Documentación del paquete **FreeGeomPhy** v. 0.5 para Octave

Aarón Bueno Villares <abv150ci@gmail.com>*

10 de julio de 2012

Índice

1. Introducción

El paquete **FreeGeomPhy** es un paquete Octave con una colección de objetos para representación de funciones, (in)ecuaciones, sistemas de (in)ecuaciones, figuras geométricas y entidades físicas. Su utilidad es la de poder definir la geometría de un objeto físico y obtener el valor de las propiedades físicas deseadas en puntos de él.

En la versión actual de **FreeGeomPhy**, v. 0.5, no se ha desarrollado el manejo de las propiedades físicas de un objeto dado, por lo que sólo se dispone de figuras geométricas y las operaciones asociadas con él.

1.1. Definiciones

Figura Sinónimo de figura geométrica, definida como un conjunto de puntos que satisfacen una serie de propiedades modeladas mediante un sistema de ecuaciones e inecuaciones.

Entidad física Figura con una serie de carácterísticas o propiedades físicas, modeladas mediante una colección de funciones, una por característica física.

*Tutorizado por Cated^co. Pedro L. Galindo Riaño

2. Estructura de directorios de FreeGeomPhy

El paquete **FreeGeomPhy** dispone actualmente de 5 clases llamadas `hfunction`, `hequation`, `hsystem`, `hfigure` y `hcube`, cada una de las cuales están implementadas en las carpetas `@hfunction`, `@equation`, `@hsystem`, `@hfigure` y `@hcube`, tal y como exige exige Octave para la definición de clases.

Dentro de cada una de dichas carpetas tenemos implementadas el constructor y cada uno de los operadores y funciones definidas para la clase en cuestión.

Además, disponemos del *script* `loadPack`, que carga a las variables `hX, hx, hy, hz` y `hend`, y las funciones `hh`, `hn`, `hv`, `hc`, `he` y `hpi`.

Por último, dispone de la carpeta `doc` que contiene el código de ésta documentación.

Como podemos observar, todas las clases, variables y funciones extra contienen el prefijo **h**. El motivo es que la primera clase que se implementó, la correspondiente al manejo de funciones, y para evitar ambigüedad con la función `function` de Octave, se llamó `hfunction`, donde dicha **h** se puede leer o interpretar como *handle* (manejador), ya que una `hfunction` no es más que un manejador de una función. De hecho, las funciones se construyen haciendo uso de la función `inline` de Octave que permite crear manejadores a función escribiendo dicha función en una cadena de carácteres. Así, todas las clases, operadores y funciones se pueden leer o interpretar como **manejador a**.

3. Clase `hfunction`

Esta clase, aun no siendo la más compleja, sí que es la que dispone de un mayor número de detalles, y por tanto, es la clase más compleja y larga de documentar, así que esta sección será bastante más larga que las restantes secciones. Permite definir y transformar funciones matemáticas con una serie de operadores diseñados para tal fin. El nombre `hfunction` significa *function handle*, colocando la h de *handle* como prefijo, convenio que se seguirá usando para el resto de clases del paquete.

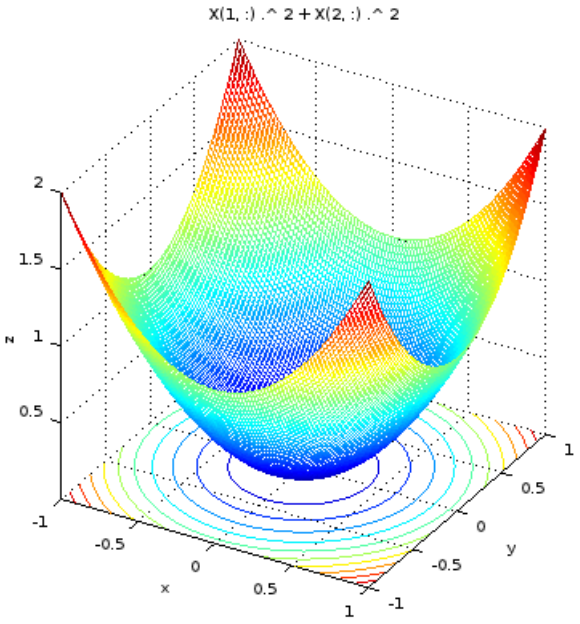
Esta clase se usa principalmente para dar un trato unificado al manejo de funciones por parte de clases más grandes como la de sistemas de ecuaciones o figuras como cubos y esferas. En la mayoría de los casos, quizás sea el usuario el que quiera definir su propia función para definir, por ejemplo, una de las caras de un cubo. En este punto, la clase `hfunction` permite crear una capa de abstracción entre los detalles internos y el paso de parámetros de las funciones definidas por el usuario, de modo que las figuras tratan a todas estas funciones y su paso de parámetros de forma unificada.

Para saber cómo se usa dicha clase, empecemos por un ejemplo de uso sencillo:

```
loadPack;

hf = hx .^ 2 + hy .^ 2;

plot3(hf, -1, 0.02, 1);
```



La primera línea de código solamente carga las variables `hx` y `hy` que usamos a continuación. En realidad carga algunas variables más, que veremos más adelante (véase §3.6).

Las variables `hx` y `hy` son dos `hfunction` que representan a las funciones $f(X) = X(1,:)$ y $f(X) = X(2,:)$. En consecuencia, `hf` representa a la función $f(X) = X(1,:)^2 + X(2,:)^2$, que es precisamente la función que estamos dibujando con la última línea. En ésta, le indicamos como primer parámetro la función a dibujar, y los tres parámetros restantes representan el intervalo sobre el que se calculará la función, tanto en la coordenada x como en la y. Es decir, en este caso, estamos dibujando la función en el intervalo $(-1, 1)$ en x y en el intervalo $(-1, 1)$ en y con incrementos de 0,02, equivale al vector `-1:0.02:1` (véase §3.9.3).

Las otras variables cargadas por `loadPack` son `hX`, que representa sin más a la función $f(X) = X$, `hz`, para $f(X) = X(3,:)$ y `hend`, cuyo uso veremos más adelante (véase §3.7).

3.1. Matriz de puntos

La clase `hfunction` supone que los datos vienen presentes en una matriz de dimensión $2 \times n, n \leq 1$ representando a una matriz de n puntos bidimensionales, es decir, los distintos puntos son las diferentes columnas $X(:, i)$ de la matriz, siendo el conjunto de coordenadas x la primera fila, y el de y la segunda. A su vez, una función, al evaluarse sobre una matriz de estas características, devuelve un vector fila con la coordenada z correspondiente a cada punto.

3.2. Otros parámetros adicionales

A las funciones se les pueden pasar otros parámetros o argumentos que quizás sean necesarios para realizar el cálculo: por ejemplo, definiendo una función paramétrica (familias de funciones) que necesita de un parámetro extra para quedar completamente especificado. Esto se detallará en el apartado que viene a continuación.

3.3. Las diferentes `hfunction`

Aunque la clase `hfunction` dispone de un constructor, no debe ser llamado directamente y, para ello, el paquete dispone de una serie de funciones auxiliares que devuelven objetos `hfunction` construidos correctamente:

hc(n) Función constante $f(X) = n$, siendo n un número cualquiera.

hv(v) Función identidad $f(X, v) = v$, siendo v un nombre de variable válido.

hh(h, v_1, ..., v_n) Crea la función $f(X, v_1, \ldots, v_n) = h(v_1, \cdots, v_n)$. El parámetro h debe ser un manejador de función válido de octave, y los parámetros $v_1, \ldots, v_n$ nombres de variables válidos. El nombre de la función puede leerse como manejador a manejador o `handle-handle`.

he Devuelve la función constante $f(X) = e$, siendo e el número de Euler.

hpi Devuelve la función constante $f(X) = \pi$.

Es importante advertir que en todos estos casos, tenemos a X como primer parámetro. Esta es una asumpción necesaria ya que es el parámetro de referencia, tanto para poder dibujar la función como para trabajar con comodidad con sistemas de ecuaciones y figuras. A su vez, es importante tener en cuenta que estas funciones son tomadas siempre como funciones matemáticas y no como funciones de Octave a modo de procedimiento. Este punto es importante para saber qué tipo de modificadores y operadores podrán ser usados con él.

Veamos ahora como usar estas funciones:

```
const_fun = hc(3)
var_fun = hv("k")
fun_fun = hh(@mod, "a", "b")
other_fun = const_fun + var_fun + fun_fun * he * hpi
```

Que, al ejecutarlo, nos muestra en la salida del intérprete:

```
f(X) = 3
f(X, k) = k
f(X, a, b) = mod(a, b)
f(X, k, a, b) = 3 + k + mod(a, b) * e * pi
```

En realidad, podemos trabajar con variables y constantes de una forma mucho más sencilla, por ejemplo, todas estas intrucciones son válidas:

```
const_fun + 3 + fun_fun
fun_fun ^ "h"
var_fun + "z" + 3 * "b"
```

Que produce las siguientes salidas:

```
f(X, a, b) = 3 + 3 + mod(a, b)
f(X, a, b, h) = mod(a, b) ^ h
f(X, k, z, b) = k + z + 3 * b
```

Esto nos podría llegar a concluir que las funciones `hn` (función constante) y `hv` (función identidad) no sirven para nada ya que no necesitamos llamarlas directamente como acabamos de ver. Pero hay que tener cierto cuidado, ya que la clase `hfunction` transforma todos los resultados intermedios a objetos de la clase, y luego los combina usando los operadores, pero si en la expresión no hay ningún objeto `hfunction`, los operadores son tratados directamente por Octave. Efectivamente, en este caso la expresión no sería de tipo `hfunction` y la clase no puede hacer nada por evitarlo, puesto que no está presente en la expresión (no podemos pretender que $3 + 4$ no sea 7 y sí $f(X) = 3 + 4$).

Aún es más, debido a que los operadores tienen, en su gran mayoría de casos, asociatividad izquierda, una expresión como `3 + "k" + fun_fun` se resolvería como `((3 + "k") + fun_fun)`, y luego al resultado indeseable `110 + fun_fun`, ya que Octave promociona `"k"` a entero, cuyo valor ASCII es 107, y le suma 3, produciendo la función no pretendida `f(X, a, b) = 110 + sum(a, b)`. Para estos casos extraños es cuando puede ser útil las funciones `hn` y `hv`.

De igual forma, puedo ejecutar una expresión como `fun_fun + @sum` que también es aceptado sin discusión, pero, debido a que no hay forma aquí de especificar los parámetros de la función `sum`, se construiría el objeto `f(X, a, b) = mod(a, b) + sum()`.

3.4. Cálculo de las funciones

Una vez que tenemos un objeto `hfunction`, para calcular sus valores se procede como con una función normal:

```
1 loadPack;
2
3 myFun = hx .^ 2 + hy .^ 2;
4
5 myFun([1 2; 3 4])
```

Que produce como salida al vector $(1^2 + 3^2, 2^2 + 4^2) = (10, 20)$. En el caso de que al primer parámetro X no lo necesitemos, como en el inútil caso anterior $other\_fun \equiv f(X, h, a, b) = 3 + h + mod(a, b)$, podemos ejecutar sin vergüenza $f([], 3, 8, 5)$, que devolvería 9, aunque en este caso significa que alguna de las variables h, a o b debería ser llamada X, los argumentos adicionales deben ser usados (o se debe pensar en ellos) como parámetros de la función a computar y no como los propios puntos del dominio de definición de la misma.

3.5. Órden de los argumentos

Si solamente trabajamos con las hfunciones (así lo denominaré de ahora en adelante) `hx`, `hy`, `hn` o constantes, y cualquier otra que no implique la introducción de nuevas variables a la hfuncion en construcción, toda hfunción tendrá un único parámetro X, y a la hora de calcular los valores de la función éste será el único argumento a considerar. Si al «llamar» a la hfunción, y pongo «llamar» entrecomillado debido a que una hfunción no es un procedimiento —o no está diseñado para pensar en él de esa forma, como ya hemos indicado—, con más parámetros de los específicados, éstos serán ignorados, del mismo modo que pasa con las llamadas a funciones normales de Octave.

En caso de que una hfunción tenga parámetros adicionales además de X, el órden de estos parámetros vendrá determinado por el órden en que aparezcan en la descripción de la misma, con la excepción del parámetro X, que siempre va el primero. Por ejemplo: `hx + 'h' + hh(@sum, a, b)` tendrá como parámetros X, h, a, b y en ese órden. Los parámetros de una hfunción se pueden consultar visualizando la definición de la función (sin poner punto y coma al final) o gracias a la función `argnames(h)`, que recibe una hfunción `h` y devuelve la celda de cadenas de sus parámetros, que coinciden en órden con el de evaluación.

Si a una hfunción le vamos añadiendo mediante el uso de operadores (suma, división, etcétera) nuevas hfunciónes, los parámetros se irán combinando mantiendo su órden relativo de aparición. Por ejemplo, si sumamos dos hfunciones, basta visualizar cada una de ellas y luego la hfunción recién construida para comprobar esta afirmación.

En caso de que al combinar dos hfunciones mediante un operador, como hemos descrito antes, tengan variables en común, se eliminarán automáticamente las repeticiones a partir de la «primera aparición» sin cambiar el órden relativo de las mismas.

3.6. Dibujado de las funciones

A la hora de dibujar una hfunción, disponemos de las sobrecargas de dos funciones bien conocidas: `plot` y `plot3`, usadas respectivamente para dibujar una función cuándo esta es 2D y 3D. Si se usa `plot3` para dibujar una función que en realidad es bidimensional, como consecuencia se dibujará una función tridimensional donde la función 2D es repetida en y.

La sintáxis de ambas funciones —tanto de `plot` como `plot3`, que llamaremos conjuntamente `plotx`— es la siguiente:

```
plotx(hf, xmin, step, xmax, params...)
plotx(hf, xmin, step, xmax, title, params...)
```

Donde `hf` es una hfunción, `xmin, step` y `xmax` definen el intervalo $xmin : step : xmax$ como vimos en la introducción de la clase, `title` (opcional) el título del dibujo (por defecto es la definición de la hfunción) y los parámetros (opcional) que puede ser ninguno, uno o más de uno separados por comas, que definen los parámetros adicionales de la hfunción.

3.7. Operador {}

Si aplicamos este operador a una hfunción con un argumento, por ejemplo, `hx{2}`, obtendremos la hfunción $f(X) = X(1,:)(2)$, o si hacemos `hv("h"){2, 3}` obtendremos $f(X,h) = h(2,3)$. Incluso podemos hacer `hv("h"){2, :, 4}` obtendremos $f(X,h) = h(2,:,4)$.

Es decir, el operador {} es equivalente al operador de acceso matricial (). Se ha usado el operador {} para simular el acceso matricial debido a que () aplicado a una hfunción se usa para evaluarla (calcular sus valores).

Por último, hace falta comentar dos pequeñas complejidades técnicas respecto a los operador `end` y `:` (por ejemplo, si queremos especificar un intervalo como `2:3`). Mientras que un número se mantiene como tal cuando se pasa como argumento a la sobrecarga del operador {} que hemos definido para la clase, y el operador `:` se transporta como un carácter, los operadores `:` (en una expresión) y `end` se evalúan antes de ser pasados como argumentos. Por tanto, si hacemos `hv("h"){end}` obtendremos $f(X,h) = h(1)$, debido a que una hfunción tiene tamaño 1, ya que es un único objeto y no una matriz o celda de objetos.

Lo mismo nos pasa con una llamada como `hv("h"){2:5}` que nos devolverá un error ya que la sobrecarga {} espera recibir como argumentos valores simples y no vectores, que es el resultado de la evaluación de `2:5`.

Pero también tenemos soluciones para ésto último:

`end` Disponemos de la variable *hend*, que no es más que la cadena `"end"`. Esta es otra de las variables que son cargadas cuando ejecutamos a `loadPack`. Con él, podemos hacer `hv("h"){hend}` para así obtener la hfunción $f(X,h) = h(end)$.

`hc(a, b, [c])` Esta función es la que nos permite definir intervalos de Octave. Puede ser llamada con dos argumentos para representar `a:b` o con tres para `a:b:c`. La `c` de `hc` significa *colon* que es el nombre en inglés del operador `:` (el nombre en inglés de dicho carácter). Con él podemos ahora escribir `hv("h"){1, hn(2, 3)}` para obtener $f(X,h) = h(1,2:3)$.

Igualmente, ambos recursos pueden ser combinados para obtener expresiones de acceso a vector/matriz como `hv("h"){1, hn(2, 4, hend), hend}` para tener $f(X,h) = h(1,2:4:end,end)$.

Lo que no tenemos por ahora son sobrecargas para construir expresiones que contengan a `end`, como por ejemplo `end - 1`, ya que `hend` es solamente una cadena. Para conseguir este tipo de expresiones, tendríamos que pasarle la expresión directamente como una cadena, por ejemplo, `hv("h"){1, "end - 1"}` y ya tendríamos a nuestra esperada función $f(X,h) = h(1,end-1)$.

3.8. Funciones como operadores

Además de los operadores conocidos +, ∗, etcétera, hay implementadas un conjunto de funciones como `log`, `exp`, `sin`, etcétera. En lo que sigue, estas funciones serán llamadas **funciones operadoras** para no confundirlas con otras funciones «especiales» como `plot` o `argnames`.

3.9. Operadores y funciones implementadas actualmente

A continuación mostramos una lista de todos los operadores y funciones implementadas actualmente. Cada una de ellas tiene su correspondiente fichero de definición dentro de la carpeta `@hfunction`.

3.9.1. Operadores

Los operadores sobrecargados para las hfunciones son todos aquellos que permite sobrecargar Octave y que tienen algún sentido para su uso con hfunciones.

`plus` +	`times` .*	`power` .^
`minus` -	`mtimes` *	`mpower` ^
`uplus` + (unario)	`rdivide` ./	`ctranspose` '
`uminus` - (unario)	`mrdivide` /	`transpose` .'

También tenemos sobrecargados los operadores () y {}, que en Octave se definen como operadores de indexación y se sobrecargan mediante la función *subsref*.

Por último, tenemos sobrecargados los operadores de comparación pero que, en realidad, construyen hecuaciones y no hfunciones. Los veremos en la siguiente sección. Los operadores de comparación sobrecargados son los siguientes:

`lt` < `le` <= `eq` == `ne` != `ge` >= `gt` >

3.9.2. Funciones operadoras

Las funciones operadoras implementadas son las de exponenciación, logarítmicas, la raíz cuadrada y las trigonométricas. A continuación mostramos ambas listas:

Funciones exponenciales y logarítmicas y de raíz cuadrada Son las siguientes:

- log
- log2
- log10
- exp
- sqrt

Funciones trigonométricas Son las siguientes, que corresponden a todas las sobrecargas de las funciones disponibles en el paquete estándar de Octave:

- sin
- csc
- atan
- sinh
- csch
- atanh
- cos
- cot
- asec
- cosh
- coth
- asech
- tan
- asin
- acsc
- tanh
- asinh
- acsch
- sec
- acos
- acot
- sech
- acosh
- acoth

3.9.3. Funciones especiales

Las funciones especiales sobrecargas son las siguientes:

`plot` Para dibujar una hfunción 2D.

`plot3` Para dibujar una hfunción 3D.

`display` Para visualizar el contenido de la hfunción.

`char` Para convertir la hfunción a una cadena de carácteres (solo el cuerpo de la función).

`argnames` Variables de la función.

3.9.4. Otras funciones auxiliares

Mostramos, por último, otras funciones relacionadas con las hfunciones pero que no constituyen funciones propias de la clase. De hecho, todas ellas están implementadas fuera, en el directorio raíz de **FreeGeomPhy**:

`loadPack` Carga las variables `hX, hx, hy, hz` y `hend`.

`hv` Función identidad.

`hc` Función constante.

`hh` Función-manejador (*handle-handle*).

`hc` Para definir "intervalos" (nuestra particular sobrecarga del operador *colon*, `:`).

`he` Para definir a la contante e (número de Euler) como una hfunción.

`pi` Para definir a la constante π como una hfunción.

3.10. Tres ejemplos paramétricos

Para ejemplificar, vamos a usar la clase hfunción para calcular tres funciones, dos bidimensionales y una tridimensional, todas paramétricas.