L3 et Magistère 1ère année, Physique fondamentale, année 2024-2025

Travaux dirigés de langage C appliqué à la physique

— **Textes d'exercices de TD (n°1 à 8)**

1. Types entier et réel, priorité des opérations
2. Limitations des valeurs entières et réelles en grandeur et en précision
   Diagnostics en cas d'opérations illégales
3. Tests et boucles
4. Fonctions
5. Pointeurs
6. Tableaux dynamiques
7. Générateur aléatoire
8. Équations différentielles

— **L'éditeur** *emacs*

— **Commandes** *Linux* **de base**

(Consulter les mises à jour sur le site : `https://hebergement.universite-paris-saclay.fr/mpo-informatique/`)

1. Types entier et réel, priorité des opérations

Rappels :

- Ouvrir le site du cours avec les énoncés des TD, les notes de cours, et plein d'autres informations utiles :

  site
- Il sera utile de créer un sous-répertoire nommé *TD1* (avec la commande *mkdir TD1*) et s'y placer (avec la commande *cd TD1*) pour faire ce TD.
- Ouvrir l'éditeur *emacs* :

  emacs prog1.cpp &
- En C il y a distinction entre minuscules et majuscules.
- Pour afficher à l'écran de la façon la plus simple, placer en tête du fichier contenant le programme :

  ```
  #include<iostream>
  using namespace std;
  ```

  et utiliser, par exemple dans le cas d'une variable k :

  ```
  cout << k << endl;
  ```
- Pour compiler et exécuter en une seule commande :

  ccc nom_de_fichier.cpp

  (Alternativement, il faut d'abord compiler : *g++ -lm nom_de_fichier.cpp* puis exécuter : *./a.out*)

Prévoir le résultat des quatre programmes C dont le texte est donné ci-dessous, puis écrire chacun de ces programmes **dans un fichier propre**, le faire exécuter et comparer au résultat prévu. Seul le programme 1 doit être tapé intégralement. A partir du programme 2, il suffit d'utiliser la copie de fichiers et de ne taper que ce qui doit être modifié.

Types entier et réel

Ici on montre les règles qu'il faut suivre pour manier les constantes ou les variables entières et réelles dans les expressions arithmétiques et algébriques.

Programme 1

```
#include<iostream>
using namespace std;
int main() {
  cout << 2/3 << " " << 2./3 << " " << 2/3. << " " << 2./3. << endl;
  return 0;
}
```

Programme 2

```
#include<iostream>
using namespace std;
int main() {
  int i = 2, j = 3, k ;
  double x = 2., q ;
  cout << i/j << " " << x/j << endl;
  k = x/j ; q = x/j ;
  cout << k << " " << q << endl;
  return 0;
}
```

Comment obtenir 0.6666667 pour les quatre quantités calculées ?

Règles de priorité des opérations

Programme 3

```
#include<iostream>
using namespace std;
int main() {
  cout << (2.*3.)/4. << " " << 2.*(3./4.) << " " << 2.*3./4. << endl;
  cout << (2./3.)*4. << " " << 2./(3.*4.) << " " << 2./3./4. << endl;
  cout << (2.+3.)/4. << " " << 2.+3./4. << endl;
  return 0;
}
```

Mélange de types

Programme 4

```
#include<iostream>
using namespace std;
int main() {
  cout << (2.*3)/4 << " " << 2.*(3/4) << " " << 2.*3/4 << endl;
  return 0;
}
```

Usage des parenthèses et des variables intermédiaires

$***$ Paquet d'onde $***$

La densité de probabilité d'un paquet d'ondes gaussien libre en mécanique quantique est :

$$\rho(x) = \sqrt{\frac{2}{\pi}} \frac{1}{a\sqrt{1+\frac{4\hbar^2 t^2}{m^2 a^4}}} \exp\left[-\frac{2\left(x-\frac{\hbar k_0}{m}t\right)^2}{a^2\left(1+\frac{4\hbar^2 t^2}{m^2 a^4}\right)}\right]$$

Prendre telles quelles les valeurs numériques suivantes qui sont exprimées dans un système d'unités (fermi, Mev) approprié au cas d'une particule telle que l'électron :

$a = 10^5$ fm (paramètre déterminant la largeur du paquet d'ondes à $t = 0$ avec 1 fm= 1 Fermi= 10^{-15} mètre)
$\hbar = 197$ MeV.fm.c^{-1} (constante de Planck avec MeV = Méga électron-volt)
$t = 10^5$ fm.c^{-1} (instant considéré, c = vitesse de la lumière)
$m = 0.5$ MeV.c^{-2} (masse de la particule)
$x = 10^5$ fm (position à laquelle on calcule la densité de probabilité)
$k_0 = 5.10^{-4}$ fm^{-1} (paramètre déterminant la vitesse $\hbar k_0/m$ du centre du paquet d'ondes)

Calculer $\rho(x)$ dans les unités indiquées. Ne pas oublier d'utiliser des variables intermédiaires pour simplifier l'expression finale (comme par exemple $u = 2\hbar t/(ma^2)$).

Remarques

Il peut être utile de laisser des parenthèses superflues si elles rendent l'expression plus lisible.

Quelle est l'unité de $\rho(x)$?

2. Limitations des valeurs entières et réelles en grandeur et en précision

Diagnostics en cas d'opérations illégales

Préambule :

Créer dans le répertoire initial un sous-répertoire nommé *TD2* et s'y placer pour faire ce TD.
(Si vous ne l'avez pas fait la semaine dernière : créer d'abord un sous-répertoire nommé *TD1* et déplacer dans *TD1* tous les fichiers déjà écrits pour faire le TD n°1 (se reporter à la partie *Linux* du polycopié de TD).)

Remarques

Les limitations et diagnostics que l'on va observer ici se retrouvent sur tous les ordinateurs mais avec des variantes selon le type de machine. Ce qui est étudié ici n'est donc valable, en détail, que pour les ordinateurs que nous utilisons.

Pour écrire tous les programmes qui suivent utiliser au maximum la copie (de lignes dans un fichier et d'un fichier dans un autre).

Résumer les résultats produits par les programmes proposés dans le tableau donné à la fin de ce chapitre de TD.

Limitations en grandeur et précision

Pour les entiers

Les entiers de type `int` sont écrits sur 4 octets (32 bits), ceux de type `long int` sur 8 octets (64 bits).
Un bit est réservé au signe, les autres aux chiffres.
Pour les `int` ne peuvent donc être écrits que les entiers compris entre -2^{31}=-2 147 483 648 et 2^{31}-1=2 147 483 647 inclus.
Pour les `long int` ne peuvent être écrits que les entiers compris entre -2^{63}=-9 223 372 036 854 775 808 et 2^{63}-1=9 223 372 036 854 775 807 inclus.
Écrire le programme suivant et remarquer le résultat produit.

```
#include<iostream>
using namespace std;
int main() {
  int m;
  m = 2147483646;
  cout << m << " " << m+1 << " " << m+2 << " " << m+3 << endl;
  long int n;
  n = 9223372036854775806;
  cout << n << " " << n+1 << " " << n+2 << " " << n+3 << endl;
  return 0;
}
```

À part cette limitation les opérations arithmétiques entre entiers sont exactes (compte tenu de ce que la division donne la partie entière du résultat).

Pour les réels

Pour les réels il y a également des limites et de plus, un nombre n'est en général écrit que de façon approchée, avec un nombre de chiffres significatifs exacts limité. Lors d'une opération algébrique entre réels le résultat sera lui-même arrondi ce qui entraîne une nouvelle approximation. Dans certains cas les erreurs peuvent s'accumuler au cours des calculs.
Les réels de type `double` sont écrits sur 8 octets (= 64 bits).

Pour les réels il existe aussi le type `float` (sur 4 octets) et le type `long double` (sur 16 octets) mais nous ne considérerons ici que le type `double`.
Le tableau suivant résume les limites et la précision des réels de type `double` pour les ordinateurs utilisés :

Type	Nb de bits signe	Nb de bits exposant	Nb de bits chiffres significatifs	Plus petite valeur absolue représentable	Plus grande valeur absolue représentable	Nb minimum de chiffres significatifs exacts en décimale
double	1	11	53	$\simeq$ 2.225e-308 ($\simeq$ 4.941e-324)	$\simeq$ 1.798e308	15

Rappel : 1.798e-308 est la notation informatique pour $1.798 \cdot 10^{-308}$. Pour les nombres sub-normaux (limites entre parenthèses) il n'y a pas de diagnostic, mais le nombre de chiffres significatifs exacts est moindre (voir chapitre 3 du poly de cours).
Pour les valeurs absolues inférieures à la plus petite valeur sub-normale ou supérieures à la plus grande valeur normale il y a un diagnostic de dépassement.

Contrairement au cas des entiers il y a donc un diagnostic de dépassement pour les réels mais l'exécution se poursuit.

Ecrire et faire exécuter le programme suivant :

```
#include<iostream>
using namespace std;
int main() {
  double x = 1.e-308, y = 1.e-10;
  cout << "1.e-308 = " << x << endl;
  x = x*y;
  cout << "1.e-318 = " << x << endl;
  x = x*y;
  cout << "1.e-328 = " << x << endl;
  return 0;
}
```

puis :

```
#include<iostream>
using namespace std;
int main() {
  double x = 1.e308, y = 1.e10;
  cout << "1.e308 = " << x << endl;
  x = x*y ;
  cout << "1.e318 = " << x << endl;
  x = x*y ;
  cout << "1.e328 = " << x << endl;
  return 0;
}
```

Remarquez bien les résultats obtenus.

$***$ Étalement d'un paquet d'onde $***$

La largeur du paquet d'ondes étudié au TD n°1 vaut, à l'instant t :

$$\Delta x(t) = \frac{a}{2}\sqrt{1 + \frac{4\hbar^2 t^2}{m^2 a^4}}$$

a) A $t = 0$ un électron libre a pour fonction d'onde un paquet d'ondes gaussien avec $a = 1\text{Å} = 10^{-10}$m.
Calculer Δx une seconde plus tard. On peut soit chercher les valeurs de $\hbar$ et m en unités SI sur internet, soit se reporter au dernier exercice du TD1 pour trouver les valeurs de $\hbar$, m et a en unités (fermi, MeV). Ne pas oublier de convertir la valeur de $t = 1$ s dans ce dernier cas.

b) Même question pour un proton (chercher la masse du proton sur internet).

c) Même question pour une sphère de diamètre 1 mm, de masse volumique 10^3 kg/m^3, avec $a = 1$ μm.

Commentaire sur cet exercice, à lire attentivement après avoir fait les questions

Il faut calculer la quantité sans dimension $u = \dfrac{2\hbar t}{ma^2}$, l'élever au carré et l'ajouter à 1. On peut utiliser indifféremment les unités microscopiques introduites au TD n°1 ou les unités SI. On obtient les ordres de grandeur suivants :

	u	u^2
Electron	2.3e+16	5.3e+32
Proton	1.2e+13	1.5e+26
Sphère	4.0e-16	1.6e-31

Il faut ensuite ajouter u^2 à 1. Dans le cas de l'électron et du proton l'intervalle entre les deux nombres réels représentables dans l'ordinateur qui encadrent la valeur de u^2 est beaucoup plus grand que 1. Ajouter 1 à u^2 n'aura donc strictement aucun effet, le 1 ne sera pas pris en compte, mais cela n'entraîne qu'une erreur négligeable. Dans le cas de la sphère, c'est l'inverse : l'intervalle entre 1 et le premier nombre représentable qui le suit est beaucoup plus grand que u^2, qui n'est donc pas pris en compte. Le résultat du calcul indique alors un élargissement nul, alors qu'en réalité il est extrêmement petit mais non nul. Pour le calculer il faut faire un développement limité

$$\sqrt{1+u^2} = 1 + u^2/2$$

justifié par le fait que, dans ce cas, l'erreur commise est inférieure ou égale à $u^4/8$, et qui indique que la variation relative de la largeur du paquet d'onde est de l'ordre de 10^{-31}.

Diagnostics en cas d'opérations illégales

Écrire et faire exécuter le programme suivant :

```
#include<math.h>
#include<iostream>
using namespace std;
int main() {
  double x = 3., y = 0., z = -1.;
  cout << endl << "Diviser par zéro : " << x/y << endl;
  cout << "Prendre la racine carrée d'un réel négatif : " << sqrt(z) << endl << endl;
  return 0;
}
```

Cas	Noter ici le résultat obtenu
Entier trop grand ou trop petit	
Réel trop petit en valeur absolue	
Réel trop grand en valeur absolue	
Diviser par 0	
Prendre la racine carrée d'un réel négatif	

3. Tests et boucles

Remarques

Un des buts des exercices suivants est de montrer que des formules équivalentes mathématiquement ne le sont pas du point de vue de la programmation.

Pour savoir comment changer la notation des réels, obtenir le nombre de chiffres voulu lors de l'affichage d'un réel avec la fonction `cout`, écrire et lire dans un fichier, consulter le chapitre du cours : **Entrées, sorties, fichiers, en C++** .

∗ ∗ ∗ Factorielle ∗ ∗ ∗

Calculer $n!$ avec :

1. une boucle `for`
2. une boucle `while`

Jusqu'à quelle valeur de n le calcul est-il juste si on travaille en entiers de type `int` ? Pour répondre à cette question, tester si la valeur maximale d'une variable du type `int` est dépassée pendant les calculs en vérifiant par exemple à chaque itération de la boucle que le nouveau résultat $i!$ divisé par le résultat de l'itération précédente $(i-1)!$ donne bien i comme il faut.

∗ ∗ ∗ Combinaison ∗ ∗ ∗

Calculer le nombre de combinaisons de k éléments pris parmi n :

$$C_n^k = \frac{n!}{k!(n-k)!}$$

en utilisant la formule de récurrence :

$$C_n^i = C_n^{i-1}\ \frac{n-i+1}{i}$$

Peut-on tirer profit de la propriété :

$$C_n^k = C_n^{n-k} \qquad ?$$

Ce programme permet-il de calculer le nombre de possibilités de choisir 10 élèves parmi 45 ? Et 44 parmi 45 ?
Pourquoi n'applique-t-on pas directement la formule en calculant $n!$, $k!$, $(n-k)!$ puis $n!/(k!(n-k)!)$?

∗ ∗ ∗ Pluie ∗ ∗ ∗

On considère un évènement qui a la probabilité p d'avoir lieu lors d'une épreuve. On note $q = 1-p$ la probabilité pour qu'il n'ait pas lieu. La probabilité $b(k)$ pour qu'il se produise k fois lors de n épreuves est donnée par la loi binomiale :

$$b(k) = C_n^k\ p^k (1-p)^{n-k}$$

(valeur moyenne de $< k >= np$ et écart-type de $k = \sqrt{npq}$)
Sachant qu'en moyenne il pleut 12 jours sur 30 au mois de novembre, quelle est la probabilité, sur une période de quatre jours, d'avoir respectivement 0, 1, 2, 3, 4 jours de pluie ? Utiliser une boucle pour afficher ces cinq résultats ; à noter qu'une seule boucle suffit, on n'a pas besoin d'une boucle séparée pour le calcul des C_n^k.
On admet que le temps qu'il fait un jour est indépendant de celui des jours précédents.

∗ ∗ ∗ Suite ∗ ∗ ∗

La limite de la suite suivante est $\sqrt{A}$:

$$u_0 = A \qquad\qquad u_{n+1} = \frac{1}{2}(u_n + \frac{A}{u_n})$$

a) Calculer ainsi une racine carrée avec une boucle `for` avec compteur, le nombre d'itérations étant fixé à l'avance à `nmax`. On fera afficher les nombres avec le maximum de chiffres significatifs (15) en écrivant, en début de programme,

l'instruction : `cout << setprecision(15);` (`setprecision` nécessite la directive `#include<iomanip>`)

b) Ajouter une condition pour que le calcul s'arrête comme au a) ou si :

$$|\frac{u_{n+1}-u_n}{u_n}| < \varepsilon$$

ε étant fixé à l'avance.
Pour cela on pourra ajouter un `break` dans le `for` ou utiliser une boucle `do ... while()`.
Pourrait-on utiliser une boucle `while()` au lieu de la boucle `do ... while()` ?

c) Avec les réels `double` combien faut-il d'itérations pour satisfaire la condition du b) lorsque $A = 2$ et ε=10^{-10} ?

d) Au vu des résultats affichés à l'écran, combien faut-il d'itérations, lorsque $A = 2$, pour que le résultat obtenu soit identique avec celui donné par la fonction de la bibliothèque mathématique `sqrt` (inclure la directive `#include<math.h>`) ?

∗ ∗ ∗ Sinus ∗ ∗ ∗

1) Calculer et afficher $\sin x$ de 0° à 90° de 10° en 10° à l'aide du développement :

$$\sin x = x - \frac{x^3}{6} + ... + (-1)^n \frac{x^{2n+1}}{(2n+1)!} + ...$$

en additionnant tous les termes de la série dont la valeur absolue est supérieure à 10^{-6}. Pour faire cela, il faut d'abord déduire une formule de récurrence pour calculer le nouveau terme à partir du terme précédent, vu qu'on ne veut surtout pas calculer x^{2n+1} et $(2n+1)!$ explicitement pour chaque terme à cause des problèmes élucidés dans les exercices précédents. Comparer au résultat donné par la fonction `sin` de la bibliothèque mathématique.
2) Pour $x = 20°$ par exemple, combien faut-il de termes de la série pour obtenir 6 chiffres significatifs identiques au résultat de la fonction `sin` ? Même question pour 12 chiffres significatifs.

∗ ∗ ∗ Test ∗ ∗ ∗

Le test de l'égalité de deux réels peut ne pas être fiable, car la plupart des réels ne peuvent pas être écrits de manière exacte dans l'ordinateur, le nombre de bits consacré à chacun d'eux étant fini (64 pour les doubles). Quel est le résultat du programme suivant ?

```
#include<iostream>
using namespace std;
int main() {
  int i; double x;
  x = -1.;
  for(i = 1; i <= 10; i++)
    x = x + 0.1;
  if(x != 0.)
    cout << "x = " << x << " le test trouve x différent de 0" << endl;
  else
    cout << "x = " << x << " le test trouve x égal à 0" << endl;
  return 0;
}
```

∗ ∗ ∗ Équation du second degré ∗ ∗ ∗

Calculer les racines réelles de l'équation du second degré à coefficients réels :

$$ax^2 + bx + c = 0$$

On supposera que a, b, c peuvent avoir des valeurs réelles quelconques, y compris $a = b = c = 0$, car s'ils sont issus d'un calcul précédent il faut prévoir tous les cas. Le programme doit afficher la valeur d'une variable entière qui vaut :

0 s'il n'y a pas de racine réelle
1 s'il y a une seule racine réelle, simple ou double
2 s'il y a deux racines réelles distinctes
3 dans le cas $a = b = c = 0$ (équation indéterminée, tout x est solution)

et les valeurs des racines quand il y en a.

∗ ∗ ∗ Fichiers ∗ ∗ ∗

Écrire un programme qui d'abord écrit les 11 couples suivants d'un nombre avec son carré

```
0 0
1 1
2 4
3 9
...
10 100
```

dans un fichier nommé *valeurs.dat* (avec donc un couple par ligne) et qui après ouvre ce même fichier pour lire et afficher à l'écran ces valeurs.

∗ ∗ ∗ Graphiques ∗ ∗ ∗

Écrire un programme en C qui fait appel au Python pour tracer une courbe à partir des valeurs dans le fichier *valeurs.dat* de l'exercice précédent. Les couples représentent l'abscisse et l'ordonnée des points. Voir la notice *Graphisme avec Python Matplotlib* du site de l'enseignement d'informatique L3-Mag1.

∗ ∗ ∗ Températures à Orly en juillet 2015 ∗ ∗ ∗

Copier le fichier *temperatures.dat* par la commande :

```
cp /public/mphyo/temperatures.dat .
```

(faites attention à l'espace suivi par un point à la fin). Ce fichier contient les températures relevées toutes les trois heures à Orly du 1er juillet 2015 à 00h au 31 à 21h. Il se présente sous la forme :

```
23.8
19.7
23
...
```

Les 8 premières lignes sont donc les températures à 0h, 3h, 6h, 9h, 12h, 15h, 18h, 21h le 1er juillet, les 8 suivantes les températures aux mêmes heures le 2 juillet, etc. Au total le fichier comporte donc $31 \times 8 = 248$ lignes.
Écrire un programme qui trace sur un même graphe :

- la courbe des minimas journaliers en fonction du jour du mois
- la courbe des maximas journaliers en fonction du jour du mois

Pour le tracé utiliser la notice *Graphisme avec Python Matplotlib* du site de l'enseignement d'informatique L3-Mag1.
Attention : il faut faire calculer tous ces maximas et minimas journaliers par le programme C, les faire écrire dans un (nouveau) fichier, puis faire appel au Python depuis le C seulement pour faire afficher les courbes (voir notice). Ceci est un cours du C et non pas du Python!

∗ ∗ ∗ Conjecture de Syracuse ∗ ∗ ∗ *(facultatif)*

Soit u_0 un entier positif et la suite :

$u_{n+1} = u_n/2$ si u_n est pair
$u_{n+1} = 3u_n + 1$ si u_n est impair

Calculer les termes de cette suite jusqu'à trouver 1.
(Il existe une conjecture selon laquelle on finit toujours par trouver 1. Pour plus de renseignements chercher « Conjecture de Syracuse » sur *google*).

4. Fonctions

Remarque

Dans tous les cas on fera afficher les données de départ et les résultats par le programme principal.

∗ ∗ ∗ Force gravitationnelle ∗ ∗ ∗

Le module de la force gravitationnelle exercée par une sphère homogène de rayon R et de masse M sur une masse ponctuelle m située à une distance d du centre de M, vaut :

$$G\frac{Mm}{d^2} \text{ pour } d \geq R \quad \text{et} \quad G\frac{Mmd}{R^3} \text{ pour } d \leq R$$

G étant la constante de la gravitation universelle.
Écrire une fonction d'arguments M, R, m et d qui retourne le module de la force.

∗ ∗ ∗ Combinaisons ∗ ∗ ∗

Transformer en fonction le programme, écrit dans un TD précédent, qui calcule le nombre de combinaisons de k éléments choisis parmi n, ces deux paramètres étant transmis en argument à la fonction.

∗ ∗ ∗ Fonction créneau ∗ ∗ ∗

Ecrire une fonction dont l'argument est un réel x et dont la valeur est celle de la fonction créneau :

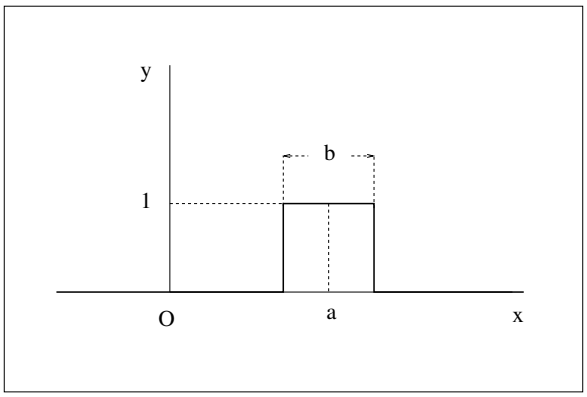

FIGURE 1 –

a et b étant deux paramètres. Traiter successivement les deux cas :

a et b sont définis en variables locales dans la fonction

a et b sont définis en variables globales, connus par le programme principal et la fonction.

∗ ∗ ∗ Fonction de répartition ∗ ∗ ∗

1) Écrire une fonction d'arguments x et n, retournant la valeur de :

$$\frac{1}{\sqrt{2\pi}} \int_0^x \exp(-\frac{t^2}{2})\, dt$$

calculée numériquement par la formule des rectangles :

$$\int_a^b f(x)\,dx \simeq h\sum_{i=1}^{n} f\left[a+(i-\frac{1}{2})h\right]$$

où n est le nombre de rectangles et $h=(b-a)/n$ le pas.

2) Ajouter une fonction qui effectue un ajustement automatique de la valeur de n de la façon suivante : on part d'une valeur quelconque de n et on la double tant que la valeur absolue de la variation relative de l'intégrale quand on passe de n à $2n$ est supérieure à ε.

3) Sachant que :

$$\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{+\infty}\exp(-\frac{t^2}{2})\,dt = 1$$

modifier la fonction précédente pour qu'elle retourne la valeur :

$$\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x}\exp(-\frac{t^2}{2})\,dt$$

qui est la fonction de répartition de la loi normale réduite.

$***$ Lentilles simples $***$ *(facultatif)*

On rappelle la relation de conjugaison d'une lentille mince :

$$\frac{1}{s}+\frac{1}{s'}=\frac{1}{f}$$

où s est la distance entre la source et la lentille, s' la distance entre l'image et la lentille, et f la distance focale. La distance s (ou s' ou f) est positive dans le cas d'une source (ou image ou point focal) réelle et négative dans le cas d'une source (ou image ou point focal) virtuelle.

1) Sur un axe Ox on note :

l l'abscisse du centre d'une lentille simple de focale f
x l'abscisse d'un objet
x' l'abscisse de l'image de l'objet donnée par la lentille

Écrire une fonction qui retourne la valeur x' et dont les arguments sont l, f et x.

2) On place maintenant sur l'axe Ox :

un objet en O
une lentille de focale F à l'abscisse u avec $0 \leq u \leq d$
une lentille de focale $f>0$ à l'abscisse $d>0$

On appelle y l'abscisse de l'image de O donnée par le système des deux lentilles quand on prend le point focal de la seconde lentille comme origine (il faut donc effectuer une translation de l'origine à la fin du calcul). Écrire les instructions qui calculent y en fonction de u, F, d et f.

3) On donne les valeurs numériques : f=20mm, d=500mm, F=50mm.
Tracer, avec *Python Matplotlib*[1], le graphe de $y(u)$, lorsque u varie entre 0 et d.

4) Au système précédent on ajoute un écran à l'abscisse $d+f$. Le système lentille de focale f plus écran schématise le cristallin et la rétine d'un œil. On suppose que l'image paraît nette pour l'œil lorsqu'elle se forme dans une zone comprise entre la rétine et le plan situé 2mm au-delà de la rétine (donc du côté opposé au cristallin).
Indiquer approximativement pour quelles valeurs de u cela se produit, en portant sur le graphe les deux droites horizontales $y=0$ et y=2mm.

5) Ajouter sur le graphe la valeur du grandissement du système en fonction de u.

1. Voir la notice *Graphisme avec Python Matplotlib* du site de l'enseignement d'informatique L3-Mag1.

5. Pointeurs

∗ ∗ ∗ Comparaison variable ordinaire, pointeur ∗ ∗ ∗

Prévoir et expliquer le résultat du programme suivant :

```
#include<iostream>
using namespace std;
int main() {
  int i, j;
  i = 1; j = i;
  cout << "i = " << i << " j = " << j << endl;
  i = 2;
  cout << "i = " << i << " j = " << j << endl;
  int *x, *y;
  x = &i;
  *x = 1; y = x;
  cout << "*x = " << *x << " *y = " << *y << endl;
  *x = 2;
  cout << "*x = " << *x << " *y = " << *y << endl;
  return 0;
}
```

Prévoir ce qui se passerait si on ne mettait pas l'instruction `x = &i;`.

∗ ∗ ∗ Équation du second degré, version fonction ∗ ∗ ∗

Transformer en fonction le programme, écrit dans le TD3, qui calcule les racines réelles de l'équation du second degré à coefficients réels

$$ax^2 + bx + c = 0$$

La fonction doit avoir le prototype suivant :

```
int rac_eqsd(double a, double b, double c, double* x1, double* x2)
```

Vu que la fonction doit renvoyer à la fois une valeur entière qui indique le nombre de racines réelles (voir l'énoncé en chapitre 3) et les valeurs des racines s'il y en a, on a donc besoin des pointeurs `x1` et `x2`. Votre fonction doit pouvoir traiter tous les cas possibles, y inclus les cas où un ou plusieurs des coefficients sont nuls.

Écrire également une fonction `main` qui fait appel à la fonction `rac_eqsd` et qui fait afficher les résultats à l'écran. Utiliser la fonction `cin` pour demander à l'utilisateur les valeurs des coefficients a, b, c.

∗ ∗ ∗ Passage par adresse d'une variable ordinaire ∗ ∗ ∗

Soit la suite :

$$u_{n+1} = au_n + b \qquad \text{avec } u_0 \text{ quelconque}$$

Écrire :

a) une fonction de type `void` ayant deux arguments de type `double` et un de type `double*` :

deux en entrée pour transmettre a et b
un qui contient u_n en entrée et u_{n+1} en sortie

b) un programme principal imprimant les N premiers termes de la suite.

∗ ∗ ∗ Trier deux nombres ∗ ∗ ∗

Écrire une fonction de deux arguments qui reçoit en entrée deux valeurs réelles quelconques et qui renvoie la valeur la plus petite des deux dans le premier argument et la valeur la plus grande dans le deuxième. Écrire également un programme principal qui fait appel à cette fonction.

∗ ∗ ∗ Dérivée ∗ ∗ ∗

Étant donné une fonction $f(x)$, une valeur approchée de sa dérivée en x est donnée par la formule à cinq points suivante :

$$f'(x) = \frac{1}{12\,\Delta x}\left[f(x-2\Delta x) - 8f(x-\Delta x) + 8f(x+\Delta x) - f(x+2\Delta x)\right] \qquad (1)$$

Δx étant une variation de x choisie par l'utilisateur.

1) Écrire une fonction à laquelle on fournit en argument :

une fonction $f(x)$
une valeur x_0
une valeur de la variation Δx

et dont la valeur est la valeur approchée de la dérivée en x_0 calculée par la formule (1).
(Tester la précision de la formule avec la fonction *arc tangente* dont la dérivée est :

$$\frac{1}{1+x^2}$$

ou toute autre fonction au choix.)

2) Étudier la précision du résultat en fonction de la valeur de Δx.

6. Tableaux dynamiques

∗ ∗ ∗ Matrice ∗ ∗ ∗

Soit la matrice suivante :

```
     0.1    10.31     8.7       2
2841.573      54    6.57  185.34
    5.03    0.73    1065    0.07
```

L'utiliser pour initialiser un tableau et la faire afficher à l'écran.
N'oubliez pas de libérer la mémoire du tableau à la fin du programme : pour chaque `malloc` dans votre programme il doit y avoir un `free` correspondant.

∗ ∗ ∗ Remplirtab ∗ ∗ ∗

Écrire un programme qui remplit un tableau d'entiers `n(10,9)` afin que le résultat final soit :

000000000
100000001
200090002
300090003
400090004
500090005
600090006
700090007
800090008
999999999

Pour vérifier le résultat, faire afficher ce tableau à l'écran.

∗ ∗ ∗ Norme d'un vecteur ∗ ∗ ∗

Les composantes d'un vecteur $\vec{v}$ dans un repère orthonormé à n dimensions sont définies dans un programme principal par un tableau `v`.
Ecrire une fonction de `v` dont la valeur est la norme de $\vec{v}$.
Dans le programme principal faire deux appels successifs à la fonction avec deux valeurs de n (et `v`) différentes.

∗ ∗ ∗ Vecteur unitaire ∗ ∗ ∗

Les composantes d'un vecteur $\vec{v}$ dans un repère orthonormé à n dimensions sont définies dans un programme principal par un tableau `v` à n éléments.

1) Écrire une fonction de type `int` qui :

si $\vec{v}$ n'est pas nul, vaut 1 et retourne en argument les composantes du vecteur unitaire $\vec{u}$ parallèle à $\vec{v}$ et de même sens que lui dans un tableau `u`
si $\vec{v}$ est nul vaut 0 et retourne le vecteur nul au lieu d'un vecteur unitaire dans le tableau `u`.

Comment faire pour interdire à priori toute modification des valeurs de `v` dans la fonction ?

2) Dans le programme principal, faire deux appels successifs de la fonction avec deux vecteurs de dimensions différentes.

∗ ∗ ∗ Trace d'une matrice ∗ ∗ ∗

Écrire une fonction dont la valeur est la somme des éléments diagonaux d'une matrice carrée de dimension quelconque fournie en argument.

∗ ∗ ∗ Produit de matrices ∗ ∗ ∗

Écrire une fonction à laquelle on passe en argument une matrice `a` à `m` lignes et `n` colonnes et une matrice `b` à `n` lignes et `p` colonnes et qui retourne en argument la matrice produit.

∗ ∗ ∗ Résolution de l'équation de Laplace : lentille électrostatique à fente ∗ ∗ ∗

Le problème étudié est à deux dimensions. Dans une enceinte conductrice rectangulaire au potentiel 0 sont placés des conducteurs de potentiels imposés.

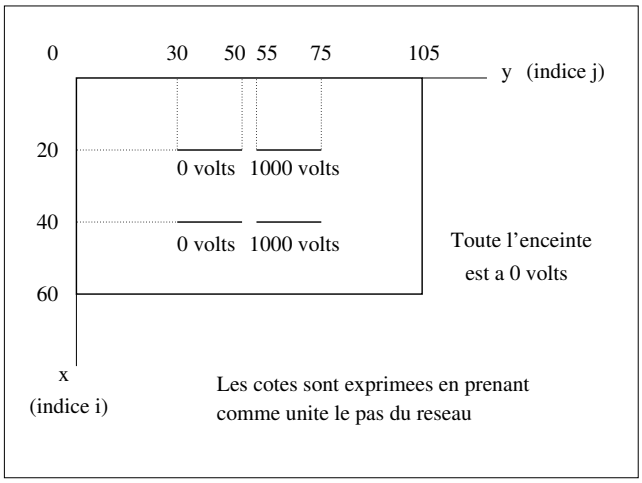

FIGURE 2 – Agrandir cette figure à l'écran pour bien distinguer les traits pleins des traits pointillés

Le but de l'exercice est de calculer le potentiel électrostatique $V(x, y)$ à l'intérieur de l'enceinte, hormis sur les conducteurs. Il s'agit de déterminer numériquement la solution unique de l'équation :

$$\begin{cases} \Delta V = 0 \text{ entre les conducteurs} \\ V \text{ a les valeurs imposées sur les conducteurs} \end{cases}$$

On effectue le calcul de V uniquement aux points d'un réseau régulier [2] appliqué sur l'enceinte, c'est à dire en des points tels que $x = i$ et $y = j$, i et j étant des entiers nuls ou positifs. La quantité $V(i, j)$ est notée $V_{i,j}$. Dans ces conditions la solution est obtenue de la façon suivante :

1. pour tous les points situés sur un conducteur, y compris l'enceinte, on attribue à V la valeur du potentiel de ce conducteur
2. pour tous les autres points de l'enceinte on attribue à V une valeur initiale nulle puis on calcule tous les $V_{i,j}$ par la formule :

$$V_{i,j} = \frac{V_{i-1,j} + V_{i,j-1} + V_{i+1,j} + V_{i,j+1}}{4} \quad \text{(moyenne arithmétique des quatre points adjacents)}$$

3. on réitére l'étape 2. jusqu'à obtenir une convergence satisfaisante.

1) Écrire un programme qui calcule le potentiel pour un nombre d'itérations fixé par l'utilisateur et imprime les valeurs finales dans un fichier sous la forme :

$$\begin{array}{ccccccc} V_{0,0} & V_{0,1} & \dots & V_{0,j} & \dots & V_{0,105} \\ V_{1,0} & V_{1,1} & \dots & V_{1,j} & \dots & V_{1,105} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ V_{i,0} & V_{i,1} & \dots & V_{i,j} & \dots & V_{i,105} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ V_{60,0} & V_{60,1} & \dots & V_{60,j} & \dots & V_{60,105} \end{array}$$

2) Compléter le programme précédent pour que le calcul s'arrête dès que l'une des deux conditions suivantes est réalisée :

2. dont on choisit le pas comme unité de longueur

le nombre d'itérations fixé par l'utilisateur est dépassé
$|V_{i,j}(\text{itération n+1}) - V_{i,j}(\text{itération n})| \leq \varepsilon$ pour *chacune* des valeurs de i et j hors conducteurs, ε étant choisi par l'utilisateur.

3) Tracer la surface $V(x, y)$ avec *Python Matplotlib*[3] sous forme de nappe puis sous forme de courbes de niveau.

∗ ∗ ∗ Échantillonnage d'une fonction ∗ ∗ ∗

On appelle échantillonnage de la fonction $f(x)$ n valeurs de cette fonction correspondant à n valeurs de x régulièrement espacées entre les bornes x_{min} et x_{max}.

1) Écrire une fonction `ech(f,xmin,xmax,n,t)` qui calcule le tableau d'échantillonnage `t`, `f` étant la fonction à échantillonner.

2) Donner ensuite un exemple d'utilisation de cette fonction en écrivant :

un programme principal complet qui l'appelle en lui fournissant les arguments nécessaires
une fonction `f` test

3) Ajouter dans le programme principal l'écriture du tableau d'échantillonnage dans un fichier.

∗ ∗ ∗ Interpolation linéaire ∗ ∗ ∗

On suppose qu'on ne connaît une fonction mathématique $f(x)$ qu'en n points d'abscisses x_i. Les valeurs des x_i et les valeurs correspondantes $y_i = f(x_i)$ de la fonction sont données dans un fichier organisé de la façon suivante :

sur la première ligne n
sur les n suivantes les couples x_i y_i

Les x_i ne sont pas nécessairement régulièrement espacées mais sont rangées par ordre croissant. L'abscisse minimum est donc x_0 et l'abscisse maximum x_{n-1}.
Exemple d'un tel fichier (que vous devez créer vous-même) :

```
8
0.2 2.5
0.35 0.198669
...
0.67 -8.472
```

Écrire un programme qui stocke les valeurs des x_i dans un tableau à un indice et les valeurs des y_i dans un autre tableau à un indice, puis qui fait appel à une fonction pour calculer la valeur interpolée linéairement de f en un point d'abscisse x quelconque (à demander à l'utilisateur avec la commande `cin`).
Écrire la fonction dont les arguments sont n, les x_i, les y_i, x et c et qui retourne la valeur interpolée de $f(x)$. Le pointeur c est utilisé pour renvoyer la valeur 1 si x est en dehors de l'intervalle $[x_0, x_{n-1}]$ (dans ce cas on ne peut pas calculer une valeur interpolée et le programme principal doit afficher un message d'erreur) et 0 sinon.

∗ ∗ ∗ Trinum ∗ ∗ ∗

On donne une liste de `n` réels dans un tableau. Réécrire cette liste dans le même tableau par ordre croissant. On pensera à ce que l'on fait quand on remet un jeu de cartes dans l'ordre.
La liste des réels sera fournie dans le programme par initialisation du tableau.

3. Voir la notice *Graphisme avec Python Matplotlib* du site de l'enseignement d'informatique L3-Mag1.

$***$ Statistique des tailles d'une population d'individus $***$

On a mesuré la taille de chacun des individus d'une population donnée et on a écrit ces tailles, exprimées en mètres, dans un fichier, à raison d'une taille par ligne. On dispose donc d'un fichier du type (à créer vous-même) :

```
1.625
1.576
1.733
...
```

On ne sait pas à priori combien de tailles, donc de lignes, contient le fichier.
On demande d'écrire un programme qui :

1) donne le nombre de tailles mesurées, la valeur minimale et la valeur maximale, la valeur moyenne $< t >$, et l'écart type $\sqrt{< t^2 > - < t >^2}$
2) constitue dans un tableau la distribution statistique des tailles par tranches de un centimètre.

Attention : il est inutile de stocker les tailles dans un tableau. Il n'est pas non plus nécessaire de lire le fichier deux fois.

$***$ Divergence et rotationnel d'un champ de vecteurs $***$ *(facultatif)*

On considère un champ de vecteurs :

$$\overrightarrow{V}(\overrightarrow{r})$$

dans l'espace à trois dimensions et on veut en calculer numériquement la divergence et le rotationnel en un point quelconque ($\overrightarrow{r}$ est le vecteur position).
Le calcul numérique des dérivées sera fait par la formule approchée :

$$\frac{f(x+h) - f(x-h)}{2h}$$

h étant le pas de dérivation numérique à choisir par l'utilisateur.
Ce champ de vecteurs est défini par une fonction du type :

```
void champ(double *x,double *v) {
  v[0] = une fonction quelconque de x[0],x[1],x[2]
  v[1] = une seconde fonction quelconque de x[0],x[1],x[2]
  v[2] = une troisième fonction quelconque de x[0],x[1],x[2]
}
```

où `v[0],v[1],v[2]` désignent les trois composantes du vecteur $\overrightarrow{V}$ et `x[0],x[1],x[2]` celles du vecteur $\overrightarrow{r}$.
Écrire une fonction qui a pour arguments d'entrée :

un champ de vecteurs, c'est à dire une fonction de même prototype que `champ`
la position à laquelle on veut calculer la divergence et le rotationnel, c'est à dire le tableau `x`
le pas de dérivation numérique `h`.

et pour arguments de sortie :

la divergence
le rotationnel.

Écrire également un exemple de programme principal appelant cette fonction.

7. Générateur aléatoire

∗ ∗ ∗ Lancer de pièces ∗ ∗ ∗

On lance n pièces identiques de rayon r dans le fond d'une boîte carrée de côté 1. Les pièces tombent à plat et peuvent se chevaucher. Si une des pièces heurte le bord de la boîte on ne compte pas le lancer.

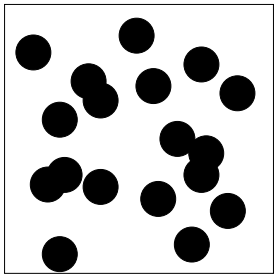

FIGURE 3 –

On veut calculer la proportion de la surface du fond de la boîte qui est recouverte en moyenne par les pièces, en utilisant une méthode de Monte Carlo. Pour cela :

- Simuler un lancer de pièces en tirant au hasard les coordonnées x_i et y_i des centres des n pièces.
- Calculer la proportion de la surface recouverte par un lancer donné. Pour cela, on tire au hasard un très grand nombre de points dans la boîte et on compte la proportion de ceux qui se trouvent sur une pièce.
- Répéter l'expérience un grand nombre de fois pour trouver la valeur moyenne de cette proportion.

Indiquer quelques conditions dans lesquelles on peut tester la justesse du programme.

Utiliser la fonction `drand48()` (nécessite la directive `#include<stdlib.h>`) dont la valeur varie aléatoirement, à chaque appel, dans l'intervalle $[0, 1]$ avec une distribution uniforme. Pour initialiser la suite des valeurs aléatoires à une valeur qui sera différente à chaque exécution du programme (à savoir le nombre de secondes depuis le 1er janvier 1970, c'est à dire l'heure UNIX), mettre la commande `srand48(time(NULL));` au début de la fonction `main` (ce qui nécessite en plus la directive `#include<time.h>`).

8. Équations différentielles

∗ ∗ ∗ Étude du pendule simple ∗ ∗ ∗

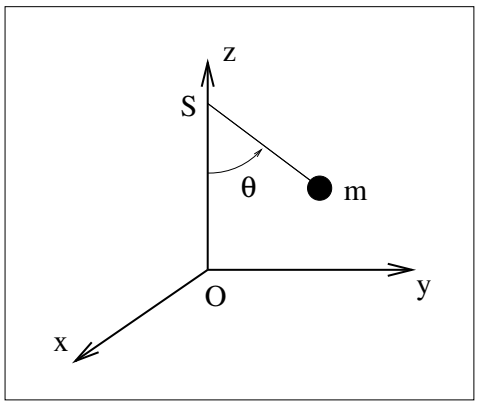

FIGURE 4 –

Un pendule simple est constitué d'une tige rigide sans masse, libre de tourner autour du point fixe S de l'axe Oz et d'une masse m située à l'autre extrémité de la tige.

La position et la vitesse initiale sont contenues dans le plan yOz. On pose $\theta = (\overrightarrow{SO}, \overrightarrow{Sm})$. Tracer la courbe de θ en fonction du temps, sans faire l'approximation des petits angles :

a) par la méthode d'Euler en programmant soi-même la formule d'Euler
b) par la méthode de Runge-Kutta d'ordre 4, en utilisant la fonction `rk4` [4] décrite dans le cours.

Tracer sur un même graphe les deux courbes obtenues aux a) et b) et celle résultant de l'approximation des petits angles (solution analytique).

∗ ∗ ∗ Inflation à deux champs ∗ ∗ ∗

L'inflation est une brève période d'expansion exponentielle dans l'univers primordial. Cette expansion énorme est engendrée par l'énergie potentielle d'un ou plusieurs champs scalaires. On considère ici un modèle d'inflation à deux champs avec un potentiel quadratique pour les deux. Les équations du mouvement des deux champs $\phi_1(t)$ et $\phi_2(t)$ sont données par

$$\begin{cases} \ddot{\phi}_1 + 3H\dot{\phi}_1 + m_1^2\phi_1 = 0 \\ \ddot{\phi}_2 + 3H\dot{\phi}_2 + m_2^2\phi_2 = 0 \end{cases}$$

où m_1 et m_2 sont deux constantes (les masses des deux champs, c'est-à-dire les masses des particules de spin-0 associées aux champs) et les points indiquent des dérivées par rapport au temps. Le paramètre de Hubble H est une fonction qui couple les deux équations :

$$H^2 = \frac{1}{6}(\dot{\phi}_1^2 + \dot{\phi}_2^2 + m_1^2\phi_1^2 + m_2^2\phi_2^2).$$

On prend $m_1 = 1 \cdot 10^{-5}$, $m_2 = 2.5 \cdot 10^{-5}$.

1) Écrire une fonction avec le prototype

```
void systeme(double* q, double t, double* qp, int n)
```

4. Pour pouvoir utiliser cette fonction, il suffit d'ajouter la directive `#include<bibli_fonctions.h>` et de compiler et exécuter avec *ccc*.

qui décrit le système d'équations différentielles ci-dessus dans une forme qui peut être utilisée par la méthode d'Euler ou la méthode de RK4. Les constantes m_1 et m_2 doivent être définies comme des constantes globales.

2) Écrire une fonction avec le prototype

```
void euler(void (*syst)(double*,double,double*,int), double* q, double t, double dt, int n)
```

qui fait une itération de la méthode d'Euler.

3) On prend $\phi_0 = 30$ comme condition initiale commune pour les deux champs, les deux vitesses initiales étant nulles. Écrire le reste du programme pour résoudre les équations du mouvement en utilisant la méthode d'Euler pour 1000 points entre $t = 0$ et $t = 3\phi_0/(2m_2)$ (bords inclus) et qui écrit les valeurs de t, ϕ_1 et ϕ_2 dans un fichier nommé `inflation.res`. Faire tracer les courbes $\phi_1(t)$ et $\phi_2(t)$.

4) Comment changer le programme pour qu'il utilise la méthode RK4 pour résoudre les équations du mouvement au lieu de la méthode d'Euler ?

L'éditeur *emacs*

Introduction

Un éditeur de texte permet d'écrire des caractères dans un fichier à partir du clavier et de la souris.
Toutes les commandes *emacs* peuvent être effectuées à partir du clavier, mais seules certaines d'entre elles peuvent l'être à l'aide de la souris, en utilisant les menus de la barre supérieure. Les commandes au clavier commencent par une des deux touches particulières *Ctrl* ou *Alt*, suivie d'un caractère. Par exemple :

> Ctrl-d signifie : maintenir la touche *Ctrl* enfoncée et appuyer sur la touche *d* (ce qui a pour effet d'effacer le caractère situé à l'emplacement du curseur).
> Alt-< signifie : maintenir la touche *Alt* enfoncée et appuyer sur la touche < (ce qui a pour effet de placer le curseur au début du fichier).

Il est à noter que la combinaison spéciale Ctrl-x est toujours suivie par un deuxième caractère (avec ou sans Ctrl), tandis que la combinaison spéciale Alt-x permet de taper le nom explicite d'une commande dans la barre de dialogue.
Dans la suite on notera ◁ pour la touche *Entrée*.

Ouvrir et fermer une session avec *emacs*

Ouvrir

Pour créer un nouveau fichier ou ouvrir un fichier déjà existant, de nom *fich.cpp* par exemple, écrire sur la ligne de commande *Linux*, après l'invite symbolisée par $:

$ emacs fich.cpp & ◁

Ceci a pour effet d'ouvrir une nouvelle fenêtre dans laquelle on va pouvoir écrire. Elle comporte :

> dans sa partie supérieure une barre de menus
> dans sa partie inférieure :
>
> > une barre affichant le nom du fichier, l'heure et le numéro de ligne de l'emplacement du curseur
> > une barre affichant le dialogue *emacs* -utilisateur.

Lorsque *emacs* pose une question dans la barre de dialogue il faut y répondre en terminant par *Entrée*.

Fermer

Sauver et fermer avec le même nom de fichier	Ctrl-x Ctrl-c
Sauver sans fermer avec le même nom de fichier	Ctrl-x Ctrl-s
Sauver sans fermer dans le fichier de nom *lili* (il y a demande du nom du fichier)	Ctrl-x Ctrl-w *lili* ◁

Ce qu'on écrit à l'écran est stocké dans une zone de la mémoire appelée *buffer* et n'est pas enregistré dans un fichier tant que l'on n'a pas sauvé d'une des trois façons précédentes. En cas d'arrêt anormal, une partie ou la totalité du contenu du buffer peut donc être perdue. A chaque fois qu'on sauve, la version précédente est conservée dans un fichier portant le même nom suivi d'un tilde : ici, par exemple, *fich.cpp*∼. Si on ferme sans sauver (dans ce cas une double confirmation est demandée) le contenu du buffer non sauvé est conservé dans un fichier de même nom encadré de dièses : ici, par exemple, #*fich.cpp*# [5]

Se déplacer dans le buffer

Le numéro de la ligne sur laquelle est situé le curseur est indiqué sur la barre inférieure de l'écran (L15 pour ligne 15).

Déplacer le curseur vers le haut, à gauche, vers le bas, à droite	↑ ← ↓ →
Avancer (reculer) d'un mot	Alt-f (b)
Aller en début de ligne	Ctrl-a
Aller en fin de ligne	Ctrl-e
Avancer d'un écran	Ctrl-v
Reculer d'un écran	Alt-v
Aller au début (à la fin) du buffer	Alt-< (>)
Placer le curseur à la $15^{\text{ème}}$ ligne	Alt-x goto-line ◁ 15 ◁
Placer le curseur au milieu de l'écran	Ctrl-l

5. Pour avoir accès à ce fichier dans une commande *Linux* il faut mettre des antislash avant les dièses : \#*fich.cpp*\#

Supprimer des caractères

Supprimer le caractère à l'emplacement du curseur	Ctrl-d
Supprimer le caractère précédent	Backspace
Supprimer depuis le curseur inclus jusqu'à la fin de la ligne	Ctrl-k
Supprimer le mot précédent	Alt-Backspace
Supprimer le mot suivant	Alt-d

Définir une région, la supprimer ou la déplacer

Définir une région	1) placer le curseur au début de la région et Ctrl-barre d'espace 2) placer le curseur immédiatement après la fin de la région
Mettre la région en mémoire et la supprimer (couper)	Ctrl-w
Mettre la région en mémoire sans la supprimer (copier)	Alt-w
Recopier la région mise en mémoire à l'emplacement du curseur (coller)	Ctrl-y

Chercher

Chercher la chaîne zzz vers l'avant (l'arrière) à partir du curseur	Ctrl-s (r) zzz (puis Ctrl-s (r) pour chercher l'endroit suivant)
Chercher et remplacer avec confirmation la chaîne zzz par la chaîne yyy	Alt-% zzz ◁ yyy ◁ et répondre y pour remplacer, n sinon et q pour quitter
Chercher et remplacer sans confirmation la chaîne zzz par la chaîne yyy	Alt-x replace-string ◁ zzz ◁ yyy ◁

Echanges avec d'autres fichiers

Insérer le contenu du fichier *lili* à l'endroit du curseur	Ctrl-x i *lili* ◁
Copier une région dans le fichier *lili*	Alt-x write-region ◁ *lili* ◁
Détruire le fichier *lili*	Alt-x delete-file ◁ *lili* ◁
Renommer le fichier *lili* en *lulu*	Alt-x rename file ◁ *lili* ◁ *lulu* ◁

Définir un rectangle, le supprimer ou le déplacer

Définir un rectangle	comme pour une région
Mettre le rectangle en mémoire et le supprimer	Ctrl-x r k
Mettre le rectangle en mémoire sans le supprimer	Ctrl-x r Alt-w
Recopier le rectangle mis en mémoire à l'emplacement du curseur	Ctrl-x r y
Supprimer le rectangle sans le mettre en mémoire	Ctrl-x r d

Indenter

Indenter une région d'un fichier C	Alt-x indent-region ◁
Idem	Alt-Ctrl-\

Ajouter une chaîne en début ou en fin de ligne

Ajouter la chaîne zzz en début (fin) de ligne	Alt-x query-replace-regexp ◁ ^ ($) ◁ zzz ◁

Recherche de commande par complétion

Cette méthode est très utile si on ne connaît pas le nom exact de la commande ou s'il est long à écrire.

Avoir la liste des commandes dont le nom commence par la chaîne zzz (longueur qqc)	Alt-x zzz Tab (touche Tabulation)
Avoir la liste de toutes les commandes	Alt-x Tab

Dans la liste affichée il suffit ensuite de cliquer sur la commande désirée pour la faire s'exécuter.
Par exemple si on veut ajouter la chaîne zzz en fin de ligne, on tape :

Alt-x qu Tab
emacs affiche alors la liste de toutes les commandes commençant par qu et, dans cette liste, se trouve la commande query-replace-regexp sur laquelle il suffit de cliquer pour que la commande s'exécute.

En cas d'erreur

Interrompre une commande en cours	Ctrl-g
Annuler les derniers changements un par un	Ctrl-x u ou Ctrl-_
Annuler tous les changements depuis la dernière sauvegarde	Alt-x revert-buffer ◁
Revenir à la version initiale, avant ouverture	Ctrl-x Ctrl-f fich.cpp∼ ◁ Ctrl-x Ctrl-w fich.cpp ◁
Accéder à l'aide en ligne	Ctrl-h
Débloquer l'écran	Ctrl-q

Bibliographie : *Emacs Debra Cameron O'REILLY* collection *précis & concis*

Commandes *Linux*

Dans ces pages on donne une liste de commandes courantes avec le minimum d'explications dans le but d'une utilisation simple et immédiate.
Linux distingue entre les minuscules et les majuscules.
Les fichiers sont rangés dans des répertoires répartis en arborescence.
Les commandes doivent être écrites dans la fenêtre Terminal, après l'invite, et validées par un *Entrée*.
Les commandes *cp*, *rm*, *mv* des tableaux suivants sont accompagnées d'une demande de confirmation si on leur ajoute l'option -*i*. Exemple :

rm fich destruction sans confirmation
rm -i fich destruction avec confirmation.

Les commandes suivies du signe (+) ont été ajoutées, elles sont propres à l'environnement du L3/Magistère.

Ctrl-b signifie : maintenir la touche *Ctrl* enfoncée et appuyer sur la touche *b* (ne signifie pas qu'il faut taper de - entre *Ctrl* et *b*).

Commandes les plus courantes

Faire la liste des fichiers et des répertoires du répertoire courant, sauf ceux qui commencent par un point	*ls*
Faire la liste de tous les fichiers et répertoires du répertoire courant	*ls -a*
Idem mais les noms de répertoires sont suivis d'un / et ceux des fichiers exécutables d'une *	*ls -aF*
Idem avec plus d'informations et par ordre chronologique inverse	*ls -altr*
Passer du répertoire courant au répertoire fils *rep*	*cd rep*
Ouvrir le site du cours dans un navigateur	*site* (+)
Editer le fichier *fich* (voir la notice de l'éditeur *emacs*)	*emacs fich &*
Faire défiler le contenu du fichier *fich* d'un seul coup	*cat fich*
Faire défiler le contenu du fichier *fich* page par page Passer à la page suivante Passer à la ligne suivante Passer à la page précédente Quitter	*less fich* *Barre d'espace* *Entrée* *b* *q*
Copier le contenu de *fich1* dans *fich2* (*fich2* est écrasé s'il existait déjà, *fich1* est conservé)	*cp fich1 fich2*
Changer le nom du fichier *fich1* en *fich2* (*fich2* est écrasé s'il existait déjà)	*mv fich1 fich2*
Déplacer le fichier *fich* dans le répertoire fils *rep* (*fich* est écrasé s'il existait déjà dans *rep*)	*mv fich rep*
Détruire le fichier *fich*	*rm fich*
Créer le répertoire *rep*	*mkdir rep*
Détruire le répertoire *rep* (Il faut d'abord l'avoir vidé de tout fichier)	*rmdir rep*
Compiler *prog.cpp*	*g++ -lm prog.cpp*
Exécuter	*./a.out*
Compiler et exécuter *prog.cpp*	*ccc prog.cpp* (+)
Interrompre et tuer un processus en cours	*Ctrl-c*
Passer un processus en cours en arrière-plan	*Ctrl-z bg*

Complétion et rappel de commandes

On peut ne taper que les premières lettres d'une commande ou d'un nom de fichier du répertoire courant, suivies de *Tab*. S'il n'y a pas d'ambiguïté le système complète tout seul, sinon il propose une liste des possibilités.
On peut également faire un copier-coller sur les noms de fichiers et de répertoires apparaissant dans la fenêtre du terminal dans lequel on écrit les commandes *Linux*. Il faut double-cliquer avec le bouton de gauche de la souris sur la chaîne de caractères que l'on veut saisir, puis cliquer sur le bouton du milieu.
On peut rappeler les commandes écrites antérieurement à l'aide des flèches verticales du clavier.
Il faut utiliser ces trois possibilités systématiquement, ce qui fait gagner beaucoup de temps.

Commandes un peu moins courantes

Ajouter le contenu du fichier *fich2* au fichier *fich1*	*cat fich2 >> fich1*
Rediriger dans le fichier *fich* le résultat de la commande *cmd*	*cmd > fich*
Caractère générique *	
Il remplace n'importe quelle chaîne de caractères dans un nom de fichier ou de répertoire.	
Désigner tous les fichiers dont le nom commence par *si* et se termine par *do*	*si∗do*
Ceux qui commencent par *fa*	*fa∗*
Ceux qui se terminent par *sol*	*∗sol*
etc...	*∗re∗*
	mi∗fa∗sol
Tous les fichiers	*
Très commode mais à utiliser avec précaution pour des opérations irréversibles :	
tester d'abord le résultat obtenu avec la commande *echo* (sans risque).	
Très utile aussi pour désigner un fichier unique en abrégé.	
Faire le ménage	
c'est à dire détruire (avec confirmation) tous les fichiers commençant par #,	
finissant par ∼, les fichiers *core* et *a.out*	*pu* (+)
Créer le fichier de commandes *fichcom*	*emacs fichcom &*
Créer le fichier de commandes *fichcom* dans le répertoire *bin* à partir	
du répertoire d'accueil et y placer la liste des commandes que l'on veut faire exécuter	*emacs bin/fichcom &*
Rendre *fichcom* exécutable	*chmod 755 fichcom*
Faire exécuter la liste de commandes	*./fichcom*
Créer l'abréviation *com* pour la commande *commande*	*alias com='commande'*
Pour le rendre permanent il faut placer cet *alias* dans le fichier *.bashrc*	
Faire apparaître les différences entre les fichiers *fich1* et *fich2*	*diff fich1 fich2*
Demander une description détaillée de la commande *com*	*man com*
Faire la liste de toute l'arborescence à partir du répertoire initial	*du -a*
Faire afficher toutes les lignes du fichier *fich* qui contiennent	
la chaîne *chaîne*	*grep chaîne fich*
Faire la liste de tous les utilisateurs connectés	*who -u*
Rechercher si l'utilisateur *util* est connecté	*who -u | grep util*
Rechercher où est placée la commande *com*	*which com*
Rechercher tous les noms de fichiers dont le nom contient la chaîne *toto*	*locate toto*

Comment corriger une ligne de commandes ?

Les commandes sont les mêmes que celles de *emacs* . Ce qui est utile ici :

Aller au début de la ligne de commande	*Ctrl-a*
Aller à la fin de la ligne de commande	*Ctrl-e*
Reculer d'un caractère	*Ctrl-b* ou ←
Avancer d'un caractère	*Ctrl-f* ou →
Détruire le caractère sous le curseur	*Ctrl-d* ou *Del/Suppr*
Détruire le caractère précédent	*Ctrl-h* ou *Backspace*
Effacer la fin de la ligne	*Ctrl-k*
Effacer toute la ligne	*Ctrl-u*
Echanger le caractère sous le curseur et celui qui le précède	*Ctrl-t*

Comment rappeler une ligne de commandes frappée précédemment ?

Faire défiler les lignes précédentes en remontant le temps	↑
Faire défiler les lignes précédentes en suivant le temps	↓

Conventions pour les noms de fichier

Il est conseillé d'adopter les conventions suivantes pour les noms de fichier :

C ou C++	Résultats	Données
fich.cpp	fich.res	fich.dat