Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Métodos Numéricos

Trabajo Práctico Nº2

Guerra lineal

Nombre	LU	Mail
Carla Livorno	424/08	carlalivorno@hotmail.com
Mariano De Sousa Bispo	389/08	marian_sabianaa@hotmail.com

Abstract

El siguiente trabajo se propone la implementación de una estrategia para la simulación de una guerra espacial usando resolución de sistemas lineales. El objetivo es calcular las coordenadas para efectuar un disparo tal que impacte en la nave enemiga, y a la vez se pretende no dar a conocer nuestra posición en el espacio mediante el disparo.

Sistemas lineales **Simulación** **Disparo** **Nave** **Cañón**

Índice

1. Introducción teórica

Multitud de problemas de la vida real se rigen por proporciones constantes entre las magnitudes implicadas: procesos físicos, reacciones químicas, costes de materias primas y sus relaciones para formar otros producto, etc. Todas estas situaciones admiten de forma natural una descripción matemática a través de sistemas de ecuaciones lineales. Además estos sistemas son útiles como una buena aproximación a sistemas más complejos o sistemas de ecuaciones no-lineales.

La resolución de muchos problemas conlleva el manejo de grandes sistemas de ecuaciones lineales, por lo que es necesario plantearse métodos eficientes para su análisis.

Los sistemas de ecuaciones lineales son un conjunto de m ecuaciones relacionadas en n variables. Se los puede expresar en forma matricial como $A\dot{x} = b$ donde A es una matriz de $m \times n$, x es un vector columna de tamaño n y b un vector de tamaño m.

Existen diversos métodos para la resolución de estos sistemas de ecuaciones. Haremos una breve introducción a los distintos métodos.

Tenemos por un lado los métodos iterativos como es el caso del método de Jacobi, Gauss-Seidel o del Gradiente Conjugado que como su nombre lo indica se los aplica en forma iterativa para lograr una aproximación a la solución real del sistema en cuestión

Por otra parte, existen los denominados métodos directos entre los cuales se encuentra el método de eliminación de Gauss el cual es una forma directa para llegar en un número finito de pasos a un sistema equivalente pero mas simple, la factorización LU que se vale de este último, la descomposición de Cholesky, y la descomposición QR. [1]

[1] Burden y J.D.Faires, Análisis numérico, International Thomson Editors, 1998

2. Desarrollo

2.1. Explicación

2.1.1. Introducción al problema

La guerra lineal consiste en dos naves situadas en el hiperespacio de n dimensiones con el objetivo de desintegrarse mutuamente mediante disparos de cañones $Warp$.

Cada turno del combate consiste en lo siguiente:

1. Nos proveen la matriz A' usada por el oponente para atacarnos y el punto d' donde impactó el proyectil. Con esa información debemos intentar descubrir la posición y donde se encuentra situada la nave enemiga, es decir, resolver el sistema $A'y = d'$.

2. Tenemos la posición a la que pretendemos disparar (d) y queremos conseguir una matriz de ataque A tal que $Ax = d$ donde x es nuestra posición.

2.1.2. Estrategia

En cada turno nuestra estrategia consiste en lo siguiente:

1. Para resolver el sistema $A'y = d'$ decicimos usar métodos directos de resolución de sistemas de ecuaciones lineales.

   - Opción 1: Buscamos la matriz inversa de A' (existe porque es parte de las reglas de la batalla) mediante el método de eliminación Gaussiana. Es decir, efectuamos a la matriz A' distintas trasformaciones lineales hasta convertirla en la matriz identidad. Al mismo tiempo hacemos dichas trasformaciones a la matriz identidad quedando en esta la inversa de A'. Las transformaciones consisten en triangular superior e inferiormente A' y poner *unos* en la diagonal. Una vez que conocemos la inversa (A'^{-1}) podemos hallar la posición buscada de la siguiente manera: $y = A'^{-1} * d'$.

   - Opción 2: Hallamos la factorización LU de A' (si no existe, conseguimos PLU). Resolvemos el sistema $LUy = d'$. LLamamos

$Ux = z$ y resolvemos usando sustitución hacia adelante el sistema $Lz = d'$. Una vez que conocemos z obtenemos x tal que $Ux = z$ utilizando sustitución en reversa. (Con PLU es el mismo procedimiento pero permutando d' según P).

Elegimos este método ya que las matrices son cuadradas y la implementación es sencilla.
PONER MAS MOTIVOS COMO COMPLEJIDAD!!

La exactitud con la que calculemos la posición buscada y depende del número de condición de la matriz A', cuanto mayor es este número menor es la precisión del resultado obtenido.

Como creemos que el adversario va a utilizar matrices mal condicionadas para atacarnos, en cada turno estariamos calculando una posición errada de donde se encuentra. Por este motivo, decidimos usar la estrategia de disparar en el punto promedio de todos las posiciones calculadas hasta el momento de donde se situa. Esperamos de esta manera ir obteniendo cada vez mejores aproximaciones a donde se encuentra realmente.

2. Para atacar necesitamos una matriz A que al multiplicarla por nuestra posición resulte en un impacto en la posición deseada d. Es decir, necesitamos que A cumpla $Ax = d$.

   Sea n la dimensión de A. Dado que tenemos n^2 incognitas y n ecuaciones, tenemos $n^2 - n$ elementos de A libres. Por este motivo, podemos elegir A convenientemente y luego adaptarla (forzarla a cumplir $Ax = d$).

   Como debemos proveer al adversario de la matriz A que usemos para el ataque tenemos que idear una estrategia para que nuestra posición no sea descubierta en el siguiente turno. La estrategia consiste en generar matrices mal condicionadas que a su vez sean inversibles (esto último por reglamento).

   Para crear una matriz mal condicionada, generamos un único vector aleatorio de dimensión n que resulta en las filas de dicha matriz. De esta manera tenemos una matriz cuyas filas son linealmente dependientes, por lo que la matriz no es inversible. Luego, sumamos a los elementos

de la diagonal un *epsilon* (QUE ELEGIMOS BLA BLA BLA BLA) obteniendo asi una matriz inversible y mal condicionada (JUSTIFICAR PORQUE).

Por último, para forzarla a satisfacer el sistema elegimos una columna j de A tal que el $j - esimo$ elemento de x sea distinto de *cero* (x no es el vector nulo por precondición ya que es nuestra posición), sino los coeficientes de A que pretendemos setear no tienen importancia, y consideramos incognitas sólo a los coeficientes de esa columna. Nos quedan así, n incognitas y n ecuaciones, por lo que podemos despejarlas y obtener la matriz final.

La matriz obtenida sigue siendo inversible ya que modficamos sólo una columna de la misma y lo que hace linealmente independiente a las filas son los elementos de la diagonal.

2.2. Implementación

En primer lugar el desarrollo consistio en implementar módulos para las operaciones entre Matrices y Vectores que fuesen necesarias para resolver sistemas lineales. Esto implicaba también implementar el algoritmo de factorización LU y el algoritmo de eliminación Gaussiana ambos con pivoteo parcial para resolver el sistema lineal.

Escribimos el módulo `Matrix` que implementa una matriz en $\mathbb{R}^{nxn}$ con las siguientes operaciones:

`Eliminación Gaussiana`	Eliminacion Gaussiana con pivoteo parcial.
`LU`	Factorización LU con pivoteo parcial.
`Inversa`	Matriz inversa con el método de Gauss.
`K`	Número de condición.

`Suma, Resta, Multiplicación, Multiplicación por escalar, traspuesta`

Usamos pivoteo parcial tanto para la eliminación Gaussiana como para la factorización LU para reducir el error de redondeo en las operaciones.

Además escribimos la clase `Vector` que implementa un vector en $\mathbb{R}^n$ con las operaciones básicas como `Suma, Resta, Producto Escalar, Producto Vectorial, Normas`, etc.

NOTA: Ambas clases heredan de una clase `MatrixBase` que implementa las funciones básicas como son la suma, resta, multiplicación, etc.

Teniendo la base (matrices, vectores y las operaciones entre ellos) implementamos una clase para la resolución de sistemas de ecuciones lineales y una clase para elegir la matriz de ataque.

El módulo `linearSystem` implementa las siguientes funciones:

`Resolución por inversa`	consigue la inversa y hace la multiplicación.
`Resolución por LU`	consigue LU (PLU) y despeja el sistema usando sustitución hacia adelante y sustitución reversa.

El módulo `warpCannon` implementa las siguientes funciones:

`Atacar`	consigue la posición de impacto deseada y arma la matriz correspondiente.

Esta clase tiene un método privado que calcula la posición del enemigo.

Nuestra primer idea fue hacer un promedio coordenada a coordenada dandole una prioridad distinta a cada vector (posición), dicha prioridad dependia del número de condición de la matriz que uso el oponente en ese turno. Es decir, y tiene mayor prioridad que z si y sólo si $K(A') < K(A)$ siendo $A'y = d'$ y $Az = d$.

Para lograr esto, teniamos un *array* de *tuplas* donde la primer coordenada correspondia al vector y la segunda al número de condición de la matriz asociada.

La idea finalmente implementada es una versión más simple (por cuestiones de tiempo), que sólo toma el promedio sin asignar prioridades. Igualmente quedo implementado con *tupla* (ignorando la segunda coordenada).

2.3. Ideas descartadas

Nuestra primer estrategia consistía en atacar con la matriz de Hilbert de orden n una cierta cantidad de turnos (que pensabamos decidir mediante pruebas) y guardar la posición del enemigo calculada en cada turno. Una vez cumplida esa cantidad de turnos, sacar un promedio entre la información obtenida y disparar a esa posición sin ningun tipo de defensa.

La ventaja de esta estrategia es que evitabamos tener que generar matrices

mal condicionadas (ya que usabamos siempre la de Hilbert).

La desventaja es que dado que estariamos atacando siempre con la misma matriz o un múltiplo de esta, el oponente estaria siempre calculando la misma posición (punto de ataque) y no generariamos nueva información al recibir los datos (en realidad depende de como genere la matriz de ataque el oponente a partir de tener el punto de ataque).

3. Resultados

4. Discusión

5. Conclusiones

6. Apéndices

6.1. Apéndice A: Enunciado

Laboratorio de Métodos Numéricos - Primer cuatrimestre 2011
Trabajo Práctico Número 2: La Guerra Lineal, Episodio 5 ...

Introducción

Corría el año 4957 de la era de paz iniciada luego de la expulsión de los mutantes invasores en el sistema solar de la estrella HAL-9000 cuando los sistemas de defensa alertaron a la comandancia por la intrusión de naves espías provenientes de lejanas galaxias dominadas por especies hostiles. Luego de estas primeras incursiones comenzaron los devastadores ataques alienígenas. Las fuerzas de dicho sistema solar se enfrentaban al ataque de un enemigo muy superior, que amenazaba con conquistar a todos los planetas del sistema y comenzar así una avanzada sobre la República Galáctica.

Los sistemas de defensa convencionales pronto fueron insuficientes para resistir el ataque y, además, las inspecciones telemétricas dieron cuenta de que, fuera de los límites del sistema solar, el enemigo estaba ensamblando un Cañón Warp, el arma más devastadora jamás construida por seres inteligentes. Los registros de la República solamente consignaban cuatro ataques previos por medio de este tipo de armas: en el primer ataque al sistema Z-80[2], en la batalla de la estrella 80286[3], en las guerras de los clones de PCs[4], y en el segundo ataque al sistema Z-80[5]. La única resistencia posible contra un Cañón Warp es utilizar otro Cañón Warp, con lo cual se hizo un llamado desesperado a las fuerzas de línea de la República para responder a este ataque. En menos de dos semanas el Cañón Warp del Ejército de la República estaba preparado para dar batalla, y el combate decisivo estaba por comenzar...

La Guerra Lineal

[2]La Guerra Lineal, Episodio 1: TP1 del segundo cuatrimestre de 2000.
[3]La Guerra Lineal, Episodio 2: TP2 del primer cuatrimestre de 2001.
[4]La Guerra Lineal, Episodio 3: TP2 del primer cuatrimestre de 2005.
[5]La Guerra Lineal, Episodio 4: TP2 del segundo cuatrimestre de 2008.

La batalla entre dos Cañones Warp no es un combate entre bandoleros espaciales, sino una caballerosa justa algebraica entre dos naves que respetan ciertas reglas. Para no ser directamente visible, cada nave se sitúa en un punto del hiperespacio de n dimensiones. Llamemos $x \in \mathbb{R}^n$ e $y \in \mathbb{R}^n$ a los puntos donde se ubican la primera y la segunda nave, respectivamente.

A intervalos regulares, cada nave realiza un disparo de su Cañón Warp con el objetivo de desintegrar al enemigo. Debido a que nos encontramos en el hiperespacio, la única forma de dirigir el disparo hacia la posición de la nave enemiga es mediante una transformación del espacio–tiempo, llamada *transformación warp*. Esta transformación modifica el espacio–tiempo de manera tal que la bomba lanzada por la nave en la posición x y sometida a la transformación warp A "explotará" en la posición $d = Ax$ luego del disparo. Una transformación warp está dada, entonces, por una matriz *inversible* $A \in \mathbb{R}^{n \times n}$. La nave enemiga es destruida si y solo si $\|d - y\|_2 \leq 1$.

Ahora bien, supongamos que la nave y no es destruida. Esta nave puede ver el punto d donde se produjo la explosión y, analizando las distorsiones del espacio–tiempo sobre el fondo de estrellas fijas, también puede deducir la transformación warp A que usó el enemigo. Entonces, al menos en teoría, puede resolver el sistema $Ax = d$ para averiguar la posición x del enemigo y eliminarlo en su siguiente disparo. Aquí entra en juego la habilidad de cada contendiente para evitar ser fácilmente descubierto luego de cada disparo propio, y para descubrir la ubicación del rival luego de cada disparo enemigo.

La Guerra Lineal consiste en una sucesión de disparos simultáneos de cada contendiente, hasta que uno de ellos es destruido (o ambos son destruidos al mismo tiempo). Luego de cada disparo, cada nave recibe la información telemétrica de la posición del disparo enemigo y la transformación warp que usó el adversario, con lo cual podrá ajustar su siguiente disparo para acercarse más a la nave enemiga.

Detalles y requerimientos

El objetivo del trabajo práctico es implementar un Control de Cañón Warp: un programa que desarrolle automáticamente una batalla de la Guerra Lineal contra el programa de otro grupo, de acuerdo con las siguientes especificaciones.

Cada ejecución del programa corresponde a un disparo del cañón. Cada programa se ejecuta por línea de comandos y toma como parámetros los nombres de varios archivos. Algunos archivos contienen la posición de la nave y los disparos enemigos, y otro archivo contendrá la salida del programa.

Una vez leído el o los archivos de entrada el programa debe seleccionar una transformación warp $A \in \mathbb{R}^{n \times n}$ y realizar un disparo $d = Ax$. A partir del segundo turno el programa contará con información que le permitirá inferir la posición del oponente y podrá elegir una transformación de manera que el disparo se acerque a la nave enemiga. Una vez decidida la transformación y calculado el disparo el programa debe informar los mismos en el archivo de salida.

Luego de escribir este archivo, el programa debe detener su ejecución y será vuelto a ejecutar cuando llegue el momento de realizar un nuevo disparo (en el siguiente turno). Este proceso continúa hasta que alguno de los dos contendientes es eliminado. Cualquier información que sea necesario registrar entre disparos se debe almacenar en archivos auxiliares (en el "directorio actual"), a criterio del grupo.

Los programas serán invocados en el primer turno con la siguiente línea de comandos:

```
<programa> <posicion> <salida>
```

y del segundo turno en adelante, una vez que se cuenta con los datos que generó el programa rival, serán invocados con la siguiente línea de comandos:

```
<programa> <posicion> <salida> <ultimo> <anteriores>
```

Por ejemplo:

```
lineal.exe datos.txt disparo.txt d001.txt otros.txt
```

De acuerdo a los siguientes detalles:

programa Es el programa de cada grupo.

posicion Es un archivo con el siguiente formato:

- La primera línea contiene un cero[6].
- La segunda línea contiene el valor de n, especificando la cantidad de dimensiones del espacio en el que se juega.
- La tercera línea contiene la posición del jugador, especificada por n valores separados por espacios.

Por ejemplo, el siguiente es un archivo válido de entrada:

```
0
2
-9.5101358720664081000e+001 3.1796508444471566000e+001
```

salida Es el archivo que debe generar el programa, conteniendo la transformación warp elegida y el disparo resultante. Su formato debe ser el siguiente:

- La primera línea debe contener el número del turno en que se efectuó el disparo (el primer turno es el número 1).
- La segunda línea debe contener la dimensión n (este valor ya es conocido, pero se incluye en el archivo para verificar que no haya inconsistencias).
- La tercera línea debe contener las n coordenadas del vector d, separadas por espacios.
- Las siguientes n líneas contienen los coeficientes de la matriz A, ubicando en cada línea una fila completa de la matriz (con los coeficientes separados por espacios).

ultimo Contiene el disparo generado por el programa oponente en el turno anterior, con el formato descripto anteriormente.

anteriores Es una concatenación de todos los archivos de disparos del oponente, salvo el último. En el segundo turno es un archivo vacío. En el tercer turno contiene el disparo del turno 1, en el cuarto turno contiene el disparo del turno 1 seguido del disparo del turno 2, etc.

[6]Por razones históricas.

En todo momento el combate estará supervisado por un programa que oficiará de árbitro entre los dos programas en juego (por razones históricas, este árbitro será llamado La Fuerza), que se encarga de ejecutar ambos programas y de pasarles como parámetros los archivos que correspondan en cada caso.

Para asegurar el desarrollo normal de la competencia, el programa jugador se debe implementar bajo el sistema operativo Windows, de manera que sea posible su ejecución en las máquinas del Laboratorio 6.

El combate debe desarrollarse en aritmética de punto flotante de doble precisión (8 bytes, correspondiente al tipo de datos `double`). La Fuerza tomará los datos como `doubles`, con lo cual es importante tomar los recaudos necesarios para la lectura y escritura de los archivos. Todos los archivos generados, tanto por La Fuerza como por los contrincantes, serán archivos ASCII conteniendo sólo números, espacios y fin de línea. Todos los números de punto flotante serán escritos usando 20 dígitos y notación científica[7], para evitar errores de representación.

Comentarios destacables

Es importante que el programa implementado cuente con estrategias inteligentes de defensa, que no le permitan al enemigo deducir fácilmente nuestra posición. Por ejemplo, no sería muy inteligente utilizar como transformación warp una matriz diagonal, dado que en este caso el enemigo puede encontrar nuestra posición con mucha precisión. El informe debe contener todas las alternativas que el grupo haya considerado con relación a este punto, junto con una discusión que justifique la opción finalmente adoptada. Además, el informe debe justificar que la matriz generada en cada turno es inversible. También es importante que el programa esté preparado para actuar ante las posibles estrategias de defensa del enemigo. El programa debe contemplar que el enemigo puede estar generando transformaciones warp que dificulten nuestro análisis de su posición, y debe implementar algún mecanismo que intente manejar esta situación. No se aceptarán transformaciones warp que no sean inversibles, disparos que están mal calculados, o el uso indebido de espadas láser entre los contrincantes. Recuerden que la Guerra Lineal es un combate entre caballeros.

[7] Una opción es `precision(20)` y `setf(std::ios_base::scientific,std::ios_base::floatfield)` de `std::ostream`. Ejemplo: -1.64741740540787990000e+002 .

Entregas parciales

6 de mayo: Implementación del (o de los) método(s) de resolución de sistemas de ecuaciones.
13 de mayo: Implementación completa y casos de prueba.

Entrega Final

Formato Electrónico: 19 de mayo de 2011, hasta las 23:59 hs, a la dirección:
metnum.lab2011@gmail.com
Formato físico: 20 de mayo de 2011, de 17 a 21 hs.
Batalla lineal entre grupos: 20 de mayo de 2011, de 20 a 21 hs.

6.2. Apéndice B: Código fuente

6.3. Modo de compilación

Para compilar se hace uso de la herramienta Makefile.

Abrir una terminal dentro la carpeta *code* entregada y escribir el comando "make", pulsar enter.

6.4. Modo de uso

Una vez compilado escriba en la terminal (posicinado sobre la misma ruta en la que lo compiló)

7. Referencias