TP6 MATLAB

CALCUL SCIENTIFIQUE 1

ALGORITHME HORNER

L'algorithme de Horner vise à réduire la charge de calcul (voir illustration ci-dessous) pour déterminer la valeur d'un polynôme en une valeur particulière x_o : P(x_o).

Le principe est le suivant :

- Nous partons du coefficient de plus haut degré : a_n
- Puis on multiplie par x_o : $a_n \rightarrow a_n x_o$ puis on additionne $a_{n-1} \rightarrow a_n x_o + a_{n-1}$
- Nous réitérons les étapes précédentes comme le montre les figures ci-dessous

Méthode de Ruffini-Horner

Elle permet de calculer la valeur d'un polynôme en $x_0 \Rightarrow P(x_0) = a_n x_0^n + a_{n-1} x_0^{n-1} + \ldots + a_0$

Charge de calcul (sans Horner) = n+(n-1)+(n-2)+...1 => pas optimisée

n produits (n-1) produits ...

La méthode consiste donc à multiplier le premier coefficient par x_0 et à lui ajouter le deuxième coefficient. On multiplie alors le nombre obtenu par x_0 et on lui ajoute le troisième coefficient, etc.

Coefficients de P	a_n	a_{n-1}	a_{n-2}	...	a_1	a_0
Facteur x_0	a_n	$a_n x_0 + a_{n-1}$	$(a_n x_0 + a_{n-1})x_0 + a_{n-2}$	...	q_0	$P(x_0) = q_0 x_0 + a_0$

Exemple pratique : Calcul de $4X^3 - 7X^2 + 3X - 5$ pour X = 2

Coefficients de P	4	−7	3	−5
Facteur 2	4	8 − 7 = 1	2 + 3 = 5	$P(2)$ = 10 − 5 = 5

=> Avec Horner, on réduit la charge de calcul

Figure 1 : Principe de l'algorithme de Horner

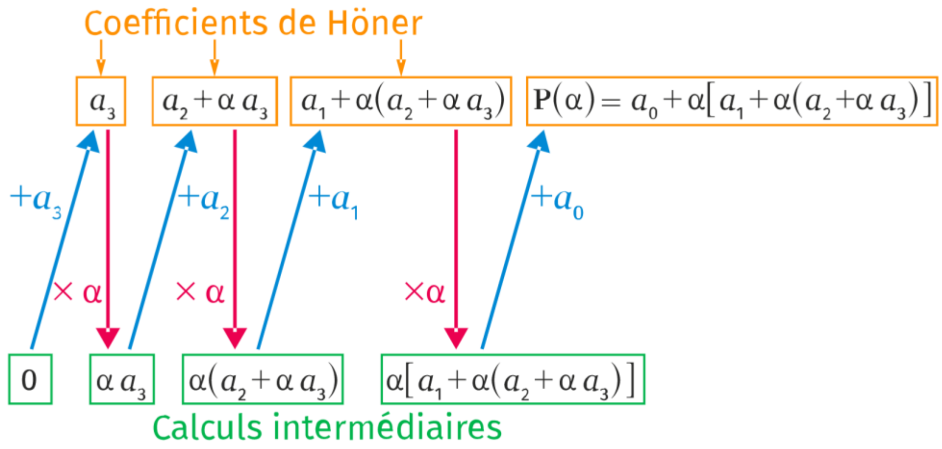


Figure 2 : Exemple avec 3 itérations

<u>**Exercice 1**</u> : Réaliser une fonction res = Horner() reposant sur l'algorithme de Horner, qui :

1. Demande à l'utilisateur le degré n du polynôme
2. Demande à l'utilisateur les coefficients $a_n, a_{n-1}, \ldots, a_o$
3. Demande à l'utilisateur la valeur de x_o et renvoie la valeur $P(x_o)$ dans res

CALCUL SUR LES POLYNOMES

Racine d'un polynôme

1. Prenons l'exemple : $3x^2 - 5x + 2 = 0$

2. Créons un vecteur p avec les coefficients du polynôme

3. La commande roots permet d'obtenir les racines du polynôme :

   ```
   >> roots(p)
   ```

<u>Remarque</u> : la commande roots traite aussi bien le corps des réels que le corps des complexes

<u>**Exercice 2**</u> : Réaliser une fonction res = racinespolynome() qui :

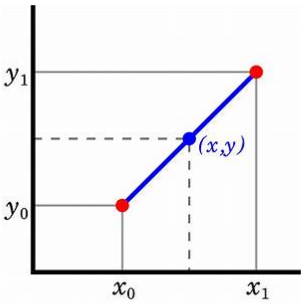
1. Demande à l'utilisateur le degré n du polynôme
2. Demande à l'utilisateur les coefficients $a_n, a_{n-1}, \ldots, a_o$
3. Renvoie les racines du polynôme dans res

NOTION D'INTERPOLATION

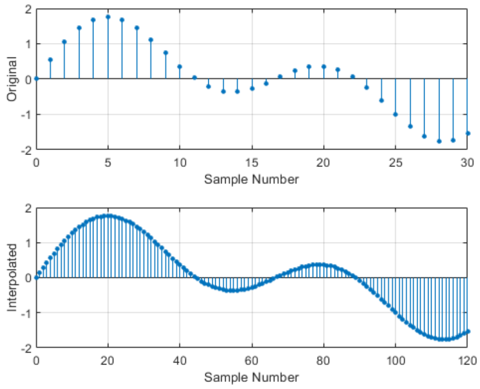
L'interpolation consiste à créer une courbe entre 2 échantillons d'un signal pour y insérer un ou plusieurs échantillons (non issus du signal d'origine). L'interpolation peut être utilisée pour, par exemple, sur-échantillonner un signal qui peut permettre :

- d'augmenter le rapport signal sur Bruit
- d'éviter de réaliser un filtrage passe bas

Interpolation : opération consistant à concevoir une courbe entre 2 points (courbe linéaire, courbe polynômiale, ...).
=> Cette méthode permet d'évaluer la courbe en des points n'appartenant pas au signal d'origine : on crée alors de nouveaux échantillons associés au signal



Cas d'une interpolation linéaire entre 2 points



Utilisation de la méthode d'interpolation pour suréchantillonner un signal

Si nous souhaitons interpoler 2 échantillons entre 2 échantillons du signal, il faut appliquer un facteur d'interpolation = 3. En effet :

- pour un échantillon signal, 3 échantillons sont générés après interpolation d'un facteur 3

La commande Matlab est la suivante :

- factsample = 3; % on veut interpoler des points entre 2 points du signal d'origine
- y2 = interp(y,factsample); % création du signal sur-échantillonné y2, le signal y étant le signal d'origine avant interpolation

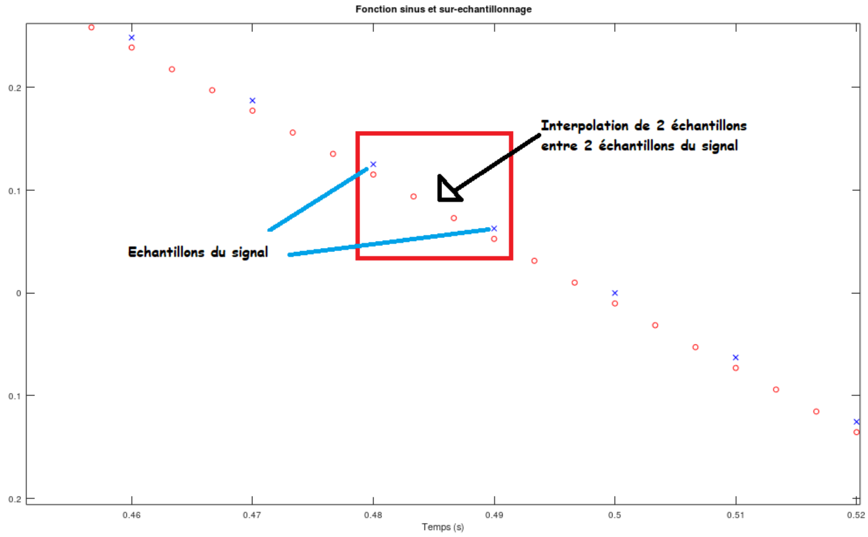


Figure 3 : interpolation d'un facteur 3 : 2 échantillons sont crées entre 2 échantillons du signal d'origine

<u>Exercice 3</u> : Réaliser un script qui respecte les étapes suivantes :

1. lancer le package signal contenant la fonction d'interpolation :
   - >> pkg load signal
2. Génération d'une sinusoide de 1 Hz et de fréquence d'échantillonnage Fe = 100Hz avec une durée du signal de 1s.
3. Réaliser un signal sur-échantillonné d'un facteur 3
4. Mettre à jour le vecteur temps (qui sera de fait également sur-échantillonné d'un facteur 3) pour ce signal sur-échantillonné
5. Superposer les deux signaux (origine et interpolé) sur une même courbe
6. Zoomer sur une portion du signal pour constater que 2 points ont bien été interpolés entre 2 échantillons du signal d'origine

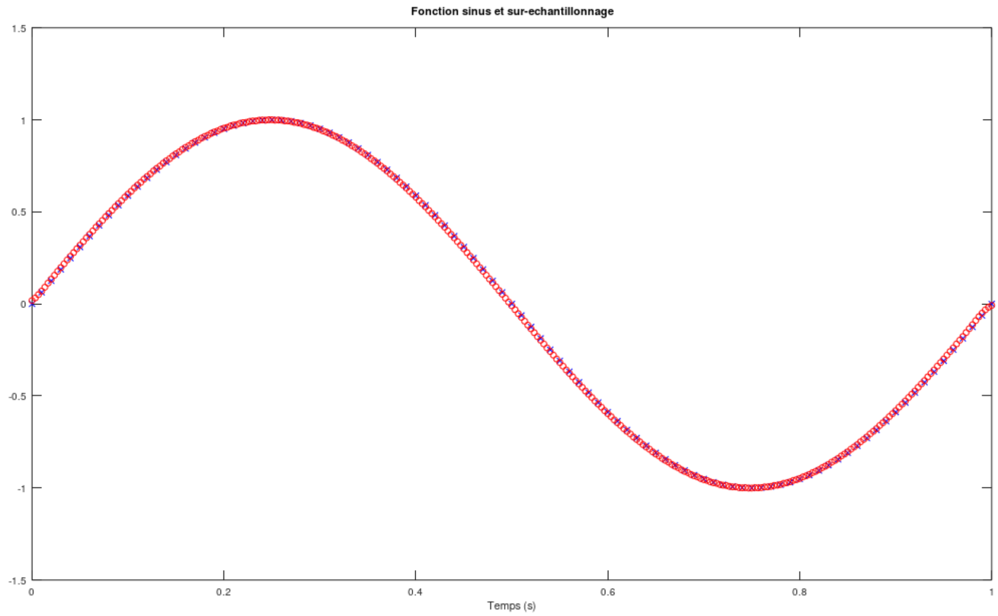


Figure 4 : Superposition d'un signal sinusoïdale et sa version sur-échantillonnée

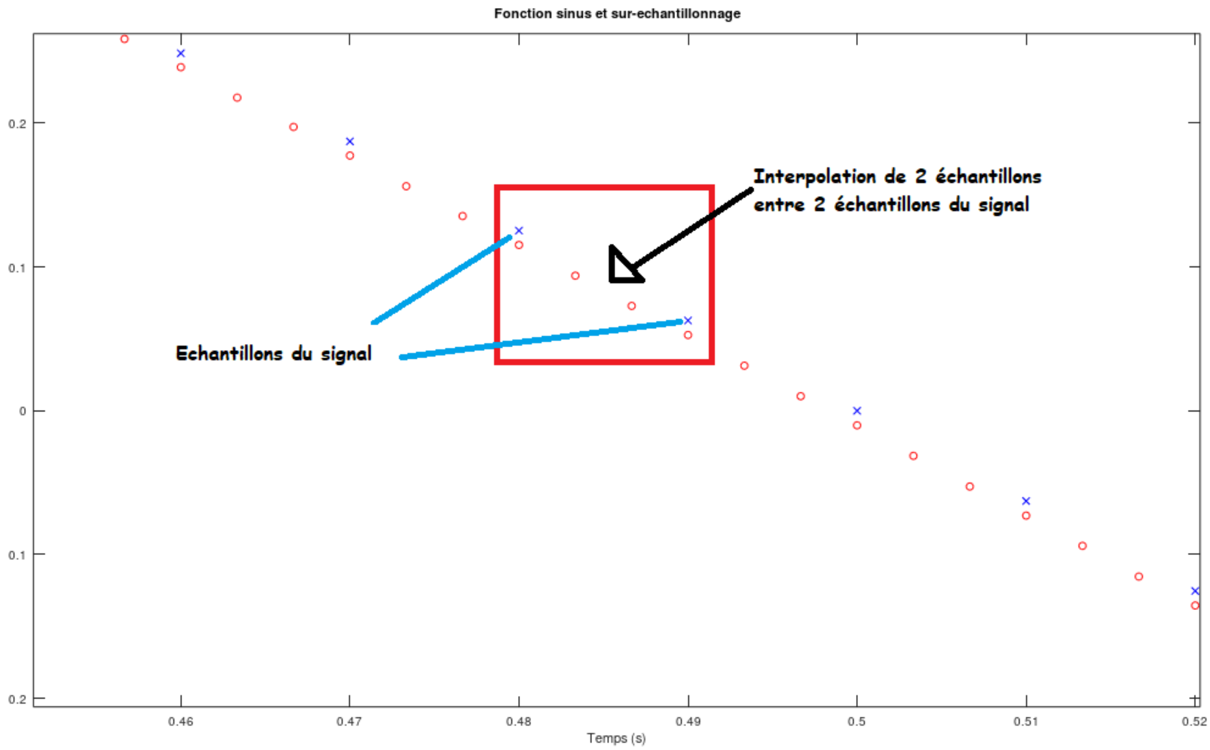


Figure 5 : Superposition d'un signal sinusoïdale et sa version sur-échantillonnée (zoom)

INTEGRATION NUMERIQUE ET DERIVEES SUCCESSIVES

Dérivée et Dérivées successives

La dérivée d'une fonction f en x est usuellement notée $f'(x)$ ou $\frac{\mathrm{d}f}{\mathrm{d}x}(x)$.

On appelle le taux d'accroissement de de f en xo, avec un pas h, la quantité :

$$t_{x_0}(h) = \frac{f(x_0+h)-f(x_0)}{h}$$

Si le taux d'accroissement admet une limite en 0, alors la dérivée f'(xo) existe et est définie par :

$$f'(x_0) = \lim_{h\to 0} t_{x_0}(h) = \lim_{h\to 0} \frac{f(x_0+h)-f(x_0)}{h}$$

Ou de manière équivalente :

$$f'(x_0) = \lim_{x \to x_0} \frac{f(x) - f(x_0)}{x - x_0}$$

Exercice 4 :

On vous donne le début d'un programme matlab générant la fonction sinus en fonction de la variable X

```
h = 0.001;       % step size
X = -pi:h:pi;    % domain
f = sin(X);      % range
```

Q1. En vous appuyant de la fonction **diff** de matlab (vous aider de l'aide et documentation Matlab si nécessaire), générer la fonction dérivée première de la fonction sin(X) et visualiser le résultat en superposant les 2 courbes.

Q2. En vous appuyant à nouveau de la fonction **diff** de matlab, générer la fonction dérivée seconde de la fonction sin(X) et visualiser le résultat en superposant la courbe à celle de sin(X).

Q3. Calculer et afficher la courbe représentative de la fonction dérivée de la fonction logarithme népérien ln(x) [identifiée par log(x) en matlab]. Comparer cette courbe avec celle de la fonction f(x) = 1/x. On prendra la variable x comprise entre 1 et 10 par pas de 0.001.

Intégration numérique

Soit $f : [a, b] \to \mathbb{R}$ une fonction continue. On rappelle les formules approchées pour l'intégrale $\int_a^b f(x)dx$. Pour cela, on choisit d'abord une subdivision

$$a = a_0 < a_1 < \cdots < a_n = b.$$

de l'intervalle $[a, b]$. La formule de Chasles donne

$$\int_a^b f(x)dx = \sum_{i=0}^{n-1} \int_{a_i}^{a_{i+1}} f(x)dx.$$

On est donc ramené au problème d'évaluer l'intégrale de f sur un petit intervalle $[a_i, a_{i+1}]$. Ce calcul est effectué au moyen de formules approchées (qui peuvent être a priori différentes sur chacun des intervalles $[a_i, a_{i+1}]$), appelées méthodes de quadrature.

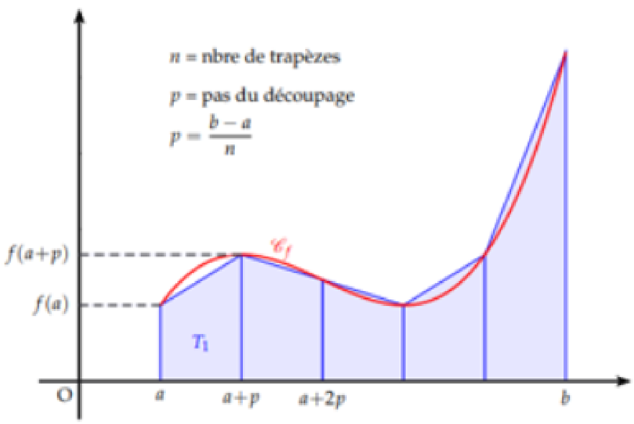
Une des méthodes très connues et implémentée dans Matlab est la méthode dite « des trapèzes » qui est présentée ci-dessous :

c'est-à-dire que chaque portion d'intégrale (si on adopte la notation précédente) :
$\int_{a_i}^{a_{i+1}} f(x)dx \approx [f(a_i) + f(a_{i+1})] * (a_{i+1} - a_i)/2$

(formule d'aire d'un trapèze : [Grande base + Petite base]*hauteur/2)

Méthode des trapèzes

On peut améliorer l'approximation en remplaçant les rectangles par des trapèzes comme le montre la figure ci-dessous :



Pour calculer l'aire du premier trapèze :

$$T_1 = \frac{(\text{Grande base} + \text{Petite base}) \times \text{hauteur}}{2} = \frac{[f(a) + f(a+p)] \times p}{2}$$

On fait ensuite un décalage de p pour calculer les aires des trapèzes suivants.

L'approximation de l'intégrale est alors :

$$\int_a^b f(x)\,\mathrm{d}x \approx \sum_{i=1}^{n} T_i \text{ somme des aires des trapèzes}$$

Q1. En vous appuyant de la fonction **cumtrapz(x,f)** de matlab (la fonction **trapz** donne le résultat de l'intégrale $\int_a^b f(x)dx$), générer la fonction primitive F(x) de la fonction f(x) = cos(x) : vérifier que la fonction obtenue est sin(x). On prendra la variable x comprise entre -pi et pi par pas de 0.001.

Q2. Vérifier que la fonction primitive de la fonction f(x) = 1/x est F(x) = log(x) On prendra la variable x comprise entre 1 et 10 par pas de 0.001.