chap. 1 : Introduction

I - Premiers pas

1) Les entiers

▶ Taper `1+1;` ← ne pas oublier le ; ou : Si on ne veut pas d'affichage

on peut aussi taper `1+2` → le ; est dessous si on l'a oublié
`;`

ou bien 2 instructions sur la même ligne :

`1+3; 7x5; 2^2000;`

▶ Opérations de base : + − * / ^
elles ont une priorité différente : ^ d'abord
*, / ensuite
+, − après

ex : `1+3*4/5^2;` est ≡ à $1 + \frac{3\times 4}{5^2}$

à comparer avec

`(1+3)*4/5^2;`

ou `1+3*(4/5)^2;`

▶ A la différence de calculatrices, Maple fait des calculs exacts sur les entiers.

exemple : `8/6;` donne comme résultat 4/3 et non 1,333...

→ il le considère comme 2 objets différents :

`whattype(4/3);` donne "fraction"

`whattype(1.333333);` donne "float"

autre exemple : `1000!/999! - 1000;` donne 0 sur maple
donne ≠0 à la calculette

▶ fonctions agissant sur les entiers :

`irem(a,b)` ou `a mod b` reste division euclidienne

`iquo(a,b)` quotient

`igcd(a,b)` pgcd

`ilcm(a,b)` ppcm

de fonction rigolotes sur les nombres premiers :

`isprime(n)` test de primalité

`prevprime(n)` } le + proche nb premier inf. ou sup.
`nextprime(n)`

`ithprime(p)` p-ième nb premier

et plus spécifiquement pour les rationnels :

`numer(a/b)` numérateur et dénominateur de
`denom(a/b)` la fraction.

ex : `numer(1/2 + 3/5);` `numer(1/2 + sqrt(3)/5);`

2) Les réels

▶ C'est l'emploi du • décimal qui force maple à faire du calcul en virgule flottante comme la calculette

ex : `1/3;` et `1./3;` ou alors `1e-7`)

`arccos(1/2)` et `arccos(0.5)`

▶ Pour changer le nombre de décimales :

`Digits := 20;`

▶ Affichage :

`cos(Pi/3);` donne 1/2

`cos(Pi/3.);` donne cos(0,333 π)

pour afficher le nombre décimal :

`evalf(cos(Pi/3));`

`evalf(cos(Pi/3), 40);` avec 40 décimales

`printf("%20.17f", cos(Pi/3));`

petit exo :

$\left(\frac{1}{2^{2000}}\right)^{2000}$ et $\left(\frac{1}{2.^{2000}}\right)^{2000}$

▶ Le calcul en flottant est + rapide que le calcul exact :

```
> ss:=time();
  y1=sin(127699);
  time()-ss;
  sss:=time();
  y2:=sin(127699.);
  time()-sss;
```

$ss := 41.241$
$y1 = \sin(127699)$
0.421 ⟶ temps calcul exact
$sss := 41.662$
$y2 := -0.4423193534$
0.040 ⟶ temps calcul approché

▶ fonctions usuelles sur les réelles :

sin , cos , tan

arcsin , arccos , arctan

log ou ln , log10

exp

sqrt

abs

mais encore

csgn	signe	(~~pas~~ il existe aussi une fn signum idem sur $\mathbb{R}$)
trunc	entier le plus proche en allant vers 0	
floor , ceil	partie entière , partie entière +1	
frac	$x - \text{trunc}(x)$	⚠ si $x<0$

3) Les complexes

C'est le nombre I qui permet de la introduire

sqrt (-1) ; donne I

un complexe peut être défini en coordonnées cartésiennes :

1 + 3*I ;

ou en polaires polar (1, 2*Pi/3) ; ou 1 * exp (I* Pi/3) ;

la fonction evalc permet de mettre un complexe sous forme canonique $x + iy$. (noter que evalf marche aussi)

→ noter que max (z_1, z_2) n'existe pas

▶ fonctions spécifiques :

Re(z) Im(z)	} parties réelle et imaginaire
abs(z) , argument(z)	module et argument
conjugate(z)	conjugué .
csgn(z)	signe de la partie réelle
signum(z)	$\frac{z}{\lvert z \rvert}$

4) Utilisation de variables

L'affectation se fait au moyen du signe :=

```
x := 3;      t := "prout";
```

pour effacer une variable : `unassign('x');`

⚠ pas `unassign(x)` : si x vaut 3 maple comprend alors `unassign(3)`

pour effacer toutes les variables : `restart;`

▶ petit exercice à se prendre la tête

`y := x*x;`	`x := 3; y := x*x;` donne 9
`x := 3; y;` donne 9	`x := 7:`
`x := 7: y;` donne 49	`y;` donne 9 et pas 49 !
affectation différée de y	affectation immédiate de y.

Dans le 1er cas, y est une expression. quand on demande de l'afficher, elle est réévaluée.

Dans le 2e cas, y est un nombre. Il aura même valeur même si on change x

II - Un peu de calcul formel

▶ dériver une fonction : `diff(cos(x), x);` |x|

... mais aussi : `diff(cos(x)^n, x);` `diff(cos(x)^n, n);`] fonction de 2 variables

Intégrer une fonction : `int(1/x, x)` donne la primitive

intégrale définie : `int(1/(1+x*x), x=0..1);`

`int(1/(1+x*x), x=0..infinity);` (infinity = ∞)

Développer des expressions : `expand((1+x)^3);`

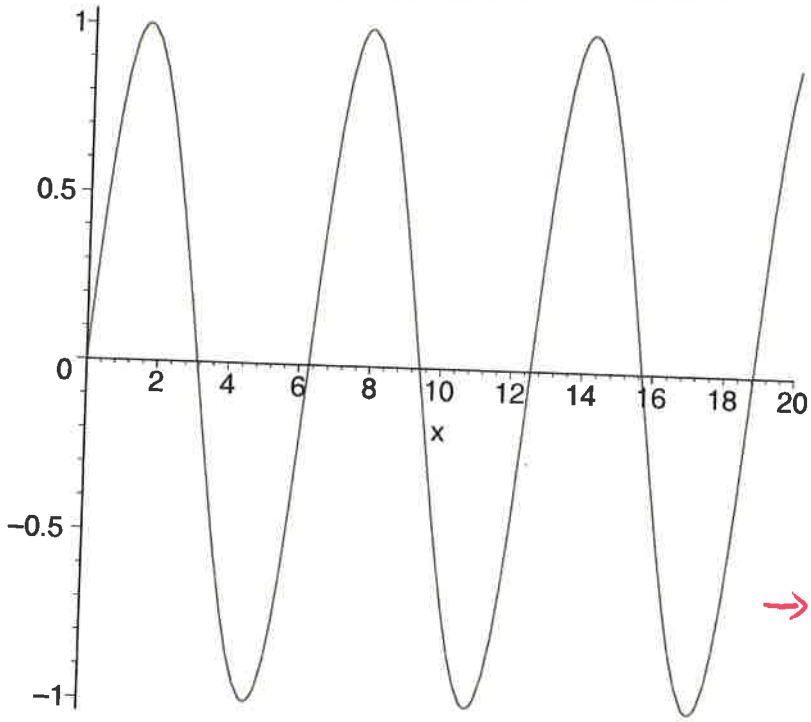
ou les factoriser : `factor(1+x^2+2x);`

prendre une limite : `limit(1/x, x=infinity);`

III. Un peu de graphique

fonction de base : plot

ex : `plot ( sin(x), x = 0 .. 20 );`



→ clic droit pour accéder aux options

Syntaxes possibles : `plot ( sin );` → trace le sinus entre -10 et 10

`plot (sin(x), x = 0..20, y = -2..2 );`

y = -2..2 : change le vertical range (on doit alors préciser le range)

options :

color = red ou blue, green, ...

linestyle = 1 à 4 1: solid 2: dot 3: dash 4: dashdot

axes = FRAME, BOXED, NORMAL, NONE : la boîte

labels = ["xlabel", "ylabel"] ; dessine les labels des axes

labeldirections : [HORIZONTAL, VERTICAL] direction des labels, horizontal par défaut

title : "Titre" ; met le titre

style : point, symbol = box

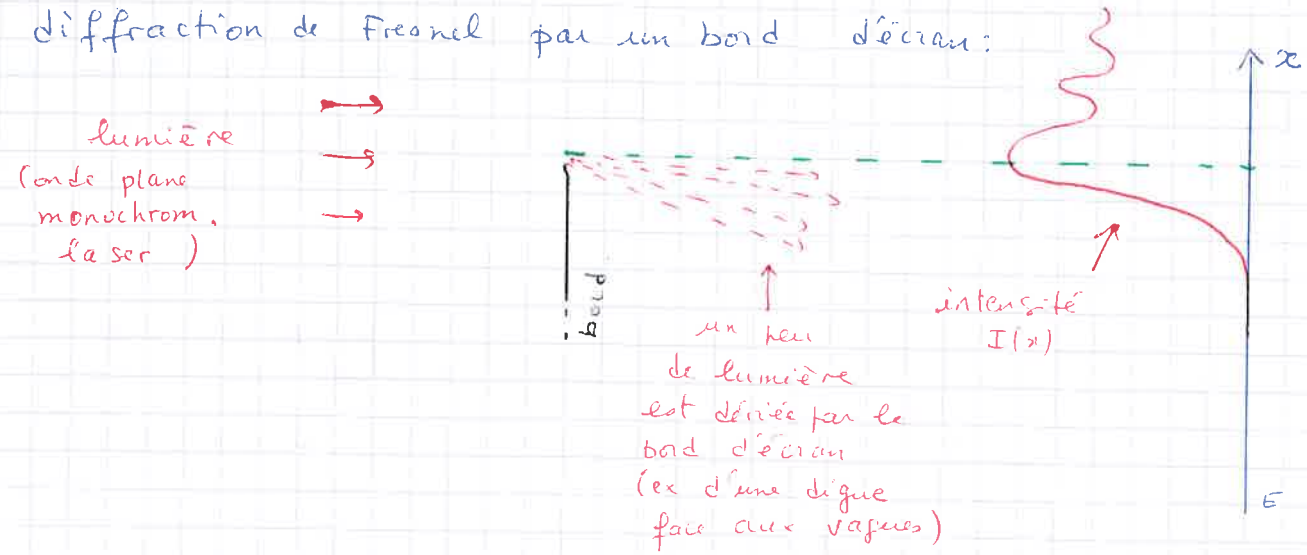
Pour tracer deux courbes : `plot ( [f, g] );` ou `plot ( [f(x), g(x)], x = 0..1 );`

dans ce cas les options peuvent prendre 2 valeurs : color = [red, blue]

et la possibilité de mettre une légende : legend = ["f", "g"]

IV. Exemple d'application.

diffraction de Fresnel par un bord d'écran :



l'intensité en un point x sur l'écran E s'écrit :

$$I(x) = Cte \left| k_0 + \int_0^x \exp\left(i\pi \frac{t^2}{\lambda z}\right) dt \right|^2$$

avec $k_0 = \sqrt{\frac{\lambda z}{2}} \cdot \left(\frac{1+i}{2}\right)$

$$Cte = \frac{I_0}{\lambda z}$$ avec I_0 l'intensité initiale.

Pour tracer cette fonction sous maple :

Diffraction par un bord d'écran

Idée : mettre z et lambda à 1 (calcul exact)
puis z:=1. (calcul approché) et voir la différence de temps
puis lambda=5e-7 m et z=384000 km pour les franges de la Lune

```
> restart; lambda:=5e-7: z:=384000000.:
  k0:=(1+I)/2*sqrt(lambda*z/2):fff:=abs(k0+int(exp(I*Pi*t*t/(lambd
  a*z)),t=0..x))^2;
```

$$fff := \left| 4.898979486 + 4.898979486\, I + (4.898979486 + 4.898979486\, I)\,\mathrm{erf}((0.09045015682 - 0.09045015682\, I)\, x) \right|^2$$

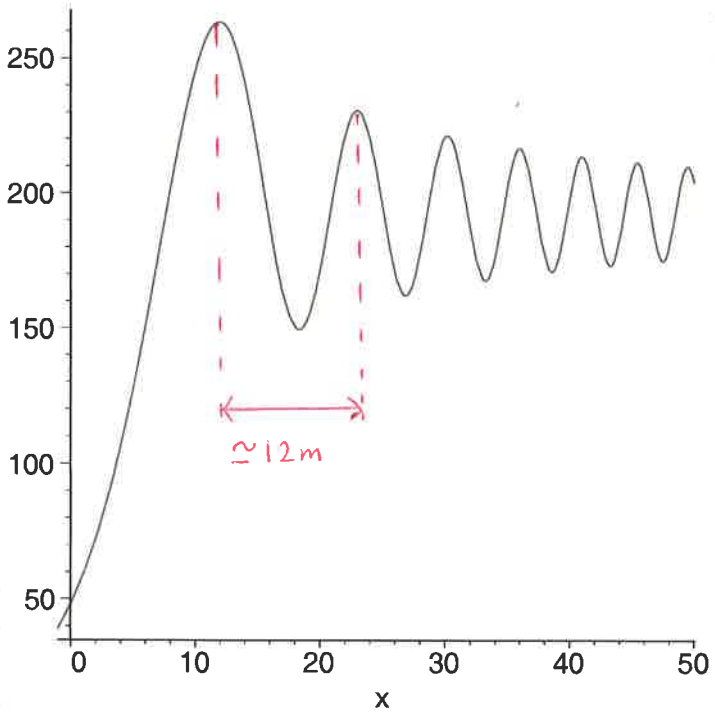
```
> sss:=time(): plot(fff,x=-1..50); time()-sss;
```

avec $z=1$ et $\lambda=1$ et $x=-1..5$ → 4,203 s

$z=1.$, $\lambda=1.$ " → 2,183 s

$z=384000000.$, $\lambda=5.10^{-7}$, $x=-1..50$ → 1,970 s

exemple du bord lunaire :



à la vitesse de la Lune

$v = \frac{2\pi \cdot 384\,000 \text{ km}}{29 \text{ j}}$

$\Rightarrow v \simeq 1 \text{ km/s}$

les franges balayent la terre en

$t \simeq 12 \text{ ms}$

V - Utilisation d'une mise en page.

Exemple à faire en live :

Feuille de Calcul

Premier exo

Si on clique, ça se ferme

Commentaire (#)

Calculer la dérivée du sinus

```
> restart: # remet tout a zero
  diff(sin(x),x);
```

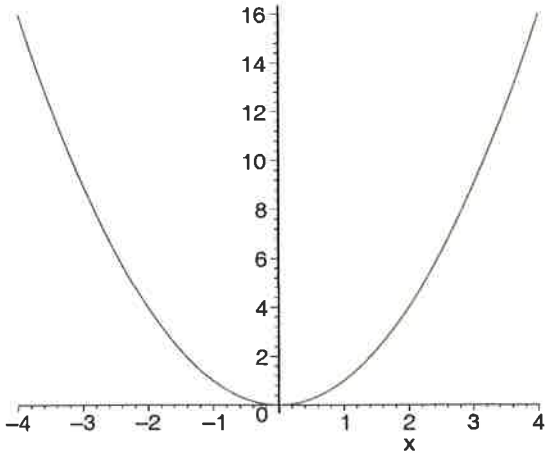
$$\cos(x)$$

```
>
```

Second exo

Tracer la fonction y=x

```
> restart:
  plot(x*x,x=-4..4);
```



Les menus
Insert → text pour écrire du texte (titre par ex)
insert → section ouvre une section
insert → maple input } pour entrer des commandes maple
insert → Execution group

Autre menu utile : Edit → Split or Join pour grouper ou diviser des cellules.

Chap 2 : Fonctions et Calcul formel

I - Calcul algébrique

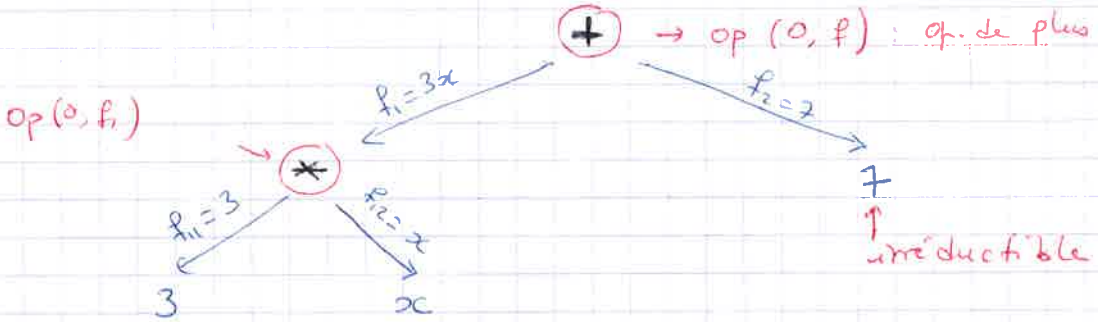
1) Les expressions [NB: on peut sauter ce §]

Une formule telle que $3x + 7$ est une expression
il s'agit de variables ou constantes ou de fonctions reliées entre elles par des opérateurs.

Les opérateurs ont une priorité :

^ est prioritaire devant * / devant + −

C'est cette priorité qui lui permet de représenter les expressions sous la forme d'un arbre : si $f := 3x + 7$



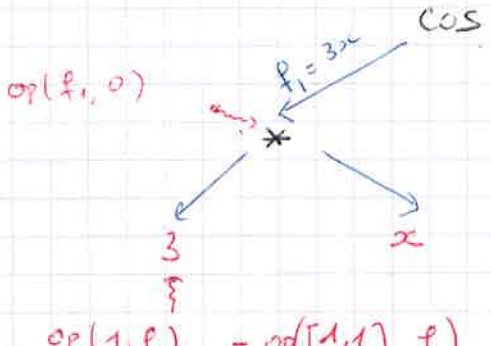
idée: on décompose l'expression en blocs de plus en plus petits jusqu'à tomber sur des éléments irréductibles.

Plusieurs fonction maple manipulant les expressions :

- `nops(expr)` donne le nombre d'opérandes
- `op(expr)` donne les opérandes (renvoie une séquence)
- `op(1, expr)` donne l'opérande 1
- `op(0, expr)` opérateur de plus basse priorité

Autre exemple :

$f := \cos(3 * x)$



`nops(f)` donne 1

`op(0,f)` donne cos

$f_1 := op(1, f)$ donne $3x$

`op([1,1], f)` donne 3

* changement de variable : [+ de détails au § suivant]

l'instruction `subs ( oldexpr = newexpr, f )`

permet de remplacer l'expression oldexpr par newexpr.

ex : `f := 3x + 4 ;`

`g := subs ( x = t, f ) ;` donne $3t + 4$

`g := subs ( x = 3t+7, f ) ;` donne $9t^2 + 25$

* changement d'une expression en une autre :

c'est la fonction `subsop ( no_op = new_e, f )`

ex : `f := 3x + sin(x) ;`

`g := subsop ( 1 = x*x, f )` donne $x^2 + \sin(x)$

(par quoi on remplace ; n° de l'opérande à remplacer)

`g := subsop ( [1,1] = 4, f )` donne $4x + \sin x$

`g := subsop ( [2,0] = exp, f )` donne $3x + e^x$

2) Simplifications et transformations

* La fonction simplify

Exemples :

`simplify ( (x-a) * (x+a) ) ;` donne $x^2 - a^2$

`simplify ( 1/(1/a + 1/b) ) ;` donne $ab/(b+a)$

`simplify ( exp(a) * exp(b) ) ;` $e^{(a+b)}$

`simplify ( log ( sqrt ( x*x - 1 ) ) ) ;` $\frac{1}{2} \log(x^2 - 1)$

`simplify ( cos^4 x - sin^4 x ) ;` $2\cos^2 x - 1$

il est généralement intéressant d'utiliser simplify après un calcul. Par exemple :

`int ( y^3 exp(-y^2), y ) ;` donne $\frac{1}{2}\frac{y^2}{e^{y^2}} - \frac{1}{2}\frac{1}{e^{y^2}}$

`simplify ( % )` donne $-\frac{1}{2} e^{-y^2}(y^2+1)$

(inutile en maple 10 (déjà simplifié))

(% : rappel du dernier résultat)

`simplify ( (x-a)^2 (x-b)^2, {a=2*b} )` ou `simplify ( cos(4*x) + cos(2*x), {cos^2 x + sin^2 x = 1} );`

(pas terrible en maple 10 ; indication supplémentaire)

* la fonction assume

▶ On donne une indication à Maple sur le type de variable : s'il s'agit d'un entier, d'un réel, positif, ...

ex: $\int_0^\infty e^{-ax}\,dx$ → renvoie une limite

assume (a, positive) → l'intégrale renvoie $\frac{1}{a}$

ex: log (a·b) → renvoie log (ab)

assume (a, positive) ; assume (b, positive)

ln (a·b) → log(a) + log (b)

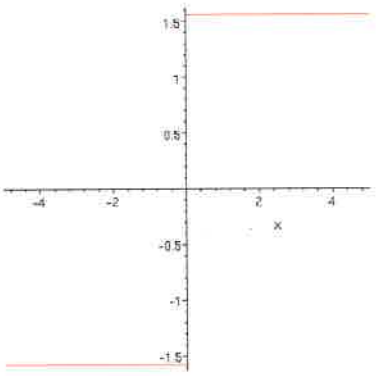
assume (b, negative)

ln (a·b) → log (a) + log(−b) + Iπ. ln (−3) pour illustrer me);

▶ Autre exemple: f := arctan (x) + arctan (1/x) ;

plot (f, x = −5..5) ;

on voit qu'elle est constante par morceaux. ←



et si on essaye de simplifier

simplify (f) → arctg x + arctg $\frac{1}{x}$

alors que la dérivée:

g := diff(f, x) ; simplify (g) ; donne 0 : fonction este

en fait il faut aider maple en lui disant que $x \in \mathbb{R}$:

assume (x, real) ; simplify (f) ;

Simplify (f, assume = real) donne $\frac{1}{2}$ signum (x) π

... ce qu'on voit bien sur le graphe.

▶ Quelques propriétés que prend assume: (help(property);)

real	$\in \mathbb{R}$
positive, negative, nonneg	$\in \mathbb{R}_+^*$, $\mathbb{R}_-^*$ $\mathbb{R}_+$
RealRange(a, b)	définit un intervalle de valeurs [a, b]
Real Range(Open(a), open(b)) ;	$\in]a, b[$
rational	$\in \mathbb{Q}$
irrational	$\in \mathbb{R} - \mathbb{Q}$ ex $\sqrt{2}$
integer	$\in \mathbb{N}$
odd, even	

▶ on peut utiliser la fonction `couldbe` pour voir si une propriété assumée est compatible avec un ensemble de valeurs.

ex :
```
assume (x, integer) ;
couldbe ( x = 1.5 ) ;        renvoie false
assume (x, real) ;
couldbe ( x = 1.5 ) ;        renvoie true.
```

▶ Un effet pervers de assume est : ~

```
assume (a>0) ;
f:= x/a ;        → renvoie x/a~
a:= 1 ; f ;      → renvoie toujours x/a~
```

Si on permute les 2 lignes ça marche !

⇒ a et a~ sont considérés comme des variables ≠

et la commande `a~ := 1` renvoie une erreur

et `` `a.~` := 1 ; f; `` renvoie encore x/a~.

ne marche pas en maple 10

le remède : une commande **unassume**

```
'~' := '' ;        ← on affecte à rien le caractère ~
unassume := parse@readlib ('convert/name') ;
```

utilisation :
```
assume (a>0) ; f:= x/a ;
a:=unassume (a) ; f:= unassume (f) ;
a:= 3 ; f ;        → renvoie bien x/3
```

On peut sauvegarder la définition de assume dans un fichier txt, par exemple unassume.txt

Pour relire le fichier `read "unassume.txt" ;`

(pas besoin de chemin d'accès si le fichier est dans C:\Program Fils\Maple 8\ sous XP)

Sinon : `read "C:\\Documents and Settings\\Administrateur\\Mes Documents\\unassume.txt" ;`

les \ doivent être doublés ; les ␣ peuvent rester tels quels.

la bonne façon :
```
assume (a>0)
f := x/a        → renvoie x/a~
g := subs (a=2, f)   → renvoie x/2   ⇒ ok !
```

⚠ ne pas ~~affecter~~ a après le assume, tout faire avec subs

* La fonction subs

Elle sert à substituer des variables ou des expressions.

Ex: `f:= 3x+4;`

`subs(x=t, f);` → $3t+4$

`subs(x=3t+7, f);` → $9t+25$

`subs(x=sin(x), f)` → $3\sin(x)+4$

il n'y a pas le `:=` ⇒ pas une affectation

on peut aussi faire des remplacements multiples:

`subs(x=cos t, y=sin t, x^2+y^2);`

`simplify(%);` → 1

mais aussi : `f:= 2*y + cos(x)^4;`

`subs(cos(x)=t+1, y=t, f);`

`simplify(%);` → $t^4+4t^3+6t^2+6t+1$

ou encore:

`subs(2=4, x^2+2cos(2x))` → $x^4+4\cos(4x)$

`subs(x=Pi/2, x^2+cos(2*x));` → $\frac{\pi^2}{4}+\cos(\pi)$;

l'expression n'est pas évaluée.

`eval(%);` → $\frac{\pi^2}{4}-1$

On a 2 façons de substituer 2 expressions.

• séquentiellement

`f:= cos^2(y)+sin^2(x);`

`subs(y=x, x=2*y, f);`

(d'abord, ensuite)

résultat:

$\sin^2 2y+\cos^2 2y$

• simultanément

`f:= cos^2(y)+sin^2(x);`

`subs({y=x, x=2*y}, f)`

(en même temps)

résultat: $\sin^2 2y+\cos^2 x$

2007: subs est intéressant à la fin d'un calcul, pour affecter des paramètres sans perdre le caractère général d'une fonction.

exemple: `f:= int(a/(1+bx^2), x)` → $\frac{a}{\sqrt{b}}\arctan(b\sqrt{x})$

`g:= subs({a=1, b=2}, f)` → $\frac{\sqrt{2}}{2}\arctan(x\sqrt{2})$

`plot(g, x=1..2)`

* la fonction expand

développe des expressions en les distribuant sur + et *

Ex: `expand ( (1+x)^2 );` → $1 + 2x + x^2$

mais aussi en utilisant des tables de simplification / combinaisons

`expand ( cos (2*x) );` → $2\cos^2 x - 1$

... et en combinaison avec simplify:

`f:= cos ( 2*x);` `f:= expand (f);` → $2\cos^2 x - 1$

`Simplify (f);` → $2\cos^2 x - 1$ → simplify n'a rien fait

`simplify (f, {cos(x)^2 + sin(x)^2 = 1});` → $1 - 2\sin^2 x$

↑ hypothèse ponctuelle donnée à simplify.

d'autres exemples :

`expand ( cos (a+b) );` → cos a cos b − sin a sin b

`f:=tan (a+b);` → `(tan(a) + tan(b)) / (1 - tan(a) tan(b));`

`simplify (f, trig);` → $\dfrac{\cos(a)\sin(a) + \cos(b)\sin(b)}{\cos^2(a) - 1 + \cos^2(b)}$

↑ simplification en utilisant seulement les règles trigo

`expand ( exp (a+b) );` → $e^a e^b$

`expand ( cosh (a+b));` → ch a ch b + sh a sh b

* la fonction combine

Compacte des expressions (inverse de expand)

combine (cos(a) * cos(b)) ; → $\frac{1}{2}\cos(a+b) + \frac{1}{2}\cos(a-b)$

combine (exp(a) * exp(b)) ; → exp(a+b)

combine (arctan(x) + arctan(1/x)); → $\frac{\pi}{2}$ signum(x)

parfois il faut l'aider :

combine (ln(x) + ln(y)); → ln(x) + ln(y)

assume x>0: assume y>0: combine(%) → ln(x~ y~)

combine ((x^y)^z) → $(x^y)^z$

assume (x>0): assume (y>0): combine(%) → $x^{(y\sim z)}$

on peut aussi utiliser combine avec des règles de transformations :

combine (expr, règle)

règle peut être : trig, exp, power, ln, ...

Ex : f:= $4\sin^3 x + e^x \times e^y$;

combine (f, trig) → $-\sin(3x) + 3\sin(x) + e^y e^x$ (il n'applique que la règle trigo)

combine (f, exp) → $4\sin^3 x + e^{x+y}$ (n'applique que la règle exp)

* la fonction factor

factorise des expressions

`factor (x^2 + 2x + 1)` → $(x+1)^2$

`factor (x^2 - y^2)` → $(x-y)(x+y)$

`factor ((x^3 + 3x^2 + 3x + 1)/(x + 3x))` → $\frac{(x+1)^3}{3x}$

mais il les factorise dans le corps ℚ :

`factor (x*x - 9/16)` → $\frac{(4x-3)(4x+3)}{16}$

factor est désemparé avec :

`factor (x*x - 5);` → $x^2 - 5$

⇒ il faut le guider : factor(expr, argument)

4 façons de faire : nombre ou option.

`factor (x*x - 5.0)`
`factor (x*x - 5, real)` } → $(x + 2{,}23)(x - 2{,}23)$: virgule flott.

`factor (x*x - 5, √5)` ← nombre → $(x - \sqrt{5})(x + \sqrt{5})$

`factor (x^2 - 5, RootOf(x^2 - 5));` `convert (%, radical);`
racine

de même pour $x^2 + 1$

* Les fonctions normal et rationalize

normal réduit les fractions rationnelles au même dénominateur

normal (1/(x-a) + 1/(x-b)) → $\frac{-2x+a+b}{(-x+a)(x-b)}$

normal ($(x^2+2x+1)/(x+1)$) → $(x-1)$

f := $(x-1)/(x^2 - x\sqrt{2} - x + \sqrt{2})$:
normal (f) → $\frac{1}{x-\sqrt{2}}$

rationalize fait à peu près la même chose, mais rend rationnel le dénominateur :

sur le dernier exemple, rationalize (f) → $\frac{x+\sqrt{2}}{x^2-2}$

* la fonction collect / sort

collect ordonne une expression par rapport à une variable

Ex : collect ($x^2 - x + 3 - 2xy$, x) → $3 + x^2 + (-1-2y)x$ → dans le désordre des puissances de x

sort (%, x) → $x^2 + (-1-2y)x + 3$ → on trie

autre ex : f := $x^2 + x\exp(x) - x - x^2\exp(x)$:

collect (%, exp(x)) → $(x - x^2)e^x + (x^2 - x)$

collect (%, x) → $(1-e^x)x^2 + (e^x - 1)x$

* la fonction convert

Convertit une expression en une autre forme.

convert (expr , forme)

forme peut être : exp ln, sin cos, radical, polynom ... de très nombreuses possibilités.

Quelques exemples :

convert (cos(x), exp ln) → $\frac{1}{2} e^{ix} + \frac{1}{2 e^{ix}}$ Transforme en exponentielles

convert (cos(x), tan) → $\frac{1 - tg^2(x/2)}{1 + tg^2(x/2)}$ tangentes

convert (arccos(x), ln) → $-i \ln\left(x + i\sqrt{1-x^2}\right)$ logarithme

convert (e^{ix}, trig) → cos x + i sin x fonction trigo

convert ($\frac{x^3+2}{x^2-1}$, parfrac , x) → $x + \frac{3}{2(x-1)} - \frac{1}{2(x+1)}$ décompose en élts simples

(3e arg.)

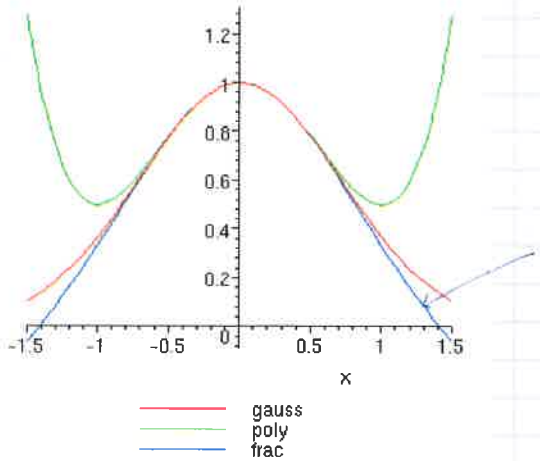
f := exp - (x * x) ; gaussienne

g := taylor (f, x = 0, 5); DL ordre 5 → $1 - x^2 + \frac{x^4}{4} + O(x^5)$

g1 := convert (g, polynom) → $1 - x^2 + \frac{x^4}{4}$ enlève le $O(x^5)$

g2 := convert (g, ratpoly) → transforme la série en fraction rationnelle (approx. de Padé fn décroissantes)

→ $\frac{1 - x^2/2}{1 + x^2/2}$



Padé reste meilleur plus loin.

Enfin une utilisation très utile : convert (n, string) → convertit en chaîne de caract.

Ex: n := 3;

plot ([cos(n*x), cos(2n*x)], x = 0..5, legend = [convert(n, string), convert (2n, string)])

ou

plot (" " , legend = [cat("n=", convert(n, string)), cat ( )]);

3) Séquences et listes

▶ Une séquence est une ~~ensemble~~ collection ordonnée d'expressions séparées par des ,

ex : `f:= x, y, x+y;`

`f[1];` donne x

▶ Une liste se différentie d'une expression par les [ ] :

`f:= [x, cos(x), sin(x)];`

la fonction plot utilise des listes de fonctions pour faire un graphe multiple

`plot(f, x=-Pi..Pi);` ⚠ si on oublie les [] dans f: erreur !

mais aussi

`diff(f, x);` donne $[0, -\sin(x), \cos(x)]$

on peut extraire une partie d'une liste.

`f[1];` donne x

`f[1..2];` donne $[x, \cos(x)]$

▶ L'instruction seq permet de construire des listes.

`seq(x^n, n=0..10)` donne la séquence $x^0, x^1, x^2 \dots x^{10}$

`[seq(x^n, n=0..10)]` donne la liste $[x^0 \dots x^{10}]$

par exemple on peut vouloir tracer les courbes $y=x^n$:

```
f:= [seq(x^n, n=0..5)];
plot(f, x=-1.2..1.2);
```

⚠ on ne peut pas utiliser seq avec des bornes variables :

`seq(x^n, n=0..nmax);` → donne une erreur.

▶ l'opérateur $ permet la même chose que seq avec des bornes variables.

`f:=[x^n $ n=0..nmax];` → x^n $ n=0..nmax

```
mmax:=3;
f;
```
→ donne $[1, x, x^2, x^3]$

Enfin, on peut définir une séquence / liste pour des valeurs quelconques de n et pas seulement des intervalles de type 1..nmax.

`seq(x^k, k=(2,7,12));` → renvoie x^2, x^7, x^{12}

⚠ marche pas avec $

4) Sommes et Produits

Les fonctions **sum** et **product** servent à sommer / multiplier des sommes / produits finis ou infinis. La syntaxe est la même.

Dans le cas de sommes finies et de produits finis conduisant à une valeur numérique, on peut utiliser **add** et **mul**

ainsi : `sum ( 1/n^2 , n = 1..infinity );` $\rightarrow \frac{\pi^2}{6}$

`sum ( x^n/n! , n=0.. infinity) ;` $\rightarrow e^x$

`sum ( q^n, n = 0.. infinity ) ;` $\rightarrow \frac{1}{1-q}$ ⚠ il faut l'hypothèse $|q| < 1$

`assume (q>1) ; sum (q^n, n=0.. infinity)` $\rightarrow \infty$

`sum ( q^n, n=0.. N ) ;` $\rightarrow \frac{q^{N+1}-1}{q-1}$ somme finie.

Sans intervalle, on obtient une somme dite indéfinie :

`sum ( k^2, k)` est équivalent à `sum (p^2, p=0.. k-1)`

`sum ( f(k), k)` donne une "primitive discrète" $G(k)$ telle que $G(k+1) - G(k) = f(k)$

Pour les produits :

`product ( k, k=1..n)` donne $\Gamma(n+1)$

↳ petit exo : le convertir en $n!$ `convert (%, factorial) ; simplify (%, {(n+1)!=(n+1)*n!})`

on peut essayer les produits suivants :

$$\prod_{n=1}^{\infty} 1 - \frac{4x^2}{\pi^2 (2n-1)^2} \rightarrow \cos x$$

$$\prod_{n=2}^{\infty} \frac{n^2-1}{n^2+1} \rightarrow \frac{\pi}{\sinh \pi}$$

et pour s'amuser : mathworld.wolfram.com/ cherche infinite product

II Fonctions d'une variable

1) Définition d'une fonction sous Maple

* A l'aide de ->

Syntaxe : `fn:=variable -> expression`

ex : `f:= x -> x^2 ;`

`f(3) ; f(t) ; f(cos(x)) ;`

▶ intérêt par rapport à `f:= x^2` par exemple si on veut la valeur en $x+1$ puis tracer la fonction

Sans définir de fonction

```
f:= x^2 ;
Subs (x=3, f)
Subs (x=x+1, f) ;

plot (f)
```

← renvoie une erreur : obligé de mettre les bornes

avec une fonction

```
f:= x -> x^2
f(3)
f(x+1) ;

plot (f)
```

▶ pas d'évaluation faite à la définition de la fonction :

```
a:=0 ;
f:= x -> x/a ;
```

→ donne $x \to \frac{x}{a}$

```
a:=0 ;
f:= x/a ;
```

→ erreur

▲ On peut définir une liste de fonctions :

```
f:= [x^2, x^4, x^8] ;
```

~~> on appelle liste quand c'est entre [ ]

```
plot ( f(x), x=0..1.2) ;
```

* A l'aide de unapply

Syntaxe : `fn:= unapply (expr, variable)`

ex : `f:=unapply (x^2, x)` définit la fonction x^2

ou : `f:= unapply ([x^2, sin(x)], x)` définit une liste de fonctions.

unapply est surtout utile pour affecter à une fonction le résultat d'un calcul :

```
int ( ln(x), x)
```

→ $x \ln x - x$

```
g:= %
```

```
f:= x -> g
```

→ ne donne pas de résultat (n'évalue pas g)

```
f:=unapply (g,x)
```

→ ça marche !

* fonctions composées

c'est l'opérateur @. Il y a 2 types de syntaxe :

- `g := f @ h` → `foh`

ex :
```
g := exp @ sin ;
g(x);   → exp(sin(x))
```

l'évaluation est automatique. ex : `g := exp @ ln ; g(x)` → donne x

on peut composer des fonctions définies par l'utilisateur.

```
f := x → x^2;
g := f @ cos ;  g(x)  → donne cos^2 x
```

- `g := f @@ n` → compose n fois la fonction f

ex : `f := x → exp(-x);`

`g := f@f` → $e^{-e^{-x}}$ } idem

`g := f@@2` → $e^{-e^{-x}}$ }

`g := f@@7` → $e^{-e^{-e^{-}}}$ } 7 niveaux.

`plot ( [ f@@3, f@@5, f@@7]);` pour s'amuser

* fonctions définies par morceaux.

Syntaxe : `fnie := x → piecewise (cond1, val1, cond2, val2, ... cond_{n-1}, val_{n-1}, val_n)`

ex : la fonction porte

$$\Pi(x) = \begin{cases} 0 & \text{si } x < -\frac{1}{2} \\ 0 & \text{si } x > 1/2 \\ 1 & \text{sinon} \end{cases}$$

`f := x → piecewise (x<-1/2, 0, x>1/2, 0, 1)`

ou

`f := x → piecewise (abs(x)<1/2, 1, 0)`

ou encore : `f := x → piecewise (x<-1/2, 0, x<1/2, 1, 0)`

cette condition n'est examinée que si la première n'est pas satisfaite donc si $x > -\frac{1}{2}$

⇒ équivalat à $x > -\frac{1}{2}$ and $x < \frac{1}{2}$

d'où la non-commutativité des arg. :

`piecewise (x<-1/2, 0, x<1/2, 1, 0)` ≠ `piecewise (x<1/2, 1, x<-1/2, 0, 0)`

On peut ensuite manipuler la fonction comme une autre :

```
f:= x -> piecewise (abs(x) < 1/2, 1, 0);
g:= int (f(x), x)
```

$$\begin{cases} g = 0 & x \leq -1/2 \\ g = x + 1/2 & -\frac{1}{2} < x \leq \frac{1}{2} \\ g = 1 & x > 1/2 \end{cases}$$

- Il existe également une fonction Heaviside qui vaut $\begin{cases} 1 \text{ si } x > 0 \\ 0 \text{ si } x < 0 \end{cases}$ et qui permet la définition de fonctions par morceaux :

ainsi la porte s'écrit `f:= Heaviside (x+1/2) - Heaviside (x-1/2)`

* Valeurs isolées.

permet de compléter la définition d'une fonction pour des valeurs isolées.

Ex : sinus cardinal.

`f:= x -> sin(x)/x;` `f(0)` → provoque une erreur

`f(0):= 1;` `f(0);` → donne 1. ok.

on aurait pu définir f par l'instruction piecewise mais alors les calculs type dérivée sont moins simples.

ex :

```
f:= x -> sin(x)/x;
f(0):= 1;
diff(f,x);
```

$$\frac{\cos x}{x} - \frac{\sin x}{x^2}$$

```
f:= x -> piecewise (x=0, 1, sin(x)/x);
diff(f,x);
```

$$\begin{cases} 0 & \text{si } x = 0 \\ \frac{\cos x}{x} - \frac{\sin x}{x^2} & x \neq 0 \end{cases}$$

* Procédures

On peut construire des fonctions élaborées grâce à la comande proc.

Exemple :

```
f:= proc (x,n);
    if (n=0) then sin(x) else cos(x) fi
    end;
```

`f(x, 0)` donne $\sin x$

`f(3*t, 1)` donne $\cos 3t$

plus vrai en V. 10

Nb : pour tracer une procédure il faut la mettre entre ' ' dans l'appel à plot :

`plot ('f(x,0)', x=-3..3);`

on peut aussi utiliser la forme fonctionnelle `plot (f, -3..3)` pour des fn d'une seule variable

* la fonction map

Elle permet d'appliquer une fonction à toutes les composants d'une liste ou toutes les opérande d'une expression.

ex: imaginons la liste de fonctions [sin, cos, tan] et qu'on veuille la mettre au carré.

$f := x \to x^2$;

g:= [sin, cos, tan];

map (f, g) → donne $[\sin^2, \cos^2, \tan^2]$

autre ex: Soit la liste $[1, x, x^2, x^3]$ à laquelle on veut appliquer cos.

map (cos, $[1, x, x^2, x^3]$) → donne $[1, \cos x, \cos x^2, \cos x^3]$

ou bien: map ($x \to x^2$, a+b+c) → $a^2 + b^2 + c^2$

2) Dérivation

▶ comme on l'a vu au chap 1, l'instruction diff permet le calcul de la dérivée d'une expression.

`diff ( sin (x), x)` donne $\cos x$

`diff ( sin (nx), x )` donne $n \cos nx$

Il existe une fonction Diff qui fait la même chose mais qui n'évalue pas : elle est dite "forme inerte" de diff

`Diff ( sin (nx), x)` donne $\frac{\partial}{\partial x} \cos nx$

`value (%)` provoque l'évaluation et donne $n \cos nx$.

On peut dériver plusieurs fois :

`diff ( f(x), x, x )` donne la dérivée du 2nd ordre $f''(x)$

`diff ( f(x), x, x, x, x)` — 4e $f^{(4)}$

mais la séquence x, x, x, x s'obtient par `x $ i=1..4`

ou plus simplement par `x $ 4`

d'où la formule pour la dérivée n-ième : `diff ( f(x), x $ n)`

▶ Il existe une version fonctionnelle de diff qui prend comme argument une fonction f (et non sa valeur f(x)) et qui renvoie la fonction dérivée f' (et non sa valeur f'(x)).

Ainsi si l'on essaye : `diff (cos)` → erreur.

Il faut taper `D (cos)` → qui donne `-sin`

ex: `f:= x → cos(x);`

`D (f);` → donne `-sin`

`D (f) (Pi/6)` → donne $-\sin x$.

ex: `f:= x → cos x`

`g:= x → x^2`

`h:=D ( g@ f);` → une fonc compliquée

`h(x);` → $-2 \cos x \sin x$

Pour une dérivée n-ième : `(D @@ n) (f)`

en fait c'est $\underbrace{D(D(\ldots D}_{n \text{ fois}}(f)))$.

Contrairement à diff, D sait dériver une fonction définie par une procédure.

ex:
```
f:= proc(x);
    if (x <0) then sin(x) else cos(x) fi
end;
```

D(f) → donne proc(x) if x<0 then cosx else -sin x fi end proc

et qu'on peut tracer par plot (D(f))

alors que si on essaye diff (f(x), x) → erreur.

3) Intégration

Cette fonction a déjà été rencontrée. On peut calculer la primitive d'une expression $f(x)$ par

`int(f(x),x)` ou sa forme inerte `Int(f(x),x)`

qui peut donner parfois des fonctions spéciales :

`int(exp(-(x*x)),x)` → $\frac{1}{2}\sqrt{\pi}\,\mathrm{erf}(x)$

noter que $\int_0^{\pi} \cos(\sin x)\,dx$ marche alors que maple ne sait pas calculer de primitive.

Pour une intégrale définie : `int(f(x), x=a..b)`

- Parfois maple ne sait pas trouver de solution. On peut alors faire une intégration numérique

ex : `f:= exp(-x^2)/(1+exp(x));`

`int(f,x);` → répond $\int \frac{e^{-x^2}}{1+e^x}\,dx$

`int(f,x=0..infinity)` → idem.

`evalf(Int(f,x=0..infinity))` → 0,3269...

forme inerte pour empêcher le calcul exact } pas obligatoire en maple 10

force l'intégration numérique.

pour des procédures, seule l'intégration numérique est possible.

```
f:=proc(x) if x<0 sin(x) else cos(x) fi: end;
evalf(Int(f,-1..1)) → 0,3817
```

réglages possibles : `evalf[n](Int(cos(x^4),x=0..10));` → précision de n digits

ou `evalf(Int(... ),x=...,digits=n);`

```
g:= evalf(int(f,x=0..y))
plot(g, y=0..5)
```

marche (maple 10)

4) Développement en série

* La commande taylor.

réalise un développement de Taylor d'une fonction f au voisinage d'un point

Syntaxe : taylor (expr, équation, ordre)

taylor (cos(x), x=0, 5) → donne $1 - \frac{x^2}{2} + \frac{x^4}{24} + O(x^5)$

convert (%, polynom) permet de supprimer le $O(x^5)$.

la commande coeftayl permet d'avoir le coefficient de l'ordre n.

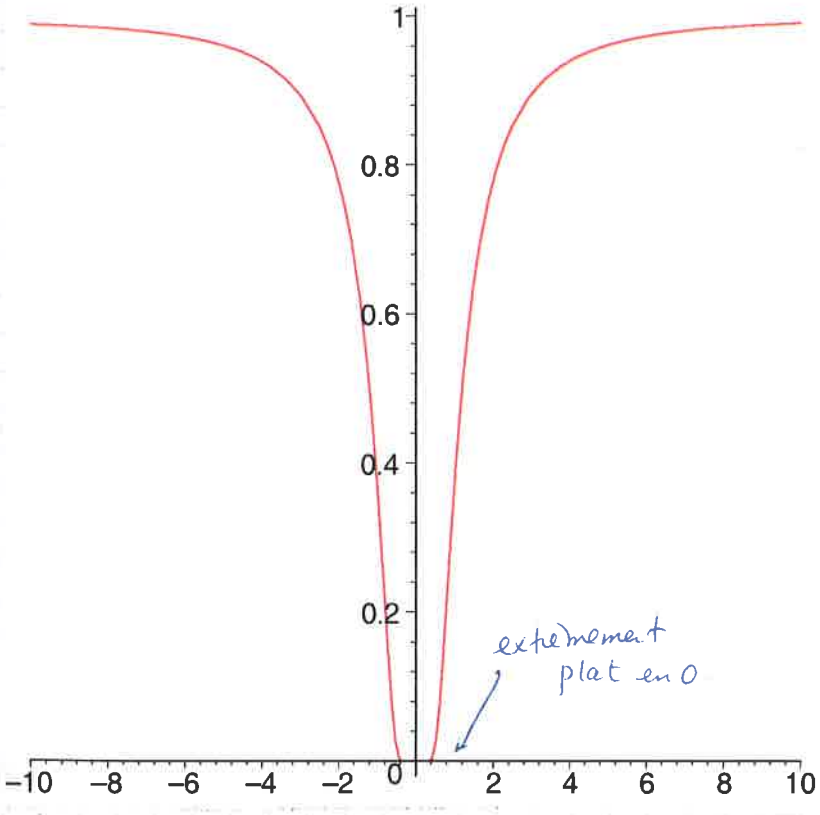
coeftayl (cos(x), x=0, 4) → $\frac{1}{24}$

Bien sur ça ne marche que pour des fonctions développables en série :

taylor ($\frac{1}{x}$, x=0, 5) → erreur

taylor (abs(x), x=0, 5) → aussi

Et il existe des curiosités, apparemment régulières :



$f := \exp - \frac{1}{x^2}$

taylor (f, x=0, 5) → erreur.

en fait cette fonction a toutes ses dérivées nulles.

h := diff (f, x) ;

lim (h, x=0) ; → 0

-h := diff (f, x $2) ;

lim (h, x=0) → 0

... pas de DL possible en 0

Singularité essentielle

* la commande asympt

Pour des fonctions qui admettent une limite à l'∞, ou qui tendent vers une asymptote, on peut faire un développement en série pour $x \to \infty$. On appelle ça un développement asymptotique.

Ex : `g := exp - 1/x^2 ;`

`asympt (g, x, 4)` → $1 - \frac{x^2}{2} + O\left(\frac{1}{x^4}\right)$

`g := x + exp - 1/x^2 ;`

`asympt (g, x, 4)` → $1 + x - \frac{1}{x^2} + O\left(\frac{1}{x^4}\right)$

`convert (%, polynom)` → $1 + x - \frac{1}{x^2}$

asympt ne marche pas pour des fonctions exponentielles ou oscillantes :

`asympt (exp(x), x, 2)` → e^x

`asympt (sin(x), x, 3)` → erreur

`asympt (sin(x)/x, 2)` → erreur

`asympt (sin(x)/x, 1)` → $O\left(\frac{1}{x}\right)$ qui s'interprète comme $0 + O\left(\frac{1}{x}\right)$ et tend vers 0 à l'∞.

* la commande series

C'est la commande générale qui fait un développement n'importe où (englobe taylor et asympt).

`series (cos(x), x=0, 5)` → identique à Taylor

`series (exp -1/x^2, x=infinity, 4)` → identique à asympt

mais aussi

`series (x^x, x = 0, 3)` → $1 + x \ln x + \frac{x^2}{2}(\ln x)^2 + O(x^3)$

5) Tracés de graphe

a) Complément sur les graphes de type $y = f(x)$

▸ pour une expression dépendant d'un seul paramètre x :

```
f := x^2 ;   plot ( f, x = -2..2 );
```

ou `plot ( f, x ) ;` → dans ce cas l'intervalle est -10..10

▸ pour une fonction (toujours d'1 seul paramètre) :

```
f := x -> x^2 ;   plot (f)          → l'intervalle est alors -20..10
                  plot ( f, -2..2 )
                  plot ( f(x), x )
                  plot ( f(x), x = -2..2 )
```

] → on est ramené ici au cas précédent f(x) est une expression

▸ Lorsque la fonction dépend d'autres variables

```
f := A cos ( 2πx / a ) ;
```

/!\ expression et pas fonction (ne marche pas pour une fonction)

pour tracer le graphe il faut donner des valeurs à A et a. Puis il faut les effacer ensuite si on veut travailler avec f.
⇒ une solution plus élégante consiste à utiliser subs.

pas bien :

```
A := 1 ;   a := 10 ;
plot ( f, x ) ;
unassign ( 'A' ) ;
unassign ( 'a' ) ;
```

bien :

```
K := { A=1, a=10 } ;
plot ( subs ( K, f ), x ) ;
```

K contient toutes les valeurs numériques

on change d'un coup toutes les constantes

▸ pour tracer une procédure :

```
f := proc (x) if x<0 then 1 else 0 fi end ;
plot ( f ) ;
plot ( f, -3..3 ) ;
```

] → s'utilise avec la notation fonctionnelle classique

`plot ( f(x), x )` → ne marche pas (il essaye d'évaluer la fn) ...

... mais `plot ( 'f(x)', x );` oui (les '' empêchent l'évaluation)

de même :

```
f := proc (x,n) if x<n then 1 else 0 fi end ;
plot ( f ) ;
```

→ ne marche pas (2 variables)

⇒ il faut utiliser `plot ( 'f(x,3)', x );`

▸ graphes log : `logplot` (axe y en log) `semilogplot` (axe x en log) `loglogplot` (les 2 axes)

b) Courbe paramétrique

Ce sont par exemple les courbes de lissajous. Les coordonnées d'un point dans le plan (x,y) sont définies par deux fonctions d'une variable t : $x(t)$ et $y(t)$.

ex : $\begin{cases} x = \cos t \text{ et } y = \sin t \\ t \in [0, 2\pi] \end{cases}$ correspond à un cercle.

↳ l'intervalle de valeurs de t est important : Si $t \in [0,\pi]$ c'est un $\frac{1}{2}$ cercle.

le système $\begin{cases} x(t) \\ y(t) \\ [t_1, t_2] \end{cases}$ s'appelle un paramétrage.

Pour tracer la courbe, 2 syntaxes :

```
plot([x, y, t1..t2])          x et y sont des fonctions...
plot([x(t), y(t), t=t1..t2])  ... ce sont cette fois des expressions
```

les autres options de la fonction plot sont disponible (couleur, ...)

Ex : `plot([cos, sin, 0..2*Pi]);` → joli cercle

`plot([cos, sin, 0..Pi]);` → $\frac{1}{2}$ cercle mais écrasé...

`plot([cos, sin, 0..Pi], scaling=CONSTRAINED);` ... repère orthonormé

Ex : Spirale de Cornu.

Elle est définie par : $\begin{cases} x(t) = \mathrm{Re}\int_0^t e^{i\pi\frac{x^2}{2}}\,dx \\ y(t) = \mathrm{Im}\int_0^t e^{i\pi\frac{x^2}{2}}\,dx \\ t \in]-\infty, +\infty[\end{cases}$

```
f:= int(exp(I*Pi*x^2/2), x=0..t);
plot([Re(f), Im(f), t=-5..5]);
```

trace la spirale de Cornu.

l'option numpoints permet le cas échéant d'augmenter le nombre de points

C) Courbes en Coordonnés polaires

Cette fois les coordonnées polaires r et θ d'un point dépendent d'un paramètre t (en physique ce sera souvent le temps)

2 syntaxes : `plot ([r, θ, t1..t2], coords = polar);`

`plot ([r(t), θ(t), t=t1..t2], coords = polar);`

ce qui est identique à la forme paramétrique, avec l'option supplémentaire `coords = polar`.

on a équivalence entre

`plot ([r, θ, t1..t2], coords = polar)` et `plot([r(t)cos θ, r sin θ(t), t=t1..t2])`

Ex : `plot ([1, t, t=0..2*Pi], coords = polar);` cercle de rayon 1

un pendule : $r(t) = cte = \ell$ longueur

$\theta(t) = \theta_0 \cos\left(\frac{2\pi t}{T}\right)$ T : période, θ_0 : amplitude

`plot ([1, cos(t), t=0..2*Pi], coords = polar)`

si la longueur s'allonge : $\ell = kt$

`plot ([t, cos(t), t=0..10*Pi], coords = polar)`

et la feuille d'érable :

`r := (100/(100 + (t - π/2)^8)) * (2 - sin(7t) - cos 30t / 2)`

`plot ([r, t, t=-Pi/2..3*Pi/2], coords = polar);`

ou encore une jolie rosace : `plot ([cos 4t, t, t=0..2π]);`

d) Courbe définie par une équation

il est possible de tracer la courbe d'une fonction $y(x)$ vérifiant $f(x,y)=0$ ou $f(x,y) =$ expression. c'est la commande **implicitplot**
pour l'utiliser il faut charger le module 'plots'

```
with(plots);        → en début de session
implicitplot(equation, h_range, v_range)
```

échanger h range ⇄ v range échange les axes.

pour tracer une ellipse : $\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$

```
implicitplot( x^2/4 + y^2 = 1, x=-2..2, y=-1..1, scaling=CONSTRAINED );
```

pour tracer une équipotentielle d'un dipôle :

au point M : $V(r,\theta) = \frac{\vec{P}\cdot\hat{r}}{r^2}\cdot\frac{1}{4\pi\varepsilon_0}$

$V = \frac{P}{4\pi\varepsilon_0}\frac{\cos\theta}{r^2}$

```
θ:= arctan(y,x); r2:= x^2+y^2;
implicitplot( cos(θ)/r2 = 1, x=-1..1, y=-1..1 );
```

on peut également tracer une famille de courbes définies par des équations eqn1, eqn2, eqn3...

ex pour tracer les équipotentielles correspondant à $V=1, 2, 3, 4$:

```
V:= cos θ/r2;
implicitplot( {V=1, V=2, V=3, V=4}, x=-1..1, y=-1..1 );
ou implicitplot( {seq(V=i, i=1..4)}, x=-1..1, y=-1..1 );
```

{} → c'est un ensemble

l'option marche aussi avec plot

e) Courbe définie par une liste de points

Commande très utile en traitement de données où on ne connait pas de forme analytique des signaux qu'on mesure, mais seulement des ~~XY~~ valeurs.

`plot ( [ [x0,y0], [x1,y1], [x3,y3]... ] )` trace le nuage de points

Exemple pour dessiner les racines n-ièmes de l'unité:

```
n:=7;                                  → racine 7e
zn:= exp 2iπp/n ;                      ← une racine
liste:= [ seq ( [Re(zn), Im(zn)], p=1..n ) ];   ↙ liste des x,y à tracer
plot ( liste, style: POINT, symbol=BOX, symbolsize=15, scaling=CONSTRAINED );
```

? sinon il relie les points

f) Superposer des graphes

La fonction display permet de superposer deux ou plusieurs graphes. Par exemple un graphe de fonction et un nuage de points.

Ex: on souhaite tracer la fonction $\cos(x^2)$ entre 0 et 10 et mettre un carré bleu sur les maxima $x_n = \sqrt{2n\pi}$.

```
with(plots);                           → display fait partie du module plots
P1:= plot ( cos(x^2), x );             ← pas d'affichage du graphe
liste:= [ seq ( [sqrt(2*n*Pi), 1], n=0..15 ) ];
```

x_n ; $y_n = 1$ (maxima) ; il y a une quinzaine de maxima entre 0 et 10

```
P2:= plot(liste, style: POINT, color= Blue);
display ( P1, P2) ;
```

⚠ Maple 10: display ne marche que si les plots sont dans le même execution group. i.e:

```
P1:= plot(      ) Shift↵
P2:= plot(      ) Shift↵
display ( { P1, P2 } )
```

g) animations

c'est la fonction **animate** qui fait partie du module plots.

Syntaxe : `animate ( f(x,t) , x = x1..x2 , t = t1..t2 , frames = n )` ← Maple 8

- `x = x1..x2` : échelle du graphe
- `t = t1..t2` : intervalle de valeurs de t pour lesquelles f(x) sera plotté
- `frames = n` : nombre de pas de t dans l'intervalle $t_1..t_2$

Maple 10 : `animate (plot, [ f(x,t) , x = x1..x2 ], t = t1..t2 , frames = n )`

Ex : `plot ( cos(2x - t), x = -10..10 , t = 0..6 , frames = 100 )`

dessine une onde

On démarre l'animation par un clic droit (menu animation).

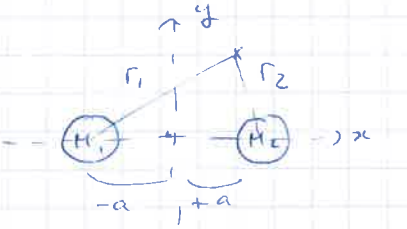
l'exemple $\cos(2x - t)$ représente une onde progressive $\cos(kx - \omega t)$ avec $k = 2$; $\omega = 1$.

Pour une onde stationnaire : $\cos(2x)\cos(t)$ en fait c'est le mélange de 2 ondes qui se propagent en sens inverse l'une de l'autre :

$$\underset{\rightarrow}{\cos(kx - \omega t)} + \underset{\leftarrow}{\cos(-kx - \omega t)} =$$

▶ Une autre façon de produire des animations est d'utiliser la fonction **display** avec l'option **insequence = true**. C'est particulièrement intéressant pour implicitplot où l'on a pas directement la fonction $f(x,t)$.

Exemple : Autour de 2 masses m_1 et m_2, quelles sont les courbes où le potentiel gravitationnel est constant ?



$$V = V_1 + V_2 = \frac{GM_1}{r_1} + \frac{GM_2}{r_2}$$

$$= \frac{GM_1}{\sqrt{(x+a)^2 + y^2}} + \frac{GM_2}{\sqrt{(x-a)^2 + y^2}}$$

l'exemple est fait avec $GM_2 = GM_1 = 1$ et $a = 1$

```
with (plots) ;
f:= 1/(sqrt((x-1)^2+y^2)) + 1/(sqrt((x+1)^2+y^2)) ;
P1:= seq (implicitplot (f=n,
                        x=-5..5, y=-5..5,
                        numpoints=10000),
                        n=(1,2,3,4,5)):
display (P1, insequence = true) ;
```

On peut exporter les animations au format gif animé (ça ouvre avec Inplayer)

III. Fonctions de plusieurs variables

1) Définition

Le principe est le même que pour les fonctions d'une variable : il y a 4 formes de définition.

- l'opérateur flèche : `f:=(x,y)->cos(x+y);`
- la commande unapply : `f:=unapply(x*x+y, x, y);`
- la commande piecewise : `f:=(x,y)->piecewise((abs(x)<1 and abs(y)<1),1,0);`
- la commande proc :

```
f:=proc(x,n)
  if (n=1) then x/2
  else 1/(1+f(x,n-1))
  fi
end;
```

(récursivité)

fonction : $\frac{x}{2}$ si $n=1$

$\frac{1}{1+\frac{x}{2}}$ si $n=2$

$\frac{1}{1+\frac{1}{1+\frac{x}{2}}}$ si $n=3$

etc... tend vers $\frac{1-\sqrt{5}}{2}$

$\forall x$, sol. de x^2-x-1

2) Opérations

a) Dérivées

* A l'aide de diff : expressions

exemple sur une fonction de 3 variables $f(x,y,z)$:

$\frac{\partial f}{\partial x}$ → `diff(f(x,y,z), x)`

$\frac{\partial f}{\partial z}$ → `diff(f(x,y,z), z)`

$\frac{\partial^2 f}{\partial y^2}$ → `diff(f(x,y,z), y$2)`

$\frac{\partial^n f}{\partial^m x\, \partial^p y\, \partial^q z}$ → `diff(f(x,y,z), x$m, y$p, z$q)`

* A l'aide de D : fonctions

pour dériver par rapport à la 1ère variable (ici x) : `D[1](f)` calcule et renvoie une fonction f'_x

2e — y : `D[2](f)`

dérivée 3e par rapport à x : `(D[1]@@3)(f)`

$\frac{\partial^5 f}{\partial^3 x\, \partial^2 y}$: `((D[1]@@3)@(D[2]@@2))(f)`

ou `D[1$3, 2$2](f);`

b) Analyse vectorielle — nécessité de charger le module **linalg**

* gradient

$\vec{E} = -\vec{\text{grad}}\ V$: fonction scalaire en entrée, vecteur en sortie.

Ex : si $V(x,y,z) = \frac{1}{r}$

```
with(linalg);
e:= grad ( 1/sqrt(x^2+y^2+z^2), [x,y,z] );
```

`1/sqrt(x^2+y^2+z^2)` : fonction ; `[x,y,z]` : liste de coordonnées

pour accéder au premier élément du vecteur : `e[1]`

On peut aussi calculer en coordonnés sphériques ou cylindriques :

ex en sphériques : `grad ( 1/r, [r,theta,phi], coords = spherical )`

cylindrique : `coords = cylindrical`

* divergence

$f = \text{div}\ \vec{E}$: un vecteur en entrée, une fonction en sortie

ex : `diverge ( [x^2,y^2,z^2], [x,y,z] );` → $2x + 2y + 2z$

`[x^2,y^2,z^2]` : 3 composants du vecteur ; `[x,y,z]` : liste des coordonnées

```
diverge ( [x,y,z], [x,y,z] );
diverge ( [r,0,0], [r,θ,φ], coords = spherical );
```

} calcul de $\text{div}\ \vec{r}$: 3

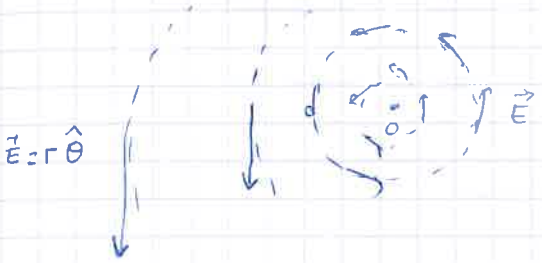
```
diverge ( [1/r^2,0,0], [r,θ,φ], coords = spherical );
```

↓ donne 0 (équation locale du champ $\vec{E}$: $\text{div}\ \vec{E} = \frac{\rho}{\varepsilon_0} = 0$ loin des sources)

Source ponctuelle : $\frac{q}{4\pi\varepsilon_0 r^2}$

* rotationnel $\vec{b} = \text{curl}\,\vec{a} = \vec{\text{rot}}\,\vec{a}$ { entrée : vecteur ; sortie : vecteur

ex : champ en $\hat{\theta}$, coordonnées cylindriques.




```
e := [0, r, 0];
v := [r, theta, phi], coords = cylindrical;
curl(e, v)
```

↓ donne $2\hat{z}$

divergence (e, v) donne 0

* laplacien $f = \Delta V$ { entrée : fonction ; sortie : vecteur

ex : $E = \exp i(kx - \omega t)$ en cartésiennes

```
e := exp(I*(k*x - omega*t));
v := [x, y, z];
laplacian(e, v)
```

→ $-k^2 e^{i(kx-\omega t)} = -k^2 E$

pour le laplacien vectoriel (n'existe pas dans Maple ≤ 8) on peut utiliser l'identité

$$\vec{\Delta} = \vec{\text{grad}}\,\text{div} - \vec{\text{rot}}\,\vec{\text{rot}}$$

c) ., développements en série

A plusieurs dimensions, le développement de taylor est possible : mtaylor.

ex : `mtaylor ( e^(x^2-y^2), [x,y], 3 ) ;` → $1 + x^2 - y^2$

(couple de variable ; ordre)

`mtaylor ( e^(x^2-y^2), [x=0, y=0], 3 ) ;`

(le point autour duquel est fait le DL)

d) Intégrale multiples

Soit l'intégrale : $\int_{x=x_1}^{x_2} \int_{y=y_1}^{y_2} f(x,y)\, dx\, dy$

le principe est de décomposer l'intégrale en 2 intégrales simples :

$$\int_{x=x_1}^{x_2} \left[\int_{y=y_1}^{y_2} f(x,y)\, dy \right] dx$$

une intégrale double sous maple se calcule en faisant deux fois appel à int

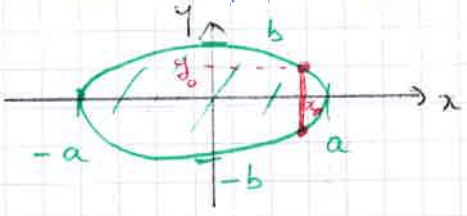
ex : `int(int( exp(-(x+y)), y=0..infinity ), x=0..infinity )`

calcule $\int_0^\infty \int_0^\infty e^{-(x+y)}\, dx\, dy$

cas d'une primitive (intégrale indéfinie)

`int(int(1/(x+y), y), x));` calcule $\iint \frac{dx\,dy}{x+y}$, donne $(x+y)(1-\ln(x+y))$

on peut aussi avoir des bornes qui dépendent de la variable. Exemple du calcul de la surface d'une ellipse :



pour un x donné, le domaine défini par l'ellipse est situé entre $-y_0$ et $+y_0$ avec $\frac{x^2}{a^2} + \frac{y_0^2}{b^2} = 1$

D'où l'aire de l'ellipse : $\int_{x=-a}^{a} \int_{y=-y_0}^{y_0} dx\,dy$

```
assume(a>0): assume(b>0):
int(int(1, y=-b*sqrt(1-x^2/a^2)..b*sqrt(1-x^2/a^2)), x=-a..a);
```

qui donne bien πab. (en fait $\pi\, a\sim b\sim$)

3) Graphes

a) Graphe d'une fonction de type $z = f(x,y)$

3 fonctions de base : `plot3d` : perspective cavalière
`densityplot` : image en fausses couleur
`contourplot` : courbes de niveau.

Ex : `plot3d` / `densityplot` / `contourplot` `(cos(x)*y^2, x=-2π..2π, y=-1..1);`

- on peut rajouter l'option `numpoints = n` ou `grid = [p, q]`
  `numpoints = n` : ~ n point calculés (grille de $\sqrt{n} \times \sqrt{n}$).
  `grid = [p, q]` : grille de $p \times q$ points
- également disponible pour les 3 fonctions : `scaling = CONSTRAINED` pour orthonormer le graphe.
- plot3d et contourplot permettent de tracer plusieurs courbes sur le même graphe :

```
[plot3d, contourplot]({x^2/10 - y^2, y^2 cos(x)}, x=-2π..2π, y=-1..1);
```

(noter que plot3d gère les parties cachées)

NB : pour contourplot il est possible d'utiliser la fonction display
pour plot3d il existe une commande display3d

ex : `g1:=plot3d(1-x^2-y^2, x=-1..1, y=-1..1); g2:=plot3d(z, x=-1..1, y=-1..1); display(g1, g2)`

▲ optins spécifiques à contour plot :

filled = true ; remplit les contours
coloring = [red, blue] ; couleurs de remplissage (min / max)

contours = n ; nombre de contours
ou
contours = [z1, z2, z3, z4, ...] ; trace les contours correspondants à f(x,y) = z1, z2 ...

▲ optins spécifiques à densityplot :

style = patchnogrid ; efface le quadrillage qui recouvre le graphe

colorstyle = [RGB / HUE] 2 palettes de couleur prédéfinies

▲ optins spécifiques à plot 3d :

axes = boxed ou frame trace un cadre autour du graphe
labels = ["xlabel", "ylabel", "zlabel"] ; label les axes (⚠ inactif par défaut utiliser avec axes)

b) Surface paramétrique - Surface définie en coordonnées sphériques ou cylindrique

la position d'un point dans le plan est définie par :

▲ trois fonctions $\begin{cases} x(u,v) \\ y(u,v) \\ z(u,v) \end{cases}$ définissent l'équation d'une surface lorsque les paramètres u et v varient

⚠ 2 paramètres sont nécessaires pour définir la surface sans ambiguité lors du changement de variable $\begin{pmatrix} u \\ v \end{pmatrix} \to \begin{pmatrix} x \\ y \end{pmatrix}$
(on peut alors se ramener à $z = F(x,y)$)

ex :

plot3d([cos(v)^3 . cos(u)^3, sin(v)^3 . cos(u)^3, sin(u)^3], u=0..2π, v=0..2π)

cos(v)^3 . cos(u)^3 = x(u,v) ; sin(v)^3 . cos(u)^3 = y(u,v) ; sin(u)^3 = z(u,v)

l'option view = [xmin..xmax, ymin..ymax, zmin..zmax] change le domaine de x, y, z (zoom).

▶ En coordonnées sphériques ou cylindriques

2 façons de faire :

la surface est définie par une équation

$r = f(\theta, \varphi)$ (Sph.)

ou

$f(\theta, z)$ (cyl.)

plot3d ($f(\theta,\varphi)$, $\theta = \theta_1 .. \theta_2$, $\varphi = \varphi_1 .. \varphi_2$, coords = spherical);

ou cylindrical pour des coords cylindriques

la surface est définie paramétriquement par 3 fonctions de u, v

$r(u,v)$ $\theta(u,v)$ $\begin{bmatrix} z \\ \varphi \end{bmatrix}(u,v)$

plot3d ($[r(u,v), \theta(u,v), z(u,v)]$, $u = u_1 .. u_2$, $v = v_1 .. v_2$, coords = cylindrical)

ex : plot3d (cos(θ)^2 * sin(2φ), θ = 0..π, φ = 0..2π, coords = spherical)

plot3d (1, θ = 0..2π, z = -1..1, coords = cylindrical)

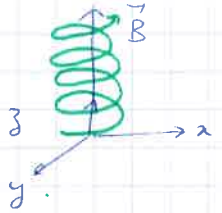
Il existe d'autres fonctions qui vont avec plot3d : implicit plot 3d, display 3d, animate 3d

c) Courbe d'un point évoluant dans l'espace.

les 3 coordonnées d'un point dépendent du temps : $\begin{cases} x(t) \\ y(t) \\ z(t) \end{cases}$

c'est une représentation paramétrique à 3d avec un seul paramètre (ne pas confondre avec la surface paramétrique définie par u et v).

ex : une particule qui spirale dans un champ $\vec{B}$:



$x(t) = r \cos t$

$y(t) = r \sin t$

$z(t) = kt$

with(plots);

SpaceCurve ([cos(6t), sin(6t), t], t = 0..30); dessine la trajectoire

ou

SpaceCurve ([1, 6t, t], t = 0..30, coords = cylindrical);

Également, on peut représenter 2 trajectoires sur le même graphe.

spacecurve ({ [cos t, sin t, t, color = red], [cos t, sin t, -t, color = blue] }, t = 0..2π);

entre { }

l'option color se met dans la 1ère liste

d) Champs de vecteurs (attention: utilisation de with(plots);)

les fonctions fieldplot et fieldplot3d permettent de tracer des champs de vecteurs dans le plan et dans l'espace.

ex: `fieldplot([x,y], x=-2..2, y=-2..2);`

composants du vecteur. ici $\vec{r} = \begin{vmatrix} x \\ y \end{vmatrix}$

ou en coordonnées polaires:

`fieldplot([r,t], r=0..2, t=0..2π, coords=polar)`

trace le champ de vecteurs $\vec{v} \begin{vmatrix} r \\ 1 \end{vmatrix}$ en polaires $= \vec{r} + \hat{\theta}$

la syntaxe de fieldplot3d est similaire.

options spécifiques: arrows = thick ou slim

fieldstrength = fixed

(pour les cas où le champ est très gd dans certaines régions, on met tous les $\vec{r}$ à la même taille)

Chap 3. Résolution d'équations

I. Une équation à une inconnue.

Il n'existe pas forcément de solution analytique simple à une équation, surtout quand elle fait intervenir des fonctions transcendantes.

La commande qui résout les équations est **solve**

Solve (eqn, var)

ex: Solve ($ax = b$, x) ; donne b/a.

Solve ($x^2 = x$, x) donne la séquence 0, 1

FWHM:
$f := x \to e^{x - x^2}$
$S := \text{solve}\left(f(x) = \frac{\partial(f)}{2}, x\right)$
$S[2] - S[1]$

on peut aussi utiliser les forms réduites

Solve ($f(x)$, x) résout alors l'équation $f(x) = 0$

Solve ($f(x)$) s'il n'y a pas d'autre constante non évaluée

solve (diff ($e^{(x-x^2)}$, x), x) donne le zéro de la dérivée : en droit où la fn est max.

▶ Il arrive des fois où Maple ne sait pas résoudre l'équation. Par exemple :

Solve ($\sin(x)/x$, x) ne renvoie rien

Dans ce cas-là il faut faire une évaluation numérique

fsolve ($\sin(x)/x$, x) donne $-\pi$ (en fait $-3.14..$)

fsolve ($\sin(x)/x$, x, $a..b$) (cherche dans l'intervalle $[a, b]$ dans le cas où il y a plusieurs sol^n)

fsolve (eqn, var, complex) cherche sur $\mathbb{C}$

/!\ sur $\mathbb{R}$ uniquement.

▶ Cas où Maple donne des solutions non explicites

ex :

```
f := x^2 - 1/sqrt(x) ;
plot ( f, x = 0..10) ;        donne une solution réelle en 1
a := solve ( f, x) ;          donne la séquence 1, RootOf(...), RootOf(...)
```

⇒ 3 solutions dont 2 sont non-explicites (il a résolu mais ne montre pas la solution).

Dans ce cas il y a plusieurs armes disponibles :

convert ([a], radical) marche pour les équations polynômiales

[ ] pour transformer en liste sinon maple croit que les , définissent des options de Convert.

`allvalues([a]);` donne aussi la forme explicite de toutes les solutions (résout les RootOf) quand c'est possible.

`evalf(a);` donne une forme numérique de la solution (ne pas confondre avec fsolve, qui cherche une solution approchée. Ici c'est bien de la solution exacte qu'on fait eval f§. Exemple de $\frac{\sin(x)}{x}$ qui ne marche pas avec solve)

Inéquations

de la même manière que pour des équations, on peut résoudre des inéquations pas trop compliquées.

ex: `solve(x^2-1>0, x);`

répond un intervalle RealRange(-∞, Open(1)), RealRange(Open(1), ∞)

`solve(x^2-1>0, {x})`

↑ la présence des {} force maple à répondre sous la forme d'un ensemble d'inégalités.

donne $\{x<-1\}, \{1<x\}$

↑ OU (union des ensembles)

`f:= x^2-1;`

`solve(f>0 and f<1, {x})`

→ donne $\left\{\frac{1-\sqrt{5}}{2}<x, x<0\right\}, \left\{1<x, x<\frac{1+\sqrt{5}}{2}\right\}$

ET (dans les {}) ; OU ; ET

noter que `solve(x^2-b^2>0, x)` ne donne rien: il faut faire des hypothèses sur b.

II - Systèmes

C'est toujours la fonction solve mais les équations et les variables sont entre { }

solve ({a+b=p, a+2b=s}, {a,b}) → plus simple pour commencer

Ex : solve ({a+b=s, a*b=p}, {a,b});

résoud le système $\begin{cases} a+b=s \\ a.b=p \end{cases}$ pour les variables a et b

⚠ faut être nécessité d'utiliser convert(...,radical)

renvoie :

$$\left\{ b = \frac{s+\sqrt{s^2-4p}}{2},\ a = \frac{s}{2} - \frac{\sqrt{s^2-4p}}{2} \right\} \text{ ET } \left\{ b = \frac{s}{2} - \frac{\sqrt{s^2-4p}}{2},\ a = \frac{s}{2} + \frac{\sqrt{s^2-4p}}{2} \right\}$$

OU

il y a 2 couples de solutions possibles.

pour récupérer les solutions dans les variables a et b :

- prenons d'abord un cas plus simple (un couple solution)

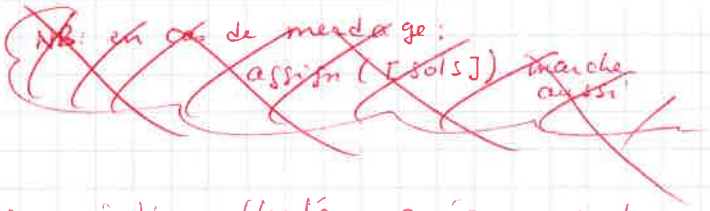
sol := solve ({a+b=s, a-b=p}, {a,b})

donne $\left\{ b = \frac{s}{2} - \frac{p}{2},\ a = \frac{s}{2} + \frac{p}{2} \right\}$: un seul couple solution.

assign (sols); → affecte aux variables leurs valeurs

a; → renvoie $\frac{s}{2} + \frac{p}{2}$

b; → renvoie $\frac{s}{2} - \frac{p}{2}$



⚠ après assign, a et b sont des variables affectées qu'on ne peut plus utiliser pour un autre solve comme variables libres.

une autre manière de faire est de passer par subs :

a1 := subs (sols, a); → a reste libre et c'est a_1 qui contient la solution.

- dans le cas de 2 couples solutions.

sol := solve ({a+b=s, a.b=p}, {a,b});

assign (sol[1]); ou a_1 := subs (sol[1], a);

{ pour le 1er couple solution.

- Si le système a plus d'inconnues que d'équations :

solve ({x+y+z=1, 2x+y=2}, {x,y,z}) → {x = 1+z, y = -2z, z = z}

↑ on conserve une des variables (z) et on exprime les 2 autres avec

• ça marche aussi avec des inéquations :

```
Solve ({x + y < 1, 2x + y = 2}, {x, y})
```

donne $\{y = -2x + 2,\ x > 1\}$

... ce qui n'est pas très intéressant dans ce cas

```
s:= Solve ({x^6 + y < 1, x·y = 2}, {x, y});
evalf(s);
```

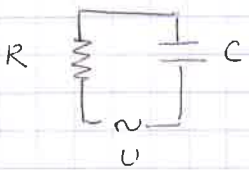
→ $\{x = \frac{2}{y},\ y < -1{,}695\ldots\}$

III - Equations différentielles : solutions

c'est la commande dsolve qui permet d

1) Cas d'une seule équation.

Exemple : le circuit RC



$$U(t) = R\,I(t) + \frac{q}{C} = R\,\dot{q} + \frac{q}{C}$$

$$\Rightarrow \dot{q} + \frac{q}{RC} = \frac{U}{R}$$ équa diff du circuit

$q(0)$ condition initiale (facultative)

Pour résoudre l'équation sans mettre la condition initiale :

```
dsolve (diff(q(t),t) + q(t)/RC = 0, q(t))
```

ou

```
dsolve (D(q)(t) + q(t)/RC = 0, q(t))
```

⇓

on obtient la forme générale de la solution : $q(t) = -C_1\, e^{-\frac{t}{RC}}$

↑ Constante d'intégration

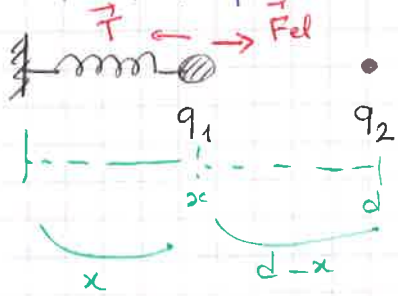
et pour tracer la solution, il faut récupérer le résultat et donner des valeurs aux constantes

```
f:= dsolve (..) ;   → donne q(t) = -C1 e^(-t/RC)
g:= rhs (f);   ← right-hand-side de l'égalité
C1:= 1 ; RC:= 1 ; plot (g, t= 0.. 5);
```

Maple 10 : pour faire le _ : taper _ (Alt-gr 8 et 8)

dans Mes Documents \ charressat.mws

Exemple physique de solve :



$\vec{T} = -kx\,\hat{x}$; $\vec{F}_{el} = \frac{1}{4\pi\varepsilon_0}\frac{q_1 q_2}{(d-x)^2}$

$\Rightarrow kx = \frac{1}{4\pi\varepsilon_0}\frac{q_1 q_2}{(d-x)^2}$

(seule la 1ère sol. est réelle)

$q_1 = 10^{-6}$; $q_2 = +10^{-6}$
$d = 10$ cm $\Rightarrow$ équilibre fausse.

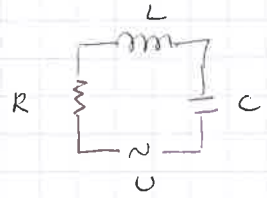
* Avec une condition initiale $q_0 = q(0)$:

dsolve ({ D(q)(t) + q(t)/RC = 0, q(0) = 1 }, q(t))

↑ le système d'équations est mis entre { } et est ...

* Equation du second ordre :

C'est le cas du circuit RLC :



$$U = \frac{q}{C} + L\dot{I} + RI$$

$$= \frac{q}{C} + L\ddot{q} + R\dot{q}$$

$$\Rightarrow \ddot{q} + \frac{R}{L}\dot{q} + \frac{q}{LC} = \frac{U}{L}$$ éq^n du 2e ordre.

$\dot{q}(0) = a$, $q(0) = b$ Cond. initiale.

Syntaxe complète (avec les CI)

eq := diff(q(t),t,t) + R/L * diff(q(t),t) + q(t)/(LC) = 0 ; (éq. homogène)

dsolve ({eq, q(0) = b, D(q)(0) = a }, q(t))

↓ la condition initiale sur $\dot{q}$ se rentre à l'aide de D.

2) Systèmes d'équations différentielles

On s'intéresse à l'exemple :

Dans l'approxim les équa diff

$$\begin{cases} \ddot{\theta}_1 + \frac{g}{l_1}\theta_1 + \\ \ddot{\theta}_2 + \frac{g}{l_2}\theta_2 + \end{cases}$$

$$\begin{cases} \dot{x} + k_1 x + k_2 y = 0 \\ \dot{y} + k_3 x = 0 \end{cases}$$

avec les CI $x(0) = 1$, $y(0) = 0$

avantage : 1 seule sol^n.

et plot avec $\begin{cases} k_1 = k_3 = 1 \\ k = -2 \end{cases}$.

la syntaxe utilisée pour résoudre un système d'e. d. complès est :

dsolve ({eqs}, {variable}) ou

dsolve ({eqs, ci}, {variable})

↑ séquence des 2 équations ↑ séquence des 4 C.I.

avec ici les variables $\{\theta_1(t), \theta_2(t)\}$
et les deux équations du système $\{eq1, eq2\}$.
Pour les CI : on en a 2 par variable puisque chaque équation est du second ordre $\Rightarrow$ 4 CI : $\theta_1(0)$ $\theta_2(0)$
$D(\theta_1)(0)$ $D(\theta_2)(0)$

$\Rightarrow$ faire tourner l'exemple pendcoupl. mws.

d'abord avec $\begin{cases} m_2 \ll m_1 \rightarrow \text{peu d'influence du pendule 2} \\ l_2 \ll l_1 \rightarrow \text{il va beaucoup plus vite} \end{cases}$

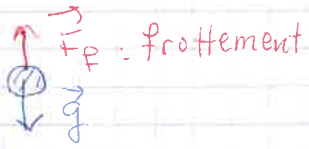
et observer aussi l'espace de phase.

putain j'ai le plâtre c'est la merde !
Tout s'est bien passé

IV - Solutions numériques d'équa. diff

1) Une seule équation, c'

Soit une masse m que l'on jette dans un fluide visqueux.



Souvent on modélise la force de frottement par un terme proportionnel à la vitesse $\vec{F}_f = -k\vec{v}$

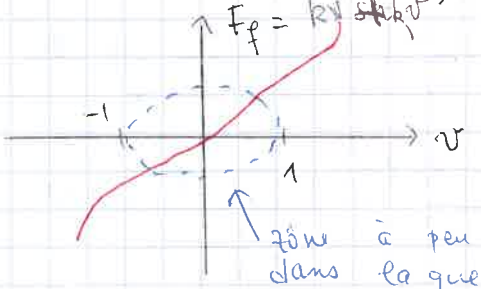
Du coup l'équation du mvt s'écrit, en utilisant la vitesse :

$$\vec{F}_f + m\vec{g} = m\vec{\dot{v}} \Rightarrow m\frac{dv}{dt} + kv - mg = 0$$

équa. diff linéaire en v et $\dot{v}$

Bien souvent on ne sait résoudre que les équa. dif. linéaires. Pour les éq. plus compliquées on doit avoir recours à une résolution numérique.

Par exemple si la force de frottement est en ~~kv~~ $kv + k_1 v^3$ presque linéaire si $v(t) \ll 1$ et de moins en moins quand $v \nearrow$



zone à peu près linéaire dans laquelle on a une solution exacte de l'équa. diff.

solution linéaire :

```
eq := m. diff(v(t), t) + k. v(t) - m g = 0 ;
ci := v(0) = 0 ;
dsolve ({eq, ci}, v(t))
```

donne $v(t) = \frac{gm}{k} - \frac{gm}{k} e^{-\frac{k}{m}t}$

Si maintenant on rajoute le terme en v^3 :

```
eq:= m diff(v(t),t) + k v(t) - k1 v(t)^3 - m g = 0;
solve ({eq, ci}, v(t))
```

donne $v(t) = \text{Root of}\left[t + \int_{0}^{-\_Z} \frac{m}{-ak - k_1 a^3 + mg}\, da\right]$

pas de solution explicite, le Root of ne donne rien de concret, il n'arrive pas à tracer un graphe

⇒ On doit demander une résolution numérique, en utilisant toujours dsolve mais avec l'option type = numeric

⚠ Toutes les constantes doivent être numériques avant l'appel à dsolve. Il faut aussi fournir les conditions initiales

```
m:=1; k:=1; k1:= 0.1; g:= 10;
eq:= ... ;
sol:= dsolve ( {eq, ci}, type = numeric );
```

réponse de maple : sol:= proc (x_rkf45) ... end proc
Il a bâti une procédure dans laquelle se trouve le programme qui calcule les valeurs de la fonction.

Ainsi sol (0.1) donne [t=0.1, v(t) = 0,949...]

voir complément ode plot à la fin

Pour récupérer le résultat en utilisant comme premier argument ea existe sol() et comme second v(t). Il remplacera alors v(t) par ce qu'il y a derrière l'égalité dans l'expression de sol.

On pourrait vouloir faire : soln := subs (sol, v(t)) mais ça ne marche pas car sol est une fonction et il faut l'appeler avec des arguments => on lui met un argument bidon, u :

```
soln := u -> subs (sol (u), v(t))
```

u : argument bidon ... ; v(t) : qui ne peut pas être t : c'est déjà pris !

l'évaluation de l'expression avec le → se fait au moment du calcul numérique donc ça marche. Avec la commande unapply qui évalue de suite, ça ne marche pas.

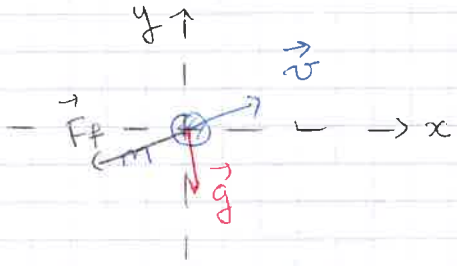
Pour tracer : `plot (soln, 0..10)`

ou : `plot (soln(t)', t = 0..10)`

présence du ' pour retarder l'évaluation.

2) Système d'équations

On considère le même problème physique mais en 2 d :



$$\vec{P} = -mg\,\hat{y}$$

$$\vec{F_f} = -k\,\|v\|\,\hat{v} - k_1\,\|v\|^3\,\hat{v}$$

Ce qui donne pour les éq. du mvt :

$$\begin{cases} m\ddot{x} = -2k\left(\sqrt{\dot{x}^2(t)+\dot{y}^2(t)}\right)^{1}\dfrac{dx}{dt} - 3k_1\left(\sqrt{\dot{x}^2(t)+\dot{y}^2(t)}\right)^{3}\dfrac{dx}{dt} \\ m\ddot{y} = -mg - \dfrac{dy}{dt}\left(k\ldots\right) \end{cases}$$

la Force totale s'écrit $\vec{F} = -k\vec{v} - k_1 v^2 \vec{v} - mg\hat{y}$

$$\text{Sur } x : \quad F_x = -k\dot{x} - k_1\left(\dot{x}^2 + \dot{y}^2\right)\dot{x} = m\ddot{x}$$

$$\text{Sur } y : \quad F_y = -k\dot{y} - k_1\left(\dot{x}^2 + \dot{y}^2\right)\dot{y} - mg = m\ddot{y}$$

} système différentiel non linéaire.

couplage

Sous maple, la résolution exacte ne donne rien. On est obligé de résoudre numériquement. On tape donc :

```
eq1 := 
eq2 := 
ci := x(0)=0, y(0)=0, D(x)(0)=1, D(y)(0)=0 ;
inc := {x(t), y(t)}
sol := dsolve({eq1, eq2, ci}, inc, type = numeric);
```

puis pour extraire les solutions :

```
xx := u -> subs(sol(u), x(t)) ;
yy := u -> subs(sol(u), y(t)) ;
```

Et pour tracer le graphe en paramétriques :

```
plot(['xx(t)', 'yy(t)', t = 0..5]);
```

Plus simple :

$$\begin{cases} \dot{x} + xy = 0 \\ x\dot{y} + 2y = 1 \end{cases}$$

2) Chap 4 - Programmation

I - Les boucles for

Syntaxe :

for <variable> from <deb> to <fin> (by <pas>)
do <instr> od
(ou end do)

Exemple : remplir un tableau de 10 éléments :

S := [0 $ 10] → on crée une liste de 10 zéros

for i from 1 to 10 do S[i] := i^2 od
→ on remplit la liste avec i^2

⚠ Nécessité de faire shift-return pour passer à une ligne à l'autre

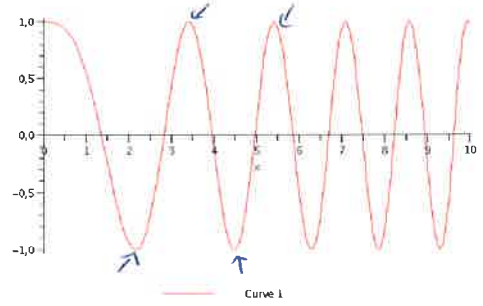
with(plots) :

listplot(S, style = point) → trace la liste

≡ plot([[i, i^2] $ i = 1..10], style = point)

2e exemple : Soit la fonction $f(x) = \cos(x^{3/2})$

plot entre 0 et 10 :



On s'aperçoit que les posit° de max/min ne sont pas régulières :
- 1 min entre 2 et 3
- 1 max 3 et 4
- 4 et 5
- :

⇒ on va chercher les zéros de la dérivée dans tous les intervalles de type i .. i+1 et les mettre dans une liste S :

S := [0 $ 10] → on crée un tableau vide

for i from 1 to 10 do S[i] := fsolve(diff(f(x), x) = 0, x = i .. i+1) od

S
↳ renvoie une liste avec les zéros de la dérivée. Le 1er élément n'est pas numérique car pas de sol°
On peut l'enlever en faisant S := S[2..10]
OU - seq(S[i], i = 2..10)

II - Les boucles while

Syntaxe : while <condition> do <commandes> end do

Les conditions s'écrivent au moyen d'opérateurs de comparaison :

=, >, <, <=, >=, <>

et d'opérateurs booléens : and, or, not, xor

Exemple d'utilisation : on veut ~~ajouter~~ créer des nombres aléatoires, les placer dans une liste et s'arrêter quand leur somme est > 100

```
l := 0          ← on démarre la liste aléatoire par 0
S := 0          → la somme est mise à 0 au début de boucle
while (S < 100) do
    x := rand(1..10)() :   → on génère le nb aléat x ∈ [0,10]
    l := l, x :            → on le met à la suite des autres
    S := S + x :           → on somme à la somme partielle
od
```

on peut ensuite avoir le nb d'éléments de l par nops(l)

III - Les Conditions

Syntaxe : if <condition> then <instr> else <instr2> fi

ou

```
if <cond1> then <inst1>
elif <Cond2> then <inst2>
elif <...> then <...>
else <inst par défaut>
fi
```

Exemple : on considère un tableau de 50 nombres aléatoires. dedans, on compte ceux dont le reste de la division par 3 donne 0, 1, ou 2.

.../...

```
x := [seq(rand(1..100)(), i=1..50)]:

r0 := 0
r1 := 0          } (= 3 tableaux : reste = 0, 1 ou 2
r2 := 0

for k from 1 to nops(x) do
  r := irem(x[k], 3):
  if (r=0) then r0 := r0+1:
  elif (r=1) then r1 := r1+1:
  else r2 := r2+1:
  fi:
od
```

IV Procédures

On peut définir des fonctions évaluées avec maple.

- ex : `f := proc(x)`

  `trunc(x)^frac(x)` → fonction qui calcule la partie entière puissance la partie fractionnaire

  `end`

On peut ensuite la tracer ou la dériver ou la manipuler comme n'importe quelle fonction.

- Autre exemple : `g := proc(x) if (x<0) then sin(x) else cos(x) fi end`

  dans ce cas pour tracer la fonction il convient de mettre des quotes pour retarder l'évaluation

  `plot('g(t)', t=-10..10)`

  De même pour dériver : `h := D(g)` et pas diff.

  pour l'intégration seule l'intégration numérique est possible :

  `evalf(Int('g(t)', t=-1..1))`

  (`Int` : forme inerte pour retarder le calcul ; `'g(t)'` : quotes)

Certaines fonctions ne marchent pas bien, par exemple Taylor. Dans ce cas particulier il vaut mieux utiliser piecewise

- Cas de plusieurs arguments :

```
f := proc(x, n)
    if (x<0) then x
    else x^n
    fi:
end
```

- Variables locales et utilisation de return :

Soit à créer une fonction qui prend { un réel x / un entier >0 } en entrée et qui renvoie en sortie une liste contenant les x^k pour k allant de 0 à n.

```
f := proc(x,n)
local tablo :        ~> variable locale n'existe que comme interne d'aide de calcul dans la fn.
if n<=0 then 0       ~> valeur retournée si n<=0
else
    tablo := [seq(x^k, k=0..n)]:
    RETURN (tablo) :  ~> valeur retournée si n>0
end
```

* Complément sur les équas diffs (solutions numériques)

après avoir fait un dsolve (, numeric) on peut utiliser ode plot pour tracer le graphe sans forcément isoler la solution dans une fonction.

Exemple :

```
eq := diff(v(t),t) + v(t)·k - m·g - k1 v(t)^3 = 0
ci := v(0) = 0
sol := dsolve({eq, ci}, v(t), numeric, [range = 0..0.6])
with(plots)
odeplot(sol)
```

argument optionnel

↳ trace la solution v(t) pour le range spécifié dans dsolve

ou bien :

```
odeplot(sol, [t1..t2])
```

↑ trace entre t_1 et t_2

```
odeplot(sol, frames = 10)
```

→ anime la solution en faisant varier t

→ cas de systèmes :

```
eq2 := diff(x(t),t) = v(t)
```

eq1 : la même que précédemment

```
ci := x(0) = 1, v(0) = 0
sol := dsolve({eq, eq2, ci}, numeric)
odeplot(sol)
odeplot(sol, [t, x(t)])
```

→ trace x en fonction de t

$[v(t), x(t)]$ → pour le portrait de phase

$[t, x(t), v(t)]$ → un graphe en 3D (trajectoire du point dans l'espace des phases)