Trabajo Práctico 2: Estamos en el horno

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)

29 de Septiembre, 2007

Resumen

RESUMEN.

Palábras clave:

- Eliminacion gaussiana.
- Pivoteo parcial.
- Diferencias finitas.
- Laplaciano.
- Metodos directos.

Introducción:

Si bien la matemática no es una ciencia aplicada, es frecuentemente utilizada en la resolución de problemas de la industria o la ingeniería, entre otras disciplinas. Estas utilizan herramientas matemáticas para modelar problemas de la vida real, que frecuentemente involucran enormes cantidades de datos y variables, además de tener que obedecer comportamientos específicos según cada caso.

Para reproducir fielmente estos problemas la matemática emplea funciones, expresadas en sistemas de ecuaciones sobre las que se vuelcan todos los datos, y de las que se pretende obtener valores de incógnitas que permitan la proyección hacia el futuro de los comportamientos emulados.

Dadas las dimensiones de estos sistemas, es normal la utilización de herramientas informáticas en su manejo. Cuando los problemas son demasiado grandes o se manifiestan en conjuntos densos de valores, se lleva a cabo una discretización, que los vuelve resolubles para una computadora.

Los métodos que se emplean en dichas resoluciones pueden ser de dos tipos: directos o iterativos.

En este trabajo se utilizo un método directo, en particular el método de Gauss, uno de eliminación, y para obtener los valores se empleo la sustitución hacia atrás.

Los métodos directos de resolución de sistemas lineales de ecuaciones son aquellos que permiten obtener la solución después de un número finito de operaciones aritméticas. Este número de operaciones es, obviamente, función del tamaño de la matriz.

Si las computadoras pudieran almacenar y operar con todas las cifras de los números reales, es decir, si emplearan una aritmética exacta, con los métodos directos se obtendría la solución exacta del sistema en un número finito de pasos.

Debido a que las computadoras trabajan con precisión finita, los errores de redondeo se propagan y la solución numérica obtenida siempre difiere de la solución exacta. La cota del error, para una matriz y término independiente dados, se asocia por lo general al número de operaciones de cada método. Se pretende, por lo tanto, obtener métodos con el mínimo número de operaciones posible.

Una particularidad de los métodos directos es que siempre conducen, después de ciertas operaciones, a la resolución de uno o varios sistemas con solución inmediata. Es decir, sistemas donde la matriz es diagonal o triangular.

En el método de eliminación de Gauss el problema original, Ax = b, se transforma mediante permutaciones adecuadas y combinaciones lineales de cada una de las ecuaciones en un sistema de la forma Ux = c donde U es una matriz triangular superior. Los elementos en la diagonal son llamados pivotes y se utilizan para eliminar a los elementos que se encuentran en su misma columna pero en filas mayores (es decir, los elementos bajo el pivote). Para esto se realiza el siguiente algoritmo:

$$E_j = E_j - \frac{a_{i,j}}{a_{i,i}} * E_i$$

Como se puede apreciar cuando el pivote es cero el algoritmo no puede llevarse a cabo, con lo que es necesaria una permutación entre las filas por anular para obtener un pivote distinto de cero.

Esta permutación de filas no sólo tiene interés cuando el pivote es exactamente cero. Es obvio que valores pequeños del pivote pueden producir grandes errores de redondeo, ya que siempre se divide por el valor del pivote.

Por consiguiente, para reducir los errores de redondeo conviene escoger el pivote máximo en valor absoluto. Para ello, hay dos técnicas posibles:

Se toma como pivote el coeficiente mayor en valor absoluto de la columna k situado por debajo de la fila k inclusive. Para ello es necesario permutar las filas k y la correspondiente al pivote escogido en la matriz y su término independiente. Esta técnica se denomina método de Gauss con pivoteo parcial. Existen otras metodologías de pivoteo pero esta fue la empleada durante el presente trabajo.

La variante es extender la búsqueda del coeficiente de mayor módulo no solo a la columna k, sino a toda la matriz no triangulada, es por ello que esta técnica se conoce como pivoteo total.

Tras aplicar el algoritmo de Gauss a todas las filas del sistema nos queda una matriz triangular superior. Este nuevo sistema equivalente al original es de resolución inmediata, sólo es necesario aplicar el algoritmo de sustitución hacia atrás.

En cuanto a los métodos iterativos para resolver ecuaciones son aquellos en los cuales una primera aproximación es usada para calcular una segunda aproximación, la cual a su vez es usada para calcular una tercera aproximación, y así sucesivamente.

Desde un punto de vista general las matrices más usuales en las ciencias aplicadas y en ingeniería pueden englobarse en dos grandes categorías:

- Matrices llenas, pero no muy grandes. Por llenas se entiende que poseen pocos elementos nulos y por no muy grandes que el número de ecuaciones es de unos pocos miles a lo sumo. Estas matrices aparecen en problemas estadísticos, matemáticos, físicos e ingenieriles.

- Matrices vacías (ralas) y muy grandes. En oposición al caso anterior, vacías indica que hay pocos elementos no nulos y además están situados con una cierta regularidad. En la mayoría de estos casos el número de ecuaciones supera los miles y puede llegar en ocasiones a los millones. Estas matrices son comunes en la resolución de ecuaciones diferenciales de problemas de ingeniería.

Parece lógico que los métodos para resolver sistemas lineales de ecuaciones se adecuen a las categorías de matrices anteriormente expuestas. En general los métodos directos se aplican al primer tipo de matrices, mientras que los métodos iterativos se emplean con el segundo grupo. Es importante observar que no existen reglas absolutas y que todavía en la actualidad existe cierta controversia sobre los métodos óptimos a aplicar en cada caso.

Desarrollo:

El problema particular a resolver en este trabajo es el comportamiento de un horno de acero, en lo que corresponde a la temperatura de su parte exterior. Este horno puede dividirse en tres 'capas', la primera, el horno propiamente dicho, esta a una temperatura constante y esta es un dato conocido; la segunda es la parte intermedia, la pared interior, su forma es arbitraria y viene dada por una función r, también conocida; la última capa es la pared exterior, y de ésta se desea conocer la temperatura.

Por ser el horno circular (con radio conocido), para su representación se utilizaron coordenadas polares.

Se sabe que la temperatura se corresponde con una función T, para cualquier ángulo y radio. La función es desconocida, con lo que para conocer la temperatura de la pared exterior es necesario hallarla. De T se sabe que cumple con tres ecuaciones:

1. $\frac{\partial^2 T(r,\theta)}{\partial r^2} + \frac{1}{r}\frac{\partial T(r,\theta)}{\partial r} + \frac{1}{r^2}\frac{\partial^2 T(r,\theta)}{\partial \theta^2} = 0$
2. $T(r_i(\theta),\theta) = T_i$
3. $-K\frac{\partial T(r_e,\theta)}{\partial r} = h(T(r_e,\theta) - T\infty)$

Siendo el dominio de T las coordenadas polares y R su imagen, se procede a realizar una discretización de la función, para poder aproximar las derivadas de las ecuaciones 1 y 3 por el método de las diferencias finitas. Como resultados, se obtuvieron las siguientes aproximaciones de las derivadas de T:

$$\frac{\partial^2 T(r,\theta)}{\partial r^2}(r_j,\theta_k) \cong \frac{t_{j-1,k} - 2t_{j,k} + t_{j+1,k}}{(\Delta r)^2}$$
$$\frac{\partial T(r,\theta)}{\partial r}(r_j,\theta_k) \cong \frac{t_{j,k} - t_{j-1,k}}{\Delta r}$$
$$\frac{\partial^2 T(r,\theta)}{\partial \theta^2}(r_j,\theta_k) \cong \frac{t_{j,k-1} - 2t_{j,k} + t_{j,k+1}}{(\Delta \theta)^2}$$

Con estas aproximaciones y las ecuaciones antes detalladas se realizó el cálculo de la función T en cada punto de la discretización de la siguiente manera:

1. Con las ecuaciones 1 y 3 se despejaron coeficientes para cada una de las variables. De esta forma cada punto de la discretización puede ser categorizado según las tres partes del horno mencionadas más arriba. Así, contamos con puntos del horno propiamente dicho, cuya temperatura es conocida (la temperatura interior del horno), puntos de la pared interior y los puntos del perímetro del horno, o de la pared exterior del mismo.

   Los puntos de la pared interior se presentan como los más complicados de manejar ya que luego de algunas manipulaciones algebraicas se encuentran expresados en función de otros cuatro puntos aledaños.

$$t_{j,k}\left(\frac{-2}{(\Delta r)^2} + \frac{1}{r\Delta r} - \frac{2}{r^2\Delta\theta^2}\right) + t_{j-1,k}\left(\frac{1}{(\Delta r)^2} - \frac{1}{r\Delta r}\right) + t_{j+1,k}\frac{1}{(\Delta r)^2}$$
$$+ t_{j,k+1}\left(\frac{1}{r^2(\Delta\theta)^2}\right) + t_{j,k-1}\left(\frac{1}{r^2(\Delta\theta)^2}\right) = 0$$

En cuanto a los puntos dentro del horno, ya se encuentran 'resueltos', expresados solo en función de la temperatura interior. Los puntos de la pared exterior quedan determinados por la ecuación de flujo, en función de un punto anterior, la temperatura exterior y las constantes térmicas H y K.

$$t_{r_e,\theta} = \frac{\frac{K*t_{r_{e-1},\theta}}{H\Delta r}+T_\infty}{1+\frac{K}{H*\Delta r}}$$

2. Tras esta caracterización de cada punto interviniente se construye una matriz donde se vuelca toda esta información, con una ecuación para cada punto. La matriz, que será el eje de la resolución del problema, tiene dimensión cantidadDePuntosDiscretizados*cantidadDePuntosDiscretizados, y recopila todas las ecuaciones despejadas en el punto anterior.

3. Se puede apreciar gracias a las ecuaciones anteriores que la forma de la matriz resultante es predecible y se corresponde con la estructura del horno, ya que tiene una primera sección que asemeja la matriz identidad, con tan solo unos en la diagonal y ceros en el resto de las filas. Se nota un segundo grupo de filas, que atañen a las variables de la pared interior del horno, estas filas tienen cinco coeficientes distintos de cero en un patrón definido, que asemeja una matriz banda un tanto peculiar en cuanto a su distribución. Esta alterna unas diagonales con ceros, una diagonal de coeficientes, otras diagonales de ceros, tres diagonales de coeficientes, unas diagonales más de ceros, una última diagonal de coeficientes y unas diagonales de ceros que terminan la matriz. Tras esta sección que podría catalogarse como pseudo-banda, se hallan las ecuaciones que describen los puntos de la pared exterior del horno, esta última serie de filas tiene tan solo dos coeficientes distintos de cero (y además son consecutivos), el correspondiente al punto descrito y su inmediato anterior en la sucesión de radios.

   Nótese que el "b"de nuestro sistema se representa en otra matriz, organizada de igual forma que la principal, con lo que su primera parte es una columna que lleva el valor de la temperatura interior del horno, en la segunda solo hay ceros y en la tercera y última el valor de la ecuación de flujo. Con este sistema nos encontramos en condiciones de calcular todos los valores de T dentro de la discretización.

4. La resolución del conjunto de ecuaciones anterior se lleva a cabo en dos pasos: triangulación , a través de eliminación gaussiana con pivoteo parcial; y sustitución hacia atrás para despejar la solución pertinente a cada ecuación. Se volverá sobre este punto más adelante.

   Con esto se obtiene un vector columna que responde al x (en una tercera matriz) de nuestro sistema Ax = b, que se utiliza para realizar los gráficos de distribución de la temperatura pertinentes.

Detalles de la implementación:

Para este trabajo se implementaron dos clases (de C++):

La clase Matriz, que modela un sistema de ecuaciones en forma de matriz de long doubles, con los métodos necesarios para crear matrices (que se inicializan con sus coeficientes en cero), asignarles valores a cada posición, ver dichos valores, triangular (incluye un método privado para el pivoteo parcial) una matriz ya creada, así como resolverla mediante la sustitución hacia atrás.

En su estructura esta clase esta sustentada por un arreglo de arreglos de long double, creados en forma dinámica durante la ejecución del programa. En cuanto al lenguaje C++, un arreglo no es más que un puntero a un espacio de memoria contiguo, con lo que esta implementación, bien puede pasar por un puntero a un puntero a long double. De hecho, esta es la forma en que C++ implementa su propio tipo nativo "matriz", como un arreglo que tiene dos índices, uno de los cuales se utiliza con un multiplicador (para moverse por las filas) y otro como 'off set' (para ubicar elementos en las columnas). Se decidió implementar una nueva clase para lograr una mayor abstracción y modularidad en el código, además de un ocultamiento de información delimitado por la parte pública y la privada. Además de este puntero a puntero la clase cuenta con otros atributos, fil y col que denotan el número de filas y columnas respectivamente. Recordar que en C++ no se puede conocer en forma directa la dimensión de un arreglo.

Como sugerencia para una próxima revisión de este trabajo resta explorar la posibilidad de utilizar la librería valarray, parte de la librería STL (librería estándar de plantillas), optimizada para computación numérica y manejo de matrices en su aspecto matemático.

Nota: a partir de este punto siempre que se emplee el término 'matriz', éste se refiere a la clase implementada y no al tipo nativo de C++.

En cuanto a los métodos de la clase podemos enumerar:

- Constructor: crea en tiempo de ejecución la estructura mencionada, inicializando todos los valores de long double en cero. La dimensión de la matriz viene dada por los dos parámetros de la función, si éstos no son completados se crea por defecto una de 4x4.

  Se cuenta también con un constructor por copia y un operator=, el primero utiliza fuertemente al segundo. El operador, copia los atributos fil y col de la matriz de la derecha en los de la izquierda, elimina la matriz "pisada"por la igualación y luego crea una nueva, en la que copia los valores de la original.

- filas() y columnas(): se implementaron sendos métodos para conocer tanto el número de filas como de columnas de la matriz creada. Tan solo devuelven una copia de los atributos privados de la clase. Lleva una cláusula const de forma que no se pueda modificar la matriz con este método.

- Ver(): este método requiere dos parámetros que se utilizan, el primero para indexar una fila, y el segundo para hacer lo propio sobre una columna y

se devuelve el elemento correspondiente de la matriz. Lleva una cláusula const de forma que no se pueda modificar la matriz con este método.

- Asignar(): éste también requiere dos parámetros iguales a los de Ver, y un tercero, un long double que será el nuevo valor de la posición seleccionada. Tanto este método como Ver, lanzan una excepción si se le pasan parámetros por fuera del rango de filas o de columnas de la matriz.

- Triangular(): recibe un puntero a una matriz 'b', con lo que la triangulación se puede llevar a cabo solo con la matriz o extenderlo para sistemas con una solucion b. Utiliza los métodos privados pivotear y restarFilas, para emular el método de resolución de sistemas de ecuaciones lineales conocido como eliminación gaussiana. De esta manera transforma nuestro sistema en uno triangular superior, susceptible de ser despejado por "sustitución hacia atrás". Se comentan primero los métodos pivotear y restarFilas para luego volver sobre éste.

- Pivotear(): recibe un entero que denota la fila en la que se encuentra el algoritmo de Gauss, y busca mediante pivoteo parcial (ya explicado en la introducción) el mejor pivote disponible.

  Se implementó este algoritmo en lugar del pivoteo total por ser más sencillo, además resta saber si la ganancia en precisión supera la perdida en tiempo que representa la búsqueda a lo largo de toda la sub-matriz, en lugar de tan solo la columna pertinente.

- RestarFilas(): recibe un long double que es el coeficiente del pivote, y dos enteros, la fila a la que hay que anularle el valor de la columna del pivote, y la fila donde se encuentra el pivote. Este procedimiento se corresponde con el algoritmo enunciado en la introducción.

  Este método se limita aplicar restarFilas luego de aplicar pivotear en la fila que se corresponde con el i-esimo paso de la triangulación.

  Se eligió la eliminación gaussiana para obtener un sistema de resolución directa porque es el más conocido y estudiado, además, no tiene un costo temporal exagerado (es de orden cúbico), ni requiere memoria adicional a la necesaria para almacenar la matriz, tiene una implementación sencilla y se adapta perfectamente a cualquier matriz cuadrada.

Queda un aspecto muy importante a mejorar en futuras versiones de este programa: ni el algoritmo que implementa la eliminación gaussiana, ni la clase matriz, aprovechan la particular estructura antes mencionada que rige el sistema resultante del volcado de los datos y su tratamiento algebraico. Este conocimiento del patrón que siguen las variables en la matriz podría ser utilizado para idear una estrategia de almacenamiento y mapeo de índices que permita ahorrar memoria no conservando los ceros. Se podría utilizar también para mejorar el rendimiento del algoritmo de triangulación, conservando la eliminación gaussiana.

Tampoco se pudo implementar un algoritmo que reduzca el espacio en memoria que ocupa la matriz triangulada, que podría ser acortado hasta un poco más de la mitad del consumo original.

Todas estas optimizaciones se discutieron durante la etapa de implementación, y se descartaron, en pos de cumplir con la fecha limite asignada durante la presentación del proyecto.

La segunda clase que interviene en este trabajo es la que utiliza la matriz y sus métodos, y que modela el horno a través de una matriz de ecuaciones que describen los valores discretizados de la ya mencionada función T. Entre sus atributos esta clase cuenta con todos los datos necesario para poder resolver el problema planteado. A saber:

- Temperaturas: una matriz con los valores de la función T ya discretizados y despejados. A pesar de que ésta es un atributo de la clase su obtención requiere un arduo trabajo de procesamiento.
- rad: un long double que representa el radio exterior del horno completo, desde el centro hasta la pared exterior.
- angs y rads: son enteros, la cantidad de ángulos y radios en los que se discretiza el horno. Recordar que se utilizan coordenadas polares.
- deltaR y deltaT: long doubles, la diferencia entre los diferentes radios y ángulos respectivamente. deltaT se expresa en radianes.
- ti y tinf: ambos enteros. Ti es la temperatura dentro del horno propiamente dicho, "donde se funde el acero", mientras que tinf es la temperatura del espacio no delimitado que esta afuera del horno, más allá de la pared exterior. Tanto ti como tinf están expresadas en grados celcius (°C).
- k y h: dos long doubles, son constantes de los materiales, k es la conductividad térmica de la pared del horno y h el coeficiente de transferencia de calor del borde de la pared.
- bordeInterno: es un puntero a entero, en realidad, un arreglo, representa la función r discretizada, sus valores son el número de radio donde empieza la pared interna del horno. Con esta función queda totalmente definida la forma del horno.

Si bien parece que el horno acapara demasiada información, toda ella es necesaria para la resolución del problema, ya que todos los datos son particulares del horno. Pasar alguno de ellos a la clase matriz rompería la modularidad.

Se analizaron distintas variantes para no sobrecargar esta clase (en particular su generador), como emplear una clase intermedia, o transferir obligaciones a matriz, pero se consideró la implementación actual la más correcta desde el punto de vista formal, ya que según nuestro modo de ver, y tras algunas consideraciones sobre qué sería lógico y esperable del comportamiento de esta clase, llegamos a esta ardua centralización de las funciones.
A continuación detallaremos cada método:

- constructor: recibe un radio exterior, cantidad de ángulos y radios de la discretización, temperaturas externa e interna, las constantes térmicas k y h y un puntero a entero con el que dinámicamente se generará una copia para modelar la función r discretizada. Para cada una de los otros parámetros se guarda una copia en el atributo pertinente. En cuanto a la matriz temperaturas, esta por ahora se mantiene en valor NULL. Para calcular los valores que finalmente resolverán el problema este constructor llama a la función privada calcular_temperaturas(), que se describe a continuación.

- calcular_temperaturas(): este método privado no recibe ni devuelve nada. Primero genera variables locales de tipo matriz (tres), que serán las matrices A, x y b del sistema de ecuaciones que nos atañe resolver, borra la matriz-atributo temperaturas, y crea una nueva matriz de cantidadDeAngulos*cantidadDeRadios de dimensión, que será el vector columna x, es decir la solución del sistema, trasladado a una matriz más cómoda de trabajar.

  Una vez creadas todas las variables involucradas se procede al llenado de la matriz A, con los coeficientes y las características descritas durante el desarrollo de este trabajo. Este proceso es simple en su estructura: se recorre punto por punto de la discretización, y según a que sector del horno pertenezca, se completa la fila correspondiente con los coeficientes pertinentes (uno, dos o cinco; como ya se mencionó en la sección anterior), pero resulta extremadamente reiterativo.

  Por ello, este método fue el que mayor discrepancias suscitó al momento de su implementación. A pesar de las discusiones no se encontró mejor forma de optimizar el recorrido de la función de manera tal que aproveche la forma regular de la matriz resultante. Una de las opciones consideradas consistiría básicamente en calcular (antes de llamar a la función) el tamaño de los tres segmentos de la matriz que se quiere formar. Sabiendo esto de antemano se puede realizar un rellenado más inteligente de la estructura.

  Nótese que el método calcular_temperaturas así como rellena la matriz A del sistema también completa, con los valores correspondientes, la matriz b.

  Una vez que todos los valores de A y b están seteados correctamente, se procede con la resolución del sistema utilizando los métodos de la clase matriz, triangular y resolver. Como resultado de la aplicación de éstos, se obtiene finalmente el vector columna x, solución del sistema planteado. Este vector es trasladado a una matriz de dimensión angs*rads para facilitar su manejo, y copiado a la matriz-atributo temperaturas. Así queda finalizada la construcción del objeto Horno.

- getTemperatura(): recibe un ángulo y un radio, ambos long double, calcula los subíndices correspondientes y muestra el valor de la matriz temperaturas que coincide con ellos.

- getRadio(), getCantidadAngulos(), getCantidadRadios(), getTi(), getTinf(), getH(), getK(): Ninguno recibe parámetros, cada uno es un proyector del atributo de la clase correspondiente a su nombre.
- funcionRadio(): devuelve la cantidad de radios que hay entre el centro y el borde interno, en el número de ángulo que le pasen como parámetro.
- operator=: copia el objeto de la derecha en la de la izquierda.

Apendice A:

enunciado:

Se debe implementar un programa que tome como entrada los datos del problema y que calcule la temperatura en la pared del horno utilizando la técnica de resolución descripta en la sección anterior.

El programa debe tomar los datos de entrada desde un archivo de texto, cuyo formato queda a criterio del grupo. Es importante mencionar que los parámetros n y m de la discretización forman parte de los datos de entrada. Un elemento importante a definir es la especificación de la función ri(θ) en este archivo de entrada, se sugiere que el programa tome los valores ya discretizados de esta función.

El programa debe generar el sistema de ecuaciones lineales planteado en la sección anterior, procediendo a su resolución por medio de cualquier método directo para la resolución de sistemas de ecuaciones lineales (es decir, un método no iterativo). El programa debe escribir la solución en un archivo, con un formato adecuado para su posterior graficación.

Se pide realizar experimentos con al menos dos instancias de prueba para Ti = 5000, T∞ = 30 y h/K = 0.05, generando distintas discretizaciones para cada una. Se recomienda fijar Re = 1 en estas instancias. Se sugiere que se presenten los resultados de estos experimentos en forma de gráficos de temperatura o gr´aficos de curvas de nivel, para ayudar a la visualización de los resultados.

Agradecemos el asesoramiento de Gabriel Acosta para la preparación de este trabajo práctico. Fecha de entrega: Lunes 1 de Octubre

Apendice B:

1. **Horno.h**

```
#ifndef _HORNO_H
#define _HORNO_H

#include <iostream>
#include <string.h>
#include <math.h>
#include "Matriz.h"

#define PI 3.14159265358979323484626433832795

using namespace std;

class Horno{
        friend void cargar(istream& archivo, Horno &h);
        friend void guardarParaGrafico(ostream &out, const Horno &h);
    public:
        /* Constructor */
```

```
        Horno(){temperaturas = NULL; bordeInterno = NULL;}
        Horno(int radio, int angulos, int radios, int tint, int text,
              long double k, long double h, int* radiosLimite);

        /* interfaz */
        long double getTemperatura(long double radio, long double theeta) const;
        long double     getRadio() const;
        int             getCantidadAngulos() const;
        int             getCantidadRadios() const;
        int             getTi() const;
        int             getTinf() const;
        long double     getK() const;
        long double     getH() const;
        int             funcionRadio(int angulo) const;
        void            operator=(const Horno &h1);

        /* Destructor */
        ~Horno();
    private:
        //Atributos
        Matriz* temperaturas;       //temperatura en cada punto discretizado del
                                    //horno
        long double rad;            //radio del horno desde el centro al borde
                                    //exterior
        int angs;                   //angulos en los que se divide el horno
        int rads;                   //cantidad de radios por angulo
        long double deltaR;         //delta Radio
        long double deltaT;         //delta Theeta
        int ti;                     //temperatura interior
        int tinf;                   //temperatura en el infinito
        long double k;              //constante k
        long double h;              //constante h
        int *bordeInterno;          //borde interno del horno.

        //Metodos privados
        void calcular_temperaturas(void);
};

#endif /*_HORNO_H*/
```

2. Horno.cpp

```
#include "Horno.h"


/************************************/
/*          METODOS PUBLICOS        */
/************************************/
```

```
Horno :: Horno(int radio, int cantAngulos, int cantRadios, int tint, int text,
               long double k, long double h, int* radiosLimite){
    rad = radio;
    angs = cantAngulos;
    rads = cantRadios;
    deltaR = (long double)rad/((long double)rads - 1);
    deltaT = 2*PI/(long double)angs;
    ti = tint;
    tinf = text;
    this->k = k;
    this->h = h;
    temperaturas = NULL;

    bordeInterno = new int [angs];

    // guardo los radios para cada angulo
    // asi queda definida la funcion de temperatura.
    for(int ang = 0; ang < angs; ang++)
        bordeInterno[ang] = radiosLimite[ang];

    calcular_temperaturas();
}


long double Horno :: getTemperatura(long double radio, long double theeta) const{
    int i = (int)fabs(radio/deltaR);
    if ( i > rads - 1 )
        i = rads - 1;

     while (theeta >= 2*PI){
        theeta -= 2*PI;
     }
     int j = (int)fabs(theeta/deltaT);
     if ( j > rads - 1 )
        j = rads - 1;

     return temperaturas->ver(i, j);
}


long double Horno :: getRadio() const{
    return rad;
}

int Horno :: getCantidadAngulos() const{
    return angs;
}
```

```
int Horno :: getCantidadRadios() const{
    return rads;
}

int Horno :: getTi() const{
    return ti;
}                       //Temperatura interior

int Horno :: getTinf() const{
    return tinf;
}                      //Temperatura exterior

long double Horno :: getK() const{
    return k;
}                         //constante K

long double Horno :: getH() const{
    return h;
}                         //constante H

void Horno :: operator=(const Horno &h1){
    rad = h1.rad;
    angs = h1.angs;
    rads = h1.rads;
    deltaR = h1.deltaR;
    deltaT = h1.deltaT;         //delta Theeta
    ti = h1.ti;
    tinf = h1.tinf;
    k = h1.k;
    h = h1.h;

    delete temperaturas;
    temperaturas = new Matriz(rads, angs);
    *temperaturas = *(h1.temperaturas);

    delete [] bordeInterno;
    bordeInterno = new int[angs];
    for(int i = 0; i < angs; i++)
        bordeInterno[i] = h1.bordeInterno[i];
}

int Horno :: funcionRadio(int angulo) const{
    return bordeInterno[angulo];
}

/* Destructor */
Horno :: ~Horno(){
    delete temperaturas;
    delete [] bordeInterno;
}
```

```
/************************************/
/*          METODOS PRIVADOS        */
/************************************/

void Horno :: calcular_temperaturas(void){

    Matriz temp(rads*angs, rads*angs);
    Matriz b(rads*angs, 1);
    Matriz X(rads*angs, 1);

    delete temperaturas;
    temperaturas = new Matriz(rads, angs);

    int filaALlenar = 0;

    for(int r = 0; r < rads; r++){
        for(int a = 0; a < angs; a++){
            if (r <= bordeInterno[a]){          //si es un punto del borde
                                                //interno
                //sabemos que por la 2da ecuacion, la temperatura de los puntos
                //dentro de este borde es 5000, entonces se que el coeficiente
                //del punto en cuestion sera 1, y su correspondiente
                //resultado en b es 5000

                //ahora asigno el coeficiente en la matriz
                //viendo la matriz como un arreglo de filas,
                //dentro de la fila a llenar:
                //T[r][a] = 1
                temp.asignar(filaALlenar, r*angs + a, 1);
                b.asignar(filaALlenar, 0, ti);
            }
            else{
                if(r != rads - 1){          //si no es un punto del borde interno
                    long double coef1 = 1/(deltaR*deltaR) - 1/(r*deltaR*deltaR);
                    long double coef2 = 1/((deltaT*deltaT)*(r*r*deltaR*deltaR));
                    long double coef3 = -2/(deltaR*deltaR) + 1/(r*deltaR*deltaR)
                                        -2/(r*deltaR*r*deltaR*deltaT*deltaT);
                    long double coef4 = 1/((r*deltaR*r*deltaR)*(deltaT*deltaT));
                    long double coef5 = 1/(deltaR*deltaR);
                    //coef 1 a 5 son los 5 coeficientes de las incognitas que
                    //quedan
                    //luego de la discretizacion del Laplaciano

                    //ahora asigno los coeficientes en la matriz
                    //viendo la matriz como un arreglo de filas,
                    //dentro de la fila a llenar:
                    //T[r-1][a] = coef1
                    //T[r][a-1] = coef2
                    //T[r][a]   = coef3
```

```
                    //T[r][a+1] = coef4
                    //T[r+1][a] = coef5
                    temp.asignar(filaALlenar, (r - 1)*angs + a, coef1);
                    temp.asignar(filaALlenar, r*angs + a - 1, coef2);
                    temp.asignar(filaALlenar, r*angs + a, coef3);
                    temp.asignar(filaALlenar, r*angs + a + 1, coef4);
                    temp.asignar(filaALlenar, (r + 1)*angs + a, coef5);
                }
                else{ //si el punto es del borde externo
                    long double coef = k/(h*deltaR);
                    //coef cubre los 2 coeficientes que resultan de discretizar
                    //la 3er ecuacion en cuestion

                    //ahora asigno los coeficientes en la matriz
                    //viendo la matriz como un arreglo de filas,
                    //dentro de la fila a llenar:
                    //T[r][a] = coef
                    //T[r-1][a] = coef + 1
                    temp.asignar(filaALlenar, r*angs + a, -coef - 1);
                    temp.asignar(filaALlenar, (r - 1)*angs + a, coef);
                    b.asignar(filaALlenar, 0, -tinf);
                }
            }
            //ahora avanzo en la fila
            filaALlenar++;
        }
    }

    temp.triangular(&b);
    temp.resolver(X,b);

     for(int r = 0; r < rads; r++){
         for(int a = 0; a < angs; a++){
             temperaturas->asignar(r,a,X.ver(r*angs + a, 0));
         }
     }
}

/************************************/
/*          FUNCIONES FRIEND        */
/************************************/

void cargar(istream& archivo, Horno &h){
    char data[128];

    //agarro el comentario
    archivo.getline(data, 100);
    //agarro radio exterior
    archivo.getline(data, 100);
    h.rad = atoi(data);
```

```
//agarro el comentario
archivo.getline(data, 100);
//agarro cantidad de angulos
archivo.getline(data, 100);
h.angs = atoi(data);

//agarro el comentario
archivo.getline(data, 100);
//agarro cantidad de radios
archivo.getline(data, 100);
h.rads = atoi(data);

h.deltaR = (long double)h.rad/((long double)h.rads - 1);
h.deltaT = 2*PI/(long double)h.angs;

//agarro el comentario
archivo.getline(data, 100);
//agarro temperatura interior
archivo.getline(data, 100);
h.ti = atoi(data);

//agarro el comentario
archivo.getline(data, 100);
//agarro temperatura exterior
archivo.getline(data, 100);
h.tinf = atoi(data);

//agarro comentario
archivo.getline(data, 100);
//agarro cosntante K
archivo.getline(data, 100);
h.k = atof(data);

//agarro comentario
archivo.getline(data, 100);
//agarro constante H
archivo.getline(data, 100);
h.h = atof(data);

//agarro comentario
archivo.getline(data, 100);
//agarro funcion de temperatura
delete [] h.bordeInterno;
h.bordeInterno = new int [h.angs];

for(int ang = 0 ; ang < h.angs ; ang++){
    // agarro el radio para el angulo cant
    archivo.getline(data, 100);
    h.bordeInterno[ang] = atoi(data);
```

```
    }

    h.calcular_temperaturas();
}

void guardarParaGrafico(ostream &out, const Horno &h){
    Matriz m(*(h.temperaturas));

    int centroR = m.filas() - 1;             //radios

    int X = 0;
    int Y = 0;

    // guardo las X
    out << "X = [ ";

    for(int i = 0 ; i < m.filas(); i++){
        for(int j = 0 ; j < m.columnas(); j++){
            X = (int)((i*(h.rad)/(h.rads))*cos(j*2*PI/centroR));
            //cuanto me muevo en "X" = Rcos(tita)

            out << " " << X;
        }
    }

    out << "];" << endl << endl;
    // guardo las Y
    out << "Y = [ ";
    for(int i = 0 ; i < m.filas(); i++){
        for(int j = 0 ; j < m.columnas(); j++){
            Y = (int)((i*(h.rad)/(h.rads))*sin(j*2*PI/centroR));
            //cuanto me muevo en "Y"

            out << " " << Y;
        }
    }


    out << "];" << endl << endl;

    //guardo los resultados del sistema
    out << "A = ";
    mostrarParaGraficar(out, *(h.temperaturas));
}
```

3. Matriz.h

```
#ifndef _MATRIZ_H
```

```
#define _MATRIZ_H

#include <iostream>
#include <string.h>
#include <assert.h>
using namespace std;

class Matriz{
    friend ostream& operator<<(ostream&, const Matriz&);
    friend ostream& mostrarParaGraficar(ostream&, const Matriz&);
  public:
    Matriz(int = 4, int = 4);
    Matriz(const Matriz& mat);
    int filas() const;
    int columnas() const;

    long double ver(int fila, int columna) const;
    void asignar(int fila, int columna, long double valor);
    void triangular(Matriz *b = NULL);
    void resolver(Matriz &X, Matriz &b);
    void operator =(const Matriz &m1);
    ~Matriz();
  private:
    long double **m;
    int fil;
    int col;
    /* metodos privados */
    void permutar(int fila1, int fila2, Matriz *b);
    void pivotear(int, Matriz *b);
    void restarFilas(long double coef, int filaAanular, int filaActual,
                     Matriz *b);
};

#endif /*_MATRIZ_H*/
```

4. Matriz.cpp

```
#include "Matriz.h"
#define MOD(a) ((a < 0) ? (-a) : (a))

/************************************/
/*          METODOS PUBLICOS        */
/************************************/

Matriz :: Matriz(int f, int c){
    //f = filas, c = columnas
    assert((f > 0) && (c > 0));
    fil = f;
```

```
    col = c;

    m = new long double* [f];
    for(int i = 0; i < f; i++){
        m[i] = new long double[c];
        for(int j = 0; j < c; j++)
            m[i][j] = 0;
    }
}

Matriz :: Matriz(const Matriz& mat){
    m = NULL;
    *this = mat;
}

int Matriz :: filas() const{
    return fil;
}

int Matriz :: columnas() const{
    return col;
}

long double Matriz :: ver(int fila, int columna) const{
    assert((fila < fil) && (columna < col));
    return m[fila][columna];
}

void Matriz :: asignar(int fila, int columna, long double valor){
    assert((fila >= 0) && (columna >= 0));
    assert((fila < fil) && (columna < col));
    m[fila][columna] = valor;
}

/*
 *  triangular: triangula la matriz mediante el metodo de eliminacion gaussiana
 *              con pivoteo parcial.
 */
void Matriz :: triangular(Matriz *b){
    int filaActual = 0; //notar que filaActual = columnaActual ya que es la que
                        //recorre la matriz en diagonal

    for(filaActual = 0 ; filaActual < fil ; filaActual++){
        pivotear(filaActual, b); //hago pivoteo parcial

        for(int filatemp = filaActual + 1 ; filatemp < fil ; filatemp++){
            long double coeficiente = m[filatemp][filaActual]/
                                      m[filaActual][filaActual];
            restarFilas(coeficiente, filatemp, filaActual, b);
        }
```

```
    }
}

/*
 * PRECONDICION: Suponemos que la matriz implicita
 * es triangular superior
 */

void Matriz :: resolver(Matriz &X, Matriz &b){
    assert(X.fil == col);
    assert((b.fil == this->fil) && (b.col == X.col));

    for (int i = fil-1; i >= 0; i--){
        long double sumaPorFil = 0;

        for (int j = col-1; j > i; j--)
            sumaPorFil += m[i][j]*X.m[j][0];
        X.m[i][0] = (b.m[i][0] - sumaPorFil)/m[i][i];
    }
}

void Matriz :: operator =(const Matriz &m1){
    fil = m1.fil;
    col = m1.col;

    //primero borro la matriz que ya estaba
    if (m != NULL){
        for(int i = 0; i < fil; i++)
            delete m[i];
        delete m;
    }

    m = new long double*[fil];

    for(int i = 0; i < fil; i++){
        m[i] = new long double[col];
        for(int j = 0; j < col; j++){
            m[i][j] = m1.m[i][j];
        }
    }
}

Matriz :: ~Matriz(){
    for(int i = 0; i < fil; i++)
        delete m[i];
    delete m;
}

/************************************/
/*          METODOS PRIVADOS        */
```

```
/************************************/

void Matriz :: permutar(int fila1, int fila2, Matriz *b){
    long double* tmp;

    tmp = m[fila1];
    m[fila1] = m[fila2];
    m[fila2] = tmp;

    if(b != NULL){
        tmp = b->m[fila1];
        b->m[fila1] = b->m[fila2];
        b->m[fila2] = tmp;
    }
}

void Matriz :: pivotear(int c, Matriz *b){
    int max = c;

    for(int i = c; i < fil; i++){
        if(MOD(m[i][c]) > MOD(m[max][c]))
            max = i;
    }

    permutar(c, max, b);
}

/*
 *  restarfilas: recibe un coeficiente, una fila a anular y la fila de pivote.
 *               y resta las filas por el coeficiente.
 */

void Matriz :: restarFilas(long double coef, int filaAanular, int filaActual,
                           Matriz *b){
    for(int i = filaActual; i < fil; i++){
        m[filaAanular][i] = m[filaAanular][i] - coef*m[filaActual][i];
        if(MOD(m[filaAanular][i]) < 1e-10) //10^(-10)
            m[filaAanular][i] = 0;
    }

    if(b != NULL)
        b->m[filaAanular][0] = b->m[filaAanular][0] - coef*b->m[filaActual][0];
}

/************************************/
/*          FUNCIONES FRIEND        */
/************************************/

ostream& operator<<(ostream& os, const Matriz& matriz){
    for(int i = 0; i < matriz.fil; i++){
```

```
        os << "\nFila" << i << ": ";
        for(int j = 0; j < matriz.col; j++)
            os << matriz.m[i][j] << " ";
    }
    return os;
}

ostream& mostrarParaGraficar(ostream& os, const Matriz& matriz){
    os << "[ ";
    for(int i = 0; i < matriz.fil; i++){
        for(int j = 0; j < matriz.col; j++)
            os << matriz.m[i][j] << " ";
    }
    os << "];";
    return os;
}
```

5. main.cpp

```
#include <iostream>
#include <fstream>
#include "Matriz.h"
#include "Horno.h"

using namespace std;

int main(){

    Horno h;
    unsigned int opcion = 0;

    cout << "Simulador para hornos de altas temperaturas\n";
    cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n";

    while(opcion != 4){
        cout << "\n\n1. Cargar un horno\n";
        cout << "2. Guardar datos del horno actual para realizar graficos\n";
        cout << "3. Ver temperatura en un punto\n";
        cout << "4. Salir\n\n";
        cout << "Ingresar opcion: ";
        cin >> opcion;

        if(opcion == 1){
            char nombreDelArchivo[256];
            cout << "\n\nIngrese el nombre del archivo: ";
            cin >> nombreDelArchivo;

            ifstream archivoEntrada;
```

```
            archivoEntrada.open(nombreDelArchivo, ifstream::in);
            while(archivoEntrada.fail()){
                archivoEntrada.clear();
                cout << "\n\nEl archivo no se encuentra.
                         Ingrese el nombre del archivo: ";
                cin >> nombreDelArchivo;
                archivoEntrada.open(nombreDelArchivo, ifstream::in);
            }

            cargar(archivoEntrada, h);
            archivoEntrada.close();
        }

        if(opcion == 2){
            char nombreDelArchivo[256];
            cout << "\n\nIngrese el nombre del archivo: ";
            cin >> nombreDelArchivo;

            ofstream archivoSalida;
            archivoSalida.open(nombreDelArchivo, ofstream::out);
            guardarParaGrafico(archivoSalida,h);
            archivoSalida.close();
        }

        if(opcion == 3){
            long double radio, angulo;
            cout << "Los datos deben ser ingresados en coordenadas polares\n";
            cout << "\nIngresar radio: ";
            cin >> radio;
            cout << "\nIngresar angulo(Radianes)";
            cin >> angulo;
            cout << h.getTemperatura(radio, angulo);
        }
    }

    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Apendice C:

En este apartado se tratará el tema de la interfaz del usuario con el programa. En particular interesa el parser utilizado para obtener información de un archivo de texto con cierto formato. Además se cuenta con una función que permite guardar un horno creado por el usuario en un archivo de texto con el formato indicado.

Estas son funciones friend de la clase Horno, 'cargar', que ejecuta un parser sobre el archivo pasado como parámetro y 'guardarParaGrafico', que almacena

el horno creado por el usuario en un formato cómodo para realizar los gráficos. En particular nos interesa el parser, ya que la otra función se limita a guardar las matrices del sistema de ecuaciones modelado.

Nuestro parser es una función que recorre un archivo de texto esperando encontrar cierta información alamecenada en un formato fijo. Si el contenido respeta esa forma particular, entonces el parser recopila en distintas variables (en nuestro caso strings o cadenas de caracteres) las piezas de información para generar con ellas un horno que responda a los valores del archivo.

Para que el parser funcione correctamente es indispensable que el archivo de texto mantenga una estructura fija y conocida.
El formato reconocido por el parser es el siguiente:

```
//comentario (radio exterior)
numero
// comentario (cantidad de angulos)
numero
// comentario (cantidad de radios contando el centro)
numero
// comentario (temperatura interior)
numero
// comentario (temperatura infinito)
numero
// comentario (constante K)
numero
// comentario (cosntante H)
numero
// comentario (definicion discretizada de la funcion r)
cantidad de numeros igual a cantidad de angulos
```

Es indispensable pasar los datos en el orden correcto, ya que el parser toma linea por linea (considerando los comentarios) el contenido, y lo transforma en valores numericos nativos de C++, para asignarselos a cada atributo de la clase Horno.

Referencias

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

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

[3] http://www.mathworks.com/matlabcentral/files/8998/content/gridfitdir/demo/html/gridfit_demo.ht
Graficos avanzados en Matlab.

[4] http://www.mathworks.com/matlabcentral/newsreader/view_thread/155703
Hacer graficos en 3d con diferentes colores en Matlab (foro).

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

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

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