Sommaire

Objectif

L'objectif du tuto est l'implémentation sur Arduino des trois approximations suivantes de la dérivée première. T indique la période d'échantillonnage inverse de la fréquence d'échantillonnage :

- Approximation avance: f'(n)=[f(n+1) – f(n)]/T
- Approximation arrière: f'(n)=[f(n) – f(n-1)]/T
- Approximation centrée: f'(n)=[f(n+1) + 0 – f(n-1)]/(2T)

Comment implémenter la dérivée ?

Principe

Les deux premières approximations tiennent en considération deux enchantions successifs. En revanche, la dernière approximation nécessite deux échantillons f(n+1) et f(n-1) symétriques par rapport à f(n). Par conséquent, durant l'implémentation on va utiliser un tableau de trois élément pour stocker les échantillons f(n), f(n-1) et f(n+1).

> *Attention, l'ordre des échantillons est primordial durant le calcul de la dérivée !*

Démarche d'implémentation

On déclare un tableau Tab[3] de trois élément ordonnés de la façon suivante [x(n) x(n-1) x(n-2)]. Pour chaque itération de la boucle on va effectuer les opérations suivantes :

1. Acquisition de nouveau échantillon x(n)
2. Décaler les éléments du tableaux Tab:
    1. Tab[2] <= Tab[1]
    2. Tab[1] <= Tab[0]
3. Incrustation du nouvel échantillon dans le tableau
    1. Tab[0] <= x(n)
4. Calcul des approximations de la dérivée première :
    1. x1_n=(Tab[0]- Tab[1])/T
    2. x1_n=(Tab[1]- Tab[2])/T
    3. x1_n=(Tab[0]- Tab[2])/ (2T)

Implémentation des approximations

On dispose de deux fonctions diff1( ) et diff1_3() pour le calcul de la dérivée première. Ci-dessous la déclaration et la définition des deux fonctions.

La fonction diff1( )

```
double diff1(double x_nn, unsigned long T_mills, byte Type)
```

La fonction prend en entrée le nouvel échantillon x(n), la période d'échantillonnage en milliseconde et le type de l'approximation. Puis, elle renvoie la valeur de l'approximation.

- Type=0 : Approximation avance
- Type=1 : Approximation arrière
- Type=2: Approximation centrée
- Type=x : Renvoie x(n)

La fonction renvoie un résultat à la fois, du coup on ne peut pas savoir les trois résultats au même temps. Il faut trois itérations de la boucle (Type=0 , Type=1 et Type=2) pour connaitre les trois résultats, du coup il faut prévoir un décalage entre les approximations. D'où l'utilisation de la deuxième fonction. Ci-dessous la définition de la fonction diff1( ).

```
double diff1(double x_nn, unsigned long T_mills, byte Type)

{

  static double x[3];

  double diff_1=0.0, T=0.0;



  // Mise à jour des échantillons

  x[0]=x[1];

  x[1]=x[2];

  x[2]=x_nn;



  // Calcul de la dérivée du 1ère Ordre
```

```
T=(double)T_mills/1000.0;



switch (Type)

{

  case 0:

    diff_1=(x[1]-x[0])/T;

    break;



  case 1:

    diff_1=(x[2]-x[1])/T;

    break;



  case 2:

    diff_1=(x[2]-x[0])/(2.0*T);

    break;



  default:

    diff_1=x_nn;

}

return diff_1;
```

```
}
```

Le mot static permet de préserver les échantillons du tableau x[ ] entre les appels de la fonction. Sans utilisation du mot static, les échantillons d'avant seront perdus.

La fonction diff1_3( )

```
void diff1_3(double *diff, double x_nn, unsigned long T_mills)
```

diff1_3() est une fonction compacte, rapide qui renvoie les trois approximations au même temps en une seule itération sous forme d'un tableau *diff (pointeur) de trois éléments. Ci-dessous la définition de la fonction.

```
void diff1_3(double *diff, double x_nn, unsigned long T_mills)

{

  double T=0.0;

  static double x[3];



  // Mise à jour des échantillons

  x[0]=x[1];

  x[1]=x[2];

  x[2]=x_nn;



  // Calcul de la dérivée du 1ère Ordre (3 techniques)
```

```
  T=(double)T_mills/1000.0;

  diff[0]=(x[1]-x[0])/T;

  diff[1]=(x[2]-x[1])/T;

  diff[2]=(x[2]-x[0])/(2.0*T);

}
```

Test du programme

Pour tester les approximations on a besoin de générer une fonction f(n), puis on calcul sa dérivée en utilisant les trois approches. L'astuce consiste à générer une fonction triangulaire sous forme d'un compteur qui monte de 0 à une valeur maximale, puis décroit de la même façon à zéro. Du coup on obtient une fonction f(n)=n croissante et décroissante. Le résultat de la dérivée doit être sous forme d'un signal rectangulaire (voir la vidéo pour plus des détails). Ci-dessous un extrait du code dédié à la génération d'un signal triangulaire.

```
…

  // La fonction f(n)=n

  if (sign==true)

  {

    n++;

    if(n==n_max)sign=false;

  }

  else

  {
```

```
    n--;
    if (n==0)sign=true;
  }


  x_n=(double)n;
…
```

Code Arduino

```
/*  www.Electronique-Mixte.fr
 *  Discrétisation de la dérivée 1ère d'une fonction
 *  1. f_n_1(n)=[ f(n+1) - f(n)]/T
 *  2. f_n_1(n)=[ f(n) - f(n-1)]/T
 *  3. f_n_1(n)=[ f(n+1) - f(n-1)]/(2T)
 */



#define   n_max   100


const unsigned long T_ms=10;
```

```
bool sign=true;

byte TypeDiff;

double d1_f_n[3], x_n;

unsigned long n=0;



void setup()

{

  // Port série: Affichage de la vitesse

  Serial.begin(9600);

}

void loop()

{

  // La fonction f(n)=n

  if (sign==true)

  {

    n++;

    if(n==n_max)sign=false;

  }

  else

  {

    n--;

    if (n==0)sign=true;
```

```
}



x_n=(double)n;



// Calcul de la dérivée 1ère (1 dérivée/cycle)

/*d1_f_n[TypeDiff]= diff1(x_n, T_ms, TypeDiff);

TypeDiff++; // Mise à jour de l'indice

TypeDiff%=3;*/



// Calcul de la dérivée 1ère (3 dérivées/cycle)

diff1_3(d1_f_n, x_n, T_ms);



// Affichage de la vitesse & la distance en mm

Serial.print(d1_f_n[0]); Serial.print(",");

Serial.print(d1_f_n[1]); Serial.print(",");

Serial.print(d1_f_n[2]); Serial.print(",");

Serial.println(x_n);



// Période d'échantillonnage

delay(T_ms);
```

```
}









double diff1(double x_nn, unsigned long T_mills, byte Type)

{

  static double x[3];

  double diff_1=0.0, T=0.0;



  // Mise à jour des échantillons

  x[0]=x[1];

  x[1]=x[2];

  x[2]=x_nn;



  // Calcul de la dérivée du 1ère Ordre



  T=(double)T_mills/1000.0;



  switch (Type)
```

```
  {
    case 0:
      diff_1=(x[1]-x[0])/T;
      break;



    case 1:
      diff_1=(x[2]-x[1])/T;
      break;



    case 2:
      diff_1=(x[2]-x[0])/(2.0*T);
      break;



    default:
      diff_1=x_nn;
  }
  return diff_1;
}
```

```
void diff1_3(double *diff, double x_nn, unsigned long T_mills)

{

  double T=0.0;

  static double x[3];



  // Mise à jour des échantillons

  x[0]=x[1];

  x[1]=x[2];

  x[2]=x_nn;



  // Calcul de la dérivée du 1ère Ordre (3 techniques)

  T=(double)T_mills/1000.0;

  diff[0]=(x[1]-x[0])/T;

  diff[1]=(x[2]-x[1])/T;

  diff[2]=(x[2]-x[0])/(2.0*T);

}
```

Accueil Asservissement avec Arduino