Trabajo Práctico 4: Impacto profundo

Universidad de Buenos Aires
Facultad de Ciencias Exáctas y Naturales
(FCEyN)
Departamento de Computación

González Sergio (gonzalezsergio2003@yahoo.com.ar)
González Emiliano (XJesse_JamesX@hotmail.com)
Ferro Mariano (eltrencitomasverde@gmail.com)

19 de noviembre, 2007

Resumen

En este trabajo abordaremos un problema particular que requiere la utilización del método de interpolación con splines cúbicos normales. Dicho problema trata sobre un supuesto bombardeo de misiles intergalácticos hacia un planeta desconocido, éstos tienen que ser destruidos antes de llegar a destino. Se cuenta con cierta cantidad conocida de mediciones experimentales provistas por un radar"de las posiciones de los misiles. Los splines se utilizan para calcular las trayectorias de los misiles haciendo extrapolación en el último polinomio, correspondiente al intervalo de tiempo donde se realizaron las dos mediciones finales. De esta forma se pretende tener información sobre la posición de cada proyectil durante el tiempo que transcurre entre la última medición y su hipotético impacto. Otra de las exigencias que conlleva el problema mencionado, radica en neutralizar la mayor cantidad de misiles teniendo un número acotado (y menor) de bombas. Para resolver esto se requiere implementar una estrategia adecuada tratando de maximizar la cantidad de misiles neutralizados y evitando destruir el propio planeta con las bombas defensivas.

Palábras clave:

- Splines cubicos naturales.
- Extrapolación.
- Simulación.
- Instante

Introducción:

En muchas ocasiones tenemos información que relaciona valores de variables, donde cada una depende de otra. En estos casos no se tiene una función que describa este conjunto de valores, por lo cual se busca un método que la obtenga. Para esto se utiliza la técnica de interpolación de datos. En el subcampo matemático, se denomina interpolación a la construcción de nuevos puntos a partir de un conjunto de valores que se encuentran discretizados. El conjunto de valores es una muestra que se obtiene a través de mediciones experimentales, a partir de las cuales se pretende construir una función que los ajuste y caracterice.

Además de la utilidad mencionada anteriormente, la interpolación de datos se utiliza para aproximar funciones complejas o difíciles de calcular, por otras más simples. El procedimiento es similar al ya explicado, es decir, se genera una muestra de la función a aproximar y luego, interpolando esos puntos, se obtiene una nueva función que aproxime a la anterior. Por ser una estimación, esta nueva función conlleva un error de aproximación con respecto a la original, que estará sujeto a las características del problema y del método de interpolación utilizado.

En todos lo casos, teniendo un conjunto de valores que ordenaremos en pares (x_i, y_i), correspondientes al resultado del experimento, se busca obtener una función que llamaremos f que verifique:

$$f(x_i) = y_i \forall i = 1..n$$

Esta función, se denomina función interpolante en los puntos $x_1..x_n$. Existen varios métodos para calcularla, entre ellos la llamada interpolación lineal, la polinómica o la interpolación mediante splines.

Para este trabajo, se utilizara el método de interpolación mediante splines. Un spline es una curva que unirá dos mediciones del muestreo de la función incógnita. Esta técnica define un conjunto de valores de a trozos mediante polinomios de grado menor o igual a al atribuido al spline. Esta característica, implica que al tener expresiones de bajo grado en cada intervalo, no se generen oscilaciones, que en la mayoría de las aplicaciones resultan indeseables. Es por esta razón que los splines utilizados casi con exclusividad son aquellos que minimizan su grado, impidiendo la aparición de dichas oscilaciones evitando interpolar mediante polinomios de grado elevado. Además esta característica apareja una cantidad de cálculos menor que los demás métodos. Esto se vuelve apreciable al momento de obtener resultados con un número grande de datos.

A continuación se detallan las condiciones que debe cumplir un spline cúbico para ser tratado como tal. Los valores x_j son los correspondientes a la muestra de la función que se desea interpolar, mientras que S_j es la candidata para unir los puntos (x_j, x_{j+1}) :

1. $S(x_j) = f(x_j)$
2. $S_{j+1}(x_{j+1}) = S_j(x_{j+1})$
3. $S'_{j+1}(x_{j+1}) = S'_j(x_{j+1})$
4. $S''_{j+1}(x_{j+1}) = S''_j(x_{j+1})$

Para que la función S_j (curva cúbica en cada intervalo) cumpla con los requisitos antes expuestos, se la fuerza a tener la siguiente forma:

$$S_j(x) = a_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3$$

De las condiciones anteriores, cumplidas por el polinomio se desprende que:

1. $a_j = f(x_j)\ \forall j = 0..n-1$
2. $a_n = S_{n-1}(x_n)$
3. $b_j = \frac{a_{j+1}-a_j}{h_j} - h_j(\frac{2}{3}c_j + \frac{c_{j+1}}{3})$
4. $b_{j+1} = b_j + 2c_j(x_{j+1} - x_j) + 3d_j(x_{j+1} - x_j)^2\ \forall j = 0..n-2$
5. $b_n = S'_{n-1}(x_n)$
6. $S''_j(x_{j+1}) = S''_{j+1}(x_{j+1})\ \forall j = 0...n-2$
7. $2c_n = S''_{n-1}(x_n)$
8. $d_j = \frac{c_{j+1}-c_j}{3h_j}\ \forall j = 0...n-1$ con $h_j = x_{j+1} - x_j$

Donde $h_j = x_{j+1} - x_j$

Luego se iguala la ecuación 3 con la 4, esta última con subíndice j en lugar de $j+1$, y se remplaza b_{j-1} por la expresión 3 con el índice correspondiente. Como resultado se obtiene:

$$\frac{1}{3}c_{j-1}h_{j-1} + c_j(\frac{2(h_j + h_{j-1})}{3}) + c_{j+1}\frac{h_j}{3} = \frac{a_{j+1} - a_j}{h_j} - \frac{a_j - a_{j-1}}{h_{j-1}}$$

$$\forall j = 0...n-1$$

Con esta ultima ecuación se obtienen las expresiones de c_j para los polinomios de cada uno de los intervalos, en función de los h y los a, todos conocidos. Para calcualar los valores reales se genera un sistema de ecuaciones, que se resuelve aplicando un método cualquiera de resolución (en nuestro caso se uso uno propio, basado en la eliminación gaussiana y la sustitución hacia atras, se lo explicará en la sección Detalles de la Implementación), este sistema no esta completo, ya que faltan dos ecuaciones para que sea cuadrado. Para esto se igualan las derivadas segundas de los polinomios de cada extremo a cero (definiendo un spline cubico natural), teniendo la misma cantidad de ecuaciones que de incognitas, dados estos datos se pueden hallar los valores exactos correspondientes a los coeficientes c_j de cada polinomio. Cabe destacar que el anterior sistema de ecuaciones es tridiagonal y además diagonal dominante, por lo cual al momento de resolverlo se pueden aprovechar, tanto desde el punto de vista analítico como computacional, las características propias de éste para hacer más óptima la solución en cuanto a tiempo y/o espacio requerido.

Una vez obtenidos los valores de c_j, se procede a calcular los d_j y b_j correspondientes (los valores de a_j son conocidos ya que se obtienen de la ecuación 1 y 2). Para esto se utilizan las ecuaciones 8 para d_j y una combinación de las ecuaciones 4 y 5 para los valores de b_j.

Con los polinomios obtenidos, se puede calcular cualquier valor dentro del intervalo que definen los datos evualuando la función correspondiente al punto (interpolación), o bien calcularlos fuera de dicho rango evaluando el último spline más alla de los limites de las mediciones (extrapolación). Obviamente, esta metodología supone que la función que se esta analizando continúa tal y como el último polinomio, y no va a tener comportamientos diferentes a éste a partir de cierto punto.

Desarrollo:

El problema que abordamos durante este trabajo consiste en implementar un programa que lea las mediciones de las posiciones de los misiles desde un archivo de texto, y que deje en otro archivo el momento y la posición en la que debería detonarse cada bomba defensiva, con el objetivo de maximizar la cantidad de misiles enemigos destruidos.

Para lograr el objetivo planteado, se evaluaron distintas estrategias que progresaban sobre la cantidad de misiles destruidos. Así, comenzamos con métodos para contrarrestar el ataque de dos misiles. La primera opción consistía en calcular el promedio de las distancias en ambos ejes cartesianos. Esta forma de situar el punto de explosión de la bomba resultaba efectiva, además demostró ser fácilmente escalable a una mayor cantidad de objetivos. Por desgracia, así como era muy adaptable, no garantizaba precisión para más de dos misiles, ya que de existir un proyectil alejado de los otros, éste desplazaría el punto donde detonar la bomba. Otro inconveniente, resulta cuando un cúmulo de objetivos acerca demasiado el punto promedio a si mismo, dejando fuera de la explosión misiles individuales alejados.

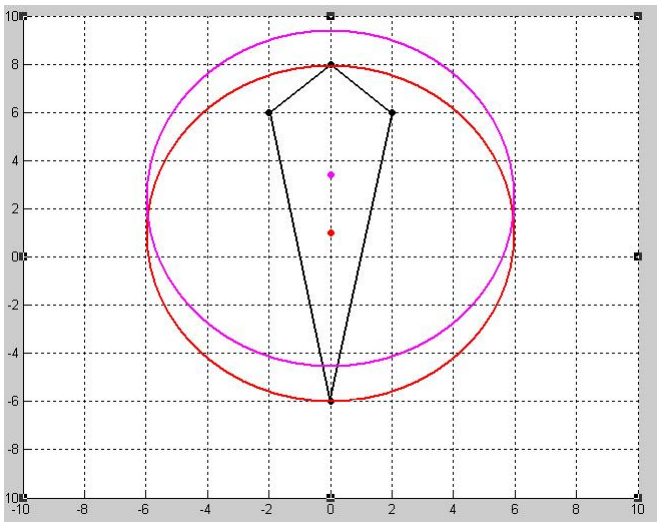

Figura 1: Aqui se puede apreciar un caso patológico para el promedio, bien resuelto por el método de la mayor distancia

Teniendo esto en cuenta, buscamos una nueva estrategia que nos asegurase un mejor rendimiento para tres o más misiles. Así surgió la idea de tomar tan solo las distancias más largas (entre los misiles) en cada eje (la distancia en cada eje para la coordenada correspondiente), dividiéndolas por dos, para obtener un punto medio. De esta forma, siempre quedan atrapados en el radio de la explosión los proyectiles más extremos, que son los más susceptibles de ser excluidos

ante el mínimo cambio en el punto a detonar. También, por ser los más alejados, de haber otros misiles cerca (siempre y cuando sea posible detonar una bomba sobre todos), éstos caen en el radio de la explosión.

A pesar de las bondades de este método, hubo que pensar un poco más, ya que para casos en los que las posiciones de alguno de los misiles no son simétricas, la figura se descompensa, y como sucedía con el caso de los promedios, se pierde el punto óptimo para la explosión. En el gráfico siguiente se muestra uno de estos casos.

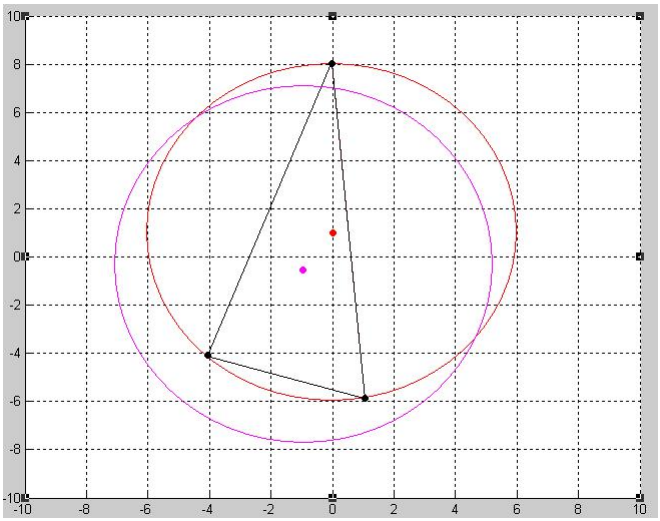

Figura 2: En este caso, la mayor distancia no funciona, pero el centro es capturado por la tecnica de las mediatrices

Al notar que los métodos elaborados fallaban para varios casos, tratamos de idear uno que sea generalizable. Esta búsqueda no resultó tan efectiva como nos hubiera gustado, aún así, obtuvimos una técnica que se adapta a varios entornos y numerosas configuraciones de misiles. Aquello que inspiró este modo de buscar el mejor punto a detonar fue la inscripción de figuras geométricas en una circunferencia. De allí que se utilizan las mediatrices de los lados de un triangulo formado por tres misiles, para, de su intersección obtener el punto deseado. Solo se utilizan tres proyectiles para formar la figura, ya que de emplear más, podría darse un trazado no regular, lo que torna imposible la inscripción en la circunferencia de la bomba de ese polígono(Figura 3).

Otro caso patológico para este método radica en que cuando los misiles se encuentran alineados (o tienden a estarlo) el punto buscado se aleja (Figura 4), mientras menor es la distancia sobre uno de los ejes del trío de misiles, más distancia los separa del punto a detonar. Este es un comportamiento esperable, ya que la técnica trata de encontrar el centro de un circulo, del cual los proyectiles serían parte de su perímetro.

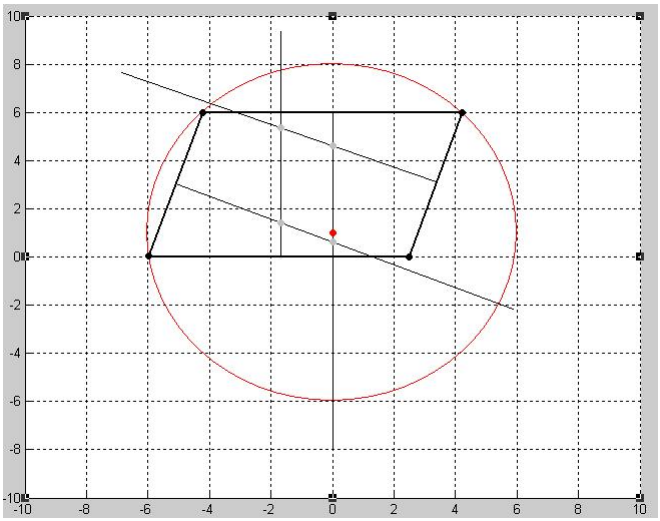

Figura 3: Este es uno de los casos en los que el sistema de mediatrices no funciona

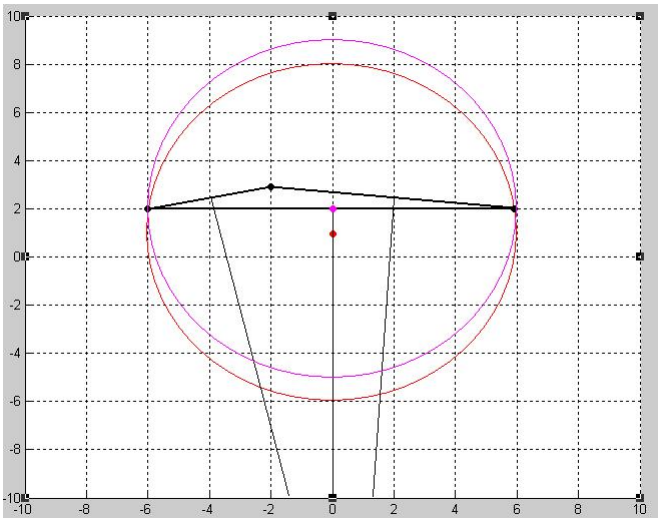

Figura 4: Este es otro caso patológico para las mediatrices, el método de la mayor distancia lo tratría correctamente

Como ninguna técnica resultó óptima para una mayoría abrumadora de casos, durante la implementación se optó por una combinación de las mismas, de forma tal, que cada método se desempeñe sobre las situaciones que mejor maneja. De esta forma, los casos de dos misiles se resuelven calculando el promedio de la distancia y los casos de tres se resuelven calculando la mitad de la distancia más larga entre los puntos, en el caso en que la formación de misiles tiende a

estar alineada, y en el resto, a través del empleo de mediatrices.

Entre otras decisiones de implementación se optó por manejar la variable tiempo en una discretización de intervalos de 0,05 segundos, considerando que cada medición de 'radar' de las posiciones de los misiles llega cada segundo. Consideramos como un "instante", dichos intervalos de tiempo, en ellos se calcula la posición extrapolada de cada misil.

Cabe aclarar que para calcular las posiciones de los proyectiles, tanto extrapolando como intrapolando utilizamos dos splines, uno para cada coordenada de los ejes cartesianos, ambos en función del tiempo. De esta forma tenemos una evaluación del progreso sobre los ejes en cada intervalo de tiempo. Combinando y evaluando los polinomios obtenidos en cada instante, llegamos a la posición exacta del misil.

Como consideración final, podemos mencionar otra visión completamente distinta a la adoptada, sin discretizar el tiempo, tomando funciones continuas, y analizando geométricamente si es posible destruir los misiles según su trayectoria. Esta alternativa no fue adoptada, ya que nos resulto inherente al problema discretizar de alguna forma el tiempo para manejar de forma simplificada los casos y las variables.

Detalles de la implementación:

Para este trabajo se implementó una nueva clase que se encarga que de describir a los objetos que vamos a interceptar, los misiles. A estos se les dio un comportamiento adecuado al problema. La clase se llama Misil y a continuación se detallan sus miembros y métodos:

- CantMediciones(Int): cada misil tiene el número de mediciones, correspondientes a la muestra que define su trayectoria. Este dato se utiliza para calcular el spline que permite extrapolar el camino que recorre el misil en los siguientes intervalos de tiempo.
- Identificador(Char): cada uno de los misiles tiene un identificador, este número se utiliza para identificar conjuntos de misiles para calcular las coordenadas en donde explotar la bomba. Estos conjuntos son determinados de acuerdo a un algoritmo que se encarga de destruir la mayor cantidad de misiles posibles, dicho método se explicará en detalle algunos párrafos más adelante.
- Destruido(bool): éste indica si el misil fue destruido. Este dato es necesario por la forma en la que se decide cuales serán los misiles a explotar. Se utiliza mayormente a fin de no solapar grupos de misiles candidatos en el espacio ni en el tiempo. Por ejemplo, si tengo un conjunto de cuatro misiles en el tiempo t, y decidí destruirlo, otra hipotética formación de tres proyectiles, la cual contenga alguno de los ya destruidos (en el instante t o posterior) será tenida en cuenta con solo dos misiles.
- PosicionX/PosicionY(long double[4]): estos arreglos contienen los coeficientes correspondientes a los polinomios calculados con el método de splines. Estos se utilizan con un par de funciones llamadas posicionX y posicionY, obteniendo asi las coordenadas con respecto al tiempo sobre el eje de las absisas y el de las ordenadas.

Ahora se detallarán las funciones métodos, tanto públicas como privadas, que tengan cierta relevancia desde el punto de vista númerico, las que sean concernientes solo al campo de la implementación, podrán ser encontradas en el código con sus debidos comentarios.

- Misil(): la clase cuenta con constructor por defecto, por copia, paramétrico, y un operador '=', como métodos de generación de instancias. En particular, el constructor paramétrico recibe un identificador, dos arreglos de long double que contienen las mediciones en los ejes X e Y y la cantidad de dichas mediciones. Es en éste donde se llama a la función splines para que generé los polinomios interpoladores que emularán la trayectoria de los misiles
- posicionX()/posicionY(): métodos mencionados más arriba, son los encargados de evaluar los polinomios formulados mediante la función spline.

Simplemente realizan las multiplicaciones pertinentes, valiéndose de los arreglos miembros de la clase, y de un instante elegido por el usuario donde efectuar la evaluación.

- spline(): este algoritmo es sin duda el más importante de la clase, ya que es el encargado de generar el polinomio interpolador entre cada punto de las mediciones recibidas en un arreglo. Recordar que como elaboramos un spline para el eje X y otro para el eje Y (ambos en función del tiempo transcurrido) solo necesitamos una coordenada por cada dato.

- spline(): este algoritmo es sin duda el más importante de la clase, ya que es el encargado de generar el polinomio interpolador entre cada punto de las mediciones recibidas en un arreglo. Recordar que como elaboramos un spline para el eje X y otro para el eje Y (ambos en función del tiempo transcurrido) solo necesitamos una coordenada por cada dato. Para calcular los coeficientes utilizamos las características de la matriz que se genera usando el método explicado en la sección Introducción. Además se implemento una optimización para esta matriz en particular, el algoritmo utiliza el método de gauss pero solo lo hace temporalmente sobre los datos que se necesitan. Como dicha matriz es tridiagonal, si se la observa con detenimiento al triangularla se descubre un patrón con respecto a los valores que tiene los elementos que van en la diagonal. Este patrón corresponde al siguiente, al triangular desde la 2da fila hacia la anteúltima, el coeficiente para anular los elementos debajo de la diagonal es siempre es $coeficiente = 1/diagonal$, y luego el valor final del casillero que corresponde a la diagonal es $diagonal = 4 - 1 * coeficiente$. Como solo es necesario el valor de la diagonal en la anteúltima fila, los valores antes mencionados se van acumulando, sin hacer gauss totalmente, ahorrando varias cuentas, lo que ayuda al rendimiento total del algoritmo.

  Al momento de trabajar con la matriz mencionada en la Introducción, notamos que por ciertas características de la misma era mejor no generarla, sino tan solo manipular los coeficientes estrictamente necesarios, para evitar un desperdicio de memoria. Al tener coordenadas enteras y consecutivas, los factores de la matriz son, para cada fila, 1, 4 y 1. A esto se le suma la propiedad de ser tridiagonal, con lo que se optó por resolver el sistema de la siguiente manera: Primero, como la fila 1 de la matriz esta igualada a cero, el c_1 resulta nulo, por lo que se lo ignora en la fila 2 (lo mismo sucederá con el c_n),y con eso se tiene la primera fila "triangulada". Segundo, se procede con esta triangulación virtual, y se nota que como los numeros son iguales fila a fila, resulta que el coeficiente siempre es 1/diagonal, y tras restar la $fila_i$ a la $fila_{i+1}$, queda $e_{i+1i+1} = 4 - coeficiente$, que será el nuevo pivote.

  Gracias a esta forma tan regular, se pudo escribir un algoritmo breve y sensillo para obtener el elemento c_{n-1} de la matriz, que era el necesario para reemplazar en las ecuaciones enunciadas en la introducción, y obtener el resto de los coeficientes del polinomio.

Es importante notar que solo son necesarios los coeficientes del último spline, ya que a partir de este es posible extrapolar puntos más alla del intervalo de mediciones, para aproximar la trayectoria del proyectil.

Hasta aquí se comprende los detalles del módulo Misil, además de esta clase, se implementaron varias funciones y estructuras que no pertenecen a ningun objeto, sino que están declaradas en el archivo main.cpp, se realizará ahora una descripción detallada de las más relevantes:

Para un mejor manejo de las coordenadas cartesianas, ya agobiados por el uso de arreglos de dos posiciones para cada (X,Y), se implemento una estructura (un struct de C) llamada Par, que simplemente contiene, dos valores de long double. Además, para facilitar el manejo de colecciones de elementos de las cuales no teníamos idea precisa sobre su longitud, y para no desperdiciar memoria generando arreglos de dimensiones exageradas, se resolvió utilizar el template 'List' de la Standard Template Library (STL) de C++. Esta plantilla, esta implementada sobre nodos doblemente encadenados, y la operación para agregar un elemento, tanto atrás como adelante, es de complejidad constante (O(1)), lo que resultaba óptimo para nuestras necesidades, ya que haríamos una lectura secuencial de los datos, y no aleatoria, caso en el que el empleo de arreglos podría reportar cierta ventaja.

Para utilizar conjuntamente con esta lista, se creo otro 'struct', denominado Grupo, que hace las veces de contenedor de misiles, y además de cierta información de cada instante analizado, en pos de encontrar la mejor opción. Esta estructura cuenta con una lista de punteros a Misiles, y tres long doubles: el instante al que hace referencia el Grupo (el tiempo en segundos, no el número de instante) y las coordenadas en X y en Y a detonar para esos misiles y en ese exacto momento.

También se implementó una función que calcula la distancia entre dos pares, y otra que obtiene la intersección de las mediatrices de un triángulo. Esta última recibe un arreglo de punteros a Misiles, un instante y un radio para la explosión de una bomba. Con estos elementos primero calcula la pendiente y los puntos de paso de las mediatrices, luego su intersección, y finalmente, compara el punto obtenido con el radio de la detonación, a fin de decidir si el punto encontrado es óptimo, o por lo menos viable.

Todos estos algoritmos fueron implementados con el objetivo de servir de funciones auxiliares al motor de este trabajo el procedimiento que dimos en llamar Armageddon. Debido a las dimensiones de éste, y a la cantidad de decisiones que toma, convenimos que lo mejor para comprender su funcionamiento era separarlo en varias secciones, según se suceden los ciclos.

Resultados:

Discusiones:

Conclusiones:

Apendice A:

Enunciado:

Laboratorio de Métodos Numéricos - Segundo cuatrimestre 2007
Trabajo Práctico Número 4: Impacto profundo

Introducción

Nos encontramos nuevamente en el Centro de Operaciones Logísticas Laterales (C.O.L.L.), en uno de los momentos más dramáticos de la XLII Guerra Intergaláctica. El planeta Z-80 está siendo atacado por un conjunto de misiles que amenazan su integridad, y nuestra misión consiste en detener este cruel ataque.

Para defender al planeta Z-80 contamos con un número reducido de bombas de destrucción masiva. Estas bombas son enviadas a un punto del espacio circundante a nuestro planeta, donde se las hace explotar. La explosión genera una onda expansiva que destruye todos los objetos (y misiles) ubicados a menos de un cierto radio crítico del centro de la explosión.

Nuestros radares nos proporcionan la posición exacta de los misiles enemigos a intervalos de tiempo aproximadamente constantes[1]. En función de estas mediciones deberemos determinar las trayectorias futuras de los misiles y, sobre la base de estas estimaciones, deberemos decidir en qué lugar y en qué momento tenemos que producir las explosiones de nuestras bombas defensivas, con el objetivo de destruir la mayor cantidad de misiles enemigos. La supervivencia de nuestro planeta está en sus manos.

Enunciado

El objetivo del trabajo práctico es implementar un programa que lea las mediciones de las posiciones de los misiles desde un archivo de texto, y que deje en otro archivo el momento y la posición en la que deberá detonarse cada bomba defensiva, con el objetivo de maximizar la cantidad de misiles enemigos destruidos. Debido a la urgencia que tenemos para activar nuestras defensas, el programa no deberá utilizar más de 20 sg. de procesamiento total.

El archivo de entrada contiene las posiciones de los misiles enemigos a intervalos de tiempo constantes de 1 sg. Para estimar la trayectoria futura de cada misil se deberán interpolar las mediciones de las posiciones enemigas con splines paramétricos naturales, utilizando la extrapolación de cada spline hacia el futuro como una estimación de la trayectoria correspondiente.

El programa debe tomar por línea de comandos los nombres del archivo de entrada y del archivo de salida, de la siguiente forma:

```
tp4.exe misiles.txt bombas.txt
```

El archivo con los datos de entrada (llamado `misiles.txt` en el ejemplo) tiene en la primera línea la cantidad n de misiles y la cantidad m de mediciones

[1] Dominamos los viajes interplanetarios pero nuestros radares usan la anticuada tecnología del siglo XXI.

para cada uno. A continuación, se incluyen m filas correspondientes al primer misil, cada una de las cuales contiene las coordenadas x e y de la posición del misil en los instantes $t = 1, 2, \ldots, m$ sg. (simplificamos el análisis considerando un espacio bidimensional). Luego de las m líneas correspondientes al primer misil, el archivo contiene m líneas con el mismo formato correspondientes al segundo misil, etc. Por último, el archivo contiene una línea con la cantidad b de bombas defensivas, el radio r de la onda expansiva de cada una y el radio R de nuestro planeta, cuyo centro suponemos ubicado en el origen de coordenadas. Por ejemplo, el siguiente archivo es un ejemplo de datos de entrada válidos:

```
3 4

1.0    2.5
1.1    2.4
1.23   2.3
1.27   2.15

1.9    3.5
1.8    3.4
1.7    3.2
1.75   2.9

-2.2   2.3
-2.15  2.23
-2.12  2.17
-2.08  2.11

2 0.7 1.0
```

Por su parte, el archivo de salida (llamado `bombas.txt` en el ejemplo) debe tener una línea por cada bomba, que contenga el instante de la explosión y las coordenadas x e y del centro de la explosión. Es importante notar que el punto central de cada explosión debe tener una distancia al origen de coordenadas de al menos $R + r$, para evitar que parte del planeta sea destruido por nuestras mismas bombas defensivas. Por ejemplo, si $b = 2$ el siguiente es un ejemplo de archivo de salida:

```
2.4   2.15 2.80
1.6   -2.04 2.50
```

El informe debe contener todas las opciones que el grupo haya considerado para determinar los instantes y posiciones en los que se deben detonar las bombas defensivas.

Fecha de entrega: Lunes 19 de Noviembre

Apendice B:

1. **Misil.h**

```
#include <iostream>
using namespace std;

#ifndef _MISIL_H
#define _MISIL_H

class Misil{
    friend ostream& operator<<(ostream& os, const Misil& misil);
public:
    Misil();
    Misil(const long double* medicionesX, const long double* medicionesY,
          int numMuestras);
    Misil(const Misil&);

    void operator= (const Misil& m2);
    bool estaDestruido (void) const;
    void destruir (void);
    long double posicionX (long double tiempo) const;
    long double posicionY (long double tiempo) const;

    ~Misil();
//private:
    int cantMediciones;
    bool destruido;
    long double x[4];
    long double y[4];

    void spline (const long double* muestra, long double* res);
};

#endif /* _MISIL_H */
```

2. **Misil.cpp**

```
#include "Misil.h"
#include <math.h>

Misil :: Misil()
{
    for (int i = 0; i < 3; i++){
        x[i] = 0;
        y[i] = 0;
    }

    x[3] = 1;
```

```
    y[3] = 0;
    destruido = false;
    cantMediciones = 2;
}

Misil :: Misil(const long double* medicionesX, const long double* medicionesY,
               int numMuestras)
{
    destruido = false;
    cantMediciones = numMuestras;
    spline(medicionesX, x);
    spline(medicionesY, y);
}

Misil :: Misil(const Misil& m)
{
    *this = m;
}

void Misil :: operator= (const Misil& m)
{
    for (int i = 0; i < 4; i++){
        x[i] = m.x[i];
        y[i] = m.y[i];
    }

    destruido = m.destruido;
    cantMediciones = m.cantMediciones;
}

bool Misil :: estaDestruido (void) const
{
    return destruido;
}

void Misil :: destruir (void)
{
    destruido = true;
}

long double Misil :: posicionX (long double tiempo) const
{
    return (x[0]*pow(tiempo - cantMediciones + 1, 3) + x[1]*pow(tiempo -
            cantMediciones + 1, 2) + x[2]*(tiempo - cantMediciones + 1) + x[3]);
}

long double Misil :: posicionY (long double tiempo) const
{
    return (y[0]*pow(tiempo - cantMediciones + 1, 3) + y[1]*pow(tiempo -
            cantMediciones + 1, 2) + y[2]*(tiempo - cantMediciones + 1) + y[3]);
```

```
}

Misil :: ~Misil(){}

void Misil :: spline (const long double* muestra, long double* res)
{
    /*
        La matriz para spline cubicos naturales es:
        3 0 0 0 ... 0 0 0 | 0
        1 4 1 0 ... 0 0 0 | 3*(a(2) - 2*a(1) + a(0))
        0 1 4 1 ... 0 0 0 | 3*(a(3) - 2*a(2) + a(1))
        .           .
        .           .
        .           .
        0 0 0 0 ... 1 4 1 | 3*(a(n) - 2*a(n-1) + a(n-2))
        0 0 0 0 ... 0 0 3 | 0

        siendo a(i) el coeficiente 'a' del spline 'i' y n = cantMediciones

        Al triangular desde la 2da fila hacia la anteultima mediante el metodo
        de gauss,
        el coeficiente siempre es 1/diagonal. Luego
        diagonal = 4 - 1*coeficiente.
        Solo necesitamos c(n-1), por lo que me interesa solamente el anteultimo
        valor de la diagonal.
    */

    long double diag;       //diag = resultado de la diagonal correspondiente a
                            //c(n-1)
    long double vectorB;    //el resultado correspondiente (del vector b)

    //CASO PARTICULAR: la matriz es de 2x2
    if (cantMediciones == 2){
        diag = 3;
        vectorB = 0;
    }
    else{
        diag = 4;
        vectorB = 3*(muestra[2] - 2*muestra[1] + muestra[0]);
        long double coef;    //auxiliar, para "restar fila1 - coef*fila2"

        for (int i = 2; i < cantMediciones - 1; i++) {
            coef = 1/diag;
            diag = 4 - coef;
            vectorB = 3*(muestra[i + 1] - 2*muestra[i] + muestra[i - 1]) -
                      coef*vectorB;
        }
    }

    res[1] = vectorB/diag;
```

```
    //ya tengo el c(n-1), ahora necesito a(n-1), b(n-1) y d(n-1)
    //a(j) = f(j) (sale de evaluar la ecuacion del spline en x(n-1))
    res[3] = muestra[cantMediciones - 2];

    //Ahora que tengo a(n-1) y c(n-1) puedo sacar b(n-1)
    //uso que: b(j) = a(j+1)-a(j) -(2/3*c(j) + c(j+1)/3)
    //con j = n - 1 => b(n-1) = a(n) - a(n-1) - (2/3*c(n-1) + c(n)/3)
    //pero c(n) = 0 => b(n-1) = a(n) - a(n-1) - 2/3*c(n-1)
    res[2] = muestra[cantMediciones - 1] - muestra[cantMediciones - 2] -
                    2*res[1]/3;

    //Ahora solo me falta d(n-1)
    //d(j) = (c(j+1) - c(j))/3
    //d(n-1) = (c(n) - c(n-1))/3
    //pero c(n) = 0 => d(n-1) = c(n-1)/3
    res[0] = -res[1]/3;
}

ostream& operator<<(ostream& os, const Misil& misil)
{
    os << endl;
    os << "Cantidad de mediciones: " << misil.cantMediciones << endl;
    (misil.destruido) ? (os << "Destruido: TRUE" << endl) : (os << "Destruido:
                                                        FALSE" << endl);
    os << "Coeficientes del spline en x: " << misil.x[0] << " " << misil.x[1]
       << " " << misil.x[2] << " " << misil.x[3] << " " << misil.x[4] << endl;
    os << "Coeficientes del spline en y: " << misil.y[0] << " " << misil.y[1]
       << " " << misil.y[2] << " " << misil.y[3] << " " << misil.x[4] << endl;
    return os;
}
```

3. main.cpp (Aqui se incluiran las aridades de las funciones y estructuras principales por motivos de tamaño)

```
struct Grupo {
    list<Misil*> grupoMisiles;
    long double instante;
    long double bombaX;
    long double bombaY;
};

struct Par {
    long double x;
    long double y;
};

void parser(ifstream &in, Misil **misiles, unsigned int &cantMisiles,
           unsigned int &cantMediciones, unsigned int &cantBombas,
           long double &radioPlaneta, long double &radioBomba);
```

```
short int destruidos(list<Misil*>& misiles);

inline long double distancia (const Par& p1, const Par& p2);

bool interseccionDeMediatrices(Misil** grupoMisiles, float instante,
                               long double radioBomba, Par& res);

unsigned int Armageddon(ifstream& entrada, ofstream& salida, float paso);

int main(int argc, char* argv[]);
```

Referencias

[1] http://es.wikipedia.org/wiki/Spline Definición de interpolación mediante splines.

[2] http://www.mismates.net/descargas/ejerbasicos.pdf Inscribir tirangulo en circulo

[3] http://pcm.dis.ulpgc.es/tutor/splines.pdf Condiciones para el spline

[4] http://www.ugr.es/ ossanche/docencia05_06/metnumericos/splines Alguna implementación

[5] http://docentes.uacj.mx/gtapia/AN/Unidad6/Contenido4.htm explicacion teorica

[6] http://www.cplusplus.com/ Referencia y consulta para C++

[7] http://148.204.224.249/esimetic/seminario2007/mod_01/graficasconmatlab933.pdf Graficos con Matlab.

[8] pcmap.unizar.es/ pilar/latex.pdf . Introduccion a latex.

[9] http://oc.wikipedia.org/wiki/Ajuda:Formulas_TeX_e_LaTeX Formulas Latex.

[10] webs.uvigo.es/mat.avanzadas/PracME_1.pdf Introducción a MatLab.