UNIVERSITE FRANÇAISE DU PACIFIQUE
Centre Universitaire de Nouvelle Calédonie

Année 1999

INFORMATIQUE 2
Notes de Travaux Dirigés

D.E.U.G. MIAS et SM 2ème niveau

Claudine Chaouiya Chantegrel

Sommaire :

1. GENERALITES

Le module est composé de 2 unités d'enseignement : Informatique 2 (INFO2) et Anglais 2. L'objectif de l'unité INFO2 est d'une part une meilleure maîtrise de la programmation, d'autre part l'apprentissage de méthodes numériques classiques. Le langage utilisé est le Turbo Pascal.

Vous réaliserez des mini-projets, en binôme tournant, certains projets seront notés, la moyenne obtenue (C_1) comptant pour 1/3 dans la note de contrôle continu. En cours d'année vous aurez un partiel (noté C_2), et en fin d'année, vous présenterez un projet individuel (noté C_3). La note C de contrôle continu est égale à : $C = (C_1 + C_2 + C_3)/3$.

La note de l'unité d'enseignement INFO2 est égale à : 1/3 C + 2/3 E, où E est la note obtenue à l'examen final.

2. POUR SE REMETTRE EN ROUTE !

NB : écrivez des programmes lisibles et commentés. Un programme qui fournit un résultat correct mais qui est mal présenté et/ou pas efficace perd la moitié de sa valeur. Ecrivez un algorithme sur papier avant de vous lancer sur la machine.

1) Comment lancer Turbo Pascal ? Quelles sont les commandes essentielles de l'éditeur ? Comment compiler (à quoi ça sert ?), exécuter ?…
2) Quelles sont les trois parties d'un programme Pascal ? Donnez la déclaration d'une constante N=20. Qu'est ce qui justifie la déclaration d'une constante ? Donnez la déclaration d'un type Tab (tableau de 30 caractères) et Point (composé d'une Abs et d'une Ord réelles). Qu'est ce qui justifie la déclaration d'un type ?
3) Pourquoi écrire des procédures et des fonctions ?
4) Donnez l'entête de la procédure stat qui, étant donné un tableau T d'entiers, calcule la moyenne arithmétique Moy des éléments de T, son maximum Max et son minimum Min.
5) Donnez le résultat de l'exécution du programme suivant :

```
Program bidule ;
Var a,b : integer ;

Procedure p1(x,y :integer) ;
Var z :integer ;
Begin
        z :=x ;
        x :=y ;
        y :=z ;
        writeln(x,y) ;
End ;

Procedure p2(var x,y :integer) ;
Var z :integer ;
Begin
        z :=x ;
        x :=y ;
        y :=z ;
        writeln(x,y) ;
End ;

Procedure p3 ;
Var z :integer ;
Begin
        z :=a ;
        a :=b ;
        b :=z ;
End ;

Begin
        a :=1 ;
        b :=2 ;
        p1(a,b) ;
        writeln(a,b) ;
        p2(a,b) ;
        writeln(a,b) ;
        p3 ;
        writeln(a,b) ;
End.
```

5) Ecrivez les instructions permettant de calculer la somme des N premiers entiers (N donné).
6) Ecrivez les instructions permettant de déterminer l'élément maximal d'un tableau d'entiers T de dimension N (T et N donnés).
7) Ecrivez les instructions permettant de déterminer si un entier N se trouve dans un tableau d'entiers triés par ordre croissant T (T et N donnés).
8) Ecrivez un programme qui permet de lire au clavier une suite de nombres non nuls (l'utilisateur entrera 0 quand il aura fini) et d'afficher la moyenne arithmétique de ces entiers.

3. REPRESENTATION DES NOMBRES

Un ordinateur ne peut contenir qu'un nombre **fini** d'informations. En mathématiques, on utilise les notions d'infini, de continu qui ne peuvent être représentées sur une machine, d'où des erreurs dont il faut bien être conscient. C'est cette constatation qui motive ce chapitre.

3.1 RAPPELS

Numération binaire : tout calculateur électronique utilise une représentation binaire de l'information. L'unité d'information est le **bit**. L'écriture en base 2 de tout entier X s'obtient à partir de son développement en puissances de 2 :

$$X = \sum_{i=0}^{p} a_i 2^i , \quad a_i = 0 \text{ ou } 1, \; i = 0,...,p.$$

X s'écrit $a_p a_{p-1} ... a_1 a_0$ en base 2

Avec n digits binaires (0,1) on peut donc représenter les entiers de 0 à $(2^n - 1)$.
Avec un octet (8 bits) on peut représenter tous les nombres compris entre 0 et 255.

Représentation des entiers : Les suites de bits organisés en octets peuvent représenter des entiers de différentes façons.
Si on dispose de deux octets, soit 16 bits, on a $2^{16} = 65\,536$ possibilités de représentations.

Les entiers sans signe : avec 16 bits, on représente les entiers de 0 à 65 535.

Les entiers avec signe : on utilise le premier bit comme bit de signe (0 si le nombre est positif, 1 sinon) et le reste des bits code la valeur absolue du nombre. On peut ainsi représenter les entiers de -32767 et +32 767 ($-(2^{n-1}-1)$ et $2^{n-1}-1$).
Remarques : 1) avec cette notation, 0 a deux représentations,
2) pour additionner 2 entiers a et b dans cette notation, si a et b sont de signes différents, il faut déterminer le plus grand en valeur absolue et lui soustraire la valeur absolue de l'autre. Le signe est celui du plus grand en valeur absolue.

Complément à 2 : dans cette notation, on considère x l'entier sans signe codé par la suite de bits. Si le premier bit est 0, la suite de bits code le x, sinon elle code $(2^n + x)$ (cas de n bits). Avec 16 bits, on peut ainsi représenter les entiers de -32768 et +32 767 ($-(2^{n-1}-1)$ et $2^{n-1}-1$).
Remarques : 1) avec cette notation, 0 a une seule représentation,
2) pour additionner deux nombres, on n'a plus à se préoccuper de leur signe. Pour la soustraction, il y a un moyen très simple d'obtenir l'opposé d'un entier, on procède ensuite comme pour l'addition.
Exemple : s/ 6 bits, la suite 011111 représente l'entier $(31)_{10}$
la suite 101100 représente $(44)_{10}-(64)_{10} = -(20)_{10}$
la somme $(12)_{10} + (-19)_{10}$ est 001100 + 101101 car $(-19)_{10}+(64)_{10}=(45)_{10}$
$= (101101)_2$
la somme $(-12)_{10} + (19)_{10}$ est 110100 + 010011 qui vaut $(\mathbf{1}000111)_2$, il faut ignorer la retenue (bit le plus à gauche), on obtient $(000111)_2 = (7)_{10}$.
la somme $(14)_{10} + (19)_{10}$ est 001110 + 010011 qui vaut $(100001)_2 = (-31)_{10}$ ce qui est faux bien sûr. Comment expliquer ceci ?

Les opérations sur les entiers s'effectuent exactement, dans la mesure où le résultat est un entier représentable en machine.

Représentation des réels : supposons que nous ayons 10 caractères dont la virgule et le signe pour représenter un nombre réel. Dans une représentation **à virgule fixe** nous ne pourrons écrire que les nombres du type ±123,45678. L'éventail des nombres représentables est très limité : de ± , 00000001 à ± 99999999, (10^8-1).

Dans la représentation **à virgule flottante** un nombre s'écrit : $X = \pm a.\beta^n$ où a est la mantisse,β la base et n l'exposant. Ainsi, si nous disposons toujours de dix caractères et que nous les répartissions de la manière suivante : 1 pour le signe, 3 pour l'exposant, 6 pour la mantisse, on obtient ± 123456 β ±78.

La représentation **standard** est de la même forme ($X = \pm a.\beta^n$) avec β égal à 2, a et n représentés par des binaires. Le format d'un nombre réel en simple précision occupe 4 octets :

- l'exposant est stocké sur un octet (de -128 à +128)
- le signe sur un bit
- la mantisse occupe les 23 bits restants, elle s'écrit $d_1d_2...d_{23}$ (d_1=1), elle correspond au nombre

$$\frac{d_1}{\beta} + \frac{d_2}{\beta^2} + ... \frac{d_n}{\beta^n} + ... \frac{d_{23}}{\beta^{23}}$$

Le plus petit nombre en valeur absolue représentable (ou *zéro machine*) est le nombre d'exposant minimal -128 et de mantisse 100...0 (d_1=1, d_i=0, i=2...23), il vaut :

$$\frac{1}{2} 2^{-128} = 2^{-127} \approx 1{,}47\ 10^{-39}$$

Le plus grand nombre en valeur absolue représentable (*infini machine*) est le nombre d'exposant +127 de mantisse 111...1, il vaut :

$$(1-2^{-23})2^{127} \approx 1{,}7\ \ 10^{38}$$

L'écart entre 2 nombres successifs d'exposant n vaut $2^{-23}\ 2^n$. Cet écart croît de façon exponentielle avec n, ce qui explique pourquoi la précision des calculs en virgule flottante est d'autant moins bonne que les nombres sont grands. La meilleure précision possible est de 2^{-23} ≈ 1,19 10^{-7}. Pour des nombres d'ordre 1000 (d'exposant 10 car 2^{10}=1024) la meilleure précision est $2^{-23}\ 2^{10} \approx 1{,}22\ 10^{-4}$.

Un grand nombre d'erreurs d'arrondis peut s'expliquer ainsi, quand on cherche à obtenir un nombre petit par différence de grands nombres.

3.2 TRAVAIL A REALISER

a) Ecrivez un programme qui calcule $\sum_{i=1}^{10\,000} x$ avec $x = 10^{-1}, 10^{-3}, 10^{-4}$. Que constatez-vous ?

b) Ecrivez un programme qui lit au clavier un entier n et affiche sa factorielle, que se passe-t-il pour n=8 ? Expliquez. Turbo Pascal met à votre disposition 5 types d'entiers prédéfinis :

`shortint`	... 127	8 bits signé
`integer`	... 32 767	16 bits signé
`longint`	... 2147483647	32 bits signé
`byte`	... 255	8 bits non signé
`word`	... 65535	16 bits non signé

Essayez maintenant de travailler en entiers longs (type `longint`) que constatez-vous ? Pourquoi ? Calculez la factorielle de 34. Proposez deux versions de ce programme : itérative et récursive.

Pour mémoire les types réels sont les suivants :

`real`	$2.9\ 10^{-39}$ ... $1.7\ 10^{38}$	11-12 chiffres significatifs	6 octets
`single`	$1.5\ 10^{-45}$ ... $3.4\ 10^{38}$	7-8 chiffres significatifs	4 octets
`double`	$5.0\ 10^{-324}$ ... $1.7\ 10^{308}$	15-16 chiffres significatifs	8 octets
`extended`	$3.4\ 10^{-4932}$ ... $1.1\ 10^{4932}$	19-20 chiffres significatifs	10 octets

c) Ecrivez un programme qui 1) lit un entier en base 2 et affiche sa conversion en base 10 ou 2) lit un entier en base 10 et affiche sa conversion en base 2.

d) Ecrivez un programme qui lit une suite de n bits représentant un entier signé et donne sa représentation en complément à 2 (n est lu au clavier). Vous écrirez d'abord une première solution qui consiste à passer par l'intermédiaire de la base 10. Vous écrirez ensuite une procédure de conversion directe.

e) (Final 2ème session 98) Numération en base 5
On rappelle que le codage d'un entier N ($N \geq 0$) en base 5 s'obtient à partir de son développement en puissances de 5 :

$$X = \sum_{i=0}^{p} a_i 5^i, \quad a_i = 0,1,2,3 \text{ ou } 4,\ i = 0,\ldots,p.$$

Ecrivez la procédure conversion(n,c) qui, étant donné un entier n, positif, détermine c son code en base 5 (vous proposerez un type Pascal pour la représentation d'un code).
Ecrivez une fonction booléenne divisible(c,j), qui détermine si c, entier positif codé en base 5 est divisible par la $j^{ième}$ puissance de 5.
Ecrivez une procédure plus(a,b,c) qui, étant donnés deux entiers positifs codés en base 5, détermine c=a+b en base 5.

4. ARITHMETIQUE ...

4.1 L'ALGORITHME D'EUCLIDE

Soient *a* et *b* 2 entiers naturels, D l'ensemble de leurs diviseurs communs et M l'ensemble de leurs multiples communs. On appelle PGCD (Plus Grand Commun Diviseur) de *a* et de *b* le plus grand élément *d* de D, PPCM (Plus Petit Commun Multiple) le plus petit élément *m* de M. On peut démontrer que $ab = md$. Si d=1 on dit que *a* et *b* sont premiers entre eux.

L'algorithme le plus performant est l'algorithme d'Euclide (III^{ème} siècle) :

le PGCD de *a* et *b* est le dernier reste non nul obtenu par divisions euclidiennes successives de *a* par *b*

$a = b\ q_1 + r_1$
$b = r\ q_2 + r_2$
.....
$r_{n-2} = r_{n-1}\ q_n + r_n$
$r_{n-1} = \mathbf{r_n}\ q_{n+1}$

Le PPCM s'obtient ensuite grâce à la relation donnée plus haut.
Il s'agit d'écrire un programme qui détermine le PGCD de deux entiers positifs *x* et *y*. Sans perte de généralité, on supposera $y \geq x$.

a) Ecrivez la fonction PGCD utilisant l'algorithme d'Euclide.
b) Utilisez la fonction PGCD pour écrire une fonction PPCM.
c) Vérifiez rapidement que: si $x = y$ leur PGCD est tout trouvé,
si $x \neq y$ tout diviseur commun de *x* et *y* est aussi diviseur commun de *x* et *y-x*,

utilisez ces propriétés pour écrire une fonction PGCD <u>récursive.</u>

4.2 TRIANGLE DE PASCAL

On rappelle que les coefficients binomiaux permettent de calculer le nombre de combinaisons de p éléments dans un ensemble de n éléments et :

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

Les coefficients binomiaux vérifient la propriété suivante :

$$(A) \quad C_n^p = C_{n-1}^{p-1} + C_{n-1}^p$$

a) Prouvez la propriété (A) et expliquez comment elle peut être utile pour spécifier un algorithme de calcul des coefficients binomiaux. Cet algorithme est en fait la construction du triangle de Pascal dont la i^{ème} ligne est constituée des coefficients C_i^p .pour p = 0,...,i.

Notre objectif maintenant est d'afficher les premières lignes du triangle de Pascal.
Pour chacune des questions suivantes, il s'agit d'écrire un programme qui lit l'entier n et affiche les n premières lignes du triangle de Pascal (vous soignerez l'affichage) :

b) Ecrivez une procédure d'affichage des *n* premières lignes du triangle de Pascal, utilisant un tableau à deux dimensions (seuls les éléments situés au dessous de la diagonale seront utiles). On remplira ce tableau ligne par ligne. Quel est l'espace mémoire occupé, en fonction de n?

c) Pour le calcul de chaque ligne, on peut noter que l'on n'a besoin que de la ligne qui précède. Avec cette constatation, améliorez l'algorithme en utilisant 2 tableaux à une dimension. Quel est l'espace mémoire occupé, en fonction de n?

d) Pour le calcul d'un élément, on n'a besoin que de 2 éléments de la ligne précédente. Il est donc inutile, dans le calcul d'une ligne, de garder la totalité de la ligne précédente. Utilisez cette remarque pour réécrire l'algorithme, avec cette fois un seul tableau à une dimension. Quel est l'espace mémoire occupé, en fonction de n?

4.3 NOMBRES PREMIERS

a) Ecrivez un programme qui recherche et affiche les nombres premiers inférieurs à M (entier lu au clavier) en utilisant une méthode de divisions successives. Utilisez les remarques suivantes pour optimiser votre programme :

- le seul nombre premier pair est 2,
- si n entier naturel non premier, alors un de ses diviseurs est inférieur ou égal à $\sqrt{n}$.

b) Ecrivez un programme qui recherche et affiche les nombres premiers inférieurs à M (entier lu au clavier) en utilisant la méthode du crible d'Eratosthène qui consiste à *barrer* dans un tableau contenant tous les entiers naturels de 2 à *n* les multiples de 2, de 3, etc... Ecrivez une procédure qui met en oeuvre la méthode du crible d'Eratosthène.
Vous utiliserez un tableau de booléens dont la case n°i contiendra *true* ou *false* selon que i est premier ou non.

5. SERIES ET SUITE DE FIBONACCI

Soit une suite de nombres réels ou complexes notée $\{u_n\}_{n\geq 0}$ ou $\{u_n\}_{n\in \mathbb{N}}$ ou encore simplement $\{u_n\}$. On s'intéresse à la suite des sommes partielles $\{S_n\}_{n\geq 0}$ définie par :

$$S_n = \sum_{i=0}^{n} u_i.$$

On dit que la série de terme général u_n notée $(u_n)_{n\in \mathbb{N}}$ (ou $(u_n)_{n\geq 0}$ ou (u_n)) converge si et seulement si la suite $\{S_n\}$ converge. Etudier la nature d'une série, c'est dire si elle converge ou non. Dans le cas où la série $(u_n)_{n\geq 0}$ converge, la limite de la suite des sommes partielles s'appelle la somme de la série :

$$\lim_{n\to+\infty} S_n = \sum_{n=0}^{+\infty} u_n .$$

La convergence de suites est un problème important. Les programmes que nous écrirons ne constituent en aucun cas une preuve de convergence ou divergence. Ils permettent d'étudier le comportement d'une suite et en déduire la tendance à converger (ou diverger), d'observer les

vitesses de convergence (ou divergence) et d'obtenir une valeur approchée de la limite éventuelle.

5.1 SERIES

On notera par une majuscule les sommes partielles des séries, par exemple $R_n = \sum_{i=1}^{n} r_n$.

Vous écrirez un seul programme avec un menu grâce auquel on choisira d'étudier la série (r_n), (s_n), (t_n), (u_n), (v_n) ou (w_n). Pour chacune de ces séries vous écrirez une fonction avec un paramètre n indiquant l'ordre du calcul de la somme partielle, et, éventuellement, un paramètre x, si le terme général est défini en fonction d'une variable.
Pour chacune des séries, vous prouverez sur le papier la convergence (ou la divergence), et vous vérifierez vos résultats avec votre programme.

terme général r_n $1/n$
:
terme général s_n $1/n^2$
:
terme général t_n nx^{n-1}, où $x \in \mathbb{R}^+$
:

terme général v_n : $(-1)^{n+1}.\frac{x^n}{n}$

terme général w_n : $(-1)^n.\frac{x^{2n+1}}{2n+1}$

Il faudra que vos calculs soient performants.

5.2 SUITE DE FIBONACCI

La suite de Fibonacci est définie par la récurrence suivante :

$$x_0 = 0,\ x_1 = 1,\ x_{n+1} = x_n + x_{n-1}.$$

Ecrivez une fonction qui calcule le $n^{ième}$ terme x_n de la suite de Fibonacci. Ecrivez une fonction qui calcule le $n^{ième}$ terme de la suite définie par $y_n = \frac{x_n}{x_{n-1}}$. Commenter sur la convergence éventuelle des deux suites.

6. RESOLUTION D'EQUATIONS NUMERIQUES NON LINEAIRES

L'objectif est d'obtenir une valeur approchée à une précision donnée ε d'une racine de l'équation numérique f(x)=0 dans un intervalle [a,b], sous les hypothèses suivantes :

f est continue sur [a,b] et f(a).f(b) < 0.

Pour cela il existe différentes techniques d'analyse numérique qui reposent toutes plus ou moins sur le principe suivant : on construit une suite (x_n) telle que (x_n) converge et si $\alpha = \lim_{n\to\infty} x_n$ alors $f(\alpha) = 0$.

Nous retrouverons ce type de méthode pour d'autres classes de problèmes.

6.1 METHODE DE DICHOTOMIE

Cette méthode est classique en algorithmique (appelée aussi "diviser pour régner"). Elle consiste à diviser un problème en deux sous-problèmes plus faciles à résoudre. En effet, si l'on calcule f((a+b)/2), on peut savoir s'il faut chercher une racine de l'équation f(x)=0 dans [a, (a+b)/2] ou dans [(a+b)/2, b].

Soit f une fonction vérifiant les hypothèses précédentes, on définit par récurrence 2 suites (a_n) et (b_n) par :

$a_0 = a$ et $b_0 = b$,

si $f(a_n).f\left(\frac{a_n + b_n}{2}\right) \le 0$ alors $a_{n+1} = a_n$ et $b_{n+1} = \frac{a_n + b_n}{2}$

sinon $a_{n+1} = \frac{a_n + b_n}{2}$ et $b_{n+1} = b_n$

Les deux suites (a_n) et (b_n) sont adjacentes et admettent pour limite une racine de l'équation f(x)=0.

a) Ecrivez deux procédures de calcul approché d'un zéro de l'équation f(x)=0 à une précision donnée ε ($b_p - a_p < \varepsilon$, que vaut p ?), l'une itérative, l'autre récursive.

b) La méthode dichotomique présente deux inconvénients : il faut connaître au départ un intervalle [a,b] sur lequel la fonction change de signe, et une seule racine est trouvée sur cet intervalle. Pour y remédier, on peut découper [a,b] en n sous-intervalles. Sur chacun de ces sous-intervalles, si la fonction change de signe, la méthode dichotomique est appliquée. Adaptez votre programme pour appliquer la méthode vue en a) à n sous-intervalles (n donné par l'utilisateur).

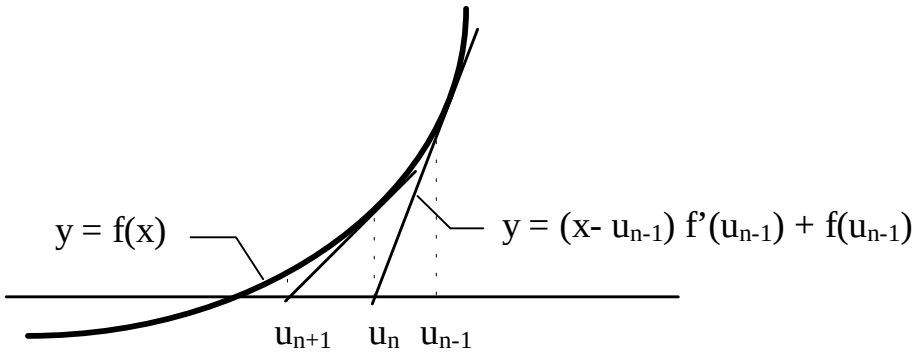
c) Applications :

- calculer $\sqrt{5}$ (racine de $f(x) = x^2 - 5$) en partant de l'intervalle [2, 3]. Combien d'itérations sont nécessaires pour obtenir 6 décimales exactes ?
- trouver les racines de $-2x^4 + 4x^3 + 90x^2 + 68x - 160$. Choisir [-10,10] pour intervalle de départ et 4 sous-intervalles puis 20 sous-intervalles. Interpréter.

- résoudre l'équation x+4*ln(x)-5, dans l'intervalle [1,10], avec 10 sous-intervalles et une précision de 10^{-10}.

6.2 METHODE DE NEWTON

On considère une fonction dérivable à dérivée continue d'un intervalle I dans $\mathbb{R}$ et un réel $x \in I$ tel que $f(x)=0$ et $f'(x) \neq 0$ (f' la dérivée de f ne s'annule pas sur un voisinage de x). Pour obtenir une valeur approchée de x, on linéarise l'équation $f(t) = 0$. Supposant pour cela connue une approximation u_n de x, on remplace alors l'équation $f(t)=0$ par $f(u_n)+f'(u_n).(x-u_n)=0$, ce qui conduit à une nouvelle approximation de x : $u_{n+1} = g(u_n)$ avec $g(t) = t - \frac{f(t)}{f'(t)}$



Graphiquement cela revient à remplacer la courbe au voisinage de u_n par sa tangente en u_n et à prendre pour nouvelle approximation de x l'abscisse u_{n+1} de l'intersection de cette tangente avec l'axe des abscisses :
On partira donc d'un u_0 et on calculera la suite des u_n jusqu'à avoir : $|u_n - u_{n-1}| < \varepsilon$ l'erreur fixée. Cette méthode ne pourra être utilisée pour la recherche de racines multiples, pourquoi ?

a) Ecrivez un programme correspondant à la méthode de Newton.

b) <u>Applications :</u>
- donnez le zéro de la fonction 4*sin(x) - x + 1 avec $u_0 = -3$, $\varepsilon=10^{-10}$. Que se passe-t-il si l'on prend $u_0 = 0$? Pourquoi ?
- testez votre programme sur la fonction x^2.

7. CALCUL MATRICIEL

L'objectif de ce projet est la réalisation d'une bibliothèque de procédures et fonctions pour la manipulation de matrices et vecteurs. On considère que l'on travaille sur des matrices rectangulaires "n lignes, m colonnes".

NB : vous serez attentifs aux performances de vos procédures. Quand un calcul est impossible il faut que votre programme s'arrête proprement.

Vous définirez les types composés suivants :

matrice avec 3 champs : le nombre de lignes, le nombre de colonnes, le contenu de la matrice (c.a.d. un tableau à 2 dimensions)

vecteur avec 2 champs : le nombre de lignes, le contenu du vecteur (c.a.d. un tableau à une dimension).

On définira un type pour les composants de vecteurs ou matrices. Dans un premier temps, on pourra ainsi travailler avec des composants entiers, puis passer à des composants réels (il faudra alors soigner l'affichage).
On considère que l'on travaille toujours avec des matrices de taille maximale (5,5). Dans la suite, on travaillera avec des matrices rectangulaires quand c'est possible.

7.1 PROCEDURES D'ENTREE/SORTIE

a) Lecture d'un vecteur ***lectvect*(V)**
b) Ecriture d'un vecteur ***ecrtvect*(V)**
c) Lecture d'une matrice ***lectmatr*(M)**
d) Ecriture d'une matrice ***ecrtmatr*(M)**

7.2 PROCEDURES DE CALCUL SUR LES VECTEURS ET MATRICES

a) Opérations de recopie : il s'agit simplement de dupliquer le vecteur ou la matrice donné(e) en paramètre. Vous écrirez les procédures ***recopvect*(V,W) et *recopmatr*(A,B)**

b) Ecrivez la procédure ***addmat***(A,B,C) qui réalise l'addition matricielle C = A+B. Quelle restriction faut-il imposer pour les dimensions de A et B?

c) Ecrivez la procédure de transposition ***transpmat***(A,B) : $B = A^T$.

d) Ecrivez la procédure de multiplication de matrice par un réel ***multRmat***(r,A,B) : B = r.A.

e) Ecrivez la procédure de multiplication de matrice par un vecteur ***multmatV***(V,A,B) : B = A.V. Quelle restriction pour les dimensions de A et de V ?

f) Ecrivez la fonction ***tracemat***(A) qui calcule la trace de la matrice A (somme des termes diagonaux).

g) Ecrivez la fonction booléenne ***symmat***(A) qui retourne la valeur *vrai* si la matrice est symétrique, *faux* sinon.

h) Ecrivez la fonction booléenne ***triangsup***(A) qui retourne la valeur *vrai* si la matrice est triangulaire supérieure, *faux* sinon, (idem pour ***trianginf***(A)).

i)

j) Ecrivez la fonction booléenne ***orthmat***(A) qui retourne la valeur *vrai* si la matrice est orthogonale, *faux* sinon. On rappelle que A est orthogonale si $AA^T = A^TA = Id$. Essayer d'optimiser le code de votre fonction.

k) Ecrivez la procédure de produit matriciel ***prodmat***(A,B,C) qui réalise le produit matriciel C = AB. Quelle restriction faut-il imposer pour les dimensions de A et B?

7.3 CALCUL DE LA PUISSANCE D'UNE MATRICE

La procédure ***puissmat***(A,p,B) calcule la puissance $p^{ième}$ de A ($B = \prod_{1}^{p} A = A^p$).

La multiplication matricielle est une opération coûteuse en temps de calcul, on va donc minimiser le nombre de multiplications.

Indication : on fait à priori N-1 multiplications de matrices pour élever A à la puissance N. On introduit ici une méthode qui conduit à un temps de calcul bien moindre quand N est grand. Le principe est : pour calculer A^{2P}, calculer A^P puis multiplier cette dernière par elle-même.

Soient C = Id et D = A
si N est pair, diviser N par 2 et élever D au carré,
si N est impair, décrémenter N de 1 et multiplier C par D,
répéter les opérations précédentes jusqu'à ce que N soit nul.

Que contient alors C ? Si N=14, combien fait-on de multiplications de matrices avec cette méthode ? La procédure ***puissmat***(A,p,B) que vous écrirez mettra en oeuvre cette méthode.

8. RESOLUTION DE SYSTEMES LINEAIRES

Dans la suite, nous chercherons à résoudre des systèmes linéaires de n équations à n inconnues : A.x = b, où A est une matrice (nxn), x et b des vecteurs de dimension n. Puis nous verrons quelques applications complémentaires.

On rappelle que si le déterminant de A est non nul, on a un système de Cramer et la solution du système existe et est unique. Dans ce cas, la solution x est déterminée par la méthode des déterminants :

$x_i = \frac{\det B_i}{\det A}$ où B_i est la matrice obtenue en remplaçant dans A la colonne i par le vecteur colonne b du second membre du système.

Cette méthode n'est pas satisfaisante car elle suppose le calcul de n+1 déterminants, et le calcul d'un déterminant est très coûteux ($D_n = nD_{n-1} + n > n!$, où D_n est le nombre d'opérations à effectuer pour le calcul d'un déterminant de matrice (n,n)).

Un cas particulièrement simple de système de Cramer est celui des matrices triangulaires avec une trace non nulle :

$a_{11}x_1$	$+ a_{12}x_2$	$+ a_{13}x_3$	+	+	$+a_{1n-1}x_{n-1}$	$+a_{1n}x_n$	=	b_1
0	$+ a_{22}x_2$	$+ a_{23}x_3$	+	+	$+a_{2n-1}x_{n-1}$	$+ a_{2n}x_n$	=	b_2
0	0	$+ a_{33}x_3$	+	$+a_{3n-1}x_{n-2}$	$+a_{3n-1}x_{n-1}$	$+ a_{3n}x_n$	=	b_3
.....	.....	.....	.....	.....	.....	.....		.....
0	+ 0	.....	+ 0	$+a_{n-2n-2}x_{n-2}$	$+a_{n-2n-1}x_{n-1}$	$+ a_{n-2n}x_n$	=	b_{n-2}
0	+ 0	.....	.....	+ 0	$+a_{n-1n-1}x_{n-1}$	$+ a_{n-1n}x_n$	=	b_{n-1}
0	+ 0	.....	.....	.....	+ 0	$+ a_{n\,n}x_n$	=	b_n

Le déterminant d'une matrice A triangulaire est égal au produit de ses éléments diagonaux. La résolution d'un système triangulaire est facile (cf. 8.1)

Pour résoudre un système linéaire de n équations à n inconnues, il existe deux types de méthodes : les méthodes directes et les méthodes itératives. Nous verrons un exemple de chaque type.

Méthodes directes : Les méthodes directes permettent d'obtenir la solution exacte d'un système en un nombre fini d'étapes. Le procédé d'élimination de Gauss que nous mettrons en oeuvre dans ce projet est la méthode directe la plus utilisée. On peut citer aussi la méthode de Cholesky qui transforme la matrice initiale en une matrice triangulaire inférieure.

Le procédé d'élimination de Gauss comprend 2 étapes :

- remplacer le système initial par un système à matrice triangulaire possédant la même solution (triangularisation),
- résolution du problème triangulaire.

Méthodes itératives : elles consistent à générer une suite de vecteurs qui converge vers la (ou une) solution du système. Ici, on mettra en oeuvre la méthode de Gauss-Seidel, mais on peut citer la méthode de Jacobi et la méthode de relaxation. Le principe général des méthodes itératives est le suivant :

– soit $A = B - C$ où $\det(B) \neq 0$. Le système $Ax = b$ s'écrit alors $Bx = Cx + b$,
– à partir d'un vecteur x^0 de IR^n quelconque, on construit la suite (x^k) définie par :

$$x^{k+1} = B^{-1} C\, x^k + B^{-1} b = T\, x^k + h,$$

si l'on pose $T = B^{-1} C$ et $h = B^{-1} b$.

Deux questions se posent :

la méthode converge-t-elle ?
comment choisir la décomposition de A pour que B soit *facilement* inversible et que la suite converge *vite*.

NB : Vous ferez une unité de la bibliothèque de fonctions et procédures écrites dans le projet précédent. Les matrices et vecteurs considérés sont à coefficients réels. Vous trouverez en annexe A un commentaire sur les unités en Pascal.

8.1 RESOLUTION DE SYSTEMES TRIANGULAIRES

a) Ecrivez la procédure de résolution d'un système triangulaire supérieur Ax = b (avec A triangulaire supérieure).

b) Ecrivez la procédure qui fournit la solution d'un système triangulaire inférieur.

8.2 METHODE DE GAUSS

Pour trianguläriser le système Ax = b, on utilise les manipulations élémentaires suivantes qui ne changent pas la solution du système,

multiplication d'une ligne par un nombre non nul
ajout une ligne à une autre
permutation de deux lignes

Si A est une matrice (nxn), il y a n–1 étapes pour la triangularisation du système :

1ère étape : on considère le terme a_{11} de la première ligne (s'il est nul voir le NB ci-dessous). On appelle ce terme le pivot. On la soustrait aux n-1 restantes autant de fois que nécessaire (c'est-à-dire en la multipliant par a_{i1}/a_{11}) de façon que les a_{i1} disparaissent (=0), pour i de 2 à n. Le système obtenu est équivalent au précédent.

2ème étape : on recommence avec la matrice ((n-1)x(n-1)) restante et le pivot a_{22}
…
kième étape : on recommence avec la matrice ((n-k+1)x(n-k+1)) restante et le pivot a_{kk}
…
jusqu'à n–1

NB: Si à la suite de soustractions on obtient un zéro sur la diagonale (pivot nul), il faut échanger deux lignes (on choisira celle du pivot optimal c'est-à-dire le plus grand en valeur absolue). Si on ne trouve aucun élément non nul dans la colonne du pivot alors c'est que le système est singulier et admet une infinité de solutions. Il faut alors que votre programme s'arrête proprement.

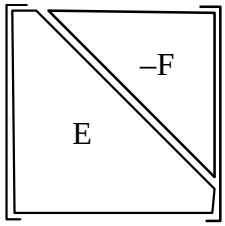
a - Appliquez la méthode du pivot de Gauss pour le système de 4 équations à 4 inconnues suivant :

$$\begin{aligned} 2x_1 &+ x_2 &&+ 4x_4 &&= 2 \\ -4x_1 &- 2x_2 &+ 3x_3 &- 7x_4 &&= -9 \\ 4x_1 &+ x_2 &- 2x_3 &+ 8x_4 &&= 2 \\ &- 3x_2 &- 12x_3 &- x_4 &&= 2 \end{aligned}$$

b - Ecrivez une procédure de résolution de système linéaire suivant le procédé de Gauss. Cette procédure fera appel à une procédure de triangularisation suivant la méthode de Gauss exposée ci-dessus, puis à la procédure de résolution de système triangulaire.

8.3 METHODE DE GAUSS-SEIDEL

La méthode de Gauss-Seidel consiste à décomposer A de la manière suivante :

A= 

A = E–F, la suite est alors définie par : x^0 fixé,
$E\,x^{k+1} = F\,x^k + b$.

Vous écrirez une procédure qui procède à la décomposition de A, et fournit les matrices D–E et F, une procédure qui calcule la solution de Ax = b par la méthode de Gauss–Seidel. On prendra comme test d'arrêt $\bullet x^{k+1}-x^k \bullet < \varepsilon$, où ε est donnée.

NB : cette méthode converge si A est symétrique, définie positive ou bien si A est à diagonale prépondérante ($\forall i, |a_{ii}| > \sum_{j\neq i} |a_{ij}|$)

Les deux applications suivantes utilisent la procédure de triangularisation de Gauss que vous adapterez.

8.4 APPLICATION : DETERMINANT D'UNE MATRICE

En adaptant la procédure de triangularisation de Gauss, écrivez une fonction qui calcule le déterminant d'une matrice A. On rappelle que $\det(A)=(-1)^x \det(D)$, où D est la matrice triangulaire supérieure obtenue avec la méthode du pivot de Gauss, et x est le nombre díinversion de lignes opérées au cours de la triangularisation.

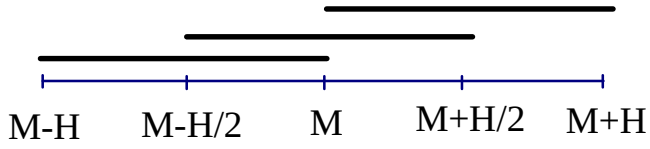
8.5 APPLICATION : INVERSE D'UNE MATRICE

En adaptant la procédure de triangularisation de Gauss, écrivez une procédure qui calcule l'inverse d'une matrice. Trouver l'inverse d'une matrice A consiste à trouver les vecteurs X_i (i=1…n) constituants de la matrice A^{-1} tels que $A.X_i=[0,\ldots,1,\ldots 0]^T$(1 à la ième position).

9. RECHERCHE DES EXTREMUMS D'UNE FONCTION

Il s'agit de trouver le maximum d'une fonction F dans un intervalle donné [a,b], en supposant que la fonction F a un unique maximum dans [a,b]. Pour cela, nous allons utiliser une méthode de type diviser pour régner.

Soit [M-H,M+H] l'intervalle qui contient le maximum (de milieu M). On le divise en 4 intervalles égaux :



Le maximum se trouve dans l'un des 3 intervalles suivants :

$I_1 = [M-H, M]$ $I_2 = [M-H/2, M+H/2]$ $I_3 = [M, M+H]$

si $f(M-H/2) > f(M)$ le maximum est dans I_1
si $f(M+H/2) > f(M)$ le maximum est dans I_3
sinon le maximum est dans I_2.

a) Selon le principe défini précédemment, écrivez une procédure qui trouve le maximum d'une fonction entre a et b, avec une précision E (donnée par l'utilisateur) ; écrivez une procédure qui trouve le minimum d'une fonction selon le même principe.

1. Application aux fonctions :

$f(x) = \sin(x)$	sur l'intervalle $[0, \pi/2]$
$f(x) = \arcsin(\sin(x)$	sur l'intervalle $[0, \pi]$
$f(x) = 1/2.(x+2)^2.(x-1)^3$	sur les intervalles [-3,-1] et [-1,1].

10. CALCULS D'INTEGRALES

Il s'agit d'obtenir une valeur approchée de l'intégrale d'une fonction continue sur un intervalle [a,b]. Pour cela on divise le segment [a,b] en N sous-intervalles de même dimension sur lesquels on utilise une interpolation polynomiale de la fonction f. Les trois méthodes décrites utilisent toutes ce principe.

10.1 METHODE DES RECTANGLES

Comme approximation de l'intégrale entre 2 points a et b, on prend la surface du rectangle correspondant, c'est-à-dire : (b-a)f(a) ou bien (b-a)f(b).

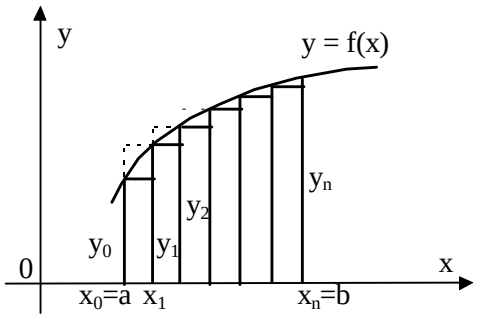


Figure 1 : méthode des rectangles

a) Ecrivez une procédure de calcul d'intégrale utilisant la méthode des rectangles. C'est l'utilisateur qui donnera le nombre de sous-intervalles.
b) Adaptez la procédure écrite précédemment de façon qu'elle puisse donner un encadrement de l'aire calculée (par défaut et par excès)

10.2 METHODE DES TRAPEZES

Comme approximation de l'intégrale entre 2 points a et b, on prend la surface du trapèze, c'est-à-dire : $(b-a)\left(\frac{f(a)+f(b)}{2}\right)$.

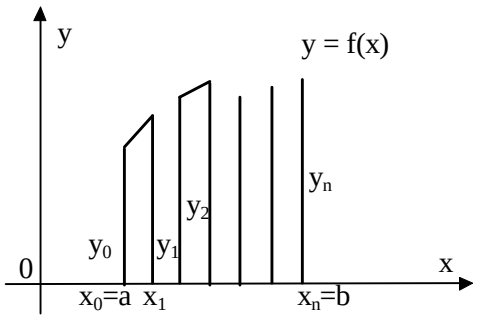


Figure 2 : méthode des trapèzes

a) Ecrivez une procédure de calcul d'intégrale utilisant la méthode des trapèzes. C'est l'utilisateur qui donnera le nombre de sous-intervalles.

10.3 METHODE DE SIMPSON (INTERPOLATION POLYNOMIALE DE DEGRE 2)

Au lieu d'approcher la fonction par une droite (polynôme de degré 1), on fait une approximation par un polynôme de degré 2. Pour cela, on partage l'intervalle sur lequel on veut approcher l'intégrale, en un nombre pair n de sous-intervalles de même dimension. On remplace alors l'aire du trapèze curviligne de base $[x_0, x_1] \cup [x_1, x_2]$, délimité par la courbe $y = f(x)$ par l'aire du trapèze curviligne ayant la même base, délimité par une parabole de second degré passant par les 3 points : $M_0 (x_0,y_0)$, $M_1 (x_1,y_1)$, $M_2 (x_2,y_2)$. On appellera un tel trapèze un trapèze parabolique. La somme des aires des n/2 trapèzes paraboliques fournira une valeur approchée de l'intégrale.

Un trapèze curviligne délimité par (1) l'axe des abscisses Ox, (2) deux droites parallèles à l'axe des ordonnées Oy distantes de 2h, (3) la parabole d'équation $y = Ax^2 + Bx + C$, a pour aire

$$S = \frac{h}{3}(y_0 + 4y_1 + y_2)$$

où y_0 et y_2 sont les ordonnées extrêmes et y_1 celle du milieu du segment (cf. Figure 3).

Comme approximation de l'intégrale entre deux points a et b, on prendra donc :

$$\frac{(b-a)}{6}\left(f(a) + 4f\left(\frac{a+b}{2} \right) + f(b) \right).$$

Figure 3 : méthode de Simpson (1) approximation de l'intégrale, (2) aire d'un trapèze parabolique

a) Ecrivez une procédure de calcul approché d'intégrale pour chaque méthode.

10.4 APPLICATION AUX FONCTIONS

$f(x) = \sin(x)$ sur l'intervalle $[0, \pi]$

$f(x) = x \sin^2(x)$ sur l'intervalle $[1,10]$

$f(x) = \frac{\sin(x)}{x}$ sur l'intervalle $[\frac{\pi}{2}, \pi]$

Pour chaque méthode, comparez les différents résultats en faisant varier le nombre de sous-intervalles. Comparez les trois méthodes.

11. SIMULATIONS UTILISANT LES NOMBRES ALEATOIRES[1]

Ce projet doit vous familiariser avec la fonction **random** de Pascal, mais aussi avec l'unité graphique (cf. Annexe B : notes sur l'unité graphique).

11.1 VALEUR APPROCHEE DE π OU COMMENT UTILISER UN CANON POUR MESURER LA SURFACE D'UN ETANG SITUE AU MILIEU D'UN CHAMP ?!

On « bombarde » le champ de telle sorte que la répartition des points d'impact soit aléatoire; il suffit ensuite de compter le nombre de projectiles qui tombent dans l'eau pour estimer la surface de l'étang : elle est donnée par la surface du champ multipliée par le nombre de projectiles coulés, divisée par le nombre total de tirs. Imaginons donc un étang circulaire inscrit dans un champ carré. Si le canon projette, au hasard, un très grand nombre de projectiles, la proportion de projectiles qui tombent dans l'étang sera égale au rapport de la surface du cercle à celle du carré. Déterminez cette proportion.
On peut effectuer cette simulation au moyen d'un programme simple. Supposons que l'on représente un quart du champ, par exemple le quart Nord-Est, par un carré de côté unité, avec le point (0,0) au centre du champ, et le point de coordonnées (1,1) au sommet N-E du champ. Le tirage successif de 2 nombres aléatoires x et y (fonction *RANDOM*) simule l'arrivée au point (x,y) d'un projectile lancé au hasard dans le quart N-E du champ. Le rapport du nombre de projectiles coulés au nombre total de projectiles est alors égal au rapport du nombre de projectiles coulés dans le quart N-E au nombre d'obus tombés dans ce quart (parce que la répartition est uniforme).
Pour savoir si le coup est tombé dans l'étang, il suffit de calculer la distance entre le pont de chute et le centre du champ.
Suivant les indications qui précèdent, écrivez un programme qui calcule une valeur approchée de π. En exécutant votre programme, étudiez l'influence du nombre de projectiles lancés sur la précision du résultat.

11.2 LA PLANCHE DE GALTON

Sir Francis Galton (1822-1911), l'un des pionniers des statistiques, imagina une planche inclinée comportant un réseau triangulaire de clous. On lâche successivement des billes à partir d'un point situé au-dessus du clou le plus élevé; elles se frayent un chemin entre les obstacles et on les récolte dans des cases disposées au bas de la pente. Lorsque la dernière bille arrive en bas, l'empilement des billes dans les réceptacles dessine un profil carctéristique. A défaut d'une véritable planche de Galton, le programme que je vous demande d'écrire, permet d'en simuler une.
Le trajet de chaque bille est simulé, à l'intérieur d'une boucle, par une suite de nombre aléatoires. Chacun de ces nombres correspond à un clou et permet de décider si la bille passe à droite ou à gauche de ce clou. Lorsque le nombre aléatoire est inférieur ou égal à 0,5 la bille passe à gauche, dans le cas contraire elle passe à droite. De quel clou s'agit-il? Aucune importance; il suffit d'augmenter d'une unité la valeur d'un compteur chaque fois que la bille passe à droite d'un clou. Vous pourrez vérifier que le numéro de la case d'arrivée d'une bille ne dépend que du nombre de passage à droite des obstacles placés sur sa route. Il est donc

[1] *Récréations Informatiques* A.Dewdney, Pour la Science, Juin 1985

parfaitement inutile de faire figurer dans le programme un tableau représentant individuellement chaque clou.
La boucle qui simule la descente de chaque bille est incluse dans une boucle plus grande, qui spécifie le nombre de billes que l'on veut utiliser. Un tableau matérialisera les cases d'arrivées des billes. Après avoir lancé n billes, le programme affichera l'histogramme du tableau des cases d'arrivée.

12. TRACE DE FONCTIONS

L'objectif de ce projet est de réaliser un programme de tracé de courbes d'équations cartésiennes, paramétrées ou polaires.

12.1 QUELQUES PRECISIONS

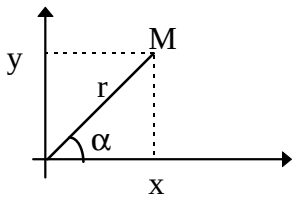
Courbes d'équation y = f(x)
L'axe horizontal étant celui des abcisses x, on associe à chaque x la valeur f(x) correspondante, on obtient ainsi le point (x,f(x)).

Méthode de tracé : la courbe est obtenue en faisant varier x régulièrement (incrément fixe lu au clavier) et en calculant f(x) pour chaque x. Pour assurer la continuité de la courbe, on trace un segment de droite entre le point (x,f(x)) et le point précédent.

Courbes d'équations paramétrées
Elles sont définies par deux équations donnant les coordonnées x et y d'un point de la courbe en fonction d'un paramètre t que l'on fait varier : $x = f(t), y = g(t)$
On peut remarquer que les courbes d'équation y = f(x) sont un cas particulier de courbes paramétrées ; en effet si on pose x = t on a : $x = t, \quad y = g(t) = g(x)$

Méthode de tracé : identique à la précédente.

Courbes d'équation polaire
Une courbe polaire est une courbe dont l'équation est définie à partir des coordonnées polaires des points qui la constituent :



Les coordonnées polaires sont alors : la distance r de M à l'origine
l'angle α défini par les demi-droites Ox et OM

Pour passer des coordonnées polaires aux coordonnées rectangulaires on applique :

$$x = r \cos \alpha, \quad y = r \sin \alpha$$

Une équation polaire sera du type $r = f(\alpha)$.

Méthode de tracé : le tracé des courbes polaires, se ramène au tracé de courbes paramétrées.

Fonctions
Les fonctions seront définies comme des fonctions Pascal avec deux paramètres : la variable x et le numéro de la fonction. Une instruction **case** permettra de calculer la bonne fonction.

12.2 TRAVAIL A EFFECTUER

NB : suivez l'ordre des tâches à effectuer. Respectez le nom des procédures et des variables.

1/ La procédure **traceaxes** effectue la préparation du tracé d'une courbe quelconque :

- elle calcule l'échelle sur chaque axe en fonction des valeurs minimales et maximales de x et de y,
- elle calcule les coordonnées du centre du repère (toujours sur l'écran) et les coefficients multiplicateurs permettant de convertir les valeurs de x et de y en nombre de points sur l'écran,
- elle trace des axes gradués, et éventuellement une grille permettant le repérage des points particuliers.

Les valeurs des graduations sont calculées de façon à optimiser la lisibilité. Ce sont des puissances entières de 10 (par exemple 0.01; 1 ; 10 ...). Ces valeurs seront affichées en bas l 'écran (PX pour l'axe des x, PY pour l'axe des y).

Procédure traceaxes

Données : xn valeur minimale de x (coordonnées papier) *real*
xm valeur maximale de x (coordonnées papier) *real*
yn valeur minimale de y (coordonnées papier) *real*
ym valeur maximale de y(coordonnées papier) *real*
axes, vrai si l'utilisateur choisit de voir les axes sur l'écran *boolean*
grille, vrai si l'utilisateur choisit un quadrillage *boolean*

Résultats : ex et ey coefficients multiplicatifs permettant de convertir des coordonnées papier en coordonnées écran (pixels) *real*
xa, ya coordonnées du centre du repère (en pixels) *integer*

Locales (entres autres) : px, valeur de la graduation en x *real*
py, valeur de la graduation en y *real*
Dx distance entre xn et xm *real*
Dy distance entre yn et ym *real*

Rectification des coordonnées maximales :
si xn.xm > 0 alors si xn < 0 alors xm :=0 sinon xn:=0
si yn.ym > 0 alors si yn < 0 alors ym :=0 sinon yn :=0

Calcul de l'échelle : c'est une règle de trois !
Calcul du centre du repère : attention aux positions du point (0,0) sur l'écran
Détermination dela graduation :
Elle doit être constituée de puissances de 10 (dx distance entre xn et xm) :
si dx = 3 ($=3.10^0$) alors px = 0,1 $= 10^{-1} = 10^{0-1}$
50 ($=5.10^1$) alors px = 1 $= 10^0 = 10^{1-1}$
400($=4.10^2$) alors px = 10 $= 10^1 = 10^{2-1}$

2/ La procédure **courbedis** affiche la représentation graphique d'une fonction discrète (définie en un nombre fini de points). On joindra deux points consécutifs par un segment de droite, on aura ainsi une ligne brisée.
Vous écrirez avant tout un type point (un point est défini par son abscisse et son ordonnée).

Procédure courbedis

Données : nbv, nombre de valeurs
valeurs, tableau contenant les nbv points
xn et xm, bornes sur l'axe horizontal
yn et ym, bornes sur l'axe vertical
axe, grille et ortho, booléens transmis à Traceaxes

Dans un premier temps, vous travaillerez avec un tableau où les points sont ordonnés selon leur abscisse croissante, (xn et xm sont alors immédiatement déterminées), l'utilisateur donnera les valeurs de yn et ym.
Vous écrirez ensuite une procédure qui détermine yn et ym (si l'utilisateur a donné des valeurs égales, c'est qu'il souhaite que le programme les détermine automatiquement). Enfin, vous adapterez votre programme de façon qu'il puisse traiter un tableau où les points ne sont pas rangés. Il faudra effectuer un tri du tableau.

3/ La procédure **courbeyfx** permet le tracé d'une courbe d'équation y=f(x).

Procédure courbeyfx

Données : xn et xm, bornes sur l'axe horizontal
yn et ym, bornes sur l'axe vertical
axe, grille et ortho, booléens transmis à Traceaxes

Attention, il ne faut pas ranger les points dans un tableau, c'est inutile ! L'axe des abscisses est balayé de ∂x en ∂x (que l'utilisateur pourra donner), et les ordonnées sont calculées. Comme précédemment, vous demanderez à l'utilisateur de donner yn et ym. S'il donne deux valeurs égales, votre programme doit alors déterminer le minimum et le maximum de la fonction sur l'intervalle [xn,xm]. Vous écrirez simplement un procédure qui balaie l'intervalle [xn,xm] (avec des sauts d'une amplitude dépendant de la taille de l'intervalle).

4/ La procédure **courbeparam** permet le tracé de courbes paramétrées ou polaires. Cette procédure utilisera deux fonctions f et g.

Procédure courbeparam

Données : tn et tm, intervalle de variation du paramètre
xn et xm, bornes sur l'axe horizontal (si xn=xm alors calcul automatique)
yn et ym, bornes sur l'axe vertical
axe, grille et ortho, booléens transmis à Traceaxes

12.3 JEUX D'ESSAI

Testez d'abord vos programmes sur des fonctions simples dont vous connaissez la représentation graphique.

Courbes y = f(x)	$y = 5x/(1+x^2)$**,**	$x \in [-10, 10]$
	$y = e^{-x^2}$	$x \in [-3, 3]$
	$y = \sin x/x$	$x \in [-20, 20]$

Courbes paramétrées et polaires

$x = 2t - 3\sin t,\ y = 2 - 3\cos t$		$t \in [-9, 9]$	(trochoïde)
$x = t - \sin t,\ y = 1 - \cos t$		$t \in [-9, 9]$	(cycloïde)
$x = \cos^3 t,\ y = \sin^3 t$		$t \in [0, 2\pi]$	(hypocycloïde)
$r = \theta$	$\theta \in [0, 40]$		(spirale d'Archimède)
$r = \frac{\sin\theta}{\theta}$	$\theta \in [-30, 30]$		(cochléoïde)
$r = 2\cos 2\theta - \cos\theta$	$\theta \in [0, 2\pi]$		(rosace à 4 branches)

13. RESOLUTION APPROCHEE D'EQUATIONS DIFFERENTIELLES

On rappelle qu'une équation différentielle est une équation dans laquelle interviennent, en plus de la variable x et de la fonction y (y=f(x)), les dérivées successives de la fonction y, jusqu'à un certain ordre, appelé ordre de l'équation différentielle. Nous allons nous contenter ici d'étudier les équations différentielles d'ordre 1, et les méthodes numériques permettant une résolution approchée.
Une équation différentielle n'admet généralement pas une solution unique. Il est donc nécessaire de fournir des informations supplémentaires sous la forme de conditions initiales qui permettent de déterminer **une** solution.
Les solutions que nous obtiendrons seront sous forme discrète, c'est-à-dire sous forme d'une suite de points. On pourra la tracer à l'aide du programme de tracé de fonctions discrètes.

13.1 EQUATIONS DIFFERENTIELLES DU 1ER ORDRE ET METHODE D'EULER

Soit l'équation y' = f(x,y) , avec les valeurs initiales x_0 et y_0. On demande de trouver sa solution dans un certain intervalle $[x_0 , x]$. Le principe de la méthode d'Euler est le suivant :

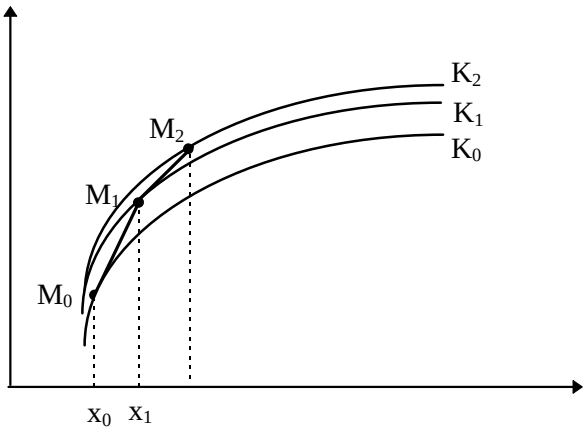
- on divise l'intervalle $[x_0 , x]$ en n sous-intervalles $[x_i , x_{i+1}]$,
- dans $[x_0 , x_1]$, on pose $y = y_0 + f(x_0 , y_0) (x-x_0)$; autrement dit, au lieu de la courbe intégrale M_0K_0 cherchée, on prend sa tangente au point x_0 (M_0M_1),
- au point x_1, on détermine la valeur approchée de la solution : $y_1= y_0 + f(x_0 , y_0) (x_1-x_0)$,
- dans $[x_1 , x_2]$, on pose $y = y_1 + f(x_1 , y_1) (x-x_1)$; autrement dit, au lieu de la courbe intégrale M_0K_0 cherchée, on prend la tangente M_1M_2 à la courbe intégrale M_1K_1 au point x_1 ; une double erreur apparaît alors : la tangente M_1M_2 s'écarte de la courbe M_1K_1, et cette dernière ne coïncide pas avec la courbe cherchée M_0K_0.
- poursuivant ce processus, nous obtenons les valeurs approchées successives :

  $y_1= y_0 + f(x_0 , y_0) (x_1-x_0)$,

  $y_2 = y_1 + f(x_1 , y_1) (x_2-x_1)$,

  ..... ,

  $y_n = y_{n-1} + f(x_{n-1} , y_{n-1}) (x_n-x_{n-1})$.



Cette méthode est simple mais peu performante en raison du cumul de l'erreur commise. Elle est utilisée pour des approximations grossières.

a) Ecrivez une procédure qui implémente la méthode d'Euler pour une équation différentielle du premier ordre.

b) Trouvez la solution approchée de l'équation : $y' = \frac{1}{2} xy$ dans l'intervalle [0,1], pour les conditions initiales $x_0=0$, $y_0=1$. Vous diviserez l'intervalle en 10 parties égales. Quelle est la solution exacte de cette équation ? Vous afficherez les valeurs approchées de y et ses valeurs exactes. Quelle est l'erreur commise pour x=1 ?

13.2 METHODE DE RUNGE-KUTTA (ORDRE 4)

Soit l'équation $y' = f(x,y)$, avec les valeurs initiales x_0 et y_0. On demande de trouver sa solution dans un certain intervalle $[x_0 , x]$. Cette méthode utilise le développement de Taylor de la fonction y à l'ordre 4.

A partir d'un point (x_i , y_i), le point suivant est déterminé par les formules suivantes (h est la distance séparant x_i et x_{i+1}) : $k_0 = h\, f(x_i , y_i)$,

$$k_1 = h\, f(x_i + \frac{h}{2}, y_i + \frac{k_0}{2})$$

$$k_2 = h\, f(x_i + \frac{h}{2}, y_i + \frac{k_1}{2})$$

$k_3 = h\, f(x_i + h, y_i + k_2)$

alors, $y_{i+1} = y_i + \frac{1}{6}(k_0 + 2\, k_1 + k_3)$.

La précision est meilleure que pour la méthode d'Euler, le principal inconvénient restant le temps de calcul (4 évaluations de la fonction à chaque pas). Notons que la méthode d'Euler suit le même principe avec un développement de Taylor à l'ordre 1.

a) Ecrivez une procédure de résolution d'équations différentielles par la méthode de Runge Kunta.

b) Trouvez la solution approchée de l'équation : $y' = \frac{4xy + 4x\sqrt{y}}{1+x^2}$ avec la condition initiale $x_0 = 0$, $y_0 = 1$. Vérifiez que la solution exacte de cette équation est $y = (2x^2 + 1)^2$. Affichez sous forme de tableau les points des solutions exacte et approchée dans l'intervalle [0, 4].

c) D'après la loi de Newton, la vitesse de refroidissement d'un corps quelconque dans l'air est proportionnelle à la différence des températures du corps et du milieu. La température de l'air étant de 20°C, le corps se refroidit de 100° à 60°C en l'espace 20 minutes. On demande en combien de temps sa température tombera à 30°C. Il faudra tout d'abord écrire l'équation différentielle du problème et donner les conditions initiales.

13.3 GENERALISATION AUX SYSTEMES A p EQUATIONS D'ORDRE 1 :

Il s'agit de déterminer non plus une mais p fonctions, solutions d'un système différentiel du premier ordre de la forme :

$$y_1' = f_1(x, y_1, y_2, ..., y_p),$$
$$y_2' = f_2(x, y_1, y_2, ..., y_p),$$
$$.....,$$
$$y_p' = f_p(x, y_1, y_2, ..., y_p).$$

a) Adaptez les procédures d'Euler et de Runge-Kutta au cas de systèmes à p équations ($p \leq 9$).
b) Comparez les résultats approchés obtenus par la méthode d'Euler, celle de Runge-Kutta et le résultat exact pour le système différentiel suivant :

$y_1' = y_1 + y_2 + \sin(x)$,

$y_2' = -y_1 + 3\,y_2$, avec $y_1(0) = \frac{-9}{25}$ et $y_2(0) = \frac{-4}{25}$.

Vérifiez que la solution exacte est : $y_1 = \frac{-1}{25}(13\sin(x) + 9\cos(x))$,

$y_2 = \frac{-1}{25}(3\sin(x) + 4\cos(x))$

14. ANNEXE A : NOTES SUR LES UNITES EN TURBO PASCAL

Les unités permettent une programmation modulaire. On utilise des unités pour créer des bibliothèques de programmes ou bien pour diviser de gros programmes en plusieurs modules. Vous savez déjà utiliser des unités, vous avez déjà fait appel à l'unité **crt**.

14.1 QUELQUES UNITES STANDARDS[2]

Nom de l'unité	**Rôle et exemples de fonctions ou procédures disponibles**
CRT :	procédures de contrôle de l'ordinateur : l'écran, le clavier, le son ... **clrscr :** efface l'écran et positionne le curseur en haut à gauche (proc.) **gotoxy :** positionne le curseur aux coordonnées spécifiées (proc.) **keypressed :** détermine si une touche du clavier a été sollicitée (fun.)
DOS :	procédures du système opérationnel. **getdate :** donne la date du système.[3] **gettime :** donne l'heure du système.
SYSTEM :	toutes les procédures et fonctions 'prédéfinies'que vous utilisez. On n'a pas besoin de spécifier qu'on va faire appel à cette unité. fonctions arithmétiques : **abs, pi, sqr, sqrt, ...** fonctions d'allocation dynamique : **dispose, new, ...** fonctions et procédures diverses : **random, read, write, dec, inc, ...**
GRAPH :	procédures pour le mode graphique. **circle :** dessine un cercle à l'écran (proc.) **line :** dessine une ligne (proc.)

14.2 DEFINIR UNE UNITE

Une unité est constituée de quatre parties :

1. **une entête,** c'est le nom de l'unité précédé du mot réservé **`unit.`** C'est ce nom que qui apparaît dans le programme utilisant cette unité. Il faut que le nom du fichier contenant l' unité soit le même que celui de l'entête.
2. **une interface,** c'est la partie où sont déclarées les constantes, types, variables, procédures et fonctions qui sont publics, c'est-à-dire visibles, accessibles lors de l'utilisation de l'unité. Les procédures et fonctions apparaissent seulement par leur entête dans cette partie, leur corps se trouve dans la partie implémentation.
3. **une implémentation,** où l'on trouve le corps de toutes les fonctions et procédures publiques. C'est la partie privée de l'unité. Elle peut contenir des déclarations qui ne seront pas visibles de l'extérieur de l'unité.

---

[2] Turbo Pascal sous MS-DOS

[3]
```
program exemple;
    uses dos;
    var a,m,j,jds:word;
    begin
        getdate(a,m,j,jds);
        writeln('c est aujourd hui le',j,'/',m,'/',a,'  ',jds,'ème jour de la semaine');
    end.
```

4. **une initialisation,** soit elle est réduite au mot réservé **end** , soit d'une partie de code qui qui doit être exécuté pour initialiser l'unité.

Voici donc la syntaxe d'une unité :

```
unit identificateur ;
interface          {symboles publics}
      uses               {appel à d'autres unités};
      const              {déclarations de constantes publiques} ;
      type               {déclarations des types  publics} ;
      var                {déclarations de variables publiques} ;
      procedure          {déclarations des procédures publiques (entêtes)} ;
      function     {déclarations des fonctions  publiques (entêtes)} ;
implementation     {symboles privés}
      uses               {appel à d'autres unités};
      const              {déclarations de constantes privées} ;
      type               {déclarations des types  privées} ;
      var                {déclarations de variables privées} ;
      procedure          {déclarations des procédures publiques et privées (corps)} ;
      function     {déclarations des fonctions  publiques et privées (corps)} ;
      begin              {initialisation}
            instruction ;
            instruction ...
end.
```

On enregistrera l'unité sous le même nom que son identificateur (ce sera un fichier `.pas`). La compilation d'une unité génère un fichier `.tpu` qui devra être enregistré sur le disque (cf. option *Destination* dans le menu *Compile*).

14.3 COMMENT UTILISER UNE UNITE

C'est le mot réservé `uses` qui permet de spécifier qu'un programme utilisera une unité. La syntaxe est la suivante :
`tpu` *ident_unit1, ident_unit2, ...* ;
Pour localiser une unité, le compilateur cherche dans le répertoire courant le fichier *ident_unit.TPU*. On peut aussi spécifier d'autres répertoires dans :

Options|Directories|Unit Directories.

15. ANNEXE B : NOTES SUR L'UNITE GRAPHIQUE (BGI)[4]

L'unité **Graph** est dotée d'une bibliothèque de routines graphiques, une démonstration est disponible (programme bgidemo.pas).
L'exécution d'un programme utilisant l'unité **Graph** nécessite un ou plusieurs pilotes graphiques (fichiers **.BGI**). Si votre programme utilise des polices vectorielles, vous aurez besoin aussi d'un ou plusieurs fichiers de polices de caractères (**.CHR**). La variable **PathToDriver** doit être positionnée au chemin d'accès au répertoire contenant ces fichiers.

Passer du mode vidéo au mode graphique :
La procédure **InitGraph** est chargée d'identifier l'équipement graphique, de charger et initialiser le pilote graphique approprié, de placer le système en mode graphique, enfin elle rend le contrôle au programme qui l'a sollicitée.
La procédure **CloseGraph** décharge le pilote de la mémoire et restaure le mode vidéo précédent.

Système de coordonnées :

Il est conseillé d'utiliser les fonctions **GetMaxX** (resp. **GetMaxY**) pour obtenir le n° de la colonne la plus à droite (resp. le n° de la ligne la plus basse) pour le mode et le pilote graphiques courants.
Le curseur du mode texte est remplacé par le pointeur courant (qui n'est pas visible à l'écran). La commande **MoveTo** est l'équivalent de **GoToXY**. Sa seule action est de déplacer le pointeur courant.

Afficher du texte en mode graphique :
Pour l'affichage de texte, le mode graphique dispose d'une police de caractères bitmap 8x8 et de plusieurs polices vectorielles. Un caractère bitmap est défini par une matrice de 8x8 pixels. Une police vectorielle est définie par une série de vecteurs qui donnent au système graphique les informations nécessaires pour dessiner cette police. Ce dernier type sera utilisé quand on voudra afficher des caractères agrandis (la résolution d'une police bitmap agrandie est dégradée).
SetTextStyle permet de choisir la police et l'échelle. La justification d'un texte graphique est contrôlée par la procédure **SetTextJustify**. L'affichage d'un texte graphique s'effectue soit en appelant la procédure **OutText** soit la procédure **OutTextXY**. La taille des polices vectorielles peut être définie au moyen de la procédure **SetUserCharSize**.

Couleurs :
GetMaxColor renvoie le plus grand numéro de couleur qui puisse être transmis à **SetColor** qui définit la couleur de tracé courante. On peut aussi changer la couleur de fond courante avec **SetBkColor**.

[4] Turbo Pascal sous MS-DOS

Figures et Styles :

Un certain nombre de figures (points, lignes, arcs, ellipses, rectangles, histogrammes, ...) et styles (remplissage, style de trait, ...) sont disponibles.

SetLineStyle permet de contrôler l'épaisseur des lignes et choisir leur motif (pointillées, continues, etc.).

SetFillStyle, **SetFillPattern**, **FillPoly** et **FloodFill** permettent le remplissage d'une région, d'un polygone par des hachures ou autres modèles.

Bar et **Bar3D** tracent un histogramme avec la couleur et le style de remplissage courants.

Circle trace un cercle centré sur (x,y).

DrawPoly trace le contour d'un polygone en utilisant la couleur et le style de remplissage courants.

Line trace une ligne de (x1,y1) à (x2,y2); **LineTo** trace une ligne du pointeur courant jusqu'au point (x,y).

PieSlice trace un camembert centré sur (x,y) à partir d'un angle de début jusqu'à un angle de fin, puis le remplit.

Rectangle trace un rectangle en utilisant la couleur et le style de remplissage courants.

... Et bien d'autres !

Exemple (planche de Galton) : (voici les procédures pour le dessin de l'histogramme; vous devrez compléter la partie qui simule le remplissage des cases et les points signalés ☛)

```
program galton;
uses crt, graph;
...
var ...
   GraphDriver    : integer;  { The Graphics device driver }
   GraphMode      : integer;  { The Graphics mode value }
   MaxX, MaxY     : word;     { The maximum resolution of the screen }
   MaxColor       : word;     { The maximum color value available }

procedure Initialize;
{ Initialisation du mode graphique  }
var
PathToDriver   : string;                { chemin DOS pour les fichiers *.BGI & *.CHR }
begin
   GraphDriver := Detect;               { autodétection }
   PathToDriver := ☛ ;
   InitGraph(GraphDriver, GraphMode, PathToDriver);
    Randomize;                          { initialisation du générateur de nb aléatoires }
    MaxColor := GetMaxColor;            { Numéro de couleur maximum }
    MaxX := GetMaxX;                    { Valeurs maximales des coordonnées }
   MaxY := GetMaxY;
end; { Initialize }

function RandColor : word; { rend une valeur de couleur non nulle au hasard }
begin
  RandColor := Random(MaxColor-2)+1;
end; { RandColor }
```

```
procedure Titre; {affiche le titre}
begin
    SetColor(MaxColor);       {couleur du titre à blanc}
    SetTextJustify(0,2);      {voir l'aide pour les constantes de justification}
    SetTextStyle(1,0,1);      {voir l'aide pour les constantes de styles de texte}
    OutTextXY(0,0,'Histogramme de');
    OutTextXY(0,20,'la planche de Galton')
end; {Titre}

procedure  Histogramme(NumBars:integer;T:tab); {dessine l'histogramme}
var
  X1,Y1,I   : integer;
  Color     : word;
  Largeur   : integer;
  Hauteur   : integer;
  Max       : integer;
begin
   Initialize; {initialisation du mode graphique}
   Max:=0;
   for I:=1 to NumBars do if T[I]>Max then Max:=T[I]; {Max est le plus grand nombre de
                                                       billes entassées dans une case}
   Largeur:= ☛  {Largeur des barres}
   X1:=- 0;
   for I := 1 to NumBars do
   begin
      Y1 := ☛  {Hauteur de la barre correspondant à T[I] }
      X1:=X1+Largeur;
      SetFillStyle(9, Color);   {définition du style de remplissage}
      Color := RandColor;       {couleur choisie au hasard}
      SetColor(Color);          {définition de la couleur}

      if Hauteur=0 then Line(☛) {si hauteur =0, tracer un trait }
      else Bar(☛); {sinon tracer une barre de la hauteur donnée}
    end;
    Titre; {affichage du titre}
    ReadLn;
    CloseGraph; {sortie du mode graphique}
end; { Histogramme}

begin {Galton}
...
end. {Galton}
```

16. EXERCICES COMPLEMENTAIRES

Ces exercices sont tous tirés d'examens des années précédentes.

16.1 NOMBRES PARFAITS

Un nombre entier est dit parfait s'il est égal à la somme de tous ses diviseurs (lui-même étant exclu). Par exemple, 6 est un nombre parfait car 6 = 1 + 2 + 3.
Ecrivez un programme qui lit un nombre n < 5 au clavier, et affiche les n premiers nombres parfaits. Ce programme utilisera une fonction booléenne *parfait*(p) , que vous définirez, qui retourne la valeur 'vrai' si le nombre p passé en paramètre est parfait, 'faux' sinon.

16.2 PARCOURS DE TABLEAU (PLATEAU D'ARSAC)

Un tableau T d'entiers positifs est trié en ordre croissant. Ecrivez une fonction *long-plat* qui donne la longueur du plus grand plateau (suite d'éléments consécutifs égaux) dans T.
Exemple : T=[2,5,5,7,7,7,7,8,9,9,9,13] *long-plat*(T) vaut 4.

16.3 ENTIERS LONGS

Nous allons remédier, dans une certaine mesure, au problème de calcul sur des entiers trop grands pour être représentés à l'aide du type prédéfini *integer* de Pascal (limité à la valeur 32 767). On va définir pour cela un nouveau type *entier_long* comme un tableau d'entiers. Nous allons découper nos entiers en tranches, chaque tranche étant stockée dans une case du tableau. Nous conviendrons qu'une tranche est composée de 2 chiffres ; par exemple, nous aurons 123456789 représenté par le tableau

5	4	3	2	1
1	23	45	67	89

Définir le type *entier_long* (on prendra une taille maximale égale à 250).

Si a est un *entier_long*, sa valeur A s'obtient à partir de sa représentation de la manière suivante :

$$A = \sum_{i=1}^{250} a[i]\ 10^{2(i-1)} ,$$

Justifiez l'égalité précédente.

Il faut maintenant écrire les procédures de base qui permettent de découper un entier en tranches, d'afficher un *entier_long* et de faire des produits.

a) Pour pouvoir initialiser nos entiers longs, nous allons écrire une procédure **allonger** qui transforme un entier en un entier long. Par exemple, allonger(12345, e) donnera le tableau e suivant

e = | 1 | 23 | 45 |

b) Ecrivez la procédure **ecriture** qui permet de faire afficher un *entier_long* e. Attention, les zéros du début ne doivent pas être affichés (sauf pour l'entier 0); pour savoir où commencer l'affichage, on écrira une fonction **taille** qui indique le plus grand indice des

cases non nulles de e. L'affichage des contenus des cases de e ne se fera donc qu'à partir de la case taille(e).

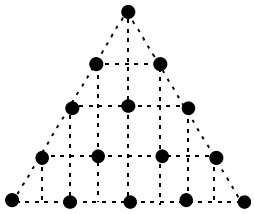
c) Tester la fonction **taille** et la procédure **ecriture** sur les entiers n1=0, n2=1234567890000 et n3=10203. Pour cela vous initialiserez trois tableaux correspondants à n1, n2 et n3. Il faut bien vérifier que la procédure **ecriture** affiche les 4 zéros de 1234567890000, et qu'elle affiche 10203 et pas 123.

d) Il s'agit maintenant d'écrire la procédure **produit** qui multiplie un entier long e par un entier usuel n, et donne dans une variable **retenue** ce qu'on mettrait dans la case 251 s'il y en avait une. Par exemple, si e[250] = 10, e[249] = e[248] = ... = e[1] = 0, le produit de e par n=35 donnera e[250] = 50 et retenue = 3. Pour écrire cette procédure, il faut procéder exactement comme lorsqu'on fait une multiplication « à la main ». On effectue le calcul tranche par tranche et, à chaque étape, on effectue le produit de n par une tranche de e, on ajoute à ce produit la retenue de l'étape précédente puis on découpe le résultat en une tranche que l'on écrit en place et en une retenue pour l'étape suivante.

e) Ecrivez un programme qui demande un nombre entier et affiche sa factorielle.

16.4 NOMBRES TRIANGULAIRES

On appelle nombre triangulaire tout entier naturel $t \in IN$ qui se met sous la forme $t = 1/2\ p(p+1)$ où $p \in IN$. Ce nom provient du fait que le nombre de points dans un triangle comme dans la figure ci-dessous est précisément un nombre triangulaire $(15 = 1/2(5)(5+1))$:



Gauss a fait l'observation suivante : un nombre t est triangulaire si et seulement si $(8\ t +1)$ est le carré d'un nombre impair.

1) En utilisant le critère de Gauss, écrivez un programme qui affiche les n premiers nombres triangulaires, n étant donné par l'utilisateur.

2) Proposez une récurrence permettant de déterminer un nombre triangulaire en fonction du ou des précédants. Ecrivez un programme qui utilise cette récurrence pour donner les n premiers nombres triangulaires.

16.5 ARITHMETIQUE SUR IN

On se propose ici d'écrire les opérations fondamentales sur IN en utilisant uniquement les axiomes de Péano, c'est-à-dire en utilisant uniquement les fonctions *succ* et *pred* (donnant respectivement le successeur et le prédécesseur d'un entier).

En utilisant le signe = (test d'égalité), l'opérateur booléen *not* et les fonctions *succ* et *pred*, écrire les procédures ou fonctions suivantes (attention, prendre le prédécesseur de 0 doit être interdit, toutes les opérations prédéfinies +, –, *, div, mod, inégalités (<, >) sont exclues, la boucle pour est également interdite car elle fait appel à l'addition d'entiers) :

- *inf_ou_egal* qui teste si un entier naturel est inférieur ou égal à un autre ;
- *addition* qui effectue l'addition de deux entiers naturels ;
- *soustraction* qui effectue la soustraction de deux entiers naturels, quand c'est possible ;
- *multiplication* qui effectue la multiplication de deux entiers naturels ;
- *division* : soient a et b, il existe un unique entier q et un unique entier r inférieur strictement à b tels que a = bq +r ; *division* détermine ces deux nombres.

16.6 ARITHMETIQUE

Il s'agit d'écrire un programme (avec les structures de données, procédures et/ou fonctions appropriées) qui affiche du plus grand au plus petit les nombres n compris (bornes incluses) entre 2 et 99 vérifiant la propriété : la somme des diviseurs de n est inférieure ou égale à la somme des diviseurs de n-1.
Vous afficherez également la somme des diviseurs du nombre entre parenthèses.
L'affichage se terminera donc par : 9(13), 7(8), 5(6).

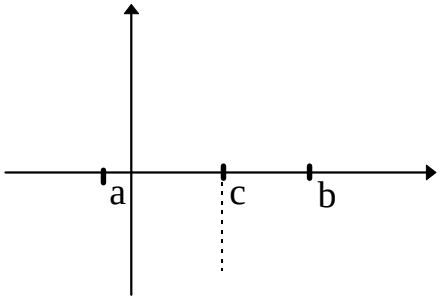
16.7 SUITES DE FIBONACCI ET MINIMUM D'UNE FONCTION

1 On rappelle que les nombres de Fibonacci sont définis par récurrence :

$u_0 = 0$, $u_1 = 1$, $u_n = u_{n-1} + u_{n-2}$

Etant donné un entier $n \geq 2$, écrivez la procédure *fibonacci* qui fournit les trois termes u_{n-1}, u_n et u_{n+1} de la suite de Fibonacci.

2 On suppose que la fonction f à valeurs réelles est deux fois continûment dérivable sur un intervalle $[a,b] \subset IR$, telle que $f''(x)>0$, pour tout $x \in [a,b]$. Enfin on suppose qu'il existe un point $c \in]a,b[$ tel que $f'(c)=0$.



Le but du problème est de donner une approximation de c. Pour cela, on définit un algorithme de dichotomie.
Soient x_1 et x_2 tels que $a \leq x_1 < x_2 \leq b$, on a les implications suivantes

- si $f(x_1) \geq f(x_2)$ alors $x_1 < c < b$ (continuer la recherche dans l'intervalle $]x_1,b[$)
- si $f(x_2) \geq f(x_1)$ alors $a < c < x_2$ (continuer la recherche dans l'intervalle $]a,x_2[$)

Pour un intervalle [a,b], les points x_1 et x_2 sont définis par (n est un entier fixé, esl termes u_{n-1}, u_n et u_{n+1} ont été calculés dans la question précédente) :

$$x_1 = a\frac{u_n}{u_{n-1}} + b\frac{u_{n-1}}{u_{n+1}}$$
$$x_2 = a\frac{u_{n-1}}{u_{n+1}} + b\frac{u_n}{u_{n+1}}$$

Ecrivez la fonction *minimum* qui donne la valeur de c, le point où f est minimale avec une précision *e* donnée Vous supposerez que f est une fonction Pascal donnée, que n est donné (par l'utilisateur).

16.8 SUITES ET CALCUL DE RACINE CARREE

On définit les suites a_n et c_n par : $a_0 = x$

$$c_0 = 1 - x$$
$$a_i = a_{i-1}\left(1 + \frac{c_{i-1}}{2}\right)$$
$$c_i = c_{i-1}^2\left(3 + \frac{c_{i-1}}{4}\right)$$

La suite a_n converge vers $\sqrt{x}$ pour $0 < x < 2$. Ecrivez une fonction *racine* qui calcule la racine carrée d'un réel compris strictement entre 0 et 2, avec une précision *e* donnée.

16.9 MATRICES

1 On appelle matrice de Hilbert d'ordre n la matrice symétrique $H=(h_{ij})_{1\le i,j\le n}$ avec

$h_{ij} = \frac{1}{(i+j-1)}$, pour $1\le i,j \le n$.

En précisant la structure de données utilisée, écrivez une procédure *hilbert*(n,H) qui détermine la matrice de Hilbert H de rang n.

2 Ecrivez la procédure *produit_triang* qui calcule le produit de deux matrices carrées A et B où A est une matrice triangulaire supérieure ($a_{ij} = 0$ pour $j<i$).

16.10 MATRICES ORTHOGONALES

Une matrice carrée A(nxn), est orthogonale si $AA^T = Id$, avec A^T matrice transposée de A.
Ecrivez une fonction booléenne qui détermine si une matrice est orthogonale ou non (il sera tenu compte des performances de l'algorithme proposé).

16.11 METHODE DE LA PUISSANCE ITEREE POUR LE RECHERCHE DE VALEURS PROPRES

Soit A une matrice carrée (nxn) ayant n vecteurs propres linéairement indépendants u_1, u_2,..., u_n ($\|u_i\| = 1$) associés aux valeurs propres λ_1., λ_2.,..., λ_n., rangées dans l'ordre des modules croissants avec $|\lambda_1.| > |\lambda_2.| \ge ... \ge |\lambda_n.|$. La méthode de la puissance itérée permet d'approcher la valeur propre de plus grand module λ_1 et le vecteur propre associé :

$x^{(0)}$ vecteur de départ ,

$$x^{(k)} = \frac{A.x^{(k-1)}}{\|A.x^{(k-1)}\|}$$

on a $\lim_{k\to\infty} x^{(k)} = u_1$ et $\lim_{k\to\infty} \| A.x^{(k)} \| = |\lambda_1|$.

1 Ecrivez une fonction *norme* qui calcule la norme d'un vecteur $x \in IR^n$: $\|x\| = \max_{i=1}^{i=n} (|x_i|)$.

2 On suppose écrite la procédure *produit*(A,x,y) qui calcule y = Ax avec A matrice nxn, x vecteur de IR^n. Ecrivez la fonction booléenne *arrêt* qui détermine si la convergence de la suite est atteinte en utilisant le critère suivant : $\left| \frac{\|A\, x^{(m)}\| - \|A\, x^{(m-1)}\|}{\|A\, x^{(m-1)}\|} \right| < \varepsilon$, (ε est donné).

3 Etant donnés $x^{(0)}$, A, ε et Max, écrivez une procédure de calcul de la valeur propre de plus grand module et du vecteur propre associé de la matrice A. Il faudra arrêter les calculs si la convergence n'est pas atteinte en Max itérations.

16.12 DETERMINANTS

Cet exercice n'est intéressant que du point du vue algorithmique, pour calculer des déterminants, il faut s'y prendre autrement !

1 Donnez une structure de données pour représenter une matrice et sa dimension (dans ce problème on ne travaillera qu'avec des matrices carrées).

2 Pour une matrice carrée A de n lignes et n colonnes, on définit la sous-matrice $A_{i,j}$ ($1 \le i,j \le n$) comme la sous-matrice de A obtenue en retirant de A la ligne i et la colonne j.
Ecrivez une procédure *ss-matrice* construisant une sous-matrice obtenue à partir d'une matrice en lui retirant une ligne et une colonne données.

Exemple : $A = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix}$ $A_{1,2} = \begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{vmatrix}$

3 a) Ecrivez une fonction *det2* qui calcule le déterminant d'une matrice (2,2). On rappelle que

$$\Delta = \begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc$$

3 b) En utilisant la procédure *ss-matrice* et la fonction *det2*, écrivez une fonction *det3* qui calcule le déterminant d'une matrice (3,3) donnée, en développant suivant la première ligne :

$$\begin{vmatrix} a1 & b1 & c1 \\ a2 & b2 & c2 \\ a3 & b3 & c3 \end{vmatrix} = a1 \begin{vmatrix} b2 & c2 \\ b3 & c3 \end{vmatrix} - b1 \begin{vmatrix} a2 & c2 \\ a3 & c3 \end{vmatrix} + c1 \begin{vmatrix} a2 & b2 \\ a3 & c3 \end{vmatrix}$$

c) Généralisez en écrivant la fonction *detn* qui calcule le déterminant d'une matrice carrée donnée, de dimension quelconque. Vous utiliserez la procédure *ss-matrice* et la fonction *det2*, une solution récursive est recommandée.

On rappelle que le déterminant d'une matrice A (n,n) est (en développant suivant la ligne 1) :

$$\det(A) = \sum_{k=1}^{n} (-1)^{k+1} \det(A_{1,k})$$

où $A_{1,k}$ est la sous-matrice de A obtenue en retirant la première ligne et la $k^{ème}$ colonne.