ECOLE NORMALE SUPÉRIEURE DE
L'ENSEIGNEMENT TECHNIQUE
MOHAMMEDIA
UNIVERSITÉ HASSAN II DE CASABLANCA

ENSET

المدرسة العليا لأساتذة التعليم التقني
المحمدية
جامعة الحسن الثاني بالدار البيضاء

Travaux pratiques

Outils de simulation des systèmes industriels

2ème année FI-GIL

Pr Moulay El houssine ECH-CHHIBAT

Département Génie mécanique

ENSET Mohammedia

Université Hassan II de Casablanca

Adresse : BP 159 Bd Hassan II,
Mohammedia - Maroc
Tél : 0523322220
Fax : 0523322546
Site web : www.enset-media.ac.ma

العنوان : صندوق البريد 159، شارع
الحسن الثاني، المحمدية
الهاتف : **0523322220**
الفاكس : **0523322546**
www.enset-media.ac.ma

TP1 Matlab

GIL2

1) Ouvrir MATLAB
2) Créer deux matrices `A` et `B` de 3×3 dans la fenêtre de commande
3) Créer un vecteur colonne `a` et un vecteur ligne `b` de longueur 3
4) Calculer les transposées des matrices `A`, `B` et des vecteurs `a`, `b`.
5) Calculer `A + B` et `A - B`.
6) Calculer `A * B` et `A .* B`, commenter
7) Calculer les vecteurs `x = A \ b` et `r = A * x`.
8) Calculer les vecteurs `x = a / A` et `r = x * A`.
9) Calculer les matrices `X = A / B` et `X = A \ B`, vérifier `A*X`, `X*A`, `B*X` et `X*B`
10) Calculer les matrices `X = A ./ B` et `X = A .\ B`, vérifier `A*X`, `X*A`, `B*X` et `X*B`
11) Calculer les matrices `R = A ^ 2`, `R = A .^2` et `R = A .^B`, commenter
12) Calculer `(A'\a')'` et `R = (b'\A')'`, commenter
13) Calculer `inv(A)*b` et `a*inv(A)` , commenter
14) Calculer `a*b` , `b*a`, `b'*a`, `a'*b`, `b'*a'`, commenter
15) Calculer `a.*b` , `b.*a`, `b'.*a`, `a'.*b`, `b'.* a'`, commenter
16) Créer trois fonctions pour les trois rotations 3D autour des axes *x*, y et *z*. Calculer une matrice de rotation `R` composée d'une rotation de 30° autour de *x* de 60° autour de *y* et de 45° autour de *z*. Calculer ensuite `Ar = R'*A*R`.
17) Comparer `trace(A)`, `trace(Ar)`, et `eig(A)`, `eig(Ar)`. Commenter
18) Créer un fichier de commandes à l'aide de l'éditeur MATLAB et refaire 2 à 10.
19) Créer un fichier texte contenant les éléments d'une matrices 3×3 et charger la matrice à l'aide d'une ligne de commande.

Application :

Créer la matrice A en utilisant la fonction `diag` et un vecteur b avec un pas fixe

```
A =
     1     1     0
     1     2     2
     0     2     3

 b =
     1
     2
     3
```

Résoudre le système d'équations $A\,x = b$ en utilisant :

1) L'opérateur de division,
2) La décomposition LU,
3) La décomposition QR.

Utiliser `help` pour les fonctions `diag`, `LU` et `QR`. Exemple `help lu`.

TP 2 : Polynômes et courbes

GIL 2

1. Introduction

Le but de ce TP est de voir les fonctions permettant de manipuler les polynômes, de tracer des courbes et de réaliser des approximations de données.

2. Polynômes

a) Généralités

Trouver les racines d'un polynôme $P(x)$ consiste à chercher les valeurs de x qui annulent ce polynôme

Matlab représente un polynôme comme un vecteur ligne contenant les coefficients du polynôme rangés par ordre décroissant d'exposant.

Par exemple, en lieu et place de $P(x) = x^4 - 12x^3 + 25x + 116$,

on écrit le vecteur $P = [1 \quad -12 \quad 0 \quad 25 \quad 116]$

On obtient alors instantanément les racines d'un polynôme avec la commande **roots.**

Regardez les possibilités de cette commande avec la fenêtre d'aide.

Dans Matlab, un polynôme et ses racines sont des vecteurs Le polynôme étant un vecteur uniligne, et les racines étant un vecteur unicolonne.

b) Application

Valeur en un point

Dans un script intitulé "poly1.m" :

- Créez le polynôme $P(x)$ précédent

- A l'aide de la fenêtre d'aide, déterminez les fonctionnalités de la commande **polyval**
- Déterminez la valeur de $P(x)$ en $x=1$ et affichez-la
- Créez un vecteur *V* dont les composantes sont les valeurs de 0 à 20 par pas de 0.5
- Calculez les valeurs de $P(x)$ aux points contenus dans le vecteur *V* et affichez les
- Affichez ces valeurs dans un graphique pour connaître l'allure de ce polynôme

Recherche de racines

- Utilisez la fenêtre d'aide pour analyser la commande **roots**

Soient les deux polynômes suivants :

$$A(x) = x^3 + 2x^2 + 3x + 4 \quad \text{et} \quad B(x) = x^3 + 4x^2 + 9x + 16$$

Dans un script intitulé "poly2.m" :

- Créez les 2 polynômes $A(x)$ et $B(x)$ précédents
- Tracez leurs courbes dans une figure simple, sur l'intervalle $[-10,10]$
- Trouvez leurs racines et affichez les
- Vérifiez ces valeurs à l'aide de **polyval**

Addition

- Ajoutez au script précédent le calcul de la somme $S(x)$ de $A(x)$ et de $B(x)$
- Tracez le résultat obtenu

Multiplication

- Utilisez la fenêtre d'aide pour analyser la commande **conv**
- Ajoutez au script précédent le produit $C(x)$ de $A(x)$ et de $B(x)$
- Tracez le résultat obtenu
- Vérifiez à l'aide du calcul théorique

Division

- Servez vous de la fenêtre d'aide pour découvrir la commande **deconv**
- Ajoutez au script précédent le calcul de la division de $C(x)$ par $B(x)$
- Tracez le résultat obtenu

3. Tracé de courbes

a) Introduction

La visualisation de données sous forme de courbes est un point important.
Lorsqu'une fonction est étudiée, comme cela est le cas en analyse numérique, il n'est pas très intéressant de lister l'ensemble des valeurs prises par la fonction. Il est plus pratique et plus lisible de tracer la courbe, comme nous venons de le voir.

Sous Matlab, l'affichage d'un graphique nécessite d'ouvrir une fenêtre graphique, ceci est réalisé par la commande **figure**.
Il est possible de redimensionner et de déplacer cette fenêtre de façon classique.
A chaque fenêtre est affecté un numéro identificateur (1 pour le première, 2 pour la seconde, etc.). La fermeture des fenêtres se fait alors avec la commande **close**.

Exemple :

- Ouvrir trois fenêtres graphiques (commande **figure**)
- Les déplacer et les redimensionner à loisir
- Fermer la seconde en utilisant la commande **close**
- Fermer tous les fenêtres graphiques restantes

b) Graphique

Comme nous l'avons déjà vu la commande plot permet de tracer un vecteur sous forme graphique.
La syntaxe la plus simple, que vous avez utilisé dans la première partie du TP, est la suivante : **plot(V)**, où V est un vecteur ligne ou colonne, peu importe).

Le graphique s'affiche alors sous forme de lignes brisées reliant les points définis par une abscisse et une ordonnée. Pour chaque point, l'abscisse est l'indice de l'élément du vecteur V et l'ordonnée est la valeur de l'élément.

Une fois l'affichage effectué, des commentaires peuvent s'ajouter au graphique : un titre (**title**), la légende de l'axe des abscisses (**xlabel**) et de l'axe des ordonnées (**ylabel**).

Le graphique complet peut alors être éventuellement sauvegardé en allant dans le menu **File/save_as**.

Une grille peut aussi être ajoutée à l'aide de la commande **grid**.

Application

- Créez un script "graphesin.m" réalisant les actions suivantes :
  - Demande à l'utilisateur de trois valeurs : *min*, *max* et *pas*
  - Génération d'un vecteur *X* contenant les points compris entre min et max avec le pas spécifié
  - Calcul du vecteur *Y* associé à la fonction $f(x) = \sin(x)$
  - Tracé du graphique correspondant à la fonction $f(x)$
  - Ajout d'un titre et des légendes des axes des *x* et des *y* par demandes à l'utilisateur
  - Ajout ou pas d'une grille par questionnement
  - Sauvegarde sur demande du graphique dans le répertoire sous un nom à demander
  - Fermeture de toutes les fenêtres graphiques

Après avoir créez plusieurs graphiques, testez la commande **open**

c) Application aux polynômes

- Créez un script "graphepoly.m" réalisant les actions suivantes :
  - Création d'un vecteur *X* contenant 100 points compris entre -1 et 3
  - Calcul des valeurs du polynôme $P(x)$ sur l'intervalle $[-1,3]$ dans un vecteur *V*
  - Affichage de la courbe ainsi obtenue.
  - Ajout du titre et des légendes aux axes
  - Création d'un fenêtre graphique pouvant contenir 2 sous fenêtres
  - Tracé de la courbe de $P(x)$ dans la première sous fenêtre.
  - Calcul de la dérivée *dP* du polynôme $P(x)$ (commande **polyder**)
  - Tracé dans la seconde sous fenêtre graphique de la courbe *dP* sur $[-1,3]$

- Déterminez les extremums du polynôme $P(x)$ sur l'intervalle $[-1,3]$.
- Affichez leurs valeurs

TP3 :Approximation de données
GIL2

a) Fonction polyfit

Une fonction très utile dans les laboratoires est la fonction **polyfit(x,y,n)** qui donne un polynôme qui approxime les données.

Dans l'instruction, on donne les valeurs respectives de *x* et de *y*, puis on indique par *n* le degré du polynôme qui approximera les données.
Polyfit retourne alors le polynôme d'approximation par le biais d'un vecteur contenant les coefficients des puissances descendantes.

Ainsi, en appelant :

Polyfit(x,y,1) : on obtient la pente de la droite et la valeur à l'origine

Polyfit(x ;y,2) : donne de plus le coefficient de l'équation quadratique

Pour tracer la courbe par la suite, il suffit d'utiliser les coefficients du polynôme pour le recréer et le tracer à l'aide de **plot** ou de **fplot**

b) Curve fitting

Si on connaît préalablement le degré du polynôme que sont supposées former les données, cette méthode est excellente. Par contre, si on ne connaît pas exactement le degré à employer, une seconde manière d'approximer s'impose.

Cette manière consiste à tracer les données désirées.
Une fois la figure produite, il faut aller dans le menu **App/Curve Fitting**.
Par la fenêtre qui apparaît, il est possible de sélectionner le groupe de données désiré, de sélectionner le ou les approximations désirées selon le degré du polynôme.
De plus, il est possible de montrer l'équation de chaque approximation.
On peut aussi indiquer sur un graphique la différence entre chaque donnée et sa valeur calculée avec l'approximation.

En cliquant en bas à gauche de la fenêtre, les résultats numériques sont disponibles.
Dans ces résultats, on trouve la fonction théorique du polynôme, ses coefficients, ainsi que la norme des différences entre chaque donnée et sa valeur calculée avec l'approximation.
De plus, on peut enregistrer ces donnés dans des variables disponibles dans le Workspace.

En étendant la fenêtre une seconde fois avec la flèche, on peut calculer f(x) d'une valeur ou d'un groupe de valeurs, selon l'équation sélectionnée.

Enfin, si cette méthode a été employée et que vous désirez conserver le graphique, n'oubliez pas de l'enregistrer afin de conserver tous les changements que vous venez d'apporter.

c) Application

- A partir des données suivantes correspondant à des mesures de courant pour différentes valeurs de tension, créez un script "ohm.m" produisant :
  - un graphique correspondant aux données expérimentales
  - une courbe associée au calcul de la résistance à chaque point
  - la courbe approximant ces résistances avec un polynôme d'ordre 7, sur le même graphique, avec les légendes

```
% mesures    tension,      courant
DATA = [      0             0
              20.5          2.03
              30.2          2.98
              39.8          4.02
              50.3          5.05
              60.0          6.01
              70.5          7.09
              80.0          8.00
              89.3          8.99
              100.5         9.97    ] ;
```

- Testez aussi ces données avec les différentes approximations du " Curve fitting"

Solve Differential Equation

Solve a differential equation analytically by using the `dsolve` function, with or without initial conditions. To solve a system of differential equations, see Solve a System of Differential Equations.

First-Order Linear ODE

Solve this differential equation.

$$\frac{dy}{dt} = ty.$$

First, represent y by using `syms` to create the symbolic function `y(t)`.

```
syms y(t)
```

Define the equation using `==` and represent differentiation using the `diff` function.

```
ode = diff(y,t) == t*y
```

```
ode(t) =
diff(y(t), t) == t*y(t)
```

Solve the equation using `dsolve`.

```
ySol(t) = dsolve(ode)
```

```
ySol(t) =
C1*exp(t^2/2)
```

Solve Differential Equation with Condition

In the previous solution, the constant `C1` appears because no condition was specified. Solve the equation with the initial condition `y(0) == 2`. The `dsolve` function finds a value of `C1` that satisfies the condition.

```
cond = y(0) == 2;
ySol(t) = dsolve(ode,cond)
```

```
ySol(t) =
2*exp(t^2/2)
```

If `dsolve` cannot solve your equation, then try solving the equation numerically. See Solve a Second-Order Differential Equation Numerically.

Nonlinear Differential Equation with Initial Condition

Solve this nonlinear differential equation with an initial condition. The equation has multiple solutions.

$$\left(\frac{dy}{dt} + y\right)^2 = 1,$$
$$y(0) = 0.$$

```
syms y(t)
ode = (diff(y,t)+y)^2 == 1;
```

```
cond = y(0) == 0;
ySol(t) = dsolve(ode,cond)
```

```
ySol(t) =
 exp(-t) - 1
 1 - exp(-t)
```

Second-Order ODE with Initial Conditions

Solve this second-order differential equation with two initial conditions.

$$\frac{d^2y}{dx^2} = \cos(2x) - y,$$
$$y(0) = 1,$$
$$y'(0) = 0.$$

Define the equation and conditions. The second initial condition involves the first derivative of `y`. Represent the derivative by creating the symbolic function `Dy = diff(y)` and then define the condition using `Dy(0)==0`.

```
syms y(x)
Dy = diff(y);

ode = diff(y,x,2) == cos(2*x)-y;
cond1 = y(0) == 1;
cond2 = Dy(0) == 0;
```

Solve `ode` for `y`. Simplify the solution using the `simplify` function.

```
conds = [cond1 cond2];
ySol(x) = dsolve(ode,conds);
ySol = simplify(ySol)
```

```
ySol(x) =
1 - (8*sin(x/2)^4)/3
```

Third-Order ODE with Initial Conditions

Solve this third-order differential equation with three initial conditions.

$$\frac{d^3u}{dx^3} = u,$$
$$u(0) = 1,$$
$$u'(0) = -1,$$
$$u''(0) = \pi.$$

Because the initial conditions contain the first- and second-order derivatives, create two symbolic functions, `Du = diff(u,x)` and `D2u = diff(u,x,2)`, to specify the initial conditions.

```
syms u(x)
Du = diff(u,x);
D2u = diff(u,x,2);
```

Create the equation and initial conditions, and solve it.

```
ode = diff(u,x,3) == u;
cond1 = u(0) == 1;
cond2 = Du(0) == -1;
cond3 = D2u(0) == pi;
conds = [cond1 cond2 cond3];

uSol(x) = dsolve(ode,conds)
```

```
uSol(x) =
```

```
(pi*exp(x))/3 - exp(-x/2)*cos((3^(1/2)*x)/2)*(pi/3 - 1) -...
(3^(1/2)*exp(-x/2)*sin((3^(1/2)*x)/2)*(pi + 1))/3
```

More ODE

This table shows examples of differential equations and their Symbolic Math Toolbox™ syntax. The last example is the Airy differential equation, whose solution is called the Airy function.

Differential Equation	**MATLAB® Commands**
$\frac{dy}{dt} + 4y(t) = e^{-t},$ $y(0) = 1.$	
$2x^2\frac{d^2y}{dx^2} + 3x\frac{dy}{dx} - y = 0.$	
The Airy equation. $\frac{d^2y}{dx^2} = xy(x).$	

Solve Differential Equation

Solve a differential equation analytically by using the `dsolve` function, with or without initial conditions. To solve a system of differential equations, see Solve a System of Differential Equations.

First-Order Linear ODE

Solve this differential equation.

$$\frac{dy}{dt} = ty.$$

First, represent y by using `syms` to create the symbolic function `y(t)`.

```
syms y(t)
```

Define the equation using `==` and represent differentiation using the `diff` function.

```
ode = diff(y,t) == t*y
```

```
ode(t) =
diff(y(t), t) == t*y(t)
```

Solve the equation using `dsolve`.

```
ySol(t) = dsolve(ode)
```

```
ySol(t) =
C1*exp(t^2/2)
```

Solve Differential Equation with Condition

In the previous solution, the constant `C1` appears because no condition was specified. Solve the equation with the initial condition `y(0) == 2`. The `dsolve` function finds a value of `C1` that satisfies the condition.

```
cond = y(0) == 2;
ySol(t) = dsolve(ode,cond)
```

```
ySol(t) =
2*exp(t^2/2)
```

If `dsolve` cannot solve your equation, then try solving the equation numerically. See Solve a Second-Order Differential Equation Numerically.

Nonlinear Differential Equation with Initial Condition

Solve this nonlinear differential equation with an initial condition. The equation has multiple solutions.

$$\left(\frac{dy}{dt} + y\right)^2 = 1,$$
$$y(0) = 0.$$

```
syms y(t)
ode = (diff(y,t)+y)^2 == 1;
```

```
cond = y(0) == 0;
ySol(t) = dsolve(ode,cond)
```

```
ySol(t) =
 exp(-t) - 1
 1 - exp(-t)
```

Second-Order ODE with Initial Conditions

Solve this second-order differential equation with two initial conditions.

$$\frac{d^2y}{dx^2} = \cos(2x) - y,$$
$$y(0) = 1,$$
$$y'(0) = 0.$$

Define the equation and conditions. The second initial condition involves the first derivative of `y`. Represent the derivative by creating the symbolic function `Dy = diff(y)` and then define the condition using `Dy(0)==0`.

```
syms y(x)
Dy = diff(y);

ode = diff(y,x,2) == cos(2*x)-y;
cond1 = y(0) == 1;
cond2 = Dy(0) == 0;
```

Solve `ode` for `y`. Simplify the solution using the `simplify` function.

```
conds = [cond1 cond2];
ySol(x) = dsolve(ode,conds);
ySol = simplify(ySol)
```

```
ySol(x) =
1 - (8*sin(x/2)^4)/3
```

Third-Order ODE with Initial Conditions

Solve this third-order differential equation with three initial conditions.

$$\frac{d^3u}{dx^3} = u,$$
$$u(0) = 1,$$
$$u'(0) = -1,$$
$$u''(0) = \pi.$$

Because the initial conditions contain the first- and second-order derivatives, create two symbolic functions, `Du = diff(u,x)` and `D2u = diff(u,x,2)`, to specify the initial conditions.

```
syms u(x)
Du = diff(u,x);
D2u = diff(u,x,2);
```

Create the equation and initial conditions, and solve it.

```
ode = diff(u,x,3) == u;
cond1 = u(0) == 1;
cond2 = Du(0) == -1;
cond3 = D2u(0) == pi;
conds = [cond1 cond2 cond3];

uSol(x) = dsolve(ode,conds)
```

```
uSol(x) =
```

```
(pi*exp(x))/3 - exp(-x/2)*cos((3^(1/2)*x)/2)*(pi/3 - 1) -...
(3^(1/2)*exp(-x/2)*sin((3^(1/2)*x)/2)*(pi + 1))/3
```

More ODE Examples

This table shows examples of differential equations and their Symbolic Math Toolbox™ syntax. The last example is the Airy differential equation, whose solution is called the Airy function.

Differential Equation	MATLAB® Commands

$\frac{dy}{dt} + 4y(t) = e^{-t},$ $y(0) = 1.$	`syms y(t)`
	`ode = diff(y)+4*y == exp(-t);`
	`cond = y(0) == 1;`
	`ySol(t) = dsolve(ode,cond)`
	
	`ySol(t) =`
	`exp(-t)/3 + (2*exp(-4*t))/3`

$2x^2\frac{d^2y}{dx^2} + 3x\frac{dy}{dx} - y = 0.$	`syms y(x)`
	`ode = 2*x^2*diff(y,x,2)+3*x*diff(y,x)-y == 0;`
	`ySol(x) = dsolve(ode)`
	
	`ySol(x) =`
	`C2/(3*x) + C3*x^(1/2)`

The Airy equation. $\frac{d^2y}{dx^2} = xy(x).$	`syms y(x)`
	`ode = diff(y,x,2) == x*y;`
	`ySol(x) = dsolve(ode)`
	
	`ySol(x) =`
	`C1*airy(0,x) + C2*airy(2,x)`

See Also

Solve a System of Differential Equations

Solve a System of Differential Equations

Solve a system of several ordinary differential equations in several variables by using the `dsolve` function, with or without initial conditions. To solve a single differential equation, see Solve Differential Equation.

- Solve System of Differential Equations
- Solve Differential Equations in Matrix Form

Solve System of Differential Equations

Solve this system of linear first-order differential equations.

$$\frac{du}{dt} = 3u + 4v,$$

$$\frac{dv}{dt} = -4u + 3v.$$

First, represent u and v by using `syms` to create the symbolic functions `u(t)` and `v(t)`.

```
syms u(t) v(t)
```

Define the equations using `==` and represent differentiation using the `diff` function.

```
ode1 = diff(u) == 3*u + 4*v;
ode2 = diff(v) == -4*u + 3*v;
odes = [ode1; ode2]
```

```
odes(t) =
 diff(u(t), t) == 3*u(t) + 4*v(t)
 diff(v(t), t) == 3*v(t) - 4*u(t)
```

Solve the system using the `dsolve` function which returns the solutions as elements of a structure.

```
S = dsolve(odes)
```

```
S =
  struct with fields:

    v: [1×1 sym]
    u: [1×1 sym]
```

If `dsolve` cannot solve your equation, then try solving the equation numerically. See Solve a Second-Order Differential Equation Numerically.

To access `u(t)` and `v(t)`, index into the structure `S`.

```
uSol(t) = S.u
vSol(t) = S.v
```

```
uSol(t) =
C2*cos(4*t)*exp(3*t) + C1*sin(4*t)*exp(3*t)
vSol(t) =
C1*cos(4*t)*exp(3*t) - C2*sin(4*t)*exp(3*t)
```

Alternatively, store `u(t)` and `v(t)` directly by providing multiple output arguments.

```
[uSol(t), vSol(t)] = dsolve(odes)
```

```
uSol(t) =
C2*cos(4*t)*exp(3*t) + C1*sin(4*t)*exp(3*t)
vSol(t) =
C1*cos(4*t)*exp(3*t) - C2*sin(4*t)*exp(3*t)
```

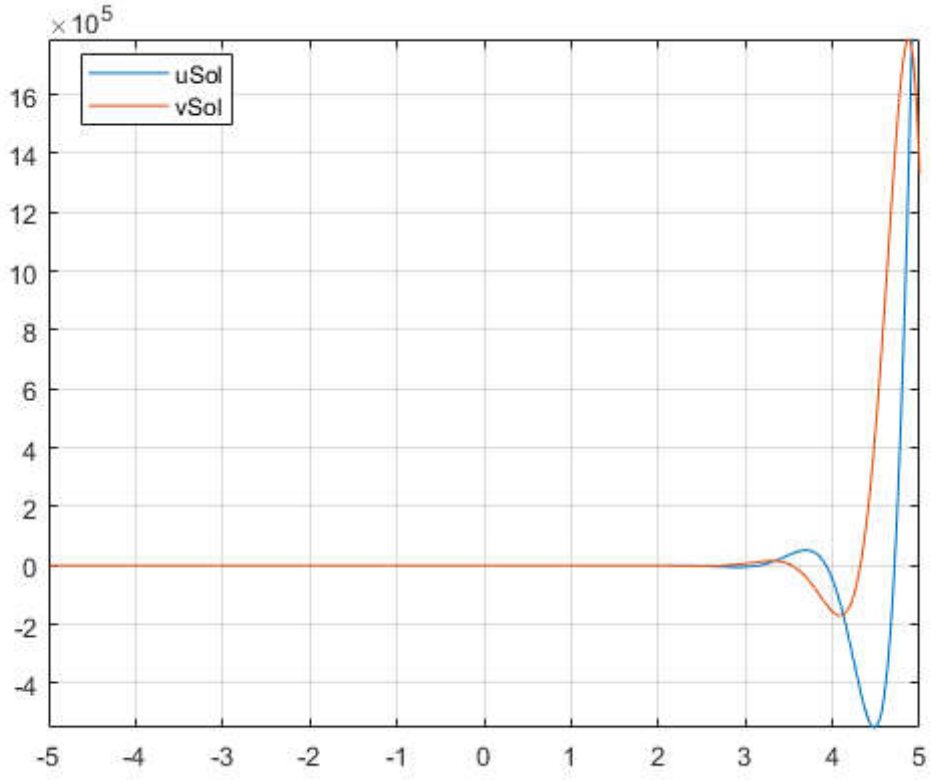
The constants `C1` and `C2` appear because no conditions are specified. Solve the system with the initial conditions `u(0) == 0` and `v(0) == 0`. The `dsolve` function finds values for the constants that satisfy these conditions.

```
cond1 = u(0) == 0;
cond2 = v(0) == 1;
conds = [cond1; cond2];
[uSol(t), vSol(t)] = dsolve(odes,conds)
```

```
uSol(t) =
sin(4*t)*exp(3*t)
vSol(t) =
cos(4*t)*exp(3*t)
```

Visualize the solution using `fplot`.

```
fplot(uSol)
hold on
fplot(vSol)
grid on
legend('uSol','vSol','Location','best')
```



Solve Differential Equations in Matrix Form

Solve differential equations in matrix form by using `dsolve`.

Consider this system of differential equations.

$$\frac{dx}{dt} = x + 2y + 1,$$

$$\frac{dy}{dt} = -x + y + t.$$

The matrix form of the system is

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 1 \\ t \end{bmatrix}.$$

Let

$$Y = \begin{bmatrix} x \\ y \end{bmatrix}, A = \begin{bmatrix} 1 & 2 \\ -1 & 1 \end{bmatrix}, B = \begin{bmatrix} 1 \\ t \end{bmatrix}.$$

The system is now $Y' = AY + B$.

Define these matrices and the matrix equation.

```
syms x(t) y(t)
A = [1 2; -1 1];
B = [1; t];
Y = [x; y];
odes = diff(Y) == A*Y + B
```

```
odes(t) =
  diff(x(t), t) == x(t) + 2*y(t) + 1
   diff(y(t), t) == t - x(t) + y(t)
```

Solve the matrix equation using `dsolve`. Simplify the solution by using the `simplify` function.

```
[xSol(t), ySol(t)] = dsolve(odes);
xSol(t) = simplify(xSol(t))
ySol(t) = simplify(ySol(t))
```

```
xSol(t) =
(2*t)/3 + 2^(1/2)*C2*exp(t)*cos(2^(1/2)*t) + 2^(1/2)*C1*exp(t)*sin(2^(1/2)*t) + 1/9
ySol(t) =
C1*exp(t)*cos(2^(1/2)*t) - t/3 - C2*exp(t)*sin(2^(1/2)*t) - 2/9
```

The constants `C1` and `C2` appear because no conditions are specified.

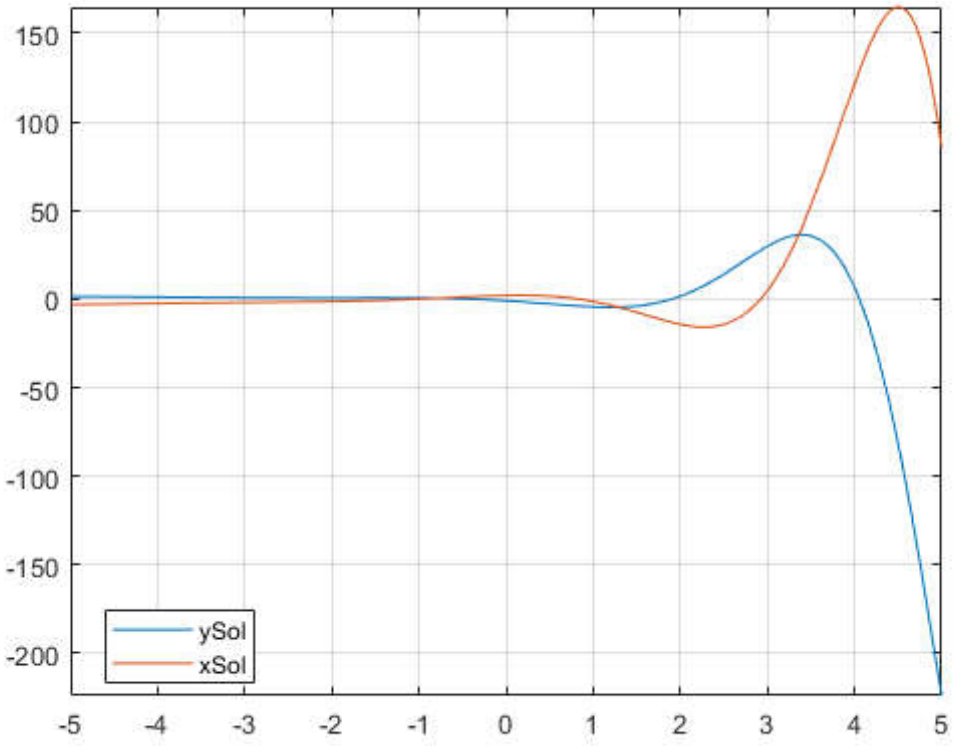
Solve the system with the initial conditions $u(0) = 2$ and $v(0) = -1$. When specifying equations in matrix form, you must specify initial conditions in matrix form too. `dsolve` finds values for the constants that satisfy these conditions.

```
C = Y(0) == [2; -1];
[xSol(t), ySol(t)] = dsolve(odes,C)
```

```
xSol(t) =
(2*t)/3 + (17*exp(t)*cos(2^(1/2)*t))/9 - (7*2^(1/2)*exp(t)*sin(2^(1/2)*t))/9 + 1/9
ySol(t) =
- t/3 - (7*exp(t)*cos(2^(1/2)*t))/9 - (17*2^(1/2)*exp(t)*sin(2^(1/2)*t))/18 - 2/9
```

Visualize the solution using `fplot`.

```
clf
fplot(ySol)
hold on
fplot(xSol)
grid on
legend('ySol','xSol','Location','best')
```


150
100
50
0
-50
-100
-150
-200
-5
-4
-3
-2
-1
0
1
2
3
4
5
ySol
xSol

TP 5 : Statistiques sous Matlab
Master GIL

Fonctions statistiques générales :

- **Statistiques descriptives** :
  - mean, median, mode → moyenne, médiane, mode
  - std, var, range → écart-type, variance, étendue
  - cov, corrcoef → covariance et corrélation
- **Quantiles et centiles** :
  - quantile, prctile
- **Cumulatives et glissantes** :
  - cumsum, cumprod, movmean, movstd
- **Tests statistiques** :
  - ttest, ttest2 → test t de Student
  - anova1, anova2 → analyse de variance
  - kruskalwallis → test non paramétrique
  - chi2gof → test du χ^2 d'ajustement
- **Lois de probabilité et tirages aléatoires** :
  - rand, randn → uniforme et normale
  - binornd, poissrnd, normpdf, normcdf

Fonctions pour histogrammes

- histogram(X) : histogramme simple
- histogram(X,nbins) : définir le nombre de classes
- histogram(X,'Normalization','pdf') : histogramme normalisé en densité
- histfit(X) : histogramme avec courbe de densité ajustée
- **Options avancées** : personnaliser les bords ('BinEdges'), les couleurs, superposer plusieurs distributions

Fonctions de régression

- **Régression linéaire** :
  - regress(y,X) → calcul des coefficients linéaires
  - fitlm(X,y) → modèle linéaire complet avec statistiques
- **Régression polynomiale** :
  - polyfit(x,y,n) → coefficients du polynôme d'ordre *n*
  - polyval(p,x) → évaluation du polynôme ajusté
- **Régression non linéaire et généralisée** :
  - fitnlm(X,y,model) → modèle non linéaire
  - glmfit(X,y,distribution) → modèles linéaires généralisés

Tableau récapitulatif

Domaine	Fonctions clés
Statistiques	mean, median, mode, std, var, cov, corrcoef, quantile, prctile, ttest, anova
Histogrammes	histogram, histfit, BinEdges, Normalization, superposition
Régression	regress, fitlm, polyfit, polyval, fitnlm, glmfit

Exemple:

```
% ================================
% Script MATLAB : Statistiques, Histogramme et Régression
% ================================
% Génération de données simulées
X = randn(200,1);   % 200 valeurs normales
Y = 2*X + 0.5 + randn(200,1); % relation linéaire avec bruit

%% --- Statistiques descriptives ---
disp('--- Statistiques descriptives ---');
disp(['Moyenne de X : ', num2str(mean(X))]);
disp(['Médiane de X : ', num2str(median(X))]);
disp(['Variance de X : ', num2str(var(X))]);
disp(['Écart-type de X : ', num2str(std(X))]);
disp(['Corrélation X-Y : ', num2str(corr(X,Y))]);

%% --- Histogramme avec courbe normale ajustée ---
figure;
histogram(X,'Normalization','pdf'); % histogramme normalisé
hold on;
mu = mean(X); sigma = std(X);
x_vals = linspace(min(X),max(X),100);
y_vals = normpdf(x_vals,mu,sigma);
plot(x_vals,y_vals,'r','LineWidth',2);
title('Histogramme de X avec courbe normale ajustée');
xlabel('Valeurs de X'); ylabel('Densité');
legend('Histogramme','Courbe normale ajustée');

%% --- Régression linéaire ---
figure;
scatter(X,Y,'filled');
hold on;
mdl = fitlm(X,Y); % modèle linéaire
plot(mdl);        % droite de régression + intervalle de confiance
title('Nuage de points X vs Y avec droite de régression');
xlabel('X'); ylabel('Y');

%% --- Résultats de la régression ---
disp('--- Résultats de la régression ---');
disp(mdl);
```

Paramètres de la droite : Calcul de a et b :

```
% Génération des données
X = randn(200,1);
Y = 2*X + 0.5 + randn(200,1);
% Ajustement du modèle linéaire
mdl = fitlm(X,Y);

% Affichage des paramètres
disp('--- Paramètres de la droite de régression ---');
coeffs = mdl.Coefficients.Estimate;
disp(['Ordonnée à l''origine (intercept) : ', num2str(coeffs(1))]);
disp(['Pente (slope) : ', num2str(coeffs(2))]);
```

Application :

1. Copier ce tableau dans une feuille Excel puis l'enregistrer dans le même dossier qui contient votre script.
2. Créer une script sous Matlab,
    a. Importer les données vers Matlab à l'aide de la fonction xlsread(), voir l'aide au besoin,
    b. Tracer la fonction réelle Y(X) (nuage de points),
    c. Associer une droite à ce nuage de point.
    d. Déterminer les deux paramètres de la droite.
    e. Tracer la courbe théorique et la courbe réelle sur le même graphique,

X	Y
624	-2,6205978
1248	-2,3880607
1872	-1,5398658
2496	-1,2663762
3120	-0,9027205
3744	-0,4958318
4368	-0,1933719
4992	-0,0014723
5616	0,09404783
6240	0,44349577
6864	0,97710617
7488	1,82690267

TP3 Matlab FI-GIL
Les polynômes et Résolution d'équations linéaires

1- les polynômes

Les polynômes sont traités comme des vecteurs de coefficients dans Matlab. Trouver les racines d'un polynôme f(x) consiste à chercher les valeurs de x qui annulent ce polynôme. MATLAB représente un polynôme comme une matrice uniligne.

Dans MATLAB, un polynôme et ses racines sont des vecteurs. Le polynôme étant un vecteur uniligne et les racines un vecteur unicolonne.

Fonctions :

conv(p,q)	produit de polynômes p et q
deconv(p,q)	Division de deux polynômes p et q
roots(p)	trouve les racines d'un polynôme
poly(racines)	trouve le polynôme à partir des ses racines
polyval(p,x)	évalue le polynôme en un ou plusieurs points(x=points)
polyder(p)	Calculer la dérivée du polynôme p
polyint(p)	Calculer l'intégrale du polynôme p

Exemple : $p(x) = 3x^3 + 7x^2 + 6x + 14$
on écrit p = [3 7 6 14];
Pour obtenir les racines d'un tel polynôme, on utilise la commande roots.

```
>>p = [3 7 6 14]
>>r = roots(p)
```

Exercice : Soit les polynômes: $a(x)=x^3+2x^2+3x+4$ et $b(x)=x^3+4x^2+9x+16$

a- Multiplier ces deux polynômes.
b- Additionner ces polynômes.
c- Soit c le polynôme obtenu après la multiplication de a et b, faire la division de c par b.
d- tracer la courbe $p(x)=x^3+4x-7x-10$ pour 100 valeurs de $x\epsilon$ [-1,3]

polyval évalue le polynôme p(x) aux différentes valeurs de x et place le résultat dans v. Ce sont ces valeurs de v en fonction de x qui constituent la courbe qui représente le polynôme p(x).

Exercice 1 : soit f_1 et f_2 deux polynômes $\begin{cases} f_1(x) = x^3 - 4x^2 + 2 \\ f_2(x) = 3x^2 + 12x - 2 \end{cases}$

Ecrire un script Matlab permettant de :

1) trouvez les racines de chaque polynôme.
2) Calculer leurs dérivées respectives Df1, Df2.
3) déterminer le polynôme S(x) somme des deux polynômes.
4) Calculer Tf= f_1 - f_2
5) déterminer P(x) le produit (ou convolution) des deux polynômes.
6) donner la division du polynôme P(x) sur le polynôme f_1
7) tracer l'évolution du polynôme P(x) sur l'intervalle [-3,3].

Solution:

Exercice 2

Soit $p(x)=3x^4-7x^3+2x^2+1$

1) trouver les racines de p(x) et reconstruire p à partir de ses racines.
2) soit x=-5 :0.5 :5, tracer la dérivé de p (dp) et la primitive de p (ip) en fonction de x dans la fenêtre graphique.
3) évaluer le polynôme p(x) en un point donné x1=5.

2- Résolution d'équations linéaires

On va voir dans cette partie comment résoudre des équations linéaires du type suivant

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n &= b_1 \\ \vdots \qquad\qquad\quad &= \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n &= b_m \end{aligned}$$

où n est le nombre d'inconnues: $x_1, ..., x_n$ et m le nombre d'équations.

La **représentation matricielle** des systèmes d'équations est sous la forme $\boxed{AX = B}$

$$A = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{bmatrix}, \quad X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad B = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

A est appelé matrice de coefficients et X le vecteur solution.

Si m=n, A est une matrice carrée et si le déterminant de A (det(A)) est non nul, alors A est inversible : $A^{-1}AX = X = A^{-1}B$

Résolution d'équations linéaires avec MATLAB : Avec MATLAB, les équations linéaires peuvent se résoudre à l'aide de l'opérateur '\', comme suit :

```
>> A = [2 -1 ; 1 1] ;
>> B = [2 ; 5] ;
>> X = A\B
```

L'opérateur **anti-slash**, '\', donne systématiquement un résultat. Dans le cas où le système d'équations n'admet pas de solution : ce résultat sera faux. Dans le cas d'une infinité de solutions, l'opérateur donnera une solution particulière (c'est-à-dire l'une des solutions possibles).

Exercice

Soit le système d'équations suivant :

$4x+3y+2z+t=1$
$3x+4y+3z+2t=1$
$2x+3y+4z+3t=-1$
$x+2y+3z+4t=-1$

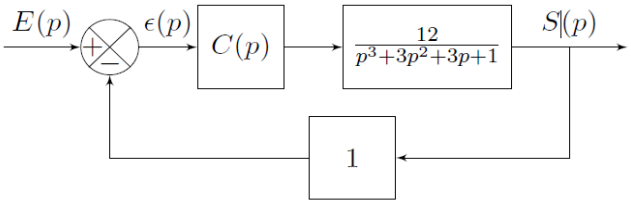
On suppose que l'écriture matricielle de ce système est A*X=b

I-donnez les commandes permettant de créer :1) la matrice A 2) le vecteur b
3) le déterminant de la matrice A
II- donnez la commande permettant de résoudre ce système.

Solution

Devoir d'automatique

Dans cet exercice, nous souhaitons corriger le système suivant :



où $C(p)$ correspond à la fonction de transfert du correcteur.

1.1 Analyse de la boucle ouverte

Manipulation 1.1 (Définition de la fonction de transfert)**.** Nous allons tout d'abord définir la fonction de transfert du système non bouclé et non corrigé dans l'environnement Matlab. Pour définir cette fonction, nous allons lancer l'instruction :

```
F=tf([12],[1 3 3 1])
```

Lorsque vous exécutez cette commande, Matlab affiche dans la fenêtre de commande l'expression de la fonction de transfert. Par défaut Matlab utilise la notation anglo-saxonne c'est à dire que la variable `p` est remplacée par la variable `s`.

Manipulation 1.2 (Affichage de la réponse indicielle)**.** Nous allons afficher la réponse indicielle du système non bouclé et non corrigé. Pour afficher la réponse du système à un échelon d'amplitude unitaire, nous allons lancer l'instruction :

```
step(F)
```

Question 1.1. A partir de la courbe précédente, déterminer la gain statique et le temps de réponse du système non bouclé et non corrigé.

Manipulation 1.3 (Affichage de la réponse harmonique)**.** Nous allons afficher la réponse harmonique du système pour avoir une idée de son comportement en boucle fermée. Pour afficher le diagramme de Bode du système, nous allons lancer l'instruction :

```
bode(F)
```

Question 1.2. Déterminer la marge de gain M_G et la marge de phase M_ϕ du système. Sans correction ($C(p) = 1$), est-ce que le système sera stable en boucle fermée ?

Pour valider votre réponse à la question précédente, nous allons boucler le système et observer sa réponse indicielle.

Manipulation 1.4 (Affichage de la réponse indicielle du système bouclé)**.** Pour afficher la réponse indicielle du système bouclé, nous allons lancer les instructions :

```
FTBF=feedback(F,1)
step(FTBF)
```

Vérifier que le système est bien instable en boucle fermée.

1.2 Correction proportionnelle

Pour stabiliser la boucle fermée, nous choisissons tout d'abord un correcteur proportionnel c-a-d

$$C(p) = K \tag{8.1}$$

Système en limite de stabilité

Question 1.3. Calculer la fonction de transfert en boucle fermée. A partir du critère de Routh, déterminer la valeur limite K, notée K_{OSC}, permettant d'obtenir un système en boucle fermée en limite de stabilité.

Nous allons maintenant déterminer la fonction de transfert du système bouclé et corrigé. Pour obtenir cette fonction de transfert, nous utiliserons les commandes suivantes :

```
C= ? ;              % a completer en fonction du correcteur
FTBO=C*F;           % calcul de la FTBO
FTBF=feedback(FTBO,1);
```

Manipulation 1.5. Afficher la réponse indicielle du système bouclé et corrigé lorsque $C(p) = K_{OSC}$.

Question 1.4. Déterminer la période T_{OSC} des oscillations en boucle fermé.

Système stable

Question 1.5. En utilisant le diagramme de Bode du système non-corrigé et non bouclé (voir manipulation 1.3), déterminer approximativement la valeur de K_1 (en dB) permettant d'obtenir une marge de gain M_G de 6dB. Exprimer K_1 en valeur naturelle.

Manipulation 1.6. Afficher la réponse indicielle du système bouclé et corrigé (avec $K = K_1$).

Question 1.6. A partir de la courbe précédente, déterminer la gain statique et le temps de réponse du système bouclé et corrigé.

1.3 Correction Proportionnelle-Intégrale

Pour améliorer la précision du système, nous allons utiliser un correcteur PI décrit par la fonction de transfert suivante :

$$C(p) = K\left(\frac{T_i p + 1}{T_i p}\right) \tag{8.2}$$

Les paramètres du correcteur seront déterminés via le technique de Ziegler-Nichols. Cette technique se base sur les paramètres K_{OSC} et T_{OSC} déterminés précédemment. Plus précisément, pour un correcteur PI, les paramètres K et T_i sont fixées de la manière suivante :

$$K = \frac{K_{OSC}}{2.2}$$
$$T_i = \frac{T_{OSC}}{1.2}$$

Question 1.7. Déterminer les valeurs numériques de K et T_i.

Manipulation 1.7. En vous inspirant des manipulations 1.1 et 1.3, afficher le diagramme de Bode du correcteur.

Manipulation 1.8. Afficher la réponse indicielle du système bouclé et corrigé.

Question 1.8. A partir de la courbe précédente, déterminer la gain statique et le temps de réponse du système bouclé et corrigé.

1.4 Correction Proportionnelle-Intégrale-Dérivée

Pour améliorer la dynamique du système, nous allons utiliser un correcteur PID décrit par la fonction de transfert suivante :

$$C(p) = K\left(\frac{T_i T_d p^2 + T_i p + 1}{T_i p}\right) \tag{8.3}$$

Les paramètres du correcteur seront déterminés via le technique de Ziegler-Nichols. Pour un correcteur PID, les paramètres sont fixées de la manière suivante :

$$K = \frac{K_{OSC}}{1.7}$$
$$T_i = \frac{T_{OSC}}{2}$$
$$T_d = \frac{T_{OSC}}{8}$$

Question 1.9. Déterminer les valeurs numériques de K, T_i et T_d.

Manipulation 1.9. Afficher le diagramme de Bode du correcteur.

Manipulation 1.10. Afficher la réponse indicielle du système bouclé et corrigé.

Question 1.10. A partir de la courbe précédente, déterminer la gain statique et le temps de réponse du système bouclé et corrigé.

RESEAUX DE NEURONES

TP7

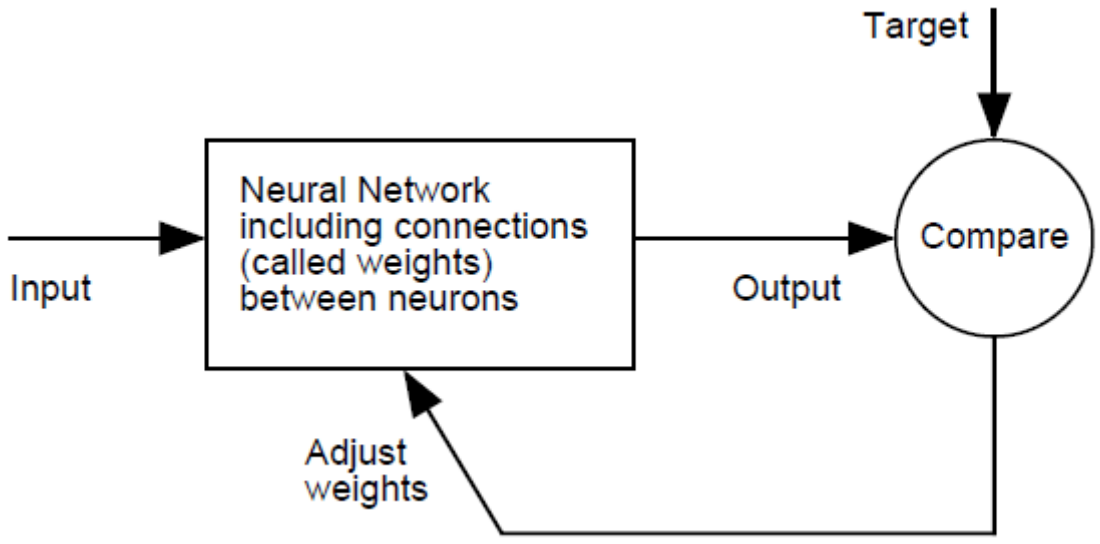
Ajustement de fonction et classification

Objectif

Ce TP explique comment utiliser deux outils graphiques : l'ajustement de fonctions et la classification. Ceci pour résoudre des problèmes en appliquant la technique des réseaux de neurones.
L'utilisation de ces deux techniques nous donnera une excellente introduction à l'utilisation de la boîte à outils des réseaux de neurones (neural network toolbox™ software).

Présentation des réseaux de neurones

Les **réseaux de neurones** sont **composés** par des **éléments simples** fonctionnant en parallèle. Ces éléments sont **inspirés** par des **systèmes nerveux biologiques**. Comme dans la nature, les **connexions** entre ces éléments déterminent en grande partie la fonction de réseau. Nous pouvons exploiter un réseau de neurone pour exécuter une fonction particulière en **ajustant** les valeurs (paramètres) des **connexions** (des poids) entre ces éléments.
Typiquement, les réseaux de neurones sont **ajustés**, ou formés, pour qu'un **objet** entré (**input**) mène à une production cible (**target**) spécifique. La figure ci dessous illustre une telle situation.
Le réseau est ajusté en se basant sur une comparaison de la production et la cible (target), jusqu'à ce que la production de réseau corresponde à la cible (target). Typiquement plusieurs tel paires Entré/Cible (Input/Target) sont nécessaires pour former un réseau.



Les réseaux de neurones ont été formés pour exécuter des fonctions complexes dans des divers domaines, comme la **reconnaissance de formes**, l'identification, la **classification**, le discours, la vision et des systèmes de commande.

TAF

- Décrire le rôle des réseaux de neurones en se basant sur la figure ci-dessus.

TAF : Exemple Illustrative de Classification des fruits

- Dans la command window tapez:

>> nnd3pc

- Cliquer sur Go
- Décrire l'exemple.
- Que fera la machine pour classer le fruit ?
- Quelle est la méthode ou technique de classification utilisée ?
- Présenter une modélisation et une description de cette technique.
- Comment cette technique pourra décider pour classifier le fruit ?
- Ecrire un algorithme qui résume cette technique de classification (préciser la règle de décision)

- Il est possible d'utiliser d'autres techniques : dans la command window tapez:

>> nnd3hamc

- Quelle est la nouvelle technique de classification utilisée ?
- Présenter une modélisation et une description de cette deuxième technique.
- Comment cette nouvelle technique pourra décider pour classifier le fruit ?
- Ecrire un algorithme qui résume cette technique de classification (préciser la règle de décision)

- Dans la command window tapez:

>>nnd3hopc

- Quelle est la nouvelle technique de classification utilisée ?
- Présenter une modélisation et une description de cette deuxième technique.
- Comment cette nouvelle technique pourra décider pour classifier le fruit ?
- Ecrire un algorithme qui résume cette technique de classification (préciser la règle de décision)
- Comment nous pouvons comparer l'efficacité de ces trois techniques ?
- Définir les réseaux de neurones en présentant leur apport pour l'Intelligence Artificielle.

(Notes de Lecture)

Les réseaux de neurones peuvent aussi être utilisés pour résoudre des problèmes qui sont difficiles même pour des ordinateurs conventionnels ou pour des êtres humains. La boîte à outils souligne l'utilisation des réseaux de neurones dans l'ingénierie, la finance et d'autres applications pratiques.

Fonctions d'ajustement

Supposons, par exemple, que nous avons des données d'une demande de logement [HaRu78]. Nous voulons concevoir un réseau qui peut prévoir la valeur d'une maison, étant donné 13 pièces d'informations géographiques et immobilières. Nous avons un total de 506 exemples de maisons pour lesquelles nous avons ces 13 packs de données.

Présentation du problème

Pour définir un problème à la boîte à outils, arrangeons un ensemble de vecteurs d'entrés Q (input) comme des colonnes dans une matrice. Ensuite, arrangeons un autre ensemble de vecteurs T cible de Q dans une deuxième matrice. Par exemple, nous pouvons définir le problème d'ajustement du Booléen ET portant sur quatre ensembles à deux éléments de vecteurs d'entrés et un élément de cible comme suit :

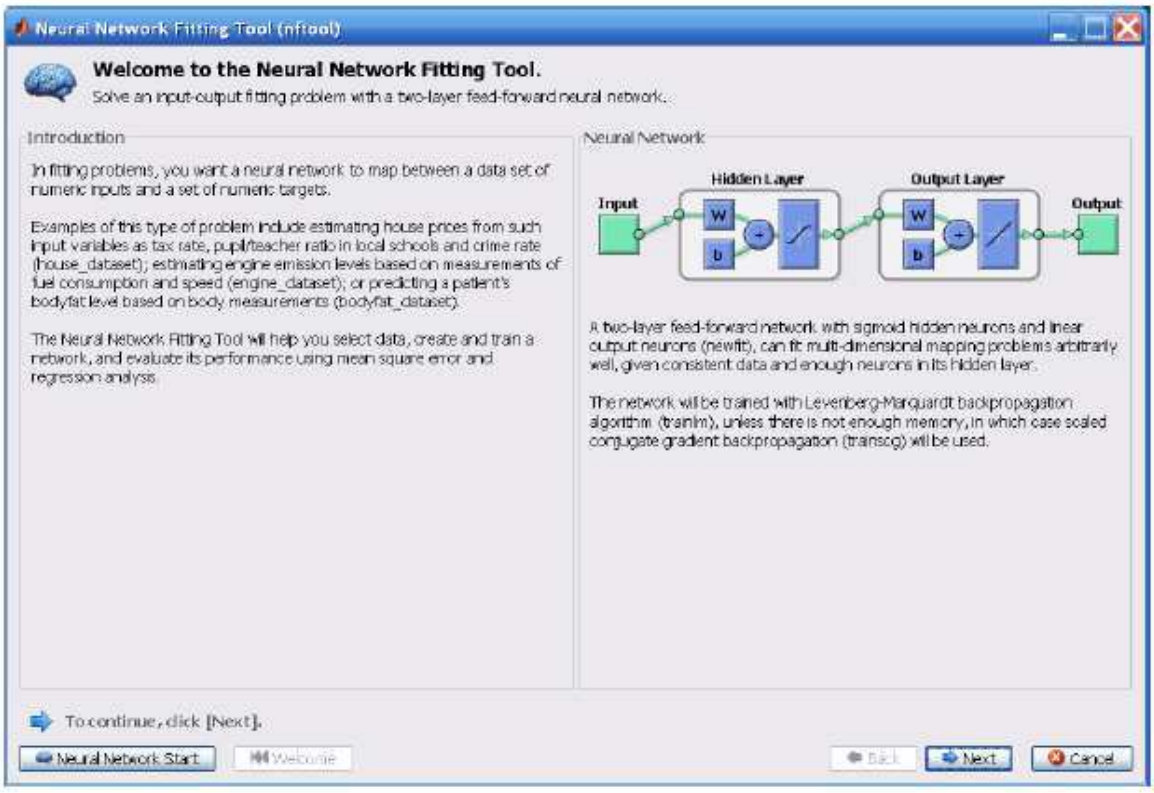
```
>>Q = [0 1 0 1; 0 0 1 1];
>>T = [0 0 0 1];
```

La sous section suivante démontre comment apprendre à un réseau d'adapter un ensemble de données, utilisant l'outil d'ajustement de réseau de neurone, nftool. Cet exemple utilise l'ensemble de données de logement fourni avec la boîte à outil.

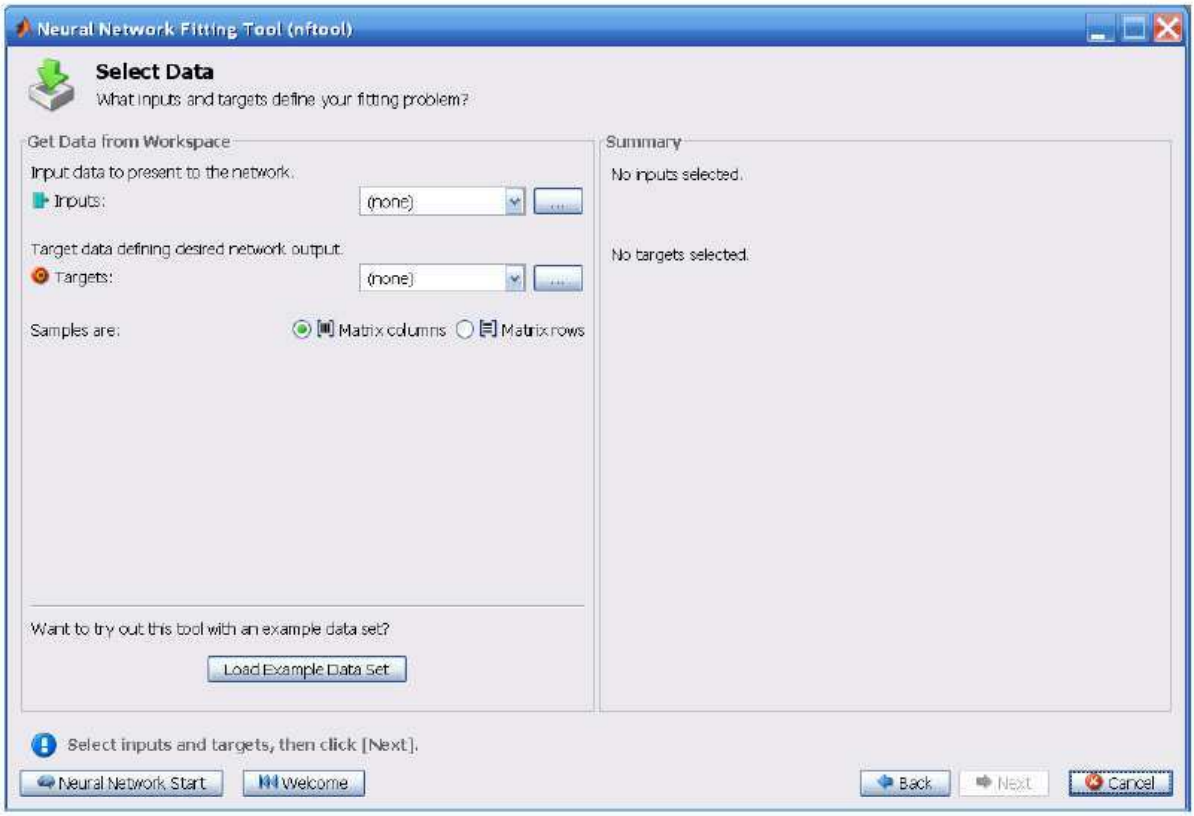
Utilisation de l'Outil d'Ajustement de Réseau de Neurone

1 Ouvrir l'**outil d'ajustement** en tapant:

```
>>nftool
```

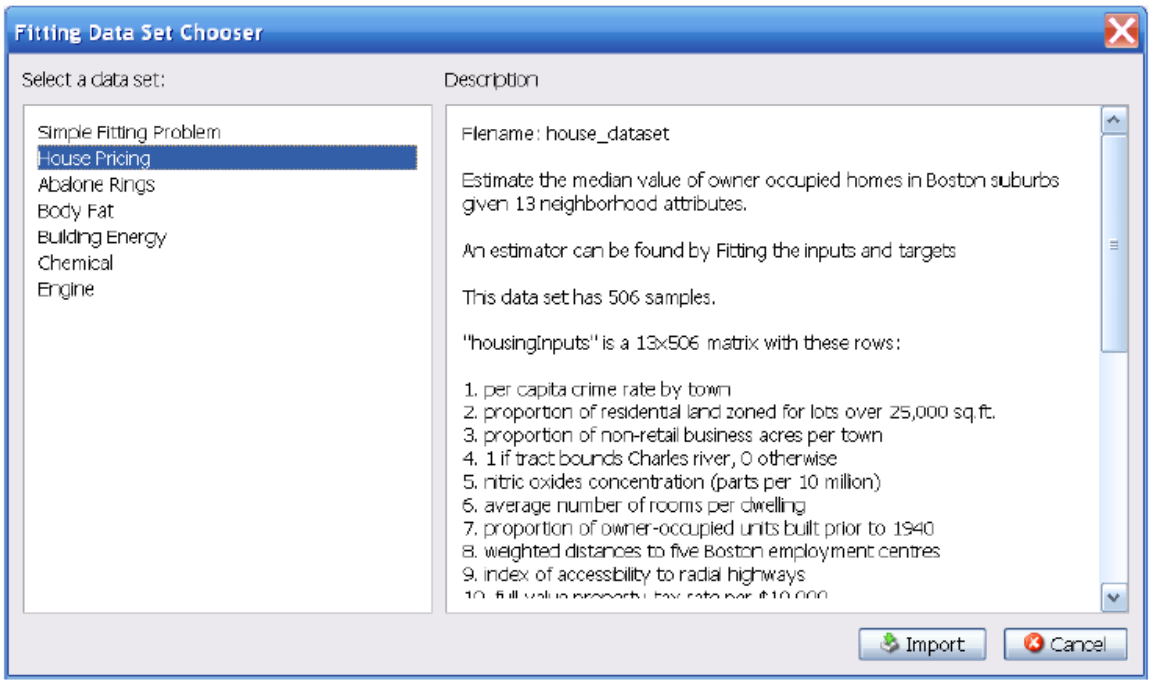


2 Cliquer **Next** pour procéder.



3 Cliquer **Load Example Data Set** dans la fenêtre Select Data. La fenêtre Fitting Data Set Chooser s'ouvre.

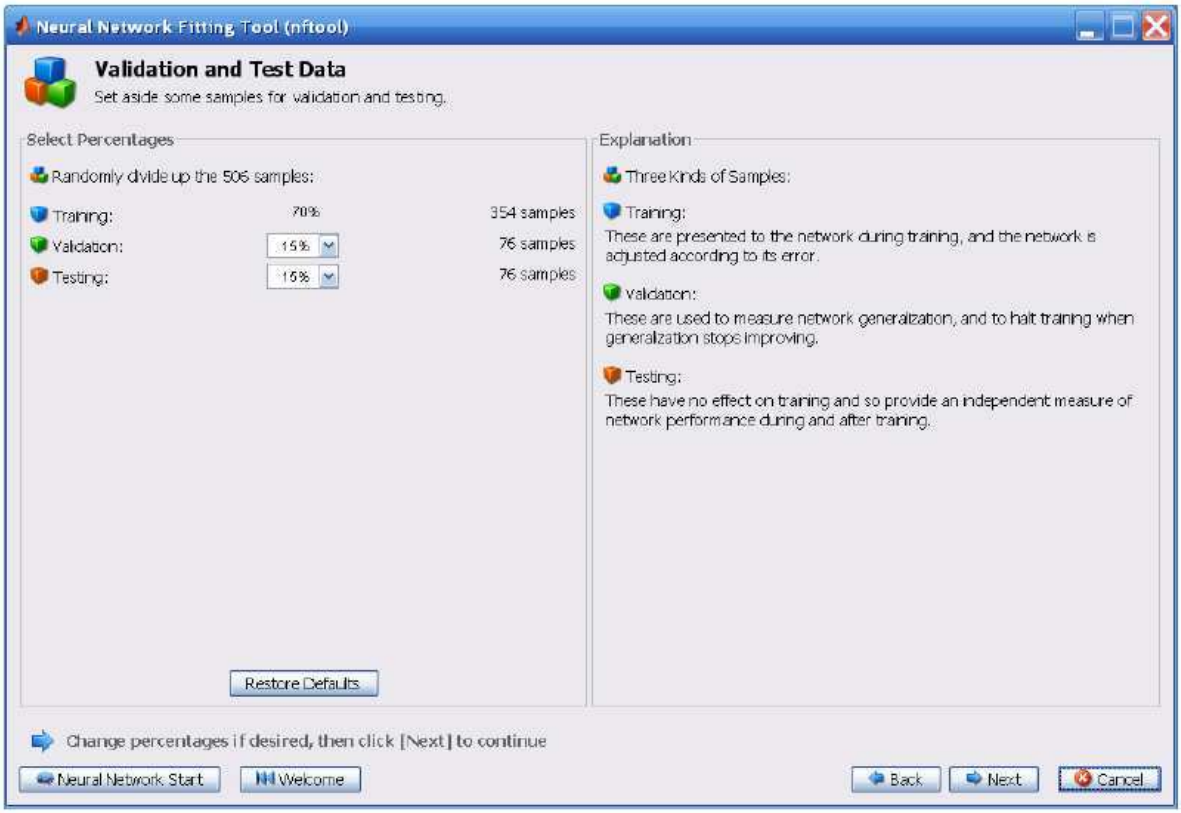
Noter Utiliser les options d'entrés (**Inputs)** et cibles (**Targets)** dans la fenêtre Select Data lorsque vous avez besoin d'importer des donnés de MATLAB workspace.



4 Selectionner **House Pricing**, et cliquer **Import**. Ceci retourne la fenêtre Select Data.

5 Cliquer **Next** pour afficher la fenêtre Validation and Test Data, voir la figure ci-dessous.

Les ensembles Validation et test data prennent chacun 15% comme valeurs d'origine.



Avec ces fixations de donnés, les vecteurs d'entrés et les vecteurs cibles seront aléatoirement divisés dans trois ensembles comme suit :

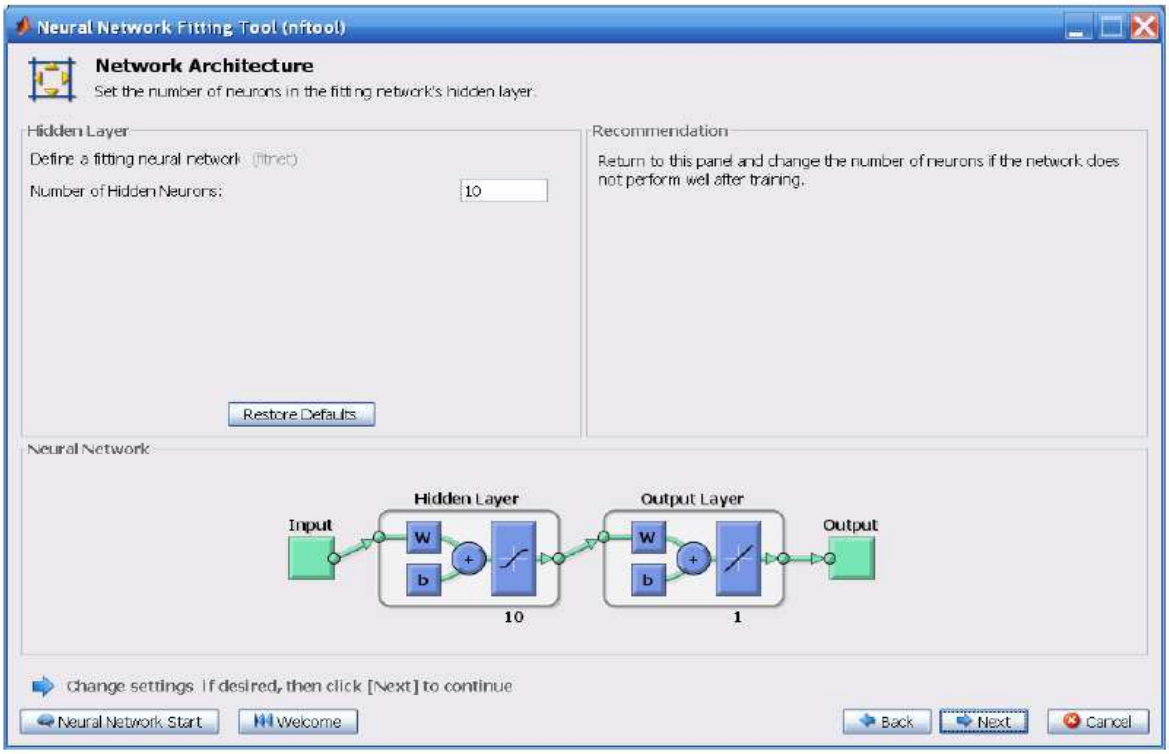
•70% sera utilisé pour la formation (training).

•15% sera utilisé pour valider le réseau généralisé et arrêter la formation auparavant de l'ajustement.

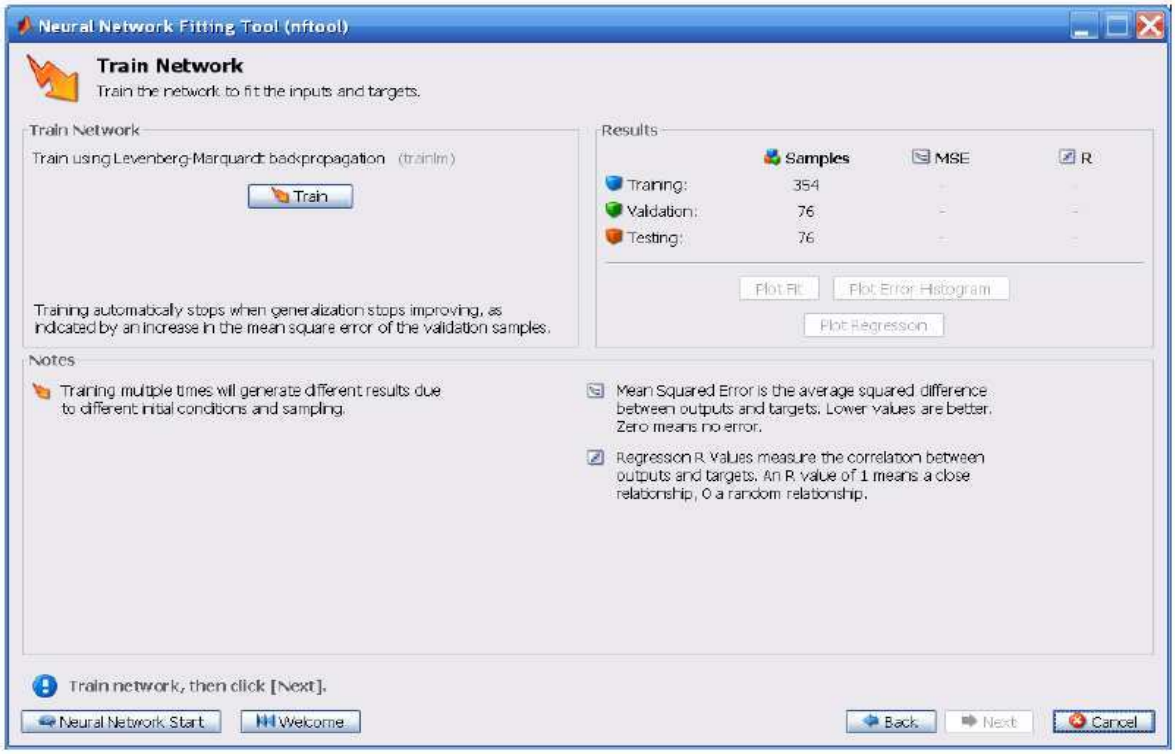
•Les 15% restante sera utilisée comme un test complètement indépendant de la généralisation de réseau.

6 Cliquer **Next**.

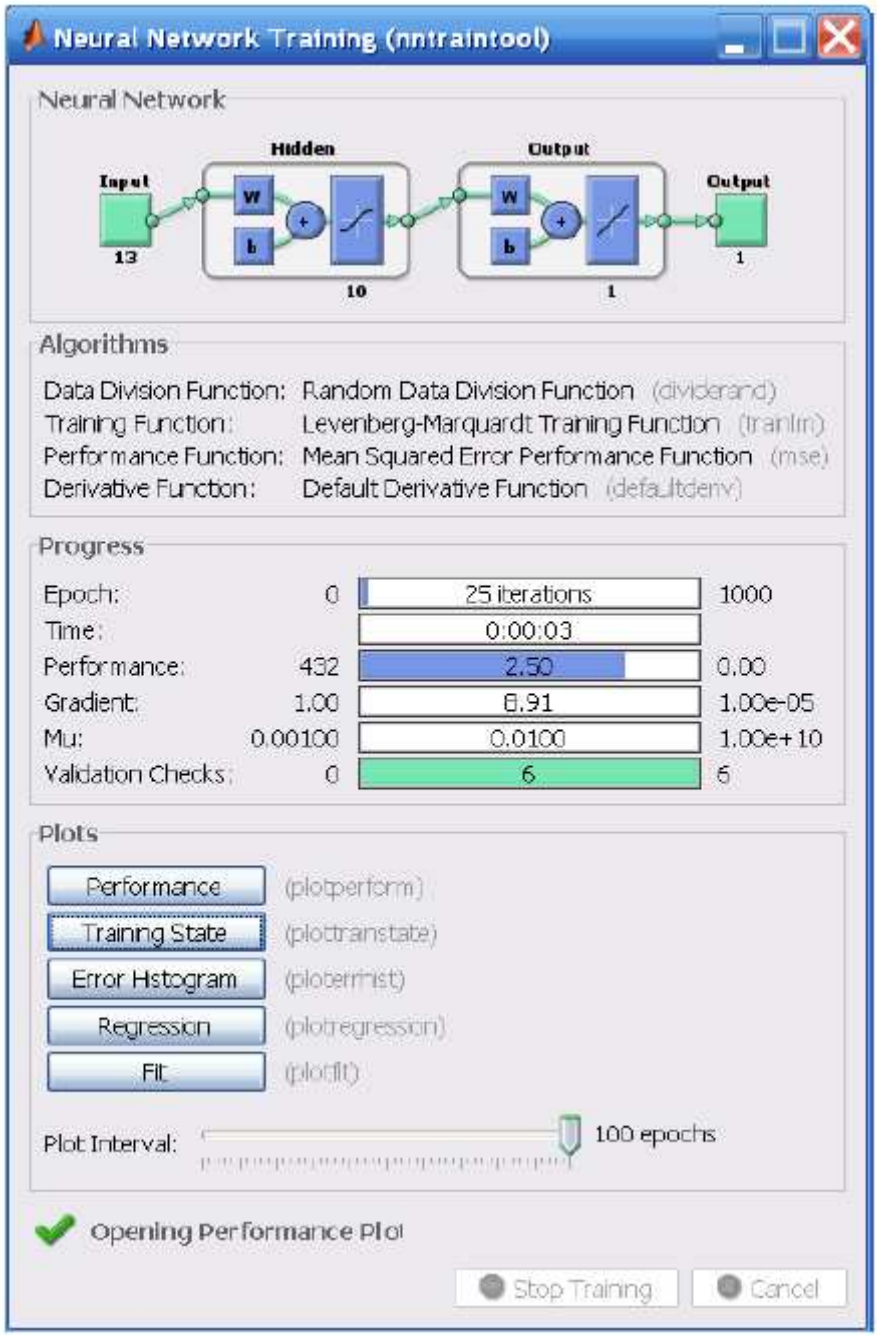
Le réseau standard qui est utilisé pour l'ajustement de fonction est un réseau feedforward à deux couches, avec une fonction de transfert de sigmoid dans la couche cachée et une fonction de transfert linéaire dans la couche de production. Le nombre par défaut de neurones cachés est mis à 10. Vous pourriez augmenter ce nombre plus tard, si le réseau formant la performance est pauvre.



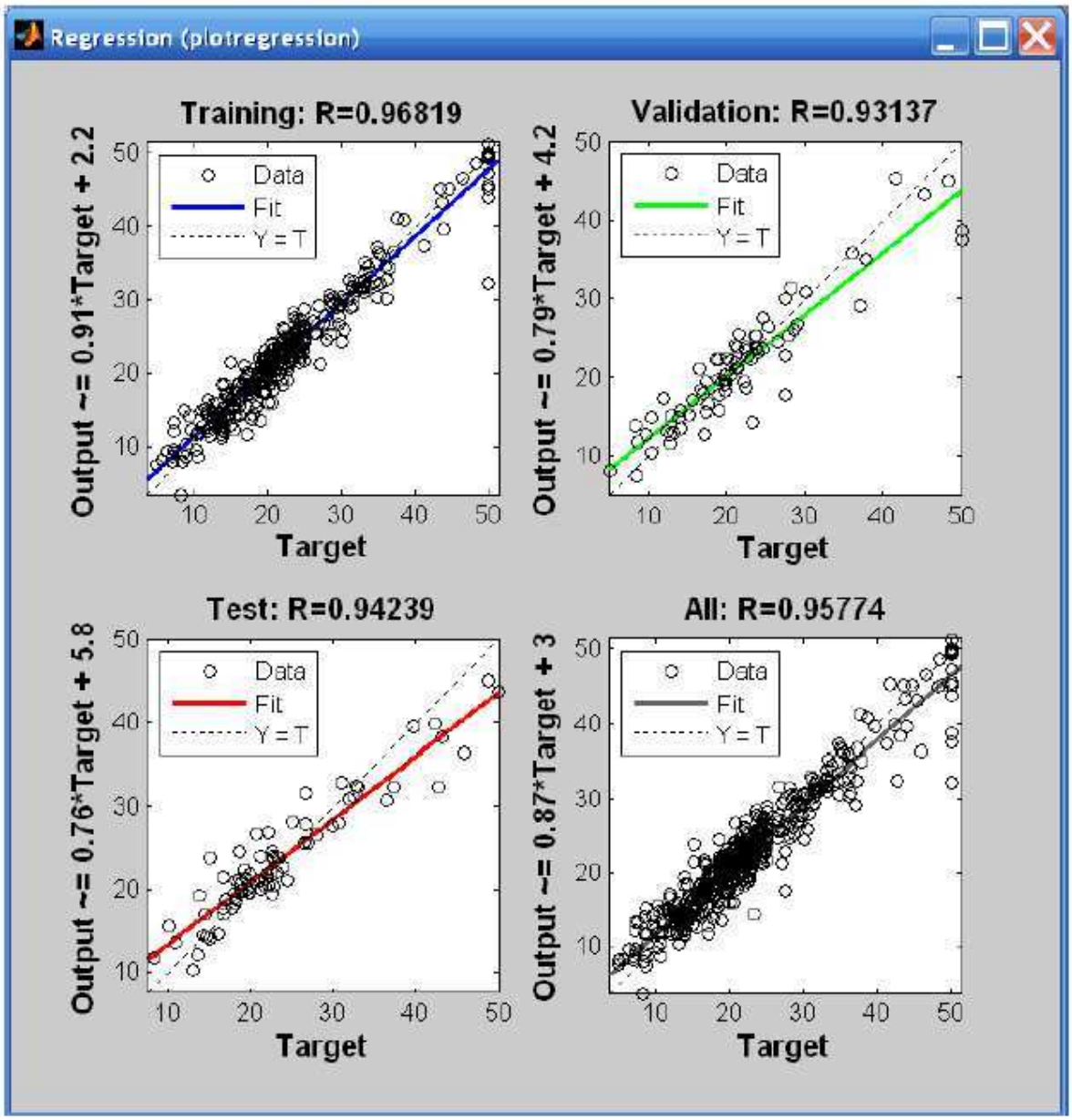
7 Cliquer **Next**.



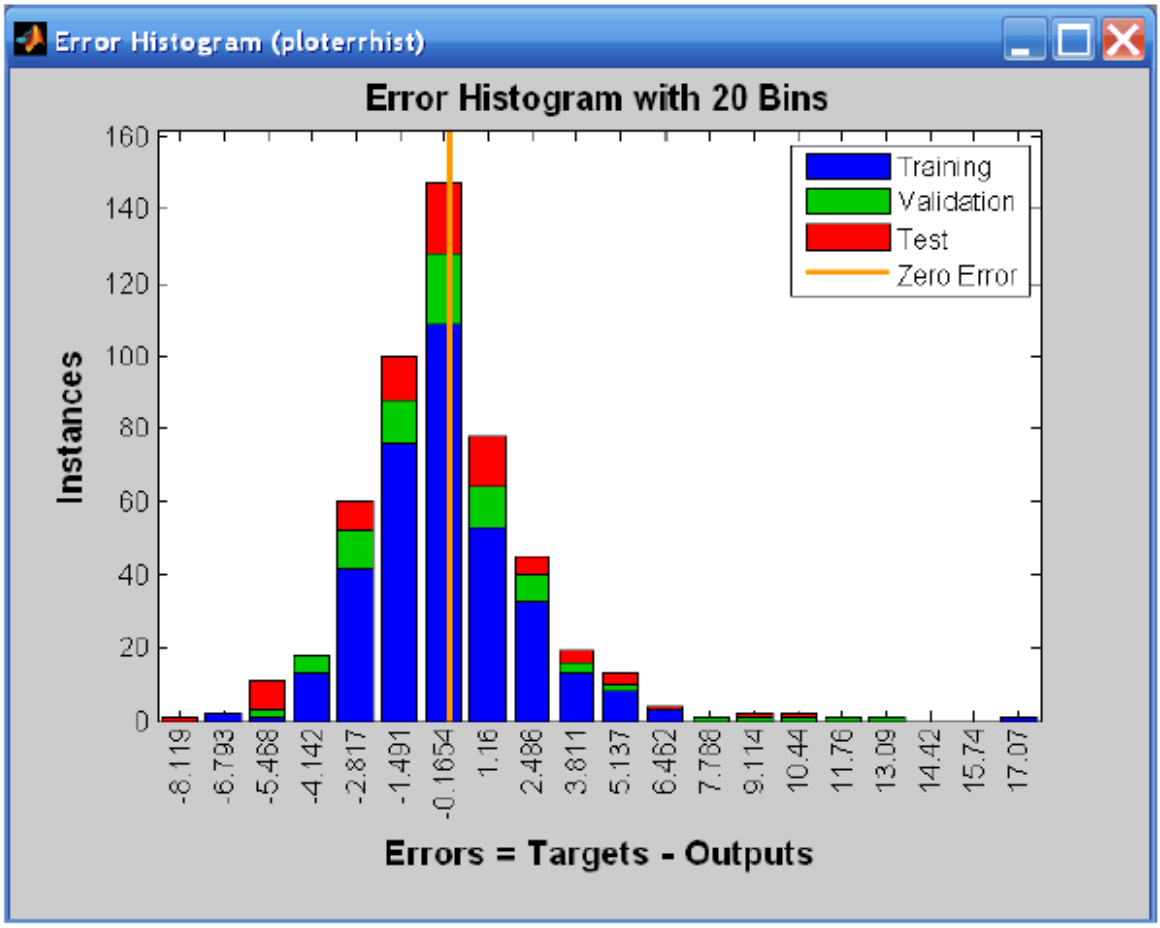
8 Cliquer **Train.**
La formation continue jusqu'à l'échec de l'erreur de validation à diminuer durant six itérations (l'arrêt de validation).



9 Cliquer Régression. Ceci est utilisé pour valider la performance de réseau.
Les représentations de régression suivantes montrent les productions de réseau en respectant les cibles formés, la validation et les testes des ensembles. Pour une crise parfaite, les données devraient chuter le long de 45 degré, où les productions de réseau sont égales aux cibles. Pour ce problème, la crise est raisonnablement bonne pour tous les ensembles de données, avec des valeurs de R de 0.93 dans chaque cas. Si des résultats encore plus précis ont été exigés, vous pourriez recycler le réseau par cliquer Recyclent dans nftool. Cela changera les poids initiaux et les préventions du réseau et peut produire un réseau amélioré après Retraining. On fournit d'autres options sur le carreau suivant.

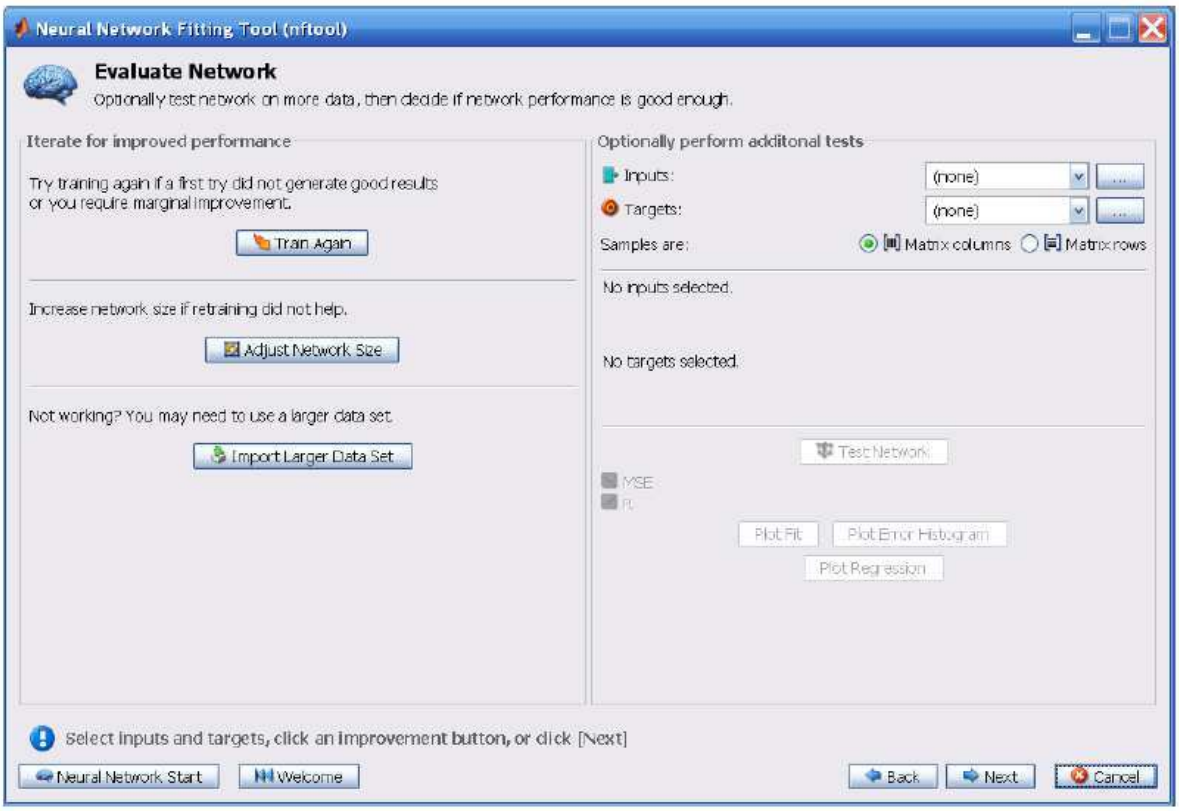


10 Considérez l'histogramme d'erreur pour obtenir une vérification supplémentaire de la performance de réseau. Sur le **Plots** pane, cliquer **Error Histogram**.



Les barres bleues représentent des données de formation, les barres vertes représentent des données de validation et les barres rouges représentent des données de test. L'histogramme peut vous donner une indication d'outliers, qui sont des points de données où la crise est significativement plus mauvaise que la majorité de données. Dans ce cas, vous pouvez voir que tandis que la plupart d'erreurs chutent entre -5 et 5, c'est un point de formation avec une erreur de 17 et des points de validation avec des erreurs de 12 et 13. Ces outliers sont aussi visibles sur le complot de régression de test. Le premier correspond au point avec une cible de 50 et la production près de 33. C'est une bonne idée de vérifier l'outliers pour déterminer si les données sont mauvaises, ou si ces points de données sont différents que le reste d'ensemble de données. Si l'outliers sont des points de données valables, mais elles sont différentes du reste des données, donc le réseau extrapole pour ces points. Vous devriez rassembler plus de données qui ressemblent aux points d'outlier et recycler le réseau.

11 Cliquer Ensuite dans l'Outil d'Ajustement de Réseau de Neurone pour évaluer le réseau.



À ce point, vous pouvez tester le réseau contre de nouvelles données. Si vous êtes peu satisfaits avec la performance du réseau sur les données originales ou nouvelles, vous pouvez faire une des choses suivantes :

·**Train** de nouveau.

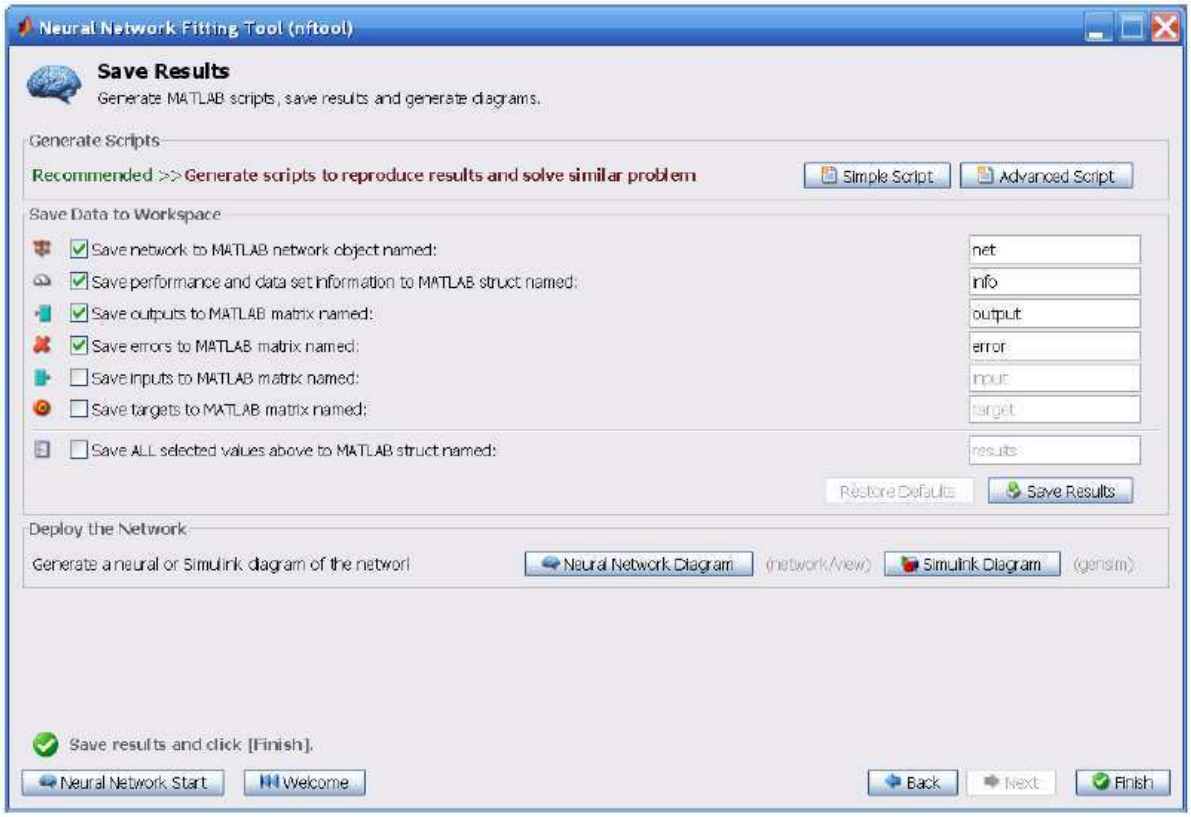
·**Augmenter** le numéro de neurones.

·**Choisir** un ensemble plus grand de données de formation.

Si la performance sur l'ensemble de formation est bonne, mais la performance de l'ensemble de test est significativement plus mauvaise, qui pourrait indiquer le sur ajustement, donc la réduction du nombre de neurones peut améliorer vos résultats. En formant une performance faible, alors vous pouvez augmenter le nombre de neurones.

12 si vous êtes satisfaits de la performance de réseau, cliquez **Next**.

13 Utilisez les boutons sur cet écran pour produire d'autres scénarios ou sauvegarder vos résultats.



- Vous pouvez cliquer sur le **Simple Script** ou **Advanced Script** pour créer le code MATLAB ® qui peut être utilisé pour reproduire tous les étapes précédents de la ligne de commande. La création du code MATLAB ® peut être utile si vous voulez apprendre comment utilisez la fonctionnalité de ligne de commande de la boîte à outils pour personnaliser le processus de formation.
- Vous pouvez aussi faire sauver le réseau comme le réseau dans l'espace de travail. Vous pouvez y exécuter des tests supplémentaires ou le mettre pour travailler sur de nouveaux entrés.
14 Quand vous avez créé le code MATLAB ® et après avoir sauvegardé vos résultats, cliquer **Finish**.

Comment utiliser la fonctionalité de la ligne de commande

La façon la plus facile d'apprendre comment utiliser la fonctionnalité de ligne de commande de la boîte à outils est de produire des scénarios du GUIS et les modifier ensuite pour personnaliser la formation de réseau. Comme un exemple, regardez le scénario simple qui a été créé à l'étape 13 de la section précédente.

```
% Solve an Input-Output Fitting problem with a Neural Network
% Script generated by NFTOOL
%
% This script assumes these variables are defined:
%
% houseInputs - input data.
% houseTargets - target data.
inputs = houseInputs;
targets = houseTargets;
% Create a Fitting Network
hiddenLayerSize = 10;
net = fitnet(hiddenLayerSize);
```

```
% Set up Division of Data for Training, Validation, Testing
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;
% Train the Network
[net,tr] = train(net,inputs,targets);
% Test the Network
outputs = net(inputs);
errors = gsubtract(outputs,targets);
performance = perform(net,targets,outputs)
% View the Network
     view(net)

% Plots
% Uncomment these lines to enable various plots.
%figure, plotperform(tr)
%figure, plottrainstate(tr)
%figure, plotfit(targets,outputs)
%figure, plotregression(targets,outputs)
%figure, ploterrhist(errors)
```

Vous pouvez sauvegarder le scénario et l'exécuter ensuite de la ligne de commande pour vous reproduire les résultats de la session GUI précédente. Vous pouvez aussi éditer le scénario à personnaliser le processus de formation. Dans ce cas, suivez chaque scénario.
0 le scénario suppose que les vecteurs d'entrés et les vecteurs cibles sont déjà chargés dans l'espace de travail. Si les données ne sont pas chargées, vous pouvez les charger comme Suit :

```
load house_dataset
inputs = houseInputs;
targets = houseTargets;
```

Cet ensemble de données est un des ensembles de données types qui fait partie de la boîte à outils. Vous pouvez voir une liste de tous les ensembles de données disponibles en entrant à la commande help nndatasets.

La commande de chargement vous permet aussi de charger les variables de n'importe quel de ces ensembles de données utilisant votre propre nom de variable.
Par exemple, la commande

```
 [inputs,targets] = house_dataset;
```

1 Créez un réseau. Le réseau par défaut pour ajustement de fonction (ou régression) les problèmes, fitnet, sont un réseau feedforward avec le bronzage-sigmoid par défaut la fonction de transfert dans la couche cachée et le transfert linéaire fonctionne dans la couche de production. Vous avez assigné dix neurones (quelque peu arbitraire) à celui couche cachée dans la section précédente. Le réseau a un neurone de production, parce qu'il y a seulement une valeur cible associée à chaque vecteur d'entrés.

```
hiddenLayerSize = 10;
net = fitnet(hiddenLayerSize);
```

2 Fondez la division de données.

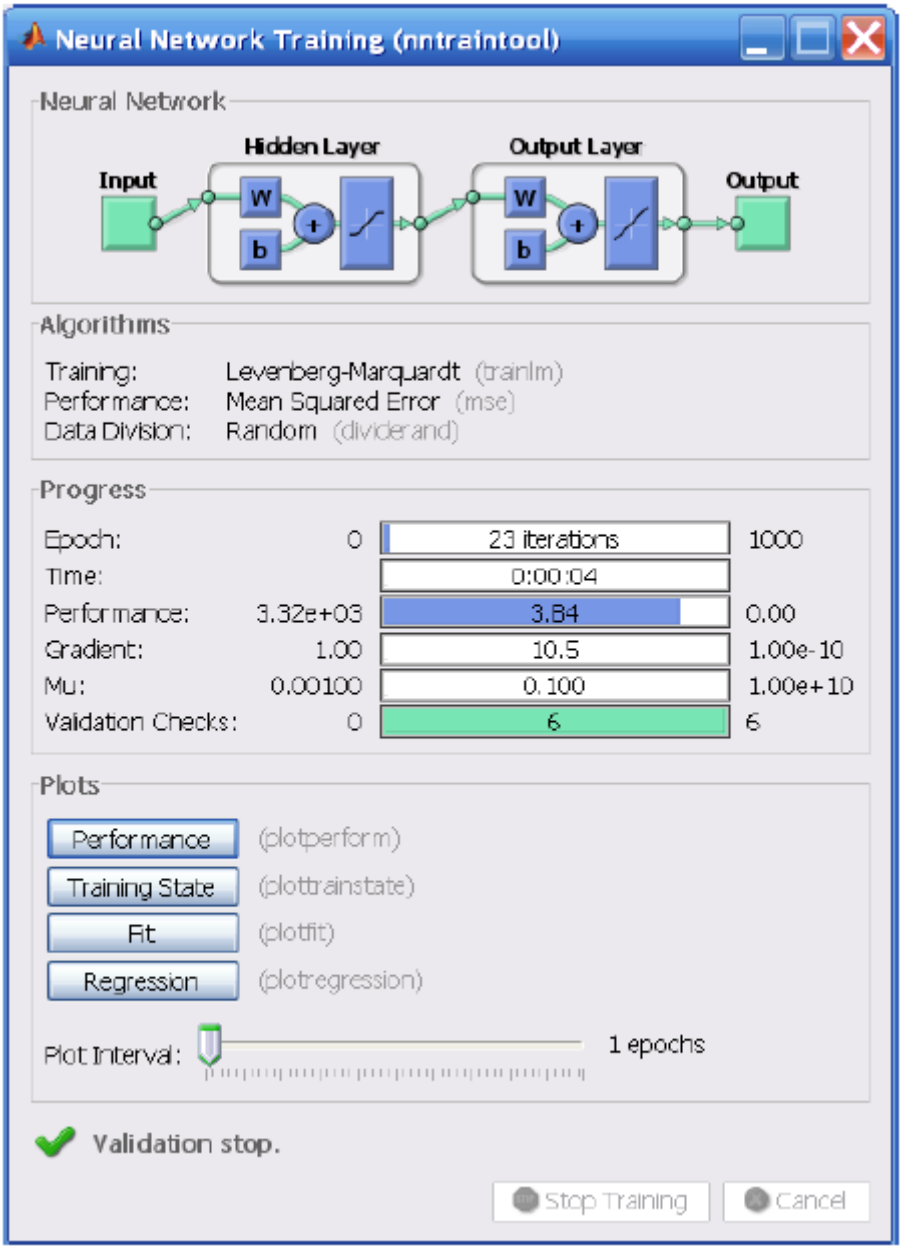
```
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;
```

Avec ces fixations, les vecteurs d'entrés et les vecteurs cibles seront aléatoirement divisés, avec 70 % utilisés pour la formation, 15 % pour la validation et 15 % pour le test.

3 Formez le réseau. Le réseau utilise par défaut l'algorthme de Levenberg-Marquardt pour la formation the network (trainlm). Pour former le réseau, entrer:

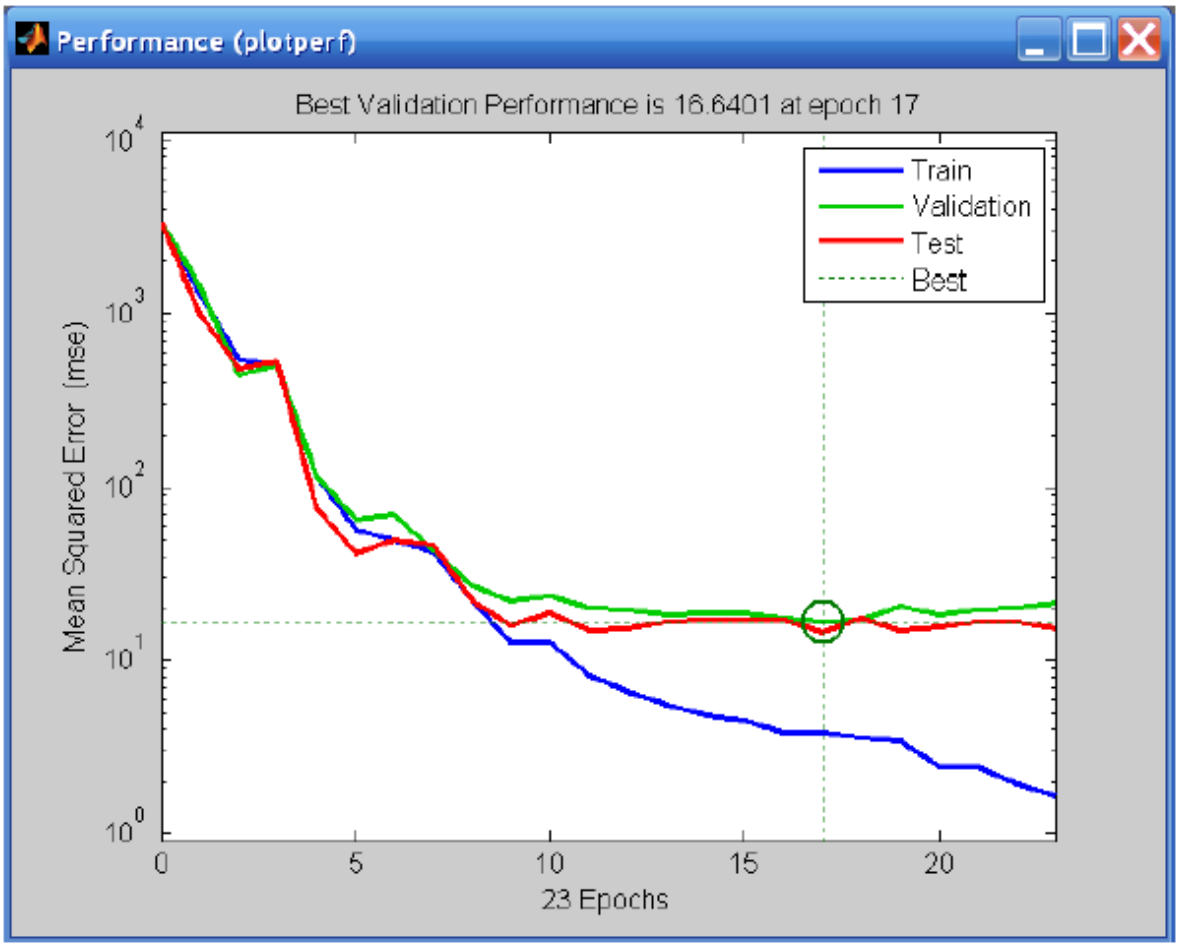
[net,tr] = train(net,inputs,targets);

Pendant la formation, la fenêtre de formation suivante s'ouvre. Ces affichages de fenêtre formant le progrès et vous permettent d'interrompre la formation à n'importe quel point en cliquant **Stop Training**.



Cette formation s'est arrêtée quand l'erreur de validation a augmenté durant six itérations, ce qui se fera à l'itération 23. Si vous cliquez **Performance** dans la fenêtre de formation, Un affichage des erreurs d'une formation, des erreurs de validation et des erreurs de test apparaît, comme indiqué dans la figure suivante.
Dans cet exemple, le résultat est raisonnable à cause des considérations suivantes :
· L'erreur carrée moyenne finale est petite.
· L'erreur de l'ensemble test et l'erreur de l'ensemble validation ont des caractéristiques semblables.

· Pas de surajustement significatif obtenu par l'itération 17 (où on a obtenu la meilleure performance de validation).



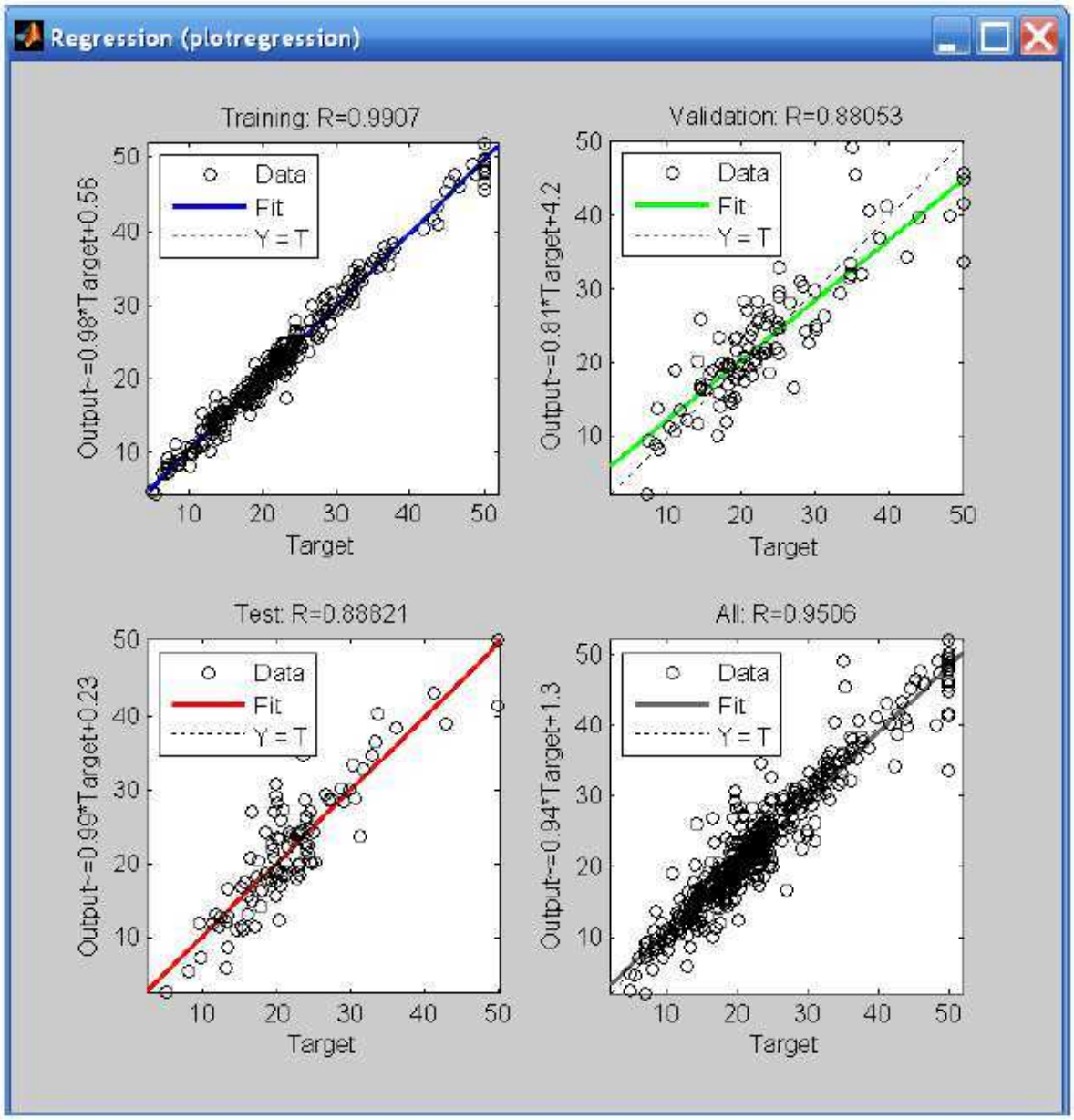
4 Testez le réseau. Après que le réseau a été formé, vous pouvez l'utiliser pour calculer les productions de réseau. Le code suivant calcule les productions de réseau, des erreurs et la performance complète.

```
outputs = net(inputs);
errors = gsubtract(targets,outputs);
performance = perform(net,targets,outputs)
```

Il est aussi possible de calculer la performance de réseau seulement sur l'ensemble de test, en utilisant les indices de test, qui sont placés dans le rapport de formation.

```
tInd = tr.testInd;
tstOutputs = net(inputs(tInd));
tstPerform = perform(net,targets(tInd),tstOutputs)
```

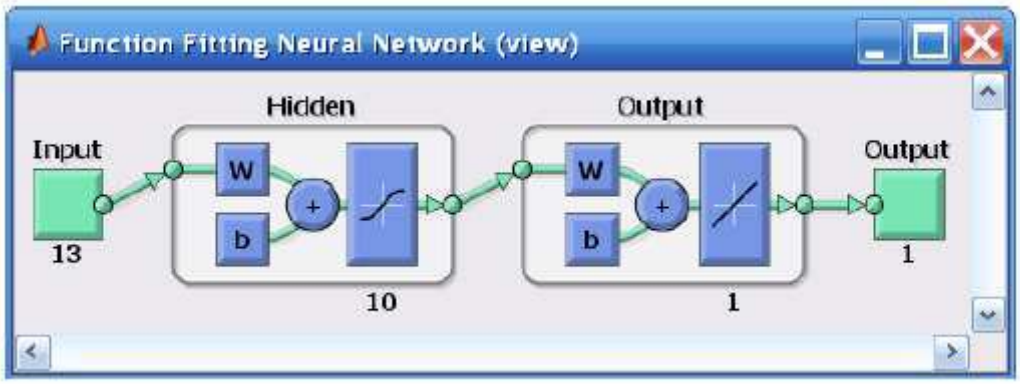
5 Exécutez quelques analyses de la réponse de réseau. Si vous cliquez **Regression** dans la fenetre de formation (training window), Vous pouvez exécuter une régression linéaire entre les productions de réseau et les cibles correspondantes.
La figure suivante montre les résultats.



La production suit à la trace les bons cibles pour la formation, le test et la validation et la R-valeur est plus de 0.95 pour la réponse totale. Si des résultats encore plus précis ont été exigés, vous pourriez essayer n'importe laquelle de ces approches :

· Reset les poids de réseau initiaux et les préventions à nouvelles valeurs avec init et train de nouveau.
· Augmenter le nombre de neurones cachés.
· Augmenter le nombre de vecteurs de formation.
· Augmenter le nombre de valeurs d'entrés, si des informations plus appropriées sont disponibles.
· Essayez un algorithme de formation différent.
Dans notre cas, la réponse de réseau est satisfaisante et vous pouvez maintenant utiliser le réseau avec de nouveaux entrés.

6 Considérez le diagramme de réseau.
view(net)
Cela crée le diagramme de réseau suivant.



Pour obtenir plus d'expérience dans des opérations de ligne de commande, essayez certaines de ces tâches :

•Pendant la formation, ouvrez une fenêtre d'affichage (comme l'affichage de régression) et animez la
• Affichez de la ligne de commande avec les fonctions comme plotfit, plotregression, plottrainstate et plotperform.