Courte introduction à *Octave* ou *Matlab*

Ce document donne une courte introduction à l'utilisation d'*Octave* ou de *Matlab*. Il présente les éléments les plus importants qu'un utilisateur débutant doit absolument connaître pour pouvoir utiliser ces logiciels de calcul. Pour l'approfondissement, il existe des documents plus détaillés qui sont à disposition en format PDF. Dans la suite, tous les exemples seront donnés pour *Octave* dans l'environnement Unix. Pour *Matlab*, les instructions sont pratiquement identiques. En cas de problèmes, voir la documentation explicite de *Matlab*.

1 Démarrer l'application

En premier lieu, il faut ouvrir un *shell*, qui est une fenêtre Unix dans laquelle on donne des instructions ligne après ligne. Une fois démarré, le shell a l'aspect suivant:

shell
```
user@hostname:~/matlab/myworks>
```

Dans la ligne de commande du shell, on peut entrer les instructions les unes après les autres. Par exemple, l'instruction `pwd` permet de connaître notre position actuelle dans l'arbre du système de fichiers:

shell
```
user@hostname:~/matlab/myworks>pwd
/home/user/matlab/myworks
user@hostname:~/matlab/myworks>
```

L'instruction `cd` permet de se déplacer dans l'arbre vers le haut ou vers le bas: `cd ..` remonte d'un niveau, alors que `cd sous-dossier` descend d'un niveau dans le répertoire intitulé `sous-dossier`. Examiner l'exemple suivant:

shell
```
user@hostname:~/matlab/myworks>cd ..
user@hostname:~/matlab>
user@hostname:~/matlab>cd myworks
user@hostname:~/matlab/myworks>
```

Pour connaître la liste de tous les fichiers ou des sous-dossiers présents à la position actuelle où l'on se trouve, on dispose de l'instruction `ls`:

shell
```
user@hostname:~/matlab/myworks>ls
bisection.m    octave.tex       mygraphic.m
f01.m          fcn.m            myfigure.pdf
user@hostname:~/matlab/myworks>
```

Avant de démarrer *Octave*, il faut encore se placer dans le répertoire dans lequel on veut travailler, en utilisant adéquatement l'instruction `cd`. Dans les exemples qui suivent, nous nous placerons toujours dans le répertoire `.../matlab/myworks`.

Pour démarrer *Octave*, il suffit d'écrire dans la ligne de commande le mot `octave`. Voici ce que l'on obtient en principe au démarrage:

shell

```
user@hostname:~/matlab/myworks>octave
GNU Octave, version 3.8.2
Copyright (C) 2014 John W. Eaton and others.
This is free software; see the source code for copying conditions.
There is ABSOLUTELY NO WARRANTY; not even for MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE.  For details, type 'warranty'.

Octave was configured for "x86_64-unknown-linux-gnu".

Additional information about Octave is available at http://www.octave.org.

Please contribute if you find this software useful.
For more information, visit http://www.octave.org/get-involved.html

Read http://www.octave.org/bugs.html to learn how to submit bug reports.
For information about changes from previous versions, type 'news'.

octave:1>
```

La dernière ligne indique qu'octave est prêt à recevoir une instruction. Le numéro des instructions est automatiquement incrémenté. A noter que les instruction `cd`, `ls` et `pwd` de gestion des répertoires sont encore valables dans *Octave*.

Pour quitter *Octave*, il suffit de taper l'instruction `quit` dans la ligne de commande.

2 Installation et documentation

Une bonne installation de *Octave* consiste en deux parties: le package de base *Octave* et le package complémentaire *Octave-Forge*. En conséquence, le processus d'installation et la documentation consistent également en plusieurs parties distinctes.

2.1 Documentation

On trouve un choix de documentation à l'adresse suivante:
`staff.ti.bfh.ch/sha1` → *Octave information.*
Comme ducumentation de base examiner les notes de Griffiths.

2.2 Installation

- ***Ordinateurs à l'intérieur de l'école***
  Octave est installé sur tous les ordinateurs Linux de la TI-BFH qui sont supportés par les services informatiques. L'installation sous Windos va suivre prochainement.

- ***Linux** pour une machine privée*
  Presque toutes les distributions de Linux (Redhat, SUSE, Ubuntu, etc.) contiennent Octave et Octave-Forge. Ceci est certainement la manière la plus simple d'installer ces packages. Il est aussi possible de compiler directement les packages sources et de les installer soi-même.

- ***Windows** pour une machine privée*
  La page web ci–dessus donne de l'information pour l'installation.

3 L'utilisation d'*Octave* ligne après ligne

La manière la plus simple d'utiliser Octave est de travailler avec des lignes de commande exécutées les unes après les autres. A chaque fois qu'une instruction a été écrite sur la ligne de commande, elle peut être exécutée par la machine de calcul grâce à la touche `<RETURN>`*. Le résultat de l'opération est donné en retour dans la ligne suivante, appelée*

`ans`. *Voici quelques exemples sans commentaires:*

```
octave:1> 3 + 4*5
ans = 23
octave:2> 1/(12+1)
ans = 0.076923
octave:3> sqrt(13)
ans = 3.6056
octave:4>
```

Si l'on ajoute un point-virgule à la fin de la ligne de commande, l'opération est exécutée mais le résultat n'est pas donné en retour. Ceci est utile lorsque le résultat est long ou qu'il n'est pas nécessaire de l'afficher.

Pour donner quand cela est nécessaire des "noms" ou des "étiquettes" aux nombres, il est possible en Octave de déclarer des variables symboliques. Les noms des variables doivent être composés en principe des caractères de l'alphabet et des chiffres (mais pas à la première place). Certains nombres particuliers sont prédéfinis, comme par exemple le nombre π (`pi`*), le nombre d'Euler e (*`e`*) ou le nombre imaginaire j (*`i`*). L'instruction* `format long` *respectivement* `format short` *donne les résultats des nombres décimaux en forme longue ou raccourcie. Examiner les exemples ci-dessous:*

```
octave:4> e
e = 2.7183
octave:5> pi
pi = 3.1416
octave:6> format long
octave:7> pi
pi = 3.14159265358979
octave:8> a = 3
a = 3
octave:9> x = 3 ;
octave:10> y = 4 ;
octave:11> z = x + i*y ;
octave:12> z*z
ans =  -7 + 24i
```

4 La construction de fichiers scripts

Lorsqu'une session Octave est exécutée ligne après ligne, les opérations successives sont réalisées mais elles sont perdues à la fin. Une manière de pouvoir ré-exécuter plusieurs fois une séquence d'opérations est de construire un fichier script. Il s'agit d'un fichier de texte qui doit être édité (au moyen d'un éditeur de texte), puis sauvé dans le répertoire de travail. Le nom du fichier sous lequel le script est sauvé doit être bien choisi, de manière à correspondre au contenu des instructions qu'il contient. Le nom d'un fichier script doit toujours posséder l'extension `.m` *pour pouvoir être reconnu par Octave. Voici un exemple d'un fichier script, qui calcule l'hypoténuse et les deux angles non droits d'un triangle rectangle dont les cathètes sont respectivement 5 et 4. Les quelques lignes au début du fichier qui commencent par le caractère* `%` *sont des commentaires qui sont ignorés dans les calculs:*

```
% Ceci est le fichier script 'triangle.m'
% Ce programme elementaire calcule l'hypotenuse et les deux angles non droits
% d'un triangle rectangle dont les cathetes sont 5 et 4.

a = 5 ; % premiere cathete
b = 4 ; % deuxieme cathete
hyp = sqrt(4*4 + 5*5)
angle1 = atan(b/a)*180/pi
angle2 = atan(a/b)*180/pi
controle = angle1 + angle2
```

Pour exécuter ce script dans Octave, il suffit d'écrire dans la ligne de commande le nom du fichier, sans l'extension `.m` *:*

```
octave:11> triangle
hyp = 6.4031
angle1 = 38.660
angle2 = 51.340
controle = 90
```

L'instruction `help triangle` *donne le contenu des lignes de commentaires du début du fichier:*

```
octave:12> help triangle
triangle is the file: /home/cip/matlab/myworks/triangle.m

Ceci est le fichier script 'triangle.m'
Ce programme elementaire calcule l'hypotenuse et les deux angles non droits
d'un triangle rectangle dont les cathetes sont 5 et 4.
```

De manière plus générale, l'instruction `help` *ou* `help ...` *permet de lire la documentation en ligne des commandes Octave. Par exemple:*

```
octave:14> help atan
atan is a built-in mapper function

 -- Mapping Function:  atan (X)
     Compute the inverse tangent of each element of X.
```

Sous le système Unix, utiliser `help -i ...` *pour accéder à des informations d'aide plus détaillées.*

5 Vecteurs et matrices

Les langages Octave et Matlab sont construits dès leur origine dans le contexte du calcul vectoriel et du calcul matriciel. La plupart des instructions ont donc un rapport plus ou moins étroit avec la structure d'un vecteur (qui est une liste de nombres) ou d'une matrice (qui est une liste de listes de nombres). Un vecteur se définit en Octave par une liste de nombres entre deux parenthèses carrées `[...]`*. Pour un vecteur-ligne, séparer les éléments par des espaces ou des virgules. Pour un vecteur-colonne, séparer les éléments par des points-virgules. La définition d'une matrice est analogue, mais on utilise en plus le point-virgule comme séparateur des lignes.*

Toutes les opérations qui sont ordinairement définies pour le calcul vectoriel ou le calcul matriciel existent en Octave. Les exemples ci-dessous sont donnés sans commentaires:

```
octave:1> u = [-1 2 4] ;
octave:2> v = [1.5 2 -1]
v =
   1.5000   2.0000  -1.0000

octave:3> u + v          % somme de deux vecteurs de meme dimension
ans =
  0.50000  4.00000  3.00000

octave:4> v'             % vecteur transpose
ans =
   1.5000
   2.0000
```

```
  -1.0000

octave:5> a = u*v'       % produit scalaire des deux vecteurs u et v
a = -1.5000
```

```
octave:6> A = [-1 1 2 ; 3 -1 1 ; -1 3 4]
A =
  -1   1   2
   3  -1   1
  -1   3   4

octave:7> det(A)         % determinant de la matrice A
ans = 10.000

octave:8> A*u'           % produit matriciel de A par u
ans =
   11
   -1
   23
```

Octave est un langage très efficace dans la résolution des systèmes d'équations linéaires, pour lesquels il a été développé. Prenons par exemple le système de 3 équations à 3 inconnues ci-dessous:

$$\begin{array}{rcrcrcr} -x_1 & + & x_2 & + & 2x_3 & = & 10 \\ 3x_1 & - & x_2 & + & x_3 & = & -20 \\ -x_1 & + & 3x_2 & + & 4x_3 & = & 40 \end{array}$$

Ce système peut s'écrire sous forme la forme matricielle $\mathbf{A} \cdot \vec{x} = \vec{b}$

$$\begin{bmatrix} -1 & 1 & 2 \\ 3 & -1 & 1 \\ -1 & 3 & 4 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 10 \\ -20 \\ 40 \end{bmatrix}$$

La résolution de ce système peut être faite en Octave par la simple instruction `x = A\b` *. Examiner les quelques lignes qui suivent:*

```
octave:9> A = [-1 1 2 ; 3 -1 1 ; -1 3 4] ;
octave:10> b = [10 -20 40]' ;
octave:11> x = A\b
x =
    1.0000
   19.0000
   -4.0000
```

6 La définition des fonctions

En Octave, il y a un certain nombre de fonctions qui sont prédéfinies: `sin(x)`, `cos(x)`, `exp(x)`, `log(x)`, *etc. Comme on le voit dans ces exemples, les arguments des fonctions se transmettent toujours entre des parenthèses rondes.*

Dans la pratique, il est toutefois plutôt rare de travailler uniquement avec des fonctions standard comme celles-ci. Le plus souvent, les fonctions sont des fonctions composées, construites à partir des fonctions de base. Il existe donc en Octave une possibilité de déclarer et de construire des fonctions composées. Pour cela, on utilise l'éditeur pour créer, écrire et sauver un fichier–fonction, de manière semblable aux fichiers scripts.

Voici un exemple: dans un certain problème, on doit évaluer la fonction $f(x) = 5\,\cos(2\pi x)\,e^{-x}$ *pour différentes valeurs de la variable* x*. Dans Octave, on crée avec l'éditeur un fichier–fonction qui contient la définition de cette*

fonction. Ce fichier sera sauvé dans le répertoire de travail, sous un nom choisi, par exemple `f01.m` *. Le contenu de ce fichier est le suivant:*

```
function y = f01(x)
  y = 5*cos(2*pi.*x).*exp(-x) ;
end
```

A noter que le nom du fichier (ici `f01.m`*) doit correspondre au nom de la fonction donné à la première ligne de celui-ci.*

Autre remarque: à la deuxième ligne, l'opération de multiplication par x est représentée dans Octave par l'opérateur `.*` *, pour une raison qui provient de la structure vectorielle de ce langage : dans Octave, l'évaluation d'une fonction peut aussi se faire lorsque l'argument est un vecteur, l'évaluation étant alors effectuée pour toutes les composantes du vecteur. En général, un point placé juste devant une opération signifie en Octave que l'opération doit être effectuée individuellement pour tous les éléments de la liste donnée, ici la liste* `x` *qui est transmise comme argument à la fonction* `f01`*. Examiner les exemples ci-dessous:*

```
octave:13> f01(1.234)
ans = 0.14609

octave:14> x = [1 1.1 1.2 1.3] ;
octave:15> f01(x)
ans =
   1.83940   1.34649   0.46537  -0.42108
```

Lorsque la fonction ne doit être utilisée qu'un petit nombre de fois au cours de la session, il est aussi possible de la déclarer à l'intérieur même de la session sans créer de fichier–fonction. Pour cela, il faut en premier lieu définir les valeurs de l'argument x de la fonction, puis écrire la fonction dans une ligne de commande. Par exemple:

```
octave:16> x = 1.2 ;
octave:17> y = 5*cos(2*pi*x)*exp(-x)
y = 0.46537

octave:18> x = [1.2 1.3 1.4 1.5] ;
octave:19> y = 5*cos(2*pi.*x).*exp(-x)
y =
   0.46537  -0.42108  -0.99751  -1.11565
```

La déclaration des fonctions à plusieurs variables se fait de manière analogue. Un exemple sera donné dans la section suivante qui présente les représentations graphiques.

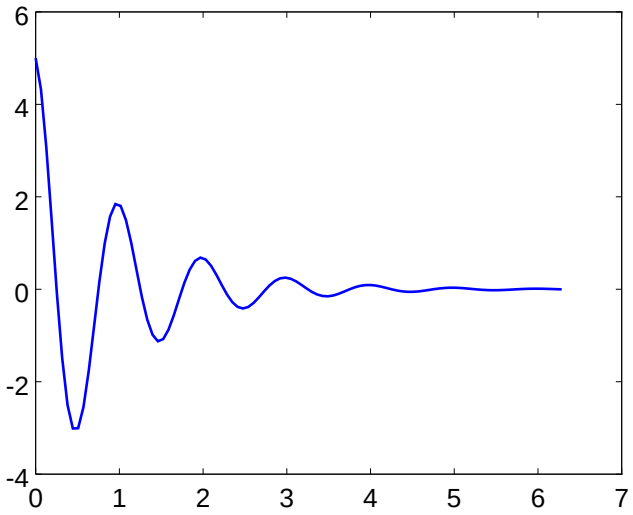
7 Graphiques des fonctions 2D et 3D

Pour représenter une fonction à une variable, le procédé habituel est de définir d'abord un intervalle $[a, b]$ dans lequel on veut dessiner la fonction, de calculer les valeurs de la fonction pour tous les points de cet intervalle, puis de reporter ces valeurs le long de l'axe Oy. En théorie, on calcule et on reporte les valeurs pour tous les points de l'intervalle. En pratique, on choisit dans l'intervalle $[a, b]$ un nombre fini de points représentatifs, et l'on ne calcule les valeurs que pour ces points-là. Le plus souvent, les points représentatifs sont distribués de manière homogène dans l'intervalle entre a et b.

En Octave, le procédé est exactement le même. On construit d'abord une liste de n points représentatifs dans l'intervalle $[a, b]$, distribués de manière régulière. Pour cela, on dispose de l'instruction `linspace(a,b,n)` *qui réalise cette tâche. Ensuite, on calcule les valeurs de la fonction pour tous les nombres de la liste. Cette tâche est exécutée en une seule fois. Enfin, on génère le graphique grâce à l'instruction* `plot()`*. Voici par exemple comment on génére un graphique de la fonction* `f01` *(définie plus haut) dans l'intervalle $[0, 2\pi]$, et en prenant 100 points intermédiaires représentatifs:*

```
octave:20> x = linspace(0, 2*pi, 100) ;
octave:21> y = f01(x) ;
octave:22> plot(x,y)
octave:23> print('graphe01.png')
```

La dernière ligne permet de sauver la figure dans un fichier graphique `graphe01.png` *de format PNG.*



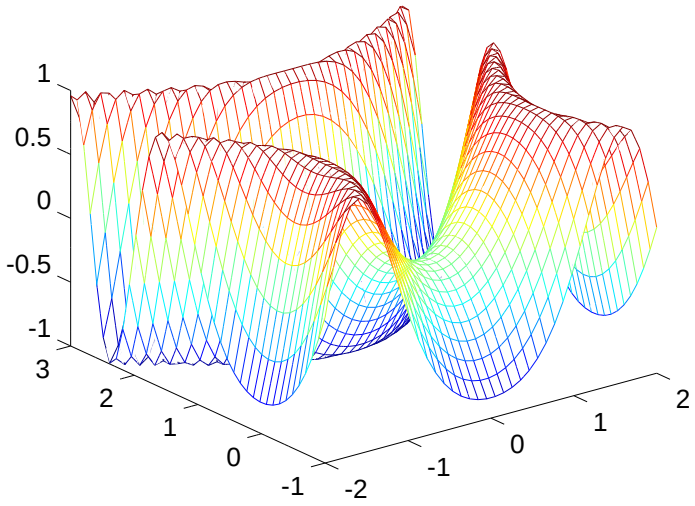
De nombreuses options graphiques sont possibles: titre de la figure, labels des axes, grille en arrière-fond, couleurs, etc. On se reportera pour cela à la documentation détaillée.

Pour le graphique d'une fonction à deux variables, le procédé est analogue, mais il faut définir à l'avance une grille 2D plutôt qu'un simple intervalle, au-dessus de laquelle on dessinera la fonction. Par exemple, on voudrait générer le graphe de la fonction $f(x,y) = \sin(x^2 - y^2)$ *sur le carré* $[-2,2] \times [-1,3]$*, en prenant chaque fois* 40 *points intermédiaires dans chaque direction. La fonction est définie dans le fichier-fonction* `f02.m` *dont le contenu est le suivant:*

```
% Contenu du fichier-fonction f02.m
function z = f02(x,y)
z = sin(x.^2 - y.^2) ;
end
```

Les instructions Octave pour générer la figure sont les suivantes:

```
octave:24> x = linspace(-2,2,40) ;
octave:25> y = linspace(-1,3,40) ;
octave:26> [xx,yy] = meshgrid(x,y) ;
octave:27> z = f02(xx,yy) ;
octave:28> mesh(x,y,z)
```



8 La manipulation des polynômes

En mathématiques, un polynôme est complètement défini par ses coefficients. Ainsi, le polynôme $p(x) = x^3 - 5x - 3$ *est défini par les coefficients* 1 *(pour* x^3*), 0 (pour* x^2*), -5 (pour* x*) et* -3*. En Octave, un polynôme est donc défini par la liste de ses coefficients dans l'ordre décroissant des puissances. Pour l'exemple, on a*

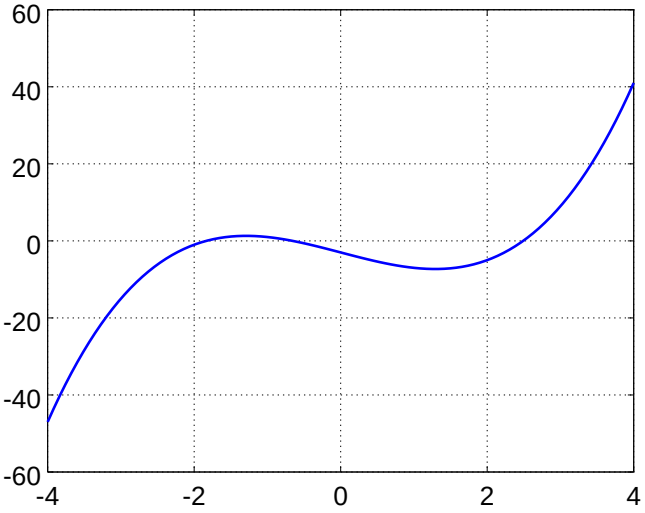
```
octave:30> p = [1 0 -5 -3] ;
```

Pour calculer la valeur du polynôme pour un certain argument, on dispose de l'instruction `polyval`*, dont l'argument peut être un nombre isolé ou une liste de nombres. Examiner l'exemple suivant:*

```
octave:31> x = 1.45 ;
octave:> polyval(p,x)
ans = -7.2014
octave:32> x = [1.4 1.5 1.6 1.7] ;
octave:33> polyval(p,x)
ans =
  -7.2560  -7.1250  -6.9040  -6.5870
```

Le dessin du graphe du polynôme se fait de manière analogue à celui de n'importe quelle fonction:

```
octave:34> x = linspace(-4,4,100) ;
octave:35> y = polyval(p,x) ;
octave:36> plot(x,y)
octave:37> grid on
```



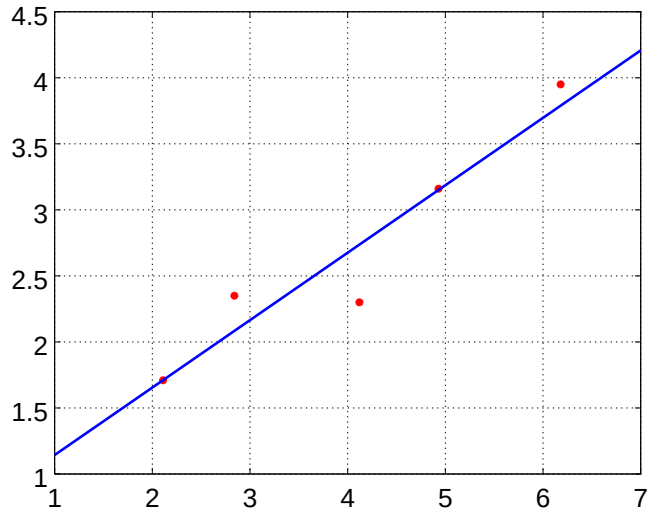
Il existe plusieurs instructions spécifiques relatives aux calculs avec des polynômes. On peut citer par exemple l'instruction `roots()` *qui calcule les zéros du polynôme:*

```
octave:39> roots(p)
ans =
   2.49086
  -1.83424
  -0.65662
```

La régression est l'une des applications les plus utiles des polynômes en calcul numérique. Etant donné une liste de points dans le plan (obtenus par exemple lors de mesures dans un laboratoire), on aimerait trouver l'équation de la "meilleure" droite possible qui approche ces points. Une droite est une fonction linéaire de la forme $y = ax + b$*, et l'exercice consiste donc à déterminer les coefficients* a *et* b *les "meilleurs possibles". L'instruction* `polyfit()` *réalise ce travail. Pour l'utiliser, on doit lui donner comme arguments les listes des coordonnées* x *et* y *des points, et le degré du polynôme que nous cherchons (*1 *pour une droite). Il est donc aussi possible d'obtenir des polynômes de régression de plus haut degré. Examiner l'exemple suivant:*

```
octave:41> xp = [2.11 2.84 4.12 4.93 6.18] ;
octave:41> yp = [1.71 2.35 2.30 3.16 3.95] ;
octave:42> p = polyfit(xp,yp,1)
p =
  0.51058  0.63331

octave:43> x = linspace(1,7,50) ;
octave:44> y = polyval(p,x) ;
octave:45> plot(xp,yp,'r*',x,y)
octave:46> grid on
```



9 Equations

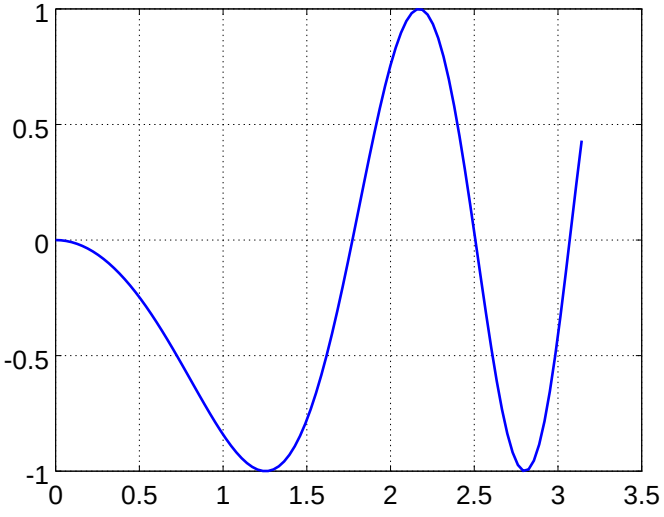
Les langages Octave et Matlab ont été développés en premier lieu pour le calcul matriciel et vectoriel. Ils sont donc particulièrement efficaces dans la résolution de systèmes d'équations linéaires, sous forme matricielle. Ce point a déjà été présenté ci-dessus dans la section "Vecteurs et matrices". La documention spécifique donne un grand nombre d'informations supplémentaires sur les multiples possibilités de résoudre des systèmes d'équations linéaires, même si ceux-ci sont très grands.

9.1 Equation non linéaire à une inconnue

Dans le cas d'une équation non linéaire, la méthode de résolution est toute différente. En Octave, c'est l'instruction `fsolve()` *qui exécute cette tâche.*

Regardons pour commencer une équation à une seule variable. Pour résoudre cette équation, il faut d'abord l'écrire sous la forme canonique $f(x) = 0$*, en supposant que* x *est l'inconnue. Sous cette forme, la résolution de l'équation revient à déterminer le ou les zéro(s) de la fonction. La méthode pour obtenir un zéro d'une fonction est un algorithme itératif, qui fournit une seule solution à la fois. L'utilisateur doit donc donner à l'algorithme une valeur de départ, doit être choisie dans une région suffisamment proche de la solution cherchée.*

Considérons l'exemple suivant: trouver les zéros de la fonction non linéaire $f(x) = \sin(x^2 + \pi)$ *dans l'intervalle* $[0, \pi]$*. Le graphe de cette fonction est donné dans la figure ci-dessous:*



Observant le graphique, on voit qu'un zéro de la fonction est localisé approximativement au voisinage de $x = 1.7$, qu'un autre est localisé au voisinage de $x = 2.5$ et un troisième au voisinage de $x = 3$. La suite de l'exemple consiste à déterminer ces zéros avec une précision beaucoup plus grande. On commence par déclarer la fonction dans un fichier–fonction, nommé `f03.m`*. Le contenu de ce fichier est*

```
function y = f03(x)
y = sin(x.^2 + pi) ;
end
```

Les zéros cherchés sont alors obtenus au moyen des instructions suivantes:

```
octave:46> format long
octave:47> fsolve('f03',1.7)
ans = 1.77245385090552

octave:48> f03(ans)
ans =  -2.44921270764475e-16

octave:49> fsolve('f03',2.5)
ans = 2.50662827463096

octave:48> f03(ans)
ans =  1.90437563721974e-13

octave:49> fsolve('f03',3)
ans = 3.06998012383947
```

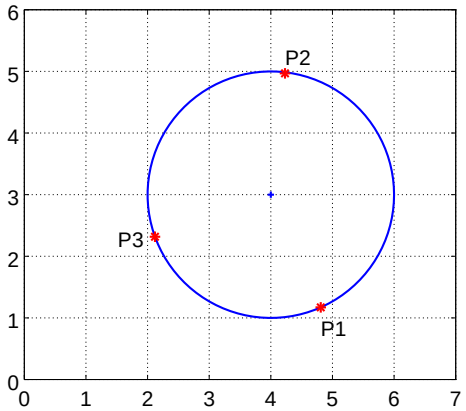
Dans les deux premier cas, on a aussi calculé pour contrôle la valeur de la fonction au point trouvé, et on voit bien que cette valeur est très proche de zéro.

9.2 Système d'équations non linéaires à plusieurs inconnues

Nous examinerons dans cette section l'exemple suivant: étant donnés trois points non alignés dans le plan, trouver le centre et le rayon du cercle qui passe par ces trois points. Résoudre ce problème revient à résoudre un système de trois équations non linéaires à trois inconnues. En effet, l'équation d'un cercle quelconque de centre (x_0, y_0) et de rayon R est

$$(x - x_0)^2 + (y - y_0)^2 = R^2$$

En écrivant cette équation pour trois points (x_k, y_k) donnés, pour $k = 1, 2, 3$, on obtient un système de trois équations pour les trois inconnues x_0, y_0 et R. Ce système est non linéaire à cause des carrés.



Prenons un exemple numérique (voir figure ci-dessus): soient les trois points $P_1(4.81, 1.171)$, $P_2(4.23, 4.968)$ et $P_3(2.12, 2.317)$. Ecrivant l'équation du cercle ci-dessus pour chacun des trois points, on obtient après une légère

transformation les équations suivantes:

$$\begin{array}{rcrcrcl} (4.81-x_0)^2 & + & (1.171-y_0)^2 & - & R^2 & = & 0 \\ (4.23-x_0)^2 & + & (4.986-y_0)^2 & - & R^2 & = & 0 \\ (2.12-x_0)^2 & + & (2.317-y_0)^2 & - & R^2 & = & 0 \end{array}$$

Ecrivant les trois lignes dans un vecteur, on obtient une fonction vectorielle à trois lignes dont il s'agit de trouver les zéros. Dans Octave, on définit un fichier–fonction `f06.m` *qui contient les trois lignes cette fonction vectorielle. Les inconnues sont appelées ici* `x(1)`, `x(2)` *et* `x(3)`*:*

```
% Fichier-fonction f06.m
function z = f06(x)
z = zeros(3,1) ;
z(1) = (4.81 -x(1)).^2 + (1.171 - x(2)).^2 - x(3).^2 ;
z(2) = (4.23 -x(1)).^2 + (4.986 - x(2)).^2 - x(3).^2 ;
z(3) = (2.12 -x(1)).^2 + (2.317 - x(2)).^2 - x(3).^2 ;
end
```

La méthode de résolution de ces équations est une méthode itérative. Elle nécessite des valeurs de départ pour les trois inconnues. Comme valeurs de départ, nous pouvons prendre par exemple le centre de gravité $S(3.72, 2.82)$ *du triangle déterminé par les trois points (comme approximation du centre du cercle) et le rayon approximatif* 1.8*. La résolution de ce système d'équations est donnée par l'instruction suivante:*

```
octave:50> fsolve('f06', [3.72 2.82 1.8])
ans =
  3.99983639831334
  2.99941877091002
  1.99987006160751
```

Selon le problème, il est possible que l'algorithme itératif de résolution des équations non linéaires ne converge pas. Il faut alors entrer une information supplémentaire dans l'instruction `fsolve`*: la matrice jacobienne (voir la théorie au cours de mathématiques). Dans notre cas, on écrit un fichier–fonction supplémentaire* `J06.m` *qui contient la matrice jacobienne:*

```
% Matrice jacobienne relative aux fonctionS f06.m
function z = J06(x)
z = [-2*4.81+2.*x(1) -2*1.171+2.*x(2) ;
     -2*4.23+2.*x(1) -2*4.986+2.*x(2) ;
     -2*2.12+2.*x(1) -2*2.317+2.*x(2)] ;
end
```

Pour résoudre l'équation, on utilise alors l'instruction suivante:

```
octave:51> [X, INFO, MSG] = fsolve(['f06' ; 'J06'], [3.72 2.82 1.8])
X =
  3.99983639831334
  2.99941877091002
  1.99987006160709

INFO = 1
MSG = solution converged within specified tolerance
```

Si le point de départ n'est pas bien choisi (trop éloigné de la solution), l'algorithme ne converge pas et la solution est fausse, comme le montre l'exemple ci-dessous:

```
octave:52> [X, INFO, MSG] = fsolve(['f06' ; 'J06'], [3.7 2.8 1.5])
X =
  3.70000000000000
  2.80000000000000
  1.50000000000000

INFO = 3
MSG = iteration is not making good progress
```

10 Exercices

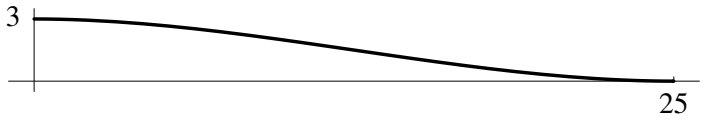
Exercice 10.1 Calculer le volume exact V et la surface extérieure S (sans la base) de la pyramide de Chéops. La pyramide a une base B carrée 227 m de côté et une hauteur h de138 m. Quelle serait la longueur d'un mur de 1 m de hauteur, 0.5 m de largeur et de volume équivalent ? Indication: $V = \frac{1}{3}Bh$.

Exercice 10.2 Soit la fonction $f(t) = \sin(\omega t)e^{-\alpha t}$, avec $\omega = 10\pi$ et $\alpha = 2$. Représenter la fonction $f(t)$ sur un graphique pour $0 \leq t \leq 1$. Représenter sur la même figure les enveloppes $g(t) = \pm e^{-\alpha t}$.

Exercice 10.3 Soit la fonction $f(t) = e^{-t^2}\left[2 - \sin(8t^2)\right]$.

1. Pour $N = 20$ et $N = 100$, générer un échantillonnage de longueur N de la fonction $f(t)$ dans l'intervalle $[0, 2]$, de la forme
$$E = \{f(0), f(\Delta t), f(2\Delta t), \ldots, f(N\Delta t)\}$$
avec $\Delta t = \frac{2}{N}$.
2. Faire une représentation graphique de l'échantillon. Utiliser par exemple l'instruction `stem()` au lieu de `plot()`.

Exercice 10.4 Dans sa manœuvre d'approche d'un aéroport, un avion suit une trajectoire donnée par le polynôme du troisième degré $p(x) = a\,x^3 + b\,x^2 + c\,x + d$. La pente de cette courbe est donné par $p'(x) = 3\,a\,x^2 + 2\,b\,x + c$.



A 25 km de la piste de l'aéroport, l'avion est en vol horizontal à l'altitude de 3000 m. Il suit alors la courbe donnée par $p(x)$, qui lui permet d'arriver exactement au point d'atterrissage, sans qu'il n'y ait aucune secousse au moment de l'impact.

Calculer les valeurs des coefficients a, b, c et d du polynôme $p(x)$ et faire un graphique de la trajectoire avec *Octave*.
Indication: système d'équations.

Exercice 10.5 La fonction $f(x, t) = e^{-0.1t}\sin(x - vt)$ représente une onde sinusoïdale qui se déplace à la vitesse v dans la direction x et qui s'amortit avec le temps. Générer un graphique de cette fonction, pour $0 \leq x \leq 10$ et pour $0 \leq t \leq 20$. Choisir différentes valeurs pour v (par exemple: $v = 1$).

---

11 Régression linéaire de la résistance en fonction de la température

11.1 Explication du problème

La résistance électrique du cuivre dépend très fortement de la température T. Cette dépendance peut être mesurée: pour le domaine de températures relativement basses, il s'agit d'une fonction affine, comme le montrent aussi les considérations théoriques. La résistance R s'exprime donc comme fonction de la température T par la formule

$$R(T) = p_1 \cdot T + p_2$$

Le but de cette section consiste à lire les valeurs mesurées de p_1 et p_2 dans un fichier de données où elles ont été sauvées, d'exécuter les instructions de calcul adéquates, et de visualiser les données.

11.2 Lecture des données dans un fichier et leur représentation

Le fichier de données `copper.dat` contient dans deux colonnes les valeurs mesurées de la température T et de la résistance R. La première colonne montre que la température augmente de $20°C$ à $80°C$, alors que la deuxième colonne montre l'évolution de la résistance, qui augmente de $4.36\ \Omega$ à $5.37\ \Omega$.

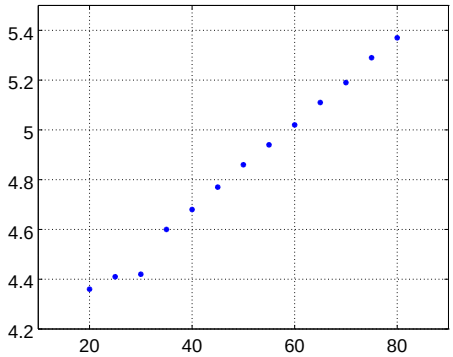
copper.dat

```
20 4.36
25 4.41
30 4.42
35 4.60
40 4.68
45 4.77
50 4.86
55 4.94
60 5.02
65 5.11
70 5.19
75 5.29
80 5.37
```

Le code ci-dessous va lire les donnés et puis afficher dans un graphique.

```
data = dlmread('copper.dat'); % read the data
k = length(data);
T = data(:,1); R = data(:, 2);

disp(sprintf('there are %d numbers of lines with data',k))
clf
plot(T,R,'*');          %% generate a basic plot
axis([10 90 4.2 5.5])
```



Le graphique montre que les données ont été lues correctement.

11.3 Exécution d'une régression linéaire au moyen de l'instruction `polyfit()`

Pour exprimer la résistance comme fonction linéaire de la température, on peut utiliser l'instruction `polyfit()` *(voir aussi section 8, p. 8). On cherche donc les valeurs optimales des deux composantes du vecteur* $\vec{p} = (p_1, p_2)$ *telles que*

$$R(T) \approx p_1\, T + p_2$$

Le résultat de l'instruction Octave

```
p = polyfit(T,R,1)
```

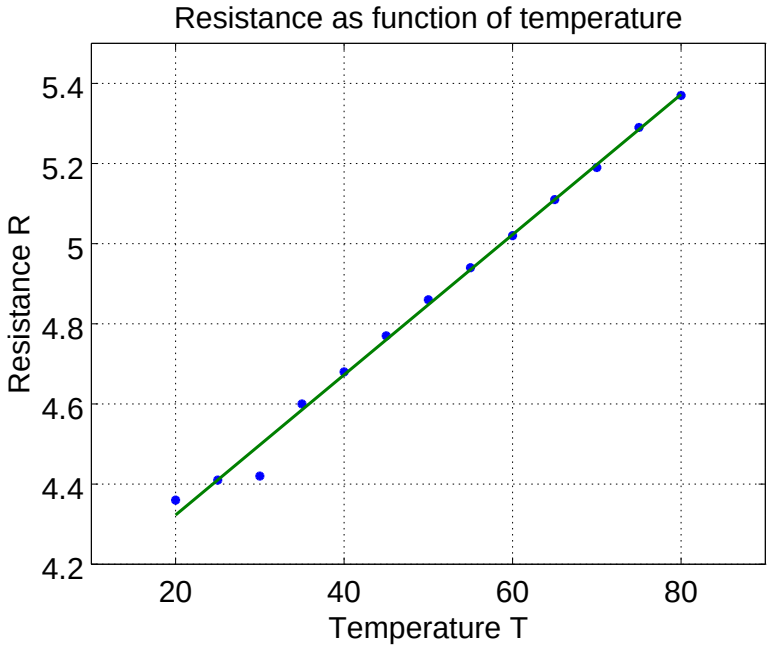
montre que

$$R(T) \approx 0.017495 \cdot T + 3.972967$$

11.4 Visualisation

A présent, il nous faut encore représenter les points mesurés et la droite de régression dans une figure unique, comprenant aussi un titre et une description des axes. Le graphique obtenu, présenté ci-dessous, pourra aussi être sauvé comme fichier PostScript pour une utilisation future éventuelle dans un rapport de mesures.

```
p = polyfit(T,R,1)
Rfit = polyval(p,T);            %% compute the fitted resistances
plot(T,R,'*',T,Rfit);           %% generate a basic plot
title('Resistance as function of temperature')
grid on
axis([10,90,4,5.6])
xlabel('Temperature T');
ylabel('Resistance R');
```



11.5 Calculs avec l'instruction `LinearRegression()`

Pour obtenir des informations plus détaillées sur la droite de régression linéaire, on peut utiliser l'instruction[1] *Octave* `LinearRegression()`*. Puisque la droite cherchée a la forme*

$$R(T) = p_1 \cdot T + p_2 \cdot 1$$

[1] Le fichier-fonction `LinearRegression.m` doit au préalable être copié dans le répertoire de travail, car cette instruction ne fait pas (encore) partie d'*Octave*.

il faut d'abord construire la matrice

$$\mathbf{F} = \begin{bmatrix} T_1 & 1 \\ T_2 & 1 \\ T_3 & 1 \\ \vdots & \vdots \\ T_n & 1 \end{bmatrix}$$

On peut alors déterminer les valeurs estimées des coefficients p_1 et p_2, ainsi que l'écart-type correspondant à ces estimations.

```
F = ones(k,2);
F(:,1) = T;
[p,y_var,r,p_var] = LinearRegression(F,R);
values = p
standarddev = sqrt(p_var)
```

Les valeurs obtenues pour p_1 et p_2 correspondent évidemment aux valeurs obtenues dans la section précédente. Les écarts-types calculés conduisent à des estimations d'erreurs:

$$\begin{aligned} p_1 &= 0.017495 \pm 0.00040 \\ p_2 &= 3.972967 \pm 0.02143 \end{aligned}$$

Selon les valeurs des écarts-types donnés ci-dessus, les coefficients donnés sont écrits avec trop de chiffres, et le résultat peut être écrit de manière plus consistante sous la forme

$$p_1 = 0.0175 \pm 0.0004 \qquad \textit{et} \qquad p_2 = 3.97 \pm 0.02$$

11.6 Régression par une parabole

Pour trouver une parabole de la forme générale

$$R(T) = p_1 \cdot T^2 + p_2 \cdot T + p_3 \cdot 1$$

qui s'adapte le mieux possible aux points donnés, on doit d'abord construire la matrice

$$\mathbf{F} = \begin{bmatrix} T_1^2 & T_1 & 1 \\ T_2^2 & T_2 & 1 \\ T_3^2 & T_3 & 1 \\ \vdots & \vdots & \vdots \\ T_n^2 & T_n & 1 \end{bmatrix}$$

Ensuite, on exécute le code

```
F = ones(k,3);
F(:,1)=(T.^2); F(:,2)=T;
[p,y_var,r,p_var] = LinearRegression(F,R);
values = p
standarddev = sqrt(p_var)
Rquad = F*values;
plot(T,R,'*',T,Rfit,T,Rquad);
```

qui produit les résultats

$$\begin{aligned} p_1 &= 3.1968 \cdot 10^{-6} \pm 2.5373^{-5} \\ p_2 &= 0.0172 \pm 0.003 \\ p_3 &= 3.9798 \pm 0.006 \end{aligned}$$

On peut remarquer les faits suivants:

- *Le terme quadratique $p_1 \cdot T \approx 3.1968 \cdot 10^{-6} \cdot T$ est très petit en comparaison des deux autres termes.*
  - *L'ordre de grandeur de la température T est $T \approx 80 \approx 100 = 10^2$.*
  - *L'ordre de grandeur du terme quadratique est $p_1 \cdot T^2 \approx 10^{-6} \cdot 10^{2 \cdot 2} = 10^{-2}$.*
  - *L'ordre de grandeur du terme linéaire est $p_2 \cdot T \approx 10^{-2} \cdot 10^2 = 1$.*
  - *L'ordre de grandeur du terme constant est $p_3 \approx 4$.*
- *L'écart-type de p_1 est considérablement plus grand que la valeur de p_1 elle-même.*
- *Les valeurs des coefficients du terme constant p_3 et du terme linéaire p_2 n'ont pas beaucoup varié (par rapport à la droite de régression), seuls les écarts-types ont des nouvelles valeurs du même ordre de grandeur.*

Les observations ci-dessus confirment que le fait de choisir une droite comme courbe pour approcher les points donnés est un bon choix. Ce fait peut aussi être confirmé par le graphique généré par le code ci-dessus. Celui-ci représente les données brutes, la droite de régression linéaire, et la parabole de régression. Visuellement, on ne peut pas constater de différence entre les deux courbes.

12 La régression appliquée à un problème de conduction de chaleur

12.1 Explication du problème

On plonge un corps métallique froid dans un très grand bassin rempli d'eau chaude à la température T_0. Le corps métallique va progressivement se réchauffer et, au fil du temps, atteindre aussi la température T_0. La "loi de refroidissement de Newton" affirme que la température T du corps est donnée en fonction du temps par

$$T(t) = T_0 - c \cdot e^{-\alpha\, t}$$

pour des constantes appropriées c et α. La valeur α permet de déterminer la capacité calorifique du corps. Pour cela, nous allons dans la suite analyser un jeu de données numériques, issues de mesures relatives à ce problème.

12.2 Lecture des données et leur représentation

Les données ont été enregistrées par un programme de mesures dans le fichier `Messdaten.prn`, dont voici les premières lignes:

Messdaten.prn

```
-0.120000004768372      61.7662492      8.01224287
2.95999991893768        61.758956       8.02274545
4.87999999523163        61.7564654      8.00599555
6.9099999666214         61.7493545      7.96234964
8.93999993801117        61.752114       7.99515274
10.8700000047684        61.7550227      8.23236488
...
```

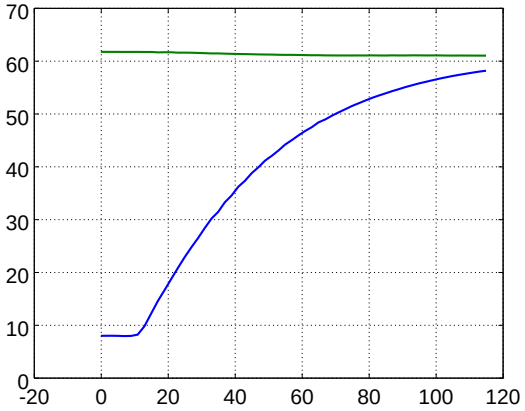
La première colonne donne les valeurs du temps t, la seconde colonne la température T_0 de l'eau du bassin, et la troisième colonne la température du corps métallique. De manière analogue à la procédure décrite dans le chapitre précédent, les données seront d'abord lues puis un graphique sera généré.

```
data = dlmread('Messdaten.prn');
k = length(data);
disp(sprintf('there are %d numbers of lines with data',k))
t  = data(:,1);
T  = data(:,3);
T0 = data(:,2);

plot(t,T,t,T0)
```

Le graphique résultant est montré ci-dessous. Un certain nombre d'effets peuvent être constatés:

- *La température de l'eau du bassin est pratiquement constante. Dans la suite des calculs, nous remplacerons cette température par la moyenne de toutes les valeurs mesurées.*
- *Qualitativement, la température $T(t)$ du corps métallique a un comportement correct, à l'exception des 13 premières secondes. Dans cette phase initiale, le corps n'était pas encore plongé dans l'eau chaude et il a donc conservé sa température originale. Cette phase initiale est à éliminer du jeu des données.*



12.3 Préparation des données

En premier lieu, il faut déterminer la température moyenne T_0 de l'eau du bassin. Ensuite, on détermine à partir de quel point dans la liste les valeurs du temps t sont plus grandes que 13. Pour la suite de l'analyse, seules ces données seront considérées. Les calculs sont contrôlés au moyen d'un graphique.

```
T0 = mean(T0);     %% replace the tepmerature of the bath by its average value

maxIndex = max(find(t<13)); %% find the maximal index for which t<13
t = t(maxIndex:end);        %% examine only t>13
T = T(maxIndex:end);
plot(t,T0-T)                %% visual verification of the result
```

La loi de refroidissement de Newton affirme que

$$\begin{aligned} T(t) &= T_0 - c\,e^{-\alpha\,t} \\ T_0 - T(t) &= c\,e^{-\alpha\,t} \\ y = \ln(T_0 - T(t)) &= \ln(c) - \alpha\,t \end{aligned}$$

Considérant $y = \ln(T_0 - T(t))$ comme fonction du temps t, nous devrions obtenir une droite de pente $-\alpha$. Ceci est confirmé par le graphique résultant du code Octave ci-dessous.

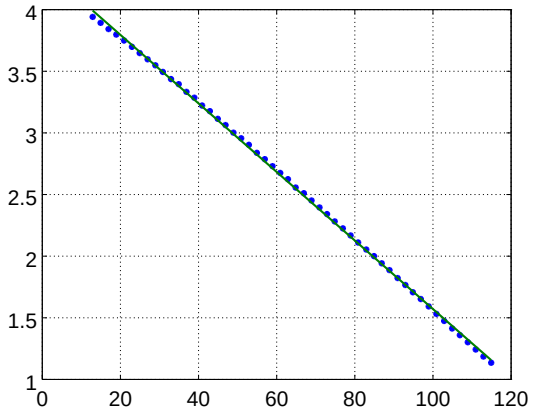
```
y = log(T0-T);
plot(t,y)
```

12.4 Régression linéaire et visualisation des résultats

Dans cette section, nous allons déterminer les paramètres de cette droite au moyen d'une régression linéaire. La qualité de cette régression sera contrôlée graphiquement.

```
F = ones(length(t),2);
F(:,1) = t;
[p,y_var,r,p_var] = LinearRegression(F,y);
values = p
variances = sqrt(p_var)

yfit = F*p;
plot(t,y,'*',t,yfit);
```



Les valeurs numériques donnent les résultats suivants:

$$\alpha \approx -0.02780 \pm 0.0009 \qquad \textit{et} \qquad \ln(c) \approx 4.351 \pm 0.006$$

A partir de ces valeurs, la courbe originale peut être reconstruite:

$$T(t) = T_0 + c\, e^{-\alpha\, t} = T_0 + e^{\ln(c)}\, e^{-\alpha\, t}$$

Le graphique résultant montre que ces calculs fournissent des résultats corrects.

```
Tfit = T0-exp(values(2))*exp(values(1)*t);

plot(t,T,'*',t,T0*ones(size(t)),t,Tfit);
title('Temperatures as function of time')
grid on
axis([0,120,0,70])
xlabel('Time t'); ylabel('Temperatures T and T0');
print('./../HeatFit.png');
```