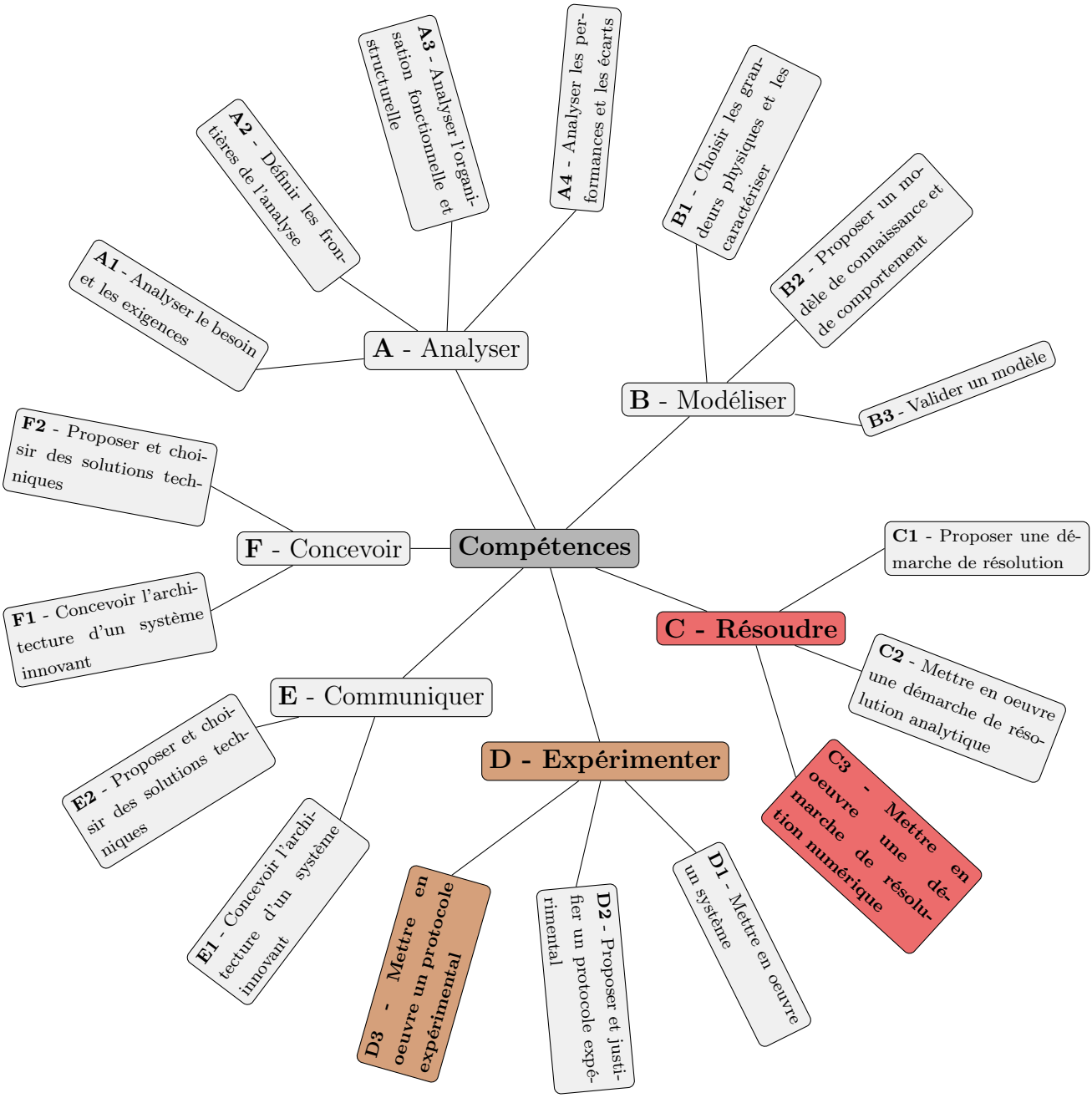
Ingénierie numérique



La marguerite des compétences

David Fournier - 30 novembre 2022

A partir du cours de Renaud Pfeiffer

Inventaire des compétences, des sous compétences, des connaissances et des savoir-faire

Points spécifiques du programme [a] abordés dans ce cours.

D3-3-1

Expérimenter : Mettre en oeuvre un protocole expérimental

Traitement de fichiers de données.
Moyenne et écart type.
Moyenne glissante et filtres numériques passe-bas du premier et du second ordre.

Effectuer des traitements à partir de données. ⇆ ITC

C3-2-1

Résoudre : Mettre en oeuvre une démarche de résolution numérique

Réécriture des équations d'un problème.
Résolution de problèmes du type $f(x) = 0$ (méthodes de dichotomie et de Newton).
Résolution d'un système linéaire du type $A \cdot X = B$.
Résolution d'équations différentielles (schéma d'Euler explicite).
Intégration et dérivation numérique (schémas arrière et avant).

Résoudre numériquement une équation ou un système d'équations. ⇆ ITC

a. Programme de la filière scientifique de la voie PCSI - PSI dans la discipline Sciences Industrielles de l'Ingénieur

Table des matières

1 Ingénierie numérique

L'ingénierie numérique est un terme qui peut désigner :

— L'ingénierie assistée par ordinateur, c'est-à-dire les outils informatiques ou numériques appliqués aux activités d'ingénierie traditionnelle ;

— L'ingénierie du numérique, c'est-à-dire la famille des disciplines d'ingénierie dans le domaine des technologies du numérique, en particulier le génie logiciel, l'ingénierie des systèmes ou des systèmes complexes, l'intégration des systèmes, et la cybersécurité ;

— L'ingénierie des produits et services qui intègrent une forte composante numérique, comme le multimédia, la robotique, les équipements numériques médicaux, l'énergie et le transport.

2 Traitement de fichiers de données

Pour valider un modèle, il faut comparer les résultats issues du modèle à ceux issues de l'expérience.

Il n'est pas rare que les résultats issues de l'expérience soient exportés dans un fichier CSV que l'on doit, par la suite, traiter.

Pour information : Comma-separated values, connu sous le sigle CSV, est un format texte ouvert représentant des données tabulaires sous forme de valeurs séparées par des virgules (point-virgule en France). Un fichier CSV est un fichier texte. Chaque ligne du texte correspond à une ligne du tableau et les virgules correspondent aux séparations entre les colonnes. Les portions de texte séparées par une virgule correspondent ainsi aux contenus des cellules du tableau. Une ligne est une suite ordonnée de caractères terminée par un caractère de fin de ligne.

2.1 Lecture d'un fichier

Une difficulté majeure peut apparaître si le fichier utilise comme séparateur décimal la virgule au lieu du point. Le séparateur décimal pour python étant le point, le conversion d'une chaine de caractère vers le flottant ne sera pas possible. Les programmes proposés seront vigilant sur ce point particulier qui arrive, en fait, assez souvent.

2.1.1 Python

Nous avons la possibilité d'écrire notre propre code pour importer les données issues du fichier vers l'environnement python.

```python
from os import path

def ouvrirFichier(nomFichier, sep=';', deb=0, fin=0, ):
    """
    Fonction permettant d'ouvrir un fichier de données au format texte

    Entrées
        * nomFichier : str - Adresse du fichier à traiter
        * sep        : str - Séparateur de colonne
        * deb        : int - Nombre de ligne à sauter au début
        * fin        : int - Nombre de ligne à sauter à la fin
    Sortie
        * une liste de listes par variable
    """
    # Précaution
    assert path.isfile(nomFichier),  'Le fichier n\'existe pas'
    assert len(sep) > 0, 'Séparateur non déclaré'

    # Ouverture du fichier
    fichier = open(nomFichier, 'r', encoding='utf-8')
    # Lecture du fichier et mise sous forme de liste
    lignes = fichier.readlines()
    # Fermeture du fichier
    fichier.close()

    # Précaution
    assert 0 <= deb < len(lignes)-fin-1 < len(lignes), 'Borne mal déclarée'

    # Créer la liste de sortie
    result = []
    # lire les lignes une à une sauf les "deb" du début et les "fin" de la fin
    for ligne in lignes[deb:-fin-1]:
        # Si besoin, mettre le bon séparateur décimal
        if sep != ',': ligne = ligne.replace(',','.')
        # Enlever le retour chariot en fin de ligne
        donnees = ligne.rstrip('\n\r')
        # Séparer des données
        donnees = ligne.split(sep)
        # Transformer les chaines de caractères en flottant
        donnees = list(map(float, donnees))
        # Ajouter les données
        result.append(donnees)

    # Transposer result pour avoir la liste des données d'une variable
    result = [[row[i] for row in result] for i in range(len(result[0]))]

    print('Le fichier contient',len(result),'variables et',len(result[0]),'acquisitions.')

    return result
```

2.1.2 Librairie

Nous avons la possibilité d'utiliser la fonction `genfromtxt` de la librairie `numpy` pour importer les données issues du fichier vers l'environnement python. Un extrait de sa documentation est donné ci-dessous :

```
genfromtxt(fname, dtype=<class 'float'>, delimiter=None, skip_header=0, skip_footer=0)
    Load data from a text file, with missing values handled as specified.

    Each line past the first `skip_header` lines is split at the `delimiter`
    character, and characters following the `comments` character are discarded.

    Parameters
    ----------
    fname : file, str, pathlib.Path, list of str, generator
        File, filename, list, or generator to read.  If the filename
        extension is ``.gz`` or ``.bz2``, the file is first decompressed. Note
        that generators must return bytes or strings. The strings
        in a list or produced by a generator are treated as lines.
    dtype : dtype, optional
        Data type of the resulting array.
        If None, the dtypes will be determined by the contents of each
        column, individually.
    delimiter : str, int, or sequence, optional
        The string used to separate values.  By default, any consecutive
        whitespaces act as delimiter.  An integer or sequence of integers
        can also be provided as width(s) of each field.
    skip_header : int, optional
        The number of lines to skip at the beginning of the file.
    skip_footer : int, optional
        The number of lines to skip at the end of the file.
    filling_values : variable, optional
        The set of values to be used as default when the data are missing.

    Returns
    -------
    out : ndarray
        Data read from the text file. If `usemask` is True, this is a
        masked array.
```

Nous pourrions également la fonction `loadtxt` mais la fonction `genfromtxt` a l'avantage de pouvoir gérer les données manquantes avec l'argument `filling_values`.

Exemple

Ce code si le séparateur décimal est une virgule.

```python
from numpy import genfromtxt, char, array

# lire le fichier texte
donnees = genfromtxt('bercebebe.txt', dtype=str, delimiter='\t', skip_header=22, unpack=True)
# remplacer la virgule par un point
donnees = char.replace(donnees,',','.')
# convertir la chaine de caractère en flottant
acq, t, iV, uV, xV, wV, fdcV, iH, uH, xH, wH, fdcH = array(donnees, dtype=float)
```

Ce code peut être simplifié si le séparateur décimal est un point.

```python
from numpy import genfromtxt

# lire le fichier texte
acq, t, iV, uV, xV, wV, fdcV, iH, uH, xH, wH, fdcH = genfromtxt('bercebebe.txt', dtype=float,
    ↳ delimiter='\t', skip_header=22, unpack=True)
```

2.2 Écrire d'un fichier

C'est hors programme mais si besoin je vous conseille de regarder la fonction `savetxt` de la librairie `numpy`.

3 Affichage des données

Nous avons la possibilité d'utiliser la librairie `matplotlib.pyplot` souvent avec l'alias `plt`.

Il existe deux méthodes :

— la méthode orienté objet

— la méthode plus basique avec les fonctions

Il ne faut pas mélanger les deux méthodes sous peine de commettre des erreurs.

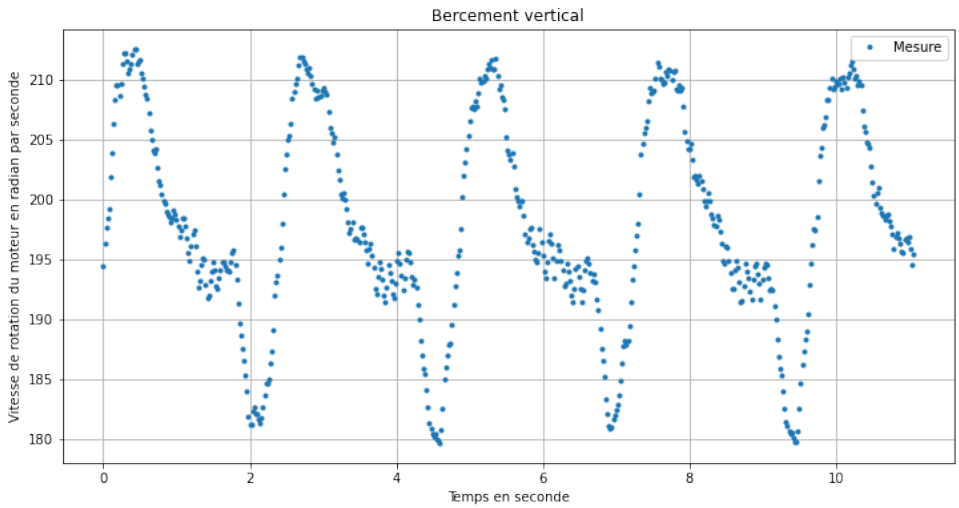
3.1 Méthode avec les fonctions

Le code ci-dessous peut être épuré mais il se veut ici exhaustif.

```python
import matplotlib.pyplot as plt

# FIGURE
# Création
plt.figure(figsize=(12,6))

# GRAPHIQUE
# Création
plt.subplot(1,1,1) # A modifier pour avoir plusieurs graphiques
# Courbe
plt.plot(t, wV, '.', label="Mesure")
# Titre
plt.title('Bercement vertical')
# Legende
plt.legend(loc="best")
# Nom des axe
plt.xlabel('Temps en seconde')
plt.ylabel('Vitesse de rotation du moteur en radian par seconde')
# Afficher une grille en arrière plan
plt.grid()

# Sauvegarder la figure
plt.savefig('figure.png')
# Afficher la figure
plt.show()
```

Si vous étiez amené à avoir plusieurs graphiques dans une figure, je vous invite à regarder l'aide sur internet de `subplot`.

Exemple



3.2 Méthode orientée objet

Je vous la déconseille car cette méthode est plus compliquée que la précédente qui vous suffira pour vos besoins.

4 Intégration et dérivation numérique

Nous avons souvent besoin d'intégrer ou de dériver pour avoir accès à une grandeur physique à partir d'une autre.

Exemple

Notre exemple manipule une vitesse de rotation.

— Pour avoir la position angulaire, il sera nécessaire d'intégrer.

— Pour avoir l'accélération angulaire, il sera nécessaire de dériver.

4.1 Intégration numérique

4.1.1 Intégration d'une fonction continue

L'intégration d'une fonction continue sur un intervalle consiste à déterminer l'aire sous la courbe sur cet intervalle.

Les méthodes des rectangles présentées ici fonctionnent sur le même principe. Il s'agit de subdiviser l'intervalle de calcul, puis d'approximer la courbe par des fonction polynomiales sur ces intervalles.

Dans ces méthodes, chaque subdivision $[x_i;x_{i+1}]$ de la fonction f à intégrer est interpolée par un polynôme de degré 0, à savoir une fonction constante. Géométriquement, l'aire sous la courbe dans la subdivision est alors approximée par un rectangle.

L'aire sous la courbe est alors la somme de l'aire des différents rectangles.

Remarque : En règle général, la largeur de la subdivision $[x_i;x_{i+1}]$ est constante.

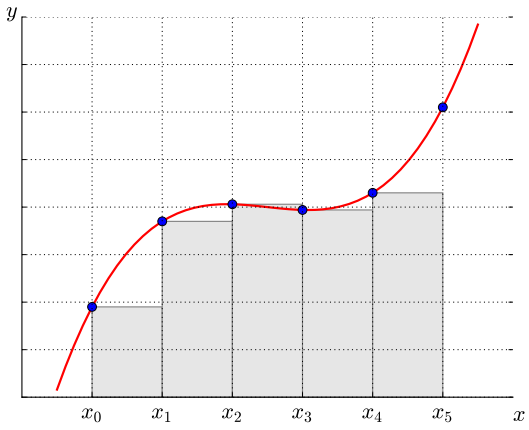
4.1.1.1 Méthode des rectangles à gauche

Une solution possible consiste utiliser la valeur de $f(x_i)$ pour déterminer la hauteur du rectangle. On appelle cette méthode la méthode des rectangles à gauche.

Ainsi : $I_i = \int_{x_i}^{x_{i+1}} f(x)\cdot \mathrm{d}t \approx f(x_i)\cdot(x_{i+1}-x_i)$

Une approximation de l'intégrale sur n subdivision s'exprime donc : $I = \sum_{i=0}^{n-1} I_i$

Remarque : On parle de *schéma numérique arrière* pour la méthode des rectangles à droite, car elle utilise la valeur en i.



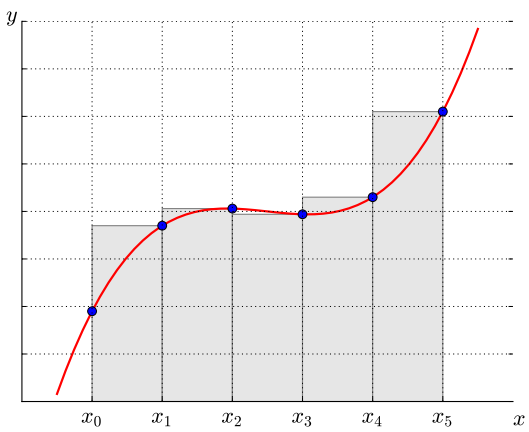
4.1.1.2 Méthode des rectangles à droite

Une solution possible consiste utiliser la valeur de $f(x_{i+1})$ pour déterminer la hauteur du rectangle. On appelle cette méthode la méthode des rectangles à droite.

Ainsi : $I_i = \int_{x_i}^{x_{i+1}} f(x)\cdot \mathrm{d}t \approx f(x_{i+1})\cdot(x_{i+1}-x_i)$

Une approximation de l'intégrale sur n subdivision s'exprime donc : $I = \sum_{i=0}^{n-1} I_i$

Remarque : On parle de *schéma numérique avant* la méthode des rectangles à droite, car elle utilise la valeur en $i+1$.



4.1.2 Intégration d'une fonction échantillonnée

Intégrer une fonction échantillonnée revient à calculer l'aire sous une nuage de points.

```python
def integrationSchemaArriere(y, x):
    # Initialisaion
    iy = []
    # Calcul par le schéma avant
    for i in range(len(y)-1):
        integrale = y[i]*(x[i+1]-x[i])
        iy.append(integrale)
    # Cumul des intégrations
    for i in range(len(iy)-1,-1,-1):
        iy[i] = sum(iy[0:i+1])
    # Retourner le résultat
    return iy
    return iy, x[:-1]
```

```python
def integrationSchemaAvant(y, x):
    # Initialisaion
    iy = []
    # Calcul par le schéma avant
    for i in range(len(y)-1):
        integrale = y[i+1]*(x[i+1]-x[i])
        iy.append(integrale)
    # Cumul des intégrations
    for i in range(len(iy)-1,-1,-1):
        iy[i] = sum(iy[0:i+1])
    # Retourner le résultat
    return iy
    return iy, x[:-1]
```

Cette même fonction avec les fonctions de la librairie `numpy`.

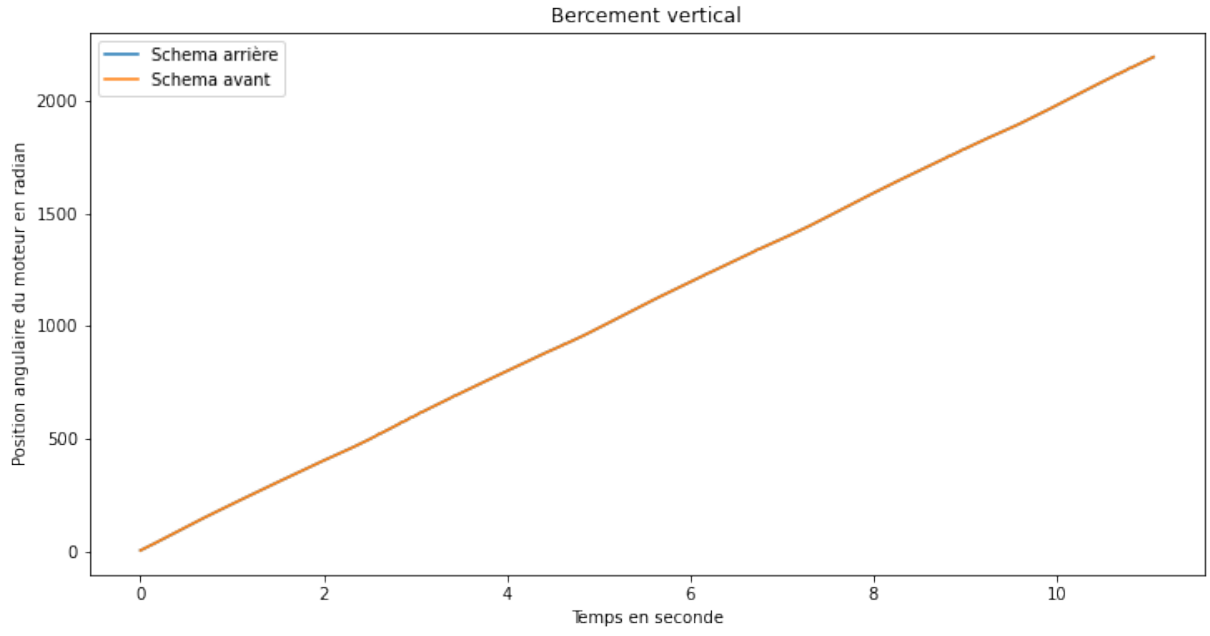
```python
from numpy import array, zeros, cumsum

def integrationSchemaArriere(y, x):
    # Précaution
    y, x = array(y), array(x)
    # Création d'un vecteur rempli de zéros
    iy = zeros(len(y)-1)
    # Calcul par le schéma avant
    iy = y[:-1]*(x[1:] - x[:-1])
    # Somme cumulée
    iy = cumsum(iy)
    # Retourner le résultat
    return iy
    return iy, x[:-1]
```

```python
from numpy import array, zeros, cumsum

def integrationSchemaAvant(y, x):
    # Précaution
    y, x = array(y), array(x)
    # Création d'un vecteur rempli de zéros
    iy = zeros(len(y))
    # Calcul par le schéma avant
    iy = y[1:]*(x[1:] - x[:-1])
    # Somme cumulée
    iy = cumsum(iy)
    # Retourner le résultat
    return iy
    return iy, x[:-1]
```

Remarque : La taille de la liste de l'intégrale a diminué de 1.

Exemple

Dans notre exemple, en intégrant la vitesse de rotation du moteur, on obtient la position angulaire du moteur.



4.2 Dérivation numérique

4.2.1 Dérivation d'une fonction continue

Par définition la dérivée d'une fonction f en un point x_0 est un nombre défini par : $\left[\frac{\mathrm{d}}{\mathrm{d}t}f(x_0)\right] = \lim_{\varepsilon \to 0} \frac{f(x_0+\varepsilon)-f(x_0)}{\varepsilon}$.

En posant $\varepsilon = h$ le pas de dérivation suffisamment petit, on peut réaliser l'approximation numérique suivante appelée *dérivée avant* : $\left[\frac{\mathrm{d}}{\mathrm{d}t}f(x_0)\right] \simeq \frac{f(x_0+h)-f(x_0)}{h}$

De même, on peut réaliser une *dérivée arrière* en utilisant le schéma : $\left[\frac{\mathrm{d}}{\mathrm{d}t}f(x_0)\right] \simeq \frac{f(x_0)-f(x_0-h)}{h}$

4.2.2 Dérivation d'une fonction échantillonnée

Le calcul de la dérivée d'une fonction échantillonnée ou d'un nuage de points est un peu plus complexe.

Soit `x` un vecteur contenant des abscisses d'un point et `y` un vecteur contenant les ordonnées de ces points.

On cherche à calculer un vecteur `dy` qui contient la dérivée de `y` par rapport à `x`.

Le schéma numérique arrière devient : `dy[i] = (y[i] - y[i-1])/(x[i] - x[i-1])`

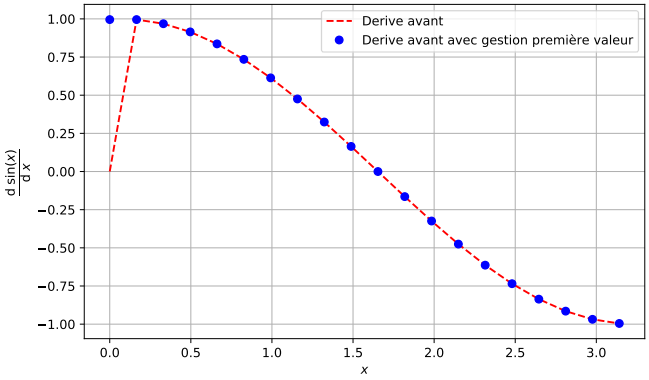
Le schéma numérique avant devient : `dy[i] = (y[i+1] - y[i])/(x[i+1] - x[i])`

Il y aura toujours un terme (le premier ou le dernier selon le schéma utilisé) qui ne pourra pas être calculé.

Pour régler le problème de la taille de la liste, une solution basique consiste à créer un vecteur dérivé rempli de zéros et de calculer la dérivée avec un schéma arrière.

Le problème étant qu'il peut y avoir un saut au début ou à la fin du vecteur dérivé. Ce saut n'étant pas physique, il conviendra de le traiter.

Une seconde solution, bien meilleure consiste à utiliser un schéma numérique avant pour le premier point du vecteur puis un schéma numérique arrière pour tous les autres points.



4.2.2.1 Python

```python
def derivationSchemaArriere(y, x):
    dy = []
    for i in range(1,len(y)):
        # Calcul par schéma arrière
        derivee = (y[i]-y[i-1])/(x[i]-x[i-1])
        dy.append(derivee)
    # rajouter le schema avant au début
    dy = [dy[0]]+dy
    # Retourner le résultat
    return dy
```

```python
def derivationSchemaAvant(y, x):
    dy = []
    for i in range(len(y)-1):
        # Calcul par schéma avant
        derivee = (y[i+1]-y[i])/(x[i+1]-x[i])
        dy.append(derivee)
    # rajouter le schema arrière à la fin
    dy = dy+[dy[-1]]
    # Retourner le résultat
    return dy
```

Cette même fonction avec les fonctions de la librairie `numpy`

```python
from numpy import array, zeros

def derivationSchemaArriere(y, x):
    # Précaution
    y, x = array(y), array(x)
    # Création d'un vecteur rempli de zéros
    dy = zeros(len(y))
    # Calcul par schéma arrière
    dy[1:] = (y[1:] - y[:-1])/(x[1:] - x[:-1])
    # Calcul du schema avant au début
    dy[0] = dy[1]
    # Retourner le résultat
    return dy
```

```python
from numpy import array, zeros

def derivationSchemaAvant(y, x):
    # Précaution
    y, x = array(y), array(x)
    # Création d'un vecteur rempli de zéros
    dy = zeros(len(y))
    # Calcul par schéma avant
    dy[:-1] = (y[1:] - y[:-1])/(x[1:] - x[:-1])
    # Calcul du schema arrière à la fin
    dy[-1] = dy[-2]
    # Retourner le résultat
    return dy
```

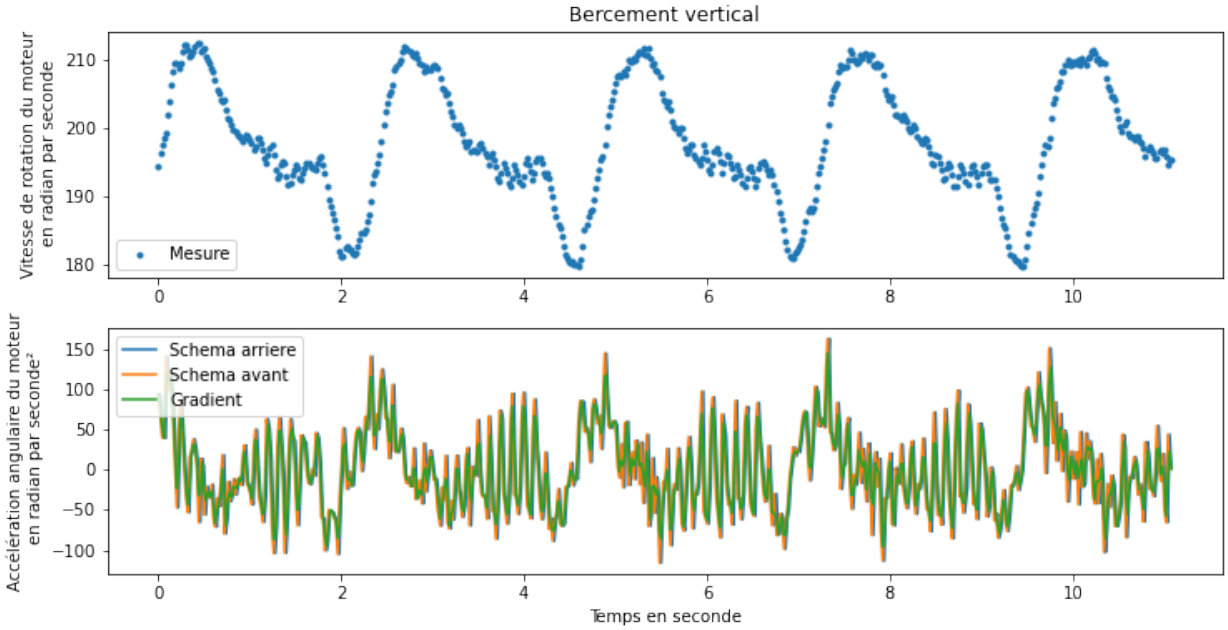
4.2.2.2 Librairie

Nous avons la possibilité d'utiliser la fonction `gradient` de la bibliothèque `numpy` pour calculer la dérivée d'un vecteur. Un extrait de sa documentation est donné ci-dessous :

```
gradient(f, *varargs, **kwargs)
    Return the gradient of an N-dimensional array.

    The gradient is computed using second order accurate central differences
    in the interior points and either first or second order accurate one-sides
    (forward or backwards) differences at the boundaries.
    The returned gradient hence has the same shape as the input array.

    Parameters
    ----------
    f : array_like
        An N-dimensional array containing samples of a scalar function.

    Returns
    -------
    gradient : ndarray or list of ndarray
        A set of ndarrays (or a single ndarray if there is only one dimension)
        corresponding to the derivatives of f with respect to each dimension.
        Each derivative has the same shape as f.
```

Exemple

```python
from numpy import gradient

acc = gradient(wV, t)
```



Mise en garde : La dérivation d'un signal ou d'une fonction échantillonnée génère beaucoup de *bruits*. Il sera nécessaire de *filtrer* le signal (avant ou après dérivation).

5 Filtrage numérique

Il n'est pas rare que les signaux acquis soient bruités et qu'il soit nécessaire de les filtrer.

5.1 Moyenne

5.1.1 Moyenne arithmétique

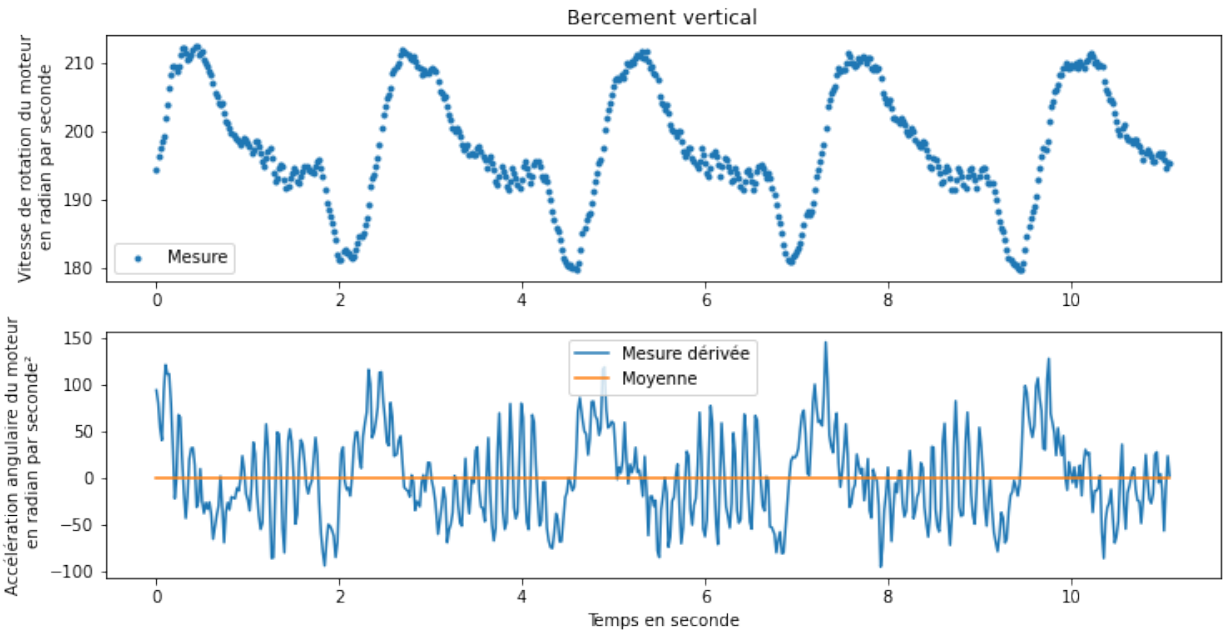
La moyenne d'une liste de nombre x_1 ; x_2 ; ... ; x_n se définit par : $\overline{x} = \dfrac{\sum_{i=1}^{n} x_i}{n}$.

Nous avons la possibilité d'utiliser la fonction `mean` de la librairie `numpy` pour obtenir la moyenne d'un vecteur. Un extrait de sa documentation est donné ci-dessous :

```
mean(a, axis=None, dtype=None, out=None, keepdims=<no value>)
    Compute the arithmetic mean along the specified axis.

    Returns the average of the array elements.  The average is taken over
    the flattened array by default, otherwise over the specified axis.
    ‘float64‘ intermediate and return values are used for integer inputs.

    Parameters
    ----------
    a : array_like
        Array containing numbers whose mean is desired. If ‘a‘ is not an
        array, a conversion is attempted.

    Returns
    -------
    m : ndarray, see dtype parameter above
        If ‘out=None‘, returns a new array containing the mean values,
        otherwise a reference to the output array is returned.
```

Exemple

```python
from numpy import mean

iVmoy = mean(wV)
```



Nous pouvons nous rendre compte que nous avons filtré le signal à l'extrême...

5.1.2 Ecart-type

Il est possible de quantifier cette écart par l'écart-type.

En mathématiques, l'écart-type est une mesure de la dispersion des valeurs d'un échantillon statistique ou d'une distribution de probabilité. **Il est défini comme la moyenne des écarts par rapport à la moyenne.** Il est homogène à la variable mesurée.

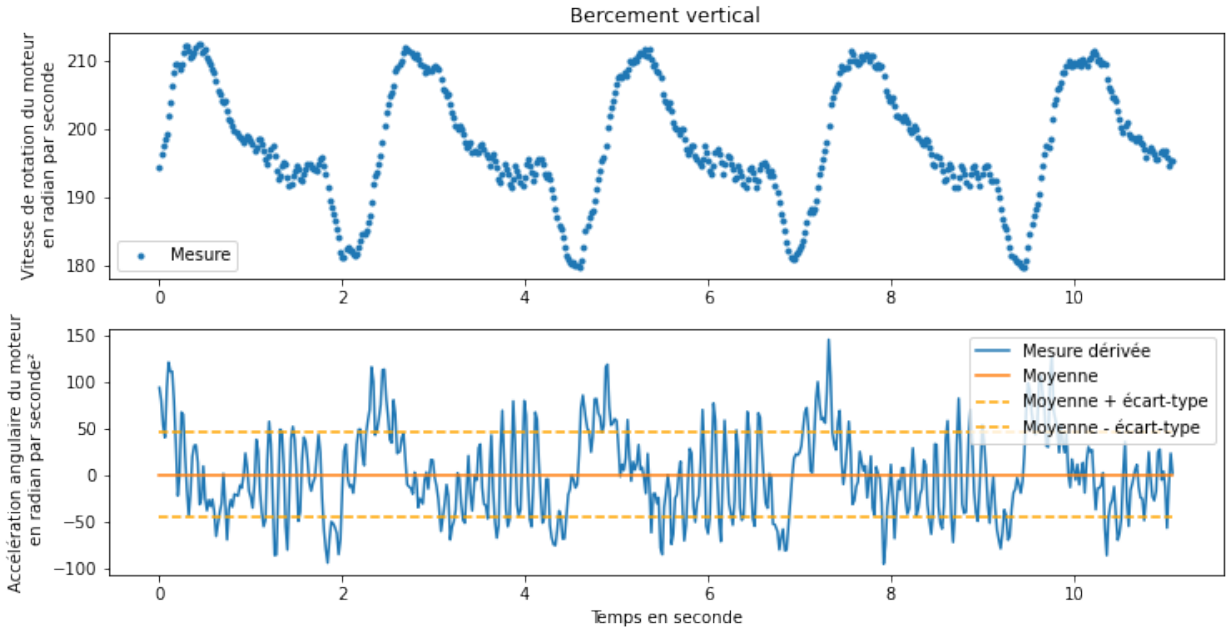
L'écart-type d'une liste de nombre x_1 ; x_2 ; ... ; x_n se définit par : $\sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \overline{x})^2}{n}}$.

Nous avons la possibilité d'utiliser la fonction `std` de la librairie `numpy` pour obtenir l'écart-type d'un vecteur. Un extrait de sa documentation est donné ci-dessous :

```
std(a, axis=None, dtype=None, out=None, ddof=0, keepdims=<no value>)
    Compute the standard deviation along the specified axis.

    Returns the standard deviation, a measure of the spread of a distribution,
    of the array elements. The standard deviation is computed for the
    flattened array by default, otherwise over the specified axis.

    Parameters
    ----------
    a : array_like
        Calculate the standard deviation of these values.
    axis : None or int or tuple of ints, optional
        Axis or axes along which the standard deviation is computed. The
        default is to compute the standard deviation of the flattened array.

    Returns
    -------
    standard_deviation : ndarray, see dtype parameter above.
        If ‘out‘ is None, return a new array containing the standard deviation,
        otherwise return a reference to the output array.
```

Exemple

```python
from numpy import mean, std

print('Moyenne :', round(mean(iV),4))  # Moyenne : 0.000213
print('Ecart-type :',round(std(iV),4)) # Ecart-type : 25.6964
```



Dans ce cas, ce filtrage n'est pas intéressant.

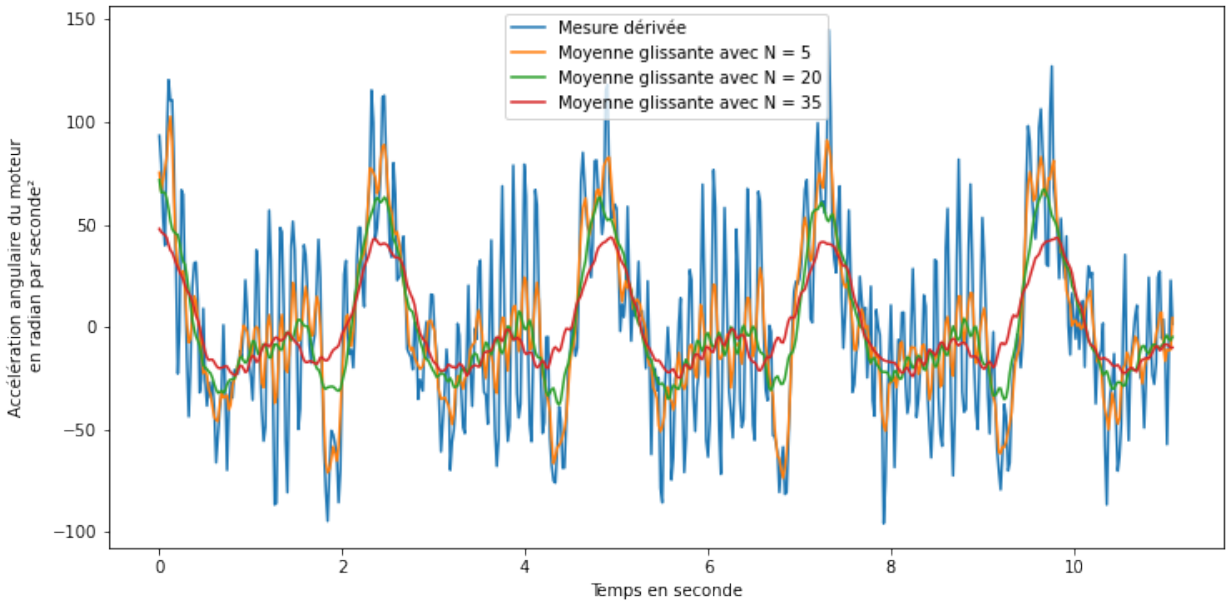
5.2 Moyenne glissante

La moyenne mobile, ou moyenne glissante, est un type de moyenne statistique utilisée pour analyser des séries ordonnées de données, le plus souvent des séries temporelles, en supprimant les fluctuations transitoires de façon à en souligner les tendances à plus long terme. Cette moyenne est dite glissante parce qu'elle est recalculée de façon continue, en utilisant à chaque calcul un sous-ensemble d'éléments dans lequel un nouvel élément remplace le plus ancien ou s'ajoute au sous-ensemble.

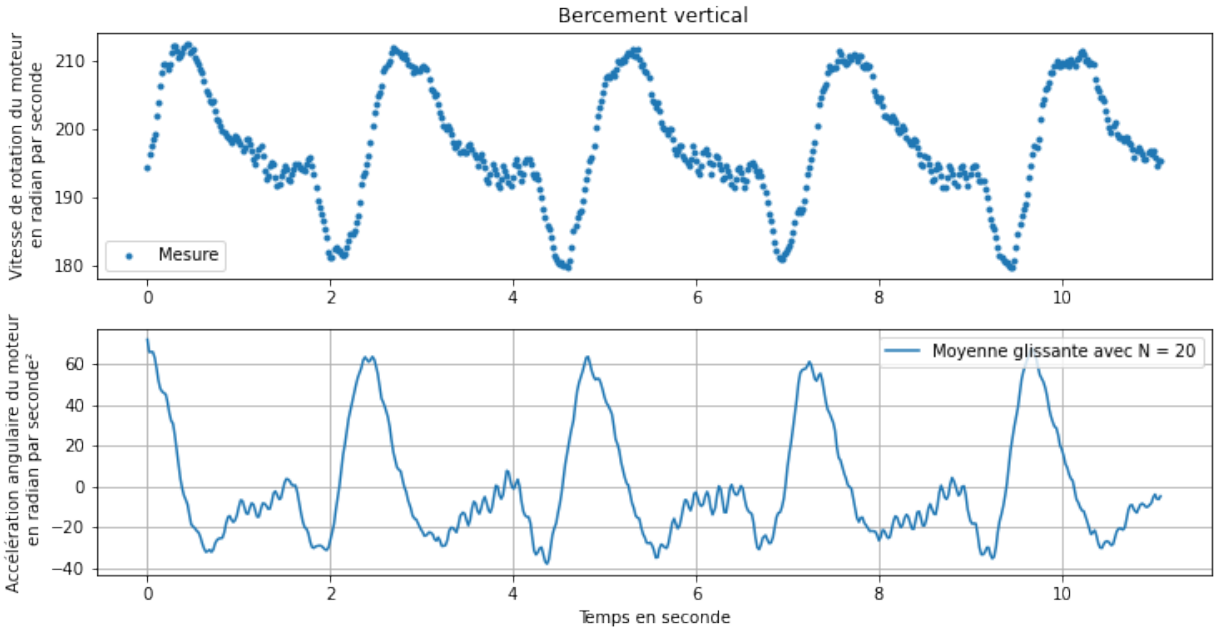
La moyenne mobile d'une liste de nombre x_1 ; x_2 ; ... ; x_n se définit par : $\overline{x}_i = \dfrac{\sum_{k=0}^{N-1} x_{n-k}}{N}$.

```python
from numpy import mean

def moyenneGlissante(l, n):
    """
    Fonction flitrant un signal par la moyenne glissante
        Entrées :
            * l : liste de flottant
            * n : étendue de la moyenne glissante
        Sortie :
            liste de flottant filtrée
    """
    # Taille de la liste
    taille = len(l)

    # Créer une liste aussi grande que les données
    result = [None] * taille

    for i in range(taille):
        # chercher les bornes de la sous liste dont on doit faire la moyenne
        a, b = i - n//2, i + n//2 + 1
        # les bornes doivent compatible avec la liste
        a, b = max(0, a), min(b, taille)

        # Faire la moyenne
        result[i] = mean(l[a:b])

    return result
```

Exemple



Nous pouvons nous rendre compte que plus N est grand plus la courbe est lissée mais également qu'elle peut en amplitude.



Dorénavant, il est plus facile d'interpréter le sens physique en pouvant maintenant observer les phases d'accélération (positives) de de décélération (négatives).

5.3 Filtre numérique passe-bas du premier ordre

La fonction de transfert d'un filtre passe-bas du premier ordre est $\frac{S(p)}{E(p)} = \frac{1}{1+\tau \cdot p}$ avec $\omega_c = \frac{1}{\tau}$ la pulsation de coupure du filtre.

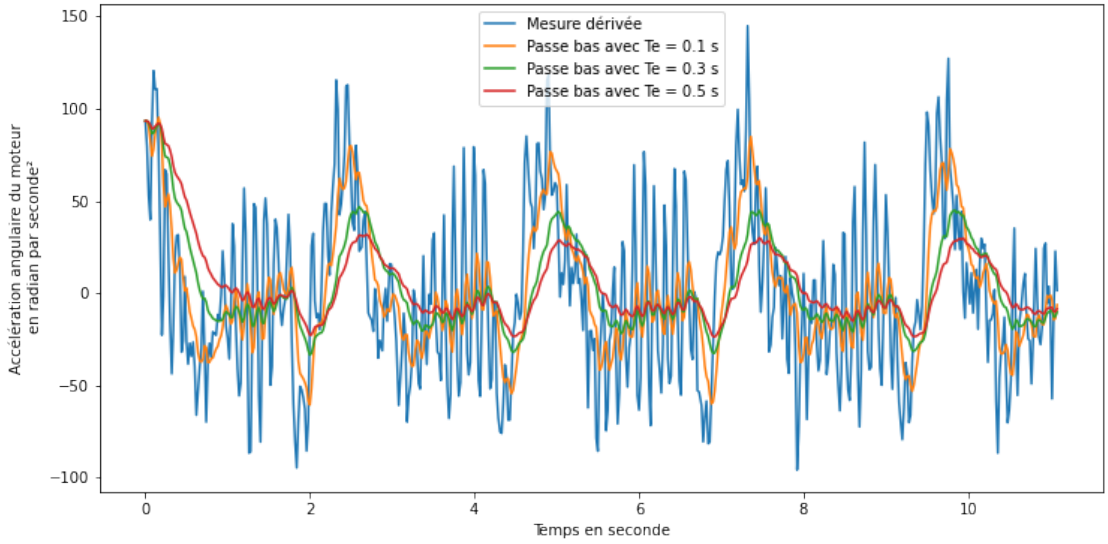
L'équation différentielle associée à ce filtre est $\tau \cdot \left[\frac{\mathrm{d}}{\mathrm{d}t} s(t)\right] + s(t) = e(t)$

L'équation discrétisée devient alors $\tau \cdot \frac{s_{i+1} - s_i}{T_e} + s_i = e_i$ avec T_e le période d'échantillonnage du signal.

L'équation permettant de calculer s_{i+1} quelque soit i est alors $s_{i+1} = s_i + \frac{T_e \cdot (e_i - s_i)}{\tau}$

```python
def filtrePasseBas(t, e, tau):
    """
    Fonction qui permet de réaliser un filtre passe bas
    du premier ordre sur un signal donné.

    Entrées: - t : vecteur contenant les instants de mesure
             - e : vecteur contenant les données à filtrer
             - tau : constante de temps pour définir la fréquence de coupure
    Sortie: s: vecteur du signal filtré
    """

    Te = (t[-1]-t[0])/(len(t)-1)

    # Création du vecteur de sortie
    s = zeros(len(e))
    s[0] = e[0]

    # Filtrage du signal
    for i in range(len(e)-1):
        s[i+1] = s[i] + Te*(e[i] - s[i])/tau

    return s
```

Exemple



Remarque : L'application du filtre passe-bas génère un retard de τ du signal.

5.4 Filtre numérique passe-bas d'ordre supérieur : Filtre de Butterworth

Le filtre Butterworth est un filtre d'ordre n qui a l'avantage de filtrer davantage au delà de la fréquence de coupure que le filtre du premier ordre.

Il s'agit d'un filtre acausal. Il ne peut donc pas filtrer en temps réel.

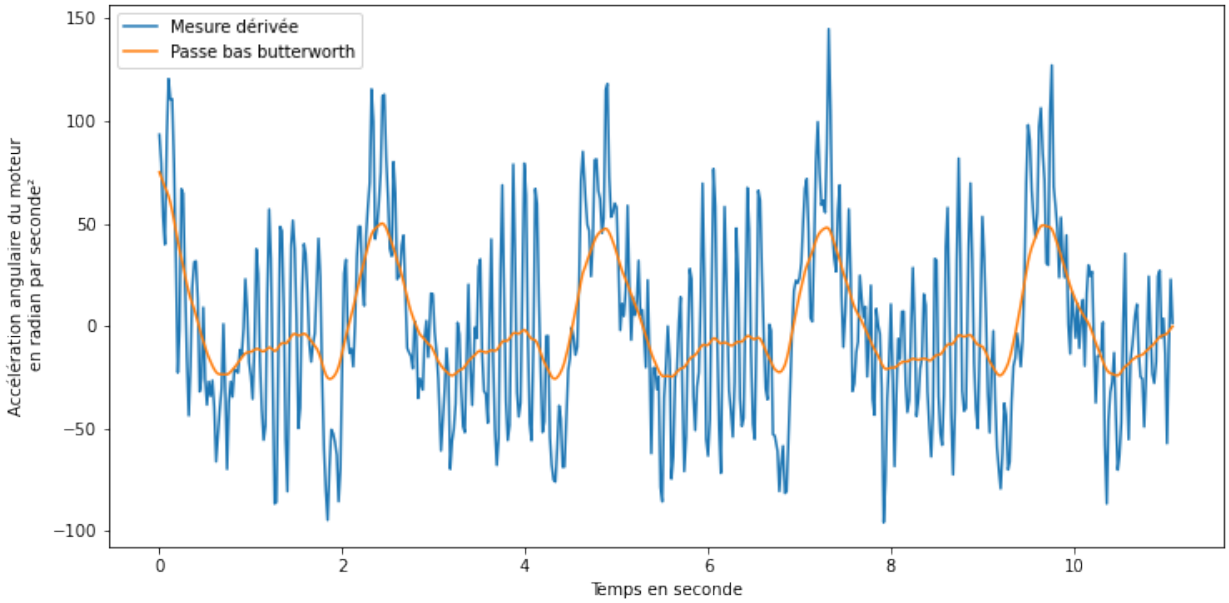
Contrairement au filtre passe-bas du premier ordre, il ne présente pas de retard.

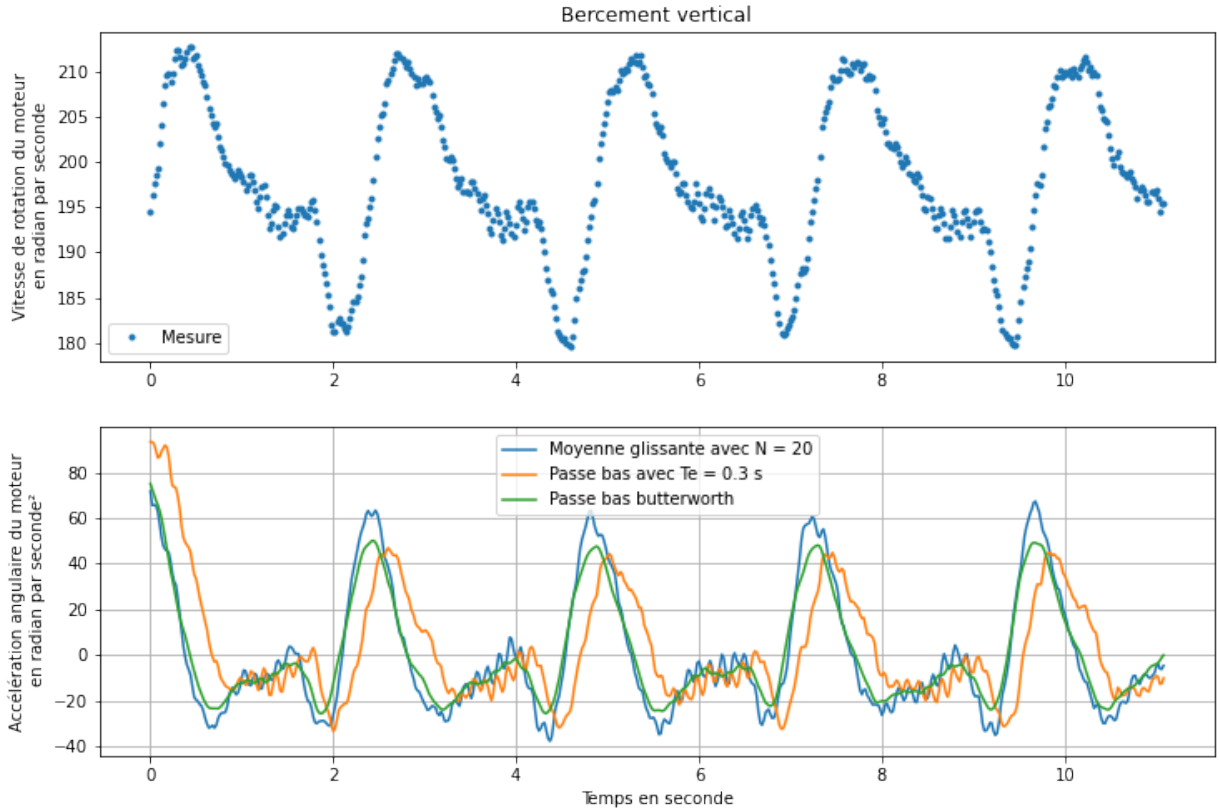
La fonction `butter` de la bibliothèque `scipy.signal` permet de calculer les coefficients du polynôme caractéristique de la fonction de transfert du filtre.
La fonction `filtfilt` de la biliothèque `scipy.signal` réalise le filtrage à partir de ces coefficients.

Exemple

On donne ci-dessous un exemple de fonction qui permet d'utiliser le filtre Butterworth en filtre passe-bas.

```python
def Butterworth(vecTemps, vecSig, ordre, fc):
    """
    Fonction qui permet de réaliser un filtre passe-bas
    butterworth sur le signal d'entrée.

    Entrées: - vecTemps: vecteur contenant les instants de mesure
             - vecSig: vecteur contenant les données à filtrer
             - ordre: ordre du filtre
             - fc: fréquence de coupure
    Sortie: vecFiltre: vecteur du signal filtré
    """

    # Période déchantillonage
    Te = vecTemps[1] - vecTemps[0]

    # Calcul de la fréquence de Nyquist
    fNyq = 1./(2*Te)

    # Préparation du filtre de Butterworth en passe-bas
    b, a = scipy.signal.butter(ordre, fc/fNyq, 'low')

    # Application du filtre
    vecFiltre = scipy.signal.filtfilt(b, a, vecSig)

    return vecFiltre
```

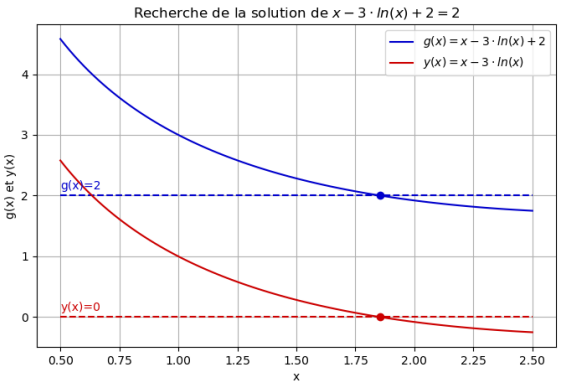

Mesure dérivée
Passe bas butterworth
Accélération angulaire du moteur en radian par seconde²
Temps en seconde


Bercement vertical
Vitesse de rotation du moteur en radian par seconde
Mesure
Moyenne glissante avec N = 20
Passe bas avec Te = 0.3 s
Passe bas butterworth
Accélération angulaire du moteur en radian par seconde²
Temps en seconde

6 Recherche de zéro

Les méthodes de recherche de zéro permettent de déterminer la solution d'une équation du type $g(t) = g_0$ avec g une fonction continue et monotone sur un intervalle et ce avec une complexité des plus faible.

Le graphique de droite montre comment rechercher la solution de l'équation $x - 3 \cdot ln(x) + 2 = 2$ sur l'intervalle $[0.25; 2.5]$. Pour cela le problème est identique à rechercher le zéro de la fonction $y(t) = g(t) - g_0$ sur ce même intervalle.



Ne pouvant pas obtenir la valeur exacte de ces zéros, on peut en obtenir une valeur approchée à un ε près.

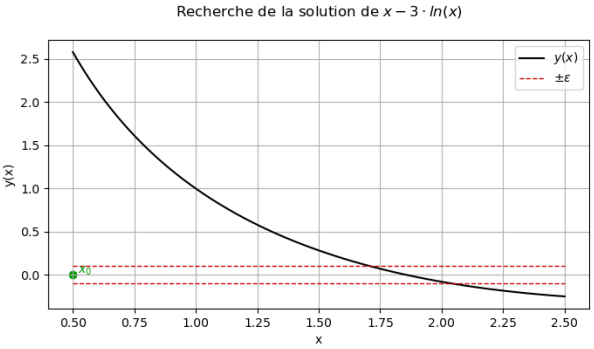
6.1 Méthode de Newton

6.1.1 Principe de fonctionnement

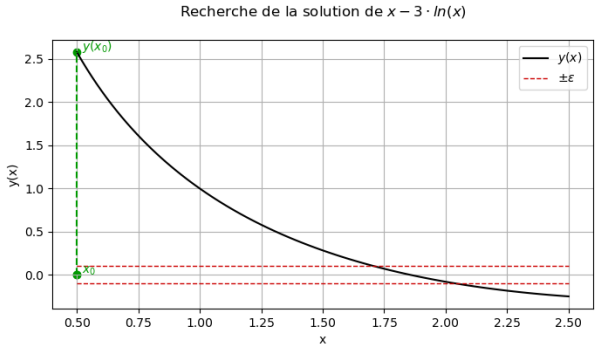
6.1.1.1 Définition

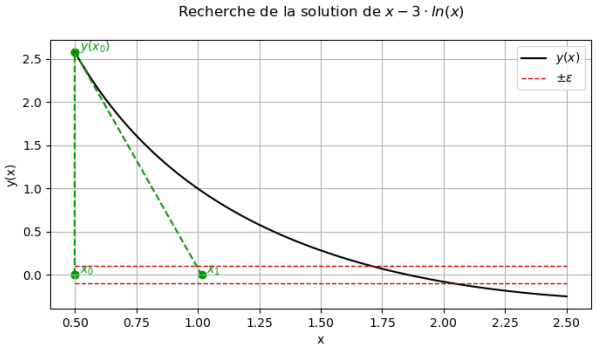
L'algorithme de recherche d'un zéro d'une fonction f par la méthode de Newton consiste à choisir une première valeur a proche du zéro de la fonction. On trace la tangente à la courbe de la fonction f en a puis on recherche le point d'intersection de cette tangente avec l'axe des abscisses.

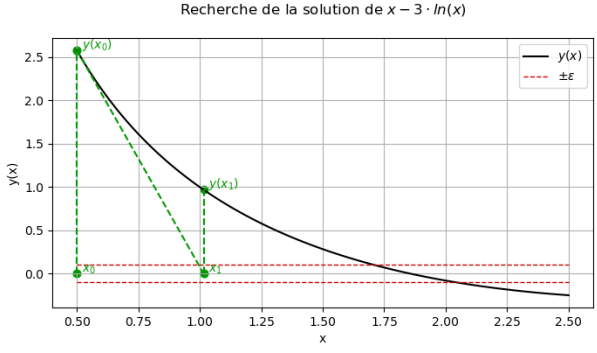
On continue avec ce point précédemment défini jusqu'à ce que la valeur de la fonction soit assez proche de zéro à une valeur ε près.

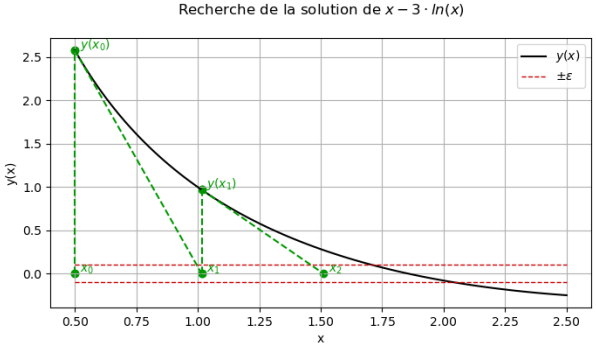


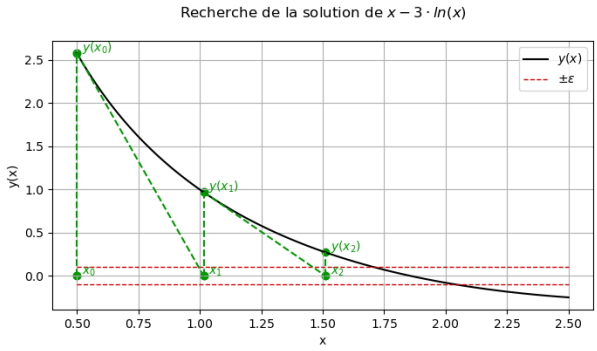
Exemple pour déterminer le zéro de la fonction $f(x) = x - 3 \cdot \ln(x)$ avec une précision $\varepsilon = 0.1$ en utilisant la méthode de Newton.

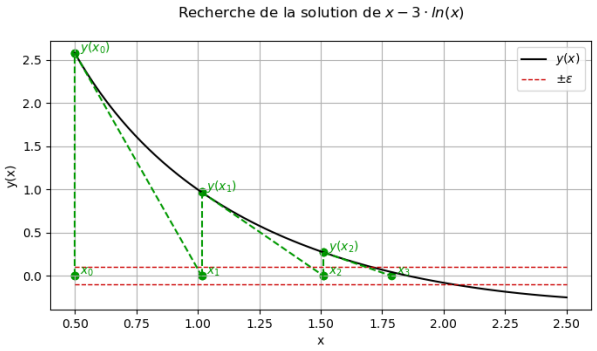


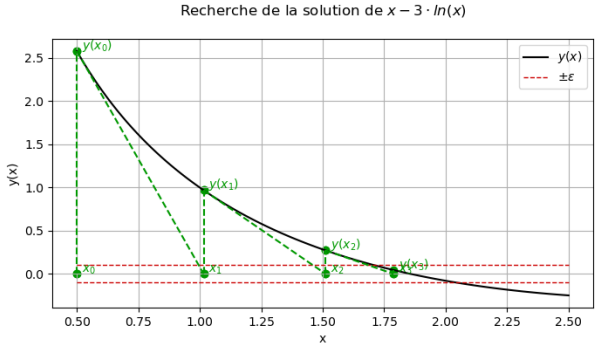












6.1.2 Librairie

La fonction `newton` bibliothèque `scipy.optimize` permet de rechercher des zéros de fonction. On donne ci-dessous un extrait de sa documentation.

```
newton(func, x0, fprime=None, args=(), tol=1.48e-08, maxiter=50)
   Find a zero using the Newton-Raphson or secant method.
   Find a zero of the function ‘func‘ given a nearby starting point ‘x0‘.
   The Newton-Rapheson method is used if the derivative ‘fprime‘ of
   ‘func‘ is provided, otherwise the secant method is used.

    Parameters
    ----------
    func : function
        The function whose zero is wanted. It must be a function of a
        single variable of the form f(x,a,b,c...), where a,b,c... are
        extra arguments that can be passed in the ‘args‘ parameter.
    x0 : float
        An initial estimate of the zero that should be somewhere near
        the actual zero.
    fprime : {None, function}, optional
        The derivative of the function when available and convenient.
        If it is None, then the secant method is used.
        The default is None.
    args : tuple, optional
        Extra arguments to be used in the function call.
    tol : float, optional
        The allowable error of the zero value.
    maxiter : int, optional
        Maximum number of iterations.
    Returns
    -------
    zero : float
        Estimated location where function is zero.
    [...]
```

6.2 Méthode par dichotomie

6.2.1 Principe de fonctionnement

6.2.1.1 Définition

La recherche de zéro par dichotomie utilise le théorème des valeurs intermédiaires énoncé ci-dessous.

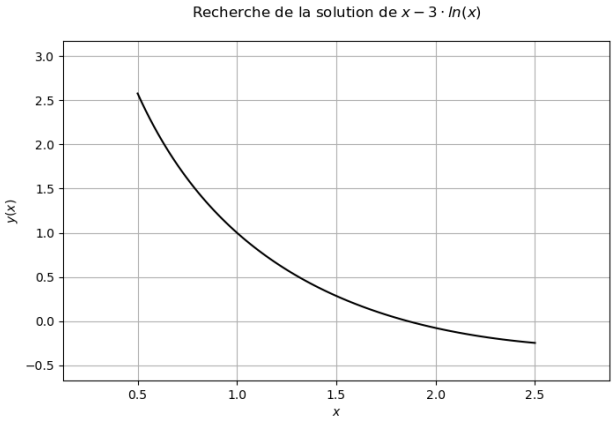
Théorème des valeurs intermédiaires Soit f une fonction définie et continue sur l'intervalle $[a,b]$ à valeur dans $\mathbb{R}$. Pour tout $u \in [f(a), f(b)]$, il existe au moins un réel $c \in [a,b]$ tel que $f(c) = u$. En particulier (Théorème de Bolzano), si $f(a)$ et $f(b)$ sont de signes différents, il existe au moins un réel c tel que $f(c) = 0$.

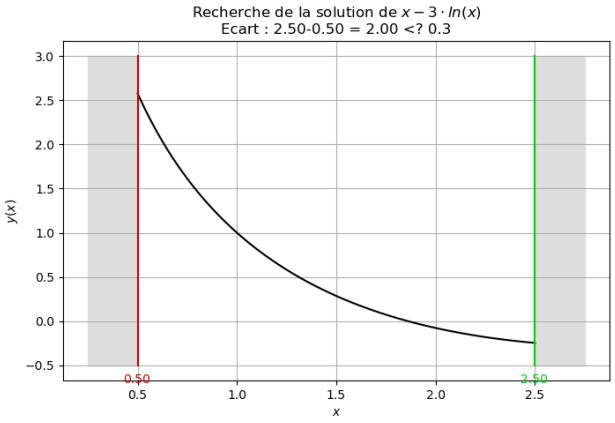
Ainsi, pour une fonction donnée définie sur un intervalle donné, le but de l'algorithme de dichotomie va être de découper en 2 l'intervalle [a,b] en deux, afin d'y trouver la solution. Par divisions successives de l'intervalle, on convergera vers la solution.

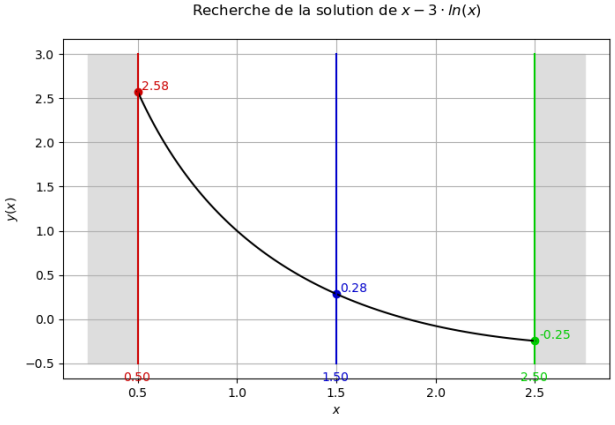
Il existe plusieurs méthodes pour tester si $f(a)$ et $f(b)$ sont de signes différents. Si on ne se préoccupe pas de savoir la relation d'ordre entre $f(a)$ et $f(b)$, un test efficace consiste en un test du signe de $f(a) \cdot f(b)$.

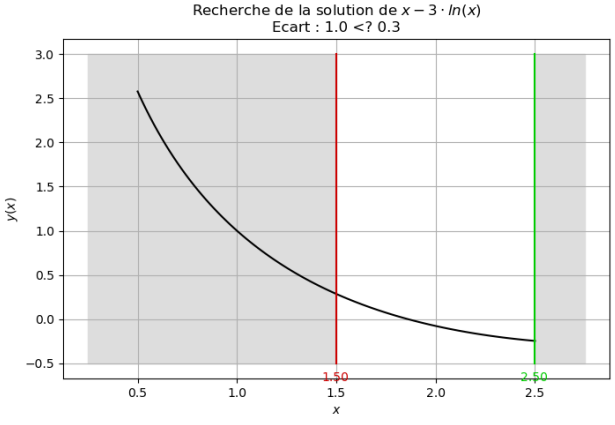
Ainsi tant que l'intervalle $[a,b]$ est plus grand que la précision ε, on divisera l'intervalle en deux pour rechercher un zéro. Si le signe de $f(a) \cdot f(b)$ est négatif, alors le zéro est dans l'intervalle. Sinon il est dans l'autre division.

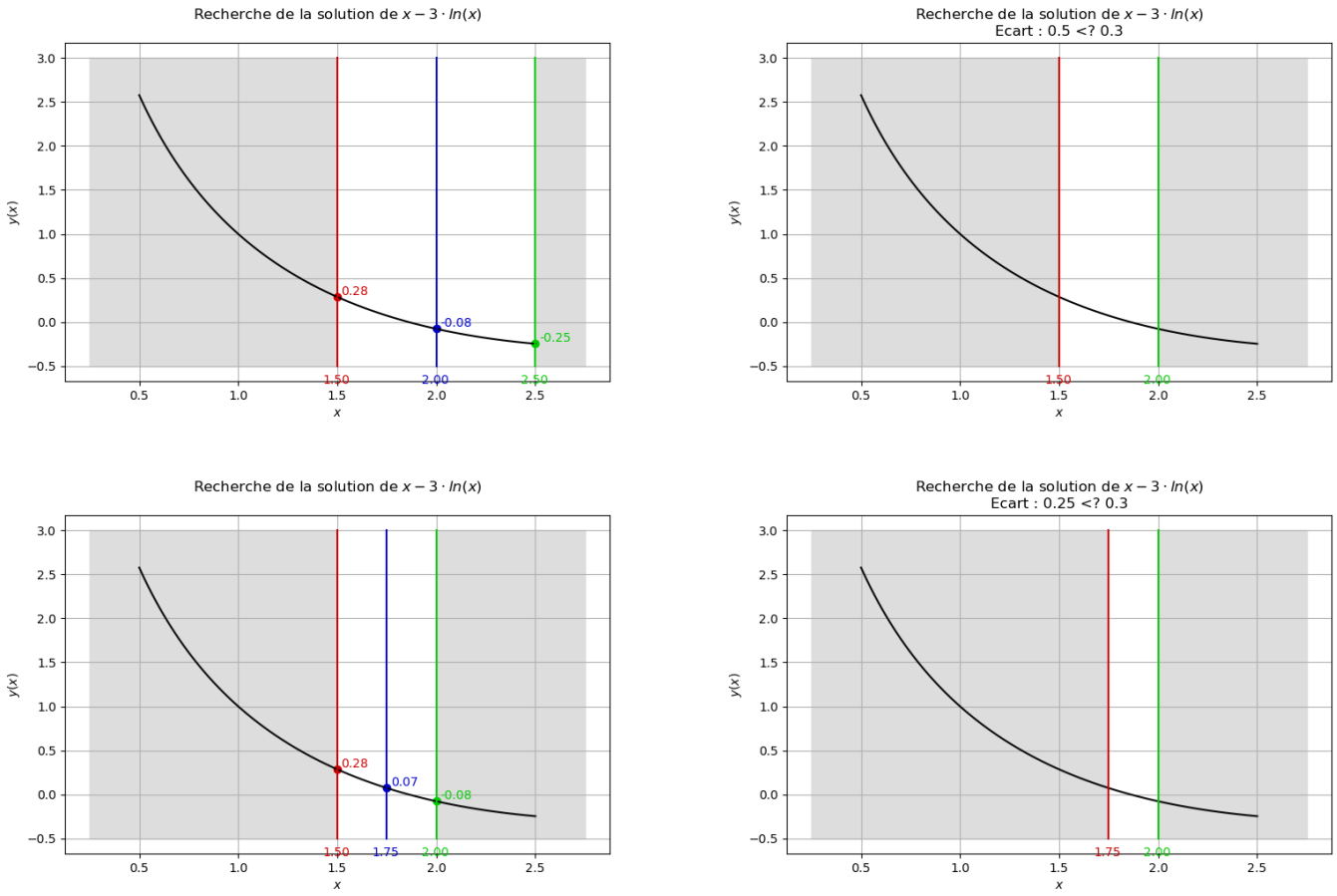
Exemple pour déterminer le zéro de la fonction $y(x) = x - 3 \cdot ln(x)$ avec une précision $\varepsilon = 0.3$ en utilisant la méthode de Newton.











6.2.2 Librairie

Le principe de dichotomie est implémenté dans la fonction `bisect()` du sous-module `optimize` du module `scipy`.

```
Help on function bisect in module scipy.optimize.zeros:

bisect(f, a, b, args=(), xtol=1e-12, rtol=4.4408920985006262e-16,
                            maxiter=100, full_output=False, disp=True)
    Find root of f in [a,b].

    Parameters
    ----------
    f : function
        Python function returning a number.  f must be continuous, and f(a)
        and f(b) must have opposite signs.
    a : number
        One end of the bracketing interval [a,b].
    b : number
        The other end of the bracketing interval [a,b].
    xtol : number, optional
        The routine converges when a root is known to lie within xtol of
        the value return. Should be >= 0.  The routine modifies this to
        take into account the relative precision of doubles.

    Returns
    -------
    x0 : float
        Zero of ‘f‘ between ‘a‘ and ‘b‘.
```

7 Résolution d'un système linéaire du type $A \cdot X = B$

L'algorithme du pivot de Gauss sert à résoudre un système linéaire au sens où, partant d'un système à n équations et p inconnues, il va fournir un système équivalent permettant de paramétrer l'ensemble des solutions.

7.1 Méthodologie

7.1.1 Système linéaire

On dit d'un système qu'il est linéaire s'il est de la forme : $\begin{cases} a_{11} \cdot x_1 + a_{12} \cdot x_2 + \ldots + a_{1p} \cdot x_p &= b_1 \\ \ldots \\ a_{n1} \cdot x_1 + a_{n2} \cdot x_2 + \ldots + a_{np} \cdot x_p &= b_n \end{cases}$

On note $n, p \in \mathbb{N}^*$ (n équations et p inconnues), $\forall i \in \{1, \ldots, n\}$, $\forall j \in \{1, \ldots, p\}$, $a_{ij} \in \mathbb{R}$. i désigne l'indice de la ligne et j l'indice de la colonne.
$b_1, ..., b_n \in \mathbb{R}$ est appelé second membre.

Une solution d'un système linéaire est un p-uplet de réels, c'est-à-dire un élément de $(x_1, \ldots, x_p)$ qui vérifie les n équations.

7.1.2 Mise sous forme matricielle

Au système défini précédemment, on associe la matrice de ses coefficients (ou matrice de transformation) :

$$A = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1(p-1)} & a_{1p} \\ \vdots & & a_{ij} & & \vdots \\ a_{n1} & a_{n2} & \ldots & a_{n(p-1)} & a_{np} \end{bmatrix}$$

On note $B = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}$ la matrice de second membre.

Sous forme matricielle le système s'écrit $A \cdot X = B$ avec X le vecteur inconnu.

7.2 Librairie

La fonction `solve` de la bibliothèque `numpy.linalg` permet de résoudre des problèmes multidimensionnels linéaires. Un extrait de la documentation de la fonction est donnée ci-dessous.

```
solve(a, b)
    Solve a linear matrix equation, or system of linear scalar equations.

    Computes the "exact" solution, `x`, of the well-determined, i.e., full
    rank, linear matrix equation `ax = b`.

    Parameters
    ----------
    a : (..., M, M) array_like
        Coefficient matrix.
    b : {(..., M,), (..., M, K)}, array_like
        Ordinate or "dependent variable" values.

    Returns
    -------
    x : {(..., M,), (..., M, K)} ndarray
        Solution to the system a x = b.  Returned shape is identical to `b`.
```

Exemple

Soit `A` une matrice de coefficients et `B` un vecteur second membre. Le vecteur `X` contient les solutions du problème.

```python
# Importation de la bibliothèque
from numpy.linalg import solve

# Résolution du problème
X = solve(A, B)
```

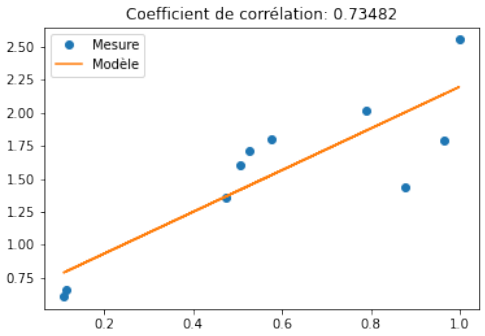
8 Régression linéaire (hors programme mais tellement utile...)

La fonction `linregress` bibliothèque `scipy.stats` permet de réaliser une régression linéaire à partir de deux vecteurs de même taille. On donne ci-dessous un extrait de sa documentation.

```
linregress(x, y=None)
    Calculate a linear least-squares regression for two sets of measurements.

    Parameters
    ----------
    x, y : array_like

    Returns
    -------
    result : ``LinregressResult`` instance
        The return value is an object with the following attributes:

        slope : float
            Slope of the regression line.
        intercept : float
            Intercept of the regression line.
        rvalue : float
            Correlation coefficient.
        pvalue : float
            Two-sided p-value for a hypothesis test whose null hypothesis is
            that the slope is zero, using Wald Test with t-distribution of
            the test statistic.
        stderr : float
            Standard error of the estimated slope (gradient), under the
            assumption of residual normality.
        intercept_stderr : float
            Standard error of the estimated intercept, under the assumption
            of residual normality.
```

Exemple

```python
# Génération aléatoire des données
from numpy import random
rng = random.default_rng()
x = rng.random(10)
y = 1.6*x + rng.random(10)

# Régression linéaire
from scipy import stats
a, b, coef, _, _ = stats.linregress(x, y)
coef = coef**2

# Visualisation du résultat
import matplotlib.pyplot as plt
plt.plot(x, y, 'o', label='Mesure')
plt.plot(x, a * x + b, '-', label='Modèle')
plt.title(f"Coefficient de corrélation: {coef:.5f}")
plt.legend()
plt.savefig('regressionLineaire.png')
plt.show()
```



9 Résolution d'équations différentielles

9.1 Introduction

La résolution d'équations différentielles linéaire ou non et d'ordre supérieur ou égal à un est possible avec python.

Par exemple l'équation différentielle régissant le mouvement d'un moteur à courant continu (lorsque l'inductance et les frottements visqueux sont négligés) :

$$\frac{R \cdot J}{K_c \cdot K_e} \cdot \left[\frac{\mathrm{d}}{\mathrm{d}t}\omega_m(t)\right] + \omega_m(t) = \frac{u(t)}{K_e}$$

avec R, K_e, K_c constantes de couplage électromécaniques du moteur (résistance de l'induit, constante de force contre électromotrice et constante de couple), $\omega_m(t)$ la vitesse de rotation du moteur et $u(t)$ la tension aux bornes de celui-ci.

On note $\omega_c(t)$ la consigne de la vitesse angulaire du moteur proportionnelle à la tension d'alimentation $u(t)$. Elle peut être constante ou non. τ la constante de temps mécanique du moteur. Ainsi on obtient l'équation différentielle suivante :

$$\tau \cdot \left[\frac{\mathrm{d}}{\mathrm{d}t}\omega_m(t)\right] + \omega_m(t) = \omega_c(t) \text{ avec } \tau = \frac{R \cdot J}{K_c \cdot K_e} \text{ et } \omega_c(t) = \frac{\omega_m(t)}{K_e}$$

Celle-ci peut aussi s'écrire sous la forme : $\left[\frac{\mathrm{d}}{\mathrm{d}t}\omega_m(t)\right] = \frac{\omega_c(t) - \omega_m(t)}{\tau} = F(\omega, t)$

Cette forme est plus pratique pour une résolution numérique. La fonction $F(\omega_m, t)$ est une fonction a deux variables qui défini l'équation différentielle.

9.2 Méthode d'Euler explicite

Le principe de la méthode est d'approcher les variations de F entre t_k et t_{k+1} en utilisant la méthode d'intégration numérique par rectangle à gauche (schéma numérique arrière) :

$$\begin{aligned} y(t_{k+1}) - y(t_k) &= \int_{t_k}^{t_{k+1}} y'(u) \cdot \mathrm{d}u \\ &\simeq y'(t_k) \cdot (t_{k+1} - t_k) \\ &= h \cdot y'(t_k) \end{aligned}$$

Comme y est solution de l'équation différentielle $y'(t_k) = F(y(t_k), t_k)$ et ainsi : $y(t_{k+1}) \simeq y(t_k) + h \cdot F(y(t_k), t_k)$

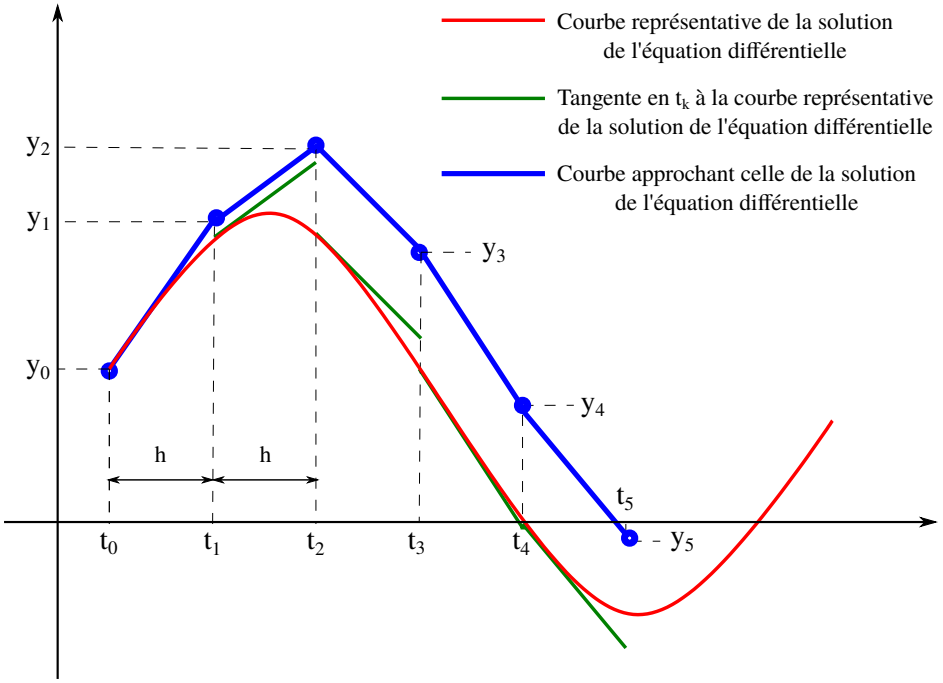
Si y_k est une approximation numérique de la valeur $y(t_k)$ alors on peut dire que $F(y_k, t_k)$ est une approximation de $F(y(t_k), t_k)$. C'est pourquoi on considère la suite (y_k) définie par son premier terme y_0 et :

$$\forall k \in \mathbb{N}, \quad y_{k+1} = y_k + h.F(y_k, t_k)$$

Cette méthode est appelée *Euler explicite* car le nouveau terme de la suite est exprimé de manière *explicite*.

Exemple

Le graphique ci-dessous permettant de déterminer une solution approchée de l'équation différentielle par la méthode Euler explicite.



9.3 Résolution d'équations différentielles d'ordre 1 avec Scipy

Le sous-module `integrate` de la bibliothèque `scipy` contient la fonction `odeint` qui résout numériquement des équations différentielles. Un extrait de l'aide est donné ci-dessous.

```
odeint(func, y0, t, args=(), Dfun=None, col_deriv=0, full_output=0,
ml=None, mu=None, rtol=None, atol=None, tcrit=None, h0=0.0, hmax=0.0,
hmin=0.0, ixpr=0, mxstep=0, mxhnil=0, mxordn=12, mxords=5, printmessg=0)

    Integrate a system of ordinary differential equations.

    Solve a system of ordinary differential equations using lsoda from
    the FORTRAN library odepack.

    Solves the initial value problem for stiff or non-stiff systems
    of first order ode-s::

        dy/dt = func(y,t0,...)

    where y can be a vector.

    Parameters
    ----------
    func : callable(y, t0, ...)
        Computes the derivative of y at t0.
    y0 : array
        Initial condition on y (can be a vector).
    t : array
        A sequence of time points for which to solve for y.  The initial
        value point should be the first element of this sequence.
    args : tuple
        Extra arguments to pass to function.

    Returns
    -------
    y : array, shape (len(y0), len(t))
        Array containing the value of y for each desired time in t,
        with the initial value y0 in the first row.
    [...]
```

Comme écrit dans l'aide, cette fonction permet de résoudre des équations différentielles du type

$\begin{cases} y' = f(t,y) \\ y(t_0) = y_0 \end{cases}$ sur un intervalle $[a,b]$.

Le paramètre `t` est un tableau de valeurs comprises entre a et b, et la valeur renvoyée est un tableau contenant une estimation de la solution en ces différentes valeurs.

Exemple

On cherche à résoudre l'équation de démarrage du moteur dans le cas suivant : La vitesse de consigne est constante à $50\,\mathrm{rad\,s^{-1}}$. La constante de temps τ est fixée à $0,2$ s.

La fonction ci-dessous représente l'équation différentielle du moteur étudié.

```python
def F(omegaM, t):
    """
    Fonction définissant l'équation différentielle du moteur avec :
    - omega : la vitesse du moteur (scalaire à chercher)
    - t : l'instant de calcul
    Elle renvoie la dérivée de la vitesse du moteur
    """

    tau = 0.2
    omegaC = 50
    return (omegaC - omegaM)/tau
```

Le script ci-dessous permet de calculer une solution approchée de ω_m en utilisant la fonction `odeint`.

```python
# Importation des bibliothèques
from numpy import linspace
import scipy.integrate

# Préparation de la solution
tMin = 0
tMax = 1
nbPoints = 100
vecTemps = linspace(tMin, tMax, nbPoints)

# Condition initiale
omega0 = 0

# Calcul de la solution approchée de l'équation différentielle
vecSol = scipy.integrate.odeint(F, omega0, vecTemps)
```

9.4 Résolution d'équations différentielles d'ordre N avec Scipy

On considère un pendule simple assimilé à une masse ponctuelle de masse m, de centre de gravité G relié à un bâti fixe au point O.

On pose $\mathcal{R}_0 = \left(O, \vec{x_0}, \vec{y_0}, \vec{z_0}\right)$ le repère galiléen lié au bâti et $\mathcal{R}_1 = \left(O, \vec{x_1}, \vec{y_1}, \vec{z_1}\right)$ le repère lié au pendule simple.

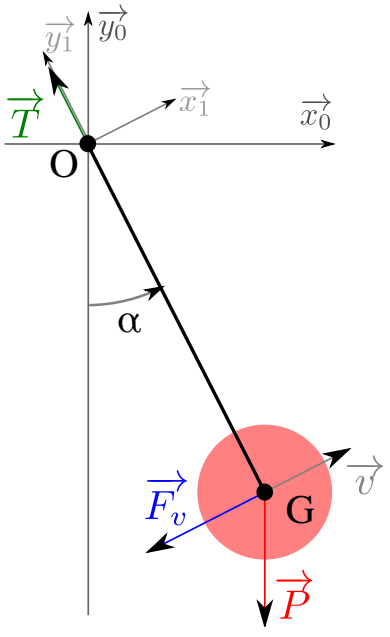
On pose $\overrightarrow{GO} = L \cdot \vec{y_1}$, $\alpha = \left(\vec{y_0}, \vec{y_1}\right)$ et $\vec{V}\left(G, 1/0\right) = L \cdot \left[\dfrac{\mathrm{d}}{\mathrm{d}t}\alpha\right] \cdot \vec{x_1}$.

Ce pendule est soumis aux actions mécaniques suivantes :

— Action de la pesanteur : $\vec{R}\left(g \to 1\right) = -m \cdot g \cdot \vec{y_0}$;

— Tension dans le fil : $\vec{R}\left(0 \xrightarrow{cable} 1\right) = T \cdot \vec{y_1}$;

— Frottement de l'air : $\vec{R}\left(air \to 1\right) = -k \cdot \vec{V}\left(G, 1/0\right) = -k \cdot L \cdot \left[\dfrac{\mathrm{d}}{\mathrm{d}t}\alpha\right] \cdot \vec{x_1}$.



On applique le théorème du moment dynamique en 0 à l'ensemble 1 dans son mouvement par rapport au référentiel galiléen $\mathcal{R}_g$. On obtient l'équation différentielle suivante :

$$\left[\frac{\mathrm{d}^2}{\mathrm{d}t^2}\alpha\right] + \frac{k}{m \cdot L}\left[\frac{\mathrm{d}}{\mathrm{d}t}\alpha\right] + \frac{g}{L} \cdot \sin(\alpha) = 0 \qquad (1)$$

Les conditions initiales sont : $\alpha(0) = \alpha_0$ et $\left[\dfrac{\mathrm{d}}{\mathrm{d}t}\alpha(0)\right] = \dot{\alpha}(0) = 0$.

9.4.1 Réduction d'ordre d'une équation différentielle

Pour résoudre une équation différentielle d'ordre deux ou plus, on peut, en utilisant le schéma Euler explicite, procéder pas à pas pour résoudre l'équation :

$$\begin{cases} y''(t) &= F(y(t), y'(t), t) \\ y'(t_0) &= y'_0 \\ y(t_0) &= y_0 \end{cases} \tag{2}$$

On obtiendrait alors : $F(\alpha, \dot{\alpha},\ t) = \ddot{\alpha} = -\dfrac{k}{m \cdot L}\dot{\alpha} - \dfrac{g}{L}\sin\alpha$

Cette mise en équation n'est pas aisée à résoudre dans le cas d'équations différentielles de grand ordre. L'idée consiste alors de diminuer son ordre en écrivant un système d'équations différentielles d'ordre 1 :

$$F(\alpha, \dot{\alpha},\ t) \Leftrightarrow \begin{cases} \dfrac{\mathrm{d}\alpha}{\mathrm{d}t} = \dot{\alpha} \\ \dfrac{\mathrm{d}\dot{\alpha}}{\mathrm{d}t} = -\dfrac{k}{m.L}\dot{\alpha} - \dfrac{g}{L}\sin(\alpha) \end{cases} \tag{3}$$

On remarque que la première équation du système est triviale. La seconde reprend l'expression de $\ddot{\alpha}$.

9.4.2 Implémentation

Le système d'équation différentielle (3) peut être représenté comme un vecteur caractérisant l'état du système. En automatique, ce type de méthode est souvent utilisé et est appelé la représentation d'état.

Posons alors le vecteur d'état $\vec{Y}$ tel que $\vec{Y} = \begin{pmatrix} \alpha \\ \dot{\alpha} \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$. Le système d'équations devient alors :

$$\begin{cases} \left[\dfrac{\mathrm{d}}{\mathrm{d}t}y_1\right] = y_2 \\ \left[\dfrac{\mathrm{d}}{\mathrm{d}t}y_2\right] = -\dfrac{k}{m \cdot L} \cdot y_2 - \dfrac{g}{L} \cdot \sin(y_1) \end{cases} \quad \Longrightarrow \quad \left[\dfrac{\mathrm{d}}{\mathrm{d}t}\vec{Y}\right] = \vec{F}\left(\vec{Y},\ t\right) \tag{4}$$

Exemple

Une fonction `F(vecEtat, t)` qui défini le système d'équations différentielles d'ordre un du pendule défini précédemment. `vecEtat` est un vecteur Numpy contenant deux éléments : α et $\dot{\alpha}$. Elle renverra un vecteur Numpy contenant $\dot{\alpha}$ et $\ddot{\alpha}$.

On posera $\dfrac{k}{m \cdot L} = 0.8$ et $\dfrac{g}{L} = 10$

```python
from numpy import array

def F(vecEtat, t):
    """
    Fonction qui défini le système d'équation différentielle
    donnant le mouvement du pendule.
    Entrées: - vecEtat: vecteur d'état du pendule [alpha, alphaPoint]
             - t: instant de calcul
    Sortie: vecteur contenant [alphaPoint, alphaPointPoint]
    """

    alphaPoint = vecEtat[1]
    alphaPointPoint = - 0.8 * vecEtat[1] - 10*np.sin(vecEtat[0])

    return array([alphaPoint, alphaPointPoint])
```

Un script qui permet de calculer la position t la vitesse angulaire du pendule au cours du temps. La position initiale sera de 1 rad et la vitesse initiale nulle. On fera le calcul sur 8 s en utilisant 400 points.

```python
# Importation des bibliothèques
from numpy import linspace
import scipy.integrate

# Condition initiale
alpha0 = 1
alphaPoint0 = 0
vecCI = np.array([alpha0, alphaPoint0])

# Vecteur du temps
tMin = 0
tMax = 8
nbPoints = 400
vecTemps = linspace(tMin, tMax, nbPoints)

# Résolution de l'ED
matSol = scipy.integrate.odeint(F, vecCI, vecTemps)
# Séparation des sorties
vecAlpha = matSol[:,0]
vecAlphaPoint = matSol[:,1]
```