Taller de Métodos y Modelos Cuantitativos
Tercer Informe Proyecto
Fast Sum of Gaussians

Camilo Valenzuela Carrasco
201173030-5

Camilo Rivas Folch
201160089-4

Cristopher Barraza Vicencio
201173002-k

5 de enero de 2016

1. Definición del problema

El **Problema de Evolución de Formas** consiste en un conjunto de N puntos que interactuan entre ellos, de tal forma que van modificando su forma conforme el tiempo avanza. Las ecuaciones que describen el problema son las siguientes:

$$\frac{dx_i(t)}{dt} = \sum_{j=1}^{N} K(x_i, x_j) \cdot \alpha_j(t) \tag{1}$$

$$\frac{d\alpha_i(t)}{dt} = -\frac{2}{N} \sum_{j=1}^{N} \{\alpha_i(t)^T \alpha_j(t)\} \cdot K(x_i, x_j) \cdot (x_j(t) - x_i(t)) \tag{2}$$

$$x_s(t_0) = x_o \tag{3}$$

$$x_s(t_f) = x_f \tag{4}$$

Con $s = 1..N$, $K(x, y) = e^{-\frac{||x-y||^2}{2\sigma^2}}$. Con $x_s(t) = (x_s^x(t), x_s^y(t), x_s^y(t))$ la posición de la partícula s en el tiempo t y $\alpha_s(t)$ el vector momento de la partícula s en el tiempo t.

Se tiene un problema de valor de frontera (BVP), donde $x_s(t_0)$ y $x_s(t_f)$ son las posiciones de las partículas en $t = 0$ y en el tiempo final respectivamente. Lo que se busca son los valores de momento inicial $\alpha_s(0)$, de tal forma que las partículas pasen de su posición inicial a la final.

2. Dificultades del problema

2.1. Suma de Gaussianas

En las ecuaciones del problema se puede notar que existe una sumatoria con respecto a N puntos.

$$\frac{dx_i(t)}{dt} = \sum_{j=1}^{N} e^{-||x_j(t)-x_i(t)||^2/\sigma^2}\alpha_j(t)$$

El cálculo de esta sumatoria representa una tarea costosa, ya que el número de puntos que se utiliza es grande, por cada punto se tiene que calcular una exponencial.

Si tenemos M puntos donde se tiene que evaluar la sumatoria, y cada sumatoria tiene N terminos, necesitamos realizar $O(MN)$ evaluaciones. Mientras mayor sea la cantidad de puntos, será mucho mayor el tiempo de computación.

Como también tenemos que calcular la posición de los puntos en cada tiempo t, mientras más rápido se realice la suma, menos costoso será resolver el BVP.

2.2. Ecuaciones no lineales

Es posible describir el problema de manera genérica como:

$$\begin{pmatrix} \frac{d\mathbf{x}}{dt} \\ \frac{d\alpha}{dt} \end{pmatrix} = \begin{pmatrix} f_1(\mathbf{x},\alpha) \\ f_2(\mathbf{x},\alpha) \end{pmatrix} = \mathbf{F}(\mathbf{x},\alpha) \tag{5}$$

El cuál es un problema autónomo y acoplado, dado que para ambas componentes hay una dependencia de ambas variables. La dificultad que existe aquí es que $\mathbf{F}(\mathbf{x},\alpha)$ es no-lineal, por lo tanto no es simple desacoplar el sistema para poder resolverlo con algún método numérico y tiene que ser estudiado con detalle.

2.3. Método de Resolución

Dado el problema en 1, hay que definir una estrategia para resolver el problema que cumpla con ciertas exigencias. Se requiere que el método que se utilice mantenga constante la suma de momentos conforme avance el tiempo, ya que si se utilizan métodos que no conservan bien el momento, como euler, el valor de $\alpha(t_0)$ puede que sea mucho mayor al real.

Se nos ha sugerido el método del trapecio para mantener la estabilidad:

$$z_{n+1} = z_n + (t_{i+1} - t_i) \cdot \mathbf{F}\left(\frac{z_{n+1} + z_n}{2}\right) \tag{6}$$

La utilización de un método implícito para mantener la estabilidad del problema requiere un esfuerzo adicional, sobre todo teniendo en cuenta la problemática planteada por la función utilizada en 5, donde no es fácil resolver para z_{n+1} dada la naturaleza explicada anteriormente.

Cabe notar, que para poder solucionarlo se requiere conocer el jacobiano de la función F, debido a que por ser un método implícito, es necesario resolverlo con el método de newton. No obstante, este jacobiano no es simple de calcular y agregaría mucho más esfuerzo computacional al problema.

Cabe destacar que el problema en sí está formulado para que el momentum del sistema se mantenga constante en el tiempo, por lo tanto, es posible usar una función más exacta, como Odeint de Scipy, que además se encarga de aproximar el jacobiano del sistema, por lo que se decide utilizar éste método por el momento.

2.4. Implementación

Una última dificultad está en la forma de implementación. Al momento de implementar las sumas rápidas para resolver el problema, se tiene que pensar en generalizar lo más posible, para poder agregar

ya que para asegurar mayor rendimiento posible en cuanto a tiempo, lo ideal sería utilizar programación paralela o algún método pueda acelerar los cálculos computacionales, como sería el uso de las GPU, en específico programando el algoritmo en CUDA. Esta dificultad está en cuanto a que la forma

3. Trabajo realizado

3.1. Sumas Rápidas

En lo que respecta a las sumas rápidas, hasta el momento se ha implementado una versión de sumas rápidas en una dimensión utilizando *Fast Multiplole Method (FMM)*, en la cual se utiliza la división de puntos en regiones cercanas y regiones lejanas, pero no se ha incluido aún la división del espacio en Boxes, que agrupan puntos cercanos entre si, y dependiendo del punto que se está evaluando, la Box se puede comportar como un punto de la región lejana o varios puntos de la región cercana.

Este trabajo se hizo para entender la base de las sumas rápidas, y comenzar a comprender cómo hacen para aproximar los valores reales en menor tiempo de ejecución.

Para esto se utilizó la Ley de Gravitación Universal de Newton, que al igual que el problema que estamos estudiando es un N-Body problem, donde cada partícula interactua con todas las otras en algún modo, en el caso de la ley de gravitación la interacción depende de la distancia de la partícula estudiada con relación a las demás.

La interacción de una carga puntual z_0 está dada por la suma de todas las interacciónes con las otras cargas puntuales z con carga q.

$$\phi_{z_0}(Z) = q \log(z - z_0) \tag{7}$$

Para utilizar FMM, primero se define un radio r , para dividir el dominio en una región cercana y una lejana. En la región cercana definida como los puntos $z \in [z_0 - r, z_0 + r]$ se evalua la interacción directamente con 7.

Para la región lejana se utiliza *Multipole Expansion* que transforma la ecuación 7 en una serie infinita, y como resultado tenemos la siguiente ecuación:

$$\phi(z) = Q \log(z) + \sum_{k=1}^{\infty} \frac{a_k}{z^k} \tag{8}$$

donde

$$Q = \sum_{i=1}^{m} q_i \qquad\qquad a_k = \frac{\sum_{i=1}^{n} -q_i z_i^k}{k}$$

Dependiendo de la precisión deseada la serie puede ser truncada a p términos, además como a_k no depende del punto z donde se está evaluando ϕ, entonces puede ser precalculado para cada $k \in [0, p]$.

Al sólo tener que calcular p términos de la serie y poder precalcular los valores de a_k se puede acelerar el tiempo de ejecución del algoritmo, en esto se basa el FMM.

Tomando en cuenta lo aprendido con el ejemplo en 1D, pasamos a implementar el algoritmo Fast Gauss Transform, que busca realizar de manera eficiente la siguiente sumatoria:

$$G(t_i) = \sum_{j=1}^{N} q_j e^{\frac{-|t_i - s_j|^2}{\sigma}} \tag{9}$$

donde t_i son los puntos Targets y s_j son los puntos Sources, y q_j el peso o carga del source. Esta ecuación es similar a la de nuestro problema, pero nuestro q_j seria el vector de momentos de la partícula j, $\alpha_j(t)$ o se tendrían que pre-calcular los $\{\alpha_j(t)^T \alpha_i(t)\} \cdot (x_j(t) - x_i(t))$ y utilizar esos valores para los $q_j(t)$.

3.1.1. Algoritmo Fast Gauss Transform

Para comenzar, se tiene como supuesto que el dominio donde están los sources y targets están pertenecen a $t_i, s_j \in B_0 = [0,1]^d$, con d la cantidad de dimensiones de los sources y targets.

El algoritmo divide el dominio B_0 en Boxes de tamaño $r\sqrt{\delta}$, con un total de $Nside^d$ Boxes, el paper dice que para la elección de $Nside$ hay que buscar el valor de $r \leq \frac{1}{2}$ de tal forma que $Nside$ de un valor entero.

Para calcular $Nside$, nos centraremos solo en una dimensión, como el dominio de cada dimensión va de [0,1], y cada Box mide $r\sqrt{\delta}$, por lo que

$$Nside = \lceil \frac{max\_dom(B_0) - min\_dom(B_0)}{r\sqrt{\delta}} \rceil = \lceil \frac{1}{r\sqrt{\delta}} \rceil$$

Como aproximamos $Nside$ al entero más cercano, calculamos el nuevo r

$$r' = \frac{1}{Nside\sqrt{\delta}}$$

Ahora ya sabemos cuantas Boxes va a tener nuestro algoritmo, vamos a tener dos tipos de Boxes, TargetBoxes (B) son las que acumularan el potencial de cada partícula y las SourceBoxes (C) que interactuan con las TargetBoxes.

Utilizando r' y la precisión ϵ calculamos p, que va a ser la variable utilizada para truncar las series infinitas, y para elegir que método de aproximación utilizar, además calculamos n que es la cantidad de Boxes que pueden interactuar con una TargetBox y depende de los parámetros δ y ϵ.

Para la aproximación se utilizan 4 métodos, todo depende de la cantidad de términos que hayan en las Boxes que se están analizando, sea M_C la cantidad de target en una TargetBox C y N_B la cantidad de sources de una SourceBox B. Primero se eligen puntos un punto de corte para cada caja, N_F para las SourceBox y M_L para las TargetBox, que en nuestro caso usaremos p^d en ambas, ya que esa es la cantidad de términos en la que se truncan las series utilizadas. Luego:

- Si $N_B < N_F$: No hay suficientes términos para truncar la serie, por lo que la SourceBox B envia N_B Gausseanas a la interacción. (Se calcula directo).
- Si $N_B >= N_F$: Hay términos suficientes para truncar la serie, por lo que B en vez de enviar Gausseanas envia una **Hermite Expansion** a la interacción.
- Si $M_C < M_L$: C evalua directamente lo que le envia B (Sea N_B Gausseanas o la Hermite Expansion)
- Si $M_C >= M_L$: C acumula lo que le envía B mediante una serie de Taylor.

La **Hermite Expansion** utiliza la *Hermite Function* h(t) y está centrada en el SourceBox con centro s_B y es de la forma:

$$G(t) = \sum_{\alpha \geq 0} A_\alpha h_\alpha \left(\frac{t - s_B}{\sqrt{\delta}} \right) \tag{10}$$

donde los coeficientes A_α están dados por:

$$A_\alpha = \frac{1}{\alpha} \sum_{j=1}^{N_B} q_j \left(\frac{s_j - s_b}{\sqrt{\delta}} \right)^\alpha \tag{11}$$

En nuestro caso α va a ser un indice múltiple que depende de la cantidad de dimensiones del problema, por ejemplo para puntos en un espacio de 3 dimensiones (x,y,z), α va a ser una 3-tupla.

En el caso de que la TargetBox C tenga que acumular la interacción como una serie de Taylor, tiene que utilizar las siguientes ecuaciones:

- Si B nos envia las N_B Gausseanas, C tiene que acumular las interacciones utilizando:

$$G(t) = \sum_{\beta \geq 0} B_\beta \left(\frac{t - t_C}{\sqrt{\delta}} \right)^\beta \tag{12}$$

  Con los coeficientes:

$$B_\beta = q \frac{(-1)^{|\beta|}}{\beta!} h_b \left(\frac{s_j - t_C}{\sqrt{\delta}} \right) \tag{13}$$

- En el caso de que B nos envié una Hermite Expansion, C tiene que acumular las interacciones utilizando:

$$G(t) = \sum_{\beta \geq 0} C_\beta \left(\frac{t - t_C}{\sqrt{\delta}} \right)^\beta \tag{14}$$

  con los coeficientes C_β se calculan con:

$$C_\beta = \frac{(-1)^{|\beta|}}{\beta!} \sum_{a \leq p} A_\alpha h_{\alpha+\beta} \left(\frac{s_B - t_C}{\sqrt{\delta}} \right) \tag{15}$$

Utilizando estas aproximaciones pasamos de tener una complejidad O(MN) a O(M+N), por lo que el tiempo de ejecución baja considerablemente.

3.2. Resolviendo el Problema de Evolución de Formas

En la formulación del problema a resolver, se ha comenzado a avanzar en el planteo de este para tratar de llegar a una representación que permita utilizar alguna técnica conocida sobre el conjunto de datos para lograr resolver el problema. Para esto se buscó alguna forma de desacoplar el sistema, pero como ámbas ecuaciones dependen de las mismas variables y debido a su naturaleza no lineal, no se encontró una forma sencilla de desacoplarlo.

Por otra parte, para resolver el BVP, se nos recomendó utilizar **Shooting Method**, para transformar la Ecuación Diferencial a un problema de valor inicial (IVP), y luego utilizar Backward Euler con la regla del punto medio, ya que es mucho mejor para mantener el momento del sistema.

3.2.1. Shooting Method

Las ecuaciones diferenciales que componen este problema y los datos con los que se cuentan componen un BVP, el cual puede ser resuelto a través de *Shooting Method*, convirtiendo el problema en un IVP, en el cual será necesario justamente encontrar el impulso inicial que llevó el sistema desde el estado inicial al estado final. Por lo tanto, resolver el problema mediante *Shooting Method* se enfocará en resolver:

$$F(\alpha) = X(1) - \hat{X}_\alpha(1) \tag{16}$$

Siendo α el impulso inicial, $X(1)$ las posiciones de cada uno de los puntos del sistema en el tiempo $t = 1$ y $\hat{X}_\alpha(1)$ la posición de los punto en el tiempo $t = 1$ dado un cierto valor de α. El objetivo el lograr minimizar el valor de $F(\alpha)$.

3.2.2. Utilizando Sumas Rápidas

Para realizar la integración sobre la variable temporal, necesitamos calcular el cambio de la posición y momentum de la partícula, para esto tenemos que calcular la ecuación (1) y (2), utilizando sumas rápidas.

La primera ecuación se puede calcular utilizando la forma estándar del Fast Gauss Transform que se puede observar en la ecuación (9).

En cambio la segunda ecuación se vuelve más compleja para evaluarla, porque además de tener el kernel gausseano se tiene un producto punto de vectores y la diferencia entre el punto target y el source.

Para poder realizar esta suma eficientemente hay que manejarla hasta tener algo parecido a la ecuación (9).

Como se está trabajando en 3 dimensiones el producto punto de $\alpha_i(t)^T \alpha_j(t)$ se puede expresar de la siguiente forma

$$\begin{bmatrix} \alpha_i^x \\ \alpha_i^y \\ \alpha_i^z \end{bmatrix} \cdot \left[\alpha_j^x, \alpha_j^y, \alpha_j^z,\right] = \alpha_i^x \cdot \alpha_j^x + \alpha_i^y \cdot \alpha_j^y + \alpha_i^z \cdot \alpha_j^z \tag{17}$$

Por lo que la segunda ecuación va quedando de la siguiente forma:

$$\frac{d\alpha_i(t)}{dt} = -\frac{2}{N} \sum_{j=1}^{N} \left(\alpha_i^x \cdot \alpha_j^x + \alpha_i^y \cdot \alpha_j^y + \alpha_i^z \cdot \alpha_j^z\right) \cdot K(x_i, x_j) \cdot (x_j(t) - x_i(t)) \tag{18}$$

Luego podemos multiplicar el kernel por cada término

$$\frac{d\alpha_i(t)}{dt} = -\frac{2}{N} \sum_{j=1}^{N} \left(\alpha_i^x \cdot \alpha_j^x K(x_i, x_j) + \alpha_i^y \cdot \alpha_j^y K(x_i, x_j) + \alpha_i^z \cdot \alpha_j^z K(x_i, x_j)\right) \cdot (x_j(t) - x_i(t)) \tag{19}$$

Ahora multiplicamos el termino de la izquierda por cada término de la derecha

$$\begin{aligned} \frac{d\alpha_i(t)}{dt} = &-\frac{2}{N}[\sum_{j=1}^{N} \left(\alpha_i^x \cdot \alpha_j^x K(x_i, x_j) x_j(t) + \alpha_i^y \cdot \alpha_j^y K(x_i, x_j) x_j(t) + \alpha_i^z \cdot \alpha_j^z K(x_i, x_j) x_j(t)\right) \\ &- \sum_{j=1}^{N} \left(\alpha_i^x \cdot \alpha_j^x K(x_i, x_j) x_i(t) + \alpha_i^y \cdot \alpha_j^y K(x_i, x_j) x_i(t) + \alpha_i^z \cdot \alpha_j^z K(x_i, x_j) x_i(t)\right)] \end{aligned}$$

Ingresamos la sumatoria a cáda término, dejando una suma de varias sumatorias pequeñas, dejamos fuera de cada sumatoria las variables que no dependen de ella

$$\begin{aligned} \frac{d\alpha_i(t)}{dt} = &-\frac{2}{N} \cdot [\alpha_i^x \sum_{j=1}^{N} \alpha_j^x x_j(t) K(x_i, x_j) + \alpha_i^y \sum_{j=1}^{N} \alpha_j^y x_j(t) K(x_i, x_j) + \alpha_i^z \sum_{j=1}^{N} \alpha_j^z x_j(t) K(x_i, x_j) \\ &- x_i(t)(\alpha_i^x \sum_{j=1}^{N} \alpha_j^x K(x_i, x_j) + \alpha_i^y \sum_{j=1}^{N} \alpha_j^y K(x_i, x_j) + \alpha_i^z \sum_{j=1}^{N} \alpha_j^z K(x_i, x_j))] \end{aligned}$$

Ahora tenemos nuestra segunda ecuación con seis sumatorias, que pueden ser evaluadas utilizando Fast Gauss Transform, las primeras tres sumatorias, los pesos que se utilizan es la multiplicación de un componente del momentum (α_j^d) con el vector de posición (x_j), las otras tres sumatorias son más parecidas a la ecuación numero (1).

Para simplificar un poco la evaluación, realizamos cada dimensión por separado, por lo que para la primera ecuación se calculan tres sumas rápidas, una por cada dimensión, y para la segunda se calculan dieciocho sumas rápidas.

Como nuestra implementación de Fast Gauss Transform utilizando Python se volvió tediosa (por la complejidad de calcular los coeficientes hermitianos utilizando recursión), además la implementación incompleta se demoraba más en computar que la suma directa, por lo que se decidió utilizar una biblioteca ya existente, llamada Figtree, la que utiliza IFGT (Improved Fast Gauss Transform) y Figtree, para aproximar las sumas.

Las primeras pruebas realizadas, con los datos entregados por Felipe Arrate, que son 4028 puntos en 3 dimensiones, nos entregaron los siguientes resultados.

	Suma Directa	Figtree	Diferencia
Primera Ecuación	3.79s	170ms	$4{,}24e^{-13}$
Segunda Ecuación	13.6s	621ms	$2{,}48e^{-14}$

4. Plan de Acción

Dado el trabajo realizado, se propone el siguiente plan de acción:

- Comprender como funciona el algoritmo Improved Fast Gauss Transform, al transformar la suma de gausseanas a una serie de Taylor.
- Unir los métodos utilizados para la optimización en el cálculo de las derivadas con el método elegido para resolver el problema.
- Realizar pruebas para verificar que las técnicas utilizadas para optimizar el cálculo y el método para resolver el problema funcionan correctamente y en un tiempo de ejecución reducido, además de verificar que al resolver el problema se esté conservado el momentum del sistema.
- Luego de verificar que todo funciona correctamente, se pasará a representar graficamente el comportamiento que tienen los puntos desde la situación inicial a la final.

5. Implementaciones de Fast Gauss Transform o Similares

- Fast computation of Gauss transforms in multiple dimensions
  `https://github.com/vmorariu/figtree`
- Fast Gauss Transform
  `https://github.com/gadomski/fgt`
- Kernel density estimation which was optimized by the fast Gauss transform.
  `https://github.com/y-mitsui/fastKDE`
- Fast Gauss Transform (Python)
  `https://github.com/josephpmcdonald/FastGauss`