Analyse Numérique I

par KHIAT Soufiane

Date de publication :

Dernière mise à jour :

Ce tutorial portera sur la manière dont nous traiterons les résultats d'une simulation physique. Interpolation Et approximation numérique. Mais aussi de la mise en garde sur l'outil que nous utilisons, les processeurs.

1 - Introduction

1.1 - Pourquoi ces tutoriaux ?

Le but de ces tutoriaux est de pouvoir faire un moteur physique qui n'a pas pour but d'être temps réel, au contraire. Vous allez Peut-être me demander pourquoi où clamer de l'inutité d'un tel programme. Puisque le temps réel à une application direct dans les jeux vidéos, ... Et bien sachez que la simulation physique a aussi des applications directs. Dans l'industrie :

- Simulation d'un Crash Test de la Renault Laguna
- Simulation de l'écoulement de l'air autour du profil de l'aile d'un F-15 eagle
- Simulation de l'écoulement de fluide à Amsterdam en cas d'inondation
- Simulation de la propagation des ondes GSM en milieu urbain
- Simulation du flux thermique sur les refroidisseurs du Chipset de votre carte mère
- Simulation des orbitals autour de l'atome He_3
- ...

Et bien toutes ces simulations nous pourrons les faires (enfin pas toute puisque certain nécessite des méthodes de résolution rien que pour elle).
De nombreux logiciel permettent de faire chacun de ces exemples moyennant quelque ? ? (en général plusieurs milliers). Donc voilà la raison pour laquelle je me lance dans cette série de tutoriaux. Je voudrais démocratiser ce genre de logiciel. De plus c'est toujours Fun de savoir ce qui ce passe lorsque l'on jette un Cube en bois remplis d'un liquide visque en le tournant sur lui même non ? Parfois il faudra s'accocher niveau math mais cela vaut le coup.

1.2 - Prérequis

Nous allons voir la base de toute simulation numérique. Je pars sur le présupposé soulignant que, vous, lecteur avez déjà une base mathématique. De plus, je ne vais pas démontrer toutes les formules utilisées puisque ce n'est pas le but du tutorial. Le but ici est de vous donner le maximum d'outils afin que vous puissiez faire tout ce dont vous avez besoin pour vos simulations physiques. Ce premier chapitre traitera de la base que vous devez avoir avant de mettre un seul pied dans la physique. Les résolutions numériques (de EDO : equations differentielle ordinaire), vous trouverez peut être cela fastidieux mais sachez que ce passage est indispensable. Le monde de la résolution numérique ce divise en 3 types de méthodes de résolutions : Les méthodes à différentielles finies, les méthodes à volumes finis, les méthodes des éléments finis. Nous allons nous penché sur la première dans ce tutorial. Les autres méthodes ferons parties d'un autre tutorial. Pour résoudre les EDO issus des équations aux dérivées partielles (EDP) nous aurons besoin de méthodes de résolutions numériques. Et bien commençons...

1.3 - Exemples

Prenons un exemple très simple : je lache un objet ponctuelle de 10 kg à 100 mètres du sol en négligeant les frottements de l'air en considérant l'accélération terrestre à $10m.s^{-2}$ (c'est une approximation accéptable) nous cherchons une fonction qui nous donnera la position de l'objet en fonction du temps. Ceux parmis vous qui on fait un peu de physique pourrons résoudre ce problème analytiquement en partant de l'accélération constante de la terre et en l'intégrant :
$a(t) = 10$ (quelque soit t)
donc :
$v(t) = 10t+k0$ (en prenant une vitesse initiale nul $v(0) = 0$ alors $k0 = 0$)
donc :
$v(t) = 10t$
$z(t) = 1/2*10t^2+k1$ (en prenant le repère initiale à l'endroit où l'on a laché l'objet alors $z(0) = 0$ donc $k1 = 0$)
donc :

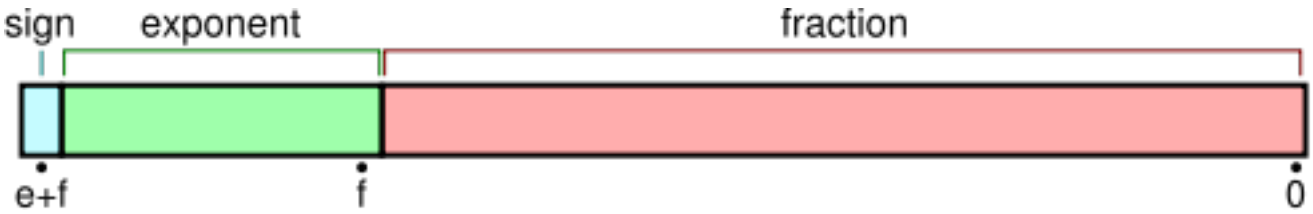
$z(t) = 5t^2$
Dans un exemple comme celui là il serait inutile de cherche des méthodes de résolution autre que celle montré ci-dessus
Maintenant prenons le même problème en y ajoutant quelques données. Le corps n'est plus ponctuelle mais torique il ne tombe plus il tourne sur lui même sur son diamètre principale ; de plus nous lui avons donnée une vitesse initiale. Et par dessus le marcher il n'est plus sur terre mais dans l'espace et Jupiter, Saturne et Uranus influs sur sont mouvement (Et pour simplifier nous négligerons les plans ecliptique nous sommes donc en 2D). Et je vous mets au défie de trouver analytiquement x(t), z(t). Donc à nous de réfléchir bien pauser, le problème faire en sorte qu'il soit stable, qu'il soit rigide au perturbation, ...
Nous allons voir tous celà.

1.4 - Arithmétique flottate

Il ne faut pas oublier les outils qui nous permettent de tel résolution numérique. L'ordinateur et plus précisément le processeur. En négligeant la connaissance du fonctionnement du processeur nous arriverons surement à des catastrophes numérique. Voici quelque exemple qui je l'espère vous ferons prendre concience que les résolutions numériques cela ne s'improvise pas.

- Les bugs coûteux (missile Patriot 1991) : Les missiles patriot sont des missiles anti-missiles de fabrication américaine. Le 25 février 1991, l'un d'eux rata le scud (missile irakien d'origine soviétique) qu'il était censé détruire. Une analyse du problème révéla que le logiciel de guidage des Patriot reposait sur un calcul incrémental sommant des dixièmes de secondes pour déterminer le temps courant. La valeur 1/10 étant incodable en binaire, ce calcul est approximatif est entraîne une dérive de l'estimation du temps. Cette dérive est de 20% après quatre heures, 50% après huit heures et plus de 100% après vingts heures. Les missiles qui échouèrent le 25 février étaient en fonction depuis plus de cent heures. Coût: 28 morts.
- Les bugs coûteux (Ariane V 1996) : L'explosion du premier exemplaire de la fusée Ariane V en 1996 était due à un bug dans le logiciel de stabilisation. 37s après le lancement, une conversion d'une valeur flottante codée sur 64 bits en entier codé sur 16 bit provoqua le crash de ce logiciel écrit en Ada. Coût: 800 millions ?
- ...

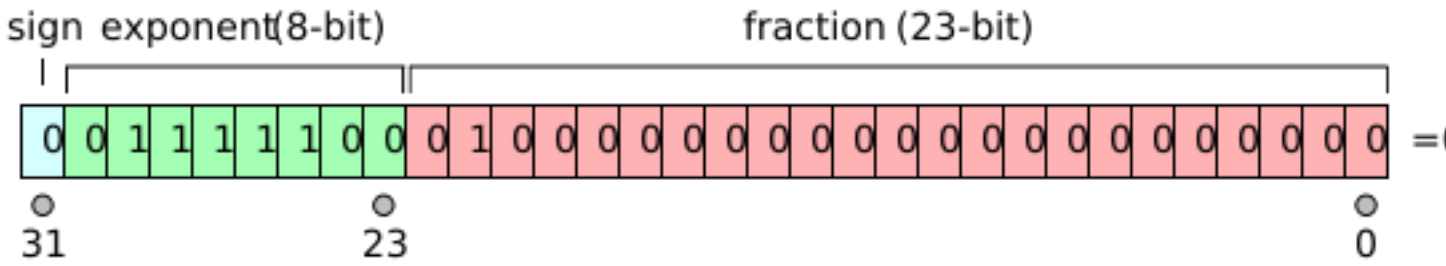
Tout d'abord comment un ordinateur représente un nombre entier ?
Et bien comme vous le savez l'ordinateur comprend seulement 2 choses 1 et 0. Présence ou Abscence de ce "quelque-chose" je vais m'arrêté là nous allons pas entrer dans des considérations HardWare ce n'est pas un cour sur les mémoires. Chaque nombre décimaux est convertie en binaire sous forme de somme de puissance de 2.
exemple :
$42 = 101010 = 1*2^5 + 0*2^4 + 1*2^3 + 0*2^2 + 1*2^1 + 0*2^0 = 32 + 8 + 2$
Ceci est un exemple sur 6 bits. Essayons d'écrire 255 combiens nous faudrait-il de bit ? Et bien 8 ; un octet (un byte) 255 = 1111 1111 en binaire donc il peut être stoquer dans un unsigned char (type du C/C++ 1 octet). donc **unsigned char** b = 255 ; nous avons aucun problème mais si nous faisons b++. Et bien nous aurons un problème bien connu l'overflow. Et inverssement si nous avons **unsigned char** b = 0; puis nous faisons b--; Nous serons face à un lowerflow. Voilà un des problèmes auquel nous pourrions être confronté en résolution numérique.
Mais il est rare que nous travaillons avec des entiers en simulation physique regardons la représentation des nombres flottants dans un ordinteur.
IEEE 754
Mais quel est cette Norme IEEE-754 ? IEEE-754 est réponse à la question de comment représenter des valeurs relatives avec seulement des valeurs entières en maximisant l'étendu de l'ensemble tout en minimisant l'espace occuper (le nombre de bit).
Et bien l'IEEE-754 simple précision :
$(-1)^S*M*2^{E-127}$ S : 1 bit
M : 23 bits (la fraction sur l'illustration)
E : 8 bits



Nous passerons sur les nombres dénormaliser en gros ce qu'il faut retenir nous pouvons représenter plus de nombre proche de 0 que les autres.
Il existe des exceptions à cette exception :

- +inf
- -inf
- NaN (Not a Number) bizarre que dans la norme de représentation de nombre il existe des "pas nombres"
- les deux 0 : 0^{+}, 0^{-}

Exemple :



Bref c'est joli comme pour les Entiers Naturels on est limité dans l'espace. Donc lorsque l'on additionne 2 nombres où à forcement une perte d'information exemple :
91 234,5 + 41,2345 = 91 285,7345
Ca c'est ce que nous devrions avoir. Mais voilà ce que nous avons :
91 234,5 + 41,2345 = 91 285,7
D'où viens cette perte d'information et bien dans la partie fraction (en 23 bits aussi appelé Mantisse). Nous avons 23 chiffres significatifs binaires alors lorsque l'on dépasse c'est tronqué ou arrondit selon les architectures.
Il est aussi possible que 1000 + 0.00000001 = 1000 !
Voilà d'arrondis en arrondis lors d'algorithme de calcul récursif on peut vraiment s'éloigner de la solution.
Mais les erreurs ne viennent pas seulement des erreurs de troncatures et d'arrondis.

1.5 - Opération instable

Nous savons que nous pouvous avoir des surprises lorsque nous faisons des opérations même simple. Ne me croyez pas sur parole essaye par vous même :

```
#include <iostream>

using namespace std;

int main()
{
 float a = 0.1f;
 cout.precision(42);
 for (int i = 0; i < 10; i++)
  cout << static_cast<float>(i)*a << endl;
    return 0;
}
```

Résultat de ce petit programme :

```
0
0.10000000149011612
0.20000000298023224
0.30000001192092896
0.40000000596046448
0.5
0.60000002384185791
0.69999998807907104
0.80000001192092896
0.90000003576278687
```

Pourquoi ces résultats étrange. Et bien il faut savoir que la multiplication est fait sur un champ de bit à décalage donc c'est une opération itérative donc sujet au erreur d'approximation.

Autre exemple étrange on cherche à trouvé le nombre 1. Nous calculerons dans une boucle $10^{k}*10^{-k}$:

```
#include <iostream>
#include <fstream>

using namespace std;

float my_pow(float a, float b)
{
 float c = a;
 while(b--)
  c *= a;
 return c;
}

int main()
{
 float a = 0.1f;
 cout.precision(42);
 for (int i = 0; i < 100; i++)

 cout << my_pow(10.0f, static_cast<float>(i)) * 1.0f/static_cast<float>(my_pow(10.0f, static_cast<float>(i))) <<
 endl;
    return 0;
}
```

Résultat :

```
1
...
1
0.99999994039535522
1
1
0.99999994039535522
1
...
1
0.99999994039535522
1
0.99999994039535522
1.#INF
...
1.#INF
```

Puis si l'on cherche les puissances de 10 :

```
10
```

```
100
1000
10000
100000
1000000
10000000
100000000
1000000000
10000000000
99999997952
999999995904
9999999827968
100000000376832
999999986991104
10000000272564224
99999998430674944
999999984306749440
9999999980506447900
100000002004087730000
1000000020040877300000
9999999778196308400000
99999997781963084000000
10000000138484279000000000
99999995620235262000000000
100000002537764290000000000
999999988484154750000000000
9999999442119689800000000000
100000001504746620000000000000
1000000015047466200000000000000
99999998482432073000000000000000
100000003318135350000000000000000
999999994495727290000000000000000
9999999790214768000000000000000000
100000004091847880000000000000000000
999999961690316250000000000000000000
9999999933815812500000000000000000000
99999996802856925000000000000000000000
1.#INF
...
1.#INF
```

Et bien je ne vais pas vous faire une liste exaustive des problèmes que l'on peu rencontré. Entrons dans le vif du sujet.

2 - Interpolations

Comme nous travaillons sur un ordinateur rien n'est continue. Alors lorsque nous faisons une simulation physique nous discrétisons le temps (avec un pas en général que l'on peut écrire en base de 2 sur un nombre fini de bit (donc pas 0.1)). Et comme nous avons à la fin une liste de donnée, par exemple une liste des positions 3D de N particules. Et si nous voulons présenter les résultats il faut interpoler les données entres les pas de temps. "A coup" de ligne, polynôme, spline, bezier, ... L'interpolation est aussi utile pour le feefback très utilisé dans les effets spéciaux. Téchnique utiliser pour faire de la simulation physique contrôler exemple un fluide se forme comme un "personnage méchant et mange les gentils" (La Momie, Spider Man 3, ...) Mais nous verrons celà plus tard lorsque nous aurons maitriser les simulations physique "de base".

2.1 - Interpolation linéraire

Les interpolations linéraires sont les plus simples à comprendre. Entre 2 images f(t) et f(t+dt) (f peut représenter des données 1D, 2D, 3D, 4D, ..., nD) nous "traçons" une ligne. Si le pas de temps est assez petit l'esthétisme de l'animation est assez accépatable.
Nous connaissons f(t) et f(t + dt) et nous cherchons à connaitre f(t + k*dt) (0 < k < 1). Pour connaitre ces valeurs il faut procéder à cette dite interpolation linéraire. Tout d'abord cherchons la pente :
p = (f(t + dt) - f(t))/(dt)
Puis nous avons simplement à utiliser une formule vue au lycée :
f(t + k*dt) = f(t) + p*k
Exemple :

Supposons notre dt = 1 sec (2^0), nous savons que f(5) = 42 et f(6) = 51.
p = (51-42)/1 = 9
donc f(5 + k) = 42 + 9k
Par exemple f(5,5) = 42 + 9*0,5 = 46,5
En fonction de votre utilisation à vous de trouvez un procésus d'automatisation de cette interpolation.

2.2 - Interpolations Polynomiales

Avant toute chose il faut savoir que par N point il existe au moins un polynome passant pas ces N points de degré N-1. (TODO : voir le deg du poly)

2.2.1 - Mauvaise méthode

Cela peut vous paraitres inaproprier mais je pense que c'est nécéssaire de voir ma mauvaise méthode puisque c'esdt la plus intuitive donc nous avons tendance à directement implémenté celle là.
Supposons que nous avons une liste d'abscisse (X0, ..., Xn) et une liste d'ordonné (U0, ..., Un). En gros f(X0) = U0. Nous nous ne connaissons pas vraiment f(x) puisque Xi et Ui sont issus des résultats d'une simulation physique ; nous nous cherchons une interpolations de f(x). Et l'on considère que l'interpolation de f(x) est assez proche d'un polynome P(x). Donc ce que nous cherchons un polynome avec les cohéficients (a0, ..., an).
Notre polynome :

Nous voulons que le polynome passe par les ordonnées (U0, ..., Un) avec leur abscisse réspéctive alors

Et là il est facile de voir que l'on peut poser le problème matriciellement :

Et bien vous l'avez reconnu c'est la matrice de vandermonde et on calcule le determinant de cette matrice les yeux fermer. Et si les X_i sont distincts alors on montre facilement que le determinant ne s'annule jamais alors le système est soluble et admet une unique solution. Et la solution est :

Et bien cette solution est la moins bonne pourquoi ? Et bien le calcul matriciel est un des calcules les plus gourmand (surtout les calcules d'inverse, determinant, multiplication) et là nous avons les 3. Il serait plus judicieux de chercher une autre solution au risque d'attendre des heures devant sont ordinateurs. Bien que j'ai dit que l'API pour le moteur physique n'as pas pour objectif d'être temps réel mais si il existe une meilleur méthode tout aussi éfficace alors pourquoi s'en privé.
TODO : Graph

2.2.2 - Interpolation de Lagrange

J'ai omis un détail lorsque je vous est présenté la mauvaise méthode. Dans un cas général nous n'avons pas des puissance de X_i sur les lignes de la matrices mais des fonctions paramètré avec comme variables des X_i et qu'en général nous prenions de simple polynome cohéficienté. Mais comme nous cherchons l'éfficacité algorithmique nous avons trouvé (surtout Lagrange) une méthode pour avoir les solutions directement en changeant la fonction en question :

C'est là où la puissance de lagrange commence lorsque i=j alors la fonction = 1 sinon 0. Et après la résolution se fait tout seul :

Je sais que cela peut paraitre brutal mais ce sont les formules de Lagrange nous ne pouvons pas pas les changer si vous ne comprennez pas, essayer la téchnique du papier crayon, tester avec des exemples, jusqu'a ce que cela rentre.

Il faut noté que le polynome trouvé est le même que celui trouvé avec la mauvaise méthode.
TODO : do Graph
TODO : Soft

Bien entendu nous commetons une erreur par rapport à la fonction que nous interpolons. Soit X0 < X1 < ... < Xn les abscisses des points d'interpolation. Si la fonction u(x) est définie sur l'intervalle [X0; Xn] et qu'elle est (n + 1) fois dérivable sur]X0;Xn[, alors pour tout x apparenant à]X0;Xn[, il existe xi(x) appartient à]X0;Xn[tel que :

Ne me croyez pas sur parole regarder la démonstation sur google. Bien sur e(Xi) = 0 puisque notre polynome passe par les Ui en Xi. Vous allez surement me dire que puisque nous interpolons une fonction inconnu alors les dérivées k-ième (0 < k <= n+1) sont inconnu, de plus xi(x) est inconnu en un point donnée donc cette relation peu sembler inutile ; et pourtant...
TODO : Graph

2.2.3 - Interpolation d'Hermite

Comme vous avez pu le remarquer avec les interpolations de Lagrange nous faisons coincider les points avec la polynome interpolant. Mais avec l'interpolation d'Hermite nous ferons coincider les dérivées $k^{ième}$ au point d'interpolation. L'interpolation de Lagrange est un cas particuler de L'interpolation d'Hermite. Soit x_0, x_1, x_2, ..., x_n ; (n + 1) points d'interpolation distinct dans [a, b]. Et f une fonction dérivable jusqu'à l'ordre k, au point x_i. On pose m = n + k_0 + k_1 + ... + k_n. Il existe alors un polynome P_m de degré <= m. Appellé polynome d'interpolation d'Hermite. Le polynome d'Hermite est donnée par :

Les h_{ij} sont données par (Avec j = 0, 1, ..., k_i - 1) :

avec :

et :

Comme pour les polynomes de Lagrange il faut s'en persuader. Ou si l'on à un espris mathématique voir la démonstration (non présenter ici). Dans le cas contraire essayer par vous mêmes essayer d'interpoler une fonction connu (ou pas ce n'est pas le problème) faite une implémentation dans un langage quelconque. Et si cela ne marche pas, cela ne veut pas dire que Hermite avait tord. Seulement que votre implémentation n'est pas bonne. **TODO : Graph ; add in soft**
TODO : chercher fonction d'erreur

2.2.4 - Interpolation de Tchebychev

2.2.5 - Différences divisées

3 - Approximations

3.1 - Approximation uniforme

3.2 - Approximation quadratique

3.3 - Fonctions Spline (B-Spline)

Polynome de Bernstein

4 - Conclusion

Vous allez surement me dire que l'on est loin du moteur après cette tirade sans fin. Et bien sachez que nous sommes obliqué de passer par cette douloureuse étape. Ne déséspérer pas, après ce tutorial nous commencerons à avoir nos premier visuel et première simulation physique (mais l'équation de Schrödinger temporelle ce n'est pas pour tout de suite).