Bitácora de trabajo
Gustavo Ramírez Hidalgo
Proyecto módulo 2 - SP2118

1. Descripción general:

El proyecto consiste en la resolución de la ecuación de Poisson a través de una implementación del método de diferencias finitas, haciendo uso de herramientas computacionales.
Las herramientas a utilizar son C, para crear un programa que luego será empaquetado en un módulo para Python. Una vez que se ha trasladado el proceso a Python, se utiliza VPython para la visualización de los resultados.

2. Sobre el método de diferencias finitas:

2.1. 1 variable...

A diferencia de métodos como *linear and nonlinear shooting method*[1], entre otros, el método de diferencias finitas es basante estable, en general (aunque requieran un poco más de poder computacional que los primeros).

Los métodos de diferencias finitas reemplazan cada una de las derivadas en la ecuación diferencial, con una aproximación de cociente de diferencias adecuada (esto no será tratado en detalle aquí, pero estas aproximaciones son básicamente distintos tipos de aproximaciones que emplean la expansión de Taylor de una función a diferentes órdenes para aproximar funciones y derivadas de distintos órdenes, dependiendo de la grilla que se elija o posea para llevar a cabo los cálculos).

El cociente de diferencias que se elija y el paso de los cálculos numéricos, dependen mucho del error que se desee.

Trabajando con funciones dependientes de 1 variable (este no será el caso a resolver, pues la ecuación de Poisson está definida a través de un problema multivariable), la primera y segunda derivada se aproximan de la siguiente manera:

$$y'(x_i) = \frac{1}{2h}[y(x_{i+1}) - y(x_{i-1})] - \frac{h^2}{6}y'''(\eta_i)$$
$$y''(x_i) = \frac{1}{h^2}[y(x_{i+1}) - 2y(x_i) + y(x_{i-1})] - \frac{h^2}{12}y^{(4)}(\zeta_i)$$

Las variables independientes en la segunda y cuarta derivada no tienen mucha importancia, pues lo importante ahí es notar que ambos términos están asociados al error. Luego, se toman estas aproximaciones, se insertan en la ED, y luego se hace el cambio:

$$y(x_i) \to w_i$$

pudiendo con esto establecer un sistema de ecuaciones lineales (es decir, para resolvelo basta con llevar a cabo la inversión de una matriz).

2.2. Multivariable...

La ecuación diferencial en derivadas parciales que se desea resolver es del tipo Elíptica. La ecuación de Poisson tendrá aquí la siguiente forma:

$$\nabla^2 u(x,y) = \frac{\partial^2 u}{\partial x^2}(x,y) + \frac{\partial^2 u}{\partial y^2}(x,y) = f(x,y)$$

[1] R. Burden, J. Faires. Numerical Analysis. 9th edition. 2011.

la cual estará definida sobre una región $R = \{(x, y) \mid a < x < b, c < y < d\}$, con $u(x, y) = g(x, y)$ para $(x, y) \in S$, con S denotando la frontera de R. Si f y g son continuas en sus dominios, entonces hay una solución única para esta ED.

Definido el problema de esta forma, y al ser 2 dimensional (muy útil este carácter de $n = 2$ en estudios de conductividad sobre superficies, por ejemplo), se establecen dos pasos: $h = (b - a)/n$ y $k = (d - c)/m$, con h y n correspondiendo con la derivada en x, y k y m con la derivada en y.

Se genera así la grilla (x_i, y_j) (más específicamente, estos puntos son los *puntos de malla* de la grilla, pero se les llama así a aquellos para los cuales $i = 1, 2, ..., n - 1$ y $j = 1, 2, ..., m - 1$, con i y j por definirse a continuación), tales que:

$$x_i = a + ih, \;\; i \text{ hasta } n, \text{ desde } 0$$
$$y_j = c + jk, \;\; j \text{ hasta } m, \text{ desde } 0$$

Se procede ahora con las expansiones a través de la serie de Taylor, generando así las fórmulas en diferencias centradas ($i = 1, 2, ..., n - 1$ y $j = 1, 2, ..., m - 1$):

$$\frac{\partial^2 u}{\partial x^2}(x_i, y_j) = \frac{u(x_{i+1}, y_j) - 2u(x_i, y_j) + u(x_{i-1}, y_j)}{h^2} - \frac{h^2}{12}\frac{\partial^4 u}{\partial x^4}(\zeta_i, y_j), \text{ con } \zeta_i \in (x_{i-1}, x_{i+1})$$
$$\frac{\partial^2 u}{\partial y^2}(x_i, y_j) = \frac{u(x_i, y_{j+1}) - 2u(x_i, y_j) + u(x_i, y_{j-1})}{k^2} - \frac{k^2}{12}\frac{\partial^4 u}{\partial y^4}(x_i, \eta_j), \text{ con } \eta_i \in (y_{i-1}, y_{i+1})$$

Así, la ED se vuelve bastante simple, desde un punto de vista numérico:

$$\frac{u(x_{i+1}, y_j) - 2u(x_i, y_j) + u(x_{i-1}, y_j)}{h^2} + \frac{u(x_i, y_{j+1}) - 2u(x_i, y_j) + u(x_i, y_{j-1})}{k^2} =$$
$$f(x_i, y_j) + \frac{h^2}{12}\frac{\partial^4 u}{\partial x^4}(\zeta_i, y_j) + \frac{k^2}{12}\frac{\partial^4 u}{\partial y^4}(x_i, \eta_j)$$

Todo esto resulta en el siguiente resumen para el método de diferencias finitas, aplicado a la ecuación de Poisson (haciendo el cambio $u(x_i, y_j) \to w_{ij}$):

$$2[(h/k)^2 + 1]w_{ij} - (w_{i+1,j} + w_{i-1,j}) - (h/k)^2(w_{i,j+1} + w_{i,j-1}) = -h^2 f(x_i, y_j), \text{ para } i = 1, 2, ..., n - 1 \text{ y } j = 1, 2, ..., m - 1$$

y con condiciones de frontera:

$$w_{0j} = g(x_0, y_j) \text{ y } w_{nj} = g(x_n, y_j), \text{ para cada } j = 0, 1, ..., m$$
$$w_{i0} = g(x_i, y_0) \text{ y } w_{im} = g(x_i, y_m), \text{ para cada } i = 1, ..., n - 1$$

Esto produce un sistema de ecuaciones lineal siendo los puntos w_{ij} (los puntos interiores de la grilla) las incógnitas.

El algoritmo que permite solucionar el sistema de ecuaciones que se presentó anteriormente (este algoritmo hace uso del método iterativo de Gauss-Seidel, por lo que permite que $m \neq n$) se puede consultar como el algoritmo 12.1 del Numerical Analysis, Burden & Faires, novena edición. En el caso de este proyecto, se implementó dicho algoritmo en el módulo de C, en el archivo llamado modulopoissonmodule.c.

La forma en que se eligió la relación entre los índices de conteo y la forma de la grilla puede que sea un poco distinta a lo normal. Se optó por colocar el punto 0,0 en la esquina inferior izquierda (viendo al plano xy de la forma habitual, desde arriba), y luego se comenzó el conteo del arreglo hacia la derecha, sobre el eje x, y después al llegar al final de la primera línea, se salta al extremo izquierdo de la segunda línea (que está encima de la primera), y así sucesivamente.

3. Sobre el empaquetado de código en C++ para creación de módulos para Python:

Sobre esto se puede encontrar muchísima información en línea[2]. Lo que se hará aquí es una extensión, no una incrustación (esto último es utilizar herramientas de Python en C).

Para generar la extensión, se llevan a cabo una serie de pasos, pero básicamente el punto principal es la creación de un archivo en C (el módulo) en el cual irá todo el código de C que se quiera ejecutar. Desde Python se llamará este código. En el caso de este proyecto, el archivo se ha nombrado moduloC.c, mientras que el módulo fue llamado moduloSolvePoisson.

Para poder hacer uso de este programa, es necesario tener instalado python-dev. En caso de no tenerlo instalado:

```
$ sudo aptitude install python-dev
```

y este paquete es necesario para la compilación del módulo.

Para evitar problemas, es preferible hacer los castings necesarios desde Python, y luego llamar al método definido a través del módulo. En el caso presente, se hizo el casting de todas las variables de entrada desde Python, como se puede apreciar en el archivo programa.py.

Se puede presentar el problema de que se haya hecho el empaquetado del módulo ($ python setup.py build) siendo superusuario, en cuyo caso hay que borrar el directorio "build" que se crea, esto para evitar conflictos de permisos a la hora de la ejecución, y llevar a cabo de nuevo el empaquetado, con su posterior instalación asociada.

En este caso, si se desea (una vez que se esté programando en el módulo de C) hacer uso de objetos de Python en C, la documentación oficial[3] resulta muy útil.

Hay que tener el cuidado de llenar de ceros el arreglo solicitado con malloc, pues con la memoria dinámica no se tienen ceros.

4. Sobre VPython:

Para llevar a cabo una instalación de VPython, es necesario hacer la descarga de dicha biblioteca, y se puede elegir entre código fuente y binario, esto en la página oficial de VPython[4]. Una vez instalado, el uso es bastante simple, pero vale la pena estacar algunos puntos importantes:

- Lo que se necesita de VPython aquí, es mínimo, por lo que no es mucha la documentación sobre esta biblioteca a la que hay que accesar[5]. Simplemente se necesita la creación de esferas, de un radio pequeño, y ubicarlas en los puntos específicos del espacio, dados por el arreglo resultante de valores obtenidos con la resolución de la ecuación de Poisson, a través del módulo de C.
- Son pocos los comandos de VPython a utilizar, en este caso:
  - from visual import *: se importan todas las herramientas que la biblioteca VPython tiene a disposición.
  - sphere(): este objeto tridimensional es dibujado con atributos por default. Hay que pasar ciertos parámetros para modificar los atributos de este objeto, o el cambio de los atributos se puede setear posterior a su creación. En el caso de este proyecto, se llevó a cabo el seteado de los atributos al mismo tiempo que la instanciación del objeto.
  - De la misma forma que la esfera, se pueden crear otros objetos box(), que se emplearon en los ejes en este caso.
  - También se hicieron varias modificaciones a la escena, y hay mucha documentación al respecto[6,7], pero

[2] http://docs.python.org/2/extending/index.html
[3] http://docs.python.org/2/c-api/concrete.html
[4] http://vpython.org/contents/download_linux.html
[5] http://www.youtube.com/watch?v=KbOyKOlWBrs
[6] http://guigui.developpez.com/cours/python/vpython/en/?page=windowseventfile
[7] http://vpython.org/contents/docs/display.html

básicamente el método empleado fue display(), con los cambios correspondientes a este caso.

5. Comparación de resultados con la solución conocida: