Implementación de rutinas OpenGL para representar una malla elástica simulada numéricamente.

A. Sztrajman

17 de septiembre de 2007

Resumen

Se simula numéricamente una malla elástica bidimensional, a través del método de diferencias finitas. Los resultados son mostrados en pantalla utilizando la librería GL, que permite realizar representaciones gráficas tridimensionales haciendo uso de una aceleradora de video. Se muestran algunas soluciones específicas de la malla y se estudian también dos utilitarios de GLU (Glut y GlutMaster).

1. Simulación de una malla elástica

La malla simulada numéricamente está modelada como un conjunto de particulas de igual masa, que interactúan a través de resortes ideales e iguales, con sus primeros vecinos. Un esquema del modelo se muestra en la figura 1.

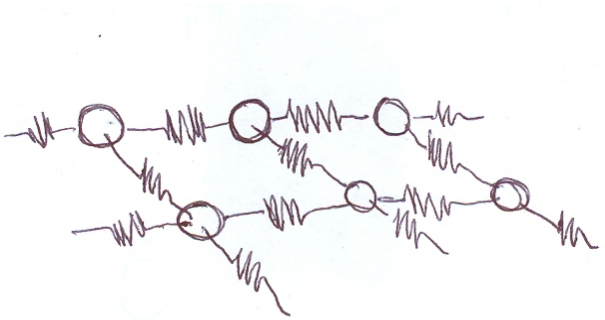

Figura 1: Esquema del modelo de malla.

La aceleración que experimenta un cuerpo en interacción con un resorte ideal es proporcional al apartamiento del resorte respecto de su posición de equilibrio (comúnmente llamada "longitud natural"). La constante de proporcionalidad de dicha interacción depende de la masa del cuerpo y de la "dureza" del resorte (es decir, de características materiales del resorte), lo cual se expresa matemáticamente:

$$a = -\frac{k}{m}.y \tag{1}$$

```
a : aceleracion del cuerpo
k : constante del resorte
m : masa del cuerpo
y : apartamiento de la posicion de equilibrio
```

De esta forma, la aceleración de cada cuerpo de la malla dependerá de su posición relativa a la de sus primeros vecinos. Si además consideramos que la aceleración de un cuerpo corresponde a la derivada segunda de su posición con respecto al tiempo, podemos deducir un sistema de ecuaciones diferenciales que describe el comportamiento del sistema. Para cualquier elemento (i,j) de la malla vale que:

$$a_{i,j} = \frac{d^2 y_{i,j}}{dt^2} = -\frac{k}{m}(4y_{i,j} - y_{i+1,j} - y_{i-1,j} - y_{i,j+1} - y_{i,j-1}) \tag{2}$$

1.1. Energía

Para un sistema de masas y resortes, existen dos formas de acumulación de energía mecánica: como excitación de un resorte (compresión o estiramiento), y como velocidad de una masa. La primera es denominada "energía potencial elástica", mientras que a la última se la refiere como "energía cinética". La energía total acumulada en estas dos formas puede transformarse de una a otra, pero su valor total no cambia a menos que una fuente externa de energía interactúe con el sistema.
Para un resorte, su energía potencial elástica puede ser calculada como:

$$E_{elastica} = \frac{1}{2}.k.y^2 \tag{3}$$

```
k : constante del resorte
y : apartamiento de la posicion de equilibrio
```

Mientras que para un cuerpo en movimiento, su energía cinética es:

$$E_{cinetica} = \frac{1}{2}.m.v^2 \tag{4}$$

```
m : masa del cuerpo
v : velocidad del cuerpo
```

1.2. Resolución numérica

El algoritmo utilizado para resolver la ecuación 2 corresponde al `metodo de Euler`. En cada paso de integración temporal se calcula la aceleración de cada cuerpo de la malla, utilizando 2 que sólo requiere el conocimiento de las posiciones. A partir del valor de aceleración se obtienen las "velocidades instantáneas" de cada cuerpo, teniendo en cuenta que la aceleración es la derivada temporal de la velocidad. Por último se calculan las nuevas posiciones de los cuerpos, a partir de sus velocidades, teniendo en cuenta que estas últimas corresponden a la derivada temporal de aquellas. La implementación del método puede verse en

detalle en la rutina `entorno_gl::integrar()` del programa. Debe notarse en dicha implementación que los cuerpos que se encuentran en los bordes de la malla no son integrados, de forma que su posición se mantiene inalterada a lo largo de la simulación. Este procedimiento da lugar a la condición de borde conocida como "extremos fijos". En otras palabras, la simulación corresponde a una malla elástica sujeta en sus extremos.

1.3. Predicción analítica

La bibliografía existente sobre sistemas sujetos a la condición de extremos fijos predice la existencia de "modos normales". Estos son configuraciones iniciales de la malla para las cuales en conjunto se comporta de manera oscilatoria a frecuencia constante. Si el sistema es puesto a oscilar en uno de sus modos normales, cada uno de sus elementos oscilará con la misma frecuencia, y la "forma funcional" del modo se conservará a lo largo del tiempo.
La cantidad de modos normales de la malla es igual al número de sus elementos conformantes. Todas las configuraciones posibles de la malla se obtienen como combinación lineal de los modos normales de la malla. Este es el motivo por el cual son una herramienta útil para estudiar esta clase de sistemas desde el álgebra lineal.
Las formas funcionales de los modos normales de la malla corresponden a combinaciones de funciones sinusoidales, con frecuencias espaciales tales que se anulan en los bordes de la malla (extremos fijos). En la sección de casos de prueba se abordan las formas funcionales del modo fundamental (aquel con la menor frecuencia de oscilación de la malla) y del primer estado excitado (correspondiente a la segunda más baja frecuencia de oscilación de la malla).
La utilización de "teoremas de conservación" es ideal para estudiar la estabilidad y precisión de algoritmos de integración numérica. En el sistema simulado, la teoría al respecto predice la conservación de la energía mecánica total del sistema. Por este motivo, se implementaron rutinas dirigidas a registrar la energía del sistema para todo tiempo. Al finalizar la simulación (por acción del usuario) se guardan en el archivo `energy.out` los registros sobre errores y variaciones de energía a lo largo de la ejecución. Dado que el algoritmo de resolución es una aproximación a primer orden del sistema, es de esperar que se registren algunas variaciones de la energía del sistema.

2. Librerías y utilitarios de OpenGL

La librería GL (graphic library) contiene las primitivas de OpenGL. En adición a éstas, se agregan comúnmente primitivas de dibujo de alto nivel correspondientes a la librería GLU (graphic library unit). Sin embargo, por cuestiones de portabilidad (y de facilidad), generalmente no se trata directamente con estas librerías, sino que se apela a utilidades (Toolkits) especialmente diseñadas para facilitar la interacción con el sistema de ventanas (X, Windows, etc). Este es el caso de GLUT (graphic library unit toolkit).

2.1. GLUT

Esta librería ofrece al programador, rutinas que facilitan el manejo de ventanas y de dispositivos del sistema (mouse, teclado, etc). Un uso elemental de esta librería permite fácilmente crear ventanas que incluyen entornos de trabajo de OpenGL, con dadas propiedades como sus proporciones y su nombre, y asignar rutinas para manejar sus eventos. Esto último se realiza a través de rutinas de GLUT que llevan como parámetro un puntero a rutina.

Ejemplo:

```
void glutDisplayFunc(void (*display)())
```

Esta rutina se encarga de asignar la rutina que pasemos como parámetro (a través de un puntero que la referencie) a la tarea de manejar el evento de ventana "display"(redibujar el contenido de la ventana). Análogamente existen rutinas de asignación para eventos de mouse, de teclado, y otros.Una vez que se han asignado las rutinas para manejar los eventos deseados, simplemente se ejecuta `void glutMainLoop()` y GLUT se ocupa de ejecutar las rutinas asignadas de acuerdo a los eventos.
El esquema de referencia de las rutinas que sugiere GLUT presenta serios problemas cuando se desea encapsular las rutinas referenciadas. Dada la definición de una rutina de asignación como la que se muestra en 2.1, no es posible introducir como parámetro un puntero a método de un objeto. Es necesario que el puntero referencie a una rutina, y no a un método.
Una forma de evitar estos problemas de referencia es declarar como ıstatic los métodos encapsulados a los que se hará referencia (los que se asignarán al manejo de eventos mediante las rutinas mencionadas). Sin embargo, esta declaración presenta dos problemas:

- Las rutinas así definidas sólo pueden acceder a las variables de tipo `static` de la clase a la que pertenecen.
- Al definir una clase con varios métodos y variables `static`, se pierde toda la idea del encapsulamiento. Más allá de una declaración formal, los métodos y variables así definidos no se instancian en cada objeto, sino que son comunes a todas las instancias de la clase. De esta forma, no pertenecen al çontexto"(`this`) de un objeto.

2.2. GLUTMaster

GLUTM es una versión encapsulada de GLUT. Los manejos de referencias mencionados están hechos en los archivos `glutMaster.cpp` y `glutWindow.cpp`. En este último se encuentran definidas y encapsuladas las funciones de manejo de eventos (ya referenciadas). Para redefinir estas funciones, es necesario declarar una nueva clase que herede los métodos de la clase `GlutWindow` que se deseen implementar. En el código del programa, la clase que hereda dichos métodos es `entorno_gl`.

3. class entorno_gl

La clase `entorno_gl` cuenta con los siguientes métodos y variables:

```
private:
    double energy_min;
    double energy_max;
    double energy;
    char *finput_name;

    window ventana;
    malla lamalla;
    vect angulo;
    double red, green, blue;
    bool view_mode;
    bool draw_mode;
    bool color_mode;

    void set_color(double colour);
    void set_camera();

    void calc_energy();
    void record_energy();

    void dibujar_malla();
    void integrar();
    void crear_malla();

public:
    entorno_gl(GlutMaster *glutMaster,
               int ancho, int alto,
               int x, int y,
               char *title);
    ~entorno_gl();

    void CallBackDisplayFunc(void);
    void CallBackReshapeFunc(int ancho, int alto);
    void CallBackIdleFunc(void);
    void CallBackPassiveMotionFunc(int x, int y);
    void CallBackKeyboardFunc(unsigned char key, int x, int y);
    void activar_funcion_idle(GlutMaster *glutMaster);

    void leer_archivo(char *name);
```

Las struct `ventana`, `lamalla`, `angulo` llevan respectivamente información sobre el estado de la ventana de trabajo, todos los elementos que componen la malla, y el ángulo de rotación de la figura en pantalla. La rotación se maneja a través de los eventos de mouse (solo movimientos) y se da en dos ejes.

`energy_min, energy_max` guardan los valores mínimo y máximo de energía del sistema, registrados hasta el momento durante la ejecución.

`energy` guarda el último valor registrado de energía del sistema.

`finput_name` apunta al nombre del archivo de entrada utilizado (pasado como parámetro de ejecución).

`red, green, blue` determinan las proporciones de rojo, verde y azul del color de dibujo. Sus valores pueden ser modificados en tiempo de ejecución presionando las teclas 'r', 'g' y 'b' (mayúsculas y minúsculas).

`view_mode` determina el tipo de proyección que realiza OpenGL para simular la tridimensionalidad de la imagen. Sus valores están codificados como directivas de preprocesador (`#define`): `VIEW_ORTHO` corresponde a la proyección ortonormal. Esta modalidad previene los cambios de proporción de los objetos (útil para aplicaciones CAD). La otra proyección disponible es `VIEW_PERSPECTIVE`, que reduce el tamao de los objetos cuando se alejan. El valor de `view_mode` se modifica en tiempo de ejecución presionando 'v'.

`draw_mode` determina la forma en que se dibuja la malla. Sus valores están codificados como directivas de preprocesador (`#define`): `DRAW_LINES` corresponde a trazar lineas entre los elementos de la malla (tipo alambre). `DRAW_QUADS` corresponde a llenar el espacio entre elementos con romboides (tipo sólido). En esta modalidad las variaciones de intensidad de color entre polígonos son fundamentales para dar claridad a la imagen. El color de cada polígono se determina proporcionalmente a su altura (apartamiento de la posición de equilibrio en la malla). El valor de `draw_mode` se modifica en tiempo de ejecución presionando 'd'.

`color_mode` determina la forma en que se da color a los polígonos. Sus valores están codificados como directivas de preprocesador (`#define`): `COLOR_FLAT` es la modalidad común, asigna un color "chato" (sin variaciones) a cada polgono. El valor se determina como el promedio de alturas de sus cuatro elementos vértice. La modalidad `COLOR_SMOOTH` permite hacer un gradiente de intensidad de color entre los elementos vértice de la malla. En la modalidad `DRAW_QUADS` (de `draw_mode`) se manifiesta una transición más suave de color entre los polígonos. El valor de `color_mode` se modifica en tiempo de ejecución presionando 'c'.

`entorno_gl(GlutMaster *glutMaster, int ancho, int alto, int x, int y, char *title)` constructor de la clase que se ocupa de crear la ventana de trabajo e inicializar un entorno de OpenGL adentro de la misma.

`void set_color(double colour)` Método auxiliar para activar la intensidad de color dada por `colour` junto con un offset de color dado por `red, green, blue`.

`void set_camera()` Método para configurar la matriz de proyección de OpenGL en la modalidad dada por `view_mode`.

`void calc_energy()` Método que se ocupa de calcular la energía mecánica total del sistema.

`void record_energy()` Método encargado de grabar en el archivo FILENAME ("energy.out") los registros de energía durante la ejecución (valor medio, variación media, error relativo porcentual), junto con detalles del archivo de entrada (nombre, dimensión de la malla, paso temporal utilizado, etc).

`void dibujar_malla()` La rutina de dibujo de OpenGL no puede estar en un contexto separado al de la estructura que contiene la información sobre la malla. De otra forma, habría que pasarle a una rutina `dibujar_recta()` (por ejemplo) punto por punto de la malla. La forma en que se maneja OpenGL hace que este procedimiento no sea óptimo. La forma más rápida de dibujar es pasarle a la placa aceleradora todo el conjunto de rectas en una sola estructura `glBegin(GL_LINES)`.

`void integrar()` Este método se encarga de realizar todas las operaciones de integración numérica sobre la malla, utilizando diferencias finitas. Cabe señalar que las condiciones de borde de la integración son "extremos fijos". Es decir, que los elementos de la malla que se encuentran en los extremos de la misma permanecen siempre en su posición inicial. Esta condición es muy común en la literatura al respecto, y da como resultado la existencia de modos normales de la malla de forma funcional sinusoidal.

`void crear_malla()` Realiza los manejos de memoria sobre el array bidimensional que contiene la información sobre la malla, e inicializa los valores de los elementos de la malla.

`void CallBackDisplayFunc(void)` Método heredado de la clase GlutWindow, para manejar el redibujado del contenido de la ventana de trabajo.

`void CallBackReshapeFunc(int ancho, int alto)` Método heredado de la clase GlutWindow, para manejar eventos de cambio de tamaño de la ventana de trabajo.

`void CallBackIdleFunc(void)` Método heredado de la clase GlutWindow, para manejar el ciclo de glut cuando no hay eventos.

`void CallBackPassiveMotionFunc(int x, int y)` Método heredado de la clase GlutWindow, para manejar eventos de movimiento del mouse.

`void CallBackKeyboardFunc(unsigned char key, int x, int y)` Método heredado de la clase GlutWindow para manejar eventos de teclado

`void activar_funcion_idle(Glut *glutMaster)` Informa un objeto GlutMaster (que es el que se encarga de las referencias con métodos `static`) que existe un método `CallBackIdleFunc(void)` definido por el usuario.

`void leer_archivo(char *name)` Método que lee la información sobre la malla de un archivo de un archivo (que será `argv[1]`) y llama a `void crear_malla()`.

4. Referencia de primitivas de OpenGL

`void glViewport(int x1, int y1, int x2, int y2)` define las proporciones del espacio de dibujo dentro de la ventana de trabajo.

`glEnable(GL_DEPTH_TEST)` Habilita el Z-buffer, el cual lleva registro de la superposición de polígonos. De esta forma, se respetan las prioridades de dibujado y se evita procesar figuras que finalmente no se muestran.

`glShadeModel()` Intercambia entre los modos de color GL_FLAT y GL_SMOOTH, de los que se habló en la sección `Class entorno_gl`. Lleva como parámetro un booleano, cuyos nombres están definidos como macros (directivas de preprocesador).

`glClear()` Borra un buffer. Los dos que se utilizan en el programa son el de dibujo (`GL_COLOR_BUFFER_BIT`) y el Z-Buffer (`GL_DEPTH_BUFFER_BIT`).

`glMatrixMode()` Elije la matriz activa. OpenGL utiliza dos matrices principales, a las que se le aplican distintas transformaciones secuencialmente, a fin de lograr la visualización buscada. Éstas son: `GL_PROJECTION` (las transformaciones se aplican a la "cámara") y `GL_MODELVIEW` (las transformaciones se aplican al modelo, es decir a los polígonos).

`glLoadIdentity()` Elimina todas las transformaciones que tenga aplicadas la matriz activa.

`glRotatef(float g, float x, float y, float z)` Aplica una rotación de ángulo `g` sobre la matriz activa, en torno al eje definido por `(x,y,z)`.

`glTranslatef(float x, float y, float z)` Aplica una translación definida por el vector `(x,y,z)`.

`glColor3d(double r, double g, double b)` Activa el color con las proporciones `(r,g,b)` (rojo, verde y azul). El valor máximo de cada color es 1,0.

`glVertex3d(double x, double y, double z)` Define un vértice en las coordenadas dadas por `(x,y,z)`.

`glBegin(GL_LINES)` Apertura de un contexto de dibujado de lineas. Dentro del contexto, cada dos vértices definidos, OpenGL traza una recta. Para iniciar un contexto de dibujado de otro tipo de figuras, basta cambiar el parámetro por GL_QUADS (o algún otro).

5. Archivos incluidos y utilizacion del programa

El programa fue probado satisfactoriamente en computadoras con aceleración gráfica, bajo sistema operativo Linux 2.6.20 i686 (Ubuntu Feisty 7.04).

5.1. Archivos necesarios

El link (como se puede ver en el archivo Makefile incluido) se realizó con los siguientes parámetros: `-lglut -lGLU -lGL -lXmu -lXext -lX11 -lm`.
En el sistema operativo utilizado, estas librerías vienen en los paquetes precompilados: `freeglut3.deb` y `libxmu-dev.deb`
Para compilar, se debe contar con los siguientes archivos (en el mismo directorio): `main.cpp entorno_gl.h entorno_gl.cpp glutMaster.h glutMaster.cpp glutWindow.h glutWind`

Al ejecutar el programa, se debe introducir como primer y único parámetro el nombre del archivo de entrada que se desea utilizar. Cuatro archivos de entrada de ejemplo se encuentran incluidos: `delta.inp gaussian.inp fundamental.inp second.inp`

5.2. Interfaz con el usuario

La interfaz con el usuario se da a través del teclado y el mouse. Los movimientos de este último producen rotaciones en torno a los ejes "x" e "y". Las teclas activas con teclados son las listadas a continuación.

Las teclas '+' y '-' transladan la camara en la direccion "z" (acercamiento).

Las teclas 'r' y 'R' cambian la proporción de rojo en el color activo (r disminuye, R aumenta).

Las teclas 'g' y 'G' cambian la proporción de verde en el color activo (g disminuye, G aumenta).

Las teclas 'b' y 'B' cambian la proporción de azul en el color activo (b disminuye, B aumenta).

La tecla 'c' cambia la modalidad de color entre color plano y gradiente.

La tecla 'v' cambia la modalidad de proyección entre ortonormal (conserva angulos) y perspectiva.

La tecla 'd' cambia la modalidad de dibujo entre líneas (tipo alambre) y relleno de romboides (tipo sólido).

La tecla "escape" interrumpe la ejecución del programa.

6. Casos de prueba

Junto con el programa se encuentran varios archivos con extensión "inp", que corresponden a archivos de entrada para el programa, cada uno con condiciones de prueba distintas. El siguiente es un archivo de entrada de ejemplo:

```
25          <- dimension de la malla (N x N)
0.003       <- paso de integracion temporal
1.0         <- constante elastica entre cada par de elementos de la malla
1.0         <- masa de cada elemento de la malla

0.0         <- aqui comienzan 25x25=625 lineas que corresponden
0.998          a las alturas iniciales de los elementos de la malla.
0.984          los primeros valores leidos corresponden a las posiciones
.              grid[0][0], grid[0][1], grid[0][2]... del array bidimensional.
.              espacialmente representan [x][z].
.
```

Descripción de los archivos de entrada de prueba y resultados

No se generan errores de ejecución si se cambian, directamente en un archivo de entrada (.inp), los valores del paso de integración temporal, de la constante de los resortes, y de la masa de los cuerpos, dejando intacto el resto del archivo. Las variaciones de energía permanecen relativamente constantes a partir de las dos primeras oscilaciones completas de la malla (en modos normales).

`fundamental.inp` Malla de dimensión 25 con estado fundamental (primer modo normal de la malla) de la forma: $y = sin(\pi * x) * sin(\pi * z)$ (m = 1,0, k = 1,0). En esta configuración la malla oscila con frecuencia constante, y su forma funcional se conserva a lo largo del tiempo. Esto es de esperar en un modo normal a extremos fijos. Resultados obtenidos en el archivo `energy.out` para distintos pasos temporales:

```
archivo de entrada: malla.inp
tamao de malla: 25
paso temporal: 0.05
constante de los resortes: 1
energia media: 2.85017
variacion media de energia registrada: 0.0140645
error relativo porcentual de energia: 0.49346 %

archivo de entrada: malla.inp
tamao de malla: 25
paso temporal: 0.01
constante de los resortes: 1
energia media: 2.84675
variacion media de energia registrada: 0.00198923
error relativo porcentual de energia: 0.0698772 %

archivo de entrada: malla.inp
tamao de malla: 25
paso temporal: 0.001
constante de los resortes: 1
energia media: 2.84704
variacion media de energia registrada: 0.00018709
error relativo porcentual de energia: 0.00657138 %
```

`fundamental99.inp` Malla de dimensión 99 con estado fundamental (primer modo normal de la malla) de la forma: $y = sin(\pi * x) * sin(\pi * z)$ (m = 1,0, k = 1,0). En esta configuración la malla oscila con frecuencia constante, y su forma funcional se conserva a lo largo del tiempo. Esto es de esperar en un modo normal a extremos fijos. Resultados obtenidos en el archivo `energy.out` para distintos pasos temporales:

```
archivo de entrada: malla.inp
tamao de malla: 99
paso temporal: 0.05
constante de los resortes: 1
energia media: 2.5718
```

```
variacion media de energia registrada: 0.00776583
error relativo porcentual de energia: 0.301961 %

archivo de entrada: inputs/fundamental99.inp
tamao de malla: 99
paso temporal: 0.1
constante de los resortes: 1
energia media: 2.58975
variacion media de energia registrada: 0.0281061
error relativo porcentual de energia: 1.08528 %
```

`fundamental10.inp` Malla de dimensión 10 con estado fundamental (primer modo normal de la malla) de la forma: $y = sin(\pi * x) * sin(\pi * z)$ (m = 1,0, k = 1,0). En esta configuración la malla oscila con frecuencia constante, y su forma funcional se conserva a lo largo del tiempo. Esto es de esperar en un modo normal a extremos fijos. Resultados obtenidos en el archivo `energy.out` para distintos pasos temporales:

```
archivo de entrada: malla.inp
tamao de malla: 10
paso temporal: 0.05
constante de los resortes: 1
energia media: 3.25009
variacion media de energia registrada: 0.0380921
error relativo porcentual de energia: 1.17203 %
```

`fundamental60.inp` Malla de dimensión 60 con estado fundamental (primer modo normal de la malla) de la forma: $y = sin(\pi * x) * sin(\pi * z)$ (m = 1,0, k = 1,0). En esta configuración la malla oscila con frecuencia constante, y su forma funcional se conserva a lo largo del tiempo. Esto es de esperar en un modo normal a extremos fijos. Resultados obtenidos en el archivo `energy.out` para distintos pasos temporales:

```
archivo de entrada: malla.inp
tamao de malla: 60
paso temporal: 0.05
constante de los resortes: 1
energia media: 2.63498
variacion media de energia registrada: 0.00890409
error relativo porcentual de energia: 0.337918 %
```

`second.inp` Malla de dimensión 60 con primer estado excitado (segundo modo normal de la malla) de la forma: $y = sin(2 * \pi * x) * sin(\pi * z)$ (m = 1,0, k = 1,0). En esta configuración la malla oscila con frecuencia constante, y su forma funcional se conserva a lo largo del tiempo. Esto es de esperar en un modo normal a extremos fijos. Resultados obtenidos en el archivo `energy.out` para distintos pasos temporales:

```
archivo de entrada: inputs/second.inp
tamao de malla: 60
```

```
paso temporal: 0.05
constante de los resortes: 1
energia media: 6.57983
variacion media de energia registrada: 0.0219224
error relativo porcentual de energia: 0.333176 %

archivo de entrada: inputs/second.inp
tamao de malla: 60
paso temporal: 0.01
constante de los resortes: 1
energia media: 6.56845
variacion media de energia registrada: 0.00189482
error relativo porcentual de energia: 0.0288473 %

archivo de entrada: inputs/second.inp
tamao de malla: 60
paso temporal: 0.2
constante de los resortes: 1
energia media: 6.81086
variacion media de energia registrada: 0.277094
error relativo porcentual de energia: 4.06841 %
```

`delta.inp` Malla de dimensión 99 con distribución tipo delta (con singularidad central) ($m = 1{,}0$, $k = 1{,}0$). Este tipo de perturbación excita todos los modos normales de la malla (están todos presentes en su descripción). Resultados obtenidos en el archivo `energy.out` para distintos pasos temporales:

```
archivo de entrada: inputs/delta.inp
tamao de malla: 99
paso temporal: 0.05
constante de los resortes: 1
energia media: 2.00462
variacion media de energia registrada: 0.00462159
error relativo porcentual de energia: 0.230547 %

archivo de entrada: inputs/delta.inp
tamao de malla: 99
paso temporal: 0.2
constante de los resortes: 1
energia media: 2.07532
variacion media de energia registrada: 0.0753156
error relativo porcentual de energia: 3.62912 %
```

`gaussian.inp` Malla de dimensión 25 con función gausseana bidimensional central de la forma: $y = e^{-(x-x0)^2} * e^{-(z-z0)^2}$ ($m = 1{,}0$, $k = 1{,}0$). Resultados obtenidos en el archivo `energy.out` para distintos pasos temporales:

```
archivo de entrada: inputs/gaussian.inp
tamao de malla: 25
```

```
paso temporal: 0.05
constante de los resortes: 1
energia media: 1.4274
variacion media de energia registrada: 0.0073164
error relativo porcentual de energia: 0.512569 %

archivo de entrada: inputs/gaussian.inp
tamao de malla: 25
paso temporal: 0.5
constante de los resortes: 1
energia media: 1.71538
variacion media de energia registrada: 0.291831
error relativo porcentual de energia: 17.0126 %
```

6.1. Conclusiones

La simulación es estable para pasos temporales por debajo de 0.1. Para pasos temporales mayores, los errores relativos se acercan al 5 %. En el último caso de prueba (la distribución gausseana) se observa que la integración arroja resultados inexactos para pasos temporales de 0.5.
El factor predominante sobre la estabilidad de la integración es el paso temporal, mientras que no se registran diferencias sustanciales entre las simulaciones del primero y el segundo modo de la malla. Incluso no se aprecian diferencias con la distribución delta, en la que se presentan excitados todos los modos de la malla. Esto sugiere que la integración es estable independientemente de los modos que se encuentren excitados en la simulación.