# Analyse Numérique II

par KHIAT Soufiane

Date de publication :

Dernière mise à jour :

Dans ce tutorial nous aurons nos premiers visuel, nos premières simulations physique. Nous ferons notre premiers petit pas dans la simulation physique.

# 1 - Introduction

Précédement nous avons vu comment interpoler et approximer des fonctions rien de trépidant vous allez me dire, En effet nous n'avons pas encore lancer la moindre particule pour "tester" la gravitation Newtonnienne. Et bien c'est l'objet de ce tutorial. Nous allons voir comment simuler numériquement les phénomènes physiques de bases.
Les objectifs de tutorials. Pouvoir simuler toute les équations différentiels linéaires. Et trouver une méthode générique nous permettant la plus grande flexibilité. Nous vérons la théorie mais aussi une implémentation dans un langage Orienté Objet le C++. Mais sachez que si c'est possible en C++ c'est possible dans n'importe quel langage Objet et si c'est possible dans un langage objet c'est aussi possible dans un langage non Orienté Objet et non objet.

## 2 - Dérivation numérique

représente-t-elle ? La dérivée représente la variation d'une fonction instantané : vers où tend la fonction. Ou si l'on préfère plus formellement (comme on nous l'a appris au lycée) : la dérivée la pente (le coéficient) directeur de la courbe. Ou :

Toute fonction n'admet pas de dérivée explicite. Nous savons que la dérivée de x*x = 2*x. Nous le savons c'est démontrable. Nous savons aussi que la dérivée de la position est la vitesse et la dérivée de la vitesse est l'accélération donc la dérivée seconde de la position est l'accélération. Mais lorsque nous faisons des simulations physique nous travaillons avec des données discréditisées nous n'avons rien de continue. Et le but d'une simulation et de trouvé une fonction et nous ne pouvons pas dérivée une fonction que nous n'avons pas. C'est ce que nous allons voir.

Il faut connaitre le "famille" de formule la plus importante pour faire des dérivées numérique : Les formules de Taylor ! Nous n'avons pas besoin de toutes les connaitres en générale Une suffit :

Qu'allons nous faire de cette formule. Et bien il faut savoir qu'elle est à la base de la majorité des formules de dérivation numérique. Avec plusieurs petits "trifouillages" mathématiques de cette formule nous pouvons trouvé n'importe quel formule de dérivée n-ième de n'importe qu'elle ordre.
Je ne vais pas vous faires les démonstrations (Si cela vous intereses calculer f(h) et f(-h) avec les formules de Taylor d'ordre K et soustraier les 2). Elles ne sont pas vraiment compliquer mais ce n'est pas un tutorial de math donc je vais listées quelques formules qui me semblent indispensables. Notons u(t) la valeur en ordonnée et t l'abscisse ; puis h la variation de l'abscisse. Il existe 3 types de dérivée numérique. Les dérivées (ou différence) centrées : qui pour calculer la dérivée en un point utilise la fonction avec des abscices amonts et avals au point calculer. Les dérivées amonts qui utilise que des informations amonts. Et les dérivées avals qui utilise juste des informations aval (beaucoup) moins utilisées puisque le but d'un calcule numérique est de déterminé une fonction donc les informations avals ne sont pas accésible.
Dérivées centrée d'ordre O(h^2) :

Différences centrées d'ordre O(h^4) :

Différences amont d'ordre O(h^2) :

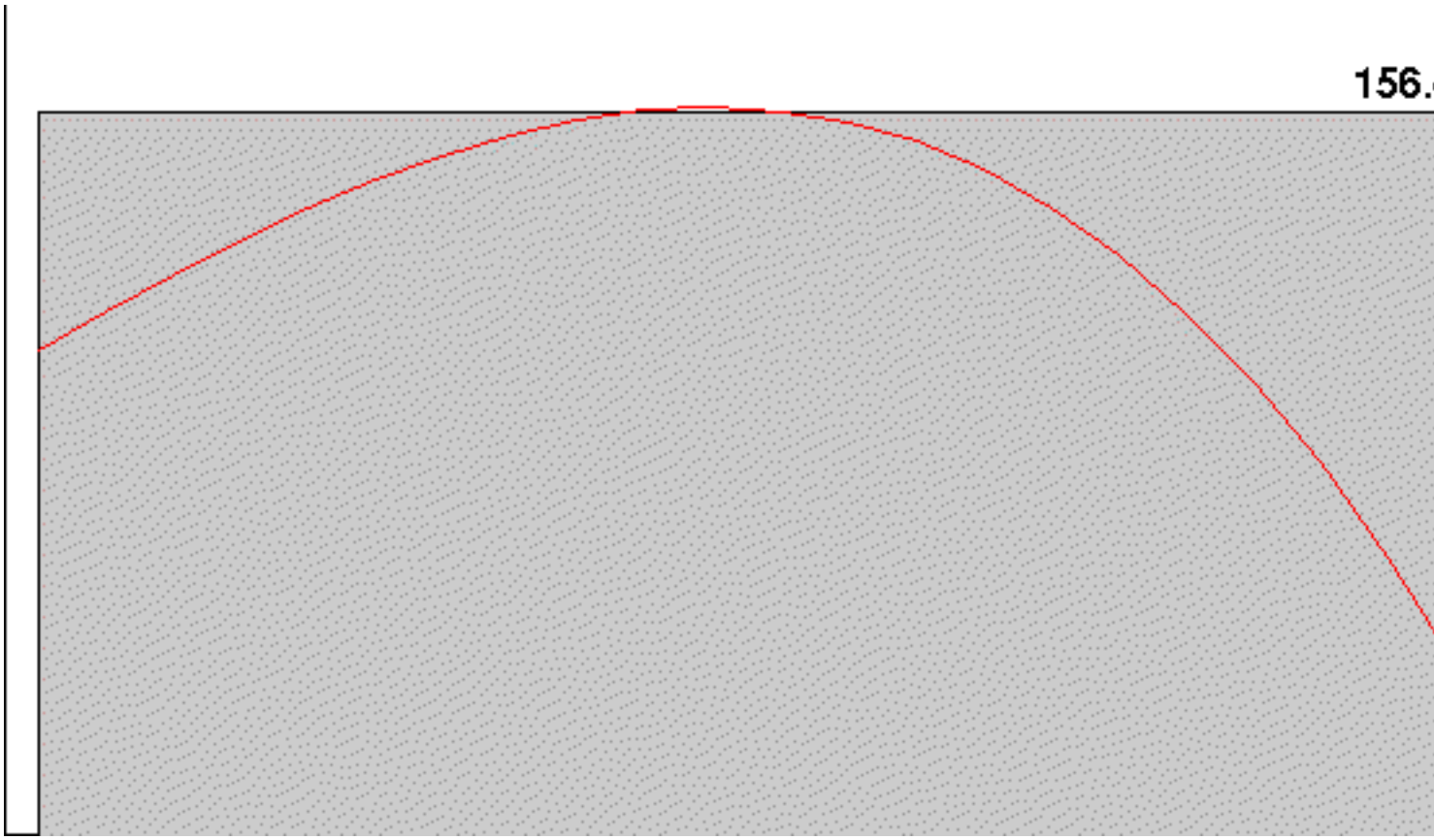
Comme vous l'avez remarquer (et comme vous vous y attendiez) plus on veut être précis plus c'est couteux en calcule donc il faudra faire un choix entre la précision et la vitesse de calcule. Mais il y a d'autre "problème" en général dans les calcules scientifique nous travaillons souvent avec des Equations différentiels donc en général nous disposons

de la dérivée et c'est la fonction que nous essayé de trouver. Donc les intégrales numériques sont beaucoup plus utiles ; ce que nous allons voir.

# 3 - Integration numérique

Souvenons nous de la première fois que nous avons vu les intégrales. Les professeurs ont introduit la notion d'intégrale par la notion d'aire. Et nous on parler de l'intégrale de Riemann. Nous noterons I l'intégrale :

Et nous découpons plein de morceau de la fonction :



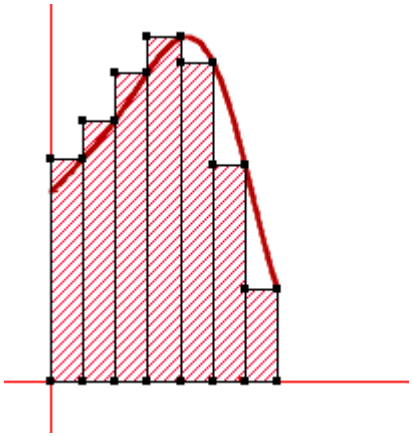

Donc l'aire de la fonction est l'encadrement de la somme de l'air des récrangles suppérieur par ceux inférieux :

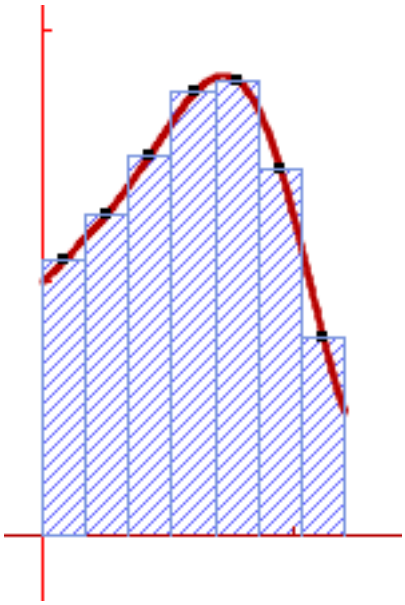
Et l'aire donc l'intégrale est la limite lorsque n tend vers +infini :

Mais comme vous l'avez devinez voilà la première méthode pour calculer l'intégrale d'une fonction.

## 3.1 - Méthode des réctanges

La aussi comme les dérivées numérique il existe 3 méthodes : amont, aval et centré :





Prenons un exemple pour comparée les méthodes et pour que ce soit plus intéréssant prenons une fonction qui n'admet pas de primitive (e^(-x^2)) et cherchons :

Avec une méthode précise nous obtenons : 0.28554313093994
Avec la méthode des réctanges amont : 0.3894003915357 (36,4% d'erreur)
Avec la méthode des réctanges aval : 0.18393972058572 (35,6% d'erreur)

La variation par rapport à la valeur "très proche de la valeur exact" est trop grande pour que les valeurs d'erreur est un sens ici on voit simplement que la méthode des réctangles n'est pas très bonne.
Maintenant si nous calculons l'intégrale à partir de la moyenne des valeurs amont et aval que ce passe-t-il ?
Et bien nous obtenons un trapèze puisque la surface d'un trapèze est la moyenne de la surface de 2 rectangles.
TODO : dessin

## 3.2 - Méthode des trapèzes

Donc si nous fesons la moyenne des rectanges amont et aval nous obtenons :

*Attention ici h = b-a*

Avec la méthode des trapèzes : 0.28667005606071 (0.4% d'erreur). C'est beaucoup mieux non ? Ici nous avons de la "chance" puisque e^(-x^2) "est assez affine" entre 1/2 et 1 d'où le résultat. Comment procéderons nous avec des courbes avec un grosse variation ou si nous cherchons l'intégrale sur dans minima ou maxima local ?

## 3.3 - Méthode de Simpson

La méthode de Simpson généralise la méthode des trapèzes. Puisqu'au lieu d'utilisé un polynome de dégré 1 pour intégré (une fonction affine) les fonctions intégré numériquement sont remplacées par un polynome de degré 2 une parabole avec les points f(x), f(x+h) et f(x+2h). La méthode de Simpson s'écrit comme suit :

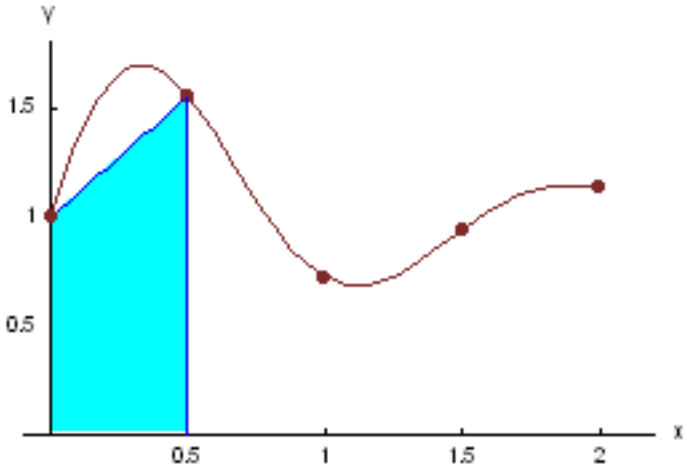
Cherchons notre intégrale avec la méthode de Simpson : 0.57096858719442 (0.9996% d'erreur). Pourquoi une méthode d'ordre supérieur (O(h^4)) est moins précise ? Et bien pour la même raison qui nous à aidé lors de l'utilisation de la méthode des trapèzes. La fonction est relativement linéaire pour des fonctions assez concave ou convexe la méthode de Simpson sera meilleur (conscient que nous manipulons des notions abstraite comme "assez concave", "relativement linaire", ... Mais nous ne pouvons pas faire autrement il n'y a pas de théorème explicite qui nous dira cette méthode dans ce cas est meilleurs qu'un autre).

## 3.4 - Méthode de Newton-Côtes

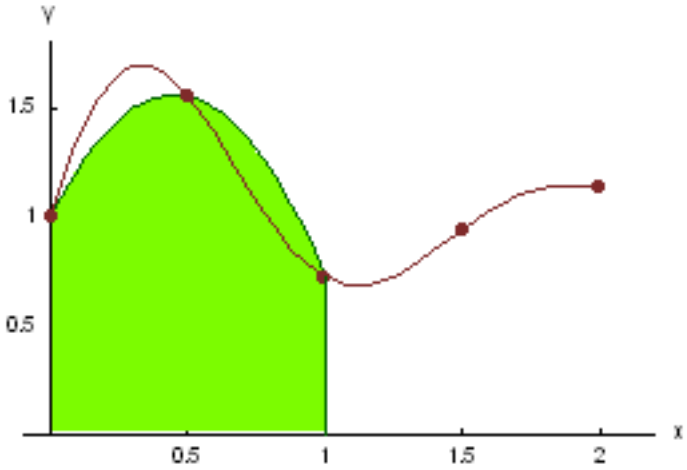
Dans certain cas un polynome de degré 2 n'est pas suffisant c'est pourquoi la méthode de Newton-Côtes généralise les méthode des trapèzes et Simpson en remplaçant la fonction à intégrer par un polynome de degré n. La méthode s'énonce comme suit :

Cette méthode est moins commode que les autres mais ne vous inquiétez pas d'autre on réfléchit pour nous et on trouver plusieurs chose intéréssante. Si nous prenons f(x) = x^k pour k = 0, 1, ... n on optient le système linéaire

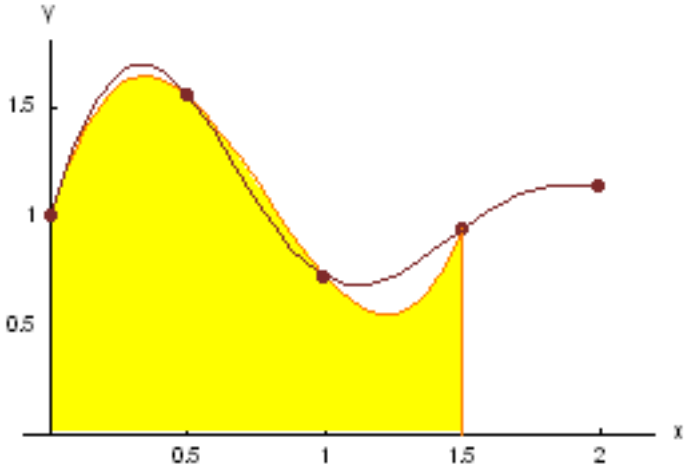
suivant :
Le détèriminant de ce système est celui de Vandermonde qui vaut (x0 - x1)(x1 - x2)...(xn - x0). Si les points sont régulièrements répartie alors pour n = 1 nous avons la méthode des trapèzes :



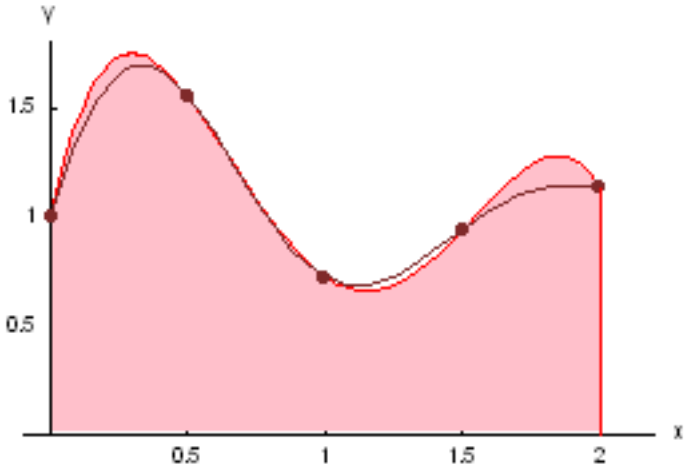

Pour n = 2 la méthode de Simpson :




Pour n = 3 une méthode sans véritable nom mais certain l'appel Simpson 3/8:




Pour n = 4 la méthode de Boole-Villarceau :




Après avous de lancer les n et de calculer les ai. De plus si vous ne faites pas de méthode pas à pas vous pouvez interpoler tous vos points avec un polynome (Lagrande, Hermite, ...) ou faire une approximation (Padé, ...)...
Aussi vous pouvez découpez votre fonction régulièrement ou non en abscisse pour avoir une méthode récursive où vous y appliquerais une des méthodes précédèments présenter.

# 4 - Résolution d'Equation Différentielle

Enfin nous entrons dans la partie la plus intéréssante. Celle qui vous permèteras de lancer des corps ponctuelle ou non de faire de la simulation d'eau, de corps mou, ... Mais avant tous cela il avoir quelque base. Connaître un minimum de méthode pour faire ce genre de résolution. Toute les méthodes présenté ici permètterons de résoudre des problème dit de Cauchy :

Et bien commençons. Dans cette partie je vais vous montrez que quelque méthode. Il y en aura un "grand" nombre d'implémenté dans le code source associer à ce tuto.

## 4.1 - Méthodes d'Euler

Les méthodes d'Euler sont les moins couteuses en calcule et assez bonne (sauf bien sur Euler Explicite). Comme la méthode d'Euler explicite est a ne surtout pas utiliser mais a absolument connaitre ! Donc voilà l'expression de la méthode d'Euler explicite et implicite :

Il existe d'autre méthode d'Euler plus exotique comme Euler-Cauchy (ou méthode de Heun) :

Et bien sur ma préférer que j'appel celle dont je connais pas le nom que je nomme Euler++ (ou Euler Plus) :

Bon je ne vais pas vous exposez toutes les méthodes qui porte le nom du Grand Euler (Euler semi-explicite (ou crank nicholson), ...) Si cela vous interesses Internet regorge de méthode, ... Sachez qu'à chaque fois que je découvre une nouvelle méthode elle sera implémenté dans la source.

## 4.2 - Méthodes de Runge Kutta

Comment parler des méthodes de résolution numérique sans parler des méthodes de Runge-Kutta. C'est une méthode correctif itérative c'est-à-dire que l'on fait une première apprixamation de la fonction, une deuxième, ... Et ainsi de suite...
Par exemple Runge-Kutta d'ordre 1 est la méthode d'Euler explicite. La méthode de Runge Kutta Explicite d'ordre 2 est la méthode de Heun (ou Euler-Cauchy). La méthode la plus connu est la Runge-Kunta d'ordre 4 (ou RK4). Cet méthode se présente comme suit :

## 4.3 - Méthodes de Gear

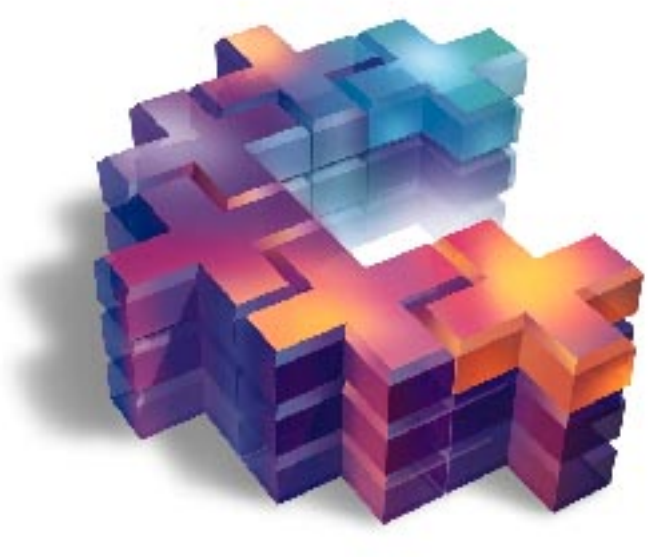
Nous avons vu les méthodes direct, les méthodes récursive et là nous allons voir les méthodes à pas multiples. En effet je ne vais pas vous parlez de toute les méthodes existances Google le fera mieux que moi. Qu'est ce qu'une méthode à pas multiple ? Comme nous simulons des phénomènes au cour du temps notre solutions trouvé a en généralement tendance à divergé de la solution exacte. Donc les méthodes à pas multiple pour palier ce problème, au lieu de considérer seulement la valeur du pas de temps d'avant et de la différentiel. Nous considérerons plusieurs valeurs au pas multiple d'avant.
La méthode de Gear d'ordre 1 corréspond à la méthode d'Euler implicite. Gear d'ordre 2 se formule comme suit :

Comme vous pouvez le voir pour connaitre u(t+dt) nous avons besoin de connaitre u(t) mais aussi u(t-dt). Mais pour Gear d'ordre 3 nous avons besoin des mêmes informations mais aussi de u(t-2dt). Ainsi de suite...

Pour plus d'information sur les méthodes de résolution d'équation différentielle vous pourrez voir la source associer à ce tutorial. A chaque fois que je découvre une méthode je l'implémente.

# 5 - Implémentation

## 5.1 - Choix des outils




Le but de ce tutorial est de s'adresse au maximum de personne donc un code multiplaforme s'est imposé de lui même. Donc un
choix à faire Qt ou wxWidgets (je sais qu'il en existe d'autre mais ce sont les plus utilisées). Avec Qt il existe une subtilité avec la licence alors j'utiliserais wxWidgets que j'ai l'habitude d'utiliser. Et maitenant le choix du langage. En effet si j'avais choisit le Java je n'aurais pas eu le problème du mutliplateforme puisque la machine virtuelle aurait fait tout le travail. Mais j'ai l'habitude d'utilise le C++. Donc surtout par attrait et aussi la notion d'objet qu'offre le C++ j'utiliserais ce langage. Pourquoi l'objet ? Et bien il nous aidera dans la modélisation des problèmes et nous permetteras de faire des surchouches pour plus de gestion de donnée abstraite (comme la notion d'interface, virtualisation, ...). Et comme j'ai voulu être multiplateforme l'OpenGL s'est imposé de lui même face au DirectX. Ceci étant dit au travail.

## 5.2 - Conception

La conception se divise en 2 parties. La première assez complexe si on n'a pas l'habitude avec les notions d'objet. Qui consiste à crée une classe **vector** totalement générique. Puisque les problèmes que nous devrons résoudre serons en 1D, 2D, 3D, 4D, ... Voir plus et aussi les éléments de la classe (x0, x1, x2, x3, ..., xn) serons de type générique puisque à long ou moyen terme nous implémenterons une classe de flottant à taille paramétrable.
Et la seconde partie de la conception un peu plus simple consiste à faire une interface pour les différents "solveur" d'EDO (Euler, RK2, RK4, Gear, Adams, ...) cela nous permetteras de de manipuler toutes ces méthodes de résolution indépendament de la méthode. Ainsi l'ajout de méthode sera une simple dérivation de classe. Et aussi crée un factory pour la création dynamique de ces "solveurs" vu le nombre de "solveur" une simple factory serait bien trop lourde ; donc nous implémenterons une Abstract Factory (pour plus de détail voir les tutoriaux internet et surtout sur developpez.com).

### 5.2.1 - Notre classe vecteur

Nous ne ferons pas une classe vecteur habituelle. Mais le but est d'avoir les calculs les plus rapides en évitant au maximum les variables temporaire. Donc d'avoir le maximum d'opération fait à la compilation. Un des principes de la méta programmation pour plus de détail voir le tutorial de Laurant Gomila (loulou) "**La meta-programmation en C++**".
Regardons un peut le détail du code.