Tema 5

Visualización

Para el estudio de este tema familiarícese con el uso del WXMAXIMA para representaciones gráficas en dos y tres dimensiones.

- Representaciones gráficas:
  - Gráficas en 2D: `plot2d(expresión,[variable, límite inferior, límite superior], opciones)`. Por ejemplo: `plot2d(sin(x),[x,-%pi,%pi])`.
  - Gráficas en 3D: `plot3d(expresión,[variable1, lím. inferior 1, lím. superior1], [variable2, lím. inferior1, lím. superior1], opciones)`. Por ejemplo: `plot3d(sin(x^2+y^2),[x,-%pi,%pi],[y,-%pi,%pi])`.
  - Cada uno de estos comandos tiene multitud de opciones para representar gráficas de distintos tipos. Consulte la ayuda del WXMAXIMA para explorar las posibilidades de estos comandos.

Los comandos `plot2d` y `plot3d` del MAXIMA funcionan bien, pero tienen el inconveniente de no permitirnos la visualización simultánea de varias gráficas distintas, ya que al usar estos comandos cualquier gráfica que hagamos se representará en una única ventana, ocupando el sitio de cualquier gráfica que hayamos visualizado previamente. Para resolver este problema una posibilidad es guardar los resultados en un archivo antes de llamar a `plot2d` o `plot3d` para realizar la gráfica siguiente. Otra alternativa es emplear los comandos `wxplot2d` y `wxplot3d` del WXMAXIMA. Estos comandos en lugar de emplear una ventana única para todas las representaciones gráficas, generan una ventana nueva (integrada) con cada nueva representación gráfica que hagamos, lo que nos permite la visualización simultánea de todas las gráficas que hayamos generado en una sesión, sin necesidad de guardarlas en archivos. De todas formas, en muchos casos querremos incluir las gráficas generadas en una sesión de cálculo en algún documento sin pérdida de calidad de la imagen, para ello es fundamental saber cómo guardar gráficas en archivos, para poder importarlas posteriormente con el procesador de textos que estemos usando.

5.1. Problemas resueltos

1. Emplee la función `plot2d()` para visualizar en una gráfica una función y sus correspondientes desarrollos en serie de Taylor en torno a un punto dado a orden cada vez superior (por ejemplo puede considerar la función $\sin(x)$ y sus desarrollos sucesivos para n entre 0 y 10).

Soluciones

Ejercicio 1

Emplee la función `plot2d` para visualizar en una gráfica una función y sus correspondientes desarrollos en serie de Taylor en torno a un punto dado a orden cada vez superior (por ejemplo puede considerar la función $\sin x$ y sus desarrollos sucesivos para n entre 0 y 10).

Lo más inmediato para este ejercicio es usar directamente la función `plot2d` del WXMAXIMA. En lugar de la función que se sugiere en el enunciado vamos a tomar como ejemplo la función $\sin x + \cos x$, y hacemos su desarrollo en torno a $x = 0$ hasta orden entre $n = 0$ y $n = 10$. Para ello en primer lugar cargamos la función definida en el ejercicio anterior dentro de una sesión con WXMAXIMA:

```
kill(all);
path : ".../Fisica-Computacional-1/Maxima/";
batchload(concat(path,"taylor_list.mc"));
```

posteriormente hacemos:

```
f(x) := sin(x) + cos(x);
orden : 10$
aux : taylorlist(f(x),x,0,orden)$
```

Estamos interesados en ver estos resultados en un cierto entorno alrededor de $x = 0$ (pongamos el intervalo $[-2\pi, 2\pi]$). Para no tener que escribir `[x, -2*%pi, 2*%pi]` muchas veces, asignamos ese dato a una variable (`dominiox`) y sencillamente le pedimos al WXMAXIMA que haga la representación gráfica de `aux`:

```
dominiox : [x, -2*%pi, 2*%pi];
plot2d(aux, dominiox);
```

Si hacemos esto obtenemos una gráfica en la que realmente no podemos ver el grado de aproximación de los desarrollos en serie de Taylor. Los desarrollos en serie de Taylor son precisos en un cierto entorno alrededor del punto respecto al que desarrollamos ($x = 0$ en este caso), pero una vez nos salimos de ese entorno crecen de manera muy rápida, alejándose de la función que queremos aproximar. Para poder ver algo tenemos que decirle al WXMAXIMA que queremos ver los resultados limitando el rango de ordenadas de la gráfica al conjunto de valores de y en el que varía $f(x)$ cuando x toma valores en el intervalo de interés (en este caso `dominiox`). Definimos entonces el rango de valores de y en el que queremos ver la gráfica y le volvemos a pedir a WXMAXIMA que nos muestre los resultados:

```
dominioy : [y,-2,2];
plot2d(aux, dominiox, dominioy);
```

Ahora ya se ve por dónde van los desarrollos en serie de Taylor en la inmediata vecindad del punto de interés ($x = 0$), y el WXMAXIMA nos informa que “algunos valores han sido cortados”, es decir, caen fuera de la ventana `dominiox` $\times$ `dominioy` que estamos representando.

El siguiente problema que vemos en esta gráfica es que la información de qué función corresponde a cada línea ocupa demasiado espacio. Para evitar esto le pedimos al WXMAXIMA que en lugar de indicar la función completa que corresponde a cada color nos indique sólo el orden del desarrollo correspondiente. Como la lista `aux` contiene los desarrollos de orden $n = 0$ hasta orden $n = 10$, para que el WXMAXIMA sólo escriba el orden correspondiente lo indicamos mediante la opción `legend` de la función `plot2d`:

```
plot2d(aux, dominiox, dominioy, [legend, makelist(i - 1, i, 1, length(aux
   ), 1)]);
```

Como puede verse le pasamos a la opción `legend` una lista de valores que van desde 0, hasta `length(aux) - 1`, (el último 1 indica que queremos esa lista a intervalos de 1), dado que el primer elemento de `aux` contiene el desarrollo de orden 0, el segundo elemento el de orden 1, y así sucesivamente.

Ahora ya se ve bastante claro por dónde va cada desarrollo. Lo único que falta es incluir la función $f(x)$, ya que el enunciado pide representar esa función con todos los desarrollos. Para ello por medio de la función `append` del WXMAXIMA construimos una lista de funciones cuyo primer elemento es $f(x)$ y cuyos restantes elementos sean los desarrollos definidos en `aux`. Análogamente empleamos la función `append` para generar la lista de "nombres" que queremos que aparezcan en la gráfica, de tal forma que para la función $f(x)$ aparezca la propia función y para cada uno de los elementos de `aux` aparezca el valor del orden correspondiente.

```
plot2d(
       append( [f(x)], aux ), dominiox, dominioy,
       [legend, append([f(x)], makelist(i-1, i, 1, length(aux), 1))]
);
```

La siguiente mejora que se nos ocurre es que nos gustaría que la función *exacta* $f(x)$ apareciese más destacada sobre los desarrollos en serie de Taylor. Por ejemplo, podríamos pintar la gráfica de $f(x)$ con un grosor de línea mayor (pongamos 4). Para indicar el grosor de línea que queremos empleamos la opción `style` de la función `plot2d`, y volvemos a emplear `append` para generar la lista completa de estilos que queremos para generar la gráfica final, con grosor 4 para $f(x)$ y grosor 1 para los desarrollos:

```
plot2d(
       append( [f(x)], aux), dominiox, dominioy,
       [legend, append([f(x)], makelist(i-1, i, 1, length(aux), 1))],
       append( [style], [[lines, 4]], makelist([lines, 1], i, 1, length(
          aux), 1) )
);
```

Una vez hemos visto cómo representar $f(x)$ junto a sus desarrollos podemos definir una función (`vertaylorlist`) que haga todo este trabajo para poder usarla posteriormente. Incluimos a continuación el listado del archivo `ver_taylor_list.mc` que contiene las funciones `vertaylorlist` y `vertaylorlistsave`:

```
/* FUNCION "vertaylorlist"
    Realiza la gr\'afica de una funci\'on junto con sus desarrollos en
       serie de
```

```
    Taylor en torno a un punto hasta un orden dado, generados con
       taylorlist
    Input:
      f = la funci\'on a estudiar
      x = variable respecto de la que se hace el desarrollo
      x0 = punto en torno al que desarrollamos
      n = orden m\'aximo del desarrollo
      xmin, xmax = valores m{\'\i}nimo y m\'aximo de x a considerar en la
         gr\'afica
      ymin, ymax = valores m{\'\i}nimo y m\'aximo de y a considerar en la
         gr\'afica
    Output:
      gr\'afica de f(x) junto a sus desarrollos */

vertaylorlist(f, x, x0, n, xmin, xmax, ymin, ymax) := block( [aux,
   dominiox, dominioy],

    aux : taylorlist( f, x, 0, n),

    dominiox : [x, xmin, xmax],

    dominioy : [y, ymin, ymax],

    plot2d( append( [f], aux ), dominiox, dominioy,

      append( [legend, string(f)], makelist(string(i-1), i, 1, length(aux)
         , 1) ),

      append( [style], [[lines, 4]], makelist( [lines, 1], i, 1, length(
         aux), 1) )

    )

)$

/* FUNCION "vertaylorlistsave"
    Definimos la misma funci\'on con el argumento adicional "filename"
       para guardar el resultado en
    un archivo con formato eps.
 */

vertaylorlistsave(f, x, x0, n, xmin, xmax, ymin, ymax, filename) := block
    ( [aux, dominiox, dominioy],

    aux : taylorlist( f, x, 0, n),

    dominiox : [x, xmin, xmax],
```

```
    dominioy : [y, ymin, ymax],

    plot2d( append( [f], aux ), dominiox, dominioy,

      append( [legend, string(f)], makelist(string(i-1), i, 1, length(aux)
         , 1) ),

      append( [style], [[lines, 4]], makelist( [lines, 1], i, 1, length(
         aux), 1) ),

      [ps_file, filename]

    )

)$
```

Una vez definida esta función podemos experimentar muy fácilmente cómo funcionan los desarrollos en serie de Taylor, probando con distintas funciones y distintos órdenes máximos. A todo esto, hemos supuesto en todo momento que la función $f(x)$ era analítica, si no es así los desarrollos en serie de Taylor no convergen a $f(x)$. En particular, si $f(x)$ tiene una singularidad en el punto en que desarrollamos al intentar calcular el desarrollo el wxMaxima nos dará un error. Una vez más, para usar ordenadores de una manera eficiente lo mejor es saber qué es lo que estamos haciendo en todo momento.

Además de la función `vertaylorlist`, en el archivo `ver_taylor_list.mc` hemos definido la función `vertaylorlistsave`, que hace exactamente lo mismo, pero en lugar de mostrar el resultado en la pantalla lo guarda en un archivo con formato `eps`. El argumento adicional de la función `vertaylorlistsave` es precisamente el nombre del archivo donde queremos guardar la gráfica, este argumento debe ser una "cadena de caracteres" (un *string*). Por ejemplo, se puede probar lo siguiente:

```
vertaylorlist( log(1+x), x, 0, 5, 0, 3, -0.5, 2 );
vertaylorlistsave( log(1+x), x, 0, 5, 0, 3, -0.5, 2, "figura-1.eps" );
```

Hemos escogido el formato `.eps` (*encapsulated postscript*) porque es el más habitual cuando se trabaja en LaTeX. La principal ventaja del `.eps` es que es un formato vectorial, de modo que la calidad es muy alta y ocupa muy poco, aparte de ser estándar y gratuito.