Introduction à Maxima

Ozan BAKIS*

Table des matières

*Je voudrais remercier tous les volontaires qui ont contribué à Maxima.
e-mail : ozanbakis@gmail.com

1 Introduction

L'objectif de ce cours est de montrer comment, on peut utiliser l'ordinateur pour vérifier nos résultats, et résoudre des problèmes typiques rencontrés en macroéconomie et microéconomie. Nous allons introduire Maxima pour le calcul formel (symbolique). Pour le calcul numérique Matlab [1] (un logiciel propriétaire) ou GNU Octave [2] (l'équivalent de Matlab mais un "logiciel libre") est mieux adapté. Pour vous donner une idée,

$$ax + b = c \Rightarrow x = \frac{c - b}{a}$$

est un calcul formel, alors que

$$2x + 5 = 1 \Rightarrow x = \frac{1 - 5}{2} = -2$$

est un calcul numérique. L'objectif de ce cours est le calcul formel.

Une fausse croyance serait de penser que l'ordinateur a toutes les réponses chez lui quelque part. C'est faux. Si vous ne métrisez pas le processus, si vous ne savez pas où chercher la réponse, l'ordinateur ne sera aucune d'utilité pour vous. En revanche, si vous savez ce dont vous avez besoin l'ordinateur *peut* être utile pour vous, –mais pas toujours. La raison est simple : d'un côté, il y a des problèmes dont il n'existe pas de solution.

1. www.mathworks.com/
2. www.gnu.org/software/octave/

De l'autre, les ordinateurs ont une capacité qui est bornée par les avancées technologiques et scientifique.

Une condition nécessaire pour utiliser l'ordinateur est de pouvoir communiquer avec lui ! A l'aide de Maxima, nous essayons de voir comment utiliser l'ordinateur pour résoudre des problèmes économiques.

Maxima est un logiciel libre. Cela signifie que non seulement, on ne paye pas pour l'utiliser, mais on a accès à son code que l'on peut modifier à son égard. Pour savoir plus sur les logiciels libres visitez le projet GNU à l'adresse : `http://www.gnu.org/`.

Il existe plusieurs manuels compréhensifs et de bonne qualite sur le site de maxima, `http://maxima.sourceforge.net/documentation.html`. Je vous conseille particulièrement "Maxima by example" de Edwin Woolet téléchargeable sur son site `http://www.csulb.edu/~woollett/`.

1.1 Comment télécharger et installer Maxima

Le site internet de maxima,`http://maxima.sourceforge.net/`, fournit tous les informations nécessaires pour installer et utiliser Maxima. Mais très brièvement : pour télécharger, allez sur le site `http://sourceforge.net/project/showfiles.php?group$_$id=4933` et télécharger la version Windows. Après exécutez le fichier et suivez les étapes vous prposées. Acceptez ce qui vous est proposé. Les paramètres par défaut suffiront pour notre cours.

1.2 Comment lancer et arréter Maxima

Pour lancer maxima vous avez, au moins, 5 choix : XMaxima, wxMaxima, avec Emacs (emaxima), avec winTexMacs, à partir du console. Nous allons utiliser les deux premiers : XMaxima et wxMaxima. Pour lancer, utilisez l'icone sur votre bureau ou Start → All programs →Maxima → Xmaxima (ou wxMaxima)

Pour terminer une session la commande est `quit();`

1.3 Comment chercher une fonction

Il y trois possibilités :
- describe(solve) ;
- ? solve ;
- ? ? sol ;

Les deux premiers options peuvent êtres utilisées quand le mot que l'on veut chercher est entré exactement. Le dernier choix est destiné pour les cas où l'on n'est pas sûr du nom de ce que l'on cherche. Faites attention à " ; " après chaque commande.

2 Expressions

L'unité de base de maxima est expression. Une expression peut être un opérateur, un nombre, une variable une constante ou une combinaison de tous ces éléments.

2.1 Opérateurs

Les opérateurs $+, -, *$ et / sont utilisés pour l'addition, soustraction, multiplication et division.ˆest utilisé pour la puissance. Par exemple, `x^2+3` veut dire $x^2 + 3$ pour maxima.

L'ordre des opérations est d'abord $*, /$ et après $+, -$. Les parenthèses peuvent être utilisés pour le changer. La priorité sera donnée aux parenthèses.

Le tableau suivant est repris de Ch.1 p.18 de E. Woollett.

```
+       addition
-       subtraction
*       multiplication
/       division
-       negation
^       exponentiation
.       non-commutative multiplication
^^      non-commutative exponentiation
!       factorial
!!      double factorial
```

2.2 Variables

Les variables sont des lettres/mots auxquels nous pouvons attribuer des valeurs. Une variable doit commencer par une lettre ; après on est libre d'utiliser soit des lettres, soit des nombres, soit le tiret bas, "_". Les noms suivants ne peuvent pas être utilisés comme nom de variable, parce qu'ils sont des mots-clés de maxima : **integrate next from diff in at limit sum for and elseif then else do or if unless product while thru step**.

Pour attribuer une valeur à une variable tapez

```
a:3;
b:a+5;
```

Comme vous le voyez, pour attribuer le signe deux-points " : ". Pour annuler les valeurs que nous avons attribuées et libérer a, b la commande est

```
remvalue(a,b);
```

Pour libérer toutes les variables

```
remvalue(all);
```

On peut attribuer une variable non-connue à une autre variable. Pour commencer, on peut recommencer de zéro : on supprime TOUT ce qui est crée par l'utilisateur par la commande

```
kill(all);
a:k-m;
b:k+p;
c:a*b;
```

Le résultat affiché est $(k - m)(p + k)$. NOus pouvons différencier c par rapport a k par la commande

```
cprime:diff(c,k);
```

ce qui va donner $p - m + 2k$. Nous pouvons même résoudre cette dérivé pour une valeur particulière, disons 4,

```
solve(cprime= 4,k);
```

La réponse est

$$\left[k = -\frac{p - m - 4}{2}\right]$$

Etant donné une expression $e : f(x)/g(x)$, on peut assigner/appeler/modifier le numérateur par la commande `num(e)` et le dénominateur par `denom(e)`. Par exemple,

```
e: (x+5)/(x-2);
```

$$\frac{x + 5}{x - 2}$$

```
e1: num(e); e2:denom(e);
solve(e1=0,x); solve(e2=0,x);
```

$$[x = -5] \quad [x = 2]$$

2.3 Nombres

Maxima est capable d'utiliser

– les nombres réels :

- les entiers : -2,+5,0, . . .
- les nombre rationnels, i.e. les nombres que l'on peut représenter par des fractions des entiers : 3/4 mais aussi 8/4=2
- les nombres irrationnels, i.e. les nombres qui ne sont pas rationnel, c'est-à-dire que l'on ne peut pas les écrire en fraction des entiers : $\sqrt{2}$

– les nombres complexes : $4 + 3i$, qui est noté comme `4+3*%i` dans maxima

Si on définit

```
a:5/3;
```

et après, si l'on demande

```
a;
```

on obtiendra toujours 5/3. Pour voir la valeur numérique de a – qui est 1.66667 –, il faut utiliser la commande `numer`

```
a, numer;
```

2.4 Constantes

Il s'agit des constantes comme π, noté

`%pi` : la constante $\pi \cong 22/7$

`%e` : la constante $e \cong 2.71828$

`%i` : $i^2 = -1$

`%gamma` : .5772156649

`inf` : ∞

`minf` : $-\infty$

Les mots réservés : le tableau suivant est repris de Ch.1 p.19 de E. Woollett.

```
af    else   ic2       plog
and   elseif if        psi
av    erf    ift       product
args  ev     ilt       put
array exp    in        rat
at    f90    ind       rem
bc2   fft    inf       rk
carg  fib     inrt      some
cf    fix     integrate step
cint  for    is        sum
col   from   li        then
cov   gcd    limit     thru
```

```
cv     gd      min        und
del    get     next       unless
diag   go      not        vers
diff   hav     op         while
do     ic1     or         zeta
```

2.5 Fonctions

Une fonction est définie grâce à l'opérateur ":=" de manière suivante :

```
f(x):=x*(1-x);
f(2);
```

$$-2$$

Comme les variables, le premier caractère d'une fonction doit être une lettre.

Nous allons définir la fonctions $f(x) = x^3 + 2 * x^2 + x - 1$ et après évaluer $f(x)$ et $f'(x)$ pour des valeurs arbiraires de x, e.g. $x = 3, 4, 7, \ldots$

```
f:  x^3 + 2*x^2 + x - 1;
df: diff(f,x);
subst([x=3],f);
subst([x=3],df);
```

$$x^3 + 2\,x^2 + x - 1$$

$$3\,x^2 + 4\,x + 1$$

...

```
expr : x^3 + 2*x^2 + x - 1;
define(g(x), expr);
define(dg(x), diff(g(x),x));
g(3);
dg(3);
```

$$x^3 + 2\,x^2 + x - 1$$

$$g\,(x) := x^3 + 2\,x^2 + x - 1$$

$$\mathrm{dg}\,(x) := 3\,x^2 + 4\,x + 1$$

...

Remarque : L'approche suivante ne marche pas !

```
f(x):=  x^3 + 2*x^2 + x - 1;
df(x):= diff(f(x),x);
f(3);
df(3);
```

Plus particluièrement, on obtient l'erreur suivante :

```
diff: second argument must be a variable; found 3
#0: df(x=3)
 -- an error. To debug this try: debugmode(true);
```

Mais celles-ci marchent

```
f(x):=  x^3 + 2*x^2 + x - 1;
df(x):= diff(f(x),x);
at(f(x), x=3);
at(diff(f(x),x), x=3);
```

...

```
f(x):=  x^3 + 2*x^2 + x - 1;
df(x):= diff(f(x),x);
y : df(x);
f(3);
y, x=3;
y, x=7;
```

...

2.6 Listes

Le terme **list** désigne un ensemble ordonné dans Maxima. Un élément d'une liste peut être n'importe quelle expression de Maxima : un nombre, un mot, variable....

```
maliste: [a,b,c,4,abc];
```

Pour ajouter un élément à la fin d'une liste :

```
endcons(11, maliste);
```
Pour ajouter un élément au début d'une liste :
```
cons(22, maliste);
```
Mais ces deux commandes ne modifient pas la liste enregistrée. Pour le voir :
```
maliste;
```
$$[a, b, c, 4, abc]$$

Pour modifier la liste, il faut faire :
```
maliste2: cons(22, maliste);
```
$$[22, a, b, c, 4, abc]$$

Pour ajouter deux éléments, on essaie :
```
cons(11, 22, maliste);
```
On reçoit une erreur ; parce que pour une liste qui a plus d'un élément, il faut que les deux soit des listes.
```
append(maliste2, maliste);
```
$$[22, a, b, c, 4, abc, a, b, c, 4, abc]$$

Pour créer une liste à partir d'une expression, il existe deux commandes. La première est **makelist(expr, i, $\mathbf{i_0}$, $\mathbf{i_T}$)**. i est la variable muette i_0 est la valeur initiale de i et i_T celle finale. **expr** doit être une expression en i comme `i*4-5`.
```
makelist(x*x, x, 1,10);
```
$$[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]$$

Et la deuxième est **makelist(expr, i, maliste**. De manière similaire, chaque élément de la liste **maliste** va être transformé selon l'expression que nous allons préciser.
```
makelist(x*x, x, maliste);
```
$$\left[a^2, b^2, c^2, 16, abc^2\right]$$

Ou encore, on peut utiliser une fonction nommée
```
f(x):=x*x;
makelist(f(x),x, maliste);
```
$$\left[a^2, b^2, c^2, 16, abc^2\right]$$

Remarque : Si l'on avait essayé `f:=x*x;`, cela n'aurait pas marché. On verra les raisons dans les sections suivantes.

Une méthode alternative est d'utiliser la commande **map** qui marche avec une fonction que nous allons préciser.

```
f(x):=x*x;
map(f, maliste);
```

$$\left[a^2, b^2, c^2, 16, abc^2\right]$$

2.7 Fichiers de données

Avant d'étudier comment nous pouvons écrire nos résultats, nos données sur un fichier, nous devrons déterminer – et si besoin, changer – les répertoires par défaut. Le répertoire assigné à **maxima_userdir;** est le domaine de travail. Quand on crée un fichier, il sera crée dans ce répertoire. Il vaut mieux l'adapter à son organisation de travail. Le répertoire désigné par **maxima_tempdir** est utilisé pour stocker les fichiers graphiques crées par Maxima. Pour les recherches le répertoire assigné à **file_search_maxima** est utilisé. Les valeurs par défaut sont :

```
maxima_userdir;
maxima_tempdir;
file_search_maxima;
```

Une fois que nous avons déterminé ces répertoires, on peut créer un fichier (s'il déja n'existe pas) qui s'apelle **maxima-init.mac** et qui contient des expressions qui changent les valeurs par défaut. Par exemple le mien ressemble à :

```
maxima_userdir:"g:/maxima"$
maxima_tempdir :"g:/maxima/temp"$
file_search_maxima:append(file_search_maxima,["g:/maxima/###.mac,max,lisp"])$
file_search_usage:append(file_search_usage,["g:/maxima/###.mac,max,lisp"])$
load("mactex-utilities");
```

Il est possible de lire et écrire des fichiers sans changer les répertoires par défauts. Dans ce cas on est obligé d'utiliser "". Supposons que nous voulons écrire une liste `maliste3` dans un fichier appelé `dene.dat`. D'abord on crée notre liste `maliste3: makelist(x*x,x,0,9);`

$$[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]$$

Après on utilise la commande `write_data` pour l'écrire. Faites attention comment nous avons indiqué notre répertoire préféré où l'on a écrit notre fichier.

```
write_data(maliste3,"g:/maxima/dene.dat");
```

La valeur par défaut pour séparer les éléments de la liste est un espace. Pour utiliser la virgule comme séparateur il faudrait que le fichier termine avec **.csv**, i.e. **dene.csv**. Ou encore :

```
write_data(maliste3,"g:/maxima/dene.dat",comma);
```

D'autres séparateurs existent : pipe (|), semicolon (;)

Pour ajouter des nouvelles données à un fichier existant la commande est (la valeur par défaut est de le remplacer) :

```
write_data(maliste3,"g:/maxima/dene.dat"), file_output_append:true;
```

Pour lire à partir d'un fichier, et assigner ces valeurs à une nouvelle liste qui s'appelle maliste4 la commande est

```
maliste4:read_list("g:/maxima/dene.dat");
```

Cette commande lit comme les données comme une liste ordinaire. On peut aussi utiliser la commande suivante pour lire les données et former une matrice, disons la matrice **A**.

```
A:read_matrix("g:/maxima/dene.dat");
```

Ici, chaque ligne sera la ligne de la matrice **A**.

```
A;
```

$$\begin{pmatrix} 0 & 1 & 4 & 9 & 16 & 25 & 36 & 49 & 64 & 81 \\ 0 & 1 & 4 & 9 & 16 & 25 & 36 & 49 & 64 & 81 \end{pmatrix}$$

Pour obtenir des matrices à plusieurs ligne, il faudrait que la liste ait la forme :

```
maliste4:[[0,1,2],[3,4,5],[6,7,8]];
write_data(maliste4,"g:/maxima/dene2.csv");
B:read_matrix("g:/maxima/dene2.csv");
```

$$\begin{pmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{pmatrix}$$

3 Algèbre

L'algèbre [3] est la branche des mathématiques qui étudie les structures algébriques. On étudie ce que c'est un pôlynome, une fonction, un inconnu, un nombre rationnel, un ensemble et les manipulations autour de ces termes.

L'analyse a pour point de départ le calcul infinitésimal. C'est la branche des mathématiques qui traite explicitement des notions de limite, continuité, dérivation et intégration.

3. Toutes les définitions sont obtenues de wikipedia [4]

Dans cette section, nous allons voir comment manipuler les expressions algèbriques.

3.1 Expansion

expand permet de développer une expression à coefficients entiers. Pour les pôlynomes la version **ratexpand** est conseillée.

```
a: (x+2+y)^3;
expand(a);
```

$$y^3 + 3\,x\,y^2 + 6\,y^2 + 3\,x^2\,y + 12\,x\,y + 12\,y + x^3 + 6\,x^2 + 12\,x + 8$$

Nous pouvons utiliser **mainvar** pour modifier la manière dont le résultat de **expand** s'affiche.

```
(declare(x,mainvar),expand(a));
```

$$x^3 + 3\,y\,x^2 + 6\,x^2 + 3\,y^2\,x + 12\,y\,x + 12\,x + y^3 + 6\,y^2 + 12\,y + 8$$

logexpand est la version pour les logs. Si l'on a $\log(a^b)$, la commande **logexpand** va le transformer en $b \log a$. Pour développer l'expression en log, il faut fixer **logexpand** à **all** (La valeur par défaut est **true**).

```
logexpand; log(a*b);
```

$\rightarrow \log(a\,b)$

Alors que

```
logexpand:all; log(a*b);
```

$\rightarrow \log b + \log a$

radexpand est la version utilisée pour les nombres rationnels. Quand cette option est fixée comme **all**, l'expression `sqrt(x^2)` sera simplifiée comme x. La valeur par défaut est **true** pour **radexpand** ; cequi fait que `sqrt(x^2)` sera simplifiée comme `abs(x)`, i.e. $|x|$. Changer l'option de **radexpand** comme **all** revient à supposer $x > 0$.

```
radexpand; sqrt(x^y); sqrt(x*y);
```

$\rightarrow \sqrt{x^y},\ \sqrt{x\,y}$

Alors que

```
radexpand:all; sqrt(x^y); sqrt(x*y);
```

$\rightarrow x^{y/2},\ \sqrt{x}\,\sqrt{y}$

Pour les pôlynomes la version **ratexpand** est conseillée. Elle est plus rapide et efficace par rapport à **expand**.

```
e:(x-1)/(x+1)^2 + 1/(x-1);
```

$$\frac{x-1}{(x+1)^2} + \frac{1}{x-1}$$

```
expand(e);
```

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

```
ratexpand(e);
```

$$\frac{2\,x^2}{x^3+x^2-x-1}+\frac{2}{x^3+x^2-x-1}$$

multthru est la commande qui multiplie les éléments d'une expression (sous forme d'une somme) par d'autres éléments de la même expression ou par une expression donnée, si c'est précisé explicitement.

```
multthru((a+b)/c); multthru((a+b)*c);
```

$$\frac{b}{c}+\frac{a}{c},\quad b\,c+a\,c$$

```
multthru(x,(a+b)/c);
```

$$\frac{(b+a)\;x}{c}$$

```
multthru(x*y*(a+b)/c);
```

$$\frac{b\,x\,y}{c}+\frac{a\,x\,y}{c}$$

Considérer l'expression

$$\frac{y^3+x\,y}{y^2}$$

où il y a plusieurs fois y à des exposants différents. **multthru** peut être utilisé pour simplifier :

```
multthru((x*y+y^3)/y^2);
```

$$y+\frac{x}{y}$$

```
eq1: x=2/(y-5);
```

$$x=\frac{2}{y-5}$$

```
multthru((y-5),eq1);
```

$$x\;(y-5)=2$$

distrib est une commande similaire à **expand**, mais elle ne développe pas entièrement ; elle s'arrête au niveau principal.

La commande **partfrac** va réaliser une décomposition en fractions partielles (ou en éléments simples) d'une fonction rationnelle[5]. Ceci consiste

5. $f(x)=\frac{P(x)}{Q(x)}$

FIGURE 1 – Comparaison entre multthru, expand et distrib

à – d'après la définition de Wikipédia – *reformuler la même expression sous une somme de fractions ayant pour dénominateurs des puissances de polynômes irréductibles et pour numérateurs un polynôme de degré inférieur au polynôme irréductible du dénominateur.*

```
ex: (x+3)/(x^4-5*x^2+4);
```

$$\frac{x+3}{x^4-5\,x^2+4}$$

```
partfrac(ex,x);
```

$$-\frac{1}{12\,(x+2)}+\frac{1}{3\,(x+1)}-\frac{2}{3\,(x-1)}+\frac{5}{12\,(x-2)}$$

3.2 Simplification

Les fractions (fonctions) rationnelles peuvent être réduites au même dénominateur par la fonction **ratsimp**, y compris les fonctions non-rationnelles, e.g. $\sin(x^2+1)$. La première étape de simplification est de regrouper les fractions partielles en une fonction rationnelle et ensuite élimine le plus grand commun diviseur du numérateur et du dénominateur. Pour les pôlynomes **ratsimp** fait soit la factorisation soit le développement suivant l'expression en question.

```
e1 : -(x^5 - x)/(x - 1);
```

$$\frac{x-x^5}{x-1}$$

```
ratsimp(e1);
```

$$-x^4-x^3-x^2-x$$

```
ratsimp(1/x+1/y);
```

$$\frac{y+x}{x\,y}$$

```
ratsimp((x+2)^2);
```

$$x^2+4\,x+4$$

```
ratsimp(a*x^2*z+y*x*z+2);
```

$$\left(x\,y+a\,x^2\right)z+2$$

La valeur par défaut de **algebraic** est **false**. En la fixant à **true**, on peut pousser encore loin la simplification.

```
ratsimp(1/(sqrt(x)-4));
```

$$\frac{1}{\sqrt{x}-4}$$

```
ratsimp(1/(sqrt(x)-4), algebraic:true;
```

$$\frac{\sqrt{x}+4}{x-16}$$

radcan est plus efficace quand il s'agit des fonctions rationnelles, logarithmiques et exponentielles. De manière similaire,l'option **algebraic** peut être fixée à **true** pour plus de simplification.

```
ex1: 1/(sqrt(x^2+1)+x)+sqrt(x^2+1)+x;
```

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

```
radcan(ex1);
```

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

`radcan(ex1),algebraic:true;` or equivalently `radcan(ex1),algebraic;`

$$2\,\sqrt{x^2+1}$$

important : When `%e_to_numlog` is true, `%e^(r*log(expr))` simplifies to `expr^r` if r is a rational number.

La commande **rootcontract** va convertir les produits de racines en racines de produits.

```
ex2: (sqrt(x)*y^(3/2));
```

$$\sqrt{x}\,y^{\frac{3}{2}}$$

```
rootscontract(ex2);
```

$$\sqrt{x\,y^3}$$

La valeur par défaut de **rootsconmode** est **true**. Si elle est changée à **all**, ou à **false** le comportement de **rootscontract** va changer.

```
rootscontract(x^(1/2)*y^(3/2)), rootsconmode:false;
```

$$\sqrt{x\,y^3}$$

`rootscontract(x^(1/2)*y^(1/4)), rootsconmode:false;`

$$\sqrt{x}\, y^{\frac{1}{4}}$$

Ces exemples montrent que quand **rootsconmode** est **false** la réduction est limitée aux exposants au dénominateur commun.
`rootscontract(x^(1/2)*y^(1/4)), rootsconmode:true;`

$$\sqrt{x\ \sqrt{y}}$$

`rootscontract(x^(1/2)*y^(1/3)), rootsconmode:true;`

$$\sqrt{x}\, y^{\frac{1}{3}}$$

Ces exemples montrent que même si **rootsconmode** est **true** la réduction sera faite si le dénominateur d'un exposant est une proportion multiple d'un autre.
`rootscontract(x^(1/2)*y^(1/3)), rootsconmode:all;`

$$\left(x^3\, y^2\right)^{\frac{1}{6}}$$

Dernièrement, dans le cas où la valeur de **rootsconmode** est **all** la réduction consiste à trouver le plus grand commun diviseur du dénominateur.

La commande **logcontract** va transformer des expressions en logarithme de façon à obtenir des expression plus compactes.
`exp:2*(a*log(x) + 2*a*log(y));`

$$2\ (2\, a\ \log y + a\ \log x)$$

`logcontract(exp);`

$$a\ \log\left(x^2\, y^4\right)$$

La commande `declare(a,integer);` va déclarer a être un entier. Ceci amène la modification suivante :
`logcontract(exp);`

$$\log\left(x^{2\,a}\, y^{4\,a}\right)$$

Considérer le problème suivant : `eq: 2*log(b/2+1.01)-log(2.91*(b-0.89));` La commande `solve(eq);` ne marche pas. Nous avons besoin de d'abord le simplifier par `ratsimp(eq);` ou même `fullratsimp(eq);` qui qui consiste à appliquer `ratsimp(eq)` plusiers fois jusqu'à ce qu'il n'y ait pas de changement dans l'expression `eq`. Ensuite, nous devrons utiliser `logcontract(eq);` et finalement `solve(eq),numer;` pour la solution.

La commande **scsimp** va simplifier une expression à partir des règles de simplification données.

```
exp: k^2*m^2*n^2-k*m;
r1: k * m=n;
scsimp(exp,r1);
```

$$n^4 - n$$

On aurait pu aussi fixer la valeur de n comme

```
r2: n=3;
scsimp(exp,r1,r2);
```

$$78$$

map

```
map(f,[a,b,c]);
```

$$[f(a), f(b), f(c)]$$

```
map(f,a+b+c);
```

$$f(a) + f(b) + f(c)$$

```
e1:x/(x^2) + (y^2+y)/y;
```

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

```
ratsimp(e1);
```

$$\frac{x\,y + x + 1}{x}$$

```
map(ratsimp,e1);
```

$$y + \frac{1}{x} + 1$$

Ou bien la commande : `multthru(ratsimp(e1));` aussi donne le même résultat.

3.3 Factorisation

De manière similaire **factor** va factoriser des expressions, i.e. nous allons réécrire les expressions en isolant les facteurs communs.

```
(%i23) eq1: x^2 - 4*x + 4;
                                  2
(%o23)                           x  - 4 x + 4
(%i24) factor(eq1);
                                       2
```

```
(%o24)                                (x - 2)
(%i25) eq2: 27*x^3 + 136*x^2 + 227*x + 126;
                                3         2
(%o25)                    27 x  + 136 x  + 227 x + 126
(%i26) factor(eq2);
                                        2
(%o26)                       (x + 2) (27 x  + 82 x + 63)
(%i27) eq3: (x - 1)*(2*x + 3) + (x - 1)^2;
                                                        2
(%o27)                     (x - 1) (2 x + 3) + (x - 1)
(%i28) factor(eq3);
(%o28)                            (x - 1) (3 x + 2)
```

3.4 Substitution

Il existe plusieurs commandes pour la substitution : **ev**, **subst**, **ratsubst**.
ev :

```
(%i29) a:3;
(%o29)                                  3
(%i30) b:5;
(%o30)                                  5
(%i31) c:a*x+b;
(%o31)                               3 x + 5
(%i32) ev(c,x=10);
(%o32)                                 35
```

```
e:a*(1-a);
```

$$(1-a)\,a$$

```
ev(e, a=x);
```

$$x\,(1-x)$$

Quand on exécute la commande ev, ceci ne change pas la valeur associée à l'expression originale. En faisant,
`e;`

$$(1-a)\,a$$

on voit que la valeur originale est gardée. Il existe deux autres opérateurs liés à l'évaluation : ' (ce qui est un simple guillemet ou une apostrophe [6]) et '' (deux apostrophes ou guillemet anglais).

L'opérateur ' empèche l'évaluation d'une expression :

– Quand elle est appliquée à une variable, elle évite l'évaluation de cette variable

```
a:20; b:a^2;
```

$$400$$

```
b:'a^2;
```

$$a^2$$

– Quand elle est appliquée à une fonction, elle évite l'évaluation de la focntion alors que les arguments de la fonction sont évalués.

```
a:c; b:d; c:2; d:4; f(x):=a*x/b+c+d;
```

$$f(x) := \frac{xa}{b} + c + d$$

```
f(x);
```

$$\frac{c\,x}{d} + 6$$

```
g:'f(x)+ a;
```

$$f(x) + c$$

```
h:'f(x)+ a+ c;
```

$$f(x) + c + 2$$

– Quand elle est aplliquée aux paranthèses rien n'est évalué.

```
'(f(x)+ a+ c);
```

$$f(x) + c + a$$

L'opérateur '' assure une évaluation supplémentaire :

– Quand elle est appliquée à une variable, elle remplace l'expression par sa valeur.

```
a:20; b:a^2;
```

$$400$$

```
b:'a^2;
```

$$a^2$$

```
c:''b;
```

$$400$$

6. Il ne faut pas confondre avec le symbole prime – une apostrophe droite – qui est utilisé, généralement, en math pour désigner la dérivée d'une fonction, e.g. f'

– Les fonctions sont évaluées de manière similaire.

```
a:c; b:d; c:2; d:4; f(x):=a*x/b+c+d;
```

$$f(x) := \frac{xa}{b} + c + d$$

```
f(x);
```

$$\frac{c\,x}{d} + 6$$

```
h:"(f(x));
```

$$\frac{x}{2} + 6$$

– Toutes les expressions dans les paranthèses sont évaluées (sauf si elles sont empéchées par ').

```
"(f(x)+ a + c);
```

$$\frac{x}{2} + 10$$

```
"('f(x)+ a + c);
```

$$f\,(x) + 4$$

subst

Pour substituer une varaible a à une variable b dans une expression c, la commande est :
`subst(a,b,c);` ou également
`subst(b=a,c);`

Pour pouvoir utiliser cette commande b doit être un atome (un nombre, une chaîne de caractères – string – ou une sous-expression complète de c). Dans le cas où b n'est pas atome, on peut utiliser **ratsubst**.

```
e: a*(1-2*a);
```

$$(1 - 2\,a)\;a$$

`subst(4,a,e);` [ou également `subst(a=4,e);`]

$$2$$

```
subst(e:a*b+c*d);
subst([a=1,b=2],e);
```

$$c\,d + 2$$

On aurait également pu utiliser `ev(e,a=1,b=2);` ou encore `scsimp(e,a=1,b=2),numer;`

ratsubst

```
e: (a+b+c)/d;
```

$$\frac{c + b + a}{d}$$

```
subst(d,a+b,e);
```

$$\frac{c+b+a}{d}$$

```
ratsubst(d,a+b,e);
```

$$\frac{d+c}{d}$$

MAIS `ratsubst(a+b=d,e);` ne marche pas.

Une utilisation importante de ratsubst est quand l'option **radsubstflag** est changée en **true**. Dans ce cas-ci, on peut substituer les expressions contenant des puissances (radicaux).

```
ratsubst(u,sqrt(x), x+2);
```

$$x+2$$

```
ratsubst(u,sqrt(x), x+2), radsubstflag:true;
```

$$u^2+2$$

3.5 Parts

part

On peut extraire et utiliser/assigner les parties d'une equation par la commande `part`.

```
(%i33) eq1:x^2+4*x+3=(y^2+5)/(x-3);
                                                 2
                                2                y  + 5
(%o33)                         x  + 4 x + 3 = ------
                                                x - 3
(%i34) part(eq1,0);
(%o34)                                  =
(%i35) part(eq1,1);
                                    2
(%o35)                             x  + 4 x + 3
(%i36) part(eq1,2);
                                      2
                                     y  + 5
(%o36)                               ------
                                     x - 3
```

On peut même accéder aux sous-parties des parties :

```
(%i37) part(eq1,2,0);
(%o37)                                  /
(%i38) part(eq1,2,1);
                                         2
(%o38)                                  y  + 5
(%i39) part(eq1,2,2);
(%o39)                                   x - 3
```

De manière similare, les commandes `lhs` et `lhs` sertvent à extraire les parties droite et gauche d'une égalité :

```
(%i40) lhs(eq1);
                                       2
(%o40)                                 x  + 4 x + 3
(%i41) rhs(eq1);
                                          2
                                         y  + 5
(%o41)                                   ------
                                         x - 3
```

Pour extraire les composantes réelles et imaginaires des nombres complexes les commandes sont `realpart` et `imagpart`

```
(%i42) e1:  3+%i*5;
(%o42)                                 5 %i + 3
(%i43) realpart(e1);
(%o43)                                    3
(%i44) imagpart(e1);
(%o44)                                    5
```

4 Algèbre linéaire

L'algèbre linéaire est la branche des mathématiques qui s'intéresse à l'étude des espaces vectoriels (ou espaces linéaires), des vecteurs, des transformations linéaires et des systèmes d'équations linéaires.

4.1 Vecteurs et matrices

Pour entrer une matrice a, la commande est :

```
a : matrix([x, y, z], [4, 5, 6], [7, 8, 9]);
```

$$\begin{pmatrix} x & y & z \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

Pour accéder à une ligne, disons la première, on fait `row(a,1);`

$$\begin{pmatrix} x & y & z \end{pmatrix}$$

Et pour la colonne 3, on fait `col(a,3);`

$$\begin{pmatrix} z \\ 6 \\ 9 \end{pmatrix}$$

Et la transposée de la matrice a, est obtenu par `transpose(a);`

$$\begin{pmatrix} x & 4 & 7 \\ y & 5 & 8 \\ z & 6 & 9 \end{pmatrix}$$

Pour accéder aux éléments de a, on utilise les commandes suivantes

```
a[1,1]:1; a[1,2]:2;a[1,3]:3;
```

Maintenant afficheons a, par la commande `a;`

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

4.2 Opérations sur les matrices

Définissons un deuxième matrice, b ;

```
b : matrix([x, y, z], [y, z, x], [z, x, y]);
```

Et maintenant les opérations :

`a+b;`

$$\begin{pmatrix} x+1 & y+2 & z+3 \\ y+4 & z+5 & x+6 \\ z+7 & x+8 & y+9 \end{pmatrix}$$

`a*b;`

$$\begin{pmatrix} x & 2y & 3z \\ 4y & 5z & 6x \\ 7z & 8x & 9y \end{pmatrix}$$

```
a.b;
```

$$\begin{pmatrix} 3z+2y+x & 2z+y+3x & z+3y+2x \\ 6z+5y+4x & 5z+4y+6x & 4z+6y+5x \\ 9z+8y+7x & 8z+7y+9x & 7z+9y+8x \end{pmatrix}$$

```
a^^2;
```

$$\begin{pmatrix} 30 & 36 & 42 \\ 66 & 81 & 96 \\ 102 & 126 & 150 \end{pmatrix}$$

```
a^2;
```

$$\begin{pmatrix} 1 & 4 & 9 \\ 16 & 25 & 36 \\ 49 & 64 & 81 \end{pmatrix}$$

4.3 Déterminant

```
d : determinant(b);
```

$$z\left(xy-z^2\right)+x\left(yz-x^2\right)-y\left(y^2-xz\right)$$

```
ratsimp(d);
```

$$-z^3+3xyz-y^3-x^3$$

```
factor(d);
```

$$-(z+y+x)\left(z^2-yz-xz+y^2-xy+x^2\right)$$

4.4 Valeurs propres-vecteurs propres

```
eigenvalues(a);
```

$$\left[\left[-\frac{3\sqrt{33}-15}{2},\frac{3\sqrt{33}+15}{2},0\right],[1,1,1]\right]$$

Le vecteur $[1,1,1]$ nous informe à propos de la multiplicité des valeurs propres. Dnas le cas présent, il n'y a pas de valeur propre répétée.

```
ev: eigenvectors(a);
```

$$\left[\left[\left[-\frac{3\sqrt{33}-15}{2},\frac{3\sqrt{33}+15}{2},0\right],\left[1,1,1\right]\right],\right.$$
$$\left.\left[1,-\frac{3\sqrt{33}-19}{16},-\frac{3\sqrt{3}\sqrt{11}-11}{8}\right],\left[1,\frac{3\sqrt{33}+19}{16},\frac{3\sqrt{3}\sqrt{11}+11}{8}\right],\left[1,-2,1\right]\right]$$

Le premier triple

$$\left[-\frac{3\sqrt{33}-15}{2}, \frac{3\sqrt{33}+15}{2}, 0\right]$$

donne les valeurs propres ; le triple suivant $\left[1,1,1\right]$ donne la multiplicité de ces valeurs. Les prochains triples sont les vecteurs propres associés. Pour afficher seulement un de ces vecteurs propres, disons le premier, et l'appeler $v1$ la commande est :

```
v1: part (ev, 2);
```

$$\left[1, -\frac{3\sqrt{33}-19}{16}, -\frac{3\sqrt{3}\sqrt{11}-11}{8}\right]$$

De manière similaire, pour afficher les autres vecteurs et attribuer les noms $v2$ et $v3$ les commandes sont :

```
v2: part (ev, 3);
v3: part (ev, 4);
```

Et pour avoir les trois en même temps : `vall:part(ev,[2,3,4]);`.

5 Résolution des équations

5.1 Une seule équation

La commande solve est une commande générique pour résoudre une equations. Si l'on a une seule équation polynomiale, solve revient à trouver les racines de ce polynôme. La commande allroots se diffère de solve ; elle n'accepte que des polynômes et sert à trouver toutes les racines d'un polynôme. Pour les racines réelles il faut utiliser la commande realroots. la commande realroots peut être utilisée comme realroots(expr) ou realroots (expr,tol) où tol est la tolérance que nous décidons.

```
(%i45) kill(all);
(%o0)                                done
(%i1) eq1: x^2-2*x+1;
                                      2
(%o1)                                 x  - 2 x + 1
(%i2) solve(eq1);
(%o2)                                [x = 1]
(%i3) multiplicities;
(%o3)                                  [2]
```

```
(%i4) allroots(eq1);
(%o4)                          [x = 1.0, x = 1.0]
(%i5) realroots(eq1);
(%o5)                                [x = 1]
(%i6) realroots(eq1,5e-6);
(%o6)                                [x = 1]
```

`solve(eq1)` est la même chose que `solve(eq1=0)`. On aurait pu écrire aussi `solve(`$x^2-2*x+1$`)` ou encore `solve(` $x^2-2*x=-1$`)`.

```
(%i7) f(x):=a* x^2+b*x+c;
                                          2
(%o7)                       f(x) := a x  + b x + c
(%i8) solve(f(x));

solve: more unknowns than equations.
Unknowns given :
[a, x, b, c]
Equations given:
    2
[a x  + b x + c]
 -- an error. To debug this try: debugmode(true);
```

Maxima n'arrive pas à résoudre le problème. Il faut préciser pour quelle variable, on voudrait résoudre cette équation.

```
(%i9) f(x):=a* x^2+b*x+c;
                                          2
(%o9)                       f(x) := a x  + b x + c
(%i10) sol:solve(f(x),x);
                          2                           2
                  sqrt(b  - 4 a c) + b        sqrt(b  - 4 a c) - b
(%o10)      [x = - --------------------, x = --------------------]
                           2 a                         2 a
```

Vérifions :

```
(%i11) s1:sol[1];
                                       2
                                  sqrt(b  - 4 a c) + b
(%o11)                     x = - --------------------
```

```
                                          2 a
(%i12) ver1:subst(s1,f(x));
                  2              2            2
           (sqrt(b  - 4 a c) + b)    b (sqrt(b  - 4 a c) + b)
(%o12)     ---------------------- - ---------------------- + c
                     4 a                         2 a
(%i13) expand(ver1);
(%o13)                                0
```

```
(%i14) ev(sol,a=2,b=6,c=4);
(%o14)                        [x = - 2, x = - 1]
(%i15) ev(sol,a=2,b=5,c=4);
                         sqrt(7) %i + 5      sqrt(7) %i - 5
(%o15)            [x = - --------------, x = --------------]
                              4                   4
```

Rappel : Faites attention à la similitude entre `subst(a=4,c);` et `subst(s1,f(x));` en fait, `s1` est la même chose que `x=...` On aurait aussi pu utiliser `ev(f(x),s1),expand;` ou bien `scsimp(f(x),s1);`

Si'il y a une équation compliquée nous pouvons d'abord voir le graphique de cette équation et après essayer de résoudre une fois d'avoir une idée sur l'intervalle dans lequel réside la/les solution/solutions.

Pour les fonctions transcendentales (exponentielles ou logarithmiques) et trigonométriques dont il est difficile de trouver les racines la commande nécessaire est find_root(expr,x,a,b) ou find_root(f(x),x,a,b). f(x) est une fonction ; expr une expression nommée ; x est la varaible indépendante ; a et b sont les bornes inférieures et supérieures. Puisque find_root utilise un algorithme de bisection il faut fournir pour chaque racine une borne supérieure et inférieure.

```
(%i16) f(x):=cos(x/%pi)*(exp(-(x/4)^2))-sin(x^(3/2))-5/4;
                             x          x 2         3/2    - 5
(%o16)          f(x) := cos(---) exp(- (-) ) - sin(x   ) + ---
                            %pi         4                   4
(%i17) plot2d( f(x),[x,0,5],
        [style, [lines,4,1] ],
         [xlabel," plot of f(x) "],[ylabel," "],
      [gnuplot_preamble, "set nokey; set xzeroaxis lw 2 "],
```

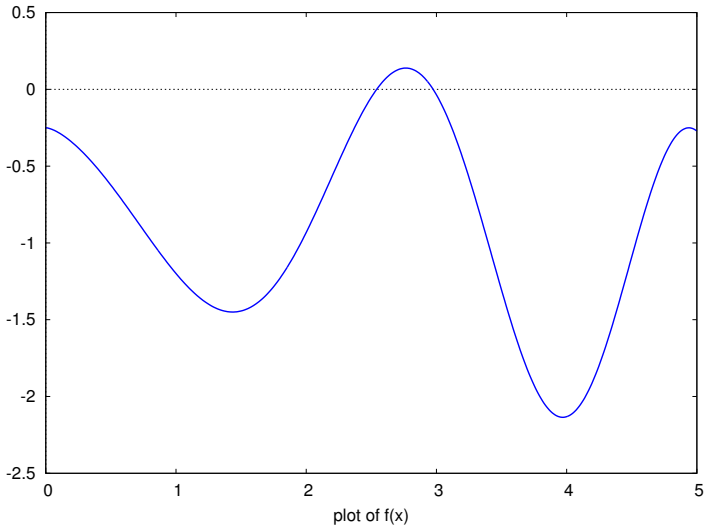

FIGURE 2 – Le graphique de la fonction $f(x) = \cos(x/\pi)\exp\left(-(x/4)^2\right) - \sin x^{\frac{3}{2}} - \frac{5}{4}$

```
      [gnuplot_term,ps],[gnuplot_out_file,"solve_plot.eps"] );
(%o17)                          solve_plot.eps
(%i18) x1:find_root(f(x),2.4,2.7);
(%o18)                         2.541050126730359
(%i19) x2:find_root(f,2.9,3.1);
(%o19)                         2.974603407751308
(%i20) x3:find_root(f,2.4,3.1);

find_root: function has same sign at endpoints:
        mequal(f(2.4), - 0.20112957752836), mequal(f(3.1), - 0.21298030271078)
 -- an error. To debug this try: debugmode(true);
```

Comme on le voit ci-dessus quand les bornes ne sont pas appropriées la commande find_root n'arrive pas à trouver des racines.

5.2 Système d'équations

Le système d'équations peut être linéaire où non-linéaire...

```
(%i21) kill(all);
(%o0)                                 done
(%i1) eq1:a*x+b*y=c;
(%o1)                              b y + a x = c
(%i2) eq2:d*x+e*y=f;
(%o2)                              e y + d x = f
(%i3) sol:solve([eq1,eq2],[x,y]);
                               c e - b f      c d - a f
(%o3)                 [[x = - ---------, y = ---------]]
                               b d - a e      b d - a e
(%i4) ev(sol,a=5,b=3,c=4,d=1,e=10,f=2);
                                    34      6
(%o4)                           [[x = --, y = --]]
                                    47      47
```

Et maintenant un système nonlinéaire :

```
(%i5) remvalue(all);
(%o5)                            [eq1, eq2, sol]
(%i6) eq3: x^2 + y^2 = 1;
                                   2    2
(%o6)                             y  + x  = 1
(%i7) eq4: x + 3*y = 0;
(%o7)                              3 y + x = 0
(%i8) sol2 : solve([eq3,eq4],[x,y]);
                 3                  1
(%o8) [[x = - --------, y = ----------------],
              sqrt(10)      sqrt(2) sqrt(5)
                                                  3                    1
                                        [x = --------, y = - ----------------]]
                                             sqrt(10)        sqrt(2) sqrt(5)
(%i9) float(sol2);
(%o9) [[x = - 0.94868329805051, y = 0.31622776601684],
                                [x = 0.94868329805051, y = - 0.31622776601684]]
```

Une autre façon de résoudre ce système est d'abord éliminer une des variables, disons x, et résoudre après pour y. La commande est

```
(%i10) eq3: x^2 + y^2 = 1;
                                   2    2
```

```
(%o10)                              y  + x  = 1
(%i11) eq4: x + 3*y = 0;
(%o11)                              3 y + x = 0
(%i12) eq_y : eliminate([eq3,eq4],[x]);
                                         2
(%o12)                              [10 y  - 1]
(%i13) sol_y : solve(eq_y,y);
                                    1             1
(%o13)                  [y = - --------, y = --------]
                               sqrt(10)      sqrt(10)
(%i14) float(sol_y);
(%o14)          [y = - 0.31622776601684, y = 0.31622776601684]
```

A la place de `sol_y:solve(eq_y,y); + float(sol_y);` on aurait pu utiliser la commande `sol_y:solve(eq_y,y),numer;`

On pourrait utiliser la commande batch pour lire et ensuite évaluer les commandes de Maxima contenues dans un fichier. Soit le fichier islm_exo.mac qui contient un modèle IS-LM de base

```
 kill(all);
linel : 60; /*satir genisligi 60 karakter olsun diyoruz.
                  default olan 79 latex icin cok uzun*/
display2d : false;
c0: 80;
c: 0.8;
C: c0+c*(Y-TN);
I0: 800;
b: 20;
I:I0-b*r;
G: 1000;
TN:1000;
Ms: 1200;
l1:0.4;
l2:40;
Md:l1*Y-l2*r;
eq1 : Y = C+I+G;
ev(solve(eq1,Y));
eq2 : Ms=Md;
ev(solve(eq2,Y));
sol : solve([eq1,eq2],[r,Y]);
```

La commande batch("/path/to/islm_exo.mac") va lire, évaluer et nous retourner la solution de ce modèle.

6 Graphiques

gilberto urroz'daki orneklere bir goz at!!!!

Pour quelques exemples montrant les opportunités disponible sous maxima, tapez :

```
plot2d(sin(x),[x,0,2*\%pi]);
plot2d([sin(x),cos(x)],[x,0,2*\%pi]);
plot2d([sin(x),cos(x),0],[x,0,2*\%pi]);
xx:[10, 20, 30, 40, 50];
yy:[.6, .9, 1.1, 1.3, 1.4];
xy:[[10,.6], [20,.9], [30,1.1], [40,1.3], [50,1.4]];
plot2d([discrete,xx,yy]);
plot2d([discrete, xy], [style, points]);
plot2d([gamma(x),1/gamma(x)],[x,-4.5,5],[y,-10,10],
    [gnuplot\_preamble,"set key bottom"]);
plot3d(atan(-x^2+y^3/4),[x,-4,4],[y,-4,4],[grid,50,50]);
plot3d(atan(-x^2+y^3/4),[x,-4,4],[y,-4,4],[grid,50,50],
    [gnuplot_pm3d,true]);
plot3d(atan(-x^2+y^3/4),[x,-4,4],[y,-4,4],[grid,50,50],
    [gnuplot_pm3d,true],[gnuplot_preamble,"set pm3d at b"]);
plot2d ([parametric, cos(t), sin(t), [t,-%pi,%pi],
      [nticks,80]], [x, -4/3, 4/3]);
plot2d ([parametric, cos(t), sin(t), [t, -%pi*2, %pi*2],
        [nticks, 8]], [x, -2, 2], [y, -1.5, 1.5]);
plot2d ([x^3+2, [parametric, cos(t), sin(t), [t, -5, 5],
        [nticks, 80]]], [x, -3, 3]);
```

Remarque : Pour indiquer une place spécifique pour le fichier sin.eps, disons c :/work on devrait écrire plutôt "...[gnuplot_out_file,"c :/work/sin.eps"]..." Par exemple, la commande

```
plot2d([sin(x),0,0.3],[x,0,2*\%pi],[gnuplot_term,ps],
[gnuplot_out_file,"../comp_eco/sin.eps"]);
```

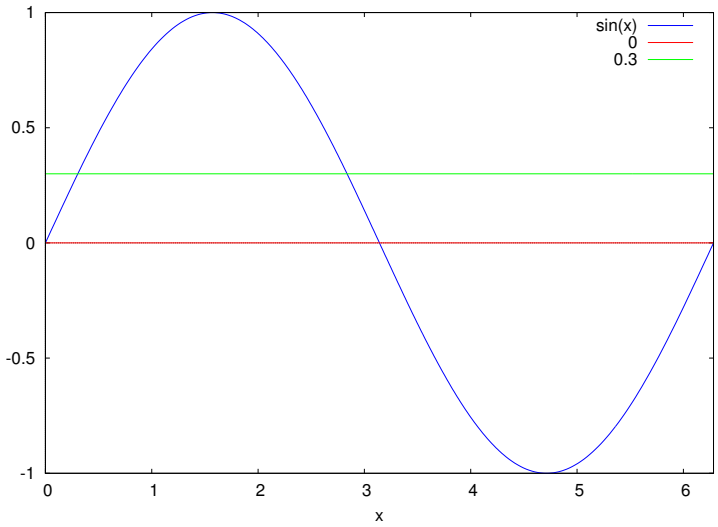

FIGURE 3 – Le graphique de la fonction sin(x)

```
../comp_eco/sin.eps
```

crée le graphique 3.

Remarque : *parametric plot* est le graphique de x, y quand x et y sont elles-mêmes fonctions d'une variable t.

Il existe un paquetage (*package*) `draw` destiné à réaliser des graphiques de manière plus simple. Il faut d'abord le charger par la commande **load(draw) ;**

```
(%i16) load(draw);
(%o16) C:/PROGRA~1/MAXIMA~1.1-G/share/maxima/5.25.1/share/draw/draw.lisp
(%i17) draw2d(/* the rational function */
       grid   = true,
       key    = "y = x^2/(x-2)",
       yrange = [-10,20],
       color  = red,
         explicit(x^2/(x-2),x,-9,15),

       /* asymptotes */
       key       = "",
       line_type = dots,
```

FIGURE 4 – Un premier graphique avec draw

```
        color     = blue,
          explicit(x+2,x,-9,15),
        nticks = 70,
          parametric(2,t,t,-10,20),
        /* labels and arrows */
        head_length = 0.3,
        color     = black,
        line_type   = solid,
            vector([5.35,2.45],[-1.53,3.25]),
            vector([-1,7.5],[3,0]),
        label_alignment = left,
            label(["y = x+2",6,2.5]),
        label_alignment = right,
            label(["x = 2",-1.7,7.5])    );
(%o17) [gr2d(explicit, explicit, parametric, vector, vector, label, label)]



draw2d(/*to create an eps file add*/
        /*terminal   = eps_color,*/
        key        = "Exponential func",
        color      = blue,
        line_width = 4,
             explicit(exp(x),x,-1,3),
        line_width = 2,
        color      = "#00ff00",   /* green, in hexadecimal */
        key        = "Cubic poly",
             explicit(%pi*x^3+sqrt(2)*x^2+10,x,0,3),
        xlabel     = "Horizontal axis",
        ylabel     = "Vertical axis");

 ------------------------------

[gr2d(explicit, explicit)]
```

FIGURE 5 – Un deuxième graphique avec draw

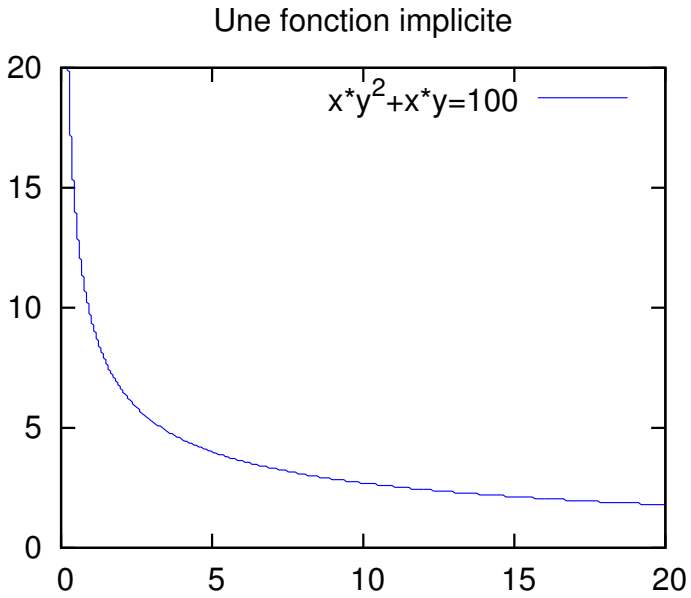

FIGURE 6 – Une courbe d'indifférence liée à la fonction d'utilité $U(x, y) = xy^2 + xy$. Cette courbe est dessinée pour le niveau d'utilité $U = 100$.

Pour produire un graphique implicite la commande est

```
(%i19) load(draw);
(%o19) C:/PROGRA~1/MAXIMA~1.1-G/share/maxima/5.25.1/share/draw/draw.lisp
(%i20) draw2d(file_name = "util", terminal= eps, key="x*y^2+x*y=100",
implicit(x*y^2+x*y=100,x,0,20,y,0,20),title= "Une fonction implicite");
(%o20)                          [gr2d(implicit)]
```

et le résultat est le graphique 6.

7 Différentiation et intégration

7.1 Différenciation simple

Pour différencier une expression par rapport à une / plusieurs variables la commande est diff(expr,var1,n1,var2,n2,...). Ici, var1 est la variable 1 et n1 montre que l'expression va être différenciée n1 fois par rapport à var1. Idem pour var2 et n2.

```
(%i21) f: x^2*y+3 ;
                                    2
(%o21)                             x  y + 3
(%i22) g(x,y):= x^2*y^2-5 ;
                                         2  2
(%o22)                      g(x, y) := x  y  - 5
(%i23) diff(f,x);
(%o23)                               2 x y
(%i24) diff(f,x,2);
(%o24)                                2 y
(%i25) diff(g(x,y),x);
                                        2
(%o25)                               2 x y
(%i26) diff(g(x,y),x,2);
                                       2
(%o26)                                2 y
```

7.2 Différenciation implicite

supposons x et y soient liés par la relation e : exp(x*y) + x*x + y = 0 ; et qu'en plus y dépend de x, i.e. $y(x)$.

```
(%i27) kill(all);
(%o0)                                done
(%i1) e : exp(x*y) + x*x + y = 0;
                                   x y         2
(%o1)                             %e    + y + x  = 0
(%i2) diff(e,x);
                                      x y
```

```
(%o2)                            y %e    + 2 x = 0
(%i3) depends(y,x);
(%o3)                                 [y(x)]
(%i4) diff(e,x);
                           x y    dy        dy
(%o4)                    %e    (x -- + y) + -- + 2 x = 0
                              dx        dx
(%i5) solve(diff(e,x),diff(y,x));
                                        x y
                              dy     y %e    + 2 x
(%o5)                        [-- = - -------------]
                              dx          x y
                                       x %e    + 1


(%i6) f: sin(x)/x;
                                      sin(x)
(%o6)                                 ------
                                        x
(%i7) df1: diff(f,x);
                                   cos(x)   sin(x)
(%o7)                              ------ - ------
                                     x         2
                                              x
(%i8) df2: diff(f,x,2);
                           sin(x)   2 sin(x)   2 cos(x)
(%o8)                    - ------ + -------- - --------
                             x          3          2
                                       x          x
(%i9) plot2d([f,df1,df2,0],[x,-30,30],[gnuplot_term,ps],[gnuplot_out_file,"../comp_
(%o9)                        ../comp_eco/plot_1.eps
```

7.3 Application : problème du consommateur

Considérons le problème du consommateur : Max $U(x, y) = x^2 y$ s.c. $px + qy = R$ où x et y sont deux biens de consommation ; R le revenu, p, q les prix de x et y. Nous allons former d'abord le Lagrangien

$$\mathcal{L}(x, y, \lambda) = x^2 y + \lambda(R - px - qy)$$

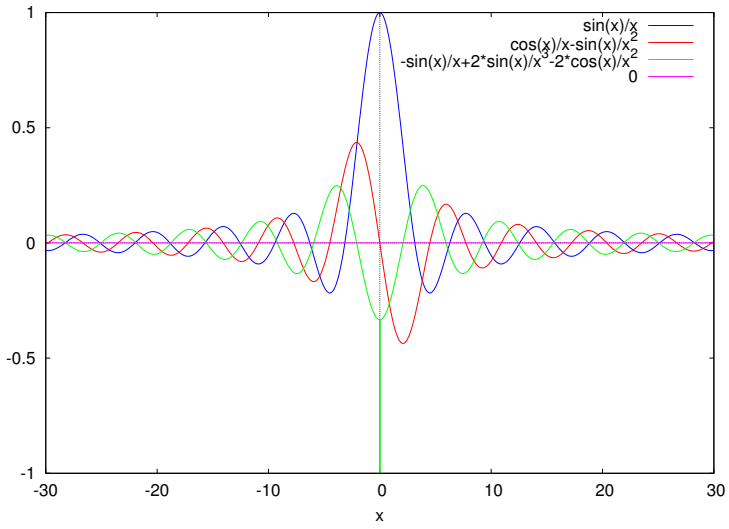

FIGURE 7 – Le graphique du $f, f'(x)$ et $f''(x)$

Après il faur écrire les conditions du premier ordre

$$\frac{\partial \mathcal{L}(x,y,\lambda)}{\partial x} = 2xy - \lambda p = 0$$
$$\frac{\partial \mathcal{L}(x,y,\lambda)}{\partial y} = x^2 - \lambda q = 0$$
$$\frac{\partial \mathcal{L}(x,y,\lambda)}{\partial \lambda} = R - px - qy = 0$$

ce qui va nous donner 3 équations en 3 inconnus. Pour faire la même chose avec Maxima les commandes sont :

```
(%i10) L: x^2*y+lambda*(R-p*x-q*y);
                                                    2
(%o10)                    (R - q y - p x) lambda + x  y
(%i11) cpo1: diff(L,x);
                             dy                   2 dy
(%o11)                 (- q -- - p) lambda + x  -- + 2 x y
                             dx                   dx
(%i12) cpo2: diff(L,y);
                                    2
(%o12)                             x  - q lambda
```

```
(%i13) cpo3: diff(L,lambda);
(%o13)                          R - q y - p x
(%i14) sol: solve([cpo1,cpo2,cpo3],[x,y,lambda]);

diff: variable must not be a number; found: 0
 -- an error. To debug this try: debugmode(true);
```

7.4 Application : Jacobien et Hessien

La matrice jacobienne est la matrice des dérivées partielles du premier ordre d'une fonction vectorielle.

En mathématiques, la matrice hessienne d'une fonction numérique f est la matrice carrée, notée $H(f)$, de ses dérivées partielles secondes.

```
(%i15) display2d:true;
(%o15)                                true
(%i16) depends([F, G],[u,v]);
(%o16)                         [F(u, v), G(u, v)]
(%i17) jacobian([F,G],[u,v]);
                                  [ dF  dF ]
                                  [ --  -- ]
                                  [ du  dv ]
(%o17)                            [        ]
                                  [ dG  dG ]
                                  [ --  -- ]
                                  [ du  dv ]
(%i18) hessian([F],[u,v]);
                              [    2        2    ]
                              [   d F      d F   ]
                              [  [---]   [-----] ]
                              [     2     du dv  ]
                              [   du             ]
(%o18)                        [                  ]
                              [    2        2    ]
                              [   d F      d F   ]
                              [ [-----]   [---]  ]
                              [  du dv       2   ]
                              [            dv    ]
```

```
(%i19) eq1: x^2*y+y^3-10;
                                3    2
(%o19)                         y  + x  y - 10
(%i20) eq2: (x+y)^2-x*y-20;
                                    2
(%o20)                        (y + x)  - x y - 20
(%i21) jacobian([eq1, eq2],[x,y]);
               [       2 dy    2 dy                   2    2   ]
               [    3 y  -- + x  -- + 2 x y       3 y  + x    ]
               [         dx      dx                            ]
(%o21)         [                                               ]
               [            dy          dy                     ]
               [ 2 (y + x) (-- + 1) - x -- - y  2 (y + x) - x ]
               [            dx          dx                     ]
(%i22) hessian([eq1],[x,y]);
       [        2        2                                                ]
       [     2 d y    2 d y         dy 2        dy              dy        ]
       [ [3 y  --- + x  --- + 6 y (--)  + 4 x -- + 2 y]  [6 y -- + 2 x] ]
       [          2        2         dx          dx              dx        ]
(%o22) [       dx       dx                                               ]
       [                                                                 ]
       [                     dy                                          ]
       [                 [6 y -- + 2 x]                     [6 y]      ]
       [                     dx                                          ]
```

7.5 Application : approximation de Taylor

L'approximation de Taylor d'ordre n d'une fonction $f(x)$ en $x = a$ est donnée par

$$T(x) = f(a) + f'(a)(x-a) + \cdots + \frac{f^{(n)}(a)}{n!}(x-a)^n$$

où la vrai valeure est

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \ldots$$

Ex : Quelle est la valeur approximative de $\sqrt{9.1}$ pour $n = 1$? On a : $f(x) = \sqrt{x}$,

$a = 9$ et $h = x - a$.

$$f(x) = \sqrt{x} \Rightarrow f(9) = 3$$
$$f'(x) = \frac{1}{2}\frac{1}{\sqrt{x}} \Rightarrow f'(9) = \frac{1}{6}$$
$$f(a+h) = f(a) + f'(a)h$$
$$\sqrt{9.0 + 0.1} = 3 + \frac{1}{6}0.1 = 3.01666$$

La valeur exacte est 3.01662.
Ex : Quelle est la valeur approximative de 1/2.1 pour $n = 1$? Soit $f(x) = 1/x$. On a $f(x) = f(a) + f'(a)(x - a)$ pour $n = 1$. Mettons $a = 2$ alors $h = x - a = 2.1 - 2 = 0.1$.

$$f(x) = 1/x \Rightarrow f(2) = 1/2$$
$$f'(x) = -1/x^2 \Rightarrow f'(2) = -1/4$$
$$f(2.1) = f(2) + f'(2)0.1$$
$$\frac{1}{2.1} = \frac{1}{2} - \frac{0.1}{4} = 0.4750$$

La valeur excate est 0.4761.

La commande `taylor`(f, x, a, n) fait un développement de Taylor d'ordre n de l'expression f pour x autour du point $x = a$.

```
(%i23) fpprintprec:8;
(%o23)                                    8
(%i24) display2d:false;

(%o24) false
(%i25) t:9.1^0.5;

(%o25) 3.0166206
(%i26) f:x^0.5;

(%o26) x^0.5
(%i27) t1:taylor (f, x, 9, 1);

rat: replaced 0.5 by 1/2 = 0.5
(%o27) 3+(x-9)/6
```

```
(%i28) float(ev(t1,x=9.1));

rat: replaced 3.0166667 by 181/60 = 3.0166667
(%o28) 3.0166667
(%i29) t2:taylor (f, x, 9, 2);

rat: replaced 0.5 by 1/2 = 0.5
(%o29) 3+(x-9)/6-(x-9)^2/216
(%i30) ev(t2,x=9.1),numer;

(%o30) 3.0166204
(%i31) t3:taylor (f, x, 9, 3);

rat: replaced 0.5 by 1/2 = 0.5
(%o31) 3+(x-9)/6-(x-9)^2/216+(x-9)^3/3888
(%i32) ev(t3,x=9.1),numer;

(%o32) 3.0166206
(%i33) t4:taylor (f, x, 8, 3);

rat: replaced 0.5 by 1/2 = 0.5
(%o33) 2*sqrt(2)+sqrt(2)*(x-8)/8-sqrt(2)*(x-8)^2/256+sqrt(2)*(x-8)^3/4096
(%i34) ev(t4,x=9.1),numer;

(%o34) 3.0166567
```

Comme on le voit dans les examples ci-dessus, le point autour duquel on a fait le développement de Taylor est important. Quand $a = 9$ un développement d'ordre 3 nous donne la bonne réponse jusqu'au septième chiffre après virgule, tandis que quand $a = 8$ on n'a que les 4 premier chiffres après virgule qui sont bons.

8 Equations differentielles

Une équation différentielle ordinaire est

$$F[x, y(x), y'(x), y''(x), ..., y^{(n)}(x)] = 0$$

Prenons le cas le plus simple où cette équation est d'ordre 1

$$\frac{dy}{dx} = f(x, y)$$

La solution générale de cette équation a la forme

$$y = y(x, c)$$

où c est une constante arbitraire. Afin de déterminer la solution exacte d'une équation différentielle on doit connaître la constante "c" qui apparaît dans la solution générale. Pour cela, il faut connaître en un point la valeur de y. Cette valeur particulière est la plupart du temps une valeur extremum : c'set soit la valeur initiale soit la valeur terminale.

Prenons l'exemple de

$$\frac{dy}{dx} = -2xy, \quad y(0) = 1$$

La solution générale est

$$y(x) = ce^{-x^2}$$

Remplaçons la condition $y(0) = 1$, i.e. $x = 0, y = 1$.

$$1 = ce^0 \Rightarrow c = 1$$

et donc la solution exacte

$$y(x) = e^{-x^2}$$

Sous maxima on peut entrer des équations différentielles de trois façons : la première est d'utiliser la commande depends

```
(%i35) depends(y,x);

(%o35) [y(x)]
(%i36) de1: diff(y,x)=-2*x*y;

(%o36) 'diff(y,x,1) = -2*x*y
(%i37) de2: diff(y,x,2)+2*y=5*x;

(%o37) 'diff(y,x,2)+2*y = 5*x
(%i38) dependencies;

(%o38) [y(x),F(u,v),G(u,v)]
```

```
(%i39) remove(y,dependency);

(%o39) done
(%i40) dependencies;

(%o40) [F(u,v),G(u,v)]
```

Et la deuxième consiste à utiliser l'opérateur (') :

```
(%i41) de1: 'diff(y,x)=-2*x*y;

(%o41) 'diff(y,x,1) = -2*x*y
(%i42) de2: 'diff(y,x,2)+2*y=5*x;

(%o42) 'diff(y,x,2)+2*y = 5*x
```

Et la dernière est décrire $y(x)$ directement

```
(%i43) de1: diff(y(x),x)=-2*x*y(x);

(%o43) 'diff(y(x),x,1) = -2*x*y(x)
(%i44) de2: diff(y(x),x,2)+2*y(x)=5*x;

(%o44) 'diff(y(x),x,2)+2*y(x) = 5*x
```

Pour résoudre une équation différentielle on peut utiliser la commande **ode2**

```
(%i45) de1: 'diff(y,x)=-2*x*y;

(%o45) 'diff(y,x,1) = -2*x*y
(%i46) sol1:ode2(de1,y,x);

(%o46) y = %c*%e^-x^2
(%i47) de2: 'diff(y,x,2)+2*y=5*x;

(%o47) 'diff(y,x,2)+2*y = 5*x
(%i48) sol2:ode2(de2,y,x);

(%o48) y = %k1*sin(sqrt(2)*x)+%k2*cos(sqrt(2)*x)+5*x/2
```

Et si l'on connait les conditions initials (seulement en un point comme x=0) on peut utiliser les commandes ic1 et ic2 pour obtenir les solutions exactes. Dans nos exemples, pour l'équation de1, la condition initiale est $y(0) = 1$. Pour de2, les conditions initials sont $y(0) = 2$ et $y'(0) = 2$.

```
(%i49) de1: 'diff(y,x)=-2*x*y;

(%o49) 'diff(y,x,1) = -2*x*y
(%i50) sol1:ode2(de1,y,x);

(%o50) y = %c*%e^-x^2
(%i51) ic1(sol1,x=0,y=1);

(%o51) y = %e^-x^2
(%i52) de2: 'diff(y,x,2)+2*y=5*x;

(%o52) 'diff(y,x,2)+2*y = 5*x
(%i53) sol2:ode2(de2,y,x);

(%o53) y = %k1*sin(sqrt(2)*x)+%k2*cos(sqrt(2)*x)+5*x/2
(%i54) ic2(sol2,x=0,y=2,'diff(y,x)=2);

(%o54) y = -sin(sqrt(2)*x)/2^(3/2)+2*cos(sqrt(2)*x)+5*x/2
```

Quand nous connaissons les valeurs de y pour différentes valeurs de x, on parle des conditions aux limites ("Boundary values"). Dans l'équation de2 on suppose avoir les conditions aux limites suivantes : $y(0) = 1$ et $y(4) = 20$.

```
(%i55) kill(all);

(%o0) done
(%i1) de2: 'diff(y,x,2)+2*y=5*x;

(%o1) 'diff(y,x,2)+2*y = 5*x
(%i2) sol2:ode2(de2,y,x);

(%o2) y = %k1*sin(sqrt(2)*x)+%k2*cos(sqrt(2)*x)+5*x/2
(%i3) bc2(sol2,x=0,y=1,x=4,y=20);

(%o3) y = -(cos(2^(5/2))-10)*sin(sqrt(2)*x)/sin(2^(5/2))+cos(sqrt(2)*x)+5*x/2
```

Pour résoudre un système d'équations différentielles à coefficients constant la commande nécessaire est desolve. **Remarque :** Pour pouvoir utiliser desolve on est obligé utiliser `diff`$(y(x), x)$ au lieu de `diff`(y, x) ou de `'diff`(y, x).

```
(%i4) linel:80;

(%o4) 80
(%i5) de1:diff(x(t),t)=diff(y(t),t)+sin(t);

(%o5) 'diff(x(t),t,1) = 'diff(y(t),t,1)+sin(t)
(%i6) de2:diff(y(t),t,2)=diff(x(t),t) - cos(t);

(%o6) 'diff(y(t),t,2) = 'diff(x(t),t,1)-cos(t)
(%i7) desolve([de1,de2],[x(t),y(t)]);

(%o7) [x(t) = %e^t*('at('diff(y(t),t,1),t = 0))
            -'at('diff(y(t),t,1),t = 0)+x(0),
       y(t) = %e^t*('at('diff(y(t),t,1),t = 0))
            -'at('diff(y(t),t,1),t = 0)+cos(t)+y(0)-1]
```

$$x(t) = e^t \left(\frac{d}{dt} y(t) \Big|_{t=0} \right) - \frac{d}{dt} y(t) \Big|_{t=0} + x(0),$$
$$y(t) = e^t \left(\frac{d}{dt} y(t) \Big|_{t=0} \right) - \frac{d}{dt} y(t) \Big|_{t=0} + \cos t + y(0) - 1$$

Pour trouver la solution exacte on peut utiliser uniquement les conditions initiales au point $x = 0$ par la commande atvalue.

```
(%i8) de1:diff(x(t),t)=diff(y(t),t)+sin(t);

(%o8) 'diff(x(t),t,1) = 'diff(y(t),t,1)+sin(t)
(%i9) de2:diff(y(t),t,2)=diff(x(t),t) - cos(t);

(%o9) 'diff(y(t),t,2) = 'diff(x(t),t,1)-cos(t)
(%i10) atvalue(x(t),t=0,1);
```

```
(%o10) 1
(%i11) atvalue(y(t),t=0,2);

(%o11) 2
(%i12) atvalue(diff(y(t),t),t=0,3);

(%o12) 3
(%i13) desolve([de1,de2],[x(t),y(t)]);

(%o13) [x(t) = 3*%e^t-2,y(t) = cos(t)+3*%e^t-2]
```

Pour résoudre le système suivant

$$\dot{x} = x - 3y$$

$$\dot{y} = -2x + y$$

$$x(0) = 6.25 \quad y(0) = 5$$

on écrira

```
(%i14) sde1:diff(x(t),t)=x(t)-3*y(t);

(%o14) 'diff(x(t),t,1) = x(t)-3*y(t)
(%i15) sde2:diff(y(t),t)=-2*x(t)+y(t);

(%o15) 'diff(y(t),t,1) = y(t)-2*x(t)
(%i16) atvalue(x(t),t=0,6.25);

(%o16) 6.25
(%i17) atvalue(y(t),t=0,5);

(%o17) 5
(%i18) desolve([sde1,sde2],[x(t),y(t)]);

rat: replaced -6.25 by -25/4 = -6.25
(%o18) [x(t) = %e^t*(25*cosh(sqrt(6)*t)/4-15*sinh(sqrt(6)*t)/sqrt(6)),
        y(t) = %e^t*(5*cosh(sqrt(6)*t)-25*sinh(sqrt(6)*t)/(2*sqrt(6)))]
```

Pour visualiser les champs de directions d'une équation différentielle du premier ordre ou d'un système d'équations différentielles autonome (avec 2 équations), on utilise la commande plotdf. Si l'on veut le graphique des champs de direction de l'équation $y'(x) = x - y^2$, pour $-1 \leq x \leq 3$,

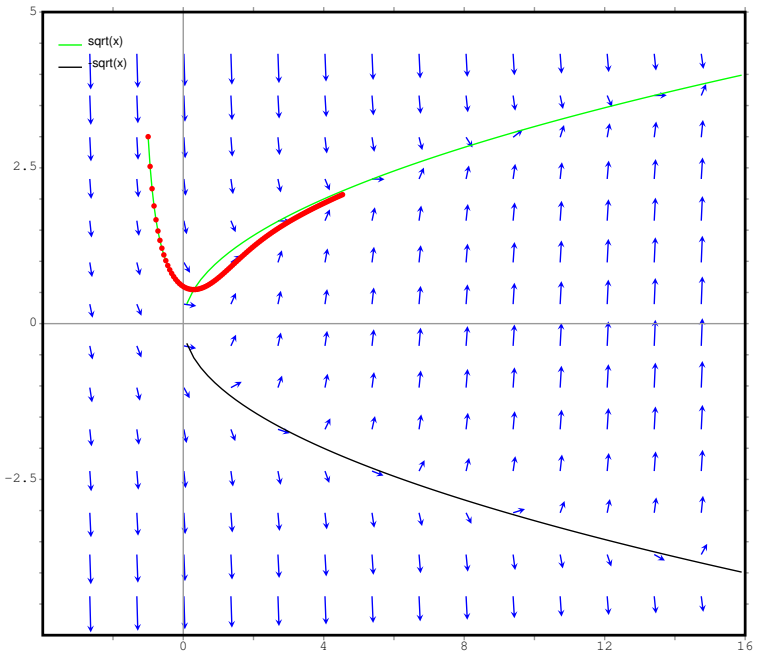

FIGURE 8 – Les champs de direction de l'équation $y'(x) = x - y^2$

```
load("plotdf");
plotdf(x-y^2,[xfun,"sqrt(x);-sqrt(x)"],[trajectory_at,-1,3],
    [y,-5,5], [x,-5,15])$
```

Et pour le système

$$\dot{x} = x - 3y$$
$$\dot{y} = -2x + y$$
$$x(0) = 6.25 \quad y(0) = 5$$

on a :

```
plotdf([x-3*y,-2*x+y],[trajectory_at,6.25,5], [y,-5,5], [x,-5,5]);
```

Références

[1] Maxima, http://maxima.sourceforge.net/

[2] http://www.csulb.edu/~woollett/

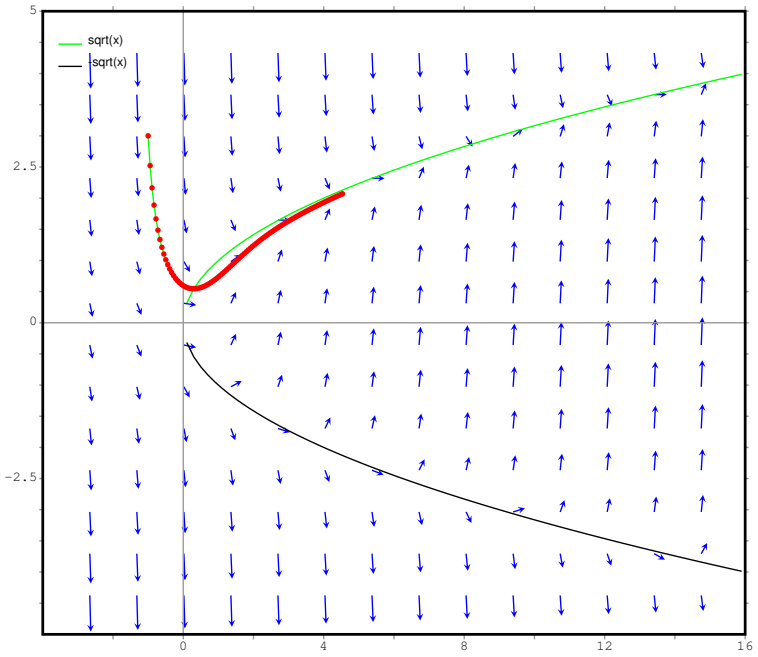

FIGURE 9 – Les champs de direction du système $\dot{x} = x - 3y, \dot{y} = -2x + y$ où $x_0 = 6.25, y_0 = 5$