RAPPORT DE

PROJET ACQUISITION, TRAITEMENT ET MODELISATION STATISTIQUES DES DONNEES PHYSIOLOGIQUES.

1ère année cycle d'ingénieur ITS

Réalisation d'une IHM

- **Présenté par : Manal NEJMI - Alicia FERREIRA - Marina KHALIFA**
- **Encadré par : Mr. Fournier**
- **Année universitaire : 2020/2021**

Table des matières

Liste des figures

Remerciements

Avant de commencer, nous tenons à remercier toutes les personnes qui ont contribué à l'accomplissement de ce projet.

Nous adressons nos remerciements dans un premier temps à **Mr FOURNIER** qui nous a accompagné tout au long de ce projet, pour son aide et ses conseils dans plusieurs étapes du projet. Grâce aux cours qu'il nous a dispensé, on a pu apprendre un nouveau langage de programmation Matlab, et a contribué au développement de nos compétences de gestion de projet.

Nous voulions également remercier **Mr MELLOUK Abdelhamid**, le directeur de la filière ITS de l'EPISEN sans qui tout ceci n'aurait été possible.

Que tous trouvent en cette œuvre le couronnement de leurs efforts consentis et le témoignage de notre sincère gratitude.

Introduction

Au cours du module acquisition, traitement et modélisation, nous avions un projet de TP à réaliser avec notre enseignant Mr FOURNIER. Celui-ci a pour but de nous apprendre à construire une interface homme-machine sous un outil de programmation : Matlab. Cette interface va permettre de visualiser la régression statique. Elle comportera différents curseurs de donnés qui seront instancier par l'utilisateur et généra différents graphiques. L'intérêt de ces graphiques est de recenser des valeurs afin de calculer l'erreur relative d'un système linéaire en fonction des valeurs définis par l'utilisateur. Afin de réaliser ce projet de TP, nous avons travaillé en groupe. Notre équipe était constitué de KHALIFA Marina, NEJMI Manal et FERREIRA Alicia.

Ce projet nous a permis de développer nos compétences en programmation, notamment apprendre à coder en Matlab pour générer des interfaces hommes-machines, appliquer des fonctions mathématiques et en gestion de projet.

La problématique présentée au cours de ce TP est la suivante : Qu'est-ce que la régression linéaire simple ? Comment peut-on concevoir un signal bruité à partir d'un signal linéaire ?

Afin de réaliser ce TP nous devons suivre plusieurs étapes. Tout d'abord, l'élaboration d'une fonction de simulation. Dans un deuxième temps, nous avons procédé à l'extraction des données simulées pour que dans un troisième temps, nous puissions les analyser.

1- Simulation

Dans cette partie, nous avons créé une fonction simulation [x, y, bruit, yb] =simulationfi (N, sigma, class, a, b, AFF, REG) qui permet d'afficher une courbe de régression linéaire, un bruit gaussien, un histogramme gaussien ainsi qu'un signal bruité combiné à la régression linaire afin d'étudier le comportement des variables statistiques.

- **Régression linéaire**

Tout d'abord, nous cherchons à tracer une régression linéaire. Pour cela nous disposons d'une équation y = a*x +b avec comme entrée a et b (réels) et N (entier). En sortie, nous observons deux variables x et y.

X est une variable indépendante explicative supposée sans erreur. Tandis que y est une variable dépendante expliquée qui présente une erreur parfois importante. Le but de cette régression linéaire est d'estimer le bruit de mesure.

On insère des curseurs qui permettent de faire varier par l'utilisateur les valeurs d'entrée et qui génèreront comme sortie un graphique que l'on peut voir ci-dessous :

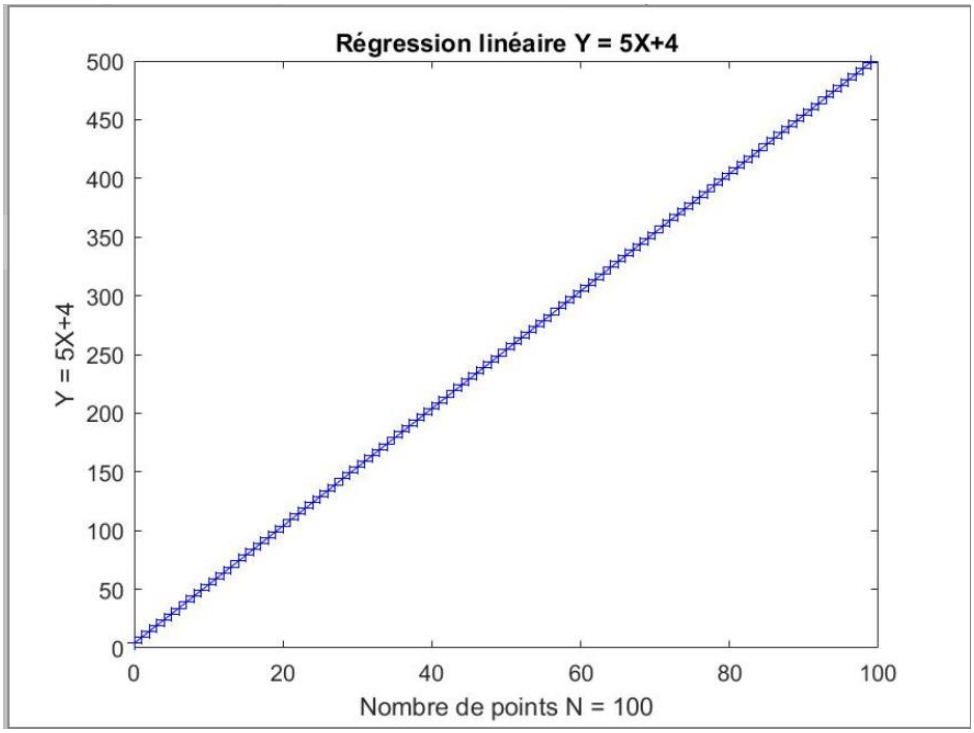


Figure 1 Signal y en fonction de x

- **Bruit gaussien & histogramme gaussien**

Pour le bruit gaussien nous avons comme entrée (N, sigma, class). En sortie, on a une perturbation réalisée de manière aléatoire. Ce bruit sera combiné avec la régression linéaire.

Nous avons pu représenter le bruit gaussien sous forme d'histogramme afin d'avoir une visualisation plus précise. Cet histogramme suit une loi normale centrée réduite (0,1).

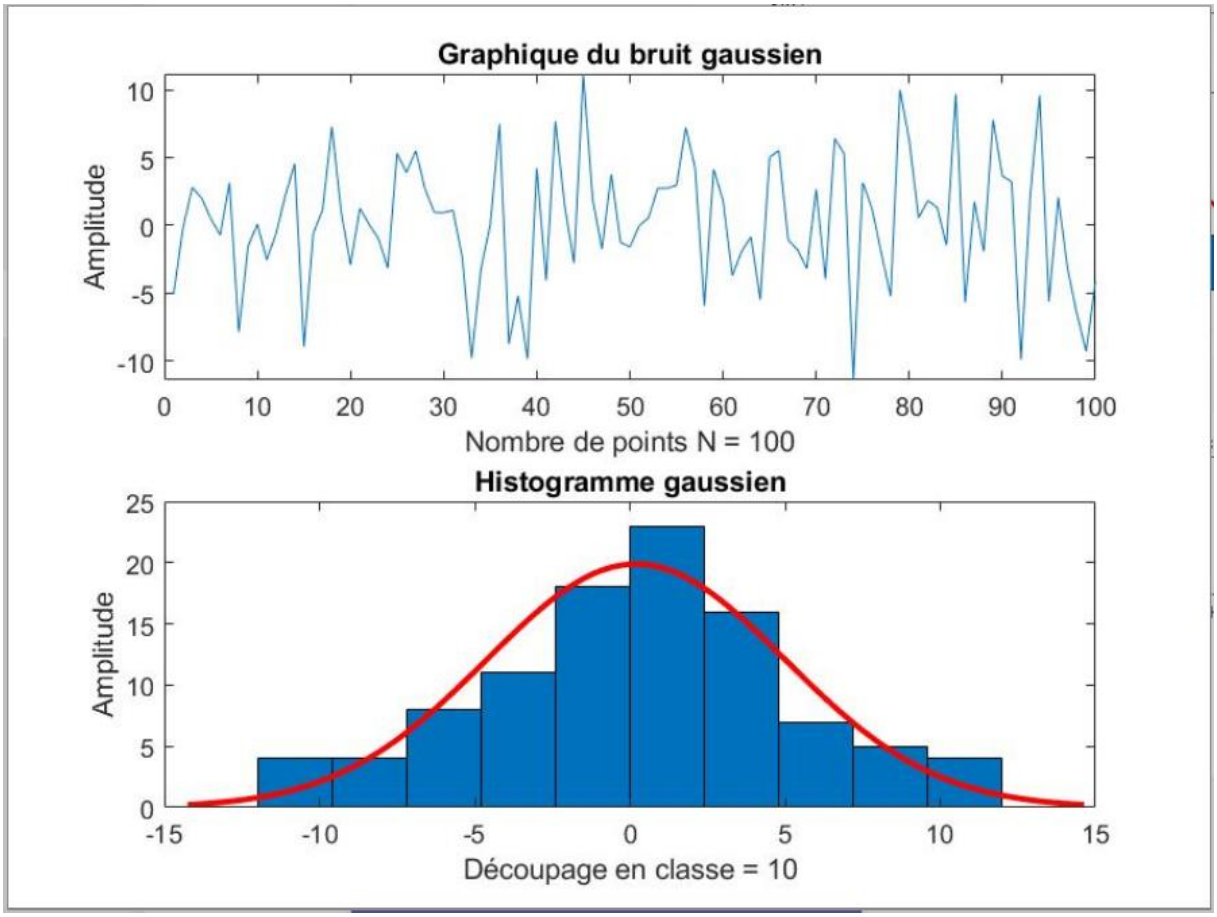


Figure 2 Signal de bruit en fonction de x - Histogramme de bruit selon la loi centré réduite (0,1)

- **Signal bruité**

Pour cette sous partie le but était de tracer le graphique d'un signal bruité. Nous avons utilisé la fonction yb= y+ bruit. Ce signal représente la fusion entre notre régression linéaire et notre bruit gaussien.

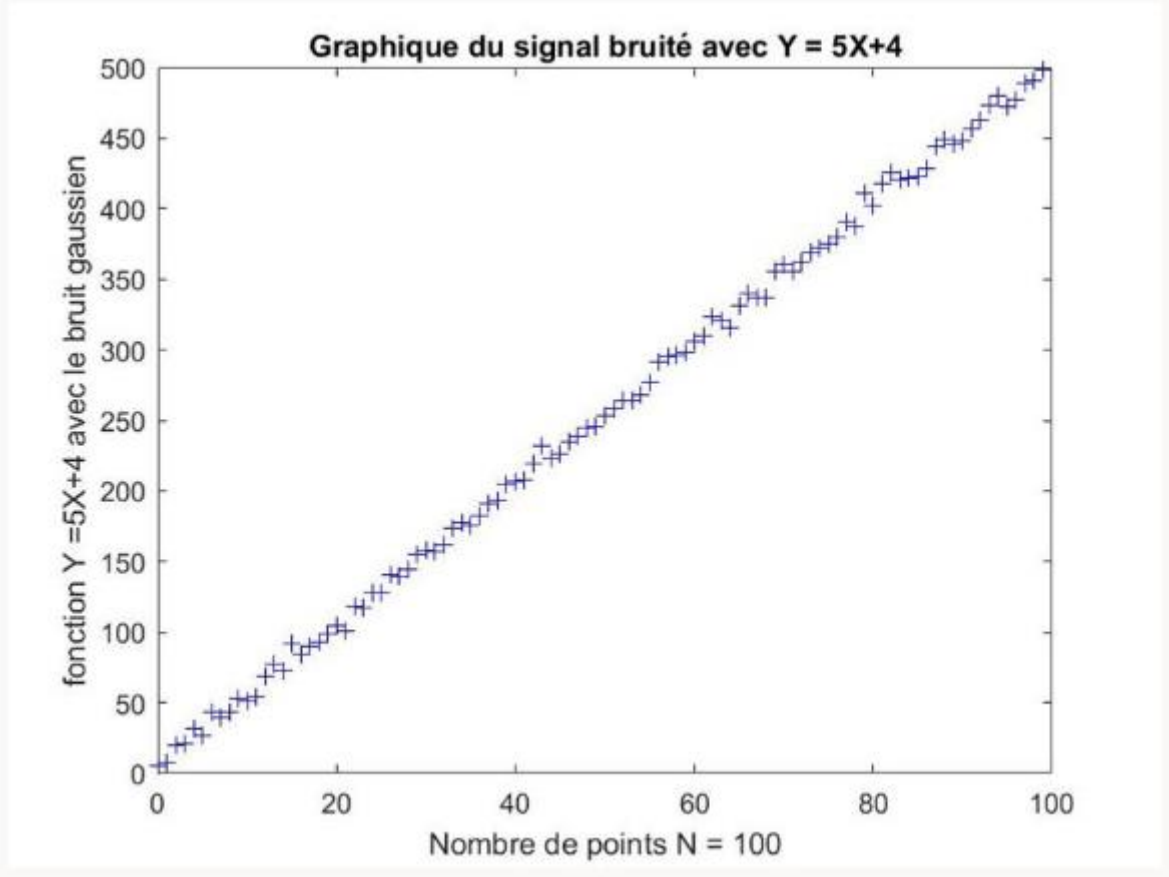


Figure 3 Signal bruité en fonction de x

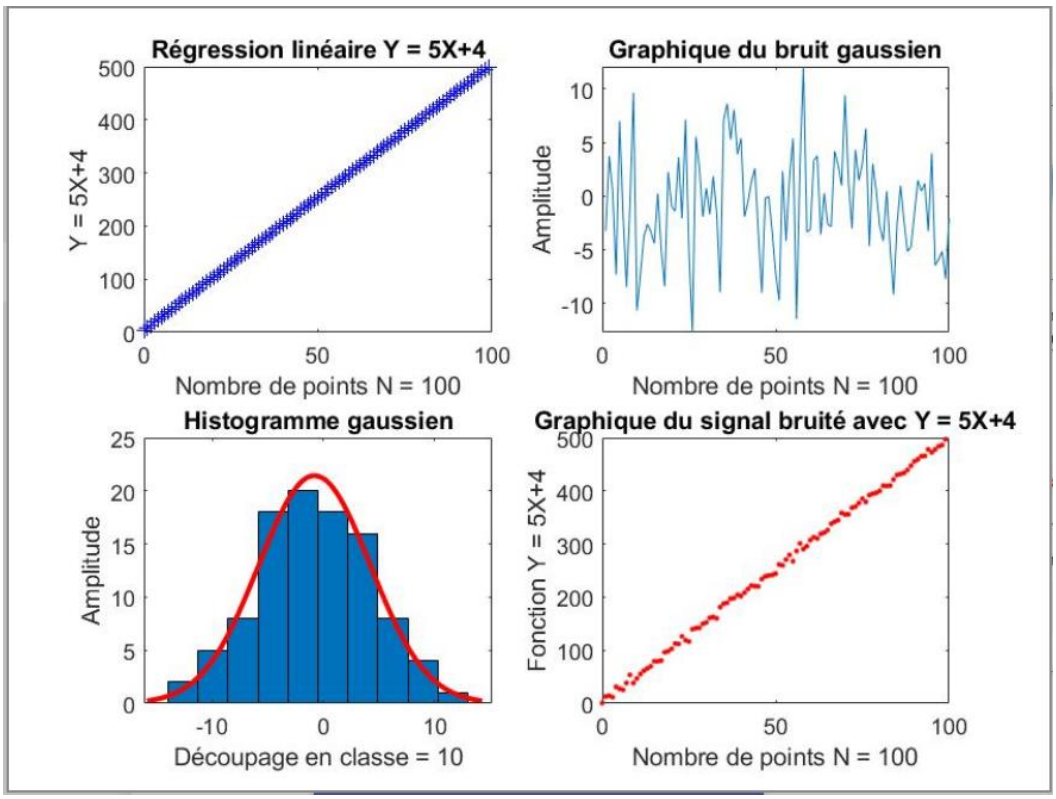


Figure 4 Affichage globale de nos graphes.

2- Extraction des données

Cette partie consiste à extraire les différentes variables de notre fonction de simulation. A partir des différents graphiques générés, nous avons obtenus des valeurs que nous allons pouvoir utiliser pour le reste du TP.

Ici nous allons avoir une nouvelle fonction [b0,b1,s0,s1,R2]=extrafi(x,yb,AFF). Grâce à cette fonction, nous aurons plusieurs paramètres tels que b0, b1 ainsi que leurs erreurs.

Afin de pouvoir déterminer le reste des variables, nous avons utilisé la méthode des moindres carrées. Cette méthode consiste à trouver les paramètres b0 et b1 qui rendent minimale la somme résiduelle des carrés des écarts entre la valeur observée (yb) et la valeur calculée (yREG ici).

Pour déterminer les variables b0 et b1, nous devons aussi passer par le calcul matriciel.

Dans notre code Matlab, nous avons comme matrice :

$$U = \begin{pmatrix} 1 & x1 \\ 1 & x2 \\ 1 & x3 \\ 1 & xn \end{pmatrix} \quad * \quad U' = A$$

C= yb*U' donc une fois A et C déterminés on peut trouver B= inv(A)*C.

Grâce à la matrice B on obtient nos variables b0 et b1.

Ainsi SSR = $\Sigma(yb-yREG)^2$

Sachant que

- yREG= bo+b1*xREG
- yb= y+bruit
- y=a*x+b

Pour connaître les erreurs sur les variables b0 et b1, nous devons calculer la matrice de co-variance grâce à la formule V= Vr*inv(A). Pour pouvoir calculer ce paramètre V nous devons aussi déterminer Sr^2.

Après avoir calculer, b0, b1, V et Vr (variance résiduelle) il nous reste à déterminer le bruit de mesure qui consiste à soustraire notre signal bruité au bruit estimé (yb-yREG). Ainsi b1 (valeur estimée) qui sera plus au moins égale à a (valeur théorique) + ou - s1.

b0(valeur estimée) qui doit être plus au moins égale à b (valeur théorique) + ou – s0.

La dernière étape consiste à déterminer le coefficient de détermination, pour cela nous utilisons la formule SSt= SSe+ SSr. SSr doit tendre vers 1 donc R^2= SSe/SSt.

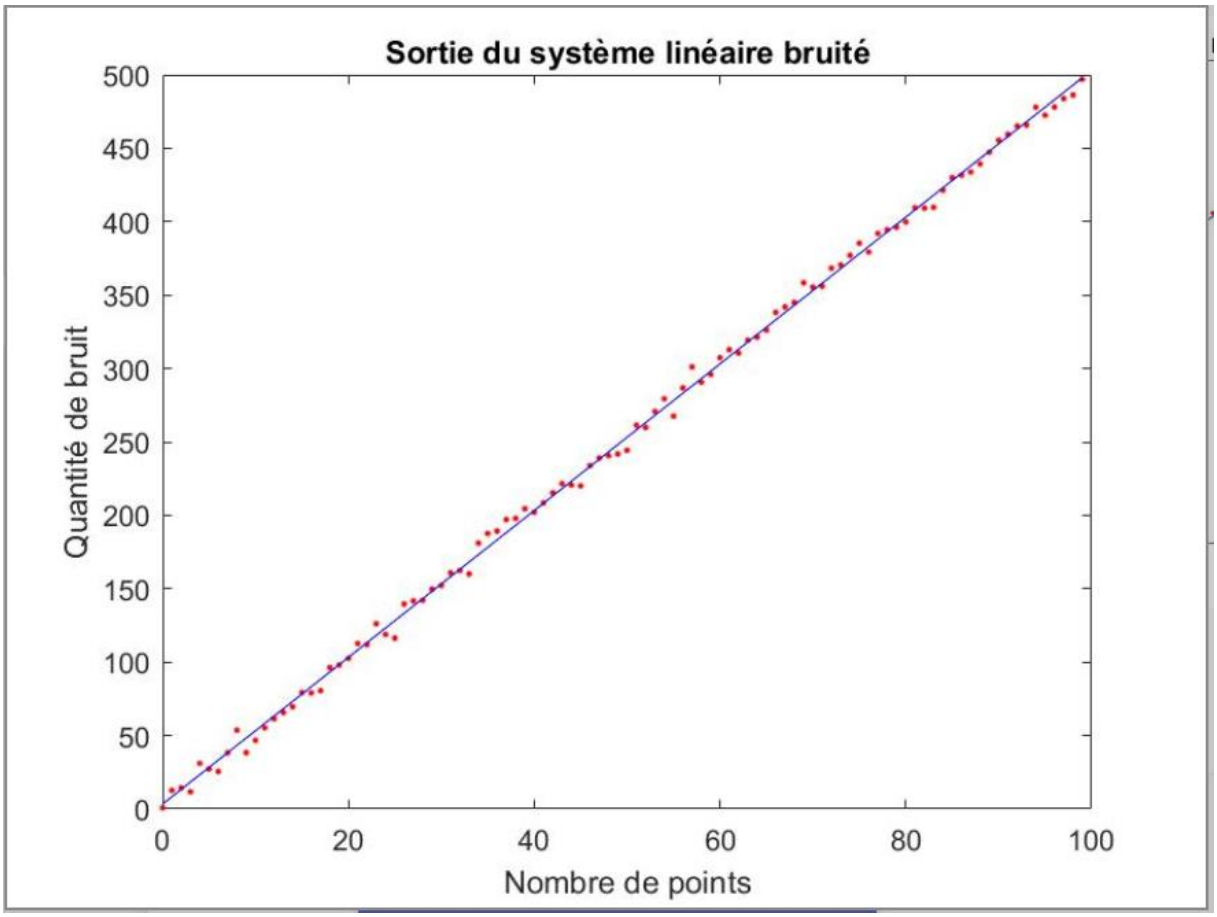


Figure 5 Signal observé et calculé

3- Analyse des résultats

Les différents objectifs réalisés durant ce TP sont les suivants :

En ce qui concerne la partie **modélisation,** nous avons pu réaliser une simulation d'un système dont la réponse est linéaire grâce à une fonction permettant d'avoir différents paramètres choisis par l'utilisateur à travers cinq sliders a,b,N,sigma,class :

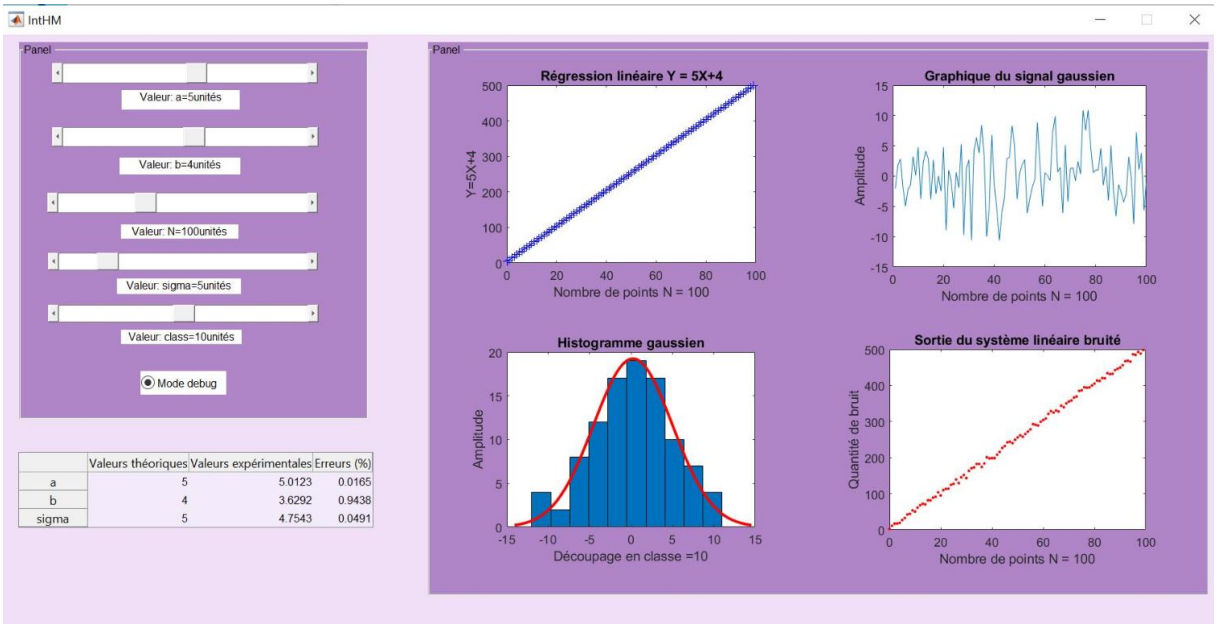
- Régression linéaire
- Bruit gaussien
- Histogramme gaussien
- Signal bruité

Cependant, le mode debug n'est pas fonctionnel malgré l'insertion d'un radio-bouton et de la fonction AFF=1.

Ensuite, pour la partie **représentation du bruit,** on a une génération d'un graphe qui affiche la distribution de bruit (dans un domaine temporel). On a ajouté un slider qui est l'écart-type. L'évolution de cet écart-type autour de la moyenne représente l'erreur de mesure. Pour la partie **génération des valeurs obtenus,** on a mis un tableau qui rassemble les valeurs théoriques et estimées, ainsi que leurs erreurs.

Il a fallu générer un fichier initialisation avec des valeurs prédéfinis afin de pouvoir afficher automatiquement nos graphiques.

Après avoir générer notre code, nous avons obtenu cette interface graphique :



	Valeurs théoriques	Valeurs expérimentales	Erreurs (%)
a	5	5.0123	0.0165
b	4	3.6292	0.9438
sigma	5	4.7543	0.0491

Figure 6 Interface graphique (version finale)

Conclusion

L'objectif de ce TP était de programmer une interface Homme-Machine utilisable afin d'obtenir différentes informations telles que le bruit de mesure, le coefficient de détermination. Au cours de ce TP, nous avons pu nous servir de différentes méthodes comme le calcul matriciel et la méthode des moindres carrées. Ce projet nous a aussi permis d'apprendre à programmer sous MATLAB, générer des graphiques, des curseurs, des tableaux pour présenter nos données.

En ce qui concerne le travail en groupe, nous trois n'avions aucune base de programmation en langage Matlab. Ce projet était une opportunité de découvrir ce langage et de concevoir une interface.

Ce projet nous a donc permis la maîtrise de nouvelles compétences utiles aux années qui vont venir.

Malheureusement, nous n'avons pas pu faire l'abaque cette année par manque de temps. Ainsi nous espérons pouvoir aborder cette notion l'année prochaine durant la continuité de nos cours avec Mr FOURNIER.

Evaluation de l'enseignement

Le module acquisition, traitement et modélisation a été très enrichissant pour nous et pour les années à venir. Marina et Alicia venant d'une licence de biologie santé ont pu apprendre les bases nécessaires aidant au bon déroulement de ce projet. Manal venant d'une licence biomédicale et une première année master en informatique biomédical a pu compléter le projet sur le côté organisationnel et répartition des parties de code. Le cours théorique dispensé au début nous a donc été très utile. Par ailleurs, nous n'avions jamais programmé en MATLAB. De ce fait, les petits tutorats nous ont semblés indispensable. Par moment, il nous est arrivé de ne pas savoir appliquer immédiatement certaines notions ou certains attendus. Cependant, grâce à la pédagogie et l'écoute de Mr FOURNIER nous avons réussis à rester concentrées et investies dans ce projet.

GLOSSAIRE

- **Clear all :** permet d'effacer toutes les variables
- **Close all :** permet de fermer toutes les fenêtres
- **Subplot(lign,colon,ind)**: permet de découper la fenêtre selon un certain nombre de ligne, de colonne et de pointer l'indice
- **Title (' chaîne de caractère')** : permet d'afficher un titre
- **Xlabel ('chaîne de caractère') :** permet d'afficher une légende pour l'axe des X
- **Ylabel ('chaîne de caractère')** : permet d'afficher une légende pour l'axe des Y
- **Plot :** permet d'afficher un graphique
- **Hist :** permet d'afficher un histogramme
- **[début:pas:fin]**: permet de construire un vecteur ligne
- **Length (vecteur)** : permet de récupérer le nombre d'élément composant le vecteur
- **Size (vecteur)** : permet de récupérer le nombre de ligne et de colonne constituant le vecteur
- **Disp('chaîne de caractère')** : permet d'afficher une chaîne de caractère
- **% :** mettre un commentaire sous MATLAB
- **Function nom_de_la_fonction(arguments) :** construction d'une fonction
- **Int2str** : permet de récupérer la valeur d'un entier et l'insérer dans une chaîne de caractère
- **Num2str :** permet de récupérer la valeur d'un réel et l'insérer dans une chaîne de caractère
- **Get**: on récupère une information
- **Set :** on positionne une information
- **Load ('nom-fichier')** : permet de récupérer toutes les données sauvegardées dans le fichier nomfichier.mat

ANNEXE

Code simulation

simulationfi.m

```
%on définit la fonction
function [x,y,bruit,yb]=simulationfi(N,sigma,class,a,b,AFF,REG)

%Affichage de la régression linéaire
%vecteur x[début:pas:fin]
x=[0:1:N-1];

% fonction affine qui représente l'équation linéaire
y=a*x+b;
%on affiche le graphique x en fonction de y pour la régression linéaire

if AFF
figure(1)
subplot(2,2,1);
plot(x,y,'+b-')
%titre
title(['Régression linéaire Y = ', num2str(a),'X+',num2str(b)])
%légendes
xlabel(['Nombre de points N = ',int2str(N)])
ylabel(['Y = ', num2str(a),'X+',num2str(b)])
%fin de l'affichage de la régression linéaire
end
%-------------------------------------------------------------
```

```
%-------------------------------------------------------------
%Affichage du bruit gaussien
bruit=sigma*randn(1,N);
if AFF
subplot(2,2,2)
% on trace le graphique de la gaussienne
plot(bruit)
% titre
title('Graphique du bruit gaussien')
xlabel(['Nombre de points N = ',int2str(N)])
ylabel('Amplitude')
end
%-------------------------------------------------------------
```

```
%-------------------------------------------------------------
%Affichage de l'histogramme
if AFF
subplot(2,2,3)
%on trace l'histogramme de la gaussienne

% utilisation du histfit à condition d'avoir "statistics and Machine Learning Toolbox"
histfit(bruit,class)
%titre
title('Histogramme gaussien')
%légendes
xlabel(['Découpage en classe = ',int2str(class)])
ylabel('Amplitude')
end
%-------------------------------------------------------------
```

```
%-------------------------------------------------------
%Affichage signal bruité
yb= y+bruit;
if AFF
subplot(2,2,4)
plot(x,yb,'.r')
%titre
title(['Graphique du signal bruité avec Y = ',num2str(a),'X+',num2str(b)])
%légendes
xlabel(['Nombre de points N = ',int2str(N)])
ylabel(['Fonction Y = ', num2str(a),'X+',num2str(b)])

end

if REG
    save resultat.mat x y yb bruit
end
```

Code extraction

extrafi.m

```
%fonction extraction
function [b0,b1,s0,s1,R2,Sr]=extrafi(x,yb,AFF)
% AFF=1
%Entrée:x,yb
%Sortie: b0,b1,s0,s1,R2,sigmaest

% %Importation
% load('resultat.mat');

%graphique
if AFF
% figure(1)
plot(x,yb,'.r')
%titre et légendes
title('Sortie du système linéaire bruité')
xlabel('Nombre de points')
ylabel('Quantité de bruit')
end
%U
U=[ones(length(x),1) x'];
%Transposée de U
U_t= U';
%Matrice A
A=U_t*U;
```

```
%Vecteur C
C=U_t*yb';
%Vecteur B
B=inv(A)*C;

b0=B(1,1)%ordonnée à l'origine
b1=B(2,1)%coeff pente

% Nouveau Y
yREG=b0+b1*x;%régression linéaire avec la pente b1 et b0 l'ordonné à l'origine
%figure(1)

if AFF
hold on
plot(x,yREG,'-b')
%title+légendes
hold off
end

%Estimation du sigma
bruitm=yb-yREG; %bruit de mesure
Sr=std(yb-yREG) %ecart-type résiduel
```

extrafi.m

```
%Matrice de variance
Vr=Sr^2;

%Matrice de Covariance
V=Vr*inv(A);

%écart-type
s0=sqrt(V(1,1))
s1=sqrt(V(2,2))

%Analyse de la variance
ymoy=mean(yb);

%somme expliquée des carrés des écarts
SSe=sum((yREG-ymoy).^2);

%somme totale des carrés des écarts
SSt=sum((yb-ymoy).^2);

%coeff de détermination
R2=SSe/SSt
end
```

Code IHM

```
function IntHM_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in texta future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to IntHM (see VARARGIN)
global a
global b
global N
global sigma
global class
global AFF
global REG

global x
global y
global bruit
global yb
global yREG

global b0
global b1
global s0
```

```
global s1
global R2
% init;
load('init.mat') %on charge les valeurs initiales de a,b,N,sigma,class

[x,y,bruit,yb]=simulationfi(N,sigma,class,a,b,0,1) %fonction simulation

[b0,b1,s0,s1,R2,Sr]=extrafi(x,yb,0)%fonction extraction

data= {a,b1,s1;b,b0,s0;sigma,Sr,abs(sigma-Sr)/sigma} %tableau valeurs théoriques, expérimentales et erreur
set(handles.table1,'Data',data)

%variable = valeur
set(handles.texta,'string',['Valeur: a=',num2str(a,2),'unités'])
set(handles.textb,'string',['Valeur: b=',num2str(b,2),'unités'])
set(handles.textN,'string',['Valeur: N=',int2str(N),'unités'])
set(handles.textsigma,'string',['Valeur: sigma=',num2str(sigma,2),'unités'])
set(handles.textclass,'string',['Valeur: class=',int2str(class),'unités'])
```

```
%graphique 1
axes(handles.axes1)
plot(x,y,'+b-')
%titre
title(['Régression linéaire Y = ', num2str(a),'X+',num2str(b)])
%légendes
xlabel(['Nombre de points N = ',int2str(N)])
ylabel(['Y=', num2str(a),'X+',num2str(b)])

%graphique 2
axes(handles.axes2)
% on trace le graphique de la gaussienne
plot(bruit)
% titre
title('Graphique du signal gaussien')
xlabel(['Nombre de points N = ',int2str(N)])
ylabel('Amplitude')
```

```
%graphique 3
axes(handles.axes3)
%on trace l'histogramme de la gaussienne
histfit(bruit,class)
%titre
title('Histogramme gaussien')
%légendes
xlabel(['Découpage en classe =',int2str(class)])
ylabel('Amplitude')

%graphique 4
axes(handles.axes4)
% subplot(2,2,4)

plot(x,yb,'.r')
%titre et légendes
title('Sortie du système linéaire bruité')
xlabel(['Nombre de points N = ', int2str(N)])
ylabel('Quantité de bruit')
% Choose default command line output for IntHM
handles.output = hObject;
```

```
% --- Executes on slider movement.
function texta_Callback(hObject, eventdata, handles)
% hObject    handle to texta (see GCBO)
% eventdata  reserved - to be defined in texta future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider

global a
a=get(hObject,'Value')
set(handles.texta,'string',['Valeur: a=',num2str(a,2),'unités'])
```

```
% --- Executes during object creation, after setting all properties.
function texta_CreateFcn(hObject, eventdata, handles)
% hObject    handle to texta (see GCBO)
% eventdata  reserved - to be defined in texta future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
global a
% load init.mat %on charge le fichier avec les valeurs
texta=findobj(0,'tag','texta')
set(texta,'Value',a)
% Hint: slider controls usually have texta light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end
```

```
% --- Executes on slider movement.
function textb_Callback(hObject, eventdata, handles)
% hObject    handle to textb (see GCBO)
% eventdata  reserved - to be defined in texta future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider

% init(N,sigma,class,a,b)
global b
b=get(hObject,'Value')
set(handles.textb,'string',['Valeur: b=',num2str(b,2),'unités'])
```

```
% --- Executes during object creation, after setting all properties.
function textb_CreateFcn(hObject, eventdata, handles)
% hObject    handle to textb (see GCBO)
% eventdata  reserved - to be defined in texta future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
global b
% load init.mat %on charge le fichier avec les valeurs
textb=findobj(0,'tag','textb')
set(textb,'Value',b)
% Hint: slider controls usually have texta light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end
```

```
% --- Executes on slider movement.
function textN_Callback(hObject, eventdata, handles)
% hObject    handle to textN (see GCBO)
% eventdata  reserved - to be defined in texta future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider

% init(N,sigma,class,a,b)
global N
N=get(hObject,'Value')
N=300*N
N=floor(N)
set(handles.textN,'string',['Valeur: N=',int2str(N),'unités'])
```

```
% --- Executes during object creation, after setting all properties.
function textN_CreateFcn(hObject, eventdata, handles)
% hObject    handle to textN (see GCBO)
% eventdata  reserved - to be defined in texta future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
global N
% load init.mat
textN=findobj(0,'tag','textN')
set(textN,'Value',N/300)
% Hint: slider controls usually have texta light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end
```

```
% --- Executes on slider movement.
function textsigma_Callback(hObject, eventdata, handles)
% hObject    handle to textsigma (see GCBO)
% eventdata  reserved - to be defined in texta future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider

% init(N,sigma,class,a,b)
global sigma
sigma=get(hObject,'Value')
sigma=30*sigma
set(handles.textsigma,'string',['Valeur: sigma=',num2str(sigma,2),'unités'])
```

```
% --- Executes during object creation, after setting all properties.
function textsigma_CreateFcn(hObject, eventdata, handles)
% hObject    handle to textsigma (see GCBO)
% eventdata  reserved - to be defined in texta future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
global sigma
% load init.mat
textsigma=findobj(0,'tag','textsigma')
set(textsigma,'Value',sigma/30)
% Hint: slider controls usually have texta light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end
```

```
% --- Executes on slider movement.
function textclass_Callback(hObject, eventdata, handles)
% hObject    handle to textclass (see GCBO)
% eventdata  reserved - to be defined in texta future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider

global class
class=get(hObject,'Value');
class=20*class
class=floor(class)
set(handles.textclass,'string',['Valeur: class=',int2str(class),'unités'])
```

```
% --- Executes during object creation, after setting all properties.
function textclass_CreateFcn(hObject, eventdata, handles)
% hObject    handle to textclass (see GCBO)
% eventdata  reserved - to be defined in texta future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
global class
% load init.mat
textclass=findobj(0,'tag','textclass')
set(textclass,'Value',class/20)
% Hint: slider controls usually have texta light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end
```

```
% --- Executes on button press in radiobutton1.
function radiobutton1_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton1
global AFF
AFF=1;
```