Informatique

**Méthode d’Euler**

***Cours***

[Méthode d’Euler 4](#_Toc199243784)

[1.I. Discrétisation des problèmes 4](#_Toc199243785)

[1.II. Dérivation numérique 5](#_Toc199243786)

[1.II.1 Dérivée première 5](#_Toc199243787)

[1.II.1.a Euler explicite : dérivée à droite 5](#_Toc199243788)

[1.II.1.a.i Approximation de la dérivée 5](#_Toc199243789)

[1.II.1.a.ii Résolution numérique Euler explicite 6](#_Toc199243790)

[1.II.1.b Euler implicite : dérivée à gauche 6](#_Toc199243791)

[1.II.1.b.i Approximation de la dérivée 6](#_Toc199243792)

[1.II.1.b.ii Résolution numérique Euler implicite 7](#_Toc199243793)

[1.II.1.c Bilan 7](#_Toc199243794)

[1.II.2 Dérivée seconde 8](#_Toc199243795)

[1.II.2.a Taylor à l’ordre 2 8](#_Toc199243796)

[1.II.2.b Double Euler explicite 9](#_Toc199243797)

[1.II.2.c Double Euler implicite 9](#_Toc199243798)

[1.II.2.d Mélange implicite – explicite 10](#_Toc199243799)

[1.II.2.e Conclusion 10](#_Toc199243800)

[1.II.3 Dérivées d’ordre supérieurs 10](#_Toc199243801)

[1.II.4 Précision des solutions 10](#_Toc199243802)

[1.III. Equations différentielles simples du premier et second ordre 11](#_Toc199243803)

[1.III.1 1° ordre 11](#_Toc199243804)

[1.III.2 2° ordre 12](#_Toc199243805)

[1.III.2.a Méthode 12](#_Toc199243806)

[1.III.2.b Remarques 13](#_Toc199243807)

[1.III.2.b.i Concernant les array 13](#_Toc199243808)

[1.III.2.b.ii Conditions initiales atypiques 13](#_Toc199243809)

[1.III.2.c Exemples de mise en équation 14](#_Toc199243810)

[1.III.2.c.i Exemple 1 : Masse en chute libre 14](#_Toc199243811)

[1.III.2.c.ii Exemple 2 : Pendule 14](#_Toc199243812)

[1.IV. Systèmes d’équations différentielles du premier et du second ordre 15](#_Toc199243813)

[1.IV.1 1° ordre 15](#_Toc199243814)

[1.IV.2 2° ordre 16](#_Toc199243815)

[1.V. Modélisation par équations aux différences 17](#_Toc199243816)

[1.V.1 Contexte 17](#_Toc199243817)

[1.V.2 Méthode de résolution 17](#_Toc199243818)

[1.V.3 Exemple 1 : Chute libre 18](#_Toc199243819)

[1.V.3.a Equation récurrente 18](#_Toc199243820)

[1.V.3.b Prise en compte des conditions initiales 19](#_Toc199243821)

[1.V.3.c Remarque : Regroupement de termes dépendant de z dans le second membre 19](#_Toc199243822)

[1.V.4 Exemple 2 : Pendule 20](#_Toc199243823)

[1.V.4.a Equation récurrente 20](#_Toc199243824)

[1.V.4.b Prise en compte des conditions initiales 20](#_Toc199243825)

[1.VI. Résolution d’équations et précision 21](#_Toc199243826)

[1.VI.1 Précision sur la solution 21](#_Toc199243827)

[1.VI.2 Précision sur le temps 22](#_Toc199243828)

[1.VII. Fonction de résolution intégrée à Python 23](#_Toc199243829)

###### Méthode d’Euler

# Discrétisation des problèmes

Jusqu’ici, nous avons étudié des systèmes continus. Les variables traitées étaient des fonctions continues du temps.

Dans bon nombre de systèmes, les variables sont échantillonnées et on ne connaît qu’une liste de valeurs à différents temps. On parle de variables discrètes.

Système continu

Système discret

Cette figure représente un signal continu en fonction du temps sur lequel ont été prises des valeurs à différents temps espacés d’un temps supposé ici constant et appelé « période d’échantillonnage ».

# Dérivation numérique

Soit la variable discrète .

Voyons comment déterminer une approximation de la dérivée de y. On parle de différences finies, de méthode d’Euler, de développement de Taylor. Voyons par ailleurs commet réaliser une résolution numérique d’une équation différentielle à l’aide de ces approximations de dérivées.

## Dérivée première

### Euler explicite : dérivée à droite

#### Approximation de la dérivée

Proposons le développement de Taylor de la variable à l’ordre 1

On a donc :

Cette approximation de la dérivée correspond à une méthode Euler Explicite.

Cela revient à approcher la dérivée à un instant en utilisant la valeur à l’instant et la valeur à l’instant

La dérivée à l’instant est approchée à l’aide de la valeur en et **après**.

Remarque : Méthode nécessaire pour calculer la dérivée d’une liste de valeurs en son **premier point** !

#### Résolution numérique Euler explicite

A l’aide de l’approximation précédente, on a : . Il est possible d’approcher la prochaine valeur en fonction de la valeur actuelle connue et de la dérivée à l’instant t connue. On dit que ce calcul est explicite car dépendant du temps actuel t, ou tout est connu.

Que la dérivée de ne dépende que du temps ou du temps et de , on peut réaliser le calcul de  car tout est connu au temps t ☺.

### Euler implicite : dérivée à gauche

#### Approximation de la dérivée

On peut aussi associer la dérivée en à cette expression :

Cela revient à écrire :

Cette approximation de la dérivée correspond à une méthode Euler implicite.

La dérivée à l’instant est approchée à l’aide de la valeur en et **avant**.

Remarque : Méthode nécessaire pour calculer la dérivée d’une liste de valeurs en son **dernier point** !

#### Résolution numérique Euler implicite

A l’aide de l’approximation précédente, on a : . La prochaine valeur s’exprime en fonction de la valeur actuelle connue et de la dérivée à l’instant , qui peut être connue… ou non ! On dit que ce calcul est implicite car dépendant du temps suivant , où tout n’est pas forcément connu…

Lorsque la dérivée de ne dépend que du temps , il est possible de réaliser le calcul de car est connue.

Cependant, lorsque la dérivée dépend aussi de la valeur de y , il n’est pas possible de calculer car est inconnu. Dans ce cas, on peut faire l’approximation afin d’estimer .

Les résolutions sont très généralement réalisées avec la méthode d’Euler explicite. Euler implicite sera plutôt utilisé pour calculer la dérivée d’une fonction représentée de manière discrète dont les valeurs sont connues à tout instant et lorsque l’on se trouve à son dernier point. En effet, n’existant pas, on calculera .

### Bilan

|  |  |
| --- | --- |
| Euler explicite | Euler implicite |
|  |  |
| Nouvelle valeur dépendant de la dérivée au temps présent  Explicite | Nouvelle valeur dépendant de dérivée au temps suivant, pas forcément calculable  Implicite |

## Dérivée seconde

On trouve plusieurs méthodes pour estimer le dérivée seconde d’une variable discrète.

### Taylor à l’ordre 2

Proposons deux développements de Taylor de la variable à l’ordre 2

Faisons la somme de ces deux expressions :

Soit :

Cette approximation calcul la dérivée seconde de manière centrée autour du temps

### Double Euler explicite

Ecrivons l’expression de la dérivée seconde de avec la méthode d’Euler Explicite vue précédemment :

On a par ailleurs :

Soit

On remarque ici que l’approximation de la dérivée est faite avec les valeurs en et après.

### Double Euler implicite

Ecrivons l’expression de la dérivée seconde de avec la méthode d’Euler Explicite vue précédemment :

On a par ailleurs :

Soit

On remarque ici que l’approximation de la dérivée est faite avec les valeurs en et avant.

### Mélange implicite – explicite

On peut aussi mélanger les méthodes :

|  |  |
| --- | --- |
| Implicite sur la dérivée seconde et explicite sur la dérivée première | Explicite sur la dérivée seconde et implicite sur la dérivée première |
|  |  |

Dans les deux cas, on retrouve l’approximation centrée issue de l’application de deux développements de Taylor à l’ordre 2.

### Conclusion

La dérivée en t s’exprime en fonction de valeurs de la fonction en :

**Double implicite**

**Double explicite**

**Double Taylor**

**Mélange implicite/explicite**

On veut généralement trouver connaissant et . On préfèrera donc écrire proprement un double développement de Taylor

## Dérivées d’ordre supérieurs

On pourra procéder de la même manière pour obtenir des dérivées d’ordres supérieurs

## Précision des solutions

Plus sera petit, plus les résultats seront proches de la réalité, la dérivée réelle étant la limite de nos formules quand tend vers 0…

# Equations différentielles simples du premier et second ordre

Nous allons dans un premier temps voire une méthode permettant de résoudre des équations du type :

On suppose connaitre les conditions initiales.

On pourrait par exemple résoudre , mais aussi des équations plus complexes comme

## 1° ordre

Soit l’équation :

Discrétisons avec la méthode d’Euler explicite :

On a donc :

Ou encore :

Il reste alors numériquement à définir la fonction , puis à partir d’une valeur connue, de de procéder par itérations en calculant jusqu’à ce que t soit supérieur ou égal à .

On pourra adapter pour arriver exactement sur .

On pourra par ailleurs ne renvoyer que la solution en , ou l’ensemble des solutions sur l’intervalle .

Illustration :

## 2° ordre

### Méthode

Soit l’équation :

Nous pourrions résoudre cette équation comme vu précédemment en introduisant un développement de Taylor à l’ordre 2 pour . Voici comment faire autrement.

Soit le vecteur .

On aura alors :

On discrétise à l’ordre 1 :

On a alors :

avec la fonction vectorielle

En effet :

On est alors ramené à la résolution d'une équation différentielle du 1° ordre (méthode détaillée au paragraphe précédent).

La solution Y obtenue contiendra les valeurs approchées de mais aussi de .

### Remarques

#### Concernant les array

Il est possible de tout programmer avec des listes, en particulier si vous utilisez odeint. Avec une fonction Euler explicite, cela dépendra de votre programmation, mais les array peuvent présenter l’avantage de permettent de calculer directement

Avec des listes, on fera la mise à jour ainsi :

* y **=** y **+** yp**\***dt si une équation d’ordre 1
* y **=** **[**y**[**i**]** **+** yp**[**i**]\***dt **for** i **in** range**(**len**(**y**))]** sinon

Quelques remarques :

* Par chance, la somme d’une liste et d’un array donne un array. Si est une liste, et que renvoie bien un array, tout se passera bien, l’inverse est faux car le produit ne passera pas (si , cas particulier qui fonctionne, et si est un entier, il y aura un problème !). Idéalement, bien faire deux fois des array
* ATTENTION : Nous l’avons déjà dit, mais un rappel est nécessaire je pense. Ajouter un array dans une liste puis modifier modifie dans aussi ! Dès que vous manipulerez des array en Euler, si vous écrivez « Y **+=** Yp**\***dt » au lieu de « Y **=** Y **+** Yp**\***dt », vous risquez d’avoir à la fin fois le même dans . Illustration de ce problème :

|  |  |
| --- | --- |
| **import** numpy **as** np  V **=** np**.**array**([**1**,**1**])**  L **=** **[]**  L**.**append**(**V**)**  **print(**L**)**  V **+=** 1  L**.**append**(**V**)**  **print(**L**)**  V **=** V **+** 1  L**.**append**(**V**)**  **print(**L**)** | **[**array**([**1**,** 1**])]**  **[**array**([**2**,** 2**]),** array**([**2**,** 2**])]**  **[**array**([**2**,** 2**]),** array**([**2**,** 2**]),** array**([**3**,** 3**])]** |

* Si vous créez un array d’entiers et qu’en plus, vous utilisez += pour ajouter un flottant, vous obtiendrez une erreur de type, la somme d’un int32 et d’un float64 ne fonctionnant pas…

|  |  |
| --- | --- |
| a **=** 10  b **=** 10  V **=** np**.**array**([**a**,**b**])**  V **+=** 0.1 | Cannot cast ufunc 'add' output from dtype('float64') to dtype('int32') with casting rule 'same\_kind' |

Solutions : V **=** V **+** 0.1 ou définir a et b en floant en ajoutant « .0 », ex a **=** 10.0

#### Conditions initiales atypiques

Lorsque les deux conditions initiales sont en position et , une résolution itérative partant de et d’une valeur arbitraire qui sera incrémentée peut permettra de trouver la solution qui respecte … (cf équation de chaînette).

### Exemples de mise en équation

#### Exemple 1 : Masse en chute libre

Soit une masse soumise à la gravité dont on veut déterminer les positions successives en chute libre sous l’action de la gravité et d’une force de frottement fluide. L’application du PFD donne :

On note donc et on a avec

On définit alors .

Et on trouvera le nouveau en utilisant :

Après résolution, sera la 1ère coordonnée de

#### Exemple 2 : Pendule

Soit un pendule assimilé à une masse ponctuelle en , en rotation avec le bâti en , en oscillations libres à partir d’une position initiale . On suppose qu'il n'est soumis à aucun frottement. L’application du TMD donne :

On note donc et on a avec

On définit alors

Et on trouvera le nouveau en utilisant :

Après résolution, sera la 1ère coordonnée de

# Systèmes d’équations différentielles du premier et du second ordre

Lorsque l’on a compris les principes précédents, il devient assez simple de résoudre des systèmes d’équations différentielles en « vectorisant » le problème.

Ci-contre, un exemple (Mines Ponts MP 2016) de modèle d’évolution d’une population face à une épidémie avec les nombres de personnes S (saine) , I (infectée), R (remise) et D (décédée).

## 1° ordre

Soit le système :

Il suffit de définir le vecteur :

On a alors :

On définit alors la fonction :

On a alors :

Puis de résoudre comme pour une équation différentielle du 1° ordre :

## 2° ordre

Soit le système :

Il suffit de définir le vecteur inconnu :

On a donc :

On définit alors la fonction :

On a alors :

Puis de résoudre comme pour une équation différentielle du 1° ordre :

# Modélisation par équations aux différences

A la différence des méthodes vues au paragraphe précédent pour la résolution d’équations simples de la forme ou , on va pouvoir ici résoudre tous types d’équations différentielles ☺. Au concours, c’est plus généralement ce type de fonctions simples qui sont proposées.

La méthode que nous allons aborder ici est toutefois utilisable aussi pour les fonctions ci-dessus !

## Contexte

La méthode des équations aux différences est une méthode plus puissante que ce que nous venons de voir précédemment. Elle permet certes de traiter les équations vues précédemment du type , mais elle va nous permettre de résoudre des équations de plusieurs variables comme l’équation de la chaleur 1D ci-dessous, l’équation d’onde etc., non solvables avec la méthode classique :

## Méthode de résolution

Supposons que est une entrée connue et une sortie recherchée.

Pour déterminer, à tout temps , on exprimer chacune des dérivées avec les méthodes vues précédemment (Euler, Taylor, différences finies) afin d’obtenir une relation entre divers états de l’entrée et de la sortie et on exprime ensuite à l’aide de cette équation la valeur recherchée en fonction des valeurs connues de l’entrée et des valeurs précédentes de la sortie.

Autrement dit, une équation aux différences est de la forme :

On veillera à traiter correctement les conditions initiales.

## Exemple 1 : Chute libre

### Equation récurrente

Reprenons l’exemple de la masse en chute libre étudiée avec la méthode précédente, dont l’équation était :

On choisit d’exprimer les dérivées première et seconde de à l’aide d’un schéma Euler Explicite pour la dérivée première et Euler centré pour la dérivée seconde :

On obtient donc l’équation aux différences suivante :

On obtient alors la position à tout instant connaissant les positions antérieures :

### Prise en compte des conditions initiales

Attention, à l’instant initial , on a .

Pour calculer la valeur suivante, il faut et qui n’existe pas !

Deux solutions (qui en réalité sont les mêmes, la première étant un cas particulier):

* La vitesse initiale est nulle, alors on sait qu’avant , et on peut donc affecter la valeur en égale à la valeur en
* La vitesse initiale est non nulle, on utilise la formule d’Euler implicite :

Ce qui permet d’imposer (en traitant le cas particulier du premier pas de temps sous Python):

### Remarque : Regroupement de termes dépendant de z dans le second membre

Il serait possible de résoudre l’équation en intégrant la force de frottement fluide en second membre :

On pose

A chaque instant, on calcule donc la valeur de que l’on appellera . Pour calculer à chaque étape, on utilise Euler implicite (utilisation de valeurs connues) à l’ordre 1 :

On peut donc trouver une nouvelle équation aux différences :

Cela peut être très utile en présence de termes non linéaires, par exemple une force en carré de la vitesse (frottements fluides réalistes…).

## Exemple 2 : Pendule

### Equation récurrente

Reprenons l’exemple du pendule étudié avec la méthode précédente, dont l’équation était :

On choisit d’exprimer la dérivée seconde de à l’aide d’un schéma Euler centré :

On obtient donc l’équation aux différences suivante :

On obtient alors l'angle à tout instant connaissant les angles antérieurs :

### Prise en compte des conditions initiales

Attention, à l’instant initial , on a .

Pour calculer la valeur suivante, il faut et . Deux solutions :

* La vitesse initiale est nulle, alors on sait qu’avant , et on peut donc affecter la valeur en égale à la valeur en
* La vitesse initiale est non nulle, on a alors :

On suppose qu’on a aussi :

On impose alors :

# Résolution d’équations et précision

## Précision sur la solution

Comme nous l’avons dit, nos approximations de dérivées ne seraient exactes que si le pas de temps était infiniment nul. Ce n’est évidemment pas le cas, et plus le pas de temps est petit, plus les temps de calcul deviennent élevés…

Il faut donc être conscient que la méthode de résolution par discrétisations d’Euler induit des erreurs ! Et celles-ci se cumulent.

Soit la fonction .

Voici le résultat de la résolution de l’équation par la méthode d’Euler explicite () et implicite ( possible car calculable au temps , la formule étant connue) décrite au paragraphe précédent entre 0 et 10 pour des pas de temps différents :

|  |  |
| --- | --- |
| Euler Explicite | Euler Implicite |
|  |  |

On remarque bien que plus le pas de temps est petit, plus la courbe tend vers la solution réelle. On peut noter que pour le pas de temps de 9, la première valeur calculée est très fausse. La suivante sera calculée à partir de cette fausse valeur… L’erreur se cumule ! Voici une illustration de ce qu’il se passe en explicite :

La méthode d’Euler est d’autant plus précise qu’il y a de points dans les zones à forte variation. Ainsi, si la solution est une droite, la méthode d’Euler ne fait aucune erreur…

## Précision sur le temps

Ceci est une simple remarque mais cela a son importance dans les simulations sur un grand nombre de pas de temps.

Lorsque le pas de temps n’est pas un nombre décimal fini, il existe une différence entre le calcul du temps avec les deux méthodes suivantes :

* Calcul absolu : t **=** t0 **+** i **\*** dt
* Calcul relatif : t **=** t **+** dt

Illustration :

|  |  |
| --- | --- |
| dt **=** 1**/**3  T **=** 1e8  t0 **=** 0  Liste\_T1 **=** **[]**  Liste\_T2 **=** **[]**  i **=** 0  t1 **=** t0  Pas **=** int**(**T**/**dt**)**  **while** i **<** Pas**:**  i **+=** 1  t1 **=** t1 **+** dt  t2 **=** t0 **+** i **\*** dt  Liste\_T1**.**append**(**t1**)**  Liste\_T2**.**append**(**t2**)**  Tf1 **=** Liste\_T1**[-**1**]**  Tf2 **=** Liste\_T2**[-**1**]**  DTf **=** Tf2 **-** Tf1  **print(**Tf1**)**  **print(**Tf2**)**  **print(**DTf**)** |  |

Lorsque les temps de simulation deviennent grands, le cumul des erreurs d’arrondi en calcul relatif peut conduire à des imprécisions non négligeables.

Lequel choisir ? Cela dépend du contexte… Il faut surtout être conscient du problème !

# Fonction de résolution intégrée à Python

|  |  |  |
| --- | --- | --- |
| 1° ordre - 1 équation | | |
|  | **def** F**(**y**,**t**):**  k**,**g**,**m **=** 0.001**,**9.81**,**0.1  dy **=** **(**k**/**m**)\*(**y**\*\***2**)-**g  **return** dy | |
| Y0 **=** 0 | |
| 1° ordre - 2 équations | | |
|  | **def** F**(**Y**,**t**):**  a**,**b**,**c**,**d **=** 3**,**1**,**1**,**2  H**,**A **=** Y  dH **=** a**\***H **-** b**\***A**\***H  dA **=** **-**c**\***A **+** d**\***A**\***H  dY **=** **[**dH**,**dA**]**  **return** dY | |
| H0**,**A0 **=** 10.0**,**1.0  Y0 **=** **[**H0**,**A0**]** | |
| 2° ordre - 1 équation | | |
|  | **def** f**(**Y**,**t**):**  m**,**R**,**k**,**g **=** 0.1**,**0.1**,**0.001**,**9.81  J **=** m**\***R**\*\***2  y**,**yp **=** Y  ypp **=** **(**1**/**J**)\*(-**k**\***yp**-**R**\***m**\***g**\***sin**(**y**))**  **return** ypp  **def** F**(**V**,**t**):**  y**,**yp **=** V  ypp **=** f**(**V**,**t**)**  dY **=** **[**yp**,**ypp**]**  **return** dY | |
| **from** math **import** pi**,**sin  y0**,**yp0 **=** 45**\***pi**/**180**,**0  Y0 **=** **[**y0**,**yp0**]** | |
| 2° ordre - 2 équations | | |
|  | **def** f**(**Y**,**t**):**  m**,**B**,**g **=** 1**,**1**,**9.81  x**,**xp**,**y**,**yp **=** Y  Vit **=** **(**xp**\*\***2**+**yp**\*\***2**)\*\*(**0.5**)**  xpp **=** **(**1**/**m**)\*(-**B**\***Vit**\***xp**)**  ypp **=** **(**1**/**m**)\*(-**g**-**B**\***Vit**\***yp**)**  **return** xpp**,**ypp  **def** F**(**Y**,**t**):**  x**,**xp**,**y**,**yp **=** Y  xpp**,**ypp **=** f**(**Y**,**t**)**  dY **=** **[**xp**,**xpp**,**yp**,**ypp**]**  **return** dY | |
| x0**,**y0 **=** 0**,**0  xp0**,**yp0 **=** 10**,**10  Y0 **=** **[**x0**,**xp0**,**y0**,**yp0**]** | |
| Code de résolution | |
| t0**,**t1 **=** 0**,**10  N **=** 1000  dt **=** **(**t1**-**t0**)/(**N**-**1**)**  Lt **=** **[**t0**+**i**\***dt **for** i **in** range**(**N**)]**  **from** scipy**.**integrate **import** odeint  Sol **=** odeint**(**F**,**Y0**,**Lt**)**  **print(**Sol**)** | |