Tema 6

Solución analítica y numérica de ecuaciones

Las instrucciones básicas del MAXIMA para resolver ecuaciones y sistemas de ecuaciones son:

- Ecuaciones algebraicas: `solve(ecuación, variable).`
- Sistemas de ecuaciones algebraicas: `solve(lista de ecuaciones, lista de variables).`
- Solución numérica de ecuaciones (método de Newton): primero `load("newton1")`, y posteriormente `newton(ecuación, variable, valor inicial, precisión).`

En este tema veremos la aplicación de dichas instrucciones para el cálculo de funciones definidas de forma implícita mediante ecuaciones.

6.1. Función implícita

En general una ecuación del tipo $F(x, y) = 0$ define de manera implícita a la variable y como función de x (y análogamente a la x como función de y). Para que esta ecuación defina una función de forma implícita deben cumplirse ciertas condiciones, tal y como establece el teorema de la función implícita; por ejemplo, para que esta condición defina una única función $y = y(x)$ es necesario que la solución sea única, y que las derivadas parciales de F sean finitas y distintas de cero. En estas notas no entraremos en los detalles del teorema de la función implícita, y nos centraremos directamente en la aplicación del WXMAXIMA para el cálculo de funciones definidas de forma implícita por medio de ecuaciones.

6.1.1. Ejercicios propuestos

1. Dada una ecuación $F(x, y) = 0$ escriba una función usando el comando `solve` que despeje de forma analítica y en función de x, y defina de esta forma la correspondiente función $y = f(x)$.

2. Con mucha frecuencia aparecen ecuaciones en las que no es posible realizar esta operación analíticamente, es ese caso no hay más remedio que resolver

la ecuación numéricamente, lo cual sólo es posible una vez que hemos suministrado un valor numérico concreto para x. Escriba una función por medio del comando `newton()` que proporcione y en función de x resolviendo la ecuación $F(x, y) = 0$ numéricamente.

Notas:

- Un dato necesario para resolver este ejercicio es el conjunto de funciones concretas $F(x, y)$ que definen las correspondientes ecuaciones que queremos resolver. Para ello puede probar con cualquier función F, pero le recomendamos que tome (entre otros casos) el ejemplo concreto que mencionamos más abajo.

- Dependiendo de la función $F(x, y)$ que consideremos, la anterior ecuación puede tener o no solución y en caso de tenerla esta podrá o no ser única. Es conveniente tener esta información antes de intentar resolver la ecuación, para ello puede emplear el comando `plot3d()`.

- En el caso de la solución numérica conviene saber que el método de Newton (que ya estudiará en la asignatura de Métodos Numéricos) es un método iterativo que, por medio de un algoritmo, va produciendo aproximaciones cada vez más precisas de la solución buscada. Para funcionar este método precisa que el programador introduzca un primer valor aproximado de la solución (el tercer argumento de la función `newton`). El éxito del método depende, entre otras cosas, del primer valor con el que iniciamos las iteraciones. En algunos casos el método sólo converge a la solución que buscamos si el primer valor que nosotros suministramos no está demasiado alejado de la solución buscada. Para calcular esta primera aproximación a la solución lo más sencillo es visualizar $F(x, y)$ por medio de `plot3d()`.

- Un consejo: considere funciones $F(x, y)$ tales que la ecuación $F(x, y) = 0$ tenga una solución única. En problemas con múltiples soluciones el método de Newton no siempre converge, lo cual es un tema muy interesante que estudiará en la asignatura de Métodos Numéricos.

6.2. Desarrollos en serie de Taylor de funciones definidas de forma implícita

En aquellos casos en los que no es posible despejar y en función de x de la ecuación $F(x, y) = 0$, el teorema de la función implícita nos garantiza que si para unos determinados x_0 e y_0 se cumple $F(x_0, y_0) = 0$ y $F(x, y)$ es continua y diferenciable en un entorno del punto (x_0, y_0), entonces la condición $F(x, y) = 0$ define la correspondiente función $y = f(x)$ al menos en un entorno pequeño del punto (x_0, y_0). (La extensión del entorno en el que puede definirse esta función (determinado por la función F) es un resultado muy interesante que se estudia en la asignatura de Análisis Matemático).

6.2.1. Ejercicios propuestos

1. Utilice el programa de la sección anterior para generar una aproximación a la función $y = f(x)$ en forma de serie de Taylor (de orden n arbitrario), válida en un entorno de x_0.

2. Visualice los resultados obtenidos mediante gráficas por medio de la función `plot2d()`.

Notas:

Para realizar este ejercicio deberá calcular las derivadas (desde la primera hasta la n-ésima) de la función $y = f(x)$ en el punto x_0. Para ello debe derivar respecto de x la relación $F[y(x), x] = 0$ utilizando `diff()`, posteriormente deberá utilizar `subst()` para sustituir x e y por x_0 e y_0 y finalmente deberá usar `solve()` para despejar la derivada buscada. Esta operación deberá hacerse de manera secuencial, desde la primera derivada hasta la de orden n.

En principio todos sabemos cómo se hace el desarrollo en serie de Taylor de una función definida de forma explícita

$$y = f(x)$$

para el desarrollo en torno a un punto x_0 sencillamente tenemos que calcular la siguiente suma (suponiendo que $f(x)$ es analítica):

$$y = \sum_{n=0}^{\infty} \frac{1}{n!} \left. \frac{d^n f}{dx^n} \right|_{x_0} (x - x_0)^n$$

Esto es precisamente lo que hace la función definida en el apartado 4.1, con la única salvedad de que en lugar de calcular el desarrollo completo (hasta $n = \infty$) lo estamos cortando en un orden finito determinado (eso es lo que se llama un *desarrollo truncado*). Es evidente que aunque matemáticamente se manejan formalmente de manera totalmente rutinaria multitud de objetos que involucran infinitos, como números con precisión infinita ($\pi, e, \sqrt{2}, \ldots$) o series infinitas, computacionalmente debemos conformarnos con las correspondientes aproximaciones finitas.

El problema que vamos a tratar en estas indicaciones es cómo se calcula un desarrollo en serie de Taylor para una función definida de forma implícita. Una función $y = f(x)$ está definida de forma implícita cuando la condición que define la función, dada de forma genérica por la ecuación

$$F(x, y) = 0$$

no puede despejarse para y como función de x.

El teorema de la función implícita indica las condiciones que debe cumplir $F(x, y)$ para que esa ecuación defina efectivamente a y como función de x, pero dado que no es el tema de esta asignatura no entraremos en los detalles de dicho teorema. Supongamos sencillamente que en un entorno del punto x_0 se cumple el teorema de la función implícita, por tanto en un entorno de ese punto existe la función $y = f(x)$ cuyo desarrollo queremos calcular. Para calcular el desarrollo lo que hay que hacer es lo siguiente:

- Calcular el primer término del desarrollo ($y_0 = f(x_0)$) resolviendo numéricamente la ecuación

$$F(x_0, y_0) = 0$$

- Para los demás términos del desarrollo sustituimos formalmente $y = f(x)$ dentro de $F(x, y)$ y calculamos las derivadas de $f(x)$ por medio de la regla de la cadena

$$\begin{array}{rcl} F(x, f(x)) = 0 & & \\ F_x + F_y f'(x) = 0 & \Rightarrow & f'(x) = -F_x/F_y \\ \dfrac{dF_x}{dx} + \dfrac{dF_y}{dx} f'(x) + F_y f''(x) = 0 & \Rightarrow & f''(x) = -\left(\dfrac{dF_x}{dx} + \dfrac{dF_y}{dx} f'(x)\right)/F_y \\ \ldots\ \ldots\ \ldots & \Rightarrow & \ldots\ \ldots\ \ldots \end{array}$$

  donde estamos empleando la notación siguiente

$$F_x = \frac{\partial F(x, f(x))}{\partial x}, \qquad F_y = \frac{\partial F(x, f(x))}{\partial f(x)}$$

- y así sucesivamente para las derivadas de orden superior.

Lo que es importante es darse cuenta de que para calcular el desarrollo sólo tenemos que resolver numéricamente una única ecuación (para calcular y_0), ya que aunque sea imposible despejar y en función de x, siempre podemos derivar por medio de la regla de la cadena y despejar de manera trivial las sucesivas derivadas de y respecto de x, que quedan evaluadas de manera analítica (no numérica). Por cierto, una de las condiciones necesarias para que se cumpla el teorema de la función implícita es precisamente que la F_y que nos aparece en todos los denominadores no se anule.

Ejemplos de funciones $F(x, y)$

En los ejercicios planteados en este tema se pide hacer una serie de funciones que resuelvan la ecuación $F(x, y) = 0$, análiticamente si es posible o numéricamente en caso contrario, pero no se propone ninguna función $F(x, y)$ concreta.

Para fijar ideas proponemos trabajar sobre los siguientes casos concretos:

- Caso con solución analítica: $F(x, y) = x^2 + y^2 - 1$.

- Caso sin solución analítica: $F(x, y) = e^y - xy$. En este caso hay que tener en cuenta que esta ecuación tiene solución única para $-\infty \leq x \leq 0$, no tiene solución para $0 < x < e$ y tiene dos soluciones para $x > e$.

Aunque para fijar ideas le proponemos trabajar sobre estos ejemplos de funciones $F(x, y)$, las soluciones a los ejercicios deben estar programadas para operar sobre cualquier función $F(x, y)$.

6.3. Función inversa

Como caso particular del teorema de la función implícita tenemos el teorema de la función inversa, que nos garantiza que la relación $y = f(x)$ define a x como función de y, al menos localmente (relación que se obtiene resolviendo para x la ecuación $f(x) - y = 0$). (La extensión del entorno en el que puede definirse la función $x = f^{-1}(y)$ (determinado por la función f) es un resultado muy interesante que se estudia en la asignatura de Análisis Matemático). Cuando no es posible despejar x en función de y analíticamente no hay más remedio que hacerlo numéricamente, para ello en este curso usaremos el comando `newton()`.

6.3.1. Ejercicios propuestos

1. Dada la relación $y = f(x)$, aplique el programa anterior para obtener una aproximación en forma de serie de Taylor para la función $x = f^{-1}(y)$, válida en un entorno de y_0.
2. Visualice los resultados obtenidos mediante gráficas por medio de la función `plot2d()`.

6.4. Funciones del Maxima que debemos aplicar

Para los ejercicios de este tema resulta conveniente emplear las siguientes funciones del WXMAXIMA:

- `solve`
- `sublis`
- `newton`, para ello previamente hay que cargar el paquete “newton” por medio de `load(newton1);`
- `subst`
- `diff`
- aparte de las que ya hemos visto `block`, `append`, `makelist`, `plot2d`

6.5. Problemas resueltos

Ejercicio 1

Dada una ecuación $F(x, y) = 0$ escriba una función usando el comando `solve` que despeje de forma analítica y en función de x, y defina de esta forma la correspondiente función $y = f(x)$.

En este ejercicio lo único que hay que hacer es usar las funciones `solve` y `sublis` del wxMaxima. Si la ecuación $F(x, y) = 0$ puede resolverse analíticamente (lo cual no siempre ocurre), la función `solve(F(x, y) = 0, y)` resuelve la ecuación y nos devuelve una lista de ecuaciones en las que la variable y está despejada, cada uno de los elementos de esta lista corresponde a una de las soluciones de la ecuación. La función `sublis` aplicada al anterior output sustituye la ecuación que hemos obtenido para la y ya despejada. Definimos entonces la función `solvesust`:

```
solvesust(F, y) := sublis( solve( F = 0, y), y )$
```

Haciéndolo de esta forma vemos que cuando la ecuación $F(x, y) = 0$ tiene más de una solución (p. ej. $x^2 + y^2 - 1 = 0$) sólo obtenemos la primera, ya que `sublis` sustituye la primera solución y prescinde de todas las restantes, a pesar de que `solve` nos genera la lista completa de soluciones. Esto es un inconveniente, ya que si la ecuación que tenemos que resolver tiene más de una solución, en principio nos interesa obtener una lista con todas las soluciones posibles.

Una posible forma de conseguir esta lista de soluciones es aplicando `sublis` de manera sucesiva sobre cada una de las soluciones encontradas con `solve`, para ello modificamos `solvesust` de la siguiente forma, listado del archivo `solve_sust.mc`:

```
/* FUNCION "solvesust"
   Resuelve anal{\'\i}ticamente la ecuaci\'on F(x, y) = 0 para
   la variable y, devolviendo la soluci\'on
   Input:
     F = la funci\'on que define la ecuaci\'on F = 0
     y = variable que queremos despejar
   Output:
     lista de soluciones de F = 0 para y */

solvesust(F, y) := block( [soluciones, soly],

   /* asignamos a la variable local soluciones la lista de soluciones de
       F = 0 */

   soluciones : solve( F = 0, y),

   /* asignamos la primera soluci\'on a la lista de soluciones soly */

   soly : [ sublis( [soluciones[1]], y) ],

   /* en este bucle sustituimos cada una de las soluciones encontradas a
       partir de
      la segunda y las vamos a\~nadiendo a la lista de soluciones soly
         */

   for i : 2 thru length(soluciones) step 1 do (

     soly : append( soly, [ sublis( [soluciones[i]], y ) ] )

   ), /* fin del bucle */
```

```
    soly

)$

/* fin */
```

Ejercicio 2

Con mucha frecuencia aparecen ecuaciones en las que no es posible realizar esta operación analíticamente, es ese caso no hay más remedio que resolver la ecuación numéricamente, lo cual sólo es posible una vez que hemos suministrado un valor numérico concreto para x. Escriba una función por medio del comando `newton` que proporcione y en función de x resolviendo la ecuación $F(x, y) = 0$ numéricamente.

En este caso no tenemos más remedio que renunciar a la solución exacta, y debemos conformarnos con obtener una solución aproximada. Esto implica que debemos añadir una parámetro que indice cuándo consideramos que la aproximación a la solución que tenemos es aceptable, este parámetro es el último argumento de la función `newton` del WXMAXIMA. En la función `solvenewton`, que hemos definido en el archivo `solve_Newton.mc`, hemos introducido este parámetro como una variable local (epsilon), no como un argumento de la función, ya que en principio este parámetro tendrá un valor fijo para todos los cálculos que hagamos (p. ej. el valor que hemos escogido es 10^{-6}, si tomamos un valor más pequeño la aproximación será algo mejor, pero el tiempo de cálculo será mayor). Listado del archivo `solve_Newton.mc`:

```
/* FUNCION "solvenewton"
    Resuelve num\'ericamente la ecuaci\'on F(x, y) = 0 para la variable y
    por medio del m\'etodo de Newton
    Input:
      F = la funci\'on que define la ecuaci\'on F = 0
      y = variable que queremos despejar
      y0 = punto de partida para el m\'etodo de Newton
    Output:
      valor de y encontrado para la soluci\'on de F = 0 */

solvenewton(F, y, y0) := block( [epsilon],

  /* cargamos el paquete newton1 */

  load (newton1),

  /* epsilon = tolerancia (suponemos que hemos encontrado la soluci\'on
      con un grado de aproximaci\'on aceptable cuando F(x, y) < epsilon)
         */
```

```
  epsilon : 10^(-6),

  newton(F, y, y0, epsilon)

)$

/* fin */
```

Una vez tenemos esto, dada una ecuación concreta $F(x, y) = 0$ para resolver este ejercicio basta con hacer

```
f(x) := solvenewton( F(x, y), y, y0)
```

donde el último argumento indica el punto de partida para el método de Newton.

Vamos a aplicar esta función para resolver la ecuación $e^y = xy$. Dependiendo del valor de x, esta ecuación puede tener una solución (si $x \in (-\infty, 0)$), ninguna (si $x \in (0, e)$), o dos soluciones (si $x > e$) para y; por otra parte, para que el método de Newton funcione razonablemente bien es necesario que el punto de partida y_0 esté razonablemente cerca de la solución buscada (de hecho, si la ecuación tiene más de una solución el método de Newton convergerá a una u otra solución dependiendo del valor de y_0). Antes de perder horas de cálculo sin estar seguros de qué estamos buscando, lo primero que tenemos que hacer es realizar un estudio de la ecuación para averiguar para qué valores de x existe la solución. Para ello podemos representar gráficamente las funciones e^y junto con xy en función de y, para varios valores de x, fijándonos en cuándo se cruzan; para esta ecuación esa estrategia permite comprender rápidamente cuándo tiene solución y cuándo no, alternativamente también se puede hacer la representación de $e^y - xy$ frente a y para varios valores de x, y ver cuándo corta al eje de abscisas. De este estudio deducimos por un lado que la solución de esta ecuación existe y es única para $x < 0$ y doble para $x > e$, mientras que no hay solución para $x \in (0, e)$, y por otro también nos permite hacernos una idea sobre el primer valor de y con que iniciar el método de Newton.

Para los valores negativos de x, y para la rama de abajo correspondiente a los valores de x por encima de e, el método de Newton converge bastante bien con $y_0 = 1/2$, de modo que definimos:

```
f(x) := solvenewton(exp(y)-x*y, y, 1/2);
data1 : makelist( [x, f(x)], x, -10, -0.1, 0.05)$
data2 : makelist( [x, f(x)], x, float(%e), 15, 0.05)$
```

Para calcular la rama de arriba correspondiente a $x > e$ es preciso dar un valor a y_0 que vaya aumentando a medida que x aumenta. Hemos visto que $y_0 = x$ funciona razonablemente bien, de modo que para calcular esta rama hacemos:

```
f(x) := solvenewton(exp(y)-x*y, y, x);
data3 : makelist( [x, f(x)], x, float(%e), 15, 0.05)$
```

En principio podríamos usar la función `f(x)` que hemos definido directamente en `plot2d` para generar las gráficas, pero eso resulta bastante lento. Para representar funciones definidas mediante cálculo numérico, como esta, es mucho más práctico calcular un conjunto discreto de puntos como hemos hecho arriba. Para representar estos puntos y ver finalmente el aspecto de la función $y = f(x)$ hacemos:

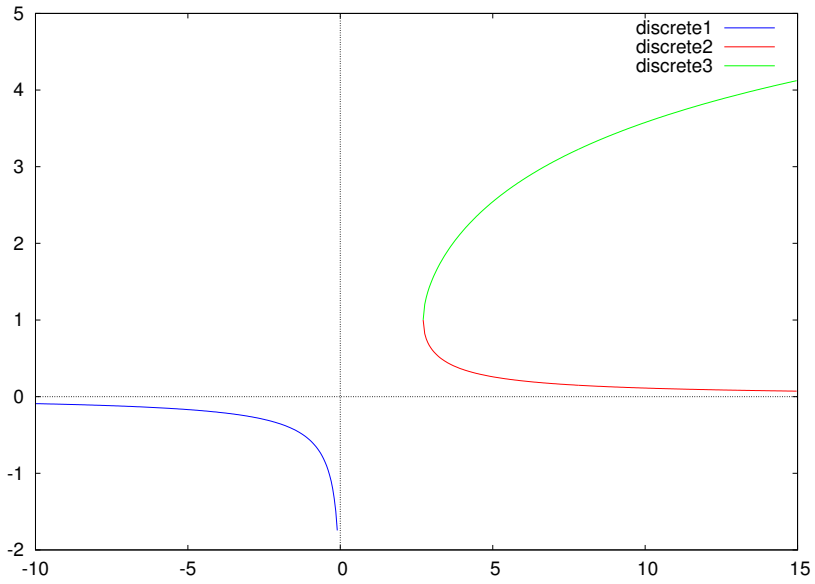

Figura 6.1:

```
plot2d( [ [discrete, data1], [discrete,data2],[discrete, data3] ] );
```

El resultado se muestra en la figura 6.1, donde se pueden ver las 3 ramas de la solución de $e^y = xy$.

Ejercicio 3

Utilice el programa de la sección anterior para generar una aproximación a la función $y = f(x)$ en forma de serie de Taylor (de orden n arbitrario), válida en un entorno de x_0.

Se sobre-entiende que la función $y = f(x)$ a la que se refiere el enunciado es la que se obtiene resolviendo una ecuación del tipo $F(x, y) = 0$, tal y como hemos considerado en los ejercicios precedentes. Por tanto, la forma de hacer el desarrollo de Taylor pedido depende de si la variable y se puede despejar o no en la ecuación $F(x, y) = 0$.

En el caso en que sí pueda despejarse este ejercicio es bastante sencillo, lo único que hay que hacer es aplicar la función `taylor` al output producido por `solvesust`

```
taylor( solvesust(F(x, y), y), x, x0, n)
```

Obsérvese que si hay más de una solución, la anterior instrucción aplica la función `taylor` sobre cada una de las soluciones y nos devuelve la correspondiente lista de desarrollos válidos en torno a x_0 hasta el orden indicado en `n`.

Realmente si es posible despejar de manera exacta y en función de x no tiene demasiado interés hacer el desarrollo, ya que conocemos la función exacta $y = f(x)$. El desarrollo en serie de Taylor tiene sentido especialmente cuando no hay forma

de despejar y en la ecuación $F(x, y) = 0$, ya que en ese caso nos proporciona una aproximación a $f(x)$, válida al menos en un cierto entorno alrededor del punto x_0 que estemos considerando. Para hacer el desarrollo en ese caso seguimos los pasos que se daban más arriba en este capítulo. La función `taylorimplicit` nos proporciona el desarrollo pedido, listado del archivo `taylor_implicit.mc`:

```
/*  FUNCION "taylorimplicit"
    Calcula el desarrollo en serie de Taylor, en torno a x0, a orden n, de
        la funci\'on
    y = f(x), definida de manera impl{\'\i}cita por la condici\'on F(x, y)
        = 0
    Input:
      1: F = la funci\'on de x e y que define la ecuaci\'on F(x, y) = 0
      2: x = variable independiente, respecto de la que desarrollamos
      3: y = variable dependiente, y = f(x)
      4: x0 = punto en torno al que desarrollamos
      5: y0 = punto de partida para resolver F(x0, y0) = 0 por el m\'etodo
          de Newton
      6: n = orden del desarrollo
    Output:
      desarrollo de f(x) en serie de potencias de (x - x0) hasta orden n
         */

taylorimplicit(F, x, y, x0, y0, n) := block( [epsilon, derivadas, auxf],

  /*  asignamos a epsilon la tolerancia con la que vamos a resolver F(x0,
     y0) = 0 */

  epsilon : 10^(-6),

  /*  cargamos el paquete newton1 y resolvemos F(x0, y0) = 0 */

  load(newton1),

  y0 : newton( subst(x0, x, F), y, y0, epsilon ),

  /*  para calcular los valores de las n primeras derivadas de y respecto
     a x en x0
     definimos primero la lista de derivadas que tenemos que calcular y
        la guardamos
     en la variable local derivadas */

  derivadas : makelist( diff(auxf(x), x, i), i, 1, n, 1),

  /*  derivando impl{\'\i}citamente n veces la ecuaci\'on F(x, auxf(x)) =
     0 encontramos
     el sistema de n ecuaciones que nos determina las n derivadas,
     planteamos este sistema, lo resolvemos con solve y sustituimos las
        soluciones en
```

```
      la variable derivadas */

  derivadas : subst(

        solve(
               makelist(
                      diff(
                             subst( auxf(x), y, F )
                      , x, i)
               , i, 1, n, 1)
        , derivadas)

  , derivadas),

  /*  a continuaci\'on tenemos que sustituir auxf(x) por y0, x por x0 */

  derivadas : subst( x0, x, subst(y0, auxf(x), derivadas) ),

  /*  finalmente el desarrollo se obtiene multiplicando cada una de estas
      derivadas por
      (1/i!) * (x - x0)^i, sumando todos estos resultados, y a\~nadiendo
         el primer
      t\'ermino del desarrollo y0, todo eso se puede hacer de una manera
         compacta por
      medio de */

  y0 + ( derivadas . makelist( (x - x0)^i / i!, i, 1, n, 1) )

)$

/*  fin  */
```

Por ejemplo, para usar esta función para calcular el desarrollo en serie de Taylor de y como función de x, definida implícitamente por $e^y = xy$, en torno al punto $x = 5$, a orden 3, podemos hacer

```
kill(all);
path : ".../Fisica-Computacional-1/Maxima/";
batchload(concat(path,"taylor_implicit.mc"));
aprox : taylorimplicit( exp(y) - x*y, x, y, 5, 1, 3);
```

Ejercicio 4

Visualice los resultados obtenidos mediante gráficas por medio de la función `plot2d`.

En el tema anterior ya indicamos cómo visualizar listas de funciones dadas por defi-

niciones explícitas. Más arriba en este mismo documento ya hemos indicado que para visualizar los resultados obtenidos con `solvenewton` lo más recomendable es calcular numéricamente cada rama de la solución por separado y posteriormente representarlas:

```
f(x) := solvenewton(exp(y)-x*y, y, 1/2);
data1 : makelist( [x, f(x)], x, -10, -0.1, 0.05)$
data2 : makelist( [x, f(x)], x, float(%e), 15, 0.05)$
f(x) := solvenewton(exp(y)-x*y, y, x);
data3 : makelist( [x, f(x)], x, float(%e), 15, 0.05)$
```

En el caso que estamos estudiando, para $x > e$ la función $f(x)$ es *bi-valuada*, tiene dos ramas, de forma que dependiendo del valor inicial y0 que empleemos en el método de Newton encontramos una u otra de estas ramas, tal y como hemos hecho más arriba. A la hora de calcular el desarrollo en serie de Taylor pasa lo mismo, dependiendo del valor que le demos a y0 encontraremos el desarrollo en serie de Taylor de una rama o de otra. En el siguiente código calculamos los desarrollos a orden 3 en torno a $x = -5$ (ver `aprox1`) y $x = +5$, en este segundo caso calculamos un desarrollo para la rama de abajo (ver `aprox2`) y otro para la de arriba (ver `aprox3`):

```
aprox1 : taylorimplicit( exp(y)-x*y, x, y, -5, 1/2, 3);
aprox2 : taylorimplicit( exp(y)-x*y, x, y, +5, 1/2, 3);
aprox3 : taylorimplicit( exp(y)-x*y, x, y, +5, 5, 3);
```

Para ver estas aproximaciones junto con la solución numérica hacemos:

```
plot2d( [ [discrete, data1], [discrete, data2], [discrete, data3],
       aprox1, aprox2, aprox3 ],[x, -10, 10],[y, -2, 5] );
```

Aquí es donde se ve la verdadera utilidad de los desarrollos en serie de Taylor, aunque es totalmente imposible despejar y de $e^y = xy$, por medio de estos desarrollos tenemos expresiones analíticas aproximadas muy precisas en un cierto entorno alrededor del punto donde desarrollamos.

Ejercicio 5

Dada la relación $y = f(x)$, aplique el programa anterior para obtener una aproximación en forma de serie de Taylor para la función $x = f^{-1}(y)$, válida en un entorno de y_0.

La ecuación $x = f^{-1}(y)$, para una determinada función conocida f, equivale a resolver para x la ecuación $F(x, y) = 0$, definida por la función $F(x, y) = y - f(x)$.

Por tanto este problema se reduce a un caso particular del problema que acabamos de resolver. De todas formas le recomendamos que practique aplicando estas funciones a diversas ecuaciones, por ejemplo puede aplicarlas a las ecuaciones que aparezcan en asignaturas como cálculo o física.

Ejercicio 6

Visualice los resultados obtenidos mediante gráficas por medio de la función `plot2d`.

Ya hemos indicado todas las instrucciones necesarias para visualizar los resultados.