίδα προγράμματος : για όλες μας τις εφαρμογές θα είναι η ίδια :

```
#include <stdio.h>
#include <math.h>
```

Το πρώτο πρόγραμμα σε C

```
#include <stdio.h>


main()
{
    printf("Hello world.\n");
}
```

Το κυρίως μέρος του προγράμματος. Η εκτέλεση των εντολών αρχίζει πάντα από τη (συνάρτηση) `main()` η οποία περιλαμβάνει ο,τι περιέχεται ανάμεσα στο `{` και το `}`

Το πρώτο πρόγραμμα σε C

```
#include <stdio.h>


main()
{
    printf("Hello world.\n");
}
```

Το καθ' αυτό πρόγραμμα που στην συγκεκριμένη περίπτωση περιέχει μόνο μια εντολή [η οποία δεν είναι εντολή (της C) αλλά κλήση μιας συνάρτησης (με το όνομα printf) η οποία τυπώνει το όρισμα της (το `Hello world`) στην καθιερωμένη έξοδο].

Τι καταλαβαίνει ένας υπολογιστής από C ?

Τι καταλαβαίνει ένας υπολογιστής από C ?

Τίποτα.

(ούτε λέξη)

Ένα πρόγραμμα σε C είναι για τον υπολογιστή τόσο κατανοητό όσο και ένα e-mail που λάβατε, ή ένας κατάλογος για ψώνια από το supermarket.

Τι καταλαβαίνει ένας υπολογιστής από C ?

Ανάμεσα σε εσάς (που γράφετε ένα πρόγραμμα σε μια υψηλού επιπέδου γλώσσα προγραμματισμού) και το μηχάνημα (το όποιο μπορεί μόνο να εκτελέσει ένα μικρό σύνολο από απλοϊκές εντολές) παρεμβάλλεται ένα άλλο πρόγραμμα,

Ο μεταγλωττιστής της C (C compiler)

ο οποίος με βάση το πρόγραμμα που φτιάξατε, δημιουργεί ένα καινούργιο αρχείο το οποίο περιέχει το ίδιο πρόγραμμα αλλά γραμμένο σε μια γλώσσα τόσο χαμηλού επιπέδου ώστε να είναι άμεσα εκτελέσιμη από τον υπολογιστή σας.

Τι καταλαβαίνει ένας υπολογιστής από C ?

```
cc myprogram.c
```

ή

```
gcc myprogram.c
```

Τι καταλαβαίνει ένας υπολογιστής από C ?

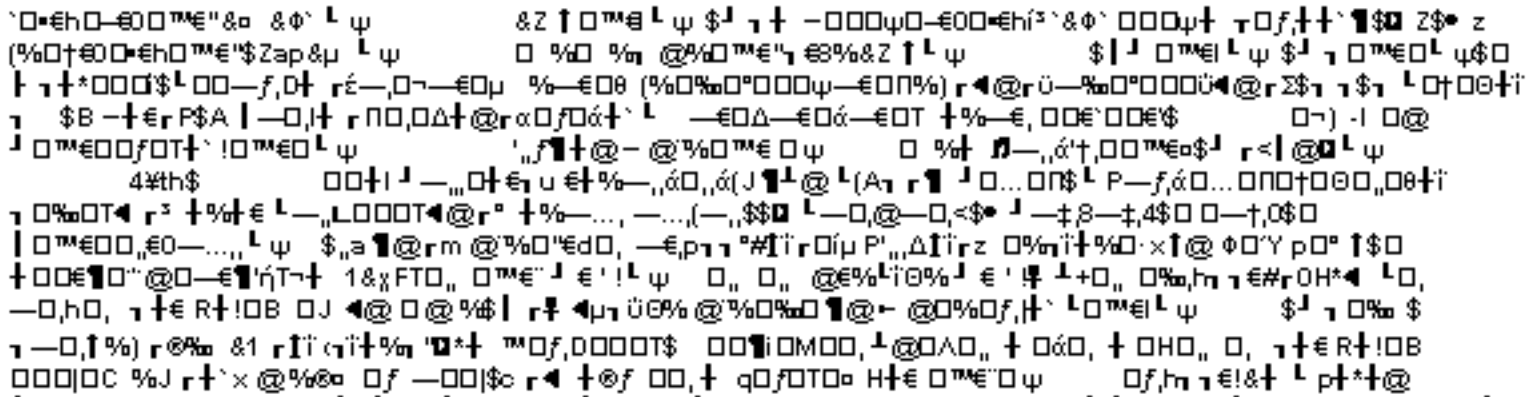
```
cc myprogram.c
```

ή

```
gcc myprogram.c
```

Με βάση το πρόγραμμα σας (που βρίσκεται στο αρχείο myprogram.c), ο compiler (cc ή gcc) δημιουργεί στον τρέχοντα κατάλογο ένα καινούργιο αρχείο (όχι απλού κειμένου) με το όνομα a.out το οποίο μπορείτε να εκτελέσετε με την εντολή `./a.out` ή και απλά `a.out`

Τι καταλαβαίνει ένας υπολογιστής από C ?



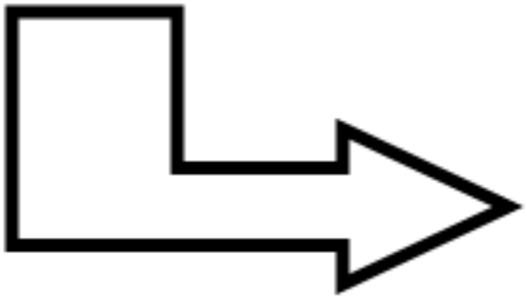
- Ένα πρόγραμμα σε C είναι ακατανόητο για τον υπολογιστή.
- Ένα πρόγραμμα κατανοητό από τον υπολογιστή είναι ακατανόητο για (τους συνηθισμένους) ανθρώπους.

Πως θα μάθετε C ?

Μέσω της επίλυσης πραγματικών προβλημάτων τα οποία από τη φύση τους να απαιτούν τόσους υπολογισμούς ώστε η χρήση υπολογιστικών μηχανών να είναι αναγκαία.

Πως θα μάθετε C ?

Μέσω της επίλυσης πραγματικών προβλημάτων τα οποία από τη φύση τους να απαιτούν τόσους υπολογισμούς ώστε η χρήση υπολογιστικών μηχανών να είναι αναγκαία.



1ο πρόβλημα :
Η μέθοδος Bradford

Η μέθοδος Bradford :

- Είναι μια φασματοσκοπική μέθοδος προσδιορισμού πρωτεϊνικής συγκέντρωσης.
- Στηρίζεται στην αλλαγή του χρώματος μιας χρωστικής (Coomassie Blue G) όταν υπάρχουν μόρια πρωτείνης στο διάλυμα.
- Για τη χρήση της μεθόδου κατασκευάζεται πρότυπη καμπύλη με τη βοήθεια πρωτεϊνικών διαλυμάτων γνωστής συγκέντρωσης : για το κάθε ένα από αυτά μετράμε την οπτική πυκνότητα (στα 595nm).

Η μέθοδος Bradford :

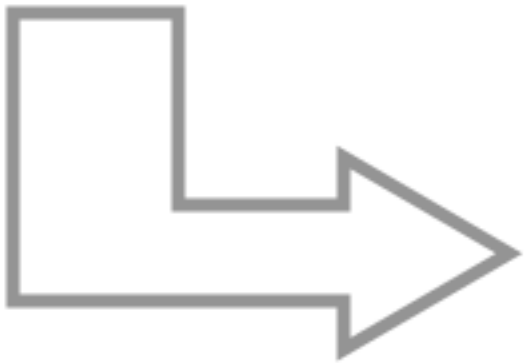
- Για χαμηλές πρωτεϊνικές συγκεντρώσεις τα σημεία της πρότυπης καμπύλης αναμένεται να ανήκουν σε μια ευθεία γραμμή.

Η μέθοδος Bradford :

- Για χαμηλές πρωτεϊνικές συγκεντρώσεις τα σημεία της πρότυπης καμπύλης αναμένεται να ανήκουν σε μια ευθεία γραμμή.
- Αλλά εάν στα αλήθεια κάνετε το πείραμα, είναι σχεδόν βέβαιο ότι τα σημεία δεν θα ανήκουν σε μια ευθεία.

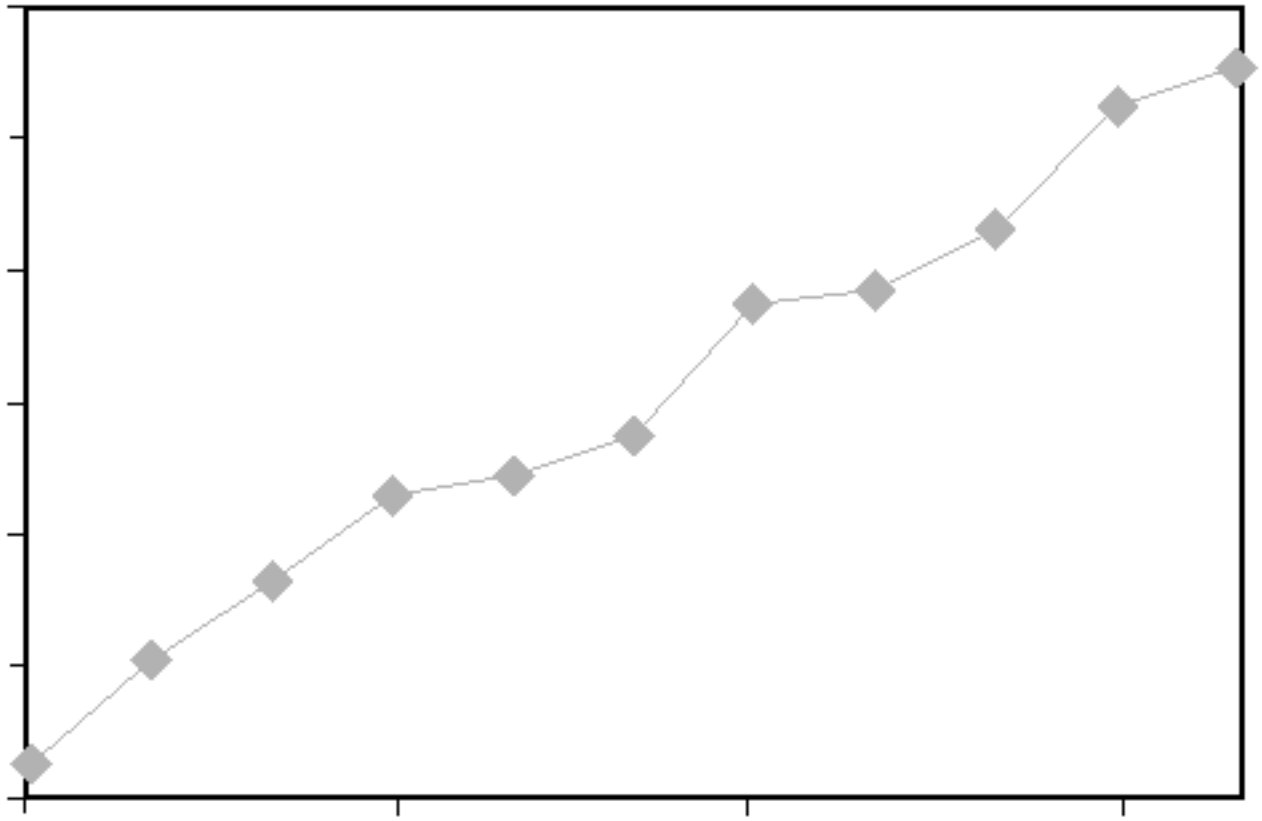
Η μέθοδος Bradford :

- Για χαμηλές πρωτεϊνικές συγκεντρώσεις τα σημεία της πρότυπης καμπύλης αναμένεται να ανήκουν σε μια ευθεία γραμμή.
- Αλλά εάν στα αλήθεια κάνετε το πείραμα, είναι σχεδόν βέβαιο ότι τα σημεία δεν θα ανήκουν σε μια ευθεία.

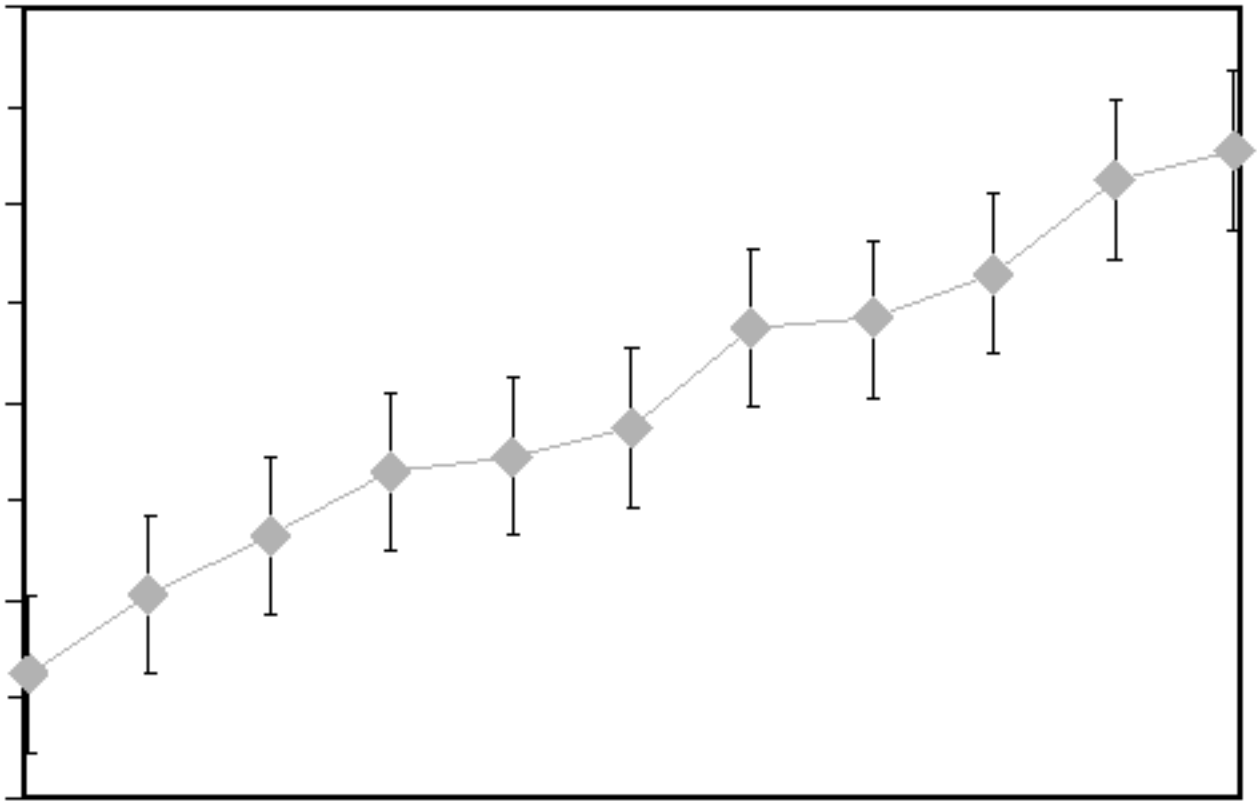


Σφάλματα μετρήσεων

Η μέθοδος Bradford :



Η μέθοδος Bradford :



Ποια είναι (πως υπολογίζουμε)
την πρότυπη ευθεία ;

Η μέθοδος Bradford :

- Λύση : η μέθοδος των ελαχίστων τετραγώνων (least squares method)
  - Υπέθεσε ότι τα σφάλματα στις τιμές της πρωτεϊνικής συγκέντρωσης στα πρότυπα διαλύματα είναι τόσο μικρές ώστε να μπορούν να θεωρηθούν αμελητέες.
  - Υπέθεσε ότι όλες οι μετρούμενες τιμές οπτικής πυκνότητας έχουν παρόμοιες τιμές σφαλμάτων.

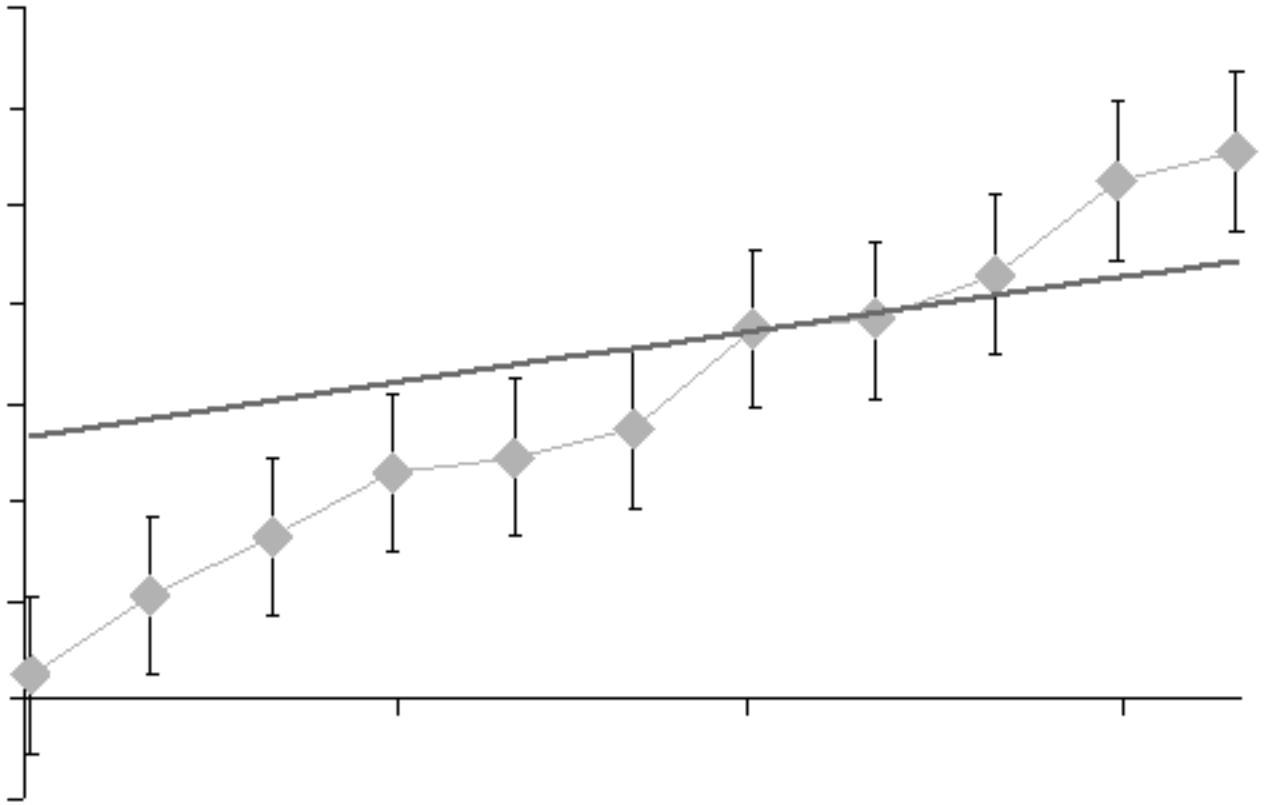
Η μέθοδος Bradford :

- Λύση : η μέθοδος των ελαχίστων τετραγώνων (least squares method)
  - Η ευθεία που θέλουμε είναι αυτή για την οποία το άθροισμα των τετραγώνων των διαφορών ανάμεσα στις παρατηρούμενες και υπολογιζόμενες τιμές της οπτικής πυκνότητας των διαλυμάτων ελαχιστοποιείται.

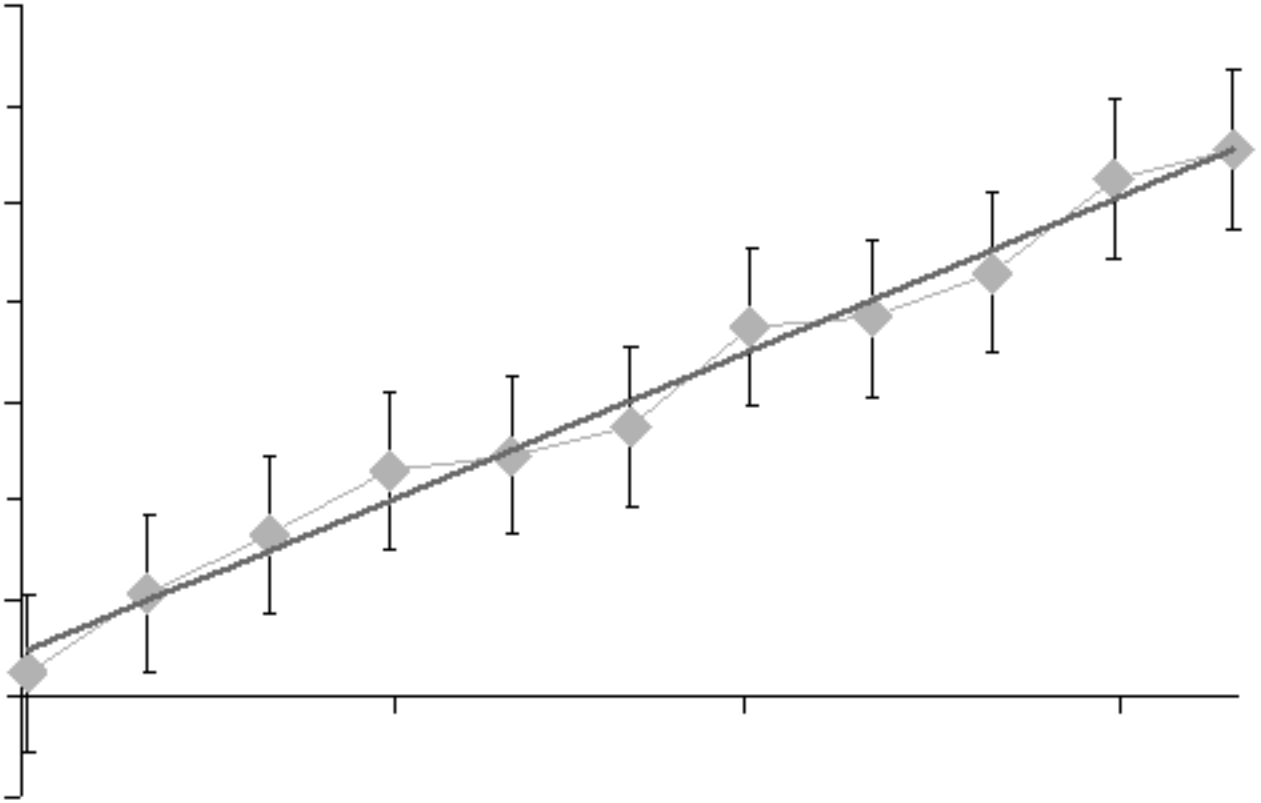
Η μέθοδος Bradford :

Δηλαδή : Έστω μια ευθεία με εξίσωση y=ax+b όπου το x είναι η πρωτεϊνική συγκέντρωση των πρότυπων διαλυμάτων και y είναι η μετρούμενη οπτική πυκνότητα. Για κάθε τιμή του x μπορούμε να βρούμε την τιμή του y που θα περιμέναμε εάν όντως αυτή η εξίσωση ίσχυε. Συνεπώς για κάθε τέτοια ευθεία, μπορούμε να υπολογίσουμε τις διαφορές ανάμεσα στις παρατηρούμενες και τις υπολογιζόμενες τιμές των y για όλα τα x.

Η μέθοδος Bradford :



Η μέθοδος Bradford :



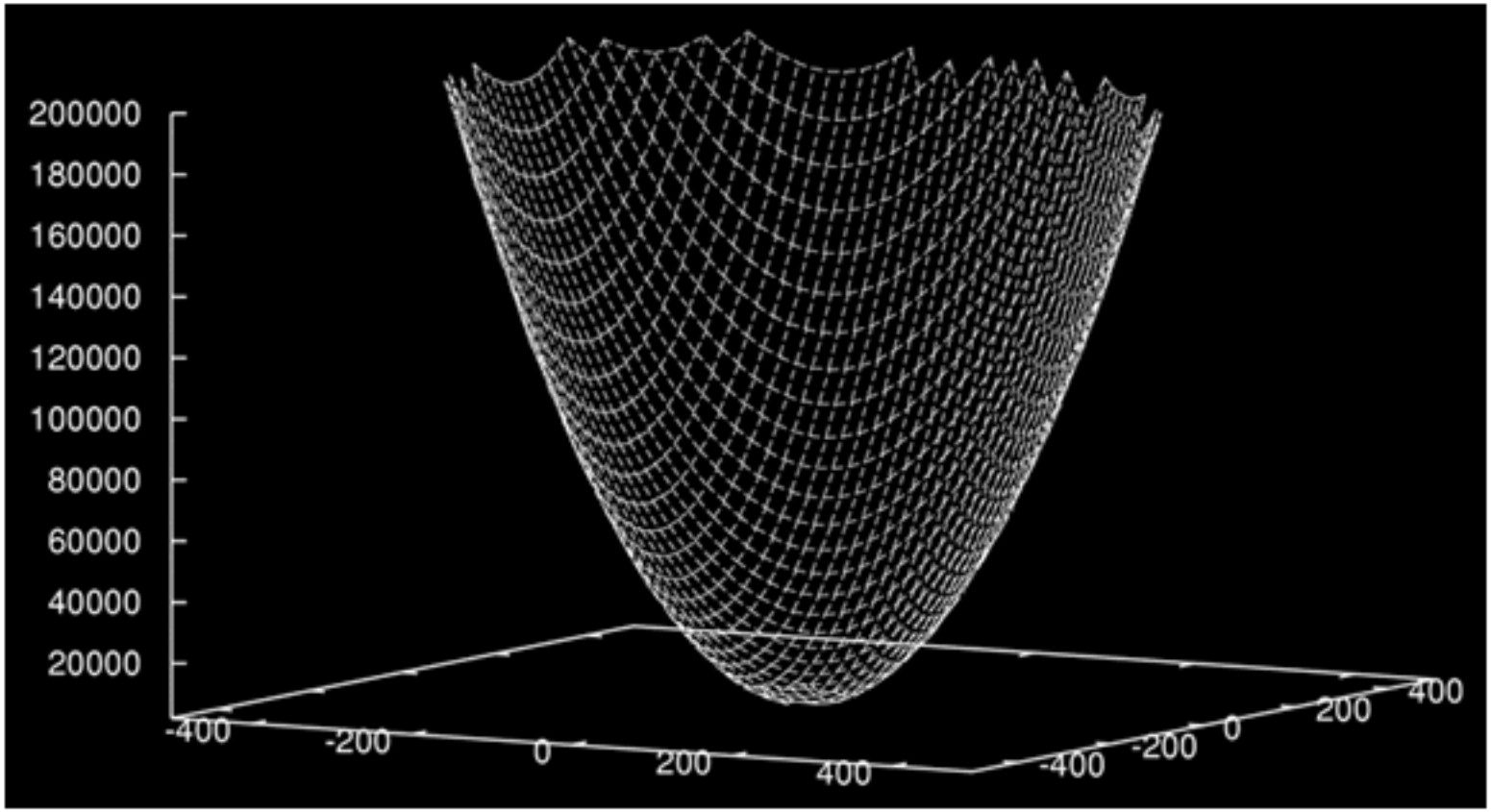
Η μέθοδος Bradford :

$$\Delta = \sum_i [y_i - y_{calc}]^2$$

$$\Delta = \sum_i [y_i - (ax_i + b)]^2$$

$$\ldots$$

Η μέθοδος Bradford :



Η μέθοδος Bradford :

$$a = \frac{N \sum yx - \sum x \sum y}{N \sum x^2 - (\sum x)^2}$$

$$b = \frac{\sum y \sum x^2 - \sum xy \sum x}{N \sum x^2 - (\sum x)^2}$$

Η μέθοδος Bradford :

$$a = \frac{N \sum yx - \sum x \sum y}{N \sum x^2 - (\sum x)^2}$$

$$b = \frac{\sum y \sum x^2 - \sum xy \sum x}{N \sum x^2 - (\sum x)^2}$$

In mathematics you don't understand things. You just get used to them.

Johann von Neumann

Η μέθοδος Bradford :

- Διάβασε τα δεδομένα x_i, y_i (N ζεύγη τιμών).
- Υπολόγισε τις τιμές των αθροισμάτων Σx, Σy, Σxy, και Σx^2.
- Υπολόγισε τα a και b.

Η μέθοδος Bradford :

- Διάβασε τα δεδομένα x_i, y_i (N ζεύγη τιμών).
- Υπολόγισε τις τιμές των αθροισμάτων Σx, Σy, Σxy, και Σx^2.
- Υπολόγισε τα a και b.

Δεν κωδικοποιείται.

Η μέθοδος Bradford :

- Διάβασε τα δεδομένα x_i, y_i (N ζεύγη τιμών).
- Υπολόγισε τις τιμές των αθροισμάτων Σx, Σy, Σxy, και Σx^2.
- Υπολόγισε τα a και b.

Δεν κωδικοποιείται.

- Τι σημαίνει «διάβασε» ;
- Τι μορφή έχουν τα δεδομένα (πως τα «διάβασα», που και πως τα τοποθέτησα, πώς να τα ανακτήσω για να τα χρησιμοποιήσω ;).

Η μέθοδος Bradford :

- Διάβασε τα δεδομένα x_i, y_i (N ζεύγη τιμών).
- Υπολόγισε τις τιμές των αθροισμάτων Σx, Σy, Σxy, και Σx^2.
- Υπολόγισε τα a και b.

Δεν κωδικοποιείται.

- Τι σημαίνει «υπολόγισε το άθροισμα» και, χειρότερα, τι είναι αυτά τα «Σ» ;

Η μέθοδος Bradford :

- Διάβασε τα δεδομένα x_i, y_i (N ζεύγη τιμών).
- Υπολόγισε τις τιμές των αθροισμάτων Σx, Σy, Σxy, και Σx^2.
- Υπολόγισε τα a και b.

Σχεδόν κωδικοποιείται …

Υπολόγισε τα a και b :

- Υποθέστε ότι έχουμε την δυνατότητα να ορίσουμε ονόματα μεταβλητών (που θα χρησιμοποιήσουμε στο πρόγραμμα μας), οι οποίες να μπορούν να αποθηκεύσουν (με περιορισμένη ακρίβεια) την τιμή ενός ρητού αριθμού.
- Υποθέστε ότι έχουμε την δυνατότητα να ορίσουμε ονόματα μεταβλητών (που θα χρησιμοποιήσουμε στο πρόγραμμα μας), οι οποίες να μπορούν να αποθηκεύσουν (με ακρίβεια) την τιμή ενός ακέραιου αριθμού.

Υπολόγισε τα a και b :

- Εάν έχουμε ορίσει τέσσερις μεταβλητές ρητού τύπου με ονόματα **`Sx, Sy, Sxy, Sx2`**, οι οποίες να περιέχουν τις τιμές των αθροισμάτων Σx, Σy, Σxy, και Σx^2, και μια μεταβλητή ακεραίου τύπου (με το όνομα **`points`**) που να περιέχει το N (δηλ. τον αριθμό των σημείων), τότε το μόνο που μας λείπει είναι να ορίσουμε δυο νέες μεταβλητές (με ονόματα **`a`** και **`b`**) και να τους αναθέσουμε τιμές με απλές αριθμητικές πράξεις.

Υπολόγισε τα a και b :

```
#include <stdio.h>
#include <math.h>

main()
{
float      Sxy, Sx, Sy, Sx2;
float      a, b;
int        points;
... ... ... ... ...
```

Υπολογισμός των Sxy, Sx, Sy, Sx2

```
... ... ... ... ...
a = (points * Sxy - Sx * Sy) / (points * Sx2 - Sx *Sx);
b = ((Sy * Sx2)-(Sxy * Sx)) / ((points * Sx2)-(Sx * Sx));
... ... ... ... ...
}
```

Υπολόγισε τα a και b :

```
#include <stdio.h>
#include <math.h>

main()
{
float       Sxy, Sx, Sy, Sx2;
float       a, b;
int         points;
... ... ... ... ...
```

$$a = \frac{N\sum yx - \sum x \sum y}{N\sum x^2 - (\sum x)^2}$$

$$b = \frac{\sum y \sum x^2 - \sum xy \sum x}{N\sum x^2 - (\sum x)^2}$$

Υπολογισμός των Sxy, Sx, Sy, Sx2

```
... ... ... ... ...
a = (points * Sxy - Sx * Sy) / (points * Sx2 - Sx *Sx);
b = ((Sy * Sx2)-(Sxy * Sx)) / ((points * Sx2)-(Sx * Sx));
... ... ... ... ...
}
```

Υπολόγισε τα a και b :

```
#include <stdio.h>
#include <math.h>

main()
{
float       Sxy, Sx, Sy, Sx2;
float       a, b;
int         points;
... ... ... ... ...
```

$$a = \frac{N\sum yx - \sum x \sum y}{N\sum x^2 - (\sum x)^2}$$

$$b = \frac{\sum y \sum x^2 - \sum xy \sum x}{N\sum x^2 - (\sum x)^2}$$

Υπολογισμός των Sxy, Sx, Sy, Sx2

```
... ... ... ... ...
a = (points * Sxy - Sx * Sy) / (points * Sx2 - Sx *Sx);
b = ((Sy * Sx2)-(Sxy * Sx)) / ((points * Sx2)-(Sx * Sx));
... ... ... ... ...
}
```

Υπολόγισε τα αθροίσματα :

- Για να μπορέσουμε να υπολογίσουμε τα αθροίσματα θα πρέπει να έχουμε πρόσβαση στα δεδομένα (τα N ζεύγη x_i, y_i).
- Υποθέστε ότι έχουμε την δυνατότητα να ορίσουμε πίνακες καθορισμένων διαστάσεων (που θα χρησιμοποιήσουμε στο πρόγραμμα μας), οι οποίοι να μπορούν να αποθηκεύσουν (με περιορισμένη ακρίβεια) τιμές ρητών αριθμών.

Υπολόγισε τα αθροίσματα :

```
#include <stdio.h>
#include <math.h>

main()
{
int         index1;
float       x[1000];
float       y[1000];
float       test[100][2];
 ... ... ... ... ...
x[0] = 132.3;
x[1] = 3455.6;
x[999] = 12776.6;
 ... ... ... ... ...
index1 = 355;
y[index1] = 3429.9;
 ... ... ... ... ...
test[34][1] = 435.45;
 ... ... ... ... ...
```

Υπολόγισε τα αθροίσματα :

```
#include <stdio.h>
#include <math.h>

main()
{
int         index1;
float       x[1000];
float       y[1000];
float       test[100][2];
 ... ... ... ... ...
x[0] = 132.3;
x[1] = 3455.6;
x[999] = 12776.6;
 ... ... ... ... ...
index1 = 355;
y[index1] = 3429.9;
 ... ... ... ... ...
test[34][1] = 435.45;
 ... ... ... ... ...
```

Προφανώς, αυτός ο τρόπος εισαγωγής δεδομένων δεν χρησιμοποιείται στην πράξη (γιατί κάθε φορά που θα άλλαζαν τα δεδομένα θα έπρεπε να ξαναγράψουμε το πρόγραμμα).

Υπολόγισε τα αθροίσματα :

- Εάν λοιπόν τα δεδομένα είναι διαθέσιμα με αυτή τη μορφή, τότε είμαστε σε θέση να γίνουμε πιο σαφής σε σχέση με το τι θέλουμε να κάνουμε. Για παράδειγμα, ο υπολογισμός του Σx_i, θα μπορούσε να περιλαμβάνει τα εξής βήματα :
  - Ξεκινάμε με αρχική τιμή αθροίσματος Sx=0.
  - Για κάθε ζεύγος τιμών που έχουμε (από το πρώτο μέχρι το N-οστό) κάνουμε τα εξής :
    - **Πρόσθεσε την τιμή του x_i (που ανήκει στο i ζεύγος τιμών) στην τρέχουσα τιμή του Sx.**

Υπολόγισε τα αθροίσματα :

- Εάν λοιπόν τα δεδομένα είναι διαθέσιμα με αυτή τη μορφή, τότε είμαστε σε θέση να γίνουμε πιο σαφής σε σχέση με το τι θέλουμε να κάνουμε. Για παράδειγμα, ο υπολογισμός του Σx_i, θα μπορούσε να περιλαμβάνει τα εξής βήματα :
  - Ξεκινάμε με αρχική τιμή αθροίσματος Sx=0.
  - Για κάθε ζεύγος τιμών που έχουμε (από το πρώτο μέχρι το N-οστό) κάνουμε τα εξής :
    - **Πρόσθεσε την τιμή του x_i (που ανήκει στο i ζεύγος τιμών) στην τρέχουσα τιμή του Sx.**

Η εντολή for :

```
for ( i=0 ; i < points ; i=i+1 )
        {
                ΕΝΤΟΛΕΣ . . .
                                        }
```

- Δώσε στην μεταβλητή **i** την τιμή **i=0** και εάν **i<points** άρχισε να εκτελείς τις εντολές που βρίσκονται ανάμεσα στις δυο αγκύλες { και }.
- Αφού εκτελεστούν όλες οι εντολές που βρίσκονται μέσα στις αγκύλες, αύξησε την τιμή του **i** κατά μια μονάδα.
- Εάν **i<points** τότε άρχισε να εκτελείς πάλι τις εντολές ανάμεσα στις αγκύλες (χρησιμοποιώντας, βέβαια, την καινούργια τιμή του **i**). Εάν **i>=points** τότε συνέχισε με τις εντολές που βρίσκονται μετά το ζεύγος των αγκυλών.

Η εντολή for :

```
for ( i=0 ; i < points ; i++ )
        {
                ΕΝΤΟΛΕΣ . . .
                                        }
```

- Δώσε στην μεταβλητή **i** την τιμή **i=0** και εάν **i<points** άρχισε να εκτελείς τις εντολές που βρίσκονται ανάμεσα στις δυο αγκύλες { και }.
- Αφού εκτελεστούν όλες οι εντολές που βρίσκονται μέσα στις αγκύλες, αύξησε την τιμή του **i** κατά μια μονάδα.
- Εάν **i<points** τότε άρχισε να εκτελείς πάλι τις εντολές ανάμεσα στις αγκύλες (χρησιμοποιώντας, βέβαια, την καινούργια τιμή του **i**). Εάν **i>=points** τότε συνέχισε με τις εντολές που βρίσκονται μετά το ζεύγος των αγκυλών.

Η εντολή for :

```
float  x;


for ( x=0.0 ; x < 5.0 ; x=x+1.0 )
        {
                ΕΝΤΟΛΕΣ . . .
                                        }
```

Όλες οι εντολές που βρίσκονται ανάμεσα στις αγκύλες θα εκτελεστούν πέντε φορές, κάθε φορά με διαφορετική τιμή της (ρητού τύπου) μεταβλητής x (τιμές του x : 0.0, 1.0, 2.0, 3.0 και 4.0).

Η εντολή for :

```
float  x;

for ( x=0.0 ; x <= 5.0 ; x += 1.0 )
       {
              ΕΝΤΟΛΕΣ . . .
                                  }
```

Όλες οι εντολές που βρίσκονται ανάμεσα στις αγκύλες θα εκτελεστούν έξι φορές, κάθε φορά με διαφορετική τιμή της μεταβλητής x (τιμές του x : 0.0, 1.0, 2.0, 3.0, 4.0 και 5.0).

Υπολόγισε τα αθροίσματα :

- Εάν λοιπόν τα δεδομένα είναι διαθέσιμα με αυτή τη μορφή, τότε είμαστε σε θέση να γίνουμε πιο σαφής σε σχέση με το τι θέλουμε να κάνουμε. Για παράδειγμα, ο υπολογισμός του Σx_i, θα μπορούσε να περιλαμβάνει τα εξής βήματα :
  - Ξεκινάμε με αρχική τιμή αθροίσματος Sx=0.
  - Για κάθε ζεύγος τιμών που έχουμε (από το πρώτο μέχρι το N-οστό) κάνουμε τα εξής :
    - **Πρόσθεσε την τιμή του x_i (που ανήκει στο i ζεύγος τιμών) στην τρέχουσα τιμή του Sx.**

Υπολόγισε τα αθροίσματα :

```
... ... ... ... ...
float       Sx;
float       x[1000];
int         points;
int         i;

... ... ... ... ...
Sx = 0.0;
for ( i=0 ; i < points ; i = i + 1 )
    {
            Sx = Sx + x[i];
                                }

... ... ... ... ...
```

Υπολόγισε τα αθροίσματα :

```
... ... ... ... ...
float       Sx;
float       x[1000];
int         points;
int         i;

... ... ... ... ...
Sx = 0.0;
for ( i=0 ; i < points ; i++ )
    {
          Sx = Sx + x[i];
                                  }

... ... ... ... ...
```

Υπολόγισε τα αθροίσματα :

```
... ... ... ... ...
float       Sx;
float       x[1000];
int         points;
int         i;

... ... ... ... ...
Sx = 0.0;
for ( i=0 ; i < points ; i++ )
    {
          Sx += x[i];
                                  }

... ... ... ... ...
```

Υπολόγισε τα αθροίσματα :

```
. . . . . .
float       x[5000];
float       y[5000];
int         points;
float       Sxy, Sx, Sy, Sx2;
int i;
. . . . . .
Sxy = 0.0;
Sx  = 0.0;
Sy  = 0.0;
Sx2 = 0.0;
for ( i=0 ; i < points ; i++ )
    {
            Sx = Sx + x[i];
            Sy = Sy + y[i];
            Sx2 += x[i]*x[i];
            Sxy += x[i]*y[i];
                                }
```

Πόσο κοντά είμαστε ;

Έχει πολύ ακόμα ;

```c
#include <stdio.h>
#include <math.h>

main()
{
    float x[5000];
    float y[5000];
    int   points;
    float val1, val2;
    float Sxy, Sx, Sy, Sx2;
    float a, b;
    int   i;

    points = 0;
    while ( scanf("%f %f", &val1, &val2) == 2 )
          {
                x[points] = val1;
                y[points] = val2;
                points++;
                                }
    Sxy = 0.0;
    Sx  = 0.0;
    Sy  = 0.0;
    Sx2 = 0.0;
    for ( i=0 ; i < points ; i++ )
          {
                Sx = Sx + x[i];
                Sy = Sy + y[i];
                Sx2 += x[i]*x[i];
                Sxy += x[i]*y[i];
                                    }
    a = (points * Sxy - Sx * Sy) / (points * Sx2 - Sx * Sx);
    b = ((Sy * Sx2)-(Sxy * Sx)) / ((points * Sx2)-(Sx * Sx));

    printf(" %f %f\n", a, b );
}
```

```c
#include <stdio.h>
#include <math.h>

main()
{
    float x[5000];
    float y[5000];
    int   points;
    float val1, val2;
    float Sxy, Sx, Sy, Sx2;
    float a, b;
    int   i;

    points = 0;
    while ( scanf("%f %f", &val1, &val2) == 2 )
          {
                x[points] = val1;
                y[points] = val2;
                points++;
                                }
    Sxy = 0.0;
    Sx  = 0.0;
    Sy  = 0.0;
    Sx2 = 0.0;
    for ( i=0 ; i < points ; i++ )
          {
                Sx = Sx + x[i];
                Sy = Sy + y[i];
                Sx2 += x[i]*x[i];
                Sxy += x[i]*y[i];
                                    }
    a = (points * Sxy - Sx * Sy) / (points * Sx2 - Sx * Sx);
    b = ((Sy * Sx2)-(Sxy * Sx)) / ((points * Sx2)-(Sx * Sx));

    printf(" %f %f\n", a, b );
}
```

Μιλώντας στον κόσμο : η συνάρτηση printf()

- Η printf() παρέχει την δυνατότητα σε ένα πρόγραμμα να τυπώσει μια ακολουθία χαρακτήρων στην καθιερωμένη έξοδο (standard output), η οποία απουσία επανακαθορισμού είναι το τερματικό (ενώ μέσω των **>** και **|** μπορεί είναι ένα αρχείο ή η είσοδος ενός άλλου προγράμματος).
- Το πλήρες συντακτικό της printf() είναι διαθέσιμο μέσω της εντολής (από το unix shell) **`man printf`**. Εμείς θα χρησιμοποιήσουμε μόνο ένα υποσύνολο αυτού.

Μιλώντας στον κόσμο : η συνάρτηση printf()

```
printf(" Hello world\n");
```

```
 Hello world
```

```
int a;
a = 3123;
printf("The value of a is %d \n", a );
```

```
The value of a is 3123
```

Μιλώντας στον κόσμο : η συνάρτηση printf()

```
float x;
x = 1.05670;
printf("The value of x is %f \n", x );

The value of x is 1.0567
```

Μιλώντας στον κόσμο : η συνάρτηση printf()

```
int   r;
float x;
x = 34.05;
r = 95;
printf("The values are %d and %f \n", r, x );

The values are 95 and 34.05
```

Μιλώντας στον κόσμο : η συνάρτηση printf()

```
printf("The values are %d and %f \n", r, x );
```

```
printf("The values are %d and %f \n", r, x );
```

```
printf("The values are %d and %f \n", r, x );
```

Υπολογισμός αθροισμάτων..

```
. . . . . .
float       x[5000];
float       y[5000];
int         points;
float       Sxy, Sx, Sy, Sx2;
int i;
. . . . . .
Sxy = 0.0;
Sx  = 0.0;
Sy  = 0.0;
Sx2 = 0.0;
for ( i=0 ; i < points ; i++ )
    {
            Sx = Sx + x[i];
            Sy = Sy + y[i];
            Sx2 += x[i]*x[i];
            Sxy += x[i]*y[i];
                                }
```

... αλλά τι γίνεται εάν έχουμε μόνο ένα σημείο ;

```
. . . . . .
float  x[5000];
float  y[5000];
int    points;
float  Sxy, Sx, Sy, Sx2;
int    i;
. . . . . .
Sxy = 0.0;
Sx  = 0.0;
Sy  = 0.0;
Sx2 = 0.0;
for ( i=0 ; i < points ; i++ )
     {
            Sx = Sx + x[i];
            Sy = Sy + y[i];
            Sx2 += x[i]*x[i];
            Sxy += x[i]*y[i];
                                    }
```

Η εντολή if :

```
if ( points < 2 )
        {
                ΕΝΤΟΛΕΣ . . .
                                        }
```

- Εάν η τιμή της μεταβλητής **points** είναι μικρότερη από το 2 τότε άρχισε να εκτελείς τις εντολές ανάμεσα στις αγκύλες που ακολουθούν την εντολή if . Αφού τις εκτελέσεις, συνέχισε κανονικά με την εκτέλεση του υπόλοιπου προγράμματος.
- Εάν η τιμή της μεταβλητής **points** είναι μεγαλύτερη ή ίση του 2, ***αγνόησε (μην εκτελείς)*** όσες εντολές είναι ανάμεσα στις αγκύλες που ακολουθούν την εντολή if .

Το ζεύγος εντολών if-else :

```
if ( points < 2 )
        {
                ΕΝΤΟΛΕΣ . . .
                                        }
else
        {
                ΕΝΤΟΛΕΣ . . .
                                        }
```

- Εάν η τιμή της μεταβλητής **points** είναι μικρότερη από το 2 τότε άρχισε να εκτελείς τις εντολές ανάμεσα στις αγκύλες που ακολουθούν την εντολή if . Αφού τις εκτελέσεις, ***αγνόησε (μην εκτελείς)*** την εντολή else και όλες τις εντολές που περιέχονται ανάμεσα στο ζεύγος των αγκυλών που την ακολουθούν.

Το ζεύγος εντολών if-else :

```
if ( points < 2 )
        {
                ΕΝΤΟΛΕΣ . . .
                                        }
else
        {
                ΕΝΤΟΛΕΣ . . .
                                        }
```

- Εάν η τιμή της μεταβλητής **points** είναι μεγαλύτερη ή ίση του 2, ***αγνόησε (μην εκτελείς)*** όσες εντολές είναι ανάμεσα στις αγκύλες που ακολουθούν την εντολή if ***αλλά εκτέλεσε*** όλες τις εντολές που περιέχονται ανάμεσα στο ζεύγος των αγκυλών που ακολουθούν την εντολή else .

... αλλά τι γίνεται εάν έχουμε μόνο ένα σημείο ;

```
. . . . . .
float  x[5000];
float  y[5000];
int    points;
float  Sxy, Sx, Sy, Sx2;
int    i;
. . . . . .

Sxy = 0.0;
Sx  = 0.0;
Sy  = 0.0;
Sx2 = 0.0;
for ( i=0 ; i < points ; i++ )
     {
             Sx = Sx + x[i];
             Sy = Sy + y[i];
             Sx2 += x[i]*x[i];
             Sxy += x[i]*y[i];
                                    }
```

… αλλά τι γίνεται εάν έχουμε μόνο ένα σημείο ;

```
. . . . . .
float   x[5000];
float   y[5000];
int     points;
float   Sxy, Sx, Sy, Sx2;
int     i;
. . . . . .
```

```
if ( points < 2)
  {
    printf("Nonsense\n");
    exit( 1 );
                }
```

```
Sxy = 0.0;
Sx  = 0.0;
Sy  = 0.0;
Sx2 = 0.0;
for ( i=0 ; i < points ; i++ )
     {
            Sx = Sx + x[i];
            Sy = Sy + y[i];
            Sx2 += x[i]*x[i];
            Sxy += x[i]*y[i];
                                }
```