Xcas au lycée

Renée De Graeve & Bernard Parisse
Université de Grenoble I

Xcas, au départ un logiciel de calcul formel, permet aujourd'hui de faire de l'algorithmique, de la géométrie interactive et analytique dans le plan et l'espace et propose un tableur formel, d'où son nom de "couteau suisse des mathématiques". Il s'agit d'un logiciel libre, disponible sous Windows, Mac OS et Linux, la version à jour se récupère en tapant `xcas` sur un moteur de recherche ou directement depuis le site de Xcas (adresse ci-dessous).

Ce fascicule contient des fiches de présentation rapide des différents modules de Xcas, accompagnées d'exemples ou/et de petits exercices corrigés, y compris sur les nouveaux programmes de lycée (proba-stats et spécialité Terminale S). Il est assez condensé pour tenir peu de place, de nombreux autres documents sont disponibles dans la documentation en ligne de Xcas, sur le site web de Xcas (page pédagogique en particulier), le forum de Xcas permet de poser des questions, et aussi de voir ce que des collègues ont pu réaliser en classe.

Site web, page pédagogique et forum de Xcas :
- `www-fourier.ujf-grenoble.fr/~parisse/giac_fr.html`
- `www-fourier.ujf-grenoble.fr/~parisse/irem.html`
- `http://xcas.e.ujf-grenoble.fr/XCAS`

Table des matières

Première partie

Fiche `Xcas` l'interface

Description de l'interface	
`Fich Edit Cfg Aide ...`	est une barre de menu cliquable
`session1.xws`	est le nom de la session (c'est `Unnamed` si la session n'a pas été sauvée)
`?`	ouvre l'index de l'aide
`Sauve`	sauvegarde la session
`Config : exact reel...`	ouvre la configuration du CAS
`STOP`	interrompt un calcul trop long
`Kbd`	fait apparaitre un clavier scientifique
`X`	ferme la session
`1`	est une ligne de commande

Chaque session est composée de niveaux numérotés qui peuvent être de différentes natures : ligne de commandes pour le calcul formel, écran de géométrie dynamique (2-d et 3-d), tableur formel, dessin tortue, éditeur de programmes etc...
`Alt+g` signifie `Alt` puis `g` en laissant `Alt` enfoncé.

Les différents niveaux possibles	
`Alt+c`	ouvre une ligne de commentaires
`Alt+d`	ouvre un niveau de dessin tortue
`Alt+e`	ouvre un éditeur d'expressions
`Alt+g`	ouvre un niveau de géométrie 2-d
`Alt+h`	ouvre un niveau de géométrie 3-d
`Alt+n`	ouvre une ligne d'entrée de commandes
`Alt+p`	ouvre un éditeur de programmes
`Alt+t`	ouvre un tableur

Les commandes sont classées par thème dans les menus, on peut aussi les retrouver par ordre alphabétique dans l'index de l'aide (`Aide`▶`Index`). Vous avez aussi plusieurs manuels disponibles avec des exercices corrigés (avec le menu `Aide`▶`Manuels`▶`...`) et des exemples (avec le menu `Aide`▶`Exemples`).

Les aides possibles	
`Aide`▶`Manuels`▶`...`	ouvre un des manuels dans votre navigateur
`Aide`▶`Exemples`▶`...`	ouvre la session de l'exemple choisi dans `Xcas`
`F12`	recherche d'un mot-clef dans les manuels
`Aide`▶`Index`	ouvre l'index de l'aide des commandes
`Cmds`▶`Reel`▶`Base`▶`ceil`	ouvre l'index de l'aide à `ceil`
`?`	ouvre l'index de l'aide des commandes
`ce` `?`	ouvre l'index de l'aide sur `ceil`
`ce` `F1`	ouvre l'index de l'aide sur `ceil`
`ce` ⇄	ouvre l'index de l'aide sur `ceil`
`?ceil`	ouvre l'aide détaillée sur `ceil`

`Xcas` manipule différents types de données :les entiers (`2`), les fractions (`3/2`), les nombres flottants (`2.0`,`1.5`), les paramètres formels (`x,t`), les variables (`a:=2`), les expressions (`x^2-1`), les fonctions (`f(x):=x^2-1`), les listes (`[1,2,3]`), les séquences (`1,2,3`) (une matrice est une liste de listes de même longueur, une séquence ne peut pas contenir de séquences), les chaines de caractères (`"na"`) et les objets géométriques.
`Xcas` peut faire du calcul formel et du calcul numérique.
Pour faire du calcul formel, on utilise les nombres exacts. Les nombres exacts sont les constantes prédéfinies comme `pi`,`e`,`i`,`infinity`, `+infinity`, `-infinity`, les entiers comme `2`, les fractions d'entiers comme `1/2` et toute expression ne contenant que des entiers et des constantes comme `sqrt(2)*e^(i*pi/3)`. Les calculs sont effectués en mode exact si tous les nombres qui interviennent sont exacts, (`3/2+1` renvoie `5/2`). La commande `evalf` renvoie la valeur approchée d'une valeur exacte (`evalf(1/2)` renvoie `0.5`).
Pour faire du calcul numérique, on utilise les nombres approchés. Les nombres approchés sont notés avec la notation scientifique standard : partie entière suivie du point de séparation et partie fractionnaire (éventuellement suivie de `e` et d'un exposant) : `0.5` ou `5e-1` est la version approchée du rationnel `1/2`.
Les calculs sont effectués en mode approché si un des nombres de l'expression est approché, (`1.5+1` renvoie `2.5`). Pour les nombres réels approchés, la précision par défaut est d'environ 15 chiffres significatifs. Elle peut être changée, en donnant le nombre de décimales désiré comme second argument de `evalf`, par exemple `evalf(sqrt(2),50)` ou en modifiant la variable `Digits`.par exemple, `Digits:=50; evalf(sqrt(2))`.

Il ne faut pas confondre expression et fonction. Une expression est une combinaison de nombres et de variables reliés par des opérations alors qu'une fonction associe à une variable une expression. Par exemple, `a:=x^2+2*x+1` définit une expression et `b(x):=x^2+2*x+1` définit une fonction. On obtient la valeur de l'expression `a` en 0, avec `subst(a,x=0)` et la valeur de la fonction `b` en 0, avec `b(0)`.

Signification des signes de ponctuation	
`.`	sépare la partie entière de la partie décimale
`,`	sépare les éléments d'une liste ou d'une séquence
`;`	termine chaque instruction d'un programme
`:;`	termine les instructions lorsqu'on ne veut pas l'affichage de la réponse
`!`	n ! est la factorielle de n (4 !=$1 \cdot 2 \cdot 3 \cdot 4 = 24$)
`:=`	`a:=2` instruction d'affectation qui stocke `2` dans la variable `a`
`[]`	délimiteurs d'une liste (`L:=[0,2,4]` et `L[1]` renvoie `2`)
`""`	délimiteurs d'une chaine de caractères (`C:="ba"` et `C[1]` renvoie `"a"`)

Les différentes configurations pour définir votre environnement	
►	désigne un sous-menu à choisir
`Cfg►Configuration du CAS`	ouvre la configuration du CAS
`Cfg►Configuration graphique`	ouvre la config. graphique par défaut
`Cfg►Configuration generale`	ouvre la configuration générale
`bouton cfg` (d'une sortie graphique)	ouvre la config. du niveau graphique
`bouton Config : exact...`	ouvre la configuration du CAS
`bouton Config tableur :`	ouvre la configuration du tableur

1 Exemples d'utilisation

1.1 Les différents niveaux

Dans une ligne d'**entrée de commande** (`Alt+n` ou `CAS▶Nouvelle entree`), on tape : `f(x):=ln(2x);simplify(f(2)+f(3))` et on obtient : `ln(24)`.
Certaines commandes ont un résultat graphique, par exemple
`plot(sin(x),x=0..pi)` trace le graphe de la fonction sinus sur $[0,\pi]$.

Dans un éditeur de **programmes** (`Alt+p` ou `Prg▶Nouveau programme`), on tape

```
f(a,b,c):={
 local d; d:=b^2-4*a*c;
 si d<0 alors retourne []; sinon
  d:=sqrt(d);
  retourne [(-b-d)/(2*a),(-b+d)/(2*a)];
 fsi;
} :;
```

on appuie sur `OK` (ou `F9`). Sur une ligne de commande, on tape `f(1,1,1)` et on obtient `[]` ou on tape `f(1,-5,6)` et on obtient `[2,3]`.

Dans un niveau de **géométrie 2d** (`Alt+g` ou `Geo▶Nouvellefigure 2d`), on choisit `Mode▶point` et on clique 3 points dans l'écran. On obtient `A,B,C` et leurs définitions dans les lignes de commandes à gauche de l'écran.
On tape à gauche de l'écran : `triangle(A,B,C);mediane(A,B,C;)`.
En `Mode▶Pointeur` on peut déplacer `A` avec la souris et modifier la figure.

Dans un **tableur** (`Alt+t` ou `Tableur▶Nouveau tableur`), on remplit la fenêtre de configuration qui s'ouvre automatiquement et on la valide avec `OK`.
On clique sur `A0` et on tape `1 Enter` et `1` s'inscrit dans la ligne de commande située sous la barre de menu du tableur. On clique sur `A1` et on tape `=A0+1 Enter`, on clique sur `B0` et on tape `=A0^2 Enter`.
Puis on recopie ces 2 formules vers le bas : on clique sur `A1` (resp `B0`) puis `Ctrl+d` (ou menu du tableur `Edit▶Remplir▶Copier vers le bas`).
On obtient : `dans A la liste des entiers et dans B la liste de leurs carrés`.

Dans un **dessin tortue** (`Alt+d` ou `Tortue▶Dessin tortue`), on a la tortue sous la forme d'une flèche dans un écran de dessin avec une ligne de commande à gauche, un éditeur à droite et en bas des boutons qui inscrivent les principales commandes là où se trouve le curseur (soit dans l'éditeur, soit dans une ligne de commande). **Attention** Tout ce que l'on valide à gauche s'inscrit à droite. On peut modifier l'éditeur pour corriger une erreur puis l'exécuter avec `OK`

1.2 Les signes de ponctuation

Si on tape `sqrt(2)`, on obtient `sqrt(2)`
Si on tape `sqrt(2.)`, on obtient `1.41421356237`. En effet, dans le deuxième cas le point après le 2 désigne un nombre approché.
On tape pour avoir la valeur approchèe avec 20 décimales :
`evalf(sqrt(2),20)`
On obtient :
`1.4142135623730950488`

On tape :
`a:=20!;b:=string(a);dim(b)`
On obtient factorielle 20, sa valeur convertie en chaine de caractères et la longueur de cette chaine c'est à dire le nombre de chiffres de 20 ! :
`2432902008176640000,"2432902008176640000",19`
On tape :
`a:=20!:;b:=string(a):;dim(b)`
On obtient, car `:;` indique qu'il ne faut pas afficher le résultat :
`"Done","Done",19`
On tape :
`v1:=[1,2,3];v2:=[-1,2,-3];v1*v2`
On obtient `v1`, `v2` et le produit scalaire de `v1` et `v2` :
`[1,2,3],[-1,2,-3],-6`
On tape :
`M1:=[[1,2],[3,4]];M2:=[[1,3],[2,4]];M1*M2`
On obtient `M1`, `M2` et le produit des matrices `M1` et `M2` :
`[[1,2],[3,4]],[[1,3],[2,4]],[[5,11],[11,25]]`

1.3 Les aides

Si on connait le nom de la commande, on clique : `Aide▶Index`
On obtient : `la liste des commandes par ordre alphabétique, un curseur permet de parcourir cette liste et une ligne d'entée permet d'ouvrir l'index à la demande`
Ou bien, on tape :
`le début d'un nom puis sur le bouton` $\boxed{?}$ `(sur fond cyan) ou sur la touche de tabulation`.
On obtient :
`l'ouverture de l'index à l'endroit indiqué`
Ou bien on utilise les menus qui classent les commandes par thème, par exemple, on clique : `Cmds▶Entier▶iegcd`
On obtient, si dans `Cfg▶Configuration generale` on n'a pas coché `Aide automatique` :
`l'aide sur cette commande dans la fenêtre des messages` ($\boxed{\texttt{kbd}}$▶$\boxed{\texttt{msg}}$) et si on a coché `Aide automatique`, on obtient en plus : `l'ouverture de l'index à iegcd`.

1.4 Les configurations

En plus des différentes configurations, il faut :
- dans un niveau de géométrie ou un graphe 2d (resp 3d) penser à utiliser le menu `M` situé dans le pavé des boutons à droite de l'écran. Il permet de gérer des animations ou des traces (resp de gérer les vues d'une figure 3d).
  de cocher ou décocher la case $\boxed{\sim}$ selon que l'on veut des points à coordonnées flottantes ou exactes.
- dans un niveau de dessin tortue, penser à utiliser le menu `M` situé sous l'écran de dessin, à droite : vous pouvez ainsi faire apparaitre (ou non) un maillage et cacher le carré jaune en haut à droite qui donne la position de la tortue.

Deuxième partie

Fiche `Xcas` calcul formel de base

L'exécution d'une ligne de commandes se fait par la touche "Entrée". Les nombres peuvent être exacts ou approchés (flottants). Les nombres exacts sont les constantes prédéfinies, les entiers, les fractions d'entiers et toutes expressions ne contenant que des entiers et des constantes. Les nombres approchés sont notés avec la notation scientifique standard : partie entière suivie du point de séparation et partie fractionnaire optionnellement suivie de `e` et d'un exposant.

Opérations	
`+`	addition
`-`	soustraction
`*`	mutiplication
`/`	division
`^`	puissance

Constantes prédéfinies	
`pi`	$\pi \simeq 3.14159265359$
`e`	$e \simeq 2.71828182846$
`i`	$i^2 = -1$ et `arg(i)`=$\frac{\pi}{2}$
`infinity`	∞
`+infinity` ou `inf`	$+\infty$
`-infinity` ou `-inf`	$-\infty$
`euler_gamma`	constante d'Euler

Fonctions classiques			
`evalf(t,n)`	évalue t avec n décimales	`sign`	signe (vaut -1, 0 ou +1)
`max`	maximum	`min`	minimum
`round`	arrondi	`frac`	partie fractionnaire
`floor`	plus grand entier $\leq$	`ceil`	plus petit entier $\geq$
`re`	partie réelle	`im`	partie imaginaire
`abs`	module ou valeur absolue	`arg`	argument
`conj`	conjugué	`affixe`	affixe
`factorial`	factorielle	`binomial`	coefficients binomiaux
`exp`	exponentielle	`sqrt`	racine carrée
`ln log`	logarithme naturel	`log10`	logarithme en base 10
`sin`	sinus	`cos`	cosinus
`tan`	tangente	`cot`	cotangente
`asin`	arc sinus	`acos`	arc cosinus
`atan`	arc tangente	`acot`	arc cotangente
`sinh`	sinus hyperbolique	`cosh`	cosinus hyperbolique
`asinh`	arc sinus hyperbolique	`acosh`	arc cosinus hyperbolique
`tanh`	tangente hyperbolique	`atanh`	arc tangente hyperbolique

Fractions	
`propfrac`	partie entière + partie fractionnaire
`getNum numer`	numérateur de la fraction simplifiée
`getDenom denom`	dénominateur de la fraction simplifiée
`f2nd`	`[numer,denom]` de la fraction simplifiée
`simp2`	simplification d'un couple
`mult_conjugue(n/d)`	multiplie par la quantité conjuguée de d ou de n si $d = 1$
`dfc`	développe un réel en fraction continue
`dfc2f`	transforme une fraction continue en réel

Chaînes de caractères, séquences et listes ou vecteurs	
`S:="abc"`	S est une chaîne de 3 caractères
`S:=a,b,c`	S est une séquence de 3 élements
`S:=[a,b,c]`	S est une liste de 3 élements
`op([a,b,c])`	renvoie la séquence `(a,b,c)`
`S:=""`	S est une chaîne de 0 caractère
`S:=NULL`	S est une séquence de 0 élement
`S:=[]`	S est une liste de 0 élement
`dim(S)`	renvoie le nombre d'éléments (ou caractères) de S
`S[0]`	renvoie le premier élément (ou caractère) de S
`S[n]`	renvoie le $n+1$ unième élément (ou caractère) de S
`S[dim(S)-1]`	renvoie le dernier élément (ou caractère) de S
`S:=S+"d"`	ajoute le caractère d à la fin de la chaîne S
`"ab"+"def"`	concaténe les 2 chaines et renvoie "abdef"
`S:=S,d`	ajoute l'élément `d` à la fin de la séquence S
`S:=append(S,d)`	ajoute l'élément `d` à la fin de la liste S

Fonctions d'arithmétique	
`a%p`	a modulo p est un élément de $\mathbb{Z}/p\mathbb{Z}$
`smod(a,b)` ou `(a%b)%0`	reste symétrique de la division euclidienne
`powmod(a,n,p)`	est l'entier égal à a^n modulo p
`irem`	reste de la division euclidienne
`iquo`	quotient de la division euclidienne
`iquorem`	quotient et reste de la division euclidienne
`ifactor`	décomposition en facteurs premiers
`ifactors`	liste des facteurs premiers
`idivis`	liste des diviseurs
`gcd`	plus grand diviseur commun
`lcm`	plus petit multiple commun
`iegcd`	identité de Bezout
`iabcuv`	renvoie $[u,v]$ tels que $au+bv=c$
`ichinrem`	restes chinois
`is_prime`	teste si l'entier est premier
`nextprime`	prochain entier pseudo-premier
`previousprime`	entier pseudo-premier précédent

Réécriture d'expressions			
`tsimplify`	simplifie (- puissant)	`simplify`	simplifie
`ratnormal`	forme normale (- puissant)	`normal`	forme normale
`partfrac`	décompose en éléments simples	`expand`	développe
`convert`	transforme en le format spécifié	`factor`	factorise
`pow2exp`	puissances vers exp	`expexpand`	développe les exp
`exp2pow`	convertit $\exp(n*\ln(x))$ en x^n	`lin`	linéarise les exp
`lncollect`	rassemble les log	`lnexpand`	développe les log
`trigsin`	utilise $\cos(x)^2=1-\sin(x)^2$	`halftan`	trig en $\tan(x/2)$
`tcollect`	linéarise et regroupe	`tlin`	linéarise
`texpand`	développe $\exp, \ln$ et trig	`trig2exp`	trig vers $\exp$
`hyp2exp`	hyperbolique vers $\exp$	`exp2trig`	$\exp$ vers trig

2 Exemples d'utilisation

2.1 Transformations

Développer
On tape : `expand((1+2x)^2)`
on obtient : `4*x^2+4*x+1`
On tape : `expand((x+1+i)*(x+i))`
on obtient : `x^2+(1+2*i)*x-1+i`
Factoriser
Des entiers : `ifactor(10^10+1)`
renvoie : `101*3541*27961`,
Des polynômes : `factor(x^5+9*x^4+30*x^3+46*x^2+33*x+9)`
renvoie : `(x+1)^3*(x+3)^2`
Simplifier
Des fractions : `normal(1/(2-x)+2/(5-2x))`
renvoie : `(-4*x+9)/(2*x^2-9*x+10)`,
et `simplify(1/(2-x)+1/(2+x))` renvoie `(-4)/(x^2-4)`
Des racines carrées :
`simplify(2*sqrt(45)+3*sqrt(12)-sqrt(20)-sqrt(3))`
renvoie : `5*sqrt(3)+4*sqrt(5)`
Trigonométrie
Linéariser : `tlin(2*cos(x)*cos(y))`
renvoie : `cos(x-y)+cos(x+y)`,
Développer : `texpand(sin(x+y))`
renvoie : `sin(x)*cos(y)+cos(x)*sin(y)`
Convertir en exponentielles : `normal(trig2exp(3*(cos(x)-i*sin(x))))`
renvoie : `3/exp((i)*x)`
et : `simplify(trig2exp(-sin(2x)+2i*cos(x)^2))`
renvoie : `(i)*exp((i)*x)^2+i`
Conversion inverse : `exp2trig(exp(i*x))`
renvoie : `cos(x)+(i)*sin(x)`
Regrouper sin et cos : `tcollect(sqrt(3)*sin(x)+cos(x))`
renvoie : `2*cos(x-1/3*pi)`
Exponentielle et Logarithme
Linéariser : `lin(1/exp(i*x))`
renvoie : `exp(-(i)*x)`
Regrouper les logs : `lncollect(ln(2)+ln(3))`
renvoie : `ln(6)`
Développer les logs : `lnexpand(ln((x+2)^2))`
renvoie : `2*ln(x+2)`

2.2 Arithmétique

Un exercice sur des nombres de $\mathbb{Z}/p\mathbb{Z}$
Trouver les 2 derniers chiffres de 19969^{19969}.
On tape : `powmod(19969,19969,100)` et on obtient : `29`
Un exercice sur le nombre de diviseurs d'un entier
Parmi les entiers naturels de 1 à 2012, quel est celui qui admet le plus de diviseurs ?

Quel est ce nombre de diviseurs ?
On tape : `2*3*5*7` et on obtient : `210`
On tape : `2*3*5*7*11` et on obtient : `2310`
Donc le nombre n cherché est de la forme :
$n = 2^a * 3^b * 5^c * 7^d$ avec $a \geq b \geq c \geq d \geq 0$
et son nombre de diviseurs est : $(a+1)(b+1)(c+1)(d+1)$
On fait une recherche systématique en tapant `size(idivis(n))` et on trouve :
$2^{10} = 1024$ a 11 diviseurs, $2^9 * 3 = 1536$ a 20 diviseurs,
$2^7 * 3^2 = 1116$ a 24 diviseurs, $2^6 * 3^3 = 1728$ a 28 diviseurs,
$2^4 * 3^4 = 1296$ a 25 diviseurs, $2^7 * 3 * 5 = 1920$ a 32 diviseurs,
$2^5 * 3^2 * 5 = 1440$ a 36 diviseurs, $2^4 * 3 * 5 * 7 = 1680$ a 40 diviseurs.
Donc $n = 2^4 * 3 * 5 * 7 = 1680$ et il a 40 diviseurs.

Un exercice sur l'identité de Bézout

Quel est le plus petit nombre entier avec lequel il faut multiplier 49 pour obtenir un nombre se terminant par 999999999 (9 neufs) ?
On a : $999999999 + 1 = 10^9$.
Donc un nombre se terminant par 999999999 (9 neufs) s'écrit : $k * 10^9 + 10^9 - 1$ c'est à dire $10^9(k+1) - 1 = 10^9 a - 1$ avec $a = k + 1$.
On cherche p pour avoir : $p * 49 = a * 10^9 - 1$ c'est à dire $1 = a * 10^9 - p * 49$.

Réponse niveau primaire

On peut faire une multiplication à trous : $49 * .... = ..999999999$
On trouve : $49 * 693877551 = 33999999999$

Réponse niveau TS

Avec `Xcas` on tape : `bezout_entiers(49,10^9)`
On obtient : `[306122449,-15,1]`
Donc : $49 * 306122449 - 15 * 10^9 = 1$ et puisque $49 * 10^9 - 49 * 10^9 = 0$, on a :
$49 * (10^9 - 306122449) + (15 - 49) * 10^9 = -1$.
Puisque $10^9 - 306122449 = 693877551$ et $(49 - 15) = 34$, on obtient la solution :
$49 * 693877551 = 34 * 10^9 - 1 = 33999999999$

Un exercice sur les congruences et les restes chinois

Trouver les nombres entiers $n, 4 < n < 10000$ vérifiant :
n est divisible par 2, $n - 1$ est divisible par 3, $n - 2$ est divisible par 5, $n - 3$ est divisible par 7, $n - 4$ est divisible par 11. Donc $n = 2p = 3q + 1 = 5m + 2$...
On tape : `ichinrem([0%2,1%3,2%5,3%7,4%11]))`
On obtient : `-788 % 2310`
On veut les entiers n soient tels que $4 < n < 10000$ donc :
$n = (2310 - 788) + 2310k = 1522 + 2310k$ pour $k \leq$ iquo$((10000 - 1522), 2310)$.
On tape : `iquo((10000-1522),2310)` et on obtient :`3`
On tape : `(1522+2310*k)$(k=0..3)`
On obtient les 4 nombres solutions : `1522,3832,6142,8452`

Décomposition en facteurs premiers

Exposant de 17 dans la décomposition en facteurs premiers de 500 !
On tape : `iquo(500,17),iquo(500,17^2),iquo(500,17^3)`
On obtient : `29,1,0`
L'exposant de 17 dans la décomposition en facteurs premiers de 500 ! est 29+1=30.
Ou, on utilise `ifactors(500!)` qui renvoie la liste `L` des facteurs premiers de `500!` avec leur multiplicié. 17 etant le 7-ième nombre premier, l'exposant de 17 sera le 13-ième élément de `L`. On tape `ifactors(500!)[13]` et on obtient `30`.

Troisième partie

Fiche Xcas les proba-stats et le tableur

Probabilités	
`comb(n,k)`	nombre de combinaisons de p objets pris parmi n
`perm(n,p)`	nombre d'arrangements de p objets pris parmi n
`factorial(n)` ou `n!`	$n!$
`randseed` ou `srand`	initialise la suite des nombres aléatoires.
`rand(n)` ou `alea(n)`	entier aléatoire uniformément distribué dans 0.. $n-1$
`rand(p,q)` ou `alea(p,q)`	réel aléatoire uniformément distribué dans $[p, q]$
`rand(p,L)` ou `alea(p,L)`	liste de p éléments pris au hasard dans la liste L
`randnorm(mu,sigma)`	réel aléatoirement distribué selon la loi normale $N(\mu, \sigma)$
`randexp(a)`	renvoie un réel aléatoire selon la loi exponentielle

Statistiques 1-d	
`moyenne`	moyenne d'une liste pondérée par le 2nd argument
`median`	médiane d'une liste pondérée par le 2nd argument
`quartiles`	[min,quartile1, médiane,quartile3,max]
`moustache`	boite à moustache d'une série statistique
`variance`	variance d'une liste pondérée par le 2nd argument
`ecart_type`	écart-type d'une liste pondérée par le 2nd argument
`histogramme`	trace l'histogramme de l'argument
Distributions	
`binomial(n,k,p)`	probabilité d'obtenir k succès avec n tirages (où p est la probabilité de succès par tirage)
`binomial_cdf(n,p,n1,n2)`	probabilité d'obtenir entre $n1$ et $n2$ succès
`binomial_cdf(n,p,n1)`	probabilité d'obtenir au plus $n1$ succès
`binomial_icdf(n,p,t)`	renvoie le plus grand entier k tel que la probabilité d'obtenir au plus k succès soit $\leq t$
`normald(x)`	densité de la loi normale moyenne 0 et écart-type 1
`normald(mu,sigma,x)`	densité de la loi normale moyenne μ et écart-type σ
`normald_cdf(mu,sigma,a)`	probabilité selon la loi normale μ, σ que x soit $\leq a$
`normald_cdf(mu,sigma,a,b)`	probabilité selon la loi normale de moyenne μ et d'écart-type σ que x soit entre a et b
`normald_icdf(mu,sigma,t)`	renvoie x tel que la probabilité d'être $\leq x$ soit $\leq t$ (selon la loi normale moyenne μ et écart-type σ)

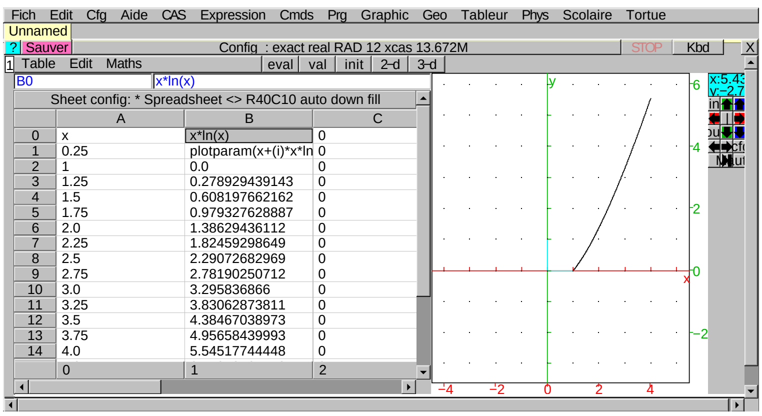
Les commandes de statistiques s'utilisent :

- dans le tableur avec le menu `Maths` : le tableur se remplit automatiquement grâce à une boite de dialogue qui demande de préciser les paramètres de la commande choisie (le curseur doit être dans un niveau de type tableur qui s'obtient avec `Alt+t`). C'est un tableur formel dans lequel on peut utiliser toutes les commandes de `Xcas` ainsi que les variables définies auparavant.
- dans des lignes de commandes : soit on les tape, soit on les sélectionne dans le menu `Cmds`▶`Proba_stats`, soit on utilise le menu `Graphic`▶`Stats` et ses boites de dialogues.

Statistiques 2-d	
`polygonplot`	ligne polygonale
`scatterplot`	nuage de points
`polygonscatterplot`	ligne polygonale pointée
`covariance`	covariance des éléments de l'argument
`correlation`	corrélation des éléments de l'argument
`exponential_regression`	(m, b) où $y = be^{mx}$ approche l'argument
`exponential_regression_plot`	graphe de $y = be^{mx}$ approchant l'arg.
`linear_regression`	(a, b) où $y = ax + b$ approche l'argument
`linear_regression_plot`	graphe de $y = ax + b$ approchant l'argument
`logarithmic_regression`	(m, b) où $y = m\ln(x) + b$ approche l'arg.
`logarithmic_regression_plot`	graphe de $y = m\ln(x) + b$, approchant l'arg.
`polynomial_regression`	$(a_n, ..a_0)$ où $y = a_n x^n + ..a_0$ approche l'arg.
`polynomial_regression_plot`	graphe de $y = a_n x^n + ..a_0$ approchant l'arg.
`power_regression`	(m, b) où $y = bx^m$ approche l'argument
`power_regression_plot`	graphe de $y = bx^m$ approchant l'argument

Le tableur fait du calcul formel et a :
- ses menus `Table Edit Maths` (un click droit permet d'obtenir ses menus),
- ses boutons `eval val init 2-d 3-d`,
- sa case de sélection qui est une case interactive : soit on sélectionne à la souris une cellule ou une plage et son nom s'y inscrit (par exemple `A0:B3`), soit on y tape le nom des cellules à sélectionner, (par exemple `A0..3,C`),
- sa ligne de commande qui permet de remplir les cellules,
- un bouton pour configurer le tableur et qui rappelle cette configuration.

Au tableur est associé un écran graphique 2D visible si on a coché `Graphe` dans la configuration du tableur (visible en dessous du tableur ou à droite selon que `Paysage` est coché ou non). On peut aussi utiliser le bouton `2-d` (ou `3-d`).
Le tableur est une matrice dont on numérote les lignes par des nombres (0,1...) et les colonnes par une ou plusieurs lettres (A,B...). Le nom d'une cellule se fait par référence relative (`C2`) ou absolue (`$C$2`) ou mixte (`$C2` ou `C$2`). Dans une cellule, on met soit une liste ou une formule de calcul commencant par `=`, soit une constante, soit une chaine de caractères entourées de guillemets (`"essai"`).
Exemple : tableau de valeurs d'une fonction Dans le tableur, on ouvre une boite de dialogue avec le menu `Maths▶Function`. On met : expression `x*ln(x)`, variable `x` de 1 à 4, xstep= 0.25, colonne `A`. On obtient, si `Paysage` est décoché :

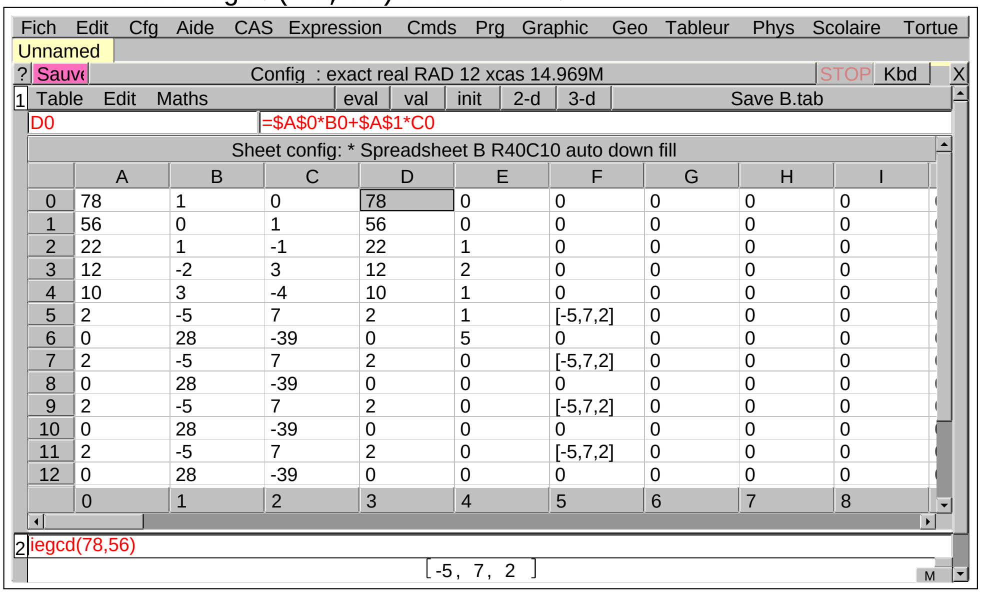


3 Exemples d'utilisation

3.1 Bézout programmé avec le tableur

- Dans une ligne de commande, on met `a:=78;b:=56` on valide puis on ouvre un niveau tableur.
- La colonne `A` est "la suites des restes" r_n. On met `=a` dans `A0`, `=b` dans `A1`, `=irem(A0,A1)` dans `A2` et on remplit vers le bas (`Ctrl+d`),
- La colonne `E` est "la suites des quotients" q_n. On met `=iquo(A0,A1)` dans `E2` et on remplit vers le bas,
- Les colonnes `B` et `C`, sont les deux suites u_n et v_n de façon qu'à chaque étape on ait : $r_n = u_n a + v_n b$. On met `1` dans `B0`, `0` dans `C0`, `0` dans `B1`, `1` dans `C1`, `B0-E2*B1` dans `B2`, `C0-E2*C1` dans `C2` et on remplit vers le bas,
- La colonne `D` est $u_n a + v_n b$ (`D =A`). On met `=B0*$A$0+C0*$A$1` dans `D0` et on remplit vers le bas
- La colonne `F` est la réponse $[u, v,$pgcd$(a, b)]$. On met `=si A1==0 alors [B0,C0,D0]; sinon 0;fsi` dans `F0` et on remplit vers le bas.

On vérifie avec `iegcd(78,56)` et on obtient :



	A	B	C	D	E	F	G	H	I
0	78	1	0	78	0	0	0	0	0
1	56	0	1	56	0	0	0	0	0
2	22	1	-1	22	1	0	0	0	0
3	12	-2	3	12	2	0	0	0	0
4	10	3	-4	10	1	0	0	0	0
5	2	-5	7	2	1	[-5,7,2]	0	0	0
6	0	28	-39	0	5	0	0	0	0
7	2	-5	7	2	0	[-5,7,2]	0	0	0
8	0	28	-39	0	0	0	0	0	0
9	2	-5	7	2	0	[-5,7,2]	0	0	0
10	0	28	-39	0	0	0	0	0	0
11	2	-5	7	2	0	[-5,7,2]	0	0	0
12	0	28	-39	0	0	0	0	0	0
	0	1	2	3	4	5	6	7	8

3.2 Loi binomiale

On tire une pièce à pile ou face 100 fois de suite. La probabilité d'obtenir 40 piles si la pièce est bien équilibrée est `binomial(100,40,0.5)` (valeur approchée) ou `binomial(100,40,1/2)` (valeur exacte).
La probabilité que le nombre de piles soit inférieure ou égal à 40 vaut :
`binomial_cdf(100,0.5,40)` (attention l'ordre des paramètres est inversé par rapport à `binomial`).
La probabilité d'être dans l'intervalle de confiance $[m - 2\sigma, m + 2\sigma]$ (ici $m = 50$ et σ=5) vaut `binomial_cdf(100,0.5,40,60)` (ici environ 0.965).
Pour avoir un intervalle de confiance à 99% centré, on calcule
`binomial_icdf(100,0.5,0.995)` qui renvoie 63 et
`binomial_icdf(100,0.5,0.005)` qui renvoie 37 (symétrique de 63).
Si la pièce n'est pas équilibrée, on remplace 0.5 par la probabilité d'obtenir pile.

3.3 Simulation

On peut simuler un pile ou face par `alea(2)`, une série de 100 lancers par `seq(alea(2),j,1,100)`, sa somme par `sum(alea(2),j,1,100)` et l'histogramme de 1000 séries de 100 lancers par
`histogram(seq(sum(alea(2),j,1,100),k,1,1000));`
On superpose la loi binomiale au graphe en rajoutant à la commande précédente
`plotlist(seq(binomial(100,k,0.5),k,0,100),couleur=rouge)`
On simule 100 lancers d'une pièce non équilibrée où la probabilité d'obtenir pile est p, par `seq(alea(0,1)<p,j,1,100)`, (le résultat du test `alea(0,1)<p` est de probabilité p car il vaut 1 si le nombre généré est plus petit que p et 0 sinon).
On simule le tirage de 2 entiers dans $I = [4; 12]$ par `alea(2,4,12)`, le tirage de 2 nombres dans `L:=[1,1,3,3,3,5,6]` par `alea(2,L)` et on simule 100 tirages selon la loi normale, par `seq(randnorm(0,1),j,1,100)`.

3.4 Convergence vers la loi normale

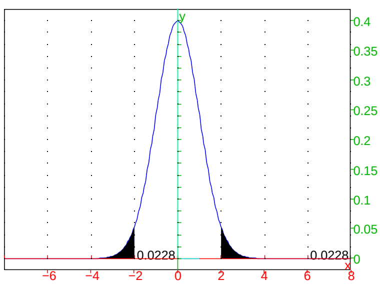
On peut tracer le diagramme en batons de la loi binomiale de paramètres par exemple $n = 100$ et $p = 0.4$ en tapant : `n:=100; p:=0.4;`
`diagramme_batons(seq(binomial(n,k,p),k,0,n));`
Cliquer plusieurs fois sur le bouton bleu flèche vers le haut pour voir tout le diagramme. Pour superposer la loi normale qui approche ce diagramme, ajouter la commande `graphe(normald(n*p,sqrt(n*p*(1-p)),x),x=0..100)`
Pour faire observer le théorème de Moivre-Laplace tel qu'il est énoncé, les commandes sont plus complexes. Il faut générer un histogramme avec en abscisse des intervalles de valeurs successives (ici centrés) que prend $(X_n - np)/\sqrt{np(1-p)}$ et en ordonnée la loi binomiale, on tapera les commandes

```
S:=seq([(k-n*p)/sqrt(n*p*(1-p)),binomial(n,k,p)],k,0,n);
H:=histogram(S,affichage=nom_cache);
graphe(normald(x),x=-10..10,couleur=rouge)
```

On calcule l'aire d'une partie de l'histogramme avec : `aire(H[a..b-1])` (rectangles d'indices a à b) ou `binomial_cdf(n,p,a,b-1)`, que l'on peut comparer avec l'aire sous la courbe `int(normald(x),x=a..b)`. On reprend les commandes précédentes en faisant varier n et p de façon interactive en définissant les paramètres n et p (menu `Edit->New parameter`, il faut décocher la case `symb` pour avoir un curseur numérique et définir la plage de n, p et le pas).

3.5 Fluctuations à un seuil donné

Visualisation de la partie hors de l'intervalle de confiance.
Commencer par tracer la loi normale :
`graphe(normald(x),x=-10..10);`
Valider. Ajouter à la fin de la commande :
`plotarea(normald(x),x=-6..-2);`
`plotarea(normald(x),x=2..6);`
Idem en remplaçant 2 et -2 par 3 et -3



Intervalle de confiance centré au seuil de 1% pour la loi normale centrée réduite : `[normal_icdf(0,1,0.005),normal_icdf(0,1,0.995)]`

Quatrième partie

Fiche `Xcas` Algèbre

Polynômes	
`normal`	forme normale (développée et réduite)
`expand`	forme développée
`ptayl(P,a)`	polynôme de Taylor Q tel que P(x)=Q(x-a)
`peval` ou `horner`	évaluation en un point par l'algorithme de Horner
`genpoly`	polynôme défini par sa valeur en un point
`canonical_form`	trinôme mis sous forme canonique
`coeff`	coefficient ou liste des coefficients
`poly2symb`	du polynôme au format Xcas à la forme symbolique
`symb2poly`	de la forme symbolique à un polynôme au format Xcas
`pcoeff`	polynôme décrit par ses racines
`degree`	degré
`lcoeff`	coefficient du terme de plus haut degré
`valuation`	degré du monôme de plus bas degré
`tcoeff`	coefficient du monôme de plus bas degré
`factor`	décomposition en facteurs irréductibles sur $\mathbb{Q}$
`cfactor`	décomposition en facteurs irréductibles sur $\mathbb{Q}[i]$
`factors`	liste des facteurs irréductibles
`divis`	liste des diviseurs
`collect`	factorisation sur le corps des coefficients
`froot`	racines avec leurs multiplicités
`proot`	valeurs approchées des racines
`sturmab`	nombre de racines dans un intervalle
`getNum`	numérateur d'une fraction rationnelle non simplifiée
`numer`	numérateur d'une fraction rationnelle simplifiée
`getDenom`	dénominateur d'une fraction rationnelle non simplifiée
`denom`	dénominateur d'une fraction rationnelle simplifiée
`propfrac`	isole partie entière et fraction propre
`partfrac`	décomposition en éléments simples
`quo`	quotient de la division euclidienne
`rem`	reste de la division euclidienne
`gcd`	plus grand diviseur commun
`lcm`	plus petit multiple commun
`egcd`	identité de Bezout
`chinrem`	restes chinois
`randpoly`	polynôme aléatoire
`cyclotomic`	polynômes cyclotomiques
`lagrange`	polynômes de Lagrange
`hermite`	polynômes de Hermite
`laguerre`	polynômes de Laguerre
`tchebyshev1`	polynômes de Tchebyshev 1ère espèce
`tchebyshev2`	polynômes de Tchebyshev 2nde espèce

Matrices	
`M:=[[a,b,c],[f,g,h]]`	`M` est une matrice de 2 lignes et 3 colonnes
`dim(M)`	est la liste [nbre_lignes, nbre_colonnes]
`nrows(M)`, `ncols(M)`	nombre de lignes, colonnes
`M[0]`	renvoie la première ligne de `M`
`M[n]` ou `row(M,n)`	renvoie la $n+1$ unième ligne de `M`
`col(M,n)`	renvoie la $n+1$ unième colonne de `M`
`M[nrows(M)-1]`	renvoie la dernière ligne de `M`
`M[n..p]`	renvoie la sous matrice de `M` de lignes [n..p]
`M:=append(M,[d,k,l])`	ajoute la ligne `[d,k,l]` à la fin de `M`
`M[nrows(M)]:=[d,k,l]`	ajoute la ligne `[d,k,l]` à la fin de `M`
`M:=border(M,[d,k])`	ajoute la colonne `[d,k]` à la fin de `M`
`M[j,k]:=a`	modifie l'élément d'indice `j,k` en copiant `M`
`M[j,k]=<a`	modifie en place l'élément d'indice `j,k`

Opérations sur les vecteurs et matrices	
`v*w`	produit scalaire
`cross(v,w)`	produit vectoriel
`A*B`	produit matriciel
`A.*B`	produit terme à terme
`A^n`	puissance entière d'une matrice
`inv(A)` ou `A^-1`	inverse de la matrice `A`
`matpow(A,n)`	puissance symbolique d'une matrice
`tran(A)`	transposée de la matrice `A`
`trn(A)`	adjointe de la matrice `A`
`rank(A)`	rang de la matrice `A`
`trace(A)`	trace de la matrice `A`
`det(A)`	déterminant de la matrice `A`
`ker(A)`	base du noyau de la matrice `A`
`image(A)`	base de l'image de la matrice `A`
`idn(n)`	matrice identité de dimension `n`
`ranm(p,q)`	matrice `p*q` à coefficients aléatoires

Systèmes linéaires	
`linsolve`	résolution d'un système
`syst2mat`	transforme $Y = AX + b$ en la matrice A bordée par $-b$
`rref`	réduction de Gauss-Jordan
`ref`	réduction de Gauss
`rank`	rang du système
`det`	déterminant du système

Réduction des matrices	
`jordan`	diagonalisation ou réduction de Jordan
`pcar`	coefficients du polynôme caractéristique
`pmin`	coefficients du polynôme minimal
`companion`	matrice compagnon d'un polynôme unitaire
`eigenvals`	valeurs propres
`eigenvects`	vecteurs propres

4 Exemples d'utilisation

4.1 Système d'équations linéaires

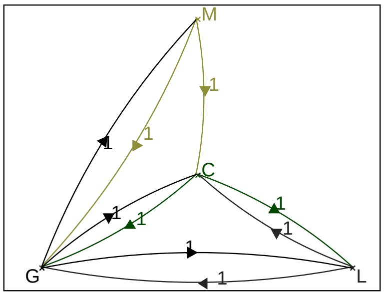
Résoudre le système linéaire en x, y, z de paramètre a :

$$\left\{\begin{array}{lcl} ax+y+z & = & 1 \\ x+ay-z & = & 2 \\ 2x+0y-z & = & 3 \end{array}\right.$$

On tape : `linsolve([a*x+y+z=1,x+a*y-z=2,2*x-z=3],[x,y,z])`
On obtient : `[(4*a+1)/(a^2+2*a+1),`
`(-a+2)/(a^2+2*a+1),(-3*a^2+2*a-1)/(a^2+2*a+1)]`
Pour déterminer les valeurs de a qui donnent 0 ou une infinité de solutions :
`solve(a^2+2*a+1,a)`, on obtient : `[-1]`
Si $a=-1$, on tape : `linsolve([-x+y+z=1,x-y-z=2,2*x-z=3],[x,y,z])`
On n'obtient pas de solutions : `[]`

4.2 Matrice et graphe

Dans une ville les lignes de bus vont : de la Gare au Centre, du Centre à la Gare, de la Gare au Lycée, du Lycée à la Gare, de la Gare à la Mairie, de la Mairie à la Gare, du Lycée au Centre, du Centre au Lycée, et de la Mairie au Centre. Donner la matrice A associée au graphe GLMC. Ci-contre :
`graphe_probabiliste(A,[G,L,M,C])`



Un voyageur a pris un billet de 5 trajets. Combien a-t-il de possibilités pour faire ces 5 trajets : si il part de la Gare et retourne à la Gare ou si il part du Lycée et doit arriver à la Gare.
On tape (A[j,k]=1 si il y a un trajet de j vers k) :
`A:=[[0,1,1,1],[1,0,0,1],[1,0,0,1],[1,1,0,0]]; B:=A^5`
On obtient pour `B` :
`[[21,23,12,22],[19,14,7,19],[19,14,7,19],[19,15,7,18]]`
Lorsqu'il part de la Gare et retourne à la Gare, il y a `B[0,0]=21` possibilités et lorsqu'il part du Lycée et doit arriver à la Gare il y a `B[1,0]=14` possibilités.

4.3 Interpolation et formule des 3 niveaux

Polynôme d'interpolation (de Lagrange) en 3 points :
On considère 3 couples de réels ou complexes $(x_j, y_j)_{j=0..2}$, les x_j étant 2 à 2 distincts. On montre qu'il existe un polynôme P unique de degré inférieur ou égal à 2 tel que pour $j=0..2$ on ait $P(x_j)=y_j$. Trouver P lorsque $x_1=(x_0+x_2)/2$.
Solution : On pose $P(x)=Ax^2+Bx+C$, il faut résoudre :

$$\left\{\begin{array}{lclcl} P(x_0) & = & Ax_0^2+Bx_0+C & = & y_0 \\ P(x_1) & = & Ax_1^2+Bx_1+C & = & y_1 \\ P(x_2) & = & Ax_2^2+Bx_2+C & = & y_2 \end{array}\right.$$

On tape :
`N:=[[x0^2,x0,1],[x1^2,x1,1],[x2^2,x2,1]]; x1:=(x0+x2)/2;`
`factor(linsolve([N*[A,B,C]-[y0,y1,y2]],[A,B,C]))`
Ou on tape `lagrange([x0,x1,x2],[y0,y1,y2])` qui renvoie le polynôme construit degré par degré dans la base $1, x - x0, (x - x0)(x - x1)$ (méthode des différences divisées) alors que les algébristes écrivent ce polynôme sous la forme :
$y_0\frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} + y_1\frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)} + y_2\frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}$

Calcul d'une intégrale par la méthode de Simpson :
On approche une fonction continue f sur l'intervalle $I = [x_0, x_2]$ par une parabole qui coïncide avec la fonction en $x_0, x_1 = (x_0 + x_2)/2, x_2$.
On cherche comme précédemment $P(x)$ un polynôme de degré ≤ 2 tel que $P(x_i) = f(x_i) = y_i$ pour $j = 0, 1, 2$. Calculons la valeur approchée de l'intégrale de f sur I obtenue en remplaçant f par le polynôme P.
On tape : `x1:=(x0+x2)/2;`
`factor(int(lagrange([x0,x1,x2],[y0,y1,y2]),x=x0..x2))`
On obtient : `((-y2-y0-4*y1)*(x0-x2))/6`
Donc
$$\int_{x_0}^{x_2} f(x)dx \simeq \frac{(x_2-x_0)}{6}(f(x_2) + 4f(\frac{x_2+x_0}{2}) + f(x_0))$$
Si f est un polynôme de degré 0, 1 ou 2, alors bien sûr $P = f$ et la formule obtenue n'est pas seulement approchée mais est exacte. Il se trouve que c'est encore vrai si f est un polynôme de degré 3, en effet, on peut se ramener à $f(x) = x^3$ (puisque la formule est exacte en degré plus petit que 2), on calcule alors :
`r:=int(x^3-lagrange([x0,x1,x2],[x0^3,x1^3,x2^3]),x,x0,x2)`
puis on tape `normal(r)` qui renvoie 0.

La formule des 3 niveaux :
Soit un solide de volume : $V = \int_a^b S(z)dz$ où $S(z)$ est la surface de la section par le plan de cote z. Si $S(z)$ est un polynôme en z de degré au plus 3 alors :
$$V = \frac{(b-a)}{6}(S(a) + 4S(\frac{a+b}{2}) + S(b))$$
Ceci résulte du résultat précédent en prenant comme fonction f la fonction S.

Remarque 1 Cette formule est en fait la formule de Simpson pour le calcul de l'aire sous une courbe, formule qui est exacte pour les polynômes de degré inférieur ou égal à 3, et sinon donne une bonne approximation de l'intégrale lorsque f est suffisamment régulière (pour améliorer on découpe l'intervalle $[a, b]$ en plusieurs subdivisions et on applique la formule de Simpson sur chaque subdivision).

Remarque 2 Dans le commerce les flacons ont souvent la taille fine : c'est pour les avoir bien en main mais aussi pour donner l'illusion d'un grand volume, puisque la section médiane compte 4 fois ! ! !

Remarque 3 Cette formule donne facilement le volume d'un tonneau de hauteur $2d$ (sphère de rayon R sans ses 2 calottes). En effet $S(z) = \pi(R^2 - z^2)$ d'où :
$V = \frac{2d}{6}(\pi(R^2-d^2) + 4\pi R^2 + \pi(R^2-d^2)) = \frac{\pi}{3}d(6R^2-2d^2) = \frac{2\pi}{3}d(3R^2-d^2)$
On retrouve bien : $V = 2\pi\int_{-d}^{d}\int_0^{\sqrt{R^2-z^2}} rdrdz = \frac{2\pi}{3}d(3R^2-d^2)$ et aussi lorsque $d = R$, le volume de la sphère égal à $\frac{4\pi R^3}{3}$.

Cinquième partie

Fiche `Xcas` pour la spécialité maths Terminale S.

Ces commandes se trouvent dans le menu `Cmds`.

Arithmétique	
`iquo(a,b), irem(a,b)`	quotient, reste de la division euclidienne de a par b
`isprime(p)`	Test de primalité
`gcd(a,b), lcm(a,b)`	PGCD, PPCM de 2 entiers
`iabcuv(a,b,c)`	Renvoie u et v tels que $au + bv = c$
`powmod(a,n,m)`	Renvoie $a^n \pmod m$
`L:=convert(a,base,b)`	liste des chiffres de a en base b. L[0]=ch des unités
`a:=convert(L,base,b)`	conversion inverse
`n:=a % b; p:=n % 0`	$n \in \mathbb{Z}/b\mathbb{Z}$ congru à a ; $p \in \mathbb{Z}$ et $p = a \pmod b$
Matrices et vecteurs	
`matrix(3,4,(j,k)->j+k)`	matrice définie par une formule
`M:=[[1,2,3],[4,5,6]]`	matrice définie par des coefficients (ou bien créer un tableur Xcas en lui donnant un nom de variable)
`v:=[0,1,0]`	vecteur défini par ses coordonnées
`M[0,1]` ou `M(1,2)`	élément de `M` ligne 1 colonne 2 : les indices commencent à 0 ou à 1 selon la notation
`M[j,k]:=a`	modifie l'élément d'indice j,k (copie de `M`)
`M[j,k]=<a` ou `M(j,k)=<a`	modifie en place l'élément d'indice j,k de `M`
`+,-,*`	addition, soustraction, produit de matrices/vecteurs
`inv(M)`	inverse d'une matrice carrée `M`
`M^n`	puissance entière d'une matrice `M`
`matpow(M,n)`	puissance (symbolique) d'une matrice `M`. Écrire `supposons(n>0)` si `M` n'est pas inversible
`P,D:=jordan(M)`	diagonalisation de la matrice `M` (hors programme)
Autres	
`asc("chaine")`	renvoie la liste des codes ASCII d'une chaine
`char(L)`	renvoie la chaine de caractères à partir d'une liste
`rsolve` et `seqsolve`	résolution de suites récurrentes
`L:=readrgb("fich")`	lecture du fichier image (jpg, png) dans `L`
`writergb("fich.png",L)`	stocke et affiche l'image contenue dans `L`
`rectangle(dx,0,dy/dx, gl_texture="fich")`	affiche l'image de taille `dx,dy` contenue dans le fichier "fich"

Attention : la notation `M(j,k):=a` est interprétée comme une définition de fonction si `j,k` est symbolique (non entier) et non comme une affectation de l'indice j,k de M. On peut utiliser la notation `M(j,k)=<a` qui modifie toutes les matrices partageant la représentation de M, est donc beaucoup plus rapide pour de grosses matrices, mais peut avoir des effets surprenants.
N.B. : il peut être nécessaire d'utiliser la version 0.9.9 ou ultérieure de `Xcas`.

Cryptographie

`s:="un message a coder"; L:=asc(s)`, la liste `L` contient une suite d'entiers compris entre 0 et 255. Pour le système RSA, on peut générer une paire p, q de nombres premiers et définir $n = pq$, par exemple :

```
p:=nextprime(10^30); q:=nextprime(10^15); n:=p*q;
```

(on peut aussi utiliser de l'aléatoire) puis on génére une paire de clefs c, d :

```
n1:=euler(n); c:=rand(); d:=inv(c % n1) % 0
```

puis on code : `N:=powmod(L,c,n);` et décode : `char(powmod(N,d,n));`
Pour éviter des attaques évidentes, on peut créer des regroupements de 8 par deux conversions en base :

```
l:=convert(L,base,256); M:=convert(l,base,256^8)
```

Pour le système de Hill, on crée par exemple une matrice aléatoire 2,2

```
n:=nextprime(512); M:=ranm(2,2) % n; Minv:=inv(M);
```

(on recommence si `M` n'est pas inversible), puis on calcule les produits par paire d'éléments de `L` (ajouter un espace dans `s` en fin de chaine si le nombre d'éléments est impair)

```
N:=seq(M*[L[2*j],L[2*j+1]],j,0,size(L)/2-1)
```

Le décodage en utilisant : `O:=seq(Minv*N[j],j,0,size(N)-1)`
puis il faut aplatir `O` avant d'appeler `char` :

```
P:=[];
pour j de 0 jusque size(O)-1 faire P:=concat(P,O[j]);
fpour; P
```

Systèmes dynamiques, graphes probabilistes

Ouvrir un tableur (menu Tableur), indiquez par exemple 4 lignes et 4 colonnes et choisissez un nom de matrice, par exemple `M` puis remplissez-le par exemple avec

$$\begin{pmatrix} \frac{1}{3} & \frac{1}{4} & 0 & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ 0 & \frac{1}{4} & 0 & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \end{pmatrix}$$

Si vous n'avez pas le bon nombre de lignes ou de colonnes ou oublié le nom de variable, cliquez sur la ligne `Sheet config...` du tableur. On a ici la somme des **colonnes** de `M` égales à 1. Donc si `V0` est le vecteur initial, pour avoir la probabilité de la cinquième étape, on écrira `V5:=M^5*V0`. Vous pouvez faire une étude formelle `supposons(n>0);Mn:=matpow(M,n);limit(Mn,n=inf)`
ou numérique avec `v:=seq(1./nrows(M),j,1,nrows(M));`

```
pour j de 1 jusque 100 faire v:=M*v; fpour;
```

Attention, les probabilistes codent habituellement par une matrice `A` dont les sommes des **lignes** sont égales à 1, transposée de `M` ci-dessus. Donc si `V0` est le vecteur initial, pour avoir la probabilité de la cinquième étape, on écrira `V5:=V0*A^5`.

Un exemple de modèle proie-prédateur se trouve sur le site pédagogique de Xcas : `www-fourier.ujf-grenoble.fr/~parisse/irem.html`

Images

Voir la session `image.xws` du menu `Aide->Exemples->demo`.

Autres

Cherchez le mot-clef fougere (F12 dans Xcas), ou fraction continue (en lien avec l'identité de Bézout, les réduites successives sont les coefficients de Bézout de $au_n - bv_n = (-1)^n r_n$), ...

5 Exemples d'utilisation

5.1 Cryptographie de Hill

On associe aux nombres 0,..9,10 les caractères `"0",.."9",":"` et aux nombres 11,..36 les lettres `"A",.."Z"` (cf `co(nbre)` et `cod(lettre)`). On utilise `asc,char,op` (par ex `op(asc("09:AZ"))=(48,57,58,65,90)`).
Le texte T à coder, (complété éventuellement avec `":"`) est découpé en p blocs successifs de 2 lettres. À ces blocs on associe une matrice à coefficients entiers B de 2 lignes et p colonnes (cf `B:=coda(T)`).
La matrice de codage M est une matrice 2*2 à coefficients entiers, connue de l'expéditeur et du destinataire du message. Ici, `M:=irem(ranm(2,2),37)`= `[[7,11],[8,11]]` et `N:=irem(Inverse(M),37)`=`[[-1,1],[-6,-4]]`
Le produit $C = MB$ est une matrice colonne que l'on transforme modulo 37 en le texte codé TC (cf `TC:=codag(C)`) et finallement `TC:=codage(T,M)`.
Pour décoder, il faudra faire le chemin inverse : `codage(TC,N)`.

```
co(c):={
  local n;
  si c<0 alors c:=c+37; fsi;
  si c<=10 alors c:=c+48; sinon c:=c+54; fsi;
  return char(c);
}:;
cod(l):={
  local n;
  n:=op(asc(l));
  si 48<=n and n<=58 alors n:= n-48; sinon n:=n-54 fsi;
  return n;
}:;
coda(s):={
  local j,n,B,p;
  n:=dim(s);
  si odd(n) alors s:=s+":"; n:=n+1;fsi;
  p:=n/2;B:=makemat(0,2,p);
  pour j de 0 jusque p-1 faire
   B[0,j]:=cod(s[2*j]);B[1,j]:=cod(s[2*j+1]);
  fpour;
  return B;
}:;
codag(B):={
  local s,n,j,k;
  n:=coldim(B);s:="";
  pour j de 0 jusque n-1 faire
   pour k de 0 jusque 1 faire
     s:=s+co(B[k,j]);
   fpour;
  fpour;
  return s;
}:;
```

```
codage(s,M):={
  local B,C;
  B:=coda(s);
  C:= irem(M*B,37);
  return codag(C);
}:;
```

On tape : `M:=[[7,11],[8,11]]`
`N:=irem(Inverse(M),37)` renvoie `[[-1,1],[-6,-4]]`
`codage("ABCD012",M)` renvoie `"NYMZAACE"`
`codage("NYMZAACE",N)` renvoie `"ABCD012:"`
Mon message à décoder : `"A8P0FPSGGZDS6FZTR1MXO1DZG9Y9B5N3"`

5.2 Systèmes dynamiques

On considère 2 urnes A et B, et 3 boules numérotées de 0 à 2. Au début, toutes les boules se trouvent dans l'urne A. Aux étapes 1, 2,,..n, on tire au hasard de façon équiprobable, un nombre entre 0 et 2 (`rand(2)`). On change d'urne la boule correspondante. Déterminer la matrice de transition M et trouver la probabilité de chacun des états lorsque $n = 5$ puis lorsque $n = 10$.
Trouver la probabilité de chacun des états lorsque n tend vers l'infini.

Pour $j = 0..3$, lorsqu'il y a $(3 - j)$ boules en A et j boules en B on dit que les urnes sont dans l'état j. La probabilité de passer à chaque étape de l'état j à l'état k est donné par $M[j, k]$, avec

$$M = \begin{pmatrix} 0 & 1 & 0 & 0 \\ \frac{1}{3} & 0 & \frac{2}{3} & 0 \\ 0 & \frac{2}{3} & 0 & \frac{1}{3} \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

On tape `(M^5)[0]`
On obtient lorsque $n = 5$: `[0,61/81,0,20/81]` (2 boules en A et 1 boule en B avec une probabilité de 61/81 ou 3 boules en B avec une probabilité de 20/81)
On tape `(M^10)[0]`
On obtient lorsque $n = 10$: `[4921/19683,0,14762/19683,0]` (3 boules en A avec une probabilité de 4921/19683 ou 1 boule en A et 2 boules en B avec une probabilité de 14762/19683)
On tape :
`supposons(n,integer):;M0:=matpow(M,2*n):;limit(M0,n=inf)`
On obtient lorsque n est pair et tend vers $+\infty$: $\begin{pmatrix} \frac{1}{4} & 0 & \frac{3}{4} & 0 \\ 0 & \frac{3}{4} & 0 & \frac{1}{4} \\ \frac{1}{4} & 0 & \frac{3}{4} & 0 \\ 0 & \frac{3}{4} & 0 & \frac{1}{4} \end{pmatrix}$
On tape :
`supposons(n,integer):;M1:=matpow(M,2*n+1):;limit(M1,n=inf)`
On obtient lorsque n est impair et tend vers $+\infty$: $\begin{pmatrix} 0 & \frac{3}{4} & 0 & \frac{1}{4} \\ \frac{1}{4} & 0 & \frac{3}{4} & 0 \\ 0 & \frac{3}{4} & 0 & \frac{1}{4} \\ \frac{1}{4} & 0 & \frac{3}{4} & 0 \end{pmatrix}$

Sixième partie

Fiche `Xcas` analyse

Dérivées	
`diff(E)` ou `E'`	expression de la dérivée de l'expression E par rapport à x
`diff(E,t)` ou `(E,t)'`	expression de la dérivée de l'expression E par rapport à t
`diff(f)` ou `f'`	fonction dérivée de la fonction f
`diff(E,x$n,y$m)`	expression $\frac{\partial E}{\partial x^n \partial y^m}$, dérivée partielle de l'expression E
`grad`	gradient
`divergence`	divergence
`curl`	rotationnel
`laplacian`	laplacien
`hessian`	matrice hessienne

Limites et développements limités	
`limite(E,x,a)`	limite qd $x \to a$ de l'expression E
`limite(E,x,a,1)`	limite qd $x \to a^+$ de l'expression E
`limite(E,x,a,-1)`	limite qd $x \to a^-$ de l'expression E
`taylor(E,a)`	développement limité de E en $x = a$ ordre 5
`series(E,x=a,n)`	développement limité de E en $x = a$ ordre $n \in \mathbb{N}$

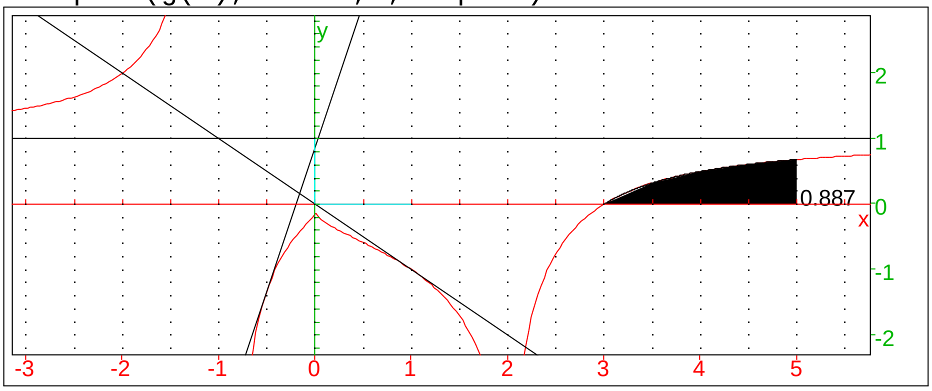
Intégrales	
`int(E,x)`	primitive de l'expression E par rapport à x
`int(f)`	fonction primitive de la fonction f
`int(E,x,a,b)`	intégrale exacte de l'expression E entre $x = a$ et $x = b$
`romberg(E,x,a,b)`	intégrale approchée de l'expression E entre $x = a$ et $x = b$

Équations	
`solve(eq,x)`	solution exacte dans $\mathbb{R}$ d'une équation polynomiale
`solve([eq1,eq2],[x,y])`	solution exacte dans $\mathbb{R}$ d'un système polynomial
`csolve(eq,x)`	solution exacte dans $\mathbb{C}$ d'une équation polynomiale
`csolve([eq1,eq2],[x,y])`	solution exacte dans $\mathbb{C}$ d'un système polynomial
`fsolve(eq,x=x0)`	solution approchée d'une équation (`x0`=`x` estimé)
`fsolve([eq],[var],[val])`	solution approchée d'un système (`val`=estimation)
`newton(f(x),x,x0,nbiter)`	racine de $f(x)$ par la méthode de Newton
`linsolve`	système linéaire
`proot`	racines approchées d'un polynôme

Équations différentielles	
`desolve`	résolution exacte
`odesolve`	résolution approchée
`plotode`	tracé d'une solution
`plotfield`	tracé du champ de vecteurs
`interactive_plotode`	tracé de solutions définies à la souris

Tracés de courbes	
`plot` ou `graphe`	graphe d'une expression d'une variable
`tangente`	tangente à une courbe
`pente`	pente d'une droite
`plotfunc`	graphe d'une expression d'1 ou 2 variable(s)
`...(...,couleur=...)`	pour choisir la couleur d'un tracé
`plotarea`	affiche l'aire sous une courbe
`plotparam`	courbe paramétrique
`plotpolar`	courbe en polaires
`plotimplicit(f(x,y),x,y)`	courbe implicite de $f(x, y) = 0$

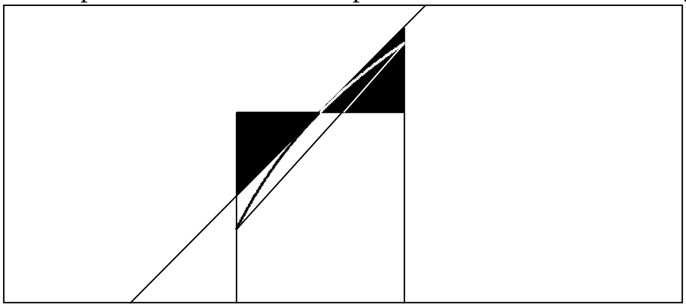
Un exemple : étude de la fonction f définie par : $f(x) = \dfrac{\ln(|2-x|)}{\ln(|x|)}$.
On va montrer que f est définie sur $\mathbb{R} - \{-1, 0, 1, 2\}$ et peut être prolongée sur $\mathbb{R} - \{-1, 2\}$. Puis on fera le graphe de f, on tracera les tangentes en $x = -1/2$, $x = 0$ et $x = 1$, on donnera une valeur approchée à 10^{-3} près de l'aire comprise entre $x = 3, x = 5, y = 0$ et la courbe avec une subdivision en 4 trapèzes.
On tape : `f(x):=ln(abs(x-2))/ln(abs(x))`
`limite(f(x),x,1)` renvoie -1 et `limite((f(x)+1)/(x-1),x,1)` renvoie -1, donc la pente de la tangente au point (1,-1) vaut -1.
`limite(f(x),x,0)` renvoie 0, `limite(f(x)/x,x,0,1)` renvoie $-\infty$ et `limite(f(x)/x,x,0,-1)` renvoie $+\infty$. Donc Oy est tangent en (0,0).
`limite(f(x),x,-1)` renvoie ∞, donc $x = -1$ est asymptote.
`limite(f(x),x,2)` renvoie $-\infty$, donc $x = 2$ est asymptote.
`limite(f(x),x,inf)`, `limite(f(x),x,-inf)` renvoie (1,1),
$y = 1$ est donc asymptote.
Pour prolonger f par continuité en $x = 0$ et $x = 1$, on tape :
`g(x):=when(x==0,0,when(x==1,-1,f(x)))`
On tape : `G:=plotfunc(g(x),x=-5..8,couleur=rouge),`
`droite(y=1),tangente(G,-1/2),droite(1-i,pente=-1),`
`areaplot(g(x),x=3..5,4,trapeze)` et on obtient :



Si on approche l'aire avec 4 trapèzes, on peut aussi taper : `Digits:=3;` puis :
`0.5*(f(3)/2+f(3.5)+f(4)+f(4.5)+f(5)/2)` et cela renvoie 0.887.
Avec `areaplot(g(x),x=3..5)` l'aire est calculée par la méthode de Romberg (c'est une accélération de la convergence de la méthode des trapèzes) et est affichée avec 3 décimales. Pour avoir plus de décimales, tapez `romberg(g(x),x,3,5)`.
Si `Digits:=12;`, on obtient `0.903226168665`.

6 Exercice : Nombre de chiffres de 1000 !

Prérequis Si f est concave ($f''(x) \leq 0$) ou convexe ($f''(x) \geq 0$) la méthode des trapèzes et la méthode du point milieu encadrent l'intégrale en effet :



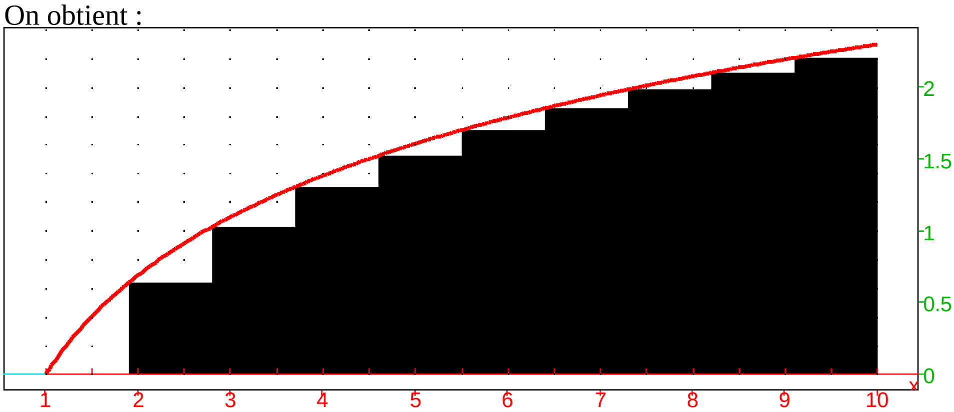
si f est concave (resp convexe), la courbe est située sous (resp au-dessus de) la tangente au point milieu et les 2 triangles noirs sont égaux.

Soient $S_n = \ln(n!) = \sum_{k=1}^{n} \ln(k)$ pour $n \in \mathbb{N}^*$ et $I_n = \int_1^n \ln(x)dx$.

- En faisant une intégration par parties calculer I_n.
- En utilisant la méthode des rectangles montrer que : $S_{n-1} < I_n < S_n$.
- En déduire que : $I_n < S_n < I_n + \ln(n)$ et donner un encadrement de I_n en fonction de n.
- Utilisez la méthode des trapèzes et du point milieu pour trouver un meilleur encadrement de I_n.
- En déduire $p \in \mathbb{N}$ tel que : $10^{p-1} < 1000! < 10^p$.
  Quel est le nombre de chiffres de 1000 ! ?

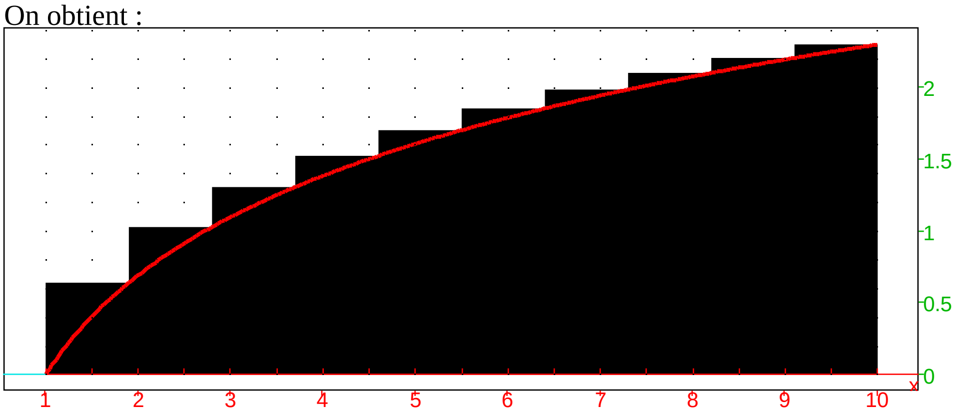
La solution avec `Xcas`

- Calcul de I_n
  On tape : `ibpu(ln(x),ln(x),x,1,n)`
  On obtient : `[n*ln(n),-1]`
  On tape : `ibpu( [n*ln(n),-1],0,x,1,n)`
  On obtient : `-n+1+n*ln(n)`
  Donc $I_n = 1 - n + n\ln(n)$
- On approche l'aire sous la courbe de $y = \ln(x)$. La somme des aires des rectangles gauches (inférieurs) est S_{n-1} et la somme des aires des rectangles droits (supérieurs) est S_n. Donc $S_{n-1} < I_n < S_n$.
  On visualise les rectangles gauches lorsque $n = 10$:
  `plotarea(ln(x),x=1..10,10,rectangle_gauche)`
  On obtient :

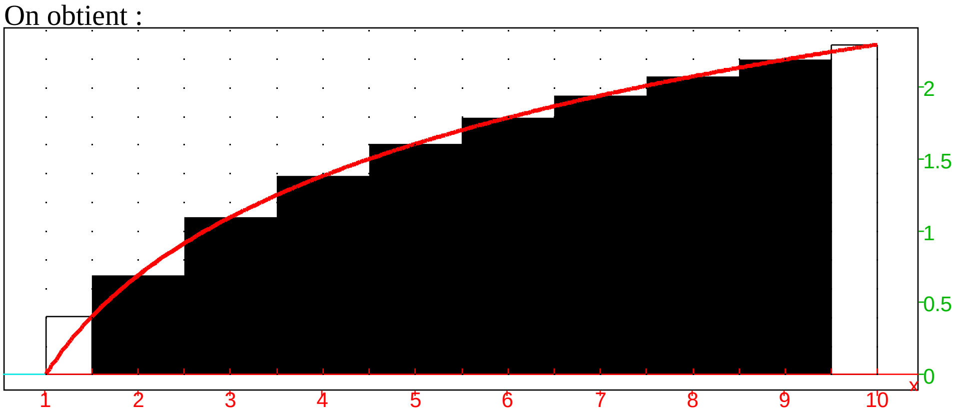


On visualise les rectangles droits lorsque $n = 10$:
`plotarea(ln(x),x=1..10,10,rectangle_droit)`

On obtient :



– Puisque $S_{n-1} + \ln(n) = S_n$ et que $S_{n-1} < I_n < S_n$, on en déduit que :
  $I_n < S_{n-1} + \ln(n) = S_n < I_n + \ln(n)$
  et donc puisque $I_n = 1 - n + n\ln(n)$:
  $1 - n + n * \ln(n) < S_n = \ln(n!) < 1 - n + n\ln(n) + \ln(n)$
– La somme des aires des trapèzes T_n est inférieure à I_n et on a :
  $T_n = \ln(2) + .. \ln(n-1) + \ln(n)/2 = \ln(n!) - \ln(n)/2 < I_n$
  On prend comme point milieu $2, 3..n-1$ et on rajoute l'aire de 2 rectangles
  (pour majorer l'aire entre 1 et 3/2 et pour majorer l'aire entre $n - 1/2$ et n).
  On visualise les rectangles de la méthode du point milieu lorsque $n = 10$:
```
plotarea(ln(x),x=3/2..19/2,8,point_milieu);
rectangle(1,1.5,ln(1.5)*2),rectangle(9.5,10,ln(10)*2)
plotfunc(ln(x),x=1..10,affichage=1+epaisseur_ligne_3)
```
  On obtient :



  On a : $I_n < M_n = \ln(3/2)/2 + \ln(2) + ... \ln(n-1) + \ln(n)/2$
  Puisque $I_n = 1 - n + n\ln(n)$, on obtient comme encadrement :

$$1 - n + n\ln(n) + \frac{\ln(n)}{2} - \frac{\ln(3/2)}{2} < \ln(n!) < 1 - n + n\ln(n) + \frac{\ln(n)}{2}$$

– On cherche p le nombre de chiffres de 1000 ! c'est á dire $p \in \mathbb{N}$ tel que $10^{p-1} < 1000! < 10^p$ c'est à dire tel que $p - 1 \leq \ln(1000!)/\ln(10) < p$
  On évalue l'encadrement de $\ln(1000!)/\ln(10)$, on tape :
```
M:=subst((1-n+n*ln(n)+ln(n)/2)/ln(10),n=1000):;
evalf(M-ln(3/2)/(2*ln(10)),2),evalf(M,2)
```
  On obtient : `2567.55,2567.64`
  Donc $p - 1 = 2567$. Le nombre de chiffres de 1000 ! est donc $p = 2568$.
  Pour vérifier, on tape `dim(string(1000!))` et on obtient : `2568`

Septième partie

Fiche Xcas la géométrie

Fonctions de géométrie 2-d	
`point`	point donné par ses coordonnées ou son affixe
`affichage=...`	dernier argument de `point` pour l'afficher selon...
`legend`	met du texte à partir d'un point donné
`segment`	segment donné par 2 points
`droite(A,B)`	droite passant par A, B
`droite(a*x+b*y+c=0)`	droite d'équation $ax + by + c = 0$
`triangle(A,B,C)`	triangle de sommets A, B, C
`bissectrice(A,B,C)`	bissectrice issue de A du triangle ABC
`angle(A,B,C)`	mesure (en radians ou degrés) de $\widehat{BAC}$
`mediane(A,B,C)`	médiane issue de A du triangle ABC
`hauteur(A,B,C)`	hauteur issue de A du triangle ABC
`mediatrice(A,B)`	médiatrice de AB
`carre(A,B)`	carré direct de côté AB
`cercle(A,r)`	cercle de centre A, rayon r
`cercle(A,B)`	cercle de diamètre AB
`rayon(c)`	longueur du rayon du cercle c
`centre(c)`	centre du cercle c
`distance(A,B)`	distance de A à B (point ou courbe)
`inter(G1,G2)`	liste des points de $G1 \cap G2$
`inter_unique(G1,G2)`	un des points de $G1 \cap G2$
`supposons`	rajout d'un paramètre symbolique ou d'hypothèses
`element`	rajout d'un paramètre numérique
`polygone`	polygone fermé
`polygone_ouvert`	polygone ouvert
`coordonnees`	coordonnées d'un point
`equation`	équation cartésienne
`parameq`	équation paramétrique
`translation(B-A,M)`	image de M par la translation $\overrightarrow{AB}$
`symetrie(A,M)`	image de M par la symétrie point (ou droite) A
`rotation(A,t,M)`	image de M par la rotation centre A et angle t
`homothetie(A,k,M)`	image de M par l'homothétie centre A et rapport k
`similitude(A,k,t,M)`	image de M par la similitude centre A, rapport k et angle t

En cochant `Landscape`, les lignes de commandes sont sous l'écran graphique.
Pour construire un nouvel objet géométrique dans une figure, on a 2 possibilités :
- soit on sélectionne un `Mode` à la souris puis on clique les objets définissant le nouvel objet. Si $\boxed{\sim}$ est décoché, les points cliqués sont à coordonnées exactes et si $\boxed{\sim}$ est coché, les points cliqués sont à coordonnées décimales.
- soit on remplit les lignes de commande, situées à gauche de la figure, en tapant la commande ou en s'aidant du menu `Geo`. L'exécution d'une ligne de commande par `Enter` entraine l'exécution des lignes suivantes.

Par exemple, pour tracer un triangle ABC, la médiatrice de AB et le cercle circonscrit à ABC, on ouvre un niveau de géométrie 2d (`Alt+g`) et avec la souris :

- On choisit comme mode : `Mode▶Polygones▶triangle`
  On clique pour désigner le premier point, puis on déplace la souris. Le segment AS où S est le point pointé par la souris, se dessine en pointillé. On clique pour désigner le 2-ième point, puis on déplace la souris un triangle en pointillé se dessine. On clique pour désigner le 3-ième point. On obtient un triangle et des lignes de commandes apparaissent (`A:=point(...)`, ...).
- On choisit comme mode : `Mode▶Lignes▶mediatrice`
  On clique sur le point A, puis on déplace la souris : la médiatrice de AS, où S est le point pointé par la souris, se dessine en pointillé. On clique sur B. On obtient la médiatrice de AB et dans la ligne de commandes on a :
  `E:=mediatrice(A,B,'affichage'=0)`
- On choisit comme mode :`Mode▶Cercles▶circonscrit`
  On clique successivement sur les points A, B : le cercle circonscrit à ABS, où S est le point pointé par la souris, se dessine en pointillé. 0n clique sur C. On obtient le cercle circonscrit à ABC et dans la ligne de commandes on a :
  `F:=circonscrit(A,B,C,'affichage'=0)`
- On choisit `Mode▶Pointeur` pour pouvoir déplacer le point A, B ou C.

0n peut aussi taper directement les commandes dans les lignes de commandes :

```
A:=point(-1,2);
B:=point(1,0);
C:=point(-3,-2);
D:=triangle(A,B,C);
E:=mediatrice(A,B);
F:=circonscrit(A,B,C);
```

Objets graphiques 3-d	
`plotfunc`	surface par équation
`plotparam`	surface ou courbe paramétrique
`point`	point donné par la liste de ses 3 coordonnées
`droite`	droite donnée par 2 équations ou 2 points
`inter`	intersection
`plan`	plan donné par son équation ou 3 points
`sphere`	sphère donnée par centre et rayon
`cone`	cône donné par sommet, axe, demi-angle au sommet
`cylindre`	cylindre donné par axe, rayon, [hauteur]
`polyedre`	polyèdre donné par ses sommets
`tetraedre`	tétraèdre régulier direct ou pyramide
`tetraedre_centre`	tétraèdre régulier direct
`cube`	cube
`cube_centre`	cube
`parallelepipede`	parallélépipede
`octaedre`	octaèdre
`dodecaedre`	dodécaèdre
`icosaedre`	icosaèdre

7 Exemples d'utilisation

7.1 Une démonstration avec Xcas

Soient 3 points A, B, C d'affixe a, b, c. Montrer que le triangle ABC est équilatéral direct si et seulement si $a + bj + cj^2 = 0$ où $j = \exp(2i\pi/3)$.

7.1.1 La solution avec Xcas

On tape dans un niveau de géométrie :

```
supposons(b1=[1.3,-5,5,0.1]);
supposons(b2=[2.3,-5,5,0.1]);
B:=point(b1+i*b2);
supposons(c1=[-2.2,-5,5,0.1]);
supposons(c2=[3.5,-5,5,0.1]);
C:=point(c1+i*c2);
j:=exp(2*i*pi/3);
a:=-b*j-c*j^2;
A:=point(a);
est_equilateral(A,B,C);
```

La réponse pour `est_equilateral(A,B,C);` est `1` et comme tous les calculs sont faits avec des paramètres formels cela vaut une démonstration.
Pour la réciproque, on remplace les 3 dernières lignes par :

```
triangle_equilateral(B,C,A);
a:=affixe(A)
normal(a+b*j+c*j^2);
```

la réponse pour `normal(a+b*j+c*j^2);` est `0` et comme tous les calculs sont faits avec des paramètres formels cela vaut une démonstration.

7.1.2 La solution sans Xcas

On a $j^2 = \exp(4i\pi/3)$, $-j^2 = \exp(i\pi/3)$ et $1 + j + j^2 = 0$
ABC est équilatéral direct équivaut à $(a - b) = -j^2(c - b)$ donc équivaut à :
$a - b(1 + j^2) + cj^2 = a + bj + cj^2 = 0$.

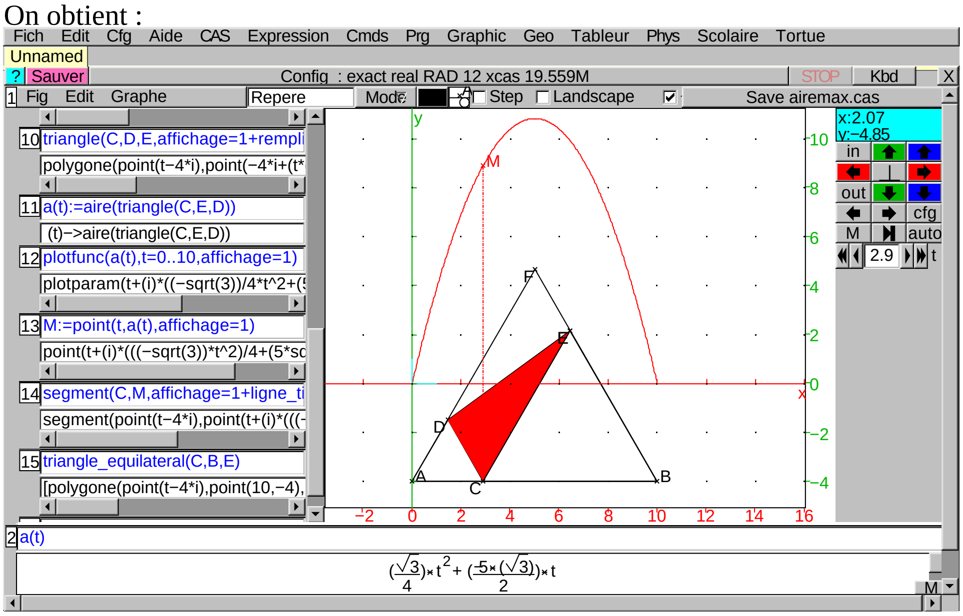
7.2 Maximiser une aire

Soient un segment AB de longueur 10 unités et un point C de ce segment. On pose $AC = t$. On construit du même côté de AB les triangles équilatéraux ACD et CBE tel que $(\overrightarrow{AC}, \overrightarrow{AD}) = (\overrightarrow{CB}, \overrightarrow{CE}) = \pi/3$.
Pour quelles valeurs de t l'aire $a(t)$ du triangle CED est-elle maximale ?
Avec Xcas, on choisit A de coordonnées $(0, -4)$ et B de coordonnées $(10, -4)$. C est alors de coordonnées $(t, -4)$. On tape dans un niveau de géométrie 2d :

```
A:=point(0,-4);
B:=point(10,-4);
supposons(t=[2.9,0,10,0.1]);
```

```
C:=point(t,-4);
triangle_equilateral(A,C,D);
triangle_equilateral(C,B,E);
segment(D,E);
triangle_equilateral(A,B,F);
triangle(C,D,E,affichage=1+rempli);
a(t):=aire(triangle(C,E,D));
plotfunc(a(t),t=0..10,affichage=1);
M:=point(t,a(t),affichage=1);
segment(C,M,affichage=1+ligne_tiret_point);
```

On obtient :



La parabole en rouge est la représentation graphique de l'aire $a(t)$ du triangle CDE (en rouge). Lorsque C d'abscisse t se déplace sur AB à l'aide du curseur t, le point M de coordonnées $(t, a(t))$ décrit cette parabole.

Attention `aire` renvoie une aire algébrique. `aire(triangle(C,E,D))` est positive car le triangle CED est direct et on a :

`aire(triangle(C,E,D))=-aire(triangle(C,D,E))`.

Dans la mesure où les coordonnées des points A et B sont des nombres exacts (par exemple $(10, -4)$ et non $(10.0, -4.0)$), l'instruction de `Xcas` :

`supposons(t=[2.9,0,10,0.1]);` a pour effet de faire le dessin avec la valeur de t donné par le curseur (ici $t = 2.9$), mais de faire tous les calculs en fonction de la variable symbolique t. On tape `a(t)` dans une ligne de commande (cf niveau 2) et on obtient : `(sqrt(3))/4*t^2+(-5*sqrt(3))/2*t`

On peut facilement calculer cette aire qui est la moitié de l'aire du parallélogramme $DCEF$ avec $CE = 10-t$, $DC = t$ et $(\overrightarrow{AC}, \overrightarrow{AD}) = \pi/3$ donc ce parallélogramme a pour hauteur $CH = t\sqrt{3}/2$. Donc $a(t) = \frac{\sqrt{3}}{4}t(10-t) = \frac{-\sqrt{3}}{4}t^2 + \frac{5\sqrt{3}}{2}t$.

On tape pour calculer $a'(t)$: `factor(diff(a(t),t))`

On obtient : `((-(sqrt(3)))*t+5*sqrt(3))/2`

Donc $a(t)$ a un maximum pour $t = 5$ c'est à dire lorsque C est le milieu de AB.

Ce maximum vaut $\frac{25\sqrt{3}}{4}$ c'est l'aire d'un triangle équilatéral de côté 5=$AB/2$.

Huitième partie

Fiche `Xcas` la programmation

1. **Pour écrire une fonction ou un programme**, il faut :
   - choisir une syntaxe, on décrit ici la syntaxe `Xcas`,
     – soit avec le menu `Cfg`▶`Mode(syntax)`▶`xcas`,
     – soit en ouvrant la fenêtre de configuration du CAS en appuyant sur le bouton `Config:...` et choisir `Xcas` dans `Prog style`,
   - ouvrir un niveau éditeur de programme soit en tapant `Alt+p`, soit avec le menu `Prg`▶`Nouveau programme`. Une boite de dialogue s'ouvre pour faciliter la définition d'une nouvelle fonction.
   - Taper la fonction ou le script (suite de commandes) en utilisant les boutons `Fonctions`, `Test` et `Boucles` (assistant de création d'une fonction,d'un test ou d'une boucle) ou le menu `Ajouter` de l'éditeur. Le nom du programme et de ses arguments ne doit pas être un mot-clef ou une commande de `Xcas`, ces mots apparaissent en bleu et brun.
   - cliquer sur `OK` ou appuyer sur `F9`, pour compiler le programme (ou exécuter le script si on n'a pas terminé son écriture par `:;`).
   - pour exécuter le programme, il suffit de se placer dans une ligne de commandes vide, de taper le nom du programme suivi entre parenthèses par les valeurs des paramètres séparées par des virgules et d'appuyer sur `Enter`.

2. **Le menu `Ajouter` d'un niveau éditeur de programme**
Ce menu vous permet d'avoir la syntaxe d'une fonction, d'un test et des boucles.

Syntaxe d'une fonction :

```
f(x,y):={
  local z,a,...,val;
  instruction1;
  instruction2;
  val:=...;
  .....
  instructionk;
  retourne val;
}:;
```

Euclide et l'identité de Bézout :

```
Bezout(a,b):={
 local la,lb,lr,q;
  la:=[1,0,a];
  lb:=[0,1,b];
 tantque b!=0 faire
   q:=iquo(la[2],b)
   lr:=la+(-q)*lb;
   la:=lb; lb:=lr;
   b:=lb[2];
 ftantque
 retourne la;
}:;
```

3. Compilation La réponse à une compilation reussie sera `Done` si `:;` termine le programme et sera la traduction du programme si c'est `;` qui le termine.
Pour `Bezout(a,b)`, on clique sur `OK` (ou touche `F9`) et on obtient `// Parsing Bezout // Success compiling Bezout` puis `Done`. Puis, on tape : `Bezout(78,56)` et on obtient `[-5,7,2]` (car -5*78+7*56=2=pgcd(78,56)).
4. Pas à pas Vous pouvez exécuter un programme commandes par commandes ou le mettre au point grâce au débugueur. On tape : `debug(Bezout(78,56))`
Une fenêtre s'ouvre et on appuie sur `sst` pour une exécution au pas à pas.

Instructions en fançais	
affectation	a:=2;(a prend la valeur 2) et purge(a);(a redevient formelle)
entrée expression	saisir("a=",a);
entrée chaine	saisir_chaine("a=",a);
sortie	afficher("a=",a);
valeur retournée	retourne a;
arrêt de la boucle	break;
alternative	si <condition> alors <inst> fsi;
	si <condition> alors <inst1> sinon <inst2> fsi;
boucle pour	pour j de a jusque b faire <inst> fpour;
	pour j de a jusque b pas p faire <inst> fpour;
boucle répéter	repeter <inst> jusqua <condition>;
boucle tantque	tantque <condition> faire <inst> ftantque;
boucle faire	faire<inst1>si(<condition>)break;<inst2>ffaire;

Instructions comme en C++	
affectation	a:=2;(a prend la valeur 2) et purge(a);(a redevient formelle)
entrée expression	input("a=",a);
entrée chaine	textinput("a=",a);
sortie	print("a=",a);
valeur retournée	return(a);
arrêt de la boucle	break;
alternative	if (<condition>) {<inst>};
	if (<condition>) {<inst1>} else {<inst2>};
boucle pour	for (j:= a;j<=b;j++) {<inst>};
	for (j:= a;j<=b;j:=j+p) {<inst>};
boucle répéter	repeat <inst> until <condition>;
boucle tantque	while (<condition>) {<inst>};
boucle faire	do <inst1> if (<condition>) break; <inst2> od;
debug	lance le debugueur pour une exécution au pas à pas de l'argument

Signification des signes de ponctuation	
.	sépare la partie entière de la partie décimale
,	sépare les éléments d'une liste ou d'une séquence
;	termine chaque instruction d'un programme
:;	termine les instructions lorsqu'on ne veut pas l'affichage de la réponse
!	$n!$ est la factorielle de $n \in \mathbb{N}$ et $!(b)$ est l'inverse logique du booléen b

Opérateurs			
+	addition	-	soustraction
*	mutiplication	/	division
^	puissance	a mod p	a modulo p
==	teste l'égalité	!=	teste la différence
<	teste la stricte infériorité	<=	teste l'infériorité ou l'égalité
>	teste la stricte supériorité	>=	teste la supériorité ou l'égalité
||, ou	opérateur booléen infixé ou	&&, et	opérateur booléen infixé et
non	inverse logique de l'argument	!(..)	inverse logique de l'argument
vrai	est le booléen vrai ou true ou 1	faux	est le booléen faux ou false ou 0
=	permet de définir une équation	:=	affectation ($a := 2$)

8 Exemples d'utilisation

8.1 Programmer un jeu

On considère le jeu suivant :

– Le joueur tire au hasard un nombre entre 1 et 10. Ce nombre est son gain, il peut le garder et la partie est finie.
– Le joueur peut choisir d'abandonner ce gain et de recommencer. Il lui est permis de tirer ainsi au plus 5 fois et son gain est le résultat du dernier tirage lorsqu'il a abandonné les gains des 4 tirages précédents.

1. Écrire un programme qui réalise ce jeu,
2. Le joueur choisit 4 nombres n_1, n_2, n_3, n_4 compris entre 1 et 10 et choisit comme stratégie de garder son gain du $j$ième essai si celui-ci est supérieur ou égal à n_j sinon il recommence (si n_1=9, n_2=8, n_3=6, n_4=5 avec le tirage 2,7,8, le gain est 8 mais avec le tirage 8,7,5,4,2 le gain est 2).
   Il joue 100 parties selon cette stratégie avec les mêmes n_j. Écrire un programme donnant la moyenne des gains obtenus. A vous de jouer !.

```
jeu1():={
 //repondre o pour dire oui
 local j,g,rep;
 pour j de 1 jusque 4 faire
   g:=alea(10)+1;afficher(g);
   lis_phrase(rep);
   si rep=="o" alors retourne g; fsi;
 fpour;
 retourne alea(10)+1;
}:;
```

`jeu2` a comme paramètre `L` qui est la séquence n_1, n_2, n_3, n_4 puis LL est la suite $n_1, n_2, n_3, n_4, 0$ de façon à ce que $LL[j]$ soit définie pour $j = 0..4$ ($LL[0] = n_1$).
`jeu102` est la moyenne des gains de 100 parties faites avec `jeu2`.

```
jeu2(L):={
 local j,g,LL;
 LL:=append(L,0);
 pour j de 0 jusque 4 faire
   g:=alea(10)+1;afficher(g);
   si g>=LL[j] alors break fsi;
 fpour;
 retourne g;
}:;
jeu102(L):={
 local g,j;
 g:=0;
 pour j de 1 jusque 100 faire
   g:=g+jeu2(L);
 fpour;
 retourne evalf(g/100);
}:;
```

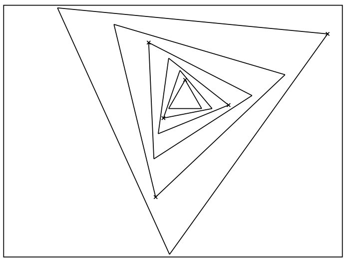
8.2 Des triangles équilatéraux emboiés

À partir d'un triangle équilatéral direct ABC on construit les points A_1, B_1, C_1 vérifiant : $\overrightarrow{AA_1} = \frac{4}{3}\overrightarrow{AB}$, $\overrightarrow{BB_1} = \frac{4}{3}\overrightarrow{BC}$, $\overrightarrow{CC_1} = \frac{4}{3}\overrightarrow{CA}$. Interpréter A_1 comme le barycentre de (A, a) et (B, b) et montrer que le triangle $A_1B_1C_1$ est équilatéral.
On recommence la même construction à partir de $A_1B_1C_1$. Écrire la procédure récursive qui réalise le dessin des n triangles obtenus par cette construction.
On a : $\overrightarrow{AA_1} = \frac{4}{3}\overrightarrow{AB} - \frac{1}{3}\overrightarrow{AA}$. Donc A_1 est le barycentre de $(A, -1)$ et $(B, 4)$.
B_1 est le barycentre de $(B, -1)$ et $(C, 4)$. C_1 est le barycentre de $(C, -1)$ et $(A, 4)$.
La rotation r de centre O, le centre de ABC, et d'angle $\frac{2\pi}{3}$ transforme A en B, B en C et C en A donc r transforme le barycentre de $(A, -1)$ et $(B, 4)$ en le barycentre de $(B, -1)$ et $(C, 4)$ c'est à dire transforme A_1 en B_1 et r transforme le barycentre de $(B, -1)$ et $(C, 4)$ en le barycentre de $(C, -1)$ et $(A, 4)$ i.e. transforme B_1 en C_1. Donc le triangle $A_1B_1C_1$ est équilatéral.

On tape dans l'éditeur de programmes :

```
triangles(A,B,n):={
  local L,C;
  L:=triangle_equilateral(A,B,C);
  si n>0 alors
    A:=barycentre([A,B],[-1,4]);
    B:=barycentre([B,C],[-1,4]);
    L:=L,triangles(A,B,n-1);
  fsi;
  return L;
}:;
```

On compile ($F9$). Dans une ligne de commandes, on tape :
`triangles(0,1,5)`
On obtient :



8.3 Les nombres amiables

Écrire un programme qui pour un entier n, donne la suite des couples amiables (a, b) ($a \leq b \leq n$). Deux nombres a et b sont **amiables**, si l'un est égal à la somme des diviseurs propres de l'autre (sauf lui-même) et inversement.

On utilise l'instruction `idivis(a)` qui renvoie la liste des diviseurs de l'entier `a` et l'instruction `sum(L)` qui renvoie la somme de la liste `L`. On tape :

```
amiable(n):={
  local j,a,b,L:=NULL;
  pour j de 2 jusque n faire
    a:=sum(idivis(j))-j;
    b:=sum(idivis(a))-a;
    si b==j et j<=a alors L:=L,[j,a]; fsi;
  fpour;
  retourne L;
}:;
```

`amiable(500)` renvoie `[6,6],[28,28],[220,284],[496,496]`
Les nombres parfaits a sont les nombres amiables (a, a).

Neuvième partie

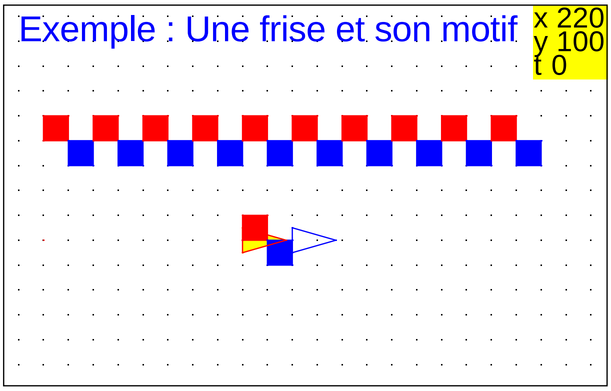
Fiche `Xcas` la tortue

Déplacements	
`efface`	pour effacer
`avance`	pour avancer
`recule`	pour reculer
`saute`	pour avancer ou reculer sans tracer
`pas_de_cote`	pour faire des pas de coté à gauche sans tracer (ou à droite si l'argument est <0)
`tourne_gauche`	pour tourner à gauche
`tourne_droite`	pour tourner à droite
Les couleurs et la tortue	
`crayon`	pour connaitre ou changer la couleur du crayon
`cache_tortue`	cache la tortue
`montre_tortue`	montre la tortue
`dessine_tortue(n)`	dessine la tortue en forme pleine (n=0) ou non (n=1)
Les formes	
`rond`	dessine un cercle ou un arc de cercle
`triangle_plein`	dessine un triangle plein
`rectangle_plein`	dessine un carré, un rectangle, un losange ou un un parallélogramme plein
`disque`	dessine disque ou secteur circulaire tangent à la tortue
`disque_centre`	dessine disque ou secteur circulaire de centre la tortue
`polygone_rempli`	remplit le polygône dessiné juste avant

Les légendes	
`ecris`	pour écrire là où se trouve la tortue ou en un point
`signe`	pour signer, en bas à gauche, son dessin
La programmation et la tortue	
`si` c `alors` $inst$ `fsi`	fait l'instruction $inst$ si la condition c est vraie
`si` c `alors` $inst1$ `sinon` $inst2$ `fsi`	fait l'instruction $inst1$ si la condition c est vraie ou fait l'instruction $inst2$ si c est fausse
`repete` $n, inst1, inst2$	repète n fois les instructions $inst1, inst2$
`pour` j `de` a `jusque` b `faire` $inst$ `fpour`	fait l'instruction $inst$ en itérant la variable j de a à b avec un pas égal à 1
`pour` j `de` a `jusque` b `pas` p `faire` $inst$ `fpour`	fait l'instruction $inst$ en itérant la variable j de a à b avec un pas égal à p
`tantque` c `faire` $inst$ `ftantque`	fait l'instruction $inst$ tant que la condition c est vraie
`retourne` obj	définit la valeur de la fonction par la valeur de obj
`lis`(a)	met dans a, une expression lue au clavier
`lis_phrase`(a)	met dans a, une phrase lue au clavier
`sauve`$(''toto'', f, g)$	sauve dans le fichier $toto$ les fonctions f, g
`ramene`$(''toto'')$	ramène les fonctions du fichier $toto$

Positionnement et Autres	
`position`	pour avoir la position de la tortue ou pour changer sa position
`cap`	pour avoir le cap de la tortue ou pour changer son cap
`vers`	pour diriger la tortue selon un point
`hasard n`	est un entier aléatoire ente 0 et `n-1`

Il ne peut y avoir qu'un seul écran de dessin tortue par session.
Pour piloter la tortue, on met des commandes à gauche du dessin : soit on les tape en toutes lettres, soit on les sélectionne dans le menu `Tortue`, soit on clique sur leurs abréviations situées sur la barre des boutons (le bouton `cr` affiche en plus la palette des 256 couleurs du crayon), soit on réutilise une commande déjà tapée.
À droite de l'écran se trouve l'enregistreur qui enregistre toutes les commandes utilisées : en cas d'erreur, il est facile de le modifier, puis d'exécuter toutes les commandes depuis le début en appuyant sur `F7`.



Cette frise est la répétition d'un même motif, isolé ci-dessus, avec les positions de départ et d'arrivée de la tortue. On réalise d'abord le dessin du motif : on ouvre un niveau de dessin tortue (`Alt+d`) et on met dans les lignes de commandes :

```
crayon 1;
rectangle_plein ;
saute ;
tourne_droite ;
crayon 4;
rectangle_plein ;
tourne_gauche ;
saute ;
```

Pour cela on appuie successivement sur les boutons `cr, rp, sa, td,...`
Ces commandes se mettent automatiquement à droite dans l'enregistreur (une erreur se corrige directement dans l'enregistreur et on réexécute celui-ci avec `F7`).
On ouvre un éditeur de programme (`Alt+p`) et d'un coup de souris on recopie les commandes de l'enregistreur dans l'éditeur. On remplace alors `efface;` par `motif():={`. Puis, on rajoute `}:;` et on appuie sur `F9`.
On tape ensuite à gauche, dans les lignes de commandes : `repete 10,motif()`
Vous pouvez zoomer le dessin, en avant ou en arrière, avec la molette de la souris et déplacer la fenêtre visible en la translatant avec la souris.
Cet exemple vous montre comment on réalise un dessin complexe en le décomposant et aussi comment on utilise l'enregistreur pour écrire facilement une procédure à partir d'un dessin exécuté en pas à pas.

9 Exercices

9.1 La toile d'araignée et la trigonométrie

La procédure `polyg(l,k)` trace un polygone régulier direct de `k` côtés de longueur `l`. La tortue part d'un sommet et est dirigée vers un sommet et revient à son point de départ. On tape dans l'éditeur de programme :

```
polyg(l,k):={
repete(k,avance l,tourne_gauche 360/k);
}:;
```

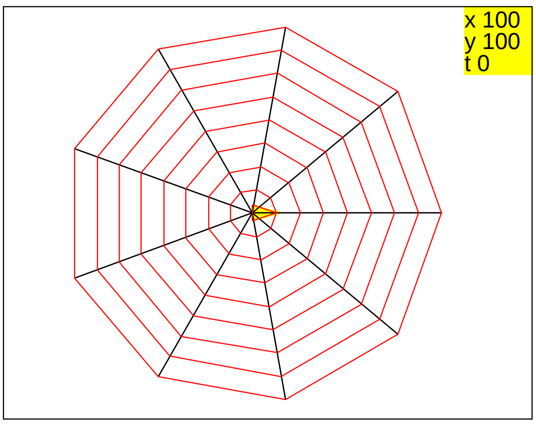
La procédure `araignee(k,p,n)` trace `n` polygones emboités, réguliers, directs et de `k` côtés où le plus petit polygone P a des côtés de longueur `p`. La distance du centre de P a un de ces sommets donne l'espacement entre les différents polygones. La tortue part du centre et est dirigée vers un sommet et revient à son point de départ. On tape dans l'éditeur de programme :

```
araignee(k,p,n):={
local r,j;
r:=(p/2)/sin(pi/k);
repete(k,avance n*r,recule n*r,tourne_gauche 360/k);
pour j de 1 jusque n faire
  saute r;
  tourne_gauche 90+180/k;
  crayon rouge;
  polyg(j*p,k);
  tourne_droite 90+180/k;
fpour;
saute -n*r;
}:;
```

Puis, on tape dans un niveau dessin tortue :

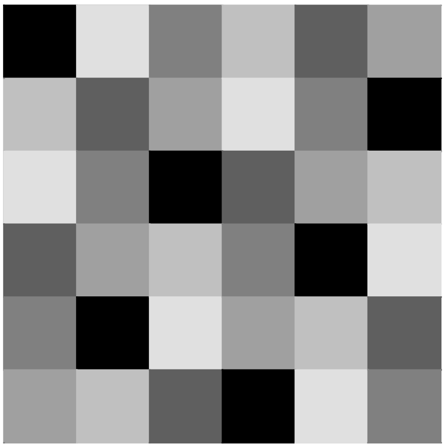
```
efface ;
crayon jaune;
dessine_tortue;
crayon noir;
araignee(9,10,8);
```

On obtient :



9.2 Les carrés magiques

Le départ de cette activité est un tableau de Richard Paul Lohse vu au musée de Grenoble. En voici une reproduction faite avec la tortue : les couleurs ne sont pas respectées...Ici, les couleurs ont comme code `[0,52,43,49,40,46]`.



On remarque que la permutation `p:=[3,4,5,1,2,0]` effectue le passage des couleurs d'une ligne à la suivante (traduit par `coul:=passage(coul,p);` ou par `coul:=coul[p[j]]$(j=0..5);`). On tape dans l'éditeur de programmes :

```
ligne(coul):={
 local j;
 pour j de 0 jusque 5 faire
  crayon coul[j];rectangle_plein(30);saute(30);
 fpour
 saute -180;
}:;
passage(l,p):={
 local n,j,lp;
 n:=size(l);lp:=l;
 pour j de 0 jusque n-1 faire
  lp[j]:=l[p[j]];
 fpour
 retourne lp;
}:;
lohse():={
 local j,p,coul;
 coul:=[0,52,43,49,40,46];//[180,168,3,136,93,78];
 pour j de 0 jusque 5 faire
  ligne(coul);p:=[3,4,5,1,2,0];
  coul:=passage(coul,p);pas_de_cote -30;
 fpour;
 pas_de_cote 180; cache_tortue;
}:;
```

On tape dans un niveau de dessin tortue `lohse()` et on obtient le dessin ci-dessus.