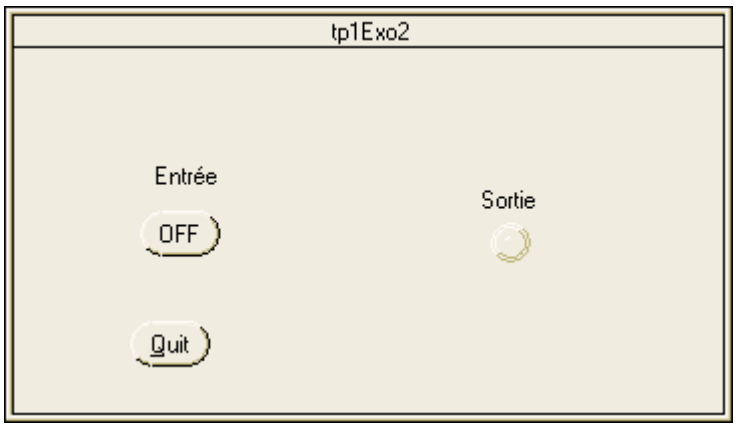
TP 1 : Introduction

Exercice 1

Utilisation du compilateur de manière classique. On a besoin que d'un programme C mais il faut faire un projet. On n'est pas obligé d'ajouter les include, le compilateur vous les proposera à la compilation et il faudra dire oui. Cet exercice est réalisé avec vidéoprojecteur par l'enseignant.

Exercice 2

L'enseignant vous montre comment faire fonctionner un programme simple avec une face avant :



Command button : quit et callback quit
Toggle Button (ON/OFF) avec comme Label "Entrée", comme Callback Name On_Off et comme Constant Name TOGGLEBUTTON)
Led avec label Sortie sans callback
Le nom de la face avant est positionné par : Edit -> Panel ->Panel Title
Vous êtes prêt à générer le canevas du programme C correspondant : Code -> Generate -> All Code : Une boite de dialogue s'ouvre : en bas il faut cocher notre callback quit comme "quit when event COMMIT recieved" ce qui permettra de quitter en appuyant sur le bouton "Quit".
Voici le programme généré :

```
#include <cvirte.h>
#include <userint.h>
#include "TP1exo1.h"

static int panelHandle;

int main (int argc, char *argv[])
{
      if (InitCVIRTE (0, argv, 0) == 0)
            return -1;  /* out of memory */
      if ((panelHandle = LoadPanel (0, "TP1exo1.uir", PANEL)) < 0)
            return -1;
      DisplayPanel (panelHandle);
      RunUserInterface ();
      DiscardPanel (panelHandle);
      return 0;
}

int CVICALLBACK On_Off (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
```

```
{
      switch (event)
            {
            case EVENT_COMMIT:
                  break;
            }
      return 0;
}

int CVICALLBACK quit (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      switch (event)
            {
            case EVENT_COMMIT:
                  QuitUserInterface (0);
                  break;
            }
      return 0;
}
```

Votre seul travail consistera à compléter le callback On_Off comme ceci :

```
int CVICALLBACK On_Off (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{   int etat;
      switch (event)
            {
            case EVENT_COMMIT:
               GetCtrlVal(panelHandle,PANEL_TOGGLEBUTTON,&etat);
               SetCtrlVal(panelHandle,PANEL_LED,etat);

                  break;
            }
      return 0;
}
```

Son objectif est de lire l'état du bouton ON/OFF avec un GetCtrlVal dans une variable etat déclarée localement et de positionner la LED allumée ou éteinte avec SetCtrlVal.
Remarquez que les noms des bouton ou led est précédé par PANEL_

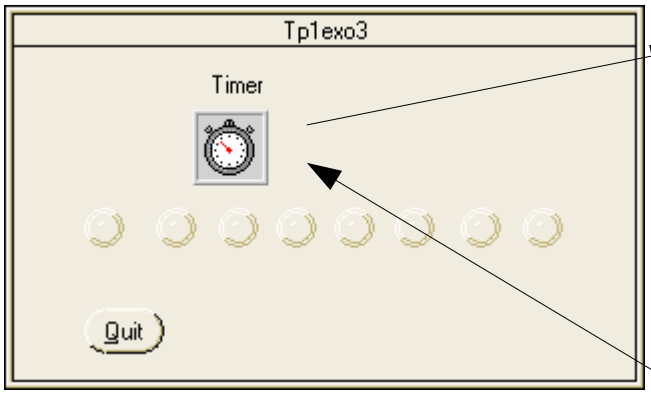
TP2 : découverte du Timer et de l'affichage numérique

Exercice 1

On désire compter sur 8 bits au rythme d'un timer. La face avant ne comportera donc que huit leds (appelées LED, LED_2, ...,LED_8) et un bouton quit. De plus, il faudra lui ajouter un timer avec son callback associé :

Type	***timer***
Constant Name	TIMER
Callback Function	MaJ
Interval (seconds)	1.000
Label	Timer

.Cela peut ressembler à la figure ci-dessous :




```
int CVICALLBACK MaJ (int panel, int control,
int event,
            void *callbackData, int
eventData1, int eventData2)
{
      switch (event)
            {
            case EVENT_COMMIT:

                  break;
            }
      return 0;
}
```

Le callback MaJ sera responsable l'incrémentation d'une variable (locale statique) sur 8 bits et de son affichage sur 8 leds. On procédera de la manière suivante : incrémentation de la variable puis conversion dans un tableau de 8 cases entières puis appel de 8 SetCtrlVal pour chacune des 8 leds. On vous donne le canevas ci-dessous que vous avez à compléter.

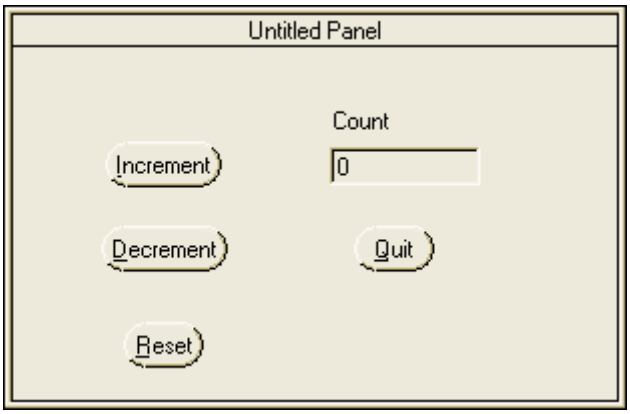
```
int CVICALLBACK MaJ (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)

{   int leds[8];
   static unsigned char nb=0; // explication du terme static !!!!

      switch (event)
            {
            case EVENT_TIMER_TICK:
                        // calcul de la nouvelle valeur
                          nb++;
            conversion(nb,leds);
            // Mise à jour des leds
             SetCtrlVal(panelHandle,PANEL_LED,leds[0]);
             SetCtrlVal(panelHandle,PANEL_LED_2,leds[1]);
             ....
}
```

`void conversion(unsigned char nb, int result[8]);` pourra être réalisé comme ci-dessous :

```
void conversion(char nb,int result[8]){
  char i;
  for(i=7;i>=0;i--) if ((nb & (1<<i)) == (1<<i)) result[i]=1;
                    else result[i]=0;
}
```

Travail optionnel (pour les plus rapides). Reprendre la face avant et faire un chenillard avec déplacement d'une led allumée.

Exercice 2 (Découverte de l'affichage d'une valeur numérique)
Soit la face avant suivante :



Si vous désirez changer le titre de la fenêtre double-cliquez sur la face avant que vous êtes en train de créer.
Pour créer le contrôle numérique : clic droit -> numéric, le premier contrôle par exemple avec les valeurs suivantes :

Constant Name	***Default Value***	***Data Types***	***Minimum***	***Maximum***	***Inc Value***	***Range Checking***	***Control Mode***	***Label***
NUMERIC	0	int	-2147483648	2147483647	1	Notify	Indicator	Count

Puis créer les boutons :

Button Name	Increment	Decrement	Reset	Quit
Constant Name	INCREMENT	DECREMENT	RESET	QUIT
Callback Function	AddOne	SubstractOne	SetToZero	Quit
Control Mode	Hot	Hot	Hot	Hot
Label	__Increment	__Decrement	__Reset	__Quit

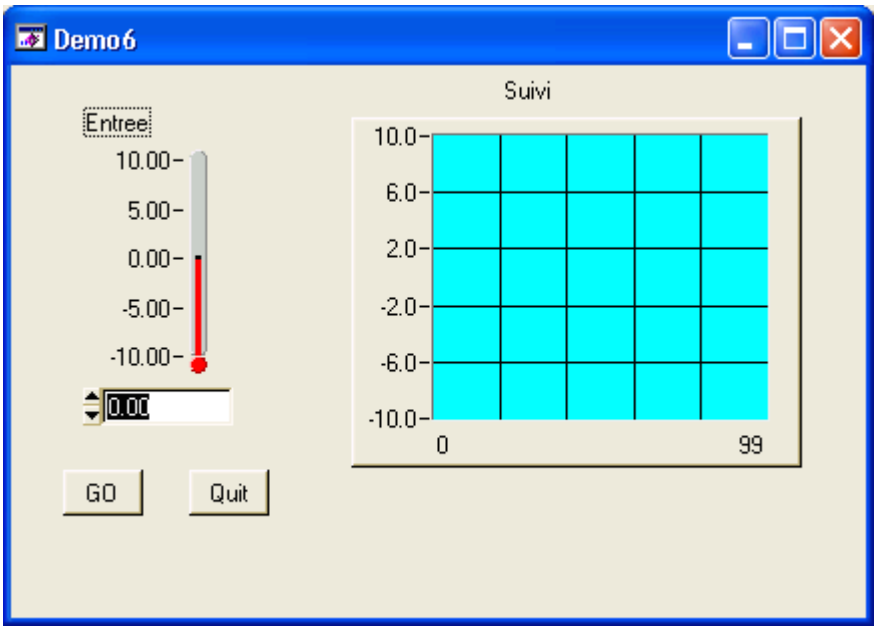
Faire fonctionner l'ensemble.

TP3 : StripChart et Thermomètre

I) Quelques fonctions

Exercice 1

Soit la face avant suivante



Un thermomètre en entrée et un stripchart en sortie.
Thermomètre (Create->Numeric->Thermometer) en « Control mode » : Hot pour qu'il fonctionne correctement à la souris.

On veut faire communiquer le thermomètre avec le strip chart.
Dans un premier temps, c'est à chaque fois que l'on positionne le thermomètre que sa valeur sera envoyée dans le stripchart. Une autre manière de dire les choses c'est qu'il faut un callback associé à ce thermomètre et que ce callback sera charger de :

```
// lire la valeur du thermomètre et la mettre dans une variable :
GetCtrlVal(panelHandle,PANEL_NUMERICTHERM,&voie[0]);
// mise de voie[0] dans le stripchart
PlotStripChart(panelHandle,PANEL_STRIPCHART,voie,1,0,0,VAL_DOUBLE);
```

On est obligé d'utiliser un tableau (de double ici) pour voie à cause de stripchart qui en nécessite absolument un.
Le « toggle button » GO/STOP n'est pas utile dans cet exercice.

Exercice 2

On cherche à faire la même chose, mais la recopie dans le stripchart sera cadencée par un timer. Le timer sera ajouté en face avant avec une fonction de rappel

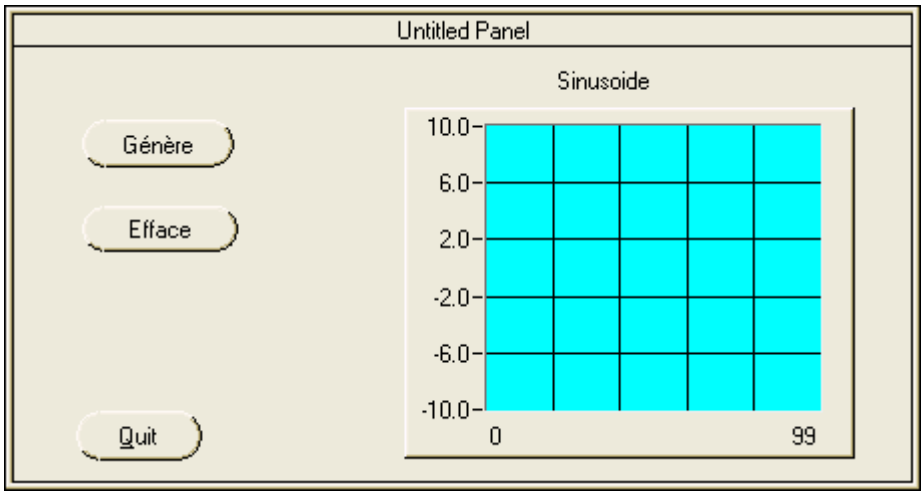
```
int CVICALLBACK Recopie (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
  switch (event){
```

```
    case EVENT_TIMER_TICK:
      // Recherche de la position du toggle button GO/STOP avec GetCtrlVal
      // sa valeur est mise dans une variable go
      if (go) {
        // mise dela valeur du thermomètre dans une variable :
        GetCtrlVal(panelHandle,PANEL_NUMERICTHERM,&voie[0]);
        // mise de voie[0] dans le stripchart
        PlotStripChart(panelHandle,PANEL_STRIPCHART,voie,1,0,0,VAL_DOUBLE);
      }
      break;
    }
   return 0;
}
```

On utilisera le « toggle button » GO/STOP dans cet exercice : un appui sur GO lancera la recopie et le bouton affichera alors STOP. Un appui sur stop arrêtera la recopie.

Exercice 3

Remplacer le « toggle button » GO/STOP par un « command button » de label « génère », ajouter un bouton « Efface» et retirer le thermomètre. Cela doit vous donner la face avant ci-dessous. L'appui sur « génère » génère une sinusoïde d'une période (100 points) dans le strip chart et l'appui sur « Efface » efface le stripchart (sous-programme `ClearStripChart(panelHandle, PANEL_STRIPCHART)`



Indications : Il existe une fonction toute faite pour générer une sinusoïde dans un tableau :

```
SinePattern (100, 1.0, 0.0, 1.0, sinus);
```

où 100 est le nombre de points à générer, 1.0 est l'amplitude, 0.0 est la phase, 1.0 est le nombre de cycles et sinus est le tableau dans lequel on va ranger tout cela (`double sinus[100];`).

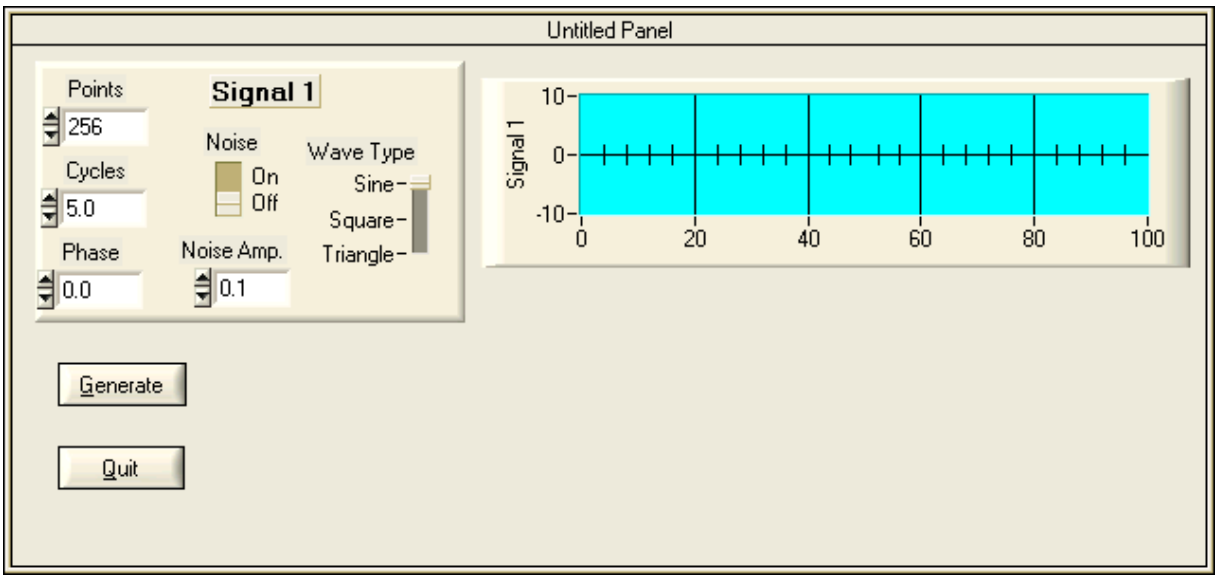
Le fait que l'on garde un stripchart complique un peu le dessin de la sinusoïde qui doit impérativement se faire avec une boucle for :

```
for(i=0;i<100;i++)
  PlotStripChart(panelHandle,PANEL_STRIPCHART,&sinus[i],1,0,0,VAL_DOUBLE);
```

Il serait plus intéressant d'utiliser un graph.

TP4 : Générations de fonctions

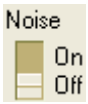
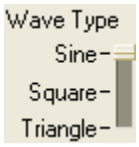
On vous demande de faire fonctionner complètement la face avant (inspirée de l'exemple (C:\Program Files\National Instruments\MeasurementStudio\CVI\samples\analysis\convolve.prj) donnée ci-dessous.



On vous présente les particularités des contrôles, noms et callbacks associés.

Contrôles	Points 256	Cycles 5.0	Phase 0.0	Noise Amp. 0.1
type	Ring	Numeric	Numeric	Numeric
Constant Name	SIG1POINTS	SIG1CYCLES	SIG1PHASE	SIG1NOISEAMP
callback	Getsig1points	Getsig1cycles	Getsig1phase	Getsig1noiseamp
Mode	Hot	Hot	Hot	Hot
particularités	Label/Value pairs 256 256 512 512 1024 1024 2048 2048	Data Type Double Default Value 5.0 minimum 0.1 maximum 100 IncValue 1.0 Range Checking Notify	Data Type Double Default Value 0.0 minimum -180 maximum 180 IncValue 1.0 Range Checking Notify	Data Type Double Default Value 0.1 minimum 0.1 maximum 5 IncValue 0.1 Range Checking Notify

Le contrôle « Points » peut prendre les valeurs 256, 512, 1024 et 2048.
Le contrôle « Cycles » prend des valeurs doubles.
Le contôle « Phase » détermine la phase.
Le contrôle « Noise Amp » détermine l'amplitude du bruit s'il est sélectionné (Noise on)
Le contrôle « Wave Type » permet de choisir entre une sinusoïde un signal carré ou triangulaire.

Contrôles				
type	Binary switch	Ring Slide	Command Button	Command Button
Constant Name	SIG1NOISE	SIG1WAVETYPE	GENERATESIG1	QUIT
callback	Getsig1noise	Getsig1wavetype	Generatesig1	Quit
Mode	Hot	Hot	Hot	Hot
particularités	Data type Int Initial State Off ON Value 1 OFF Value 0 ON Text ON OFF Text OFF	Label/Value pairs Sine 0 Square 1 Triangle 2		

Le contrôle graphique est un Graph (valeur SIG1GRAPH). Sur « Left Y-Axis » il faut retirer « auto scale » pour modifier les ordonnées en -10.0 +10.0 et retirer « auto division » pour le passer à 2. La même chose (éventuellement) sur « Right Y-Axis » donnera l'apparence ci-dessus.

Exercice 1

Réaliser la face avant complète et générer le code associé.
Chaque callback est responsable de la mise à jour d'une variable globale associée : On s'intéressera dans cet exercice aux trois premières variables ci-dessous :

```
static int nbPts = 256; // Contrôle Points
static double sig1cycles = 5.0; //Contrôle Cycle
static double sig1phase = 0.0; // Contrôle Phase
static double wave1[2048];
```

Mettre à jour les trois callback associés : Getsig1points, Getsig1cycles et Getsig1phase pour qu'ils positionnent correctement ces trois variables.
Ecrire le callback associé au bouton « Generate » pour qu'il génère une sinusoïde dans un tableau (de 2048 cases) et qu'il la dessine dans le graph.
Indications : Reprendre SinePattern pour générer la sinusoïde (d'amplitude 7.0) :

```
SinePattern (nbPts, 7.0, sig1phase, sig1cycles, wave1);
```

Le dessin dans le graphe se fait avec :

```
DeleteGraphPlot (panelHandle, PANEL_SIG1GRAPH, -1, VAL_IMMEDIATE_DRAW);
PlotY (panelHandle, PANEL_SIG1GRAPH, wave1, nbPts, VAL_DOUBLE,
      VAL_THIN_LINE, VAL_EMPTY_SQUARE, VAL_SOLID, 1, VAL_RED);
```

Exercice 2

Compléter l'exercice 1 en traitant les variables globales :

```
static int sig1noise = 0;
static double sig1noiseamp = 0.1;
static int seed = 1;
static double noisewave1[2048];
```

et les callbacks associés : Getsig1noise, et Getsig1noiseamp
Indications : la génération de bruit se fait avec dans le tableau noisewave1

```
GaussNoise (nbPts, sig1noiseamp, seed, noisewave1);
```

Si la variable sig1noise est 1 on ajoute les deux tableaux noisewave1 et wave1 dans wave1 avant de l'envoyer au graphe.

Exercice 3
Compléter l'exercice 2 en traitant la variable globale :

```
static int sig1wavetype = 0;
```

et le callback associé : `Getsig1wavetype`

Indications : la génération de signal carré se fait avec :

```
SquareWave (nbPts, 7.0, (sig1cycles/nbPts), &sig1phase,50.0, wave1);
```

50.0 est le rapport cyclique en %. Le deuxième paramètre est l'amplitude tandis que le troisième est la fréquence.

la génération de signal triangulaire se fait avec :

```
TriangleWave (nbPts, 7.0, (sig1cycles/nbPts), &sig1phase,wave1);
```

Le deuxième paramètre est l'amplitude tandis que le troisième est la fréquence.

TP5 : Systèmes d'équations surdéterminés et moindres carrés

Le but de ce TP est d'approfondir l'évaluation par les moindres carrés et de l'appliquer à la mesure de phase.

I) Systèmes surdéterminés d'équations

1°) Présentation

Lorsqu'un système d'équations comporte plus d'équations que d'inconnues, il est dit surdéterminé. Sauf cas exceptionnel un tel système d'équations n'a pas de solution exacte. De tels systèmes se rencontrent souvent en métrologie, en traitement des signaux sismiques ou en traitement de la parole. Formalisons un peu. Le système est représenté par l'équation matricielle :

H.b = y

où $y = [y1\ y2\ ...\ yQ]^t$ (^{t} désignant la transposition) et $b = [b1\ b2...\ bN]^t$ avec Q>N. H et y sont supposés connus et b est un ensemble d'inconnues.
Puisqu'il n'y a pas de solution exacte, la solution cherchée aura alors la propriété suivante : si l'on calcule

z = H.b

il doit être le plus proche possible de y en un sens qu'il convient de déterminer. Le plus proche possible est en général pris au sens de la norme habituelle des vecteurs, autrement dit on cherchera à rendre minimale le vecteur : e = y - z ou plutôt sa norme $E = (y - z)^t(y - z)$

Théorème : On montre que la solution d'un tel système est :

$$\mathbf{b = (H^t.H)^{-1}H^t.y}$$

Ce théorème est très important car il nous montre la voie à suivre pour résoudre le système sans passer par les calculs de la dérivée que l'on annule.

2°) Au-delà du théorème

Les théorèmes sont bien gentils mais si l'on suit celui-ci, on s'aperçoit que l'on a une inversion de matrice à réaliser. Heureusement LabWindows dispose d'un ensemble de primitives qui permettent de résoudre ce problème (pas l'inversion de matrice mais carrément la résolution des moindre carrés). La seule chose qu'il reste à faire est alors de poser correctement le problème.

Exercice 1
Résoudre le système surdéterminé suivant :

$2.b_1 + b_2 = 1$

$b_2 = 1$

$b_1 + 2.b_2 = 1$

Ce système peut être écrit comme :

$2.b_1 + 1.b_2 = 1$

$0.b1 + 1.b_2 = 1$ ou encore : $\begin{pmatrix} 2 & 1 \\ 0 & 1 \\ 1 & 2 \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$

$1.b_1 + 2.b_2 = 1$

avec $H=\begin{pmatrix}2 & 1\\ 0 & 1\\ 1 & 2\end{pmatrix}, b=\begin{pmatrix}b_1\\ b_2\end{pmatrix}, y=\begin{pmatrix}1\\ 1\\ 1\end{pmatrix}$

La primitive que l'on va utiliser s'appelle GenLSFitCoef de prototype :

```
GenLSFitCoef (void *H, int n, int k, double y[], double b[], int algorithm);
```

n est le nombre d'équations et sera donc 3, k est le nombre d'inconnues sera donc 2, algorithme sera pris à 0.
H sera une variable : `double H[3][2];` et sera initialisée dans le programme.
b sera un tableau `double b[2];` qui contiendra le résultat et y sera double y[3]; et initialisée dans le programme.

3°) Méthode des moindres carrés

Le calcul des moindres carrés consiste aussi à résoudre un système surdéterminé. On veut en général minimaliser suivant un critère quadratique une certaine erreur.

Exercice 2

Nous considérons une boule qui roule à une vitesse constante v. La relation entre le temps t et la position y de la boule est donnée par l'équation linéaire y = a + v.t
Un ensemble de mesures nous donnent les résultats suivants :

t	0	3	4	7	8	10
y	1	4	5	6	7	10

On cherche à trouver a et v qui rendent compte le mieux possible de l'équation linéaire.
Ecrire le système d'équations surdéterminé que l'on doit résoudre. Le récrire sous forma matricielle : trouver ainsi H et y
Résoudre le système.

Exercice 3

Déterminer l'angle de tir de la trajectoire parabolique issue de l'origine qu'on estime par la méthode des moindres carrés disposant des points P1(1;2), P2(2;3), P3(3;3.2) P4(4;3.5) P5(5;2.5).
Optionnel : Grapher la trajectoire estimée (et si possible les points de mesure). On utilisera par exemple

```
PlotXY (, , , , , VAL_DOUBLE, VAL_DOUBLE, VAL_THIN_LINE,
        VAL_EMPTY_SQUARE, VAL_SOLID, 1, VAL_RED);
```

qui nécessite une face avant car un handle.

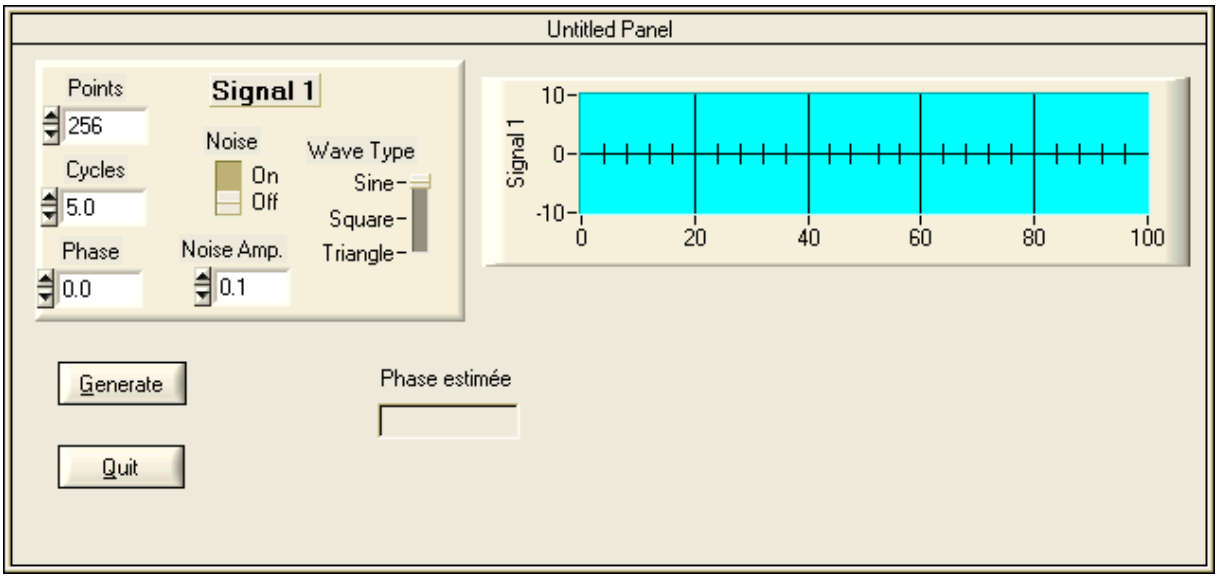
Indications : la trajectoire est décrite par l'équation $y=ax^2+bx$ et on cherche à estimer a et b. On écrit donc l'équation pour chaque point et on résout. Et comment fait-on pour trouver l'angle de tir ? Quel est le rapport avec la dérivée à l'origine ?

II) Application finale

Exercice 4

Recherche d'une phase
On cherche à évaluer la phase d'un signal sinusoïdal dont on connaît la fréquence. Ce sous-programme sera utilisé plus tard avec des signaux qui viennent d'un oscilloscope.
On définit pour cela une structure :

```
struct infoSinus {
  double amplitude,phase,offset;
};
```
qui sert à donner les trois paramètres importants d'une sinusoïde, à savoir, son amplitude, sa phase et son offset.



Cette interface peut être réalisée facilement à partir de celle du TP précédant. Il suffit d'ajouter un contrôle numérique (voir exercice 2 TP2) appelé phase estimée.

Ajouter un sous programme qui soit capable de retrouver les informations importantes de la sinusoïde :

```
struct infoSinus calculModPhas();
```

Dans ce sous-programme, on générera dans un tableau de 2048 valeurs une sinusoïdale. avec une certaine phase et un nombre de points déterminé par l'utilisateur (voir TP précédant). Générer ensuite une sinusoïde vraie (sans phase) et d'amplitude 1 dans un tableau sine[i], de même période que la première sinusoïde.
Générer un cosinus (sinusoïdal de phase 90°) d'amplitude 1 dans un tableau cos[i]
On cherche alors par la méthode des moindre carrés 3 coefficients qui modélisent au mieux nos 500 données sous la forme :
wave1[i] = b0 + b1*sine[i] + b2cos[i]
Construire la matrice H de ce problème (par programme).
Déduire de ces trois coefficients, l'offset, le module et la phase.
Vérifiez que tout ce que vous obtenez correspond bien à votre sinusoïde de départ.
Grapher la sinusoïde générée et la sinusoïde estimée :

```
PlotY (panelHandle, PANEL_SIG1GRAPH, waveEst, nbPts, VAL_DOUBLE,
          VAL_THIN_LINE, VAL_EMPTY_SQUARE, VAL_SOLID, 1, VAL_BLUE);
```
graphe la sinusoïde estimée en bleu (si elle est dans le tableau waveEst et qu'elle comporte nbPts points.
Ce sous-programme sera appelé à chaque génération (appui sur le bouton generate).
Indication supplémentaire : on utilisera comme dans l'exemple convolve.prj (et le TP précédant) un certain nombre de variables globales :

```
static int panelHandle;
```

```
static int nbPts = 256;
static double sig1cycles = 5.0;
static double sig1phase = 0.0;
static double sig1noiseamp = 0.1;
static int sig1wavetype = 0;
static int sig1noise = 0;
static double wave1[2048];
static double noisewave1[2048];
//static int mainpnl;
static int seed = 1;

struct infoSinus {
  double amplitude,phase,offset;
}infoSin;

void GenerateGraphs(void);
struct infoSinus calculModPhas(void);
```

Informations pour plus tard :
Recherche de racine :
AdvancedAnalysis -> class Additional Numerical Method ->Complex Polynomial roots.

TP6 : Projet final et évaluation (contrôle continu)

On cherche à faire le travail du TP 5 mais en utilisant un oscilloscope TDS220 comme source de tensions. On communiquera avec cet oscilloscope à l'aide d'une liaison série. L'objectif sera toujours le calcul de la phase. On lui ajoutera l'évaluation du gain.

I) Communication série avec LabWindows

LabWindows propose des primitives simplifiant la programmation de la liaison série. On utilisera par exemple :

```
#define COM 1
...
  int longueur;
  char message[100];
  unsigned char reponse[3000];
  OpenComConfig(COM,"",9600,0,8,1,512,512);
  longueur = sprintf(message,"*IDN?\n");
  ComWrt(COM,message,longueur);
  memset(reponse,0,3000);
  ComRdTerm(COM,reponse,70,13);
```

qui envoie un message à l'oscilloscope lui demandant de s'identifier. L'ouverture de la liaison série à l'aide de

```
OpenComConfig(COM,"",9600,0,8,1,512,512);
```

ne se fera qu'une seule fois (sera donc dans le main() avant l'initialisation de l'interface).

Une autre primitive de lecture sera utilisée : `ComRd(COM,reponse,3000);`

II)Quelques sous-programmes donnés

On vous donne un certain nombre de sous programmes qu'il vous faudra utiliser pour mener à bien votre projet :

Un sous-programme capable de lire une voie de l'oscilloscope :

```
void litVoie(char voie, unsigned char reponse[3000]){
// selection de la voie 1
  int longueur,i;
  char message[100];
  if ((voie==1) || (voie==2)) {
    longueur = sprintf(message,"DAT:SOU CH%d\n",voie);
    ComWrt(COM,message,longueur);
// selection du format des données
    longueur = sprintf(message,"DAT:ENC SRP\n");
    ComWrt(COM,message,longueur);
// selection du format des données 1 octet
    longueur = sprintf(message,"DAT:WID 1\n");
    ComWrt(COM,message,longueur);
// début du transfert 1
    longueur = sprintf(message,"DAT:STAR 1\n");
```

```
    ComWrt(COM,message,longueur);
// début du transfert 2500
    longueur = sprintf(message,"DAT:STOP 2500\n");
    ComWrt(COM,message,longueur);
// transfert des données
    longueur = sprintf(message,"CURV?\n");
      ComWrt(COM,message,longueur);
    memset(reponse,0,3000);
    ComRd(COM,reponse,3000);
// suppression de l'entete
    for(i=0;i<2500;i++)
      reponse[i]=reponse[i+7];
  } else MessagePopup("ERREUR","numéro de voie incorrect !!!");
}
```

qui comme le montre le code ne fonctionne que pour voie 1 ou voie 2. Il y a deux voies sur l'oscilloscope.

Cet oscilloscope est capable de mesurer la fréquence du signal. On donne un sous-programme capable de récupérer cette information :

```
// Mesure de la frequence par oscillo
float mesureFreq(char voie){
  int longueur;
  char message[50],reponse[50];
  float freq;
  if ((voie==1) || (voie==2)) {
    longueur = sprintf(message,"MEASU:MEAS1:TYP FREQ\n");//MEAS1 to MEAS4
    ComWrt(COM,message,longueur);
    longueur = sprintf(message,"MEASU:MEAS1:SOU CH%d\n",voie);
    ComWrt(COM,message,longueur);
    longueur = sprintf(message,"MEASU:MEAS1:VAL?\n");
    ComWrt(COM,message,longueur);
    memset(reponse,0,50);
    ComRdTerm(COM,reponse,50,13);
    //sscanf(reponse,"%e",&freq);
    freq = atof(reponse);
    return freq;
  } else {
    MessagePopup("ERREUR","numéro de voie incorrect !!!");
    return -1;
  }
}
```

Mesure de la période d'échantillonnage :

```
// mesure de la période d'échantillonnage
float mesureTEchant(){
// Mesure en s/division (10 divisions en 2500 points
  int longueur;
  char message[50],reponse[50];
  float periode;
  longueur = sprintf(message,"HOR:MAI:SCA?\n");
  ComWrt(COM,message,longueur);
  memset(reponse,0,50);
```

```
  ComRdTerm(COM,reponse,50,13);
  periode = atof(reponse)/250;
  return periode;
//  MessagePopup("Echantillonnage",reponse);
}
```

Si un offset est présent sur une voie, il est possible de le récupérer en le demandant à l'oscilloscope :

```
// mesure de l'offset (en carreaux verticaux)
float mesureOffset(char voie){
  int longueur;
  char message[50],reponse[50];
  float offset;
  if ((voie==1) || (voie==2)) {
    longueur = sprintf(message,"CH%d:POS?\n",voie);
    ComWrt(COM,message,longueur);
    memset(reponse,0,50);
    ComRdTerm(COM,reponse,50,13);
    //sscanf(reponse,"%e",&freq);
    offset = atof(reponse);
    return offset;
  } else {
    MessagePopup("ERREUR","numéro de voie incorrect !!!");
    return -1;
  }
}
```

Le calibre vertical est aussi une information importante pour le gain. Voici comment on peut le récupérer :

```
// mesure du calibre en V/carreau
float mesureCalibre(char voie){
  int longueur;
  char message[50],reponse[50];
  float voltParDiv;
  if ((voie==1) || (voie==2)) {
    longueur = sprintf(message,"CH%d:SCA?\n",voie);
    ComWrt(COM,message,longueur);
    memset(reponse,0,50);
    ComRdTerm(COM,reponse,50,13);
    //sscanf(reponse,"%e",&freq);
    voltParDiv = atof(reponse);
    return voltParDiv;
  } else {
    MessagePopup("ERREUR","numéro de voie incorrect !!!");
    return -1;
  }
}
```

Et enfin voici comment on peut récupérer le module et la phase d'une voie (méthode des moindre carré que vous deviez utiliser dans le TP 5.

```
//struct infoSinus {
//  double amplitude,phase,offset;
```

```
//};

struct infoSinus calculModPhas(char voie){
  int i,nbelt;
  double sin[3000],cos[3000],y[3000],H[3000]
[3],b[3],Techant,Freq,offset,calibre;
  unsigned char reponse[3000];
  struct infoSinus temp;
  if ((voie==1) || (voie==2)) {
    Techant=mesureTEchant();
    Freq=mesureFreq(voie);
    nbelt=1/(Techant*Freq);
    litVoie(voie,reponse);
// generation d'une sinusoïde
    SinePattern (nbelt, 1.0, 0.0, 1.0, sin);
// génération d'un cosinus
    SinePattern (nbelt, 1.0, 90.0, 1.0, cos);
// construction de H
    for(i=0;i<nbelt;i++){
      H[i][0]=1;
      H[i][1]=sin[i];
      H[i][2]=cos[i];
    }
// construction de y  en retirant l'offset et en calibrant en volt
    calibre = mesureCalibre(voie);
//    printf("calibre : %f\n",calibre);
    offset = mesureOffset(voie)*255/10;
//    printf("offset : %f\n",offset);
    for(i=0;i<nbelt;i++){
      y[i]= (((reponse[i]-offset-128)*10*calibre)/255);
    }
// calcul des moindre carrés
    GenLSFitCoef (H, nbelt, 3, y, b, 0);
    temp.phase =  atan2(b[2],b[1])*180/3.14159;
    temp.amplitude = sqrt(b[2]*b[2]+b[1]*b[1]);
    temp.offset = b[0];
    return temp;
  } else {
    MessagePopup("ERREUR","numéro de voie incorrect !!!");
    temp.phase =0;
    temp.amplitude=0;
    temp.offset=0;
    return temp;
  }
}
```

On rappelle que des tests peuvent être réalisés avec des PopUp :

```
char messagePopup[200];
...
result= calculModPhas(2);
  sprintf(messagePopup,"Reponses : module : %f V ; phase %f° ; offset %f
V",result.amplitude,result.phase,result.offset);
  MessagePopup("Résultat final",messagePopup);
```

III)Travail à réaliser

Au minimum on vous demande de réaliser un programme qui présente un bouton de mesure et trois champs numériques qui seront la phase, le gain et la fréquence. Chaque appui sur le bouton provoquera l'acquisition des deux voies de l'oscilloscope, le calcul du gain (rapport des amplitudes des deux voies) et de la phase (différence des deux phases des deux voies).

Vous pourrez agrémenter ce programme avec une face avant ressemblant à celle du TP 5. Vous demanderez donc à l'oscilloscope les valeurs correspondant à chacune des voies et les tracerez donc sur le graphe. Vous devrez gérer pour cela un ou plusieurs gains pour que les courbes soient visibles dans tous les cas, sachant que le graphe va de -10 à +10 (ce qui ne sera pas toujours le cas des échantillons récupérés).

TP2

Correction exo1

```
int CVICALLBACK MaJ (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)

{   int leds[8];
    static unsigned char nb=0;
      switch (event)
            {
            case EVENT_TIMER_TICK:
            // calcul de la nouvelle valeur
            nb++;
            conversion(nb,leds);
            // Mise à jour des leds
             SetCtrlVal(panelHandle,PANEL_LED,leds[0]);
             SetCtrlVal(panelHandle,PANEL_LED_2,leds[1]);
             SetCtrlVal(panelHandle,PANEL_LED_3,leds[2]);
             SetCtrlVal(panelHandle,PANEL_LED_4,leds[3]);
             SetCtrlVal(panelHandle,PANEL_LED_5,leds[4]);
             SetCtrlVal(panelHandle,PANEL_LED_6,leds[5]);
             SetCtrlVal(panelHandle,PANEL_LED_7,leds[6]);
             SetCtrlVal(panelHandle,PANEL_LED_8,leds[7]);
                  break;
            }
      return 0;
}
```

Correction exo1 du tp2 (travail optionnel): On peut faire plus simple en utilisant la même technique que dans l'exo1.

```
int CVICALLBACK MaJ (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)

{   int leds[8];

      switch (event)
            {
            case EVENT_TIMER_TICK:
            // lecture de l'état des leds dans un tableau
             GetCtrlVal(panelHandle,PANEL_LED,&leds[0]);
             GetCtrlVal(panelHandle,PANEL_LED_2,&leds[1]);
             GetCtrlVal(panelHandle,PANEL_LED_3,&leds[2]);
             GetCtrlVal(panelHandle,PANEL_LED_4,&leds[3]);
             GetCtrlVal(panelHandle,PANEL_LED_5,&leds[4]);
             GetCtrlVal(panelHandle,PANEL_LED_6,&leds[5]);
             GetCtrlVal(panelHandle,PANEL_LED_7,&leds[6]);
             GetCtrlVal(panelHandle,PANEL_LED_8,&leds[7]);
            // calcul de la nouvelle valeur
            calcul(leds);
            // Mise à jour des leds
             SetCtrlVal(panelHandle,PANEL_LED,leds[0]);
             SetCtrlVal(panelHandle,PANEL_LED_2,leds[1]);
             SetCtrlVal(panelHandle,PANEL_LED_3,leds[2]);
             SetCtrlVal(panelHandle,PANEL_LED_4,leds[3]);
             SetCtrlVal(panelHandle,PANEL_LED_5,leds[4]);
             SetCtrlVal(panelHandle,PANEL_LED_6,leds[5]);
             SetCtrlVal(panelHandle,PANEL_LED_7,leds[6]);
             SetCtrlVal(panelHandle,PANEL_LED_8,leds[7]);
                  break;
            }
      return 0;
```

```
}

void calcul(int leds[8]){
  unsigned char i,ii,lesleds;
//passage d'un tableau à un octet
  for (i=0;i<8;i++){
    ii = 1<<i;
    if (leds[i]==1)
      lesleds = lesleds | ii;
    else
        lesleds = lesleds & ~ii;
  }
  // décalage
  lesleds =lesleds << 1   ;
  // permet l'initialisation :
  if (lesleds == 0) lesleds=1;
// passage d'un octet à un tableau
  for (i=0;i<8;i++){
    ii = 1<<i;
    if ((lesleds & ii) ==ii)
      leds[i]=1;
    else
        leds[i]=0;
  }

}
```

TP3
Exo2

```
int CVICALLBACK start (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{   // go déclarée en globale : int go=0;
      switch (event)
            {
            case EVENT_COMMIT:
                  go=(go+1)%2;
                  break;
            }
      return 0;
}

// Timer
int CVICALLBACK MaJ (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{   double voie[1];
      switch (event)
            {
            case EVENT_TIMER_TICK:
                  if (go) {
        // mise dela valeur du thermomètre dans une variable :
                    GetCtrlVal(panelHandle,PANEL_NUMERICTHERM,&voie[0]);
        // mise de voie[0] dans le stripchart
                    PlotStripChart(panelHandle,PANEL_STRIPCHART,voie,1,0,0,VAL_DOUBLE);
      }
                  break;
            }
      return 0;
}
```

Exo3 :

```
int CVICALLBACK Generate (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{   double sinus[100]; int i;
      switch (event)
            {
            case EVENT_COMMIT:
                  SinePattern (100, 1.0, 0.0, 1.0, sinus);
                  for(i=0;i<100;i++)

PlotStripChart(panelHandle,PANEL_STRIPCHART,&sinus[i],1,0,0,VAL_DOUBLE);
                  break;
            }
      return 0;
}

int CVICALLBACK Clear (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      switch (event)
            {
            case EVENT_COMMIT:
                  ClearStripChart(panelHandle, PANEL_STRIPCHART);
                  break;
            }
      return 0;
}
```

TP4

Exo1

```
_CVICALLBACK Getsig1points (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      switch (event)
            {
            case EVENT_COMMIT:
                  GetCtrlVal(panelHandle,PANEL_SIG1POINTS,&nbPts);
                  break;
            }
      return 0;
}

int CVICALLBACK Getsig1cycles (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      switch (event)
            {
            case EVENT_COMMIT:
                  GetCtrlVal(panelHandle,PANEL_SIG1CYCLES,&sig1cycles);
                  break;
            }
      return 0;
}

int CVICALLBACK Getsig1phase (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      switch (event)
            {
            case EVENT_COMMIT:
                  GetCtrlVal(panelHandle,PANEL_SIG1PHASE,&sig1phase);
                  break;
```

```
            }
      return 0;
}
int CVICALLBACK Generatesig1 (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      switch (event)
            {
            case EVENT_COMMIT:
                  SinePattern (nbPts, 7.0, sig1phase, sig1cycles, wave1);
                  DeleteGraphPlot (panelHandle, PANEL_SIG1GRAPH, -1,
                                   VAL_IMMEDIATE_DRAW);
                  PlotY (panelHandle, PANEL_SIG1GRAPH, wave1, nbPts, VAL_DOUBLE,
                              VAL_THIN_LINE, VAL_EMPTY_SQUARE, VAL_SOLID, 1, VAL_RED);

                  break;
            }
      return 0;
}
```

Exo2

```
int CVICALLBACK Getsig1noiseamp (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      switch (event)
            {
            case EVENT_COMMIT:
                  GetCtrlVal(panelHandle,PANEL_SIG1NOISEAMP,&sig1noiseamp);
                  break;
            }
      return 0;
}
int CVICALLBACK Getsig1noise (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      switch (event)
            {
            case EVENT_COMMIT:
                  GetCtrlVal(panelHandle,PANEL_SIG1NOISE,&sig1noise);
                  break;
            }
      return 0;
}
int CVICALLBACK Generatesig1 (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{   int i;
      switch (event)
            {
            case EVENT_COMMIT:
                  SinePattern (nbPts, 7.0, sig1phase, sig1cycles, wave1);
                  if (sig1noise) {
                    GaussNoise (nbPts, sig1noiseamp, seed, noisewave1);
                    for (i=0;i<nbPts;i++)
                      wave1[i]=wave1[i]+noisewave1[i];
                  }
                  DeleteGraphPlot (panelHandle, PANEL_SIG1GRAPH, -1,
                              VAL_IMMEDIATE_DRAW);
                  PlotY (panelHandle, PANEL_SIG1GRAPH, wave1, nbPts, VAL_DOUBLE,
                              VAL_THIN_LINE, VAL_EMPTY_SQUARE, VAL_SOLID, 1, VAL_RED);
```

```
                break;
            }
      return 0;
}
```

Exo3

```
int CVICALLBACK Getsig1wavetype (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      switch (event)
            {
            case EVENT_COMMIT:
                  GetCtrlVal(panelHandle,PANEL_SIG1WAVETYPE,&sig1wavetype);
                  break;
            }
      return 0;
}
int CVICALLBACK Generatesig1 (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{   int i;
      switch (event)
            {
            case EVENT_COMMIT:
                switch (sig1wavetype) {
                    case 0:SinePattern (nbPts, 7.0, sig1phase, sig1cycles, wave1);
                           break;
                    case 1:SquareWave (nbPts, 7.0, (sig1cycles/nbPts),
                                       &sig1phase,50.0, wave1);
                           break;
                    case 2:TriangleWave (nbPts, 7.0, (sig1cycles/nbPts),
                                        &sig1phase,wave1);
                           break;
                  }
                  if (sig1noise) {
                    GaussNoise (nbPts, sig1noiseamp, seed, noisewave1);
                    for (i=0;i<nbPts;i++)
                      wave1[i]=wave1[i]+noisewave1[i];
                  }
                DeleteGraphPlot (panelHandle, PANEL_SIG1GRAPH, -1,
                           VAL_IMMEDIATE_DRAW);
                PlotY (panelHandle, PANEL_SIG1GRAPH, wave1, nbPts, VAL_DOUBLE,
                             VAL_THIN_LINE, VAL_EMPTY_SQUARE, VAL_SOLID, 1, VAL_RED);

                  break;
            }
      return 0;
}
```