Cálculo Numérico con Maxima

José Ramírez Labrador

Departamento de Matemáticas

Universidad de Cádiz

Año 2008

Esta es una versión preliminar, falta depurar la redacción y comprobar posibles erratas.

Para comentarios y sugerencias contactar con

pepe PUNTO ramirez ARROBA uca PUNTO es

Índice

Prólogo.

Este curso está orientado a:

- Interpretar y sacar conclusiones prácticas de algunos modelos matemáticos aplicados a las ciencias y a la ingeniería.
- Conocer algunas técnicas de análisis numérico y ser capaz de aplicarlas.
- Conocer y aplicar el lenguaje simbólico de propósito general Maxima.

Como prerrequisitos están un conocimiento de matemáticas a nivel de primero de carrera universitaria: derivadas, integrales, sistemas de ecuaciones y matrices. Buena parte de esto aparece en la asignatura virtual "Matemáticas de Nivelación" que se imparte en la Universidad de Cádiz.

Utilizaremos $e \sim 2.718281828459045...$ para indicar la base de los logaritmos neperianos; usaremos log o ln indistintamente para indicar los logaritmos neperianos y nunca los logaritmos en base 10. Las funciones trigonométricas siempre tendrán sus argumentos en radianes, nunca en grados.

Además se usará en algún momento el polinomio de Taylor que consiste en que, dada una función $f(x)$ suficientemente derivable en un entorno de x_0, se puede aproximar localmente por $f(x_0)+\frac{f'(x_0)}{1}(x-x_0)+\frac{f''(x_0)}{2}(x-x_0)^2++\frac{f'''(x_0)}{3*2}(x-x_0)^3+...+\frac{f^{(n)}(x_0)}{n!}(x-x_0)^n$; a éste polinomio se llama polinomio de Taylor de orden n de f en el punto x_0.

Por ejemplo, calculando las derivadas en $x=0$ hasta orden 4 se tiene que $e^x = exp(x)$ se aproxima cerca de $x=0$ por $1+\frac{x}{1}+\frac{x^2}{2}+\frac{x^3}{6}+\frac{x^4}{24}$.

Se puede probar que el error de la aproximación es $\frac{f^{(n+1)}(\xi)}{(n+1)!}(x-x_0)^{n+1}$ donde ξ es un punto intermedio entre x y x_0. Si un error es de la forma $C\,(x-x_0)^k$ con C una constante se suele abreviar diciendo que el error es de orden de $(x-x_0)^k$ o bien $O((x-x_0)^k)$. Podemos pues decir que el polinomio de Taylor de orden n aproxima a $f(x)$ en x_0 con un error $O((x-x_0)^{n+1})$.

Este manual está orientado a no matemáticos por lo que raramente daremos la deducción de los métodos numéricos que presentamos: el lector interesado puede consultar la bibliografía, por ejemplo [3],[2],[5],[11]. Un clásico sobre fórmulas y funciones matemáticas es [1]. [8] es interesante sobre problemas reales de computación. Diversos modelos matemáticos están en [6] y [4].

Queremos hacer constar que no pretendemos que los programas que se construyen estén optimizados en cuanto a velocidad ni utilicen todos los recursos de Maxima, sino más bien que sean simples, fáciles de entender e ilustren los aspectos más sencillos del cálculo numérico.

En cuanto a los ejercicios, los hay de aplicación del método expuesto, de obtención del resultado a través de instrucciones ya implementadas en Maxima y de programación en Maxima de forma que cada alumno o profesor pueda elegir los que más convengan a sus intereses. Hemos procurado que los ejercicios tengan siempre suficientes indicaciones para que sean realizables con facilidad.

CAPÍTULO 1

Maxima.

Índice del Tema

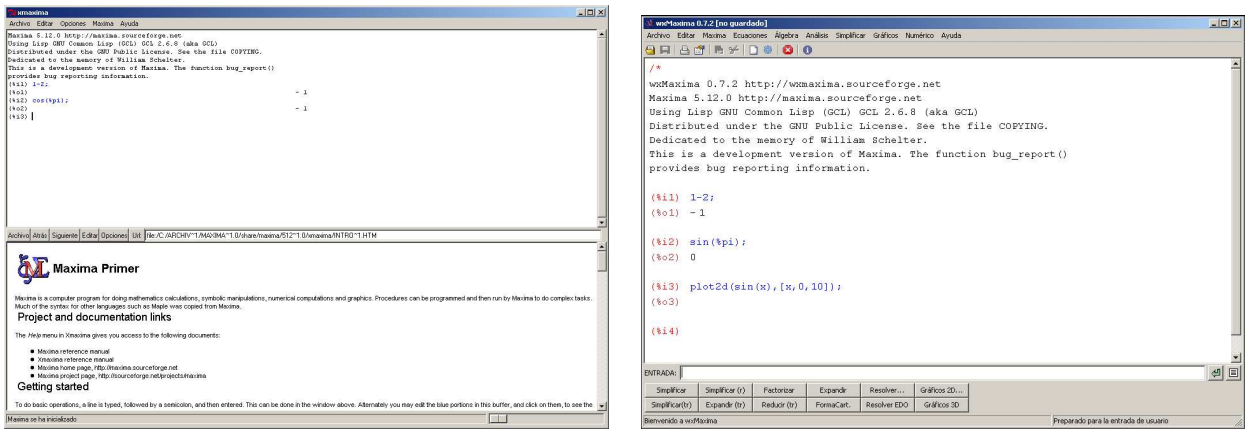

Figura 1.0.1: Pantallas de xMaxima y wxMaxima

Este capítulo tiene como objetivo familiarizarte con un programa que sustituye con ventaja a una calculadora y que permite manipular funciones, derivar, integrar de forma simbólica y mucho más; además es gratuito. Es decir, se puede calcular la integral de $x\cos(x)$ simplemente escribiendo integrate(x*cos(x),x); o bien factorizar el polinomio $x^3+3*x^2+3*x+1$ escribiendo factor(x^3+3*x^2+3*x+1); también puedes resolver $x^3+x^2+x+1=0$ escribiendo solve(x^3+x^2+x+1=0,x);

Los **Objetivos** de este tema son:

- Saber utilizar el programa Maxima como calculadora.
- Saber resolver ecuaciones, integrar y derivar con Maxima.
- Saber utilizar Maxima para dibujar funciones y conjuntos de puntos.
- Conocer las herramientas que Maxima tiene para multitud de aplicaciones matemáticas: determinantes, autovalores, solución de ecuaciones no lineales, soluciones de ecuaciones diferenciales...
- Entender programas sencillos (bucles y condicionales) hechos en Maxima.

Maxima es un programa de cálculo simbólico que puede descargarse de forma gratuita bajo licencia pública GNU de http://maxima.sourceforge.net; wxMaxima es un interface gráfico que incorpora gran cantidad de menús que facilitan la entrada de fórmulas, pero es mas rápido hacerlo a mano. Además completa los paréntesis (, los corchetes [, las llaves { y añade $ ó ; al final, lo que facilita las cosas. En caso de duda hay un menú de ayuda o puedes pulsar la tecla $F1$ (indicamos pulsar una tecla con el nombre de la tecla entre <> ejemplo $< F1 >$).

En la bibliografía indicamos donde puedes encontrar algunos manuales gratuitos [10],[7],[9] que te pueden servir de introducción o complemento a lo que sigue.

- para obtener $[,],\{,\}$, hay que mantener pulsada la tecla $< AltGr >$, para ^ se pulsa la tecla <^> y la tecla espacio.

- Las instrucciones de Maxima terminan con ; o con $. Si terminan con ; se presenta el resultado en la pantalla, si terminan con $ se realiza el cálculo pero no se muestra el resultado.
- Las operaciones aritméticas son + - * y /; * es obligatorio y no se puede sustituir por un espacio entre los factores, ejemplo 3*2+1; devuelve 7. 23+1; da error.
- Los paréntesis se usan normalmente, a^b se escribe $a\hat{}b$, o bien a**b; se usa . para indicar la coma decimal es decir 3.1416 es una aproximación de π, además el signo . se usa para multiplicar matrices.
- Las instrucciones se numeran con %i1, %i2, %i3,... y los resultados con %o1,%o2,... Para referirse a un resultado anterior puede escribirse el %o con el número correspondiente, el último resultado se puede indicar con %
- %pi representa $\pi = 3.141592....$, el número $e = 2.718281...$ se indica con %e, %i=$\sqrt{-1}$ es la unidad imaginaria.
- Para el argumento de funciones se usa () y las funciones incorporadas empiezan por minúsculas (como en matemáticas, pero en inglés) ejemplos: *sin*(x), *cos*(x), *sqrt*(4), *log*(%e), *exp*(x), *simplify*(2*sin(x)*cos(x)); ! indica el factorial, por ejemplo 100!
- Maxima intenta hacer los cálculos exactamente, por ejemplo, la instrucción 1/100+1/10000; devuelve 101/10000. Si hay numeros irracionales se dejan en forma simbólica, por ejemplo (1+sqrt(2))^3; devuelve $(1+\sqrt{2})^3$ si hacemos *expand*(%); devuelve 7+5$\sqrt{2}$. Para expresar el resultado de forma decimal se escribe *ev*(%,*numer*); que puede abreviarse como %,numer; alternativamente puede escribirse (1+sqrt(2))^3,numer; o bien *float*((1+sqrt(2))^3);
- Las instrucciones *numer* o *float* devuelven los resultados en notación punto flotante, que en esta version de Maxima ofrece 16 dígitos, pero los cálculos se pueden realizar con una precisión arbitraria usando la función *bfloat*, poniendo *fpprec* (que por defecto es 16) al valor que queramos, por ejemplo, haz

  float(sqrt(2)); bfloat(sqrt(2)); fpprec:50; bfloat(sqrt(2)); y observa que ahora se devuelven 50 dígitos de $\sqrt{2}$
- Para asignar algo a una variable se emplea el símbolo : y el signo = se reserva para las ecuaciones, por ejemplo xx:3; hace que la variable de nombre xx tome el valor 3. Maxima es sensible a la escritura mayúsculas-minúsculas. No es lo mismo unavariable:3; que Unavariable:3; ó que unaVariable:3; (son tres variables distintas). Una variable puede contener muchos tipos de datos, por ejemplo $y : 3(x+1)(x-1)(x+2)$; hace que la variable y tenga como valor el polinomio anterior. Ahora puedes calcular, por ejemplo, $y^2, 3y + 2z$ ó $cos(y + \pi)$.
- Para eliminar una variable se usa *kill*(nombre) por ejemplo kill(xx) elimina la variable xx de la lista de variables.
- = indica igualdad, por ejemplo, x(x+1)=0 es una ecuación con raíces 0 y -1.
- Para delimitar una lista se usa [ ], por ejemplo, $[x_0, y_0, z_0]$; los elementos se indican con [], por ejemplo, la orden li:[1,25,16]; guarda en la variable li una lista, de elementos 1, 25 y 16, el segundo elemento se indica con li[2].
- ' ' fuerza la evaluación, por ejemplo ' ' %i2 repite la evaluación de la orden %i2 , en cambio un único apóstrofo impide la evaluación, por ejemplo, hacer xx:2; seguido de xx; devuelve 2 en cambio, hacer 'xx; devuelve xx (aunque en la variable xx hemos guardado el valor 2).

- Para detener un cálculo se pulsa $< Control > G$ ó puedes usar el menu Maxima.

- Puedes cortar y pegar entradas anteriores y editarlas. Puedes dar varias instrucciones seguidas separadas por ; o $. En wxMaxima puedes usar $\uparrow\downarrow$ del teclado para recuperar instrucciones anteriores y existe al lado de la entrada un botón para abrir la entrada multilínea.

- La instrucción *showtime*:true; devuelve el tiempo que se tarda en realizar cada cálculo, si quieres dejar de verlo, haz showtime:false;

- Además del menu de ayuda puedes escribir ? y un espacio seguido de las primeras letras de la instrucción que quieres consultar, por ejemplo ?showtime, (observa que hay un espacio entre ? y la instrucción). Para describir completamente la instrucción escribe ??showtime o ??expand; También puedes usar describe(instrucción); por ejemplo, describe(factor);

- En Maxima los comentarios se escriben entre /* y */ . Todo lo que esté entre ellos es ignorado por Maxima. Es muy importante que los programas que se realicen estén debidamente documentados para que sean fáciles de entender y, en su caso, modificar. Con frecuencia escribiremos comentarios en los programas que presentamos como ejemplos para facilitar su comprensión o ilustrar qué hacen las variables etc.. Por ejemplo, si haces

  /* esto es ignorado por Maxima */1+2; devuelve 3.

Ejercicios:

- Calcula 2+2, 0.23+1/3, 1/2+1/3, 1/2.+1/3; calcula 30 dígitos de π. Suma los dos últimos resultados. Calcula 3^{100} y da el resultado con 10 dígitos.

- Calcula qué tan cerca de un entero está $e^{\pi\sqrt{163}}$. Mira si e^{π} es mayor que π^e. Calcula sum((-1)^n/n,n,1,5000); y sum(float((-1)^n/n),n,1,5000); y comprueba el tiempo que tardan los cálculos.

1 Cálculo y ecuaciones

- *expand*(expresión); desarrolla la expresión, por ejemplo, expand((x+1)^2); *factor*(expresión); factoriza la expresión, por ejemplo, factor(x^2+2*x+1); *ratsimp*(expresión) saca denominadores comunes, por ejemplo, ratsimp(1/(x+1)+1/(x-a)); *radcan* simplifica expresiones que pueden tener raices, logaritmos o exponenciales, *trigexpand* expande expresiones trigonométricas, *trigsimp* simplifica expresiones trigonométricas ...

- *ev*(expresion,cond1,cond2,..) evalúa la expresión sujeta a las condiciones cond1,cond2 ... por ejemplo, ev(-x^2+3*y,x=1,y=%pi) devuelve $3\pi - 1$, alguna de las condiciones pueden ser órdenes de Maxima como float que devuelve un resultado numérico, por ejemplo, ev(-x^2+3*y,x=1,y=%pi,float); simp, trigsimp, etc., por ejemplo, ev(cos(x)^2+sin(x)^2,trigsimp); etc. ev(expresion,cond1,cond2,cond3); puede escribirse (no en un programa) de forma abreviada como expresion,cond1,cond2,cond3; por ejemplo, cos(x)^2+sin(x)^2,trigsimp;

- *rhs*(ec) devuelve el lado derecho de una ecuación, *lhs* el lado izquierdo; *coeff*(pol,x,n) devuelve el coeficiente de x^n en el polinomio pol.

- *sum*(expresión,n,nini,nfin); calcula la suma de los valores de la expresión desde n=nini hasta nfin, por ejemplo, sum(n^2,n,1,10); calcula $1^2 + 2^2 + ... + 10^2 = 385$.

  product(expresión,n,nini,nfin); calcula el producto de los valores de la expresión desde n=nini hasta nfin, por ejemplo, product(n,n,1,10); calcula $1 * 2 * 3 * ... * 10 = 10!$

- *solve* se usa para resolver ecuaciones algebraicas (si están igualadas a 0 basta escribir en primer miembro), por ejemplo, solve(x^2+2*x+1,x); en caso de duda hay que indicar la variable a resolver, por ejemplo, solve(a*x^2+b*x+c,x); solve también resuelve sistemas de ecuaciones algebraicas haciendo solve([lista ecuaciones],[lista de incógnitas]); por ejemplo solve([a*x+b*y=c,x^2+y^2=1],[x,y]); observa que devuelve una lista con las soluciones; *allroots*(pol) devuelve las raíces de un polinomio numéricamente

- *find_root*(fun,x,a,b) busca soluciones de una función fun en el intervalo [a,b], la función debe tener signos distintos en a y b, por ejemplo, find_root(sin(x)-x/2,x,.2,3); el paquete *mnewton* que se carga con load(mnewton); permite resolver varias ecuaciones no lineales con mnewton(listafun, listavar,listaaprox), por ejemplo,

  mnewton([x1^2+x2^2-2,x1+3sin(x2)],[x1,x2],[1,1]); busca por el método de Newton una solución al sistema de ecuaciones $x1^2 + x2^2 - 2 = 0, x1 + 3 * sin(x2) = 0$ partiendo de $x1 = 1, x2 = 1$.

- Los números complejos se pueden escribir como a+%i*b y se operan normalmente, la parte real e imaginaria se indica con *realpart*, *imagpart*. Por ejemplo, puedes hacer z:1+%i; imagpart(z*(z-1)); para calcular la parte imaginaria de $(1 + i)(1 + i - 1)$. Para calcular las raíces cuartas de 1+i puedes hacer solve(z^4=1+%i);

- Podemos calcular derivadas, integrales, por ejemplo, si guardamos en una variable ff la función ff:x*sin(a*x), con *diff*(ff,x); Maxima calcula la derivada respecto x, diff(ff,x,2); calcula la derivada segunda, con *integrate*(ff,x); Maxima calcula la primitiva, con integrate(ff,x,0,1) la integral definida entre 0 y 1.

- *taylor*(fun,var,var_0,n) calcula el polinomio de Taylor de fun respecto de la variable var centrado en var_0 hasta orden n, por ejemplo, taylor(cos(x),x,0,5); *limit*(fun,var,var_0) calcula el límite de fun cuando var se acerca a var_0, por ejemplo, limit(sin(x)/x,x,0);

- el paquete *interpol* permite generar polinomios de interpolación de Lagrange y funciones ranura o splines

- *random*(n) devuelve un número seudoaleatorio entre 0 y n-1 si n es un número natural, por ejemplo, random(6)+1; simula la tirada de un dado, en cambio random(float(2)) devuelve un número aleatorio en $[0, 2)$. Si quieres que el generador de números aleatorios que usa random empiece en un lugar predeterminado puedes usar s1:make_random_state(valornumerico); set_random_state(s1); dependiendo del número que coloques en valornumerico el generador de random empezará en un sitio u otro, por ejemplo, haz create_list(random(6)+1,i,1,10); para simular 10 tiradas de dado, si repites verás que las tiradas del dado son distintas. En cambio si haces

  s1:make_random_state(3141592); set_random_state(s1);create_list(random(6)+1,i,1,10);

  la siguiente vez que hagas

  s1:make_random_state(3141592); set_random_state(s1);create_list(random(6)+1,i,1,10);

  te dará los mismos resultados del dado.

1.1.1 Ejercicios

- Haz que la variable z tome el valor (x+1)*(x-1), elévala al cuadrado, súmale 2x+2 y simplifica el resultado. Desasigna z.
- Factoriza el polinomio $x^{10}-1$.
- Calcula las tres primeras derivadas de sen(3x).
- Integra $x^2 sen^2(x)$ y comprueba el resultado derivando.
- Halla las raíces del polinomio x^3-7x^2+3ax.
- Resuelve el sistema $ax+by=0, x+y=c$ para las incógnitas x, y.
- Halla una solución numérica de $x=cos(x)$. Comprueba hasta que punto satisface la ecuación. (Observa que x-cos(x) vale -1 para $x=0$ y $\frac{\pi}{2}$ para $x=\frac{\pi}{2}$).
- Halla el area de un cuadrante del circulo unidad calculando la integral $\int_0^1 \sqrt{1-x^2}dx$.
- Calcula los primeros 5 términos del polinomio de Taylor de $cos(x)$ en $x=0$.

2 Matrices, listas y arrays

- Se pueden calcular determinantes, autovalores, etc de matrices (que pueden tener símbolos en sus elementos) con Maxima, para introducir sus elementos se usa *entermatrix*(n filas, n columnas) ejemplo A:entermatrix(2,2). Alternativamente A:*matrix* (a1,a2,..an) crea una matriz con las filas dadas por las listas a1,a2, etc. ejemplo A:matrix([1,2],[3,4]); . y no $*$ es la operación de multiplicar matrices, ejemplo a.b; *ident*(n) da la matriz identidad nxn, dada una matriz *transpose*(matriz) devuelve la matriz transpuesta, *determinant*(matriz) calcula el determinante, *invert* la inversa, *eigenvectors*(matriz) calcula los autovalores y autovectores.

- *triangularize* convierte en forma triangular superior una matriz cuadrada.

- El paquete *linearalgebra* que se carga con load(linearalgebra); permite calcular la factorización de Cholesky de una matriz, la descomposición LU, etc.

- *create_list*(expresion,i,iini,imax,j,jini,jmax) crea una lista con los valores de expresión con i variando de iini hasta imax, j variando de jini hasta jmax etc. por ejemplo create_list(x^i*y^j,i,0,2,j,0,3) devuelve $[1,y,y^2,y^3,x,xy,xy^2,xy^3,x^2,x^2y,x^2y^2,x^2y^3]$. *append* sirve para concatenar listas, *length*(lista) devuelve la longitud de una lista, *first*(lista) da el primer elemento, *rest*(lista,n) quita los primeros n elementos, lista[i] devuelve el elemento i-esimo, *delete*(elem,lista,n) quita de lista las primeras n apariciones de elem. *map*(función,lista) aplica la función a los elementos de la lista, por ejemplo map(cos,[0,1,%pi/2]); devuelve [1,cos(1),0].

- Un *array* corresponde matemáticamente a vectores, matrices o tensores y puede hacerse que sus datos sean del mismo tipo (float, etc). Los array se declaran con array(nombre,dimension) ejemplo array([punr,puni],31); declara dos arrays de 32 elementos de nombre punr,puni ya que los subíndices empiezan por 0. Las arrays se llenan por ejemplo con *fillarray*(punr,makelist(ev(sen(j/16*%pi),numer),j,0,31)); para que tenga los valores de $sen(x)$ en 32 puntos distribuidos entre [0,2π]. También se pueden asignar los valores punr[2]:17; para convertir un array en una lista se usa *listarray*.

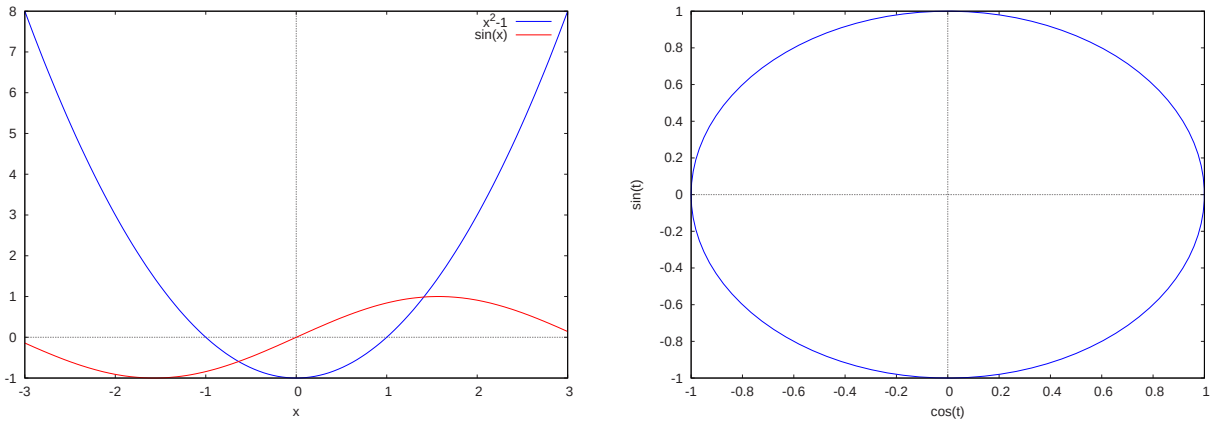

Figura 1.3.1: Dibujos de dos curvas explícitas y de una curva paramétrica

1.2.1 Ejercicios:

- Escribe una matriz cuadrada calcula su determinante y, si es posible, su inversa. Comprueba multiplicando que la matriz por su inversa es la matriz unidad. Calcula el polinomio característico a partir del determinante de $A - \lambda I$.
- Escribe como matrices un vector fila y uno columna. Comprueba los productos por A por un vector a la derecha y a la izquierda.
- Define una matriz (se llama de Hilbert) de orden n con $a_{ij} = \frac{1}{i+j-1}$; (en el paquete *linearalgebra* se puede generar con la instrucción *hilbert_matrix(n)*) halla los autovalores de A para $n = 5, 10, 15$ y comprueba que hay algunos muy pequeños.

3 Dibujos

- *plot2d* muestra un gráfico de dos o mas expresiones de una variable, por ejemplo

  plot2d([x^2-1,sin(x)],[x,-3,3]); dibuja las funciones $x^2 - 1, sen(x)$ para $x \in [-3, 3]$, (ver la primera de las figuras 1.3.1) hay muchas opciones por ejemplo

  plot2d([x^2+1,sin(x)],[x,-3,3],[y,-1,2]); restringe el valor de la función visualizado a [-1,2] (compara plot2d([sec(x)],[x,0,5]); y plot2d([sec(x)],[x,0,5],[y,-10,10]);)

  Podemos pintar curvas en paramétricas, por ejemplo

  plot2d([parametric,cos(t),sin(t),[t,-%pi,%pi]]); no hace bien la circunferencia porque usa pocos puntos, compara con

  plot2d([parametric,cos(t),sin(t),[t,-%pi,%pi],[nticks,100]]); que toma al menos 100 puntos en el dibujo (ver la segunda de las figuras 1.3.1). Se pueden pintar en un mismo dibujo curvas explícitas y curvas paramétricas

  plot2d([x^2-1,[parametric,cos(t),sin(t),[t,-%pi,%pi],[nticks,100]]],[x,-2,2]);

  Para pintar una lista de puntos la podemos crear a mano como pares de [x,y] por ejemplo

  lis:[[1,0],[2,3],[-1,5],[4,3],[3,-2]]; o crearla con make_list y luego dibujar los puntos aislados

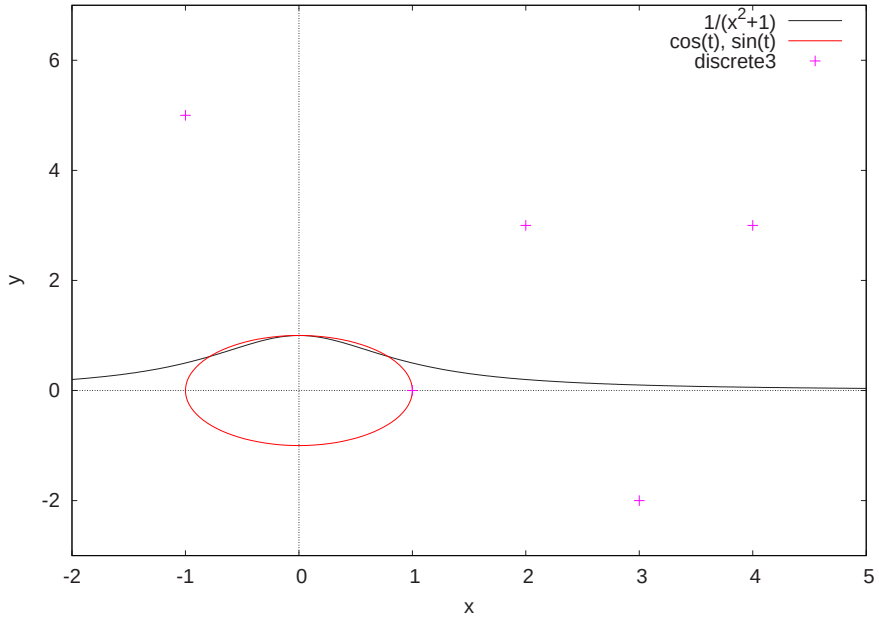

Figura 1.3.2: Dibujo de una curva explícita, una paramétrica y un conjunto de puntos

plot2d([discrete,lis], [style, [points,3,2]]); /* los pintamos con tamaño 3 y color 2*/ o bien unirlos por segmentos

plot2d([discrete,lis],[style, [lines,2,4]]);/* ahora el tamaño es 2 y el color 4*/ se pueden mezclar tipos de gráficas (ver la figura 1.3.2)

plot2d([1/(x^2+1),[parametric,cos(t),sin(t),[t,-%pi,%pi],[nticks,100]],[discrete,lis]],[x,-2,5],

[style, [lines,2,4],[lines,1,2],[points,4,3]],[y,-3,7]);

- *plot3d* permite dibujar superficies por ejemplo

  plot3d(1/(x^2+y^2+1),[x,-2,2],[y,-2,2]); si picas el dibujo con el ratón manteniendo pulsado el botón izquierdo al mover el ratón puedes mover el dibujo o girarlo para verlo desde otro punto de vista.

  Puedes dibujar superficies en paramétricas por ejemplo

  plot3d([x,y,x^2+y^2],[x,-3,3],[y,-3,3]); por ejemplo

  plot3d([cos(u)*(3+v*cos(u/2)),sin(u)*(3+v*cos(u/2)),v*sin(u/2)],[u,-%pi,%pi],[v,-1,1]); dibuja una banda de Moebius.

  Para pintar dos o mas superficies es conveniente usar el programa *draw* para eso lo cargamos con load(draw); y luego hacemos, por ejemplo

  draw3d(key = " Gauss", color = red, explicit(20*exp(-x^2-y^2)-10,x,-3,3,y,-3,3), yv_grid = 10, color = blue,

  key = "Paraboloide", explicit(x^2+y^2,x,-5,5,y,-5,5), surface_hide = true); o bien

  draw3d(color = turquoise,nticks=50, explicit(1/(.2+sin(y)^2+cos(x)^2),x,-3,3,y,-3,3), color = red,

  nticks=100, line_width = 2, parametric(cos(3*u),sin(3*u),u,u,0,6), title = "Superficie y curva"); para pintar una curva en paramétricas y una superficie (ver la figura 1.3.3).

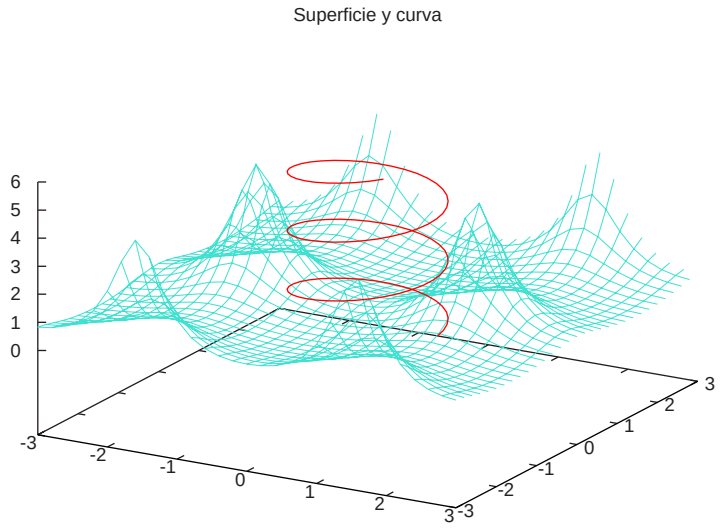

Figura 1.3.3: Dibujo de una curva paramétrica y una superficie

1.3.1 Ejercicios

- Pinta cos(x)-x y estima donde se hace cero.
- Pinta sen(x)/x desde -10 hasta 100.
- Dibuja simultáneamente e^x, x^e y estima donde se cortan las curvas.
- Dibuja $\frac{|x|}{x}$ desde -5 a 5; si no lo ves bien repite el dibujo con un rango vertical de [-2,2] y deduce que pasaba antes. Ayuda: prueba plot2d(abs(x)/x,[x,-5,5],[y,-2,2]); antes parte del dibujo estaba tapado por el borde y no se veía.
- Dibuja las curvas en paramétricas x(t)=cos(a t)cos(b t), y(t)=cos(a t)sen(b t) entre 0 y 2π para diversos valores de a,b; por ejemplo a=4, b=1; a=7, b=3; a=7, b=11.
- Dibuja la función exponencial, su tangente en $x = 0$ (es y=x+1) y el polinomio de taylor de orden 2 y 3 en $x = 0$ (son $y = 1 + x + \frac{x^2}{2}, y = 1 + x + \frac{x^2}{2} + \frac{x^3}{6}$) en una misma gráfica.
- Dibuja la superficie $\frac{1}{1+x^2+y^2}$.
- Dibuja un trozo de esfera $z = \sqrt{1 - x^2 - y^2}$; de paraboloide elíptico $z = 1 + x^2 + y^2$ y de paraboloide hiperbólico $z = x^2 - y^2$.

4 Ecuaciones diferenciales

- Para indicar que una variable depende de otra, a efectos de ecuaciones diferenciales se usa *depends*(y,x); dada la dependencia podemos escribir ecuaciones diferenciales ec:diff(y,x,2)+4*y; e intentar resolverlas (hasta ecuaciones de segundo orden) con *ode2*(ec,y,x);. Una forma alternativa de resolver ecuaciones diferenciales sin definir las dependencias es dejar las derivadas sin evaluar utilizando ' por ejemplo, para resolver $y'' - y = 0$ podemos escribir ode2('diff(y,x,2)-y,y,x);.

Veamos cómo comprobar si una función es solución de una ecuación diferencial o ecuación en derivadas parciales por ejemplo para comprobar si $u = cos(x + b\,t)$ es solución de la ecuación de ondas $\frac{\partial^2 u}{\partial t^2} - c^2 \frac{\partial^2 u}{\partial x^2} = 0$ puedes hacer

depends(u,[x,t]); edp:diff(u,t,2)-c^2*diff(u,x,2); ev(edp,u=cos(x+b*t),diff,expand);

y se obtiene que ha de ser $c = \pm a$ para que sea solución.

En el paquete *dynamics* de Maxima se incorpora la función *rk* que resuelve numéricamente una ecuación diferencial (o un sistema de ecuaciones diferenciales) de primer orden por el método de Runge-Kutta de orden 4. Si la ecuación diferencial está escrita en la forma $y' = f(y,t)$ con la condición inicial $y(t_0) = y_0$ para calcular los valores de w_i desde t_0 hasta $t = b$ con paso h se escribe $load(dynamics)$; $rk(f(y,t), y, y_0, [t, t_0, b, h])$; y Maxima devuelve una lista de los puntos $[t_i, w_i]$. Por ejemplo para resolver numéricamente $y' = cos(y) + t + 1$ con $y(1) = 2$ desde $t = 1$ hasta $t = 4$ con $h = .05$ se escribe $rk(cos(y) + t + 1, y, 2, [t, 1, 4, .05])$;

Para resolver numéricamente sistemas de ecuaciones diferenciales de primer orden $y' = f1(t,y,z)$, $z' = f2(t,y,z)$ con condiciones iniciales $y(t_0) = y_0, z(t_0) = z_0$ desde t_0 hasta $t = b$ con paso h se escribe $rk([f1(t,y,z), f2(t,y,z)], [y,z], [y_0, z_0], [t, t_0, b, h])$; y devuelve una lista de valores $[t_i, y_i, z_i]$. Ejemplo, para resolver $y' = y * z + t, z' = y + z - t$ con $y(0) = 1, z(0) = 2$ desde $t = 0$ hasta $t = 3$ con paso $h = 0.1$ se escribe $rk([y * z + t, y + z - t], [y,z], [1,2], [t, 0, 3, .1])$;

5 Funciones y programas

Todo programa tiene una introducción de datos, un cálculo intermedio y una salida de los resultados. Los datos pueden incorporarse al programa o ya estar en alguna variable definida antes en Maxima.

La forma más fácil de escribir programas dentro Maxima es a través de funciones o bloques, por ejemplo suma(x,y):=x+y; crea una función que devuelve la suma de los argumentos, para usarla puedes hacer suma(1,2); que devuelve 3.

- Para definir funciones se utiliza := por ejemplo

  ff(x,y):=sqrt(x^2+y^2); hace que ff(1,-1) sea $\sqrt{2}$ si queremos que el cuerpo de la función sea una sucesión de expresiones se hace f(x):=(expr1,expr2,...,exprn); y se devuelve la última. Es una buena costumbre, al empezar a programar, que antes de definir una función de nombre (el que sea), la borres con *kill*(nombre) (el que sea) así evitas que haya definiciones anteriores de la función, con dispfun(all); puedes ver las definiciones de las funciones que has hecho.

Es muy importante que los programas estén documentados con comentarios situados entre /* */ para que se comprenda lo que hacen y cómo lo hacen. Por supuesto los comentarios no afectan a lo que hace el programa sino le facilitan la vida al usuario o al programador.

En un programa $block$ se pueden utilizar variables locales que no interfieren con el resto de Maxima, esto es importante para no machacar los valores externos si la variable se llama igual que otra definida fuera.

Los programas de Maxima devuelven usualmente la última instrucción realizada, por lo que si alguna vez vamos a usar la salida de un programa como entrada de otro (o del mismo) y se necesitan varios datos de entrada es posible que tengamos que poner la entrada y la salida del programa como una lista de valores.

Las estructuras básicas de programación son:

Los condicionales, si una condición es cierta se realiza algo y si no es cierta se hace otra cosa (o nada). Por ejemplo para calcular el valor absoluto de un número, si el número es menor que cero se devuelve el número con el signo cambiado y si no el número como estaba.

- *if* cond *then* expresión1 *else* expresión2 es la forma de los condicionales, si la condición es cierta se realiza la expresión1, si no lo es la expresión2. Por ejemplo

  absoluto(x):=if x>0 then x else -x;

  devuelve $|x|$. La parte *else* expresión2 es optativa. Los condicionales pueden ponerse dentro de otros condicionales, los relaciones son $>, <, <=, and, or, not, equal...$

Otra estructura básica de programación son los bucles, consisten en repetir un mismo proceso un número fijo de veces, o aplicarlo mientras se cumple una condición o bien aplicarlo a todos los elementos de una lista. Intuitivamente escribir la tabla de multiplicar del 5 equivale a multiplicar por 5 los números $1, 2, 3, ...10$ así que podemos hacer que una variable (pongamos que se llama i) tome los valores del 1 al 10 de uno en uno.

- En Maxima hay varias formas de bucles, entre ellas:

  for var:varini *thru* vfinal *step* inc *do* cuerpo; si inc vale 1 puede eliminarse; en cuerpo pueden ponerse varias instrucciones de Maxima separadas por , y metidas en un paréntesis. Por ejemplo

  for i:1 thru 10 step 2 do print("i vale ",i); escribe i vale 1, i vale 3, ... hasta i vale 9

  for j:1 thru 10 do print("5 por ",j," vale ",5*j); escribe la tabla de multiplicar del 5.

  for j:1 thru 10 do (print("5 por ",j," vale "),print(5*j)); hace casi lo mismo (el producto lo escribe en otra línea) con dos instrucciones print.

  suma:0; for i:1 thru 100 do suma:suma+i; acumula en la variable suma el valor de 1+2+3+....100

  for var *in* lista *do* cuerpo

  por ejemplo, for i in [1,2,3,4] do print("el cuadrado de ", i, " es ",i^2); escribe los elementos de la lista y sus cuadrados.

  while cond *do* cuerpo se ejecuta el cuerpo mientras que la condición sea cierta, por ejemplo,

  suma:0;i:1;while i<10 do (suma:suma+i,i:i+1);

  guarda en la variable suma 1+2+...+9=45, observa que no escribe el resultado y que tenemos que cambiar el valor de i dentro del cuerpo, si no podemos entrar en un bucle que no termine nunca.

- Es útil usar *block* para crear subrutinas, de esta forma se pueden usar variables locales que no interfieren con las variables que usemos fuera ejemplo f(x):=block([lista de variables locales, si es preciso asignadas],cuerpo); por ejemplo,

  media(x,y):=block([xm:(x+y)/2],xm); devuelve la media aritmética de x,y usando una variable local que llamamos xm. También podíamos hacer media(x,y):=(x+y)/2;

- Maxima evalúa una o dos veces las sentencias (busca ev en la ayuda o en el manual para más precisión), en cambio otros programas de Cálculo simbólico (Maple,Mathematica) evalúan las sentencias hasta que no hay cambios, esto hace que los procedimientos no sean idénticos. Prueba a escribir

  kill(all);

  /* Para eliminar todas las asignaciones */

a:b; b:c; c:d; d:1;

/*si haces ahora c; veras que tiene el valor d pero no lo que d vale que es 1, ev(b); te da d es decir ev evalua un nivel más hacia dentro, si quieres evaluar un nivel más haz ev(b,ev); que te dará d, finalmente ev(a,infeval); o equivalentemente a,infeval; evalúa hasta que no hay cambios es decir hasta el valor 1, por supuesto ev(x,x=x+1,infeval) genera un bucle sin fin, compruébalo */

1.5.1 Ejercicios:

- Define una función que valga 1 para $x < -1$ y x^2 para $-1 \leq x < 2$, dibújala en [-5,5]. (Ayuda: usa un condicional, por ejemplo if $x < 1$ then 1 else x^2)
- Define una función que valga 1 para $x < -1$, x^2 para $-1 \leq x < 2$ y $x - 2$ para $x \geq 2$, dibújala. (Ayuda: usa dos condicionales, por ejemplo if $x < 1$ then 1 else (if $x < 2$ then x^2 else $x - 2$))
- Define una función que calcule el factorial de un número a partir de un bucle
  (Ayuda, 10!=1*2*3*..*9*10, haz que prod valga 1 y acumula en prod el producto 1*2*...*9*10).
- Una aproximación a la derivada de f en x esta dada por $\frac{f(x+h)-f(x-h)}{2h}$ con h pequeño. Define una función que tenga como entradas x, h y devuelva la aproximación de la derivada de la función exponencial con esos valores de x, h; elige $x = 1$ y $h = 1., 10.^{-3}, 10.^{-6}, 10.^{12}, 10.^{-18}, 10.^{-24}$ e interpreta los resultados.

CAPÍTULO 2

Fuentes y propagación de errores.

Índice del Tema

Los métodos numéricos combinan dos herramientas fundamentales para el ingeniero y el científico: Matemáticas y Ordenador. En este tema nos centramos en la segunda parte: el Ordenador. Lo primero que hacemos es conocer el lenguaje con el que nos comunicamos con él, después pasamos a ver como se almacenan los números en el mismo y, como consecuencia de este almacenamiento, los errores que cometemos.

Los **Objetivos** de este tema son los siguientes:

- Repasar el sistema binario.
- Conocer la notación científica normalizada.
- Saber cómo se almacenan los números en un ordenador.
- Comprender las consecuencias de las limitaciones de la capacidad de un ordenador.
- Entender los diversos tipos de errores y su influencia.
- Comperder el significado de los términos algoritmo, estabilidad y condición.
- Comprender la importancia de la estabilidad de un algoritmo.

1 Notación en punto fijo y en punto flotante

A fin de analizar en detalle el error por redondeo es necesario comprender cómo se representan las cantidades numéricas en el ordenador.

Basándose en el hecho de que la unidad lógica básica de los ordenadores usa componentes electrónicos con dos posibilidades apagado/encendido, los números en el ordenador se representan en el sistema binario. De igual modo que en base 10 el número 4123 equivale a 4*1000+1*100+2*10+3*1, en base 2 el número 1010 equivale a 1*8+0*4+1*2+0=10.

Para representar números enteros en el ordenador se emplea la notación en punto fijo. Para ello el programa fija el número de bits que se van a dedicar a representar los números (por ejemplo 16 ó 32) Una vez fijado el número de bits es sencillo representar enteros en la computadora. Para ello, se puede emplear un bit para indicar el signo, por ejemplo un 0 para positivo y un 1 para el negativo y usar los bits sobrantes para guardar el número escrito en base dos. Dependiendo del número de bits que dediquemos podremos representar exactamente números mas o menos grandes, por ejemplo, si dedicamos 8 bits, quitando uno que corresponde al signo, podemos llegar hasta $1*2^6+1*2^5+1*2^4+1*2^3+1*2^2+1*2+1*2^0=127$. Estos números se representan exactamente y dependiendo de los bits que dediquemos habrá mas o menos de ellos pero al sumar o multiplicar dos números podemos "salirnos" de los números que se pueden representar, esto genera un error de desbordamiento o "overflow".

Veamos la representación gráfica de algunos números en punto fijo; por ejemplo, los números enteros desde -10 a 10, que están igualmente separados como vemos el primero de los gráficos 2.1.1. /* guardamos en una variable llamada puntos una lista con las coordenadas de los numeros del -10 al 10 en la x y 0 en la coordenada y */

```
puntos:create_list([i,0],i,-10,10);
```

/* los pintamos con tamaño 3 y color 2*/

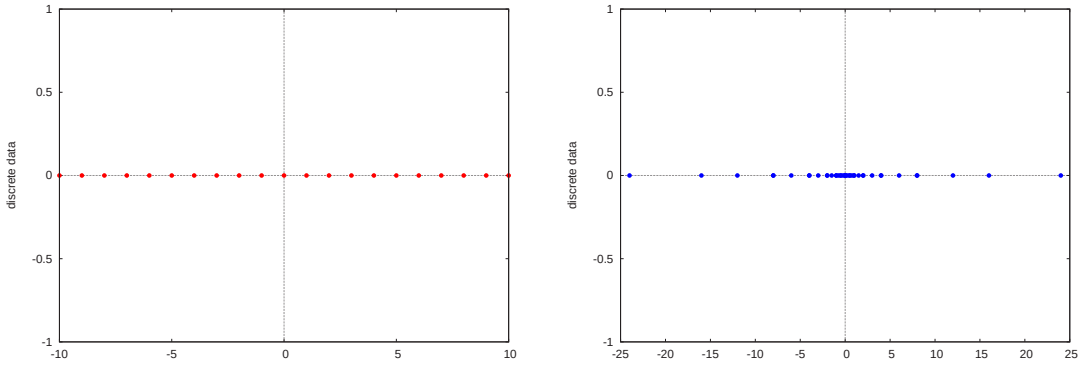

Figura 2.1.1: Números en punto fijo y punto flotante

plot2d([discrete,puntos], [style, [points,3,2]]);

Los demás números se representan en el ordenador usando la notación científica o *punto flotante*. En esta representación el programa fija el numero de bits que se destinan a la parte decimal con su signo (la mantisa) y el número de bits que se destinan al exponente (también con su signo). En casi todos los casos, los números se almacenan como cantidades de punto flotante. Para ello si el número es distinto de cero desplazamos el punto decimal (suministrando las potencias de 2 necesarias) de forma que todos los dígitos queden a la derecha del punto decimal y además el primer dígito sea distinto de cero. El ordenador representa los números de punto flotante en la forma

$$\pm 0.d_1 d_2 d_3 \cdots d_p * B^e$$

donde los d_i son dígitos o bits con valores desde cero hasta $B-1$ y

B= número de la base que se utiliza, normalmente, 2.

p= número de bits (dígitos) significativos.

e= exponente entero.

Los números 12.358, 0.013, -4.123 y 0.008005 se guardarían en notación en punto flotante (suponiendo que la base es 10) de la forma: 0.1238 $\times 10^3$, 0.13$\times 10^{-1}$, -0.4123 $\times 10^1$ y 0.8005 $\times 10^{-2}$

Dependiendo del numero de bits que dediquemos a la mantisa y al exponente podremos representar mas o menos numeros mas o menos grandes o pequeños pero, igual que antes, solo se representarán exactamente algunos (pocos) números.

Observaremos cómo los números en punto flotante están más agrupados en torno al origen. Para ello consideraremos un caso simple: los números flotantes de la forma $\pm 0.d_1 d_2 \times 2^e$ donde d_i representa un dígito en el sistema binario, $d_1 \neq 0$ y e es el exponente tal que $-2 \leq e \leq 3$; éstos son: $\pm 0.10_2 \times 2^e$ y $\pm 0.11_2 \times 2^e$. Expresamos estos números en el sistema decimal y, en el segundo de los gráficos 2.1.1, se puede observar cómo se concentran alrededor del cero.

/* guardamos en una variable llamada puntosf una lista con las coordenadas de los numeros de la forma i*2^j con i variando de -3 a 3 y j variando de -2 a 3 en la x y 0 en la coordenada y */

puntosf:create_list([i*2^j,0],i,-3,3, j,-2,3);

```
/* los pintamos con tamaño 3, color 1 */
plot2d([discrete,puntosf], [style, [points,1,2]]);
```

2 Operaciones matemáticas en los ordenadores, redondeo

Para que las operaciones numéricas entre números en punto flotante se realicen con rapidez en el ordenador hay circuitos dedicados a sumar y multiplicar, para utilizarlos está definido el número de bits dedicados a la mantisa y al exponente, cómo se interpreta el signo y cómo se representa el cero. En un ordenador los números en punto flotante se representan mediante la notación científica normalizada en sistema binario,

$$x = \pm m \times 2^e, \quad \frac{1}{2} \leq m < 1$$

con $x \neq 0$ y e entero.

Suponemos que el ordenador para representar un número usa 32 bits, (las cosas suelen ser más sofisticadas) distribuidos de la siguiente forma: 1 bit: signo del número real, 8 bits: exponente, 23 bits: mantisa.

La mantisa mas pequeña será $1 * 2^{-1} = 0,5$ y la mas grande $1 * 2^{-1} + 1 * 2^{-2} + ... + 1 * 2^{-23} = 1 - 2^{-23} = \frac{8388607}{8388608} \sim 0.9999998807907$ que es casi 1. Como $2^{23} = 10^{23 * log_{10}(2)} \simeq 10^{6.92369}$ esto equivale a cerca de 7 dígitos en base 10 para representar el número.

De igual modo, si dedicamos uno de los bits del exponente al signo, el exponente variará entre ± 127 así que los numeros positivos variarán entre $0.5 * 2^{-127} \simeq 2.93874 * 10^{-39}$ y $\simeq 1 * 2^{127} \simeq 1.7014 * 10^{38}$. Por tanto no podremos representar números mayores que estos.

En realidad Maxima permite guardar numeros con exponente 10^{308} y 17 dígitos en base 10 de mantisa en simple precision (float).

Fijado el tamaño que dedicamos a guardar los números si el número que queremos considerar es demasiado grande se produce un desbordamiento por exceso y el ordenador no puede seguir trabajando. Si el número es demasiado pequeño se produce un desbordamiento por defecto: el ordenador toma el valor como cero.

Cuando realizamos operaciones se pueden sobrepasar los límites de almacenamiento del exponente por ejemplo multiplicando 10^{200} por 10^{300} dará 10^{500} o de la mantisa, por ejemplo si multiplicamos dos números de 9 dígitos, el resultado puede tener 18 dígitos o si dividimos $1/3 = 0,333333....$ el resultado tiene infinitos dígitos. Por otra parte existen números como π ó $\sqrt{2}$ que necesitan infinitos dígitos para representarse exactamente. Usualmente se calculan las operaciones con más dígitos de los que se pueden guardar y luego

- Se cortan los dígitos que sobran. Por ejemplo si sólo guardamos 4 dígitos decimales y tenemos que representar 0,12377 guardaríamos 0,1237 y si tenemos que guardar 0,333333 guardamos 0,3333.
- Se elige el número mas cercano que se representa exactamente en el ordenador. A esto se le llama *redondeo*. Por ejemplo si sólo guardamos 4 dígitos decimales y tenemos que representar 0,12377 guardaríamos 0,1238 y si tenemos que guardar 0,333333 guardamos 0,3333. Si consideramos el número $\pi = 3,14159265\ldots = 0,314159265\ldots \times 10$ y suponemos que sólo queremos guardar 5 dígitos nos quedará $0,31416 \times 10$, para guardar 4 dígitos tomaremos $0,3142 \times 10$ y si guardamos 3 dígitos tomamos $0,314 \times 10$.

  En general, si tenemos el número $0.d_1 d_2 \cdots d_k d_{k+1} \ldots \times 10^e$ y queremos quedarnos con k dígitos, entonces:

Si $d_{k+1} \geq 5$: sumamos 1 a d_k. Si $d_{k+1} < 5$: cortamos.

En Maxima la función floor(x) devuelve el mayor entero menor o igual que x, la función de Maxima ?round(x) redondea x al entero mas cercano. El error que se comete al redondear es mas pequeño.

En cualquier caso casi siempre se cometerán errores al realizar operaciones con el ordenador porque, en general, el resultado no representará exactamente.

Los programas de manipulación simbólica como Maxima, Mathematica o Maple pueden representar de forma exacta los numeros enteros y racionales a cambio de utilizar más memoria en el ordenador y realizar los cálculos más lentamente.

Además podemos definir con fpprec el número de dígitos (por defecto 16) que queremos utilizar en los números de punto flotante grandes (bfloat). De igual modo se utiliza más memoria en el ordenador y los cálculos tardan más.

Compara los resultados de las siguientes instrucciones de Maxima

```
/* guardamos en la variable x el valor en punto flotante de 123456789123456*10^-15 */

x:float(123456789123456*10^-15);

/* calculamos x^2 */

x^2;

/* reservamos 50 digitos para los números en punto flotante grandes */

fpprec:50;

/* guardamos en la variable xg el valor en punto flotante grande de 123456789123456*10^-15 */

xg:bfloat(123456789123456*10^-15);

/* calculamos xg^2 */

xg^2;
```

También puedes usar float y bfloat para calcular resultados, recuerda que π se escribe en Maxima %pi

Compara el resultado en Maxima de las instrucciones float(%pi); y bfloat(%pi);

3 Error absoluto y relativo, propagación de errores

Un número aproximado a es un número tal que difiere ligeramente de un número exacto A y se utiliza en los cálculos en lugar de este último. Si $a < A$, se dice que a es una aproximación por defecto. Si $a > A$ se dice que la aproximación es por exceso.

Sea p un número cualquiera y sea p^* una aproximación del mismo, a la diferencia $p - p^*$ denominamos **error** y lo denotamos por la letra e,

$$e = p - p^*.$$

En muchos casos, el signo del error es desconocido por lo que resulta aconsejable utilizar el *error absoluto de un número aproximado*.

Si p es un número cualquiera y p^* una aproximación del mismo se denomina **error absoluto** a $\mid p - p^* \mid$.

Pero la definición de error absoluto tiene un pequeño defecto, como demostraremos al considerar los valores verdaderos de la longitud de un puente y la de un lápiz (por ejemplo 10000 y 10 cm, respectivamente) y aproximados (por ejemplo 19 999 y 9 cm), ya que aunque el error absoluto es de 1cm en ambos casos, tiene mayor trascendencia el error en la magnitud del lápiz. Por tanto, necesitamos comparar el error absoluto con el valor verdadero y definiremos el concepto de *error relativo*.

Si p es un número cualquiera y p^* una aproximación del mismo se denomina **error relativo** a $\dfrac{\mid p - p^* \mid}{\mid p \mid}$ si $p \neq 0$.

El error relativo en la longitud del puente anterior del ejemplo anterior es de $\frac{1}{10000} = 0.0001$ y el error relativo en la longitud del lápiz es $\frac{1}{10} = 0.1$. El error absoluto en ambos casos es el mismo, pero es evidente que la aproximación en la longitud del puente es mejor.

En situaciones reales se suele desconocer el valor exacto de las magnitudes. Por ello es conveniente utilizar *cotas del error absoluto* y *cotas del error relativo*

Propagación de errores. Cuando realizamos una operación matemática los errores de los datos afectan al resultado. Se puede probar que el error absoluto de la suma o resta de dos números es aproximadamente la suma de los errores absolutos y que el error relativo del producto de dos números es del orden de la suma de los errores relativos de los factores, es decir que en general los errores se propagan de forma controlada. Sin embargo si restamos dos números muy cercanos el error relativo puede crecer mucho porque los dígitos que están representados en el ordenador se cancelan. Por ejemplo si $\pi = 3.141592...$ esta representada como 3.1416 y $\frac{22}{7} = 3.14285714...$ se representa como 3.1429 los errores relativos son $2.338..10^{-6}$ y $1.3636..10^{-5}$ bastante aceptables, sin embargo si restamos tenemos que la representación en el ordenador de $\frac{22}{7} - \pi$ es 0.0013, el error absoluto es del orden de $3.551..10^{-5}$ pero el error relativo es del orden de 0.028 porque $\frac{22}{7} - \pi = 0.0012644..$ Este error relativo es muy grande. Del mismo modo que *restar números cercanos, multiplicar por números grandes (o dividir por números pequeños) afecta mucho a los errores* ya que en este último caso aumenta el error absoluto.

4 Algoritmos, condición y estabilidad

Un **algoritmo** es una secuencia finita de operaciones descrita de una forma precisa. Por ejemplo cuando te enseñaron en la escuela a sumar o multiplicar números te explicaron un algoritmo para hacerlo.

Por ejemplo, para calcular en el punto x_0 el valor de un polinomio $p(x) = a_0 + a_1x + a_2x^2 + ... + a_{n-1}x^{n-1} + a_nx^n$ puedes seguir dos algoritmos:

- Sustituye la x por x_0 y realiza las operaciones. Es lo que estás acostumbrado a hacer, por ejemplo con una calculadora de bolsillo, escribes a_0 y lo introduces en la memoria, escribes x_0 lo multiplicas por a_1 y lo sumas a la memoria, escribes x_0, lo elevas al cuadrado, lo multiplicas por a_2 y lo sumas a la memoria ... hasta que escribes x_0, lo elevas a n, lo multiplicas por a_n, lo sumas a la memoria y recuperas el valor que hay en la memoria. Este es el valor buscado.

- Escribes el polinomio *forma anidada* es decir como $p(x) = a_0 + x(a_1 + x(a_2 + x(a_3 + x(...(a_{n-1} + xa_n)))))$ y realizas las operaciones empezando por el paréntesis más interior. En una calculadora de bolsillo, escribes a_n y lo multiplicas por x_0, le sumas a_{n-1} lo multiplicas por x_0, le sumas a_{n-2} lo

multiplicas por x_0,..... hasta que sumas a_1 lo multiplicas por x_0 y sumas a_0. Este es el valor buscado. Por ejemplo $7x^3-4x^2+2x+3$ se escribe como $((7x-4)x+2)x+3$. Para calcularlo se hace $7*x_0$, se le suma -4, se multiplica por x_0, se le suma 2, se multiplica por x_0, y se suma 3 y se acabó. De esta forma se realizan 3 sumas y 3 productos, en cambio para calcular $7x^3-4x^2+2x+3$ de la forma tradicional se necesitan 6 productos (3 para $7x^3$, 2 para $-4x^2$ y 1 para $2x$) y 3 sumas. Así se tarda menos tiempo y se minimiza el error de redondeo. La forma anidada es más eficaz para calcular numéricamente el valor de un polinomio. En Maxima la instrucción *horner(pol)* devuelve la forma anidada del polinomio pol.

Algunos procesos matemáticos no pueden calcularse exactamente con un número finito de pasos y se usan algoritmos para acercarse a la solución. Por ejemplo sumar una serie infinita, calcular un límite, una derivada o una integral en general no puede hacerse con un número finito de operaciones.

Si queremos sumar una serie nos interesa dar un algoritmo que se aproxime tanto como queramos a la suma de la serie. Por ejemplo si queremos sumar $\sum_{i=1}^{\infty} a_i = a_1+a_2+a_3+.....$ (*aqui hay infinitos sumandos*)... un algoritmo para aproximarse a la solución sería calcular sólo un número finito de sumandos, por ejemplo 100. Esto puede hacerse en una calculadora de la forma siguiente: escribes a_1, lo metes en la memoria, escribes a_2, lo sumas a la memoria, escribes a_3, lo sumas a la memoria ... hasta que escribes a_{100}, lo sumas a la memoria y recuperas el valor de la memoria. Este es el resultado del algoritmo. Dependiendo de la serie concreta las sumas parciales se acercarán mas o menos rápidamente a la suma de la serie.

La finalidad de los algoritmos es dar un procedimiento para resolver un problema o para aproximar una posible solución.

Algunos *algoritmos son iterativos* es decir, las aproximaciones a la solución se obtienen aplicando reiteradas veces una función en la forma $x_{n+1}=f(x_n)$ y la sucesión $x_0, x_1, x_2, x_3, x_4, ...$ converge a la solución. Por ejemplo para calcular $\sqrt{2}$ tomamos la función $f(x)=\frac{1}{x}+\frac{x}{2}$, elegimos un valor inicial $x_0 \neq 0$, calculamos $x_1=f(x_0)$, calculamos $x_2=f(x_1)$, $x_3=f(x_2)$ y repetimos el proceso pongamos 5 veces. Por ejemplo partiendo de $x_0=1$ se obtiene $x_0=1$, $x_1=\frac{3}{2}$, $x_2=\frac{17}{12}$, $x_3=\frac{577}{408}$, $x_4=\frac{665857}{470832}$, $x_5=\frac{886731088897}{627013566048}$ cuyo valor numérico con 30 dígitos es $x_0=1.,x_1=1.5$,

$x_2=1.41666666666666666666666666667$, $x_3=1.41421568627450980392156862745$,

$x_4=1.41421356237468991062629557889$, $x_5=1.41421356237309504880168962350$

Considerando que $\sqrt{2}=1.41421356237309504880168872421$ este algoritmo converge muy rápidamente a $\sqrt{2}$.

¿Como afectan a un problema los cambios en los datos? Algunos cálculos son muy sensibles a los errores de redondeo y otros no. Es importante saber si esto depende del problema en si o bien del algoritmo que usamos para resolverlo. Ya que cuando se resuelve un problema numéricamente siempre aparecerán errores, es básico saber si el problema depende mucho o poco de las variaciones de los datos iniciales, un problema tal que variaciones pequeñas de los datos corresponden con variaciones pequeñas de la solución se dice *bien condicionado* y si variaciones pequeñas de los datos generan grandes variaciones de la solución se dice *mal condicionado.* Por ejemplo un ladrillo situado en el borde de una mesa plana puede caerse si se mueve un poco.

Por ejemplo resolver el sistema $x+y=1,\ 1.1x+y=2$ es un problema mal condicionado ya que tiene como solución $x=10, y=-9$, y si modificamos un poco los coeficientes el sistema $x+y=1,\ 1.05x+y=2$ tiene como solución $x=20, y=-19$. Un cambio de un 5% en un coeficiente cambia en un 100% la solución. La razón del mal condicionamiento del sistema es que las rectas que representan las ecuaciones

son casi paralelas, un pequeño cambio en los coeficientes hace que el punto de intersección se aleje mucho. Haz en Maxima plot2d([1-x,2-1.1*x],[x,5,25]) y plot2d([1-x,2-1.05*x],[x,5,25]) para verlo.

Otro ejemplo de problema mal condicionado es calcular el valor de la función $sen(\frac{\pi}{x})$ para x cerca de cero, por ejemplo en Maxima float(sin(%pi/.01)) da $1.964791... * 10^{-15}$ pero float(sin(%pi/.01001)) da $-0.308718494...$, la causa del mal condicionamiento es que la derivada de $sen(\frac{\pi}{x})$ toma valores grandes para x cerca de cero pero los valores de la función son pequeños; también calcular $cos(\pi\, x)$ está mal condicionado para x cerca de 10^6 ó calcular e^{-x} para x cerca de 10^6 (compruébalo) . Un problema mal condicionado es muy difícil de resolver, independientemente del algoritmo utilizado, porque los cambios producidos por los errores de redondeo cambiarán mucho el resultado.

¿Como afectan al algoritmo los errores de redondeo? Un algoritmo en el que los errores de redondeo se trasladan poco al resultado se dice *estable*, si no es así se dice que el algoritmo es *inestable*. Por ejemplo el algoritmo iterativo que dimos para calcular $\sqrt{2}$ es estable, en cambio si sumamos un millon de términos para calcular una serie es posible que los errores de redondeo afecten gravemente a la solución como se verá mas adelante al calcular e^{-10} sumando una serie.

Si consideramos resolver la ecuación $x^2 + 1000x + 1 = 0$ a través de la fórmula $x = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$ numéricamente, la solución $x_1 = \frac{-b-\sqrt{b^2-4ac}}{2a}$ es poco sensible a los cambios pequeños de b pero la solución $x_2 = \frac{-b+\sqrt{b^2-4ac}}{2a}$ si es sensible a los cambios pequeños de b por la cancelación de dígitos al restar los dos números cercanos b y $\sqrt{b^2-4ac}$. Este algoritmo es inestable para calcular x_2 si b es grande comparado con a y c (compruébalo). Sin embargo calcular x_2 a partir de $x_2 = \frac{1}{x_1}$ es un algoritmo estable (compruébalo). Como las raíces de una ecuación de segundo grado dependen continuamente de los coeficientes para $a \neq 0$ el problema está bien condicionado.

5 Fuentes básicas de errores

Los diversos tipos de errores que pueden aparecer al resolver numéricamente un problema son:

- Errores del *modelo matemático*. Son inevitables si el modelo matemático de la situación real es moderadamente simple.

  Por ejemplo al calcular la fuerza de tensión del cable de soporte del lado izquierdo del mástil de un barco, suponiendo que el soporte del cable derecho está flojo, que el mástil está unido al casco de manera que transmite fuerzas horizontales y verticales y que la fuerza del viento ejercida sobre el mástil varía en función de la distancia sobre la cubierta del barco. Si no se tienen en cuenta ciertos factores, por ejemplo la resistencia del aire, se produciría un error en la formulación del problema y las soluciones, tanto analíticas como numéricas serán erróneas.

  Otro ejemplo es la caída de un objeto, suponer que no hay resistencia del aire es válido para una caída de pocos metros de un objeto pesado pero no para un avión que vuela a gran velocidad o para una pluma. Por otra parte es más difícil obtener las soluciones de un modelo muy complicado.

- Errores producidos por la *inexactitud de los datos* físicos sobre los que se basa el modelo. Por ejemplo como consecuencia de medidas no exactas. En general los resultados de mediciones serán inexactos siempre y estos errores afectarán a la solución numérica del modelo (busca en Internet el principio de incertidumbre de Heisenberg).

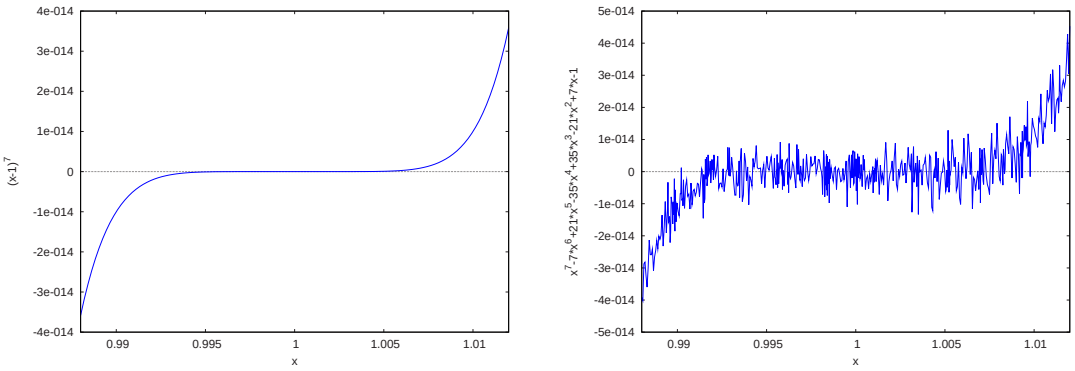

Figura 2.5.1: Errores de redondeo al evaluar una función

- Errores por *equivocación humana* al introducir mal los datos o programar incorrectamente. Debes comprobar siempre si los datos introducidos en el ordenador son correctos y si el programa funciona correctamente. Se conocen históricamente errores en las operaciones matemáticas de los ordenadores y en programas comerciales. Por otra parte es impensable realizar a mano los cálculos de un modelo medianamente sencillo. Es conveniente que dos personas distintas realicen los cálculos de forma independiente, con paquetes matemáticos distintos, y comprobar si los resultados coinciden.

- Errores de *truncamiento matemático*. Muchos de los modelos matemáticos, incluso los más simples, corresponden a procesos infinitos: integrar una función, derivar, tomar límite, procesos iterativos cuyo límite es la solución (método de Newton, métodos de punto fijo,...) Es imposible programarlos con un número infinito de pasos, pues el ordenador tardaría un tiempo infinito en hacerlos y no accederíamos nunca al resultado, con lo que al implementarlos en el ordenador se cometerá inevitablemente un error al sustituir, por ejemplo, el proceso infinito de calcular una integral evaluando una de sus sumas. Este tipo de error se conoce como error de truncamiento matemático y aparece en multitud de ocasiones.

  Muchas veces este error es malentendido al considerar el truncamiento sólo como una forma de obtener aproximaciones de los números.

- Error de *redondeo*. Éste se transmite al realizar operaciones y son debidos a que las computadoras solo guardan un número finito de cifras significativas: al realizar operaciones aritméticas el resultado debe ser nuevamente redondeado.

  Una consecuencia importante es que algunas propiedades aritméticas dejan de ser ciertas. Por ejemplo si denotamos por $a +^* b$ el resultado de redondear $a + b$ se tiene que $(a +^* b) +^* c \neq a +^* (b +^* c)$. Si sumamos en punto flotante 1 con un número suficientemente pequeño el resultado será 1 lo que matemáticamente es absurdo pero si sólo disponemos de un número dado de bits para guardar los dígitos esto es inevitable. Además cuantas más operaciones hacemos el resultado estará afectado de más errores.

  Este hecho se puede ver muy bien gráficamente, ya que si consideramos el polinomio $p(x) = x^7 - 7x^6 + 21x^5 - 35x^4 + 35x^3 - 21x^2 + 7x - 1 = (x-1)^7$ y, usando el paquete Maxima, realizamos la gráfica en el intervalo $[0.988, 1.012]$, observamos que lo que debería de ser una gráfica suave de un polinomio tiene una forma muy angulosa y aparentemente aleatoria, a este fenómeno se le suele llamar ruido al evaluar la función y ocurre cuando el intervalo en que se dibuja la función es demasiado pequeño.

  /* dibujemos el polinomio $(x-1)^7$ escrito factorizado en la variable p1 y expandido en la variable p2

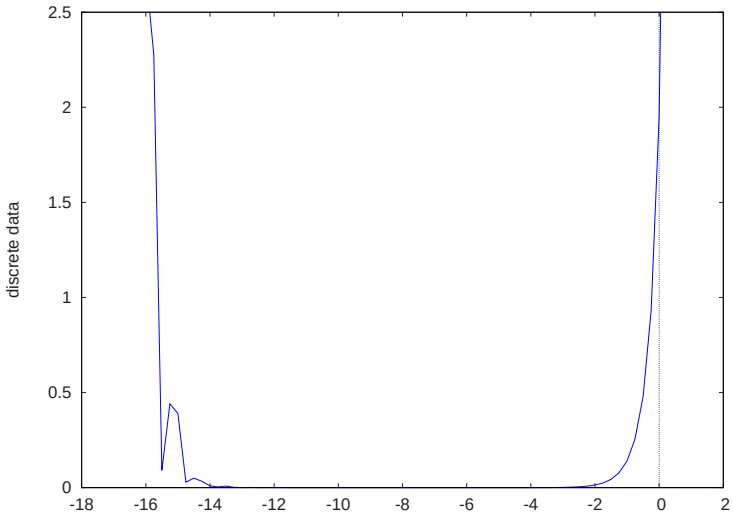

Figura 2.6.1: Errores al aproximar la derivada de exp(x) en $x = 1$ en función de h

*/

p1:(x-1)^7;

p2:expand(p1);

plot2d(p1,[x,0.988,1.012]);

plot2d(p2,[x,0.988,1.012], plot2d(p2,[x,0.988,1.012],[nticks,100]);

Esto ocurre por la cancelación severa de cifras significativas en el cálculo y se debe a que, para x cerca de 1, al calcular $x^7 - 7x^6 + 21x^5 - 35x^4 + 35x^3 - 21x^2 + 7x - 1$ se restan muchos números cercanos ($35x^3$ está cerca de $35x^4$ para x cerca de 1 etc.) además de que realizamos más operaciones que al calcular $(x - 1)^7$.

6 Relación entre el error de redondeo y el error de truncamiento

Nos parece interesante comentar la relación que existe entre el error de truncamiento matemático y el error de redondeo. Por ejemplo, si, para minimizar el error de truncamiento al calcular una integral, tomamos una suma con un millón de términos, además de tardar más en efectuarse el cálculo, nos encontraremos que aumenta el número de operaciones y por tanto el error de redondeo acumulado. Recíprocamente, si por disminuir el error de redondeo reducimos mucho el número de términos de la suma, la aproximación a la integral puede ser inaceptable.

El comportamiento de la influencia conjunta de los dos errores puede representarse, de modo intuitivo, en la figura 2.6.1 que está generada en Maxima con

kill(errder);errder(n):=ev(abs((exp(1.+10^n)-exp(1))/10^n-exp(1.)),numer);

lis:makelist([n/4,errder(n/4)],n,-72,4); plot2d([discrete,lis],[y,0,2.5]);

donde dibujamos el error cometido al aproximar la derivada de $exp(x)$ en $x = 1$ por la fórmula $f'(x) =$

$\frac{f(x+h)-f(x)}{h}$, tomamos h en la forma 10^n y dibujamos los valores de n y los valores del valor absoluto de $\frac{exp(1+10^n)-f(1)}{10^n} - exp(1)$. Observa que en el eje horizontal están los logaritmos de h.

Si el paso es grande el error de truncamiento se hace grande y por tanto el error total también se hace grande. Si el paso es excesivamente pequeño el error de redondeo se hace grande y por tanto el error total aumenta. Es preciso un compromiso de forma que no se dispare ninguno de los dos errores.

Otro ejemplo puede ser tomar 10^4 términos de la suma para calcular $\sum_1^\infty \frac{(-1)^n}{n} = -1 + \frac{1}{2} - \frac{1}{3} + \frac{1}{4} - \frac{1}{5} + ...$ que trivialmente es $-\ln(2) \sim -0.693$, además como los sumandos cambian de signo se producen errores por cancelación de cifras.

También podemos considerar el cálculo de e^{-10} sumando la serie de Taylor directamente $\sum_0^\infty \frac{(-10)^n}{n!} = 1 - \frac{10}{1} + \frac{10^2}{2} - \frac{10^3}{6} + \frac{10^4}{24} + ...$ Al ser una serie alternada se tiene que el primer término despreciado acota el error. Sin embargo, como los términos $\frac{(-10)^n}{n!}$ crecen al principio en valor absoluto (para $n = 10$ vale 2.75573×10^3), se producen cancelaciones tan severas que arruinan completamente el resultado. En efecto, observamos que la aproximación de la serie de Taylor llega a tomar el valor de 3.102×10^{-2}, en contraste con el valor de la función que es aproximadamente 4.54×10^{-5}. Ejecuta en Maxima las instrucciones siguientes para observar que efectivamente la aproximación de la serie de Taylor llega a tomar el valor indicado.

/*Primero creamos una lista con los valores de las sumas de la serie de taylor, observa que Maxima trabaja con numeros racionales exactos */

create_list(sum((-10)^i/i!,i,0,n),n,1,40);

/* estos valores expresados aproximadamente son */

create_list(float(sum((-10)^i/i!,i,0,n)),n,1,40);

/* que son muy exactos */

/* Si hacemos los calculos en punto flotante tenemos */

create_list(sum(float((-10)^i/i!),i,0,n),n,1,40);

/* que no están mal ya que float usa 16 dígitos */

/*usamos fpprec para generar menos dígitos en bfloat y que se vean las cancelaciones*/

fpprec:4;

create_list(sum(bfloat((-10)^i/i!),i,0,n),n,1,40);

/* los errores relativos son muy grandes y no mejoran sumando más términos */

La solución, por supuesto, es calcular el inverso de e^{10} o, mejor aún, calcular e y elevarlo a -10.

7 Ejercicios

1. Calcula el número x, $x > 0$ más pequeño que Maxima puede representar en punto flotante. (Sugerencia prueba con float(10^-n) con $n \sim 310$ a ver si redondea a cero o no)

2. Calcula el número x, $x > 0$ más grande que Maxima puede representar en punto flotante. (Sugerencia

prueba con float(10^n) con $n \sim 310$ a ver si da error)

3. Ejecuta en Maxima la orden create_list(float(1+10^-i)-1,i,1,30); y observa como no hay más de 16 dígitos en formato punto flotante y cómo al representar los números en formato binario se pierde precisión (el resultado exacto sería $1 - 10^{-i} + 1 = 10^{-i}$ para i=1,2,3...30).

4. ¿Es posible calcular con el ordenador el valor de $\sum_{n=1}^{\infty} \frac{1}{n} = \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + ...$?

5. ¿Vale lo mismo calculando en punto flotante en el ordenador $\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + ... \frac{1}{10000}$ si lo sumamos empezando por 1 o si empezamos por $\frac{1}{10000}$? ¿Por qué? (Ayuda: Prueba sum(float(1/i),i,1,10000); para sumar empezando por 1 y sum(float(1/(10001-i)),i,1,10000); para empezar por $\frac{1}{10000}$). Si no hay gran diferencia haz $fpprec : 5$; y usa $bfloat$ en lugar de $float$.

6. Considera los siguientes valores de p y $p*$. ¿Cuál es el error absoluto y el error relativo al aproximar $p*$ por p?

7. $p = 0.6371 \times 10^2$, $p* = 0.9 \times 10^2$

8. $p = 0.12 \times 10$, $p* = 0.1 \times 10$

9. $p = 0.1289725 \times 10^5$, $p* = 0.1289705 \times 10^5$

10. Obtener las soluciones de la ecuación $x^2 + 62.10x + 1 = 0$, con redondeo a cuatro dígitos. Reformular el problema para evitar la pérdida de cifras significativas debida al error de redondeo (observa que b y $\sqrt{b^2 - 4ac}$ son muy cercanos en valor absoluto).

CAPÍTULO 3

Ecuaciones no lineales.

Índice del Tema

Este capítulo trataremos el problema de determinar raíces de ecuaciones (o ceros de funciones). Es un problema que surge en trabajos científicos, por ejemplo, en la teoría de la difracción de la luz necesitamos las raíces de la ecuación

$$x - \tan x = 0.$$

En el cálculo de las órbitas planetarias necesitamos las raíces de la ecuación de Kepler

$$x - a \operatorname{sen} b = 0,$$

para valores de a y b.

Los **Objetivos** de éste tema son:

- Conocer algunos métodos de resolución de ecuaciones no lineales: bisección, falsa posición, newton y secante.
- Ser capaz de utilizar las herramientas que existen en Maxima y los programas que se describen para resolver ecuaciones o sistemas no lineales.
- Comprender el funcionamiento de los métodos iterativos.
- Entender los conceptos de rapidez de convergencia de un método y la importancia de tener un criterio de parada.

Recordemos el **Teorema de Bolzano:** Si $f : [a, b] \in \mathbb{R} \to \mathbb{R}$ es continua $f(a)$ y $f(b)$ tienen distinto signo entonces existe un $c \in$(a,b) tal que $f(c) = 0$.

Comenzamos con los métodos basados en el teorema de Bolzano, también denominados métodos de intervalos: método de bisección y el método de la cuerda o de la régula falsi o de la regla falsa o de la falsa posición.

1 Método de bisección

Sea f una función continua, definida en el intervalo [a,b], con $f(a)$ y $f(b)$ de signos distintos. Entonces, por el teorema de Bolzano, existe p, $a < p < b$, tal que $f(p) = 0$. La pregunta es *¿cómo encontrarlo?*. La idea es ir partiendo el intervalo en dos trozos iguales hasta encontrarla. Para empezar tomamos $a_1 = a$ y $b_1 = b$ y sea p_1 el punto medio de [a,b], o sea, $p_1 = \dfrac{a_1 + b_1}{2}$.

Puede ocurrir:

- Si $f(p_1) = 0$, entonces $p = p_1$ hemos encontrado la raíz y paramos.
- Si $f(p_1)f(a_1) > 0$, entonces $p \in (p_1, b_1)$, y tomamos $a_2 = p_1$ y $b_2 = b_1$.
- Si $f(p_1)f(a_1) < 0$, entonces $p \in (a_1, p_1)$, y tomamos $a_2 = a_1$ y $b_2 = p_1$.

Observa que $b_2 - a_2 = \frac{b-a}{2}$ y que por la elección de a_2, b_2 la raíz está dentro del intervalo $[a_2, b_2]$ con lo que el error se ha dividido por 2.

Ahora aplicamos el proceso al intervalo $[a_2, b_2]$ y tomamos $p_2 = \dfrac{a_2 + b_2}{2}$ el punto medio de $[a_2, b_2]$

- Si $f(p_2) = 0$, entonces $p = p_1$ hemos encontrado la raíz y paramos.
- Si $f(a_2)f(p_2) > 0$, entonces $p \in (b_2, p_2)$, y tomamos $a_3 = p_2$ y $b_3 = b_2$.
- Si $f(a_2)f(p_2) < 0$, entonces $p \in (a_2, p_2)$, y tomamos $a_3 = a_2$ y $b_3 = p_2$.

Observa que $b_3 - a_3 = \frac{b-a}{2^2}$ y que por la elección de a_3, b_3 la raíz está dentro del intervalo $[a_3, b_3]$ con lo que el error se ha dividido por 2.

Repetimos el proceso hasta encontrar $f(p)$ o hasta que el error sea suficientemente pequeño.

Nótese que para empezar el algoritmo se debe de encontrar un intervalo $[a, b]$ tal que $f(a) * f(b) < 0$. Es conveniente escoger el intervalo tan pequeño como sea posible porque en el método de la bisección, la longitud del intervalo que contiene al cero de f se reduce por un factor de dos que no es demasiado rápido.

Hay veces que para buscar el intervalo $[a, b]$, tal que $f(a) \cdot f(b) < 0$, es conveniente ver la gráfica de la función,

Usar el método de la bisección para encontrar la raíz más cercana a 0 *de la ecuación* $e^x = \operatorname{sen} x$

3.1.1 Criterio de paro

Observamos que en el método de la bisección el proceso para buscar la solución aproximada se repite hasta obtener una buena aproximación. Entonces nos preguntamos *¿cuándo debe terminar el método?*. Una sugerencia sería finalizar el cálculo cuando el error se encuentre por debajo de algún nivel prefijado. Así, dada una tolerancia $\epsilon > 0$, podemos generar $p_1, p_2, \cdots, p_N$ hasta que se verifique una de las condiciones siguientes:

- $| p_N - p_{N-1} | < \epsilon$ (error absoluto).
- $| \dfrac{p_N - p_{N-1}}{p_N} | < \epsilon, \qquad p_N \neq 0$ (error relativo).
- $| f(p_n) | < \epsilon$ ($f(p_n)$ cerca de cero).

Pueden surgir dificultades usando cualquiera de estos criterios. Usualmente la segunda desigualdad es la mejor para utilizar, porque:

(a) Existen sucesiones $\{p_n\}$ tal que las diferencias $p_n - p_{n-1}$ converjan a cero, mientras que la sucesión diverge.

(b) Es posible que $f(p_n)$ esté muy próximo a cero pero p_n difiera de p.

Aunque si el valor de la raíz es cercano a cero puede haber problemas.

3.1.2 Convergencia

El algoritmo de la bisección tiene la ventaja que converge siempre a una solución pero tiene el inconveniente que converge lentamente y, además, una buena aproximación intermedia puede ser desechada. Como la longitud del intervalo en que está la solución se divide por 2 cada vez que repetimos el proceso, si repetimos el proceso de bisección 3 veces la longitud de intervalo se divide por 8, si son 4 la longitud de intervalo se divide por 16 etc. Para que la longitud se divida por 10^k hay que repetir el proceso $\frac{k}{\log_{10}(2)} \simeq \frac{k}{0.301030}$ asi que para que la longitud del intervalo se divida por 10^6 hay que repetir el proceso 20 veces.

3.1.3 Algoritmo de bisección con Maxima

Aplicamos el método de la bisección a la ecuación $\cos(x)\cosh(x) + 1 = 0$.

En primer lugar obtenemos la gráfica de la ecuación que nos indicará el intervalo donde se encuentra la solución. Así, en la siguiente figura, vemos que la función $\cos(x)\cosh(x) + 1$ tiene en el intervalo $[-2, 2]$ dos soluciones,

/* pintamos la función */

plot2d(cos(x)*cosh(x)+1,[x,-2,2]);

/*Vamos a implementar el algoritmo con un bucle, observa que no comprobamos si hay alternancia de signo al principio, por lo que puede dar resultados erróneos */

bisec(f,a,b,n,err):=block([nn:0,tem:(a+b)/2,x0:a,x1:b],while (nn<n and abs(x1-x0)>err) do (nn:nn+1,if (ev(f,x=tem)*ev(f,x=x0)<0) then (x1:tem, tem:(x0+x1)/2) else (x0:tem, tem:(x0+x1)/2)), if nn=n then print(" no alcanzamos la solucion con el error aceptable, la aproximacion es", tem) else tem);

/*comprueba buscando $\sqrt{2}$ entre 1 y 2 con hasta 30 iteraciones */

bisec(x^2-2,1,2,30,10^-3);

/* si quieres verlo en forma decimal haz*/

%,numer;

/*idem buscando $\sqrt{2}$ entre 1 y 2 con pocas iteraciones */

bisec(x^2-2,1,2,3,10^-3);

/* si no hay alternancia de signo el resultado es impredecible (observa que la raíz esta entre 1 y 2 y no entre 0 y 1*/

bisec(x^2-2,0,1,3,10^-3);

/* Vamos a hacerlo de otra manera, poco a poco y usando una lista que tiene los valores de los extremos del intervalo en cada momento a, b y la función f(x) recuerda que las listas se escriben entre [] y que los elementos se indican como lista[1], lista[2] etc., usamos xm para guardar $\frac{a+b}{2}$, esta version sólo devuelve el intervalo pequeño correcto, observa que no comprobamos si $f(\frac{a+b}{2}) = 0$ o no, ni tampoco si hay alternancia del signo de f en los extremos del intervalo al principio*/

kill(bolza1);

bolza1(arg):=block([a:arg[1],b:arg[2],f:arg[3],xm:(arg[1]+arg[2])/2], if (ev(f,x=xm)*ev(f,x=a)<0) then [a,xm,f] else [xm,b,f]);

/* preparamos el valor de a,b,f y ahora repetimos 20 veces, lo hacemos para calcular $\sqrt{2} \sim 1.414213562373095$ resolviendo $x^2 - 2 = 0$ en el intervalo [1,2] */

ini:[1,2,x^2-2]; for i:1 thru 20 do (print(ini),ini:bolza1(ini));

/* Observa que el resultado esta dado como fracciones porque Maxima intenta minimizar los errores, repetimos poniendo print(float(ini)) en el bucle, fíjate como se acerca a la solución y que la solución está dentro de cada intervalo*/

ini:[1,2,x^2-2]; for i:1 thru 20 do (print(ini),ini:bolza1(ini));

/* vamos a controlar ahora que la longitud del intervalo (que es una cota del error absoluto) se haga pequeña por ejemplo menor que err=10^{-3} */

ini:[1,2,x^2-2]; err:10^-3;

while abs(ini[1]-ini[2])>err do (print(float(ini)),ini:bolza1(ini));

/* quizás te interese ver solamente el resultado */

ini:[1,2,x^2-2]; err:10^-3;

while abs(ini[1]-ini[2])>err do (ini:bolza1(ini)); print(" el valor aproximado es ", float((ini[1]+ini[2])/2));

/* Comprobamos que la función tiene alternancia de signo en los extremos al principio, si no es así devolvemos un error y paramos*/

ini:[1,2,x^2-2]$ err:10^-3$

if ev(ini[3],x=ini[1])*ev(ini[3],x=ini[2])>0 then print("error en los datos f(a) y f(b) tienen igual signo") else (while abs(ini[1]-ini[2])>err do (ini:bolza1(ini)), print(" el valor aproximado es ", float((ini[1]+ini[2])/2)))

/* comprueba que funciona usando ini=[1,2,x^2-2] que va bien y ini=[0,1,x^2-2] que no cumple la condición $f(0) * f(1) < 0$ */

/* prueba con otros valores de error (si el error es excesivamente pequeño puede tardar bastante) y otras funciones, observa que el programa se puede mejorar para que refleje si $f(xm) = 0$, se puede cambiar la condición de parada basada en el error absoluto por otra basada en el error relativo, etc. */

A continuación exponemos una variante del método de la bisección.

2 Método de regula falsi o de falsa posición

Suponiendo que una raíz de la ecuación $f(x) = 0$ que se encuentra en el intervalo $[a_i, b_i]$, calculamos la intersección con el eje x de la recta que une los puntos $(a_i, f(a_i))$ y $(b_i, f(b_i))$, denotando este punto por p_i:

$$\frac{x-a_i}{b_i-a_i}=\frac{y-f(a_i)}{f(b_i)-f(a_i)}$$

Expresamos la ecuación en forma explícita,

$$y=\frac{f(b_i)-f(a_i)}{b_i-a_i}(x-a_i)+f(a_i)$$

El punto p_i se obtendrá haciendo la intersección de esta recta con el eje horizontal $y=0$, operando se llega a que

$$p_i=x=\frac{a_if(b_i)-b_if(a_i)}{f(b_i)-f(a_i)}$$

Si $f(p_i)f(a_i)<0$, definimos $a_{i+1}=a_i$ y $b_{i+1}=p_i$. Si $f(p_i)f(b_i)<0$, definimos $a_{i+1}=p_i$ y $b_{i+1}=b_i$. Generamos de esta forma una sucesión de intervalos que contendrá a la solución.

/*El programa en Maxima es muy similar, sólo hay que cambiar la forma de obtener el nuevo punto*/

kill(falsapos);

falsapos(arg):=block([a:arg[1],b:arg[2],f:arg[3],xm], xm:(a*ev(f,x=b)-b*ev(f,x=a))/(ev(f,x=b)-ev(f,x=a)),

if (ev(f,x=xm)*ev(f,x=a)<0) then [a,xm,f] else [xm,b,f]);

/*el resto es igual, por ejemplo haz */

ini:[1,2,x^2-2]; for i:1 thru 20 do (print(ini),ini:falsapos(ini));

Ejercicio:Encontrar una raíz de $6x-e^x=0$ en el intervalo $[0,1]$ usando el método de la falsa posición.

El método de la falsa posición puede ser mucho más rápido que el de la bisección, pero también puede darse el caso contrario dependiendo de la función concreta y del orden de los puntos iniciales. Este hecho lo podemos demostrar aplicando ambos métodos a la función $f(x)=x^6-1$ y comprobando que mientras que el método de la bisección requiere 8 aproximaciones para encontrar una solución en el intervalo [0,1.5] con un error aproximado porcentual menor que 1 centésima, el método de la cuerda necesita hasta 16 aproximaciones.

3 Iteración de punto fijo

Se trata de hallar aproximaciones de ecuaciones del tipo $x=g(x)$. A una solución x_0 de esta ecuación se le denomina un **punto fijo** de la función g, ya que $g(x)$ lleva x_0 en él mismo.

El problema de búsqueda de las raíces de $f(x)=0$ tiene soluciones que corresponden a los puntos fijos de $g(x)=x$ con $g(x)=x-f(x)$. Por tanto, lo que primero tenemos que hacer es estudiar cuándo una función tiene un punto fijo y cómo se puede determinar este punto fijo.

Veamos gráficamente que la función $g(x)=x^2-2$ tiene exactamente dos puntos fijos en $x=-1, x=2$

/* pintamos las dos funciones */

plot2d([x,x^2-2],[x,-2,4]);

Unas condiciones suficientes para la existencia y unicidad de un punto fijo son:

Si g es continua en $[a,b]$ y $g(x) \in [a,b]$ para todo $x \in [a,b]$, entonces g tiene un punto fijo en $[a,b]$. Si además, g es derivable en (a,b) y $\mid g'(x) \mid \leq k < 1$ para todo $x \in (a,b)$, entonces g tiene un único punto fijo p en $x \in [a,b]$.

Por ejemplo $g(x) = \frac{x^2-1}{3}$ es continua en [-1,1], el valor absoluto de su derivada $|g'(x)| = |\frac{2x}{3}|$ esta acotada en [-1,1] por $\frac{2}{3}$ y $g([-1,1]) \subset [-1,1]$ así que satisface las condiciones anteriores. Haciendo float(solve(x=(x^2-1)/x)); se ve que hay puntos fijos en $x \sim -0.30277563773199$ y $x \sim 3.302775637731995$

3.3.1 Iteración funcional

Para aproximar un punto fijo de una función g, escogemos una aproximación inicial p_0 y generamos una sucesión $\{p_n\}_{n=1}^{\infty}$ tal que $p_n = g(p_{n-1})$, para todo $n \geq 1$, los primeros elementos de la sucesión son $p_0, g(p_0), g(g(p_0)), g(g(g(p_0))), ...$

Ejercicio: toma una calculadora de bolsillo escribe un número positivo, y pulsa repetidas veces la tecla de raíz cuadrada. Observa si el resultado se acerca al valor 1.

Si $\{p_n\}_{n=1}^{\infty}$ converge a p y g es continua resulta que

$$p = \lim_{n\to\infty} p_n = \lim_{n\to\infty} g(p_{n-1}) = g(\lim_{n\to\infty} p_{n-1}) = g(p),$$

es decir, el punto fijo es el límite de la sucesión. Esta técnica es conocida como **técnica iterativa de punto fijo** o **iteración funcional**. El procedimiento lo detallamos en la siguiente figura.

La programación en Maxima es mas simple porque, si definimos g fuera, solo tenemos que llevar el valor de p_n

/* guardamos en la variable ggg la función g por ejemplo $\frac{x^2-1}{3}$ y llamamos al valor actual pn*/

ggg:(x^2-1)/3; kill(iter);iter(pn):=ev(ggg,x=pn);

/* ahora repetimos el proceso igual que antes, por ejemplo empezando desde $pn = 0$ */

pn:0; for i:1 thru 20 do (print(float(pn)),pn:iter(pn));

/* observa lo bien que converge */

La cuestión es garantizar que la función g converja a una solución de $x = g(x)$ y escoger g de tal manera que la convergencia sea tan rápida como sea posible.

Dada la ecuación $x^3 + 4x^2 - 10 = 0$. Determina de cinco formas diferentes la función g (por ejemplo prueba $x = \sqrt[3]{10 - 4x^2}$ despejando la x de x^3 o bien $x = \sqrt{10 - x^3}/2$ despejando de x^2 o bien $x = x^3 + 4x^2 - 10$ sumando x a ambos lados de la ecuación).

Ejercicio: Aplica el método de iteración de punto fijo para las cinco alternativas de g y estudia si convergen.

Con las condiciones anteriores de existencia y unicidad se puede garantizar que, si p_0 es un número cualquiera de $[a,b]$, entonces la sucesión

$$p_n = g(p_{n-1}), \qquad n \geq 1,$$

converge al único punto fijo $p \in$ [a,b], se puede demostrar también que

Si g satisface las condiciones (3), una cota para el error involucrado al usar p_n para aproximar a p está dada por

$$| p_n - p | \leq k^n \, max\{p_0 - a, b - p_0\} \qquad \text{para cada} \quad n \geq 1.$$

Por tanto la rapidez de convergencia está relacionada con la cota k de la derivada de g en [a,b]. Es claro que cuanto más pequeño se puede hacer k, más rápida será la convergencia.

Ejercicio: Obtener mediante el método de iteración una solución aproximada de la ecuación $e^{-x} = x$ en el intervalo [0.3,1]. ¿Cuántas iteraciones hacen falta para que el error sea menor que 10^{-3}?

(Si Maxima deja las exponenciales sin calcular para mantener más precisión prueba a cambiar

ggg:float(exp(-x)); o bien kill(iter);iter(pn):=ev(ggg,x=pn,numer);

o bien pn:0; for i:1 thru 20 do (print(float(pn)),pn:float(iter(pn)));)

Dada $g \in C^1([a,b])$ Es posible que el método de iteración dado por p_0, $p_{n+1} = g(p_n)$ no converja a un punto fijo para todo valor inicial $p_0 \in [a, b]$ y, sin embargo, el método *converge localmente* es decir existe un $\epsilon > 0$ tal que para todo $p_0 \in (p - \epsilon, p + \epsilon)$ la sucesión p_n converge a p. En este caso se dice que p es un **punto atractivo**.

Se puede probar que si g una función definida en $[a, b]$, $g \in C^1([a,b])$. Si g tiene un punto fijo p en $[a, b]$ y $| g'(p) | < 1$ entonces p es un punto atractivo. Por tanto, a efectos de convergencia, es conveniente que la derivada de g en el punto fijo sea < 1 en valor absoluto. En algunos problemas es suficiente tomar p_0 arbitrariamente, mientras que en otros es muy importante seleccionar una buena aproximación inicial.

4 El método de Newton

El método de Newton, a veces llamado de Newton-Raphson, es uno de los métodos numéricos más conocidos y poderosos para la resolución del problema de búsquedas de raíces de $f(x) = 0$. Hay diversas maneras de introducir este método, una de ellas es mediante un enfoque intuitivo basado en el polinomio de Taylor.

Supongamos que f es continuamente diferenciable dos veces en el intervalo $[a, b]$, es decir, $f \in C^2[a, b]$. Sea p una solución de la ecuación $f(x) = 0$. Sea $p_0 \in [a, b]$ una aproximación a p. Por el teorema de Taylor sabemos que

$$0 = f(p) = f(p_0 + h) = f(p_0) + f'(p_0)h + \mathcal{O}(h^2),$$

donde $h = p - p_0$. Si h es pequeña (es decir, p_0 es próximo a p), es razonable ignorar el término $\mathcal{O}(h^2)$ y resolver el resto de la ecuación para h. En consecuencia,

$$h = -\frac{f(p_0)}{f'(p_0)}.$$

Como $h = p - p_0$, deducimos

$$p = p_0 - \frac{f(p_0)}{f'(p_0)}.$$

El método de Newton comienza con una estimación p_0 de p a partir de la cual se define inductivamente una sucesión de aproximaciones

$$p_n = p_{n-1} - \frac{f(p_{n-1})}{f'(p_{n-1})}, \qquad n \geq 1 \quad y \quad f'(p_{n-1}) \neq 0$$

Geométricamente los p_n son los puntos de corte con el eje X de la recta tangente a f en p_{n-1}.

Este método es una técnica de iteración funcional $p_n = g(p_{n-1}), \quad n \geq 1$ para la cual

$$p_n = g(p_{n-1}) = p_{n-1} - \frac{f(p_{n-1})}{f'(p_{n-1})}, \qquad n \geq 1.$$

Se ve claramente que si $f'(p_{n-1}) = 0$ para algún n, el método no puede continuarse.

A continuación ilustramos gráficamente cómo se obtiene las aproximaciones usando tangentes sucesivas.

Se puede probar que el método de Newton converge si $f \in C^2[a,b]$, $p \in [a,b]$ con $f(p) = 0$, $f'(p) \neq 0$ y la aproximación inicial p_0 está lo suficientemente cerca a p.

Igual que antes es preciso imponer una *condición de parada* fijado $\epsilon > 0$ construimos una sucesión $p_1, \ldots, p_n$ hasta que ocurra uno de los tres casos siguientes:

1. $| p_n - p_{n-1} | < \epsilon$, (error absoluto).
2. $\frac{| p_n - p_{n-1} |}{| p_n |} < \epsilon, \qquad p_n \neq 0$ (error relativo).
3. $| f(p_n) | < \epsilon$ (función pequeña).

Cualquiera de las condiciones anteriores tiene sus ventajas e inconvenientes.

La programación del método de Newton en Maxima es simple, si definimos g y g' fuera,

/* definimos g en la variable ggg por ejemplo $\frac{x^2-1}{3}$ en ggp ponemos la derivada y llamamos al valor actual pn*/

ggg:(x^2-1)/3;ggp:diff(ggg,x); kill(newt);newt(pn):=ev(x-ggg/ggp,x=pn);

/* ahora repetimos el proceso igual que antes, por ejemplo empezando desde $p_n = 0.9$ */

pn:0.9; for i:1 thru 10 do (print(float(pn)),pn:newt(pn));

/* observa lo bien que converge */

/* empieza por otros valores de g_0 y observa el comportamiento de la sucesión, por ejemplo toma $p_0 = -2.9$, $p_0 = 0, p_0 = 100$ Si los cálculos son lentos haz pn:float(newt(pn)) para que se hagan en punto flotante */

Ejercicio: Obtener la solución única de $x^3 + 4x^2 - 10 = 0$ en el intervalo $[1,2]$ mediante el método de Newton. Comparar los resultados obtenidos con los obtenidos en el ejemplo 3.3.1 por el método iterativo.

Puedes programar en Maxima el método de Newton de forma más sofisticada haciendo

newton(f,x,err):=([y,df,dfx],df:diff(f('x),'x),

do (y:ev(df),x:x-f(x)/y,if abs(f(x))<err then return (x)));

/* Observa que 'x impide la evaluación de x para poder calcular la derivada. Si queremos calcular $\sqrt{2}$ partiendo de $g_0 = 100$ con la condición de parada que el valor de $g(x_n) < 10^{-3}$ ponemos */

fun(x):=x^ 2-2; newton(fun,100,10^ -3);

Ejercicio: Aproximar la solución de $e^{-x} - x = 0$ por el método de Newton en $[0, 1]$, hasta que el error relativo sea menor que 10^{-5}.

Normalmente, si partimos de p_0 suficientemente próximo a la solución el método de Newton converge muy rápidamente. En la práctica se suele usar un método tipo bisección para aproximarse a la raíz, y cuando estamos suficientemente cerca, se usa el método de Newton.

3.4.1 Raices múltiples

Si resolvemos $sen(x) - x = 0$ con $p_0 \sim 0$ tenemos que la convergencia a cero es muy lenta, para verlo haz en Maxima

ggg:sin(x)-x;ggp:diff(ggg,x); kill(newt);newt(pn):=ev(x-ggg/ggp,x=pn);

pn:0.1; for i:1 thru 10 do (print(float(pn)),pn:newt(pn));

y observa cómo después de 10 iteraciones la aproximación es 0.0026004496692008. Esto no es normal porque el método de Newton converge rápidamente cerca de la raíz, en general dobla el número de dígitos exactos en cada iteración.

Si observas que el método de Newton converge lentamente debes considerar la posibilidad de raíces múltiples. El método de Newton funciona mal en el caso en que la raíz p sea también un cero de la derivada ya que en $p_n = p_{n-1} - \frac{f(p_{n-1})}{f'(p_{n-1})}$ el denominador estará cerca de cero. La solución es modificar el método usando en lugar de la función g la función $h = \frac{g}{g'}$, es fácil comprobar que si g tiene una raíz múltiple entonces h tiene una raíz simple y el método de Newton funcionará más deprisa (el inconveniente es que se realizan más cálculos). Compruébalo haciendo

ggg:(sin(x)-x)/diff(sin(x)-x,x);ggp:diff(ggg,x);

pn:0.1; for i:1 thru 10 do (print(float(pn)),pn:newt(pn));

en la segunda iteración ya da $2.2140276631911308 * 10^{-11}$ y en la siguiente aparece una división por cero (como es normal ya que es un cero múltiple).

5 Método de la secante

Algunas veces el método de Newton no es adecuado, por ejemplo porque la derivada de la función es muy complicada, y se usa una variación del método de la *falsa posición* para afinar la raíz. Recuerda que calculamos la intersección con el eje x de la recta *secante* que une los puntos $(a, f(a))$ y $(b, f(b))$, haciendo la intersección de esta recta con el eje horizontal $y = 0$, se llega a que $x = \dfrac{af(b) - bf(a)}{f(b) - f(a)}$.

El *método de la secante* consiste en dados dos puntos iniciales p_0, p_1 generar la sucesión $p_0, p_1, p_2, ...$ a partir de la relación

$$p_n = \frac{p_{n-2}f(p_{n-1}) - p_{n-1}f(p_{n-2})}{f(p_{n-1}) - f(p_{n-2})}. \tag{3.5.1}$$

Observa que, a diferencia del método de la *falsa posición*, no comprobamos si hay alternancia de signos para elegir un intervalo, por tanto no está garantizado que se acerque a la solución; sin embargo, si partimos de dos puntos suficientemente próximos a la solución el método de la secante converge rápidamente (pero no tanto como el método de Newton).

/*El programa en Maxima es sencillo, necesitamos llevar los dos puntos, de paso llevamos la función también, observa que la entrada es [a,b,f] y la salida es [b,xsec,f]*/

kill(secante);

secante(arg):=block([a:arg[1],b:arg[2],f:arg[3],xsec], xsec:(a*ev(f,x=b)-b*ev(f,x=a))/(ev(f,x=b)-ev(f,x=a)), [b,xsec,f]);

/*el resto es igual, por ejemplo para calcular $\sqrt{2}$ haz */

ini:[1,2,x^2-2]; for i:1 thru 10 do (print(float(ini)),ini:secante(ini));

Observa que converge muy bien a $\sqrt{2}$ pero un poco peor que el método de Newton.

Maxima incorpora la función *find_root(f,x,a,b)* y *newton*, hay que hacer load (newton1); para cargarla.

6 Sistemas de ecuaciones no lineales

El método de Newton puede extenderse a sistemas de ecuaciones no lineales.

En Maxima el paquete $mnewton$ incluye la función

$mnewton([listafunciones],[listavariables],[listavaloresiniciales])$; para resolver sistemas de ecuaciones no lineales. Por ejemplo, para resolver numéricamente $3x-cos(x+y)-1/2=0, exp(x+y)=x*cos(y)+2$ a partir de los valores iniciales $x=0, y=0$,

hacemos load(mnewton);

mnewton([3*x-cos(x+y)-1/2,exp(x+y)-x*cos(y)-2],[x,y],[0,0]);

que nos devuelve [[x=0.33333333333333,y=0.5042735042735]]

sustituyendo los valores obtenidos en 3*x-cos(x+y)-1/2,exp(x+y)-x*cos(y)-2 haciendo

ev([3*x-cos(x+y)-1/2,exp(x+y)-x*cos(y)-2],[[x=0.33333333333333,y=0.5042735042735]],numer);

tenemos [-0.16924296469694,0.018988253175311] que no es excesivamente bueno. En el caso de varias ecuaciones no lineales con varias variables es muy importante partir de buenas aproximaciones iniciales.

7 Ejercicios

1. Sea la ecuación
$$\operatorname{sen}(x) - 2\cos(2x) + x^2 = \pi^2 - 2$$
   (a) Demostrar, gráficamente y aplicando Bolzano, que admite al menos una raíz en el intervalo $[0, 3\pi]$.

(b) Utilizar el comando *find_root* para aproximar la raíz considerando como valor inicial $x_0 = 5$.

(c) ¿Se obtiene raíz exacta? ¿Con qué error ha sido aproximada?.

2. Aplica reiteradamente el método de bisección para hallar, con error menor que 10^{-3}, la raíz de $x = \tan(x)$ que está entre 4 y 4.5. Compara con la solución obtenida por *find_root*.

3. Usa el método de bisección para aproximar la raíz de:

(a) $f(x) = \sqrt{x^2+1} - \tan x$, comenzando en el intervalo $[0.5, 1]$ hasta que $|\epsilon_a| < 1\%$.

(b) $f(x) = e^{-x^3} - 2x + 1$ comenzando en el intervalo $[0.75, 1]$, hasta que $|\epsilon_a| < 1\%$.

4. Demostrar que la función $f(x) = x^3 + e^x - 10$ tiene una única raíz real.

(a) Determinar un intervalo en el cual esté la raíz.

(b) Dibujar la gráfica de la función.

(c) Calcular el número de iteraciones para obtener una aproximación con 10 decimales con el algoritmo de la bisección.

5. Define una función que implemente el método de la régula falsi, dados a, b. Aplícala reiteradamente para hallar, con error menor que 10^{-3} la raíz de $x = \cos(x)$ que está entre 0 y 1. Compara con la solución obtenida por *find_root*.

6. Usa el método de la régula falsi para aproximar la raíz de,

(a) $f(x) = 4 - x^2 - x^3$ comenzando en el intervalo $[1, 2]$, hasta que $|\epsilon_a| < 1\%$.

(b) $f(x) = \ln x + x^2 - 4$ comenzando en el intervalo $[1, 2]$, hasta que $|\epsilon_a| < 1\%$.

7. Aplicar el método de la bisección y de la regla falsa a la función $f(x) = x^6 - 1$, comenzando en el intervalo [0,1.5] hasta que $|\epsilon_a| < 1\%$.¿Cuál de los dos métodos converge más rápido?

8. Los primeros ordenadores calculaban la raíz cuadrada de a resolviendo por Newton la ecuación $x^2 - a = 0$. Comprueba para varios valores de a (por ejemplo $a = 3, 5, 10, 17$) que tal funciona el método partiendo de la raíz cuadrada entera $(1, 2, 2, 4)$.

9. Los primeros ordenadores calculaban $\frac{1}{a}$ el inverso de un número $a \neq 0$ resolviendo por Newton la ecuación $xa - 1 = 0$. Deduce a mano la forma de p_{n+1} y mira si es adecuada para un ordenador que no sabe dividir. Escribe la ecuación $xa - 1 = 0$ en la forma $a - \frac{1}{x} = 0$ y deduce a mano la forma de p_{n+1} ahora. Comprueba para varios valores de a (por ejemplo $a = 3, 5, 10, 17$) que tal funciona el método dependiendo del valor inicial (prueba para $p_0 > \frac{a}{2}$ y $0 < p_0 < \frac{a}{2}$).

10. Define una función que implemente el método de Newton, dado x_0. Úsala para afinar las soluciones de los apartados 2 y 3. Compara con las soluciones obtenidas.

11. Usa el método de Newton para aproximar la raíz de,

(a) $f(x) = 1 - x^2 - \arctan x$ comenzando con $x_0 = 0.5$ hasta que $|\epsilon_a| < 1\%$.

(b) $f(x) = \cos x - x$, comenzando con $x_0 = 1$ hasta que $|\epsilon_a| < 1\%$

12. Define una función que implemente el método de la secante, dados a y b. Úsala para afinar las soluciones de los apartados 2 y 3. Estudia que método es más rápido en acercarse a la solución.

13. Determinar la raíz real mínima de $f(x) = -11 - 22x + 17x^2 - 2.5x^3$:

    (a) Gráficamente.

    (b) Usando el método de la secante para $|\epsilon_a| < 1\%$

14. Determinar la raíz positiva mínima de $f(x) = 7\,\mathrm{sen}(x)e^{-1} - 1$:

    (a) Gráficamente.

    (b) Usando el método de Newton.

    (c) Usando el método de la secante.

15. Modifica las funciones correspondientes a los métodos de bisección, régula falsi, Newton y secante de forma que se introduzca la tolerancia como entrada y se devuelva la solución dentro de la tolerancia. Modifica de nuevo las funciones para que devuelva también el número de veces que se ejecuta el método.

16. Resuelve las ecuaciones

    $x = \cos(x)$, que tiene una raíz cerca de $x = 0.739085$,

    $x^{20} - 1 = 0$ que tiene una raíz en $x = 1$,

    $\tan(x) - 30 = 0$ que tiene una raíz cerca de $x = 1.537475$,

    $\frac{1}{1+|x|^3} = 0$ que no tiene ninguna raíz,

    $x^2 - 2.2x + 1.21 = 0$ que tiene una raíz en $x = 1.1$

    por los diversos métodos que hemos estudiado compara como se comportan y que tan rápido convergen. ($x^{20} - 1 = 0$ es complicada porque la función es muy plana cerca de la solución y crece muy deprisa fuera; $\tan(x) - 30 = 0$ necesita un buen valor inicial y también crece deprisa a uno de los lados de la solución; la última ecuación tiene una raíz doble).

17. Aplica el método de Newton y de la secante al polinomio $x^3 - 5.56x^2 + 9.1389x - 4.68999$. Dibújalo y observa que tiene una raíz doble cerca de 1.2 y otra simple cerca de 3. Aplícale el método de Newton para raíces múltiples y comprueba la rapidez de convergencia.

18. Usar:

    (a) la iteración de punto fijo y,

    (b) el método de Newton-Raphson

    para determinar la raíz de $f(x) = -0.9x^2 + 1.7x + 2.5$ usando $x_0 = 0$. Efectúe el cálculo hasta que ϵ_a sea menor que 0.01%.

19. El desplazamiento de una estructura está definido por la siguiente ecuación para una oscilación amortiguada:

$$y = 8e^{-kt}\cos\omega t$$

    donde $k = 0.5$ y $\omega = 3$.

(a) Usar el método gráfico para hacer una primera estimación del tiempo requerido para que el desplazamiento disminuya a 4.

(b) Usar el método de Newton-Raphson para determinar la raíz al $\epsilon_a = 0.01\%$.

(c) Usar el método de la secante para determinar la raíz al $\epsilon_a = 0.01\%$.

20. Partiendo de que en un circuito $T_1 + T_2 = \frac{1}{f}$, f es la frecuencia, y C el Ciclo de trabajo, $C = \frac{T_1}{T_1 + T_2} \times 100\%$ demostrar que:

(a) $T_1 = RA \cdot C \cdot \ln(2)$

(b) $T_2 = \frac{RA \cdot RB \cdot C}{RA + RB} \ln\left(|\frac{RA - 2RB}{2RA - RB}|\right)$

(c) Hallar, para $RA = 8670$, $C = 0.01 \times 10^{-6}$ y $T_2 = 1.4 \times 10^{-4}$, T_1, f y el ciclo de trabajo.

(d) Hallar RB.

21. La relación entre la longitud de onda λ y el periodo T de una onda de gravedad que se desplaza sobre la superficie de un cuerpo de agua está dada por la siguiente relación de dispersión:

$$\omega^2 = g\, k\, tanh(k\, h)$$

donde $\omega = \frac{2\pi}{T}$ es la frecuencia angular, g la aceleración de la gravedad (9.810 m/s^2), h la profundidad y $k = \frac{2\pi}{\lambda}$ el número de onda. El forzante que genera la onda determina su periodo T (y por tanto su frecuencia angular). El problema consiste en encontrar la longitud de onda λ que le corresponde a cada profundidad h.

(a) Plantear la forma de resolución de este problema utilizando el método de Newton-Raphson.

(b) Aplicarlo para obtener la longitud de las olas generadas por vientos locales con un periodo de 8 segundos en una zona de 3 metros de profundidad.

(c) Aplicarlo para obtener la longitud de las olas generadas por una marea con un periodo de 12,42 horas y una profundidad de 7 metros.

(d) Comprobar que verifica la solución.

Obtener los resultados con 4 dígitos significativos y expresar los resultados con su error.

CAPÍTULO 4

Sistemas de ecuaciones lineales.

Índice del Tema

Los sistemas lineales de ecuaciones se presentan en muchos problemas de ingeniería. Por ejemplo las leyes de Kirchhoff para circuitos eléctricos establecen que el flujo neto de corriente a través de cada unión de un circuito es cero y que la caída neta de voltaje alrededor de una sección cerrada del circuito es cero. Si suponemos que se aplica un potencial de V volts entre los puntos A y G en el circuito y que v_b, v_c, v_d, v_e y v_f son los potenciales en los puntos B,C,D,E y F respectivamente. Usando G como punto de referencia, las leyes de Kirchhoff implican que estos potenciales satisfacen el siguiente sistema de ecuaciones

$$\begin{array}{rcrcrcrcr}
31v_b & - & 10v_c & & & - & 6v_f & = & 15V, \\
2v_b & - & 8v_c & + & 3v_d & + & 3v_e & = & 0, \\
 & & v_c & - & 3v_d & + & v_e & = & 0, \\
 & & 2v_c & + & 4v_d & - & 7v_e & +v_f = & 0, \\
12v_b & & & + & 15v_d & - & 47v_f & = & 0.
\end{array}$$

Resolver a mano un sistema de 5 ecuaciones con 5 incógnitas no es tarea fácil y en realidad es muy frecuente tener que resolver sistemas de ecuaciones lineales de miles de ecuaciones con miles de incógnitas.

La idea de aplicar la regla de Cramer para resolver sistemas de ecuaciones solamente es posible en la práctica para valores pequeños de n, pues este método involucra tal cantidad de operaciones que una computadora puede tardar años en resolver un sistema de veinte ecuaciones y veinte incógnitas por el método de Cramer. Afortunadamente existen otros métodos más eficientes para obtener la solución de un sistema lineal y a eso dedicaremos este tema. Además, en general la forma más fácil de calcular el determinante o el rango de una matriz es pasarla a forma triangular.

Los **Objetivos** de éste tema son:

- Conocer algunos métodos de resolución de sistemas de ecuaciones lineales: Gauss con y sin pivoteo, factorización LU, métodos iterativos de Jacobi y Gauss-Seidel.
- Ser capaz de utilizar las herramientas que existen en Maxima y los programas que se describen para resolver sistemas lineales.
- Saber utilizar las funciones de Maxima para calcular determinantes, hallar matrices inversas, calcular autovalores y autovectores,...
- Entender el conceptos de matriz diagonalmente dominante y su importancia para la convergencia de los métodos iterativos.

Los métodos numéricos que nos permiten resolver con el uso de computadoras sistemas de ecuaciones de la forma $Ax = B$ pueden ser de dos tipos: directos e iterativos.

A continuación pasamos a estudiar los

1 Métodos directos de resolución de sistemas de ecuaciones lineales

Si una matriz tiene todos los elementos situados por debajo de la diagonal nulos decimos que es *triangular superior* y si tiene todos los elementos situados por encima de la diagonal nulos decimos que es *triangular*

inferior. Por ejemplo la matriz de los coeficientes del sistema

$$\begin{aligned} 6x_1 - 2x_2 &= 12 \\ 12x_2 &= 34 \end{aligned}$$

es triangular superior. Para resolver éste sistema basta empezar por la última ecuación para calcular x_2 y sustituir su valor en la ecuación anterior para hallar x_1. De esta forma resolver un sistema de n ecuaciones con n incógnitas cuya matriz sea triangular es muy sencillo porque basta (para el caso de triangular superior) empezar por la última ecuación para hallar la última incognita, sustituir en la penúltima ecuación para hallar la penúltima incognita, sustituir las incógnitas halladas en la antepenúltima ecuación para hallar la antepenúltima incognita, etc. En definitiva resolver el sistema empezando por abajo. Observa que, haciéndolo de esta forma, cada ecuación sólo tiene una incógnita al resolverla. En el caso de que la matriz sea triangular inferior se resuelve el sistema de arriba abajo empezando por la primera ecuación.

El sistema siguiente tiene la matriz de los coeficientes triangular inferior y se resuelve fácilmente hallando x_1 de la primera ecuación, sustituyendo el valor de x_1 en la segunda y hallando x_2.

$$\begin{aligned} 6x_1 &= 12 \\ 22x_1 + 12x_2 &= 24 \end{aligned}$$

Observa que calcular el determinante de una matriz triangular es muy sencillo, basta multiplicar los elementos de la diagonal principal.

Los métodos directos son aquéllos que calculan la solución del sistema en un número finito de operaciones, pero esta solución puede estar sujeta a los errores de redondeo e inestabilidad, por lo que es muy importante en la elección del método el número de operaciones y la propagación de los errores. Los tipos más comunes de métodos directos son:

i) De eliminación.

ii) De factorización.

Estos métodos se basan en que la resolución de un sistema con matriz triangular es inmediata.

Como método de eliminación estudiamos el método de triangulación de Gauss que se basa en pasar el sistema a uno equivalente cuya matriz de coeficientes sea triangular.

4.1.1 Método de Gauss, pivoteo

Este método transforma el sistema inicial, $Ax = B$, en otro equivalente cuya matriz de coeficientes es triangular superior y la solución se obtiene mediante sustitución hacia atrás. Ya que las soluciones de un sistema de ecuaciones no cambian si sumamos (o restamos) a una ecuación un múltiplo de otra lo que queremos hacer es obtener un sistema equivalente cuya matriz de coeficientes sea triangular. Una vez el sistema esté escrito en forma triangular (pongamos superior) la solución se obtiene resolviendo las ecuaciones empezando desde abajo. Para conseguir obtener un sistema de forma triangular superior, si $a_{11} \neq 0$ hacemos cero los elementos de la primera columna que están por debajo de a_{11} sumando a las demás ecuaciones un múltiplo adecuado de la primera ecuación. Si el nuevo coeficiente $a_{22}^{(2)} \neq 0$ restamos a la tercera, cuarta,... ecuación un múltiplo de la segunda para que los nuevos elementos de la segunda columna que están por debajo de $a_{22}^{(2)}$ sean cero y repetimos el proceso hasta que todos los elementos por debajo de la diagonal

principal sean cero. A los elementos $a_{11}, a_{22}^{(2)}, \ldots$ que vamos usando para hacer cero los elementos de la columna que están abajo se les llama *pivotes*.

Para ilustrar el método resolvemos el sistema de ecuaciones siguiente:

$$\begin{aligned} x_1 - x_2 + 2x_3 &= -8 \\ 2x_1 - 3x_2 + 3x_3 &= -20 \\ x_1 + x_2 + x_3 &= -2 \end{aligned} \tag{4.1.1}$$

El elemento $a_{11} = 1 \neq 0$; como $a_{12} = 2$ restamos 2 veces la primera ecuación de la segunda y la segunda ecuación queda $-x_2 - x_3 = -4$. Restando la primera de la tercera queda $2x_2 - x_3 = 6$ (observa que -2 es $-\frac{a_{12}}{a_{11}}$ y que -1 es $-\frac{a_{13}}{a_{11}}$) asi que un sistema equivalente es

$$\begin{aligned} x_1 - x_2 + 2x_3 &= -8 \\ -x_2 - x_3 &= -4 \\ 2x_2 - x_3 &= 6 \end{aligned}$$

Como ahora el nuevo elemento $a_{22}^{(2)} = -1 \neq 0$ podemos sumarle a la tercera ecuación 2 veces la segunda (observa que 2=$-\frac{a_{23}}{a_{22}}$) para hacer que la tercera ecuación sea $-3x_3 = -2$ y un sistema equivalente es

$$\begin{aligned} x_1 - x_2 + 2x_3 &= -8 \\ -x_2 - x_3 &= -4 \\ -3x_3 &= -2 \end{aligned}$$

Este sistema está en forma triangular superior. De la última ecuación se deduce que $x_3 = \frac{2}{3}$; sustituyendo en la segunda ecuación tenemos que $-x_2 - \frac{2}{3} = -4$ así que $x_2 = \frac{10}{3}$ y sustituyendo en la primera ecuación tenemos que $x_1 - \frac{10}{3} + \frac{4}{3} = -8$ así que $x_1 = -6$.

Destacamos que el procedimiento no funcionará si alguno de los elementos de la diagonal $a_{11}^{(1)}$, $a_{22}^{(2)}$, $a_{33}^{(3)}, \ldots, a_{nn}^{(n)}$ que van apareciendo es cero. Compruébalo aplicando el método al siguiente sistema de ecuaciones

$$\begin{aligned} x_1 - x_2 + 2x_3 &= -8 \\ 2x_1 - 2x_2 + 3x_3 &= -20 \\ x_1 + x_2 + x_3 &= -2 \end{aligned} \tag{4.1.2}$$

Verás que el $a_{22}^{(2)}$ que aparece es cero. ¿Que hacer? Por ejemplo cambia la segunda ecuación de sitio con la tercera.

/* implementamos el metodo de Gauss en Maxima

Escribimos por filas las ecuaciones con los términos independientes, así la ecuación x+2y+3z=5 la escribimos en una lista [1,2,3,5], observa que los tres primeros números corresponden a los coeficientes y el último al término independiente. El sistema de ecuaciones será pues una lista formada por las listas correspondientes a las ecuaciones, por ejemplo el sistema (4.1.1) lo escribiremos [[1,-1,2,-8],[2,-3,3,-20],[1,1,1,-2]], recuerda que el segundo elemento de la lista llamada li se indica por li[2] */

metemos la matriz ampliada en forma de lista en la variable li, en n esta el numero de incognitas */

```
li:[[1,-1,2,-8],[2,-3,3,-20],[1,1,1,-2]]; n:length(li);
```

```
for i:1 thru n do (
for j:i+1 thru n do (
coe:li[j][i]/li[i][i],
for k:i thru n+1 do (li[j][k]:li[j][k]-coe*li[i][k]) ) );
li;
```

Observa que no comprobamos si los sucesivos pivotes son cero.

Observa que utilizamos tres bucles: en el mas interno (con variable k) restamos de cada ecuación que está por debajo la ecuación donde esta el pivote por el factor adecuado para que se haga cero el elemento que esta por debajo del pivote, para ahorrar operaciones guardamos el factor en la variable coe. El segundo bucle (con variable j) recorre las ecuaciones que están por debajo del pivote y el primero (con variable i) recorre las columnas eligiendo los pivotes.

Finalmente presentamos el resultado

Observa también que podemos restar las listas directamente sustituyendo el bucle más interior por la instrucción

```
li[j]:li[j]-coe*li[i] ;
```

el programa quedaría

```
for i:1 thru n do (
for j:i+1 thru n do (
coe:li[j][i]/li[i][i], li[j]:li[j]-coe*li[i]  ) ); */
```

Ten en cuenta que si aplicamos este programa al sistema (4.1.2) da error de dividir por cero y no lo pasa a forma triangular.

Observa que si queremos resolver el sistema lineal Ax=b varias veces con la misma matriz de coeficientes y distintos valores del vector b podemos usar el método de Gauss poniendo a continuación de los coeficientes de cada ecuación los distintos términos independientes. Por ejemplo si queremos resolver los sistemas

$$\begin{array}{rl} x_1 - x_2 &= -8; \\ x_1 + x_2 &= -2; \end{array} \qquad \begin{array}{l} x_1 - x_2 = -10; \\ x_1 + x_2 = 0; \end{array}$$

podemos usar el método de gauss escribiendo las listas [[1,-1,-8,-10], [1,1,-2,0]]. Si el programa esta escrito con tres bucles hay que tener en cuenta el aumento de elementos de las listas en el bucle mas interno (en este caso sería

```
for k:i thru n+2 do (li[j][k]:li[j][k]-coe*li[i][k])
```

para reflejar que hay 2 sistemas en lugar de uno).

Utilizando la eliminación por Gauss se puede calcular el rango de una matriz y su determinante ya que el determinante de una matriz triangular es el producto de los elementos de la diagonal principal y el rango el número de elementos de la diagonal principal distintos de cero.

También podemos usar la eliminación por Gauss para calcular la matriz inversa. Ya que A^{-1} satisface que $AA^{-1} = I$ con I la matriz unidad multiplicando tenemos que la columna i-ésima de A^{-1} satisface el sistema $Ax = b$ con b la columna i-ésima de la matriz unidad. Por tanto para obtener los elementos de A^{-1} podemos resolver los sistemas $Ax = b$ con b las columnas de la matriz unidad correspondiente. Es decir podemos resolver por el método de Gauss n sistemas con coeficientes la matriz A y términos independientes las columnas de la matriz unidad.

Por ejemplo para hallar la inversa de la matriz

$$\begin{pmatrix} 1, & 2, & -1 \\ 2, & 1, & 0 \\ -1, & 1, & 2 \end{pmatrix}$$

podemos resolver los tres sistemas siguientes

$$\begin{array}{llll} x_1 + 2x_2 - x_3 & = 1; & x_1 + 2x_2 - x_3 & = 0; \\ 2x_1 + x_2 & = 0; & 2x_1 + x_2 & = 1; \\ -x_1 + x_2 + 2x_3 & = 0; & -x_1 + x_2 + 2x_3 & = 0; \end{array} \qquad \begin{array}{ll} x_1 + 2x_2 - x_3 & = 0; \\ 2x_1 + x_2 & = 0; \\ -x_1 + x_2 + 2x_3 & = 1; \end{array}$$

y podemos usar el método de gauss usando las listas [[1,2,-1,1,0,0],[2,1,0,0,1,0],[-1,1,2,0,0,1]]; reflejando que estamos resolviendo 3 sistemas de ecuaciones a la vez. Observa que detrás de los coeficientes aparece la matriz unidad.

Ejercicio: Pasa el sistema a forma triangular y resuelve las ecuaciones empezando desde abajo para comprobar si la matriz inversa es

$$\begin{pmatrix} \frac{-2}{9}, & \frac{5}{9}, & \frac{-1}{9} \\ \frac{4}{9}, & \frac{-1}{9}, & \frac{2}{9} \\ \frac{-1}{3}, & \frac{1}{3}, & \frac{1}{3} \end{pmatrix}.$$

Esta forma de calcular la matriz inversa es mucho más eficiente que hacerlo por determinantes excepto para dimensión muy pequeña.

El método de Gauss tal como lo hemos programado es eficiente cuando los elementos de la matriz A y de los sistemas que se van obteniendo sucesivamente no varían mucho en magnitud. Por tanto no siempre es satisfactorio debido a problemas de inestabilidad numérica o mal condicionamiento. Este hecho puede observarse resolviendo el siguiente sistema de ecuaciones,

$$\begin{array}{rl} 0.003x_1 + 59.14x_2 = & 59.17 \\ 5.291x_1 - 6.130x_2 = & 46.78 \end{array}$$

por Gauss tanto en el orden en que están dadas las ecuaciones y como cambiando el orden. Debes hacerlo manualmente porque Maxima procura pasar los números en punto flotante a racionales para hacer los cálculos lo que mejora la precisión. Alternativamente haz fpprec:4; y usa los números en bfloat. Compruébalo haciendo

```
fpprec:4;

bfloat(triangularize(matrix(bfloat([5.291,-6.130,46.78]),bfloat([0.003,59.14,59.17]))));

bfloat(triangularize(matrix(bfloat([0.003,59.14,59.17]),bfloat([5.291,-6.130,46.78]))));
```

y resolviendo el sistema resultante en cada caso. Recuerda que Maxima devuelve la forma triangular superior de una matriz rectangular con la instrucción *triangularize.*

Pivoteo

El algoritmo anterior falla si alguno de los pivotes es cero, además es conveniente elegir como pivote el elemento de la columna de mayor valor absoluto, para hacer esto permutamos las filas. Al algoritmo puede ser: elegimos provisionalmente la fila i-ésima (que corresponde a la usada en el programa anterior), comparamos el valor absoluto de a_{ii} con el del elemento de la siguiente fila $a_{i+1,i}$, si $|a_{ii}| < |a_{i+1,i}|$ permutamos las filas y si no, dejamos la fila como está (no hacemos nada), repetimos este proceso para todas las filas que están por debajo. El programa queda

```
for i:1 thru n do ( /* elección de pivote*/

for j:i+1 thru n do (

if abs(li[i][i])<abs(li[j][i]) then

(filate:li[i],li[i]:li[j],li[j]:filate ) ),

/* fin eleccion de pivote */

for j:i+1 thru n do (

coe:li[j][i]/li[i][i], li[j]:li[j]-coe*li[i]  ) );
```

*/Observa que cuando tenemos que ejecutar varias instrucciones las separamos por , y las metemos en un paréntesis, además usamos una variable temporal (la llamamos filate) para permutar las dos ecuaciones */

Si ejecutas este programa con el sistema (4.1.1) notarás que la primera ecuación del sistema triangular es $2x_1 - 3x_2 + 3x_3 = -20$ que es la que tiene el mayor coeficiente en valor absoluto de x_1 y que evidentemente las soluciones son las mismas.

Ejecuta este programa con el sistema (4.1.2) y verás como lo pasa ahora a forma triangular. Observa que ahora si un pivote es cero es porque toda la columna por debajo de él es cero también y eso se puede aprovechar para estudiar el sistema, modificando el programa.

Aunque hemos presentado el pivoteo por columnas se puede, prestando atención a la posición de las variables en el sistema, elegir como pivote el elemento mayor en valor absoluto de las filas y columnas que quedan por tratar, permutando filas y columnas; éste procedimiento es mejor desde el punto de vista de los errores pero exige muchas comparaciones (del orden de n^2 para elegir el primer pivote, $(n-1)^2$ para el segundo, ...) por lo que es poco práctico. Calcula cuantas comparaciones hacemos en el programa anterior.

4.1.2 Factorización LU. Método de Cholesky

El método de Gauss permite introducir la factorización $A = LU$ para cierta clase de matrices A, donde L es una matriz triangular inferior ("lower" en inglés) y U es una matriz triangular superior ("upper" en inglés). En el caso en que todos los elementos de la diagonal de L sean $L_{ii} = 1$, la factorización que se obtiene recibe el nombre de *método de Doolitte*. Si todos los elementos de la diagonal de U son $U_{ii} = 1$, la factorización que se obtiene recibe el nombre de *método de Crout* y si es $L_{ii} = U_{ii}$ para todo i se dice de *Cholesky.*

La resolución del sistema $Ax = b$ se realiza en tres pasos: primero factorizamos la matriz A, $A = LU$, en segundo lugar resolvemos $Lz = b$ que tiene matriz triangular mediante un algoritmo de bajada empezando por la primera ecuación y finalmente resolvemos $Ux = z$ mediante un algoritmo de subida empezando por la última ecuación.

La factorización LU es especialmente útil si tenemos que resolver varios sistemas de ecuaciones con la misma matriz. Se puede demostrar que, si no hay permutar filas, la matriz U es la que se obtiene por la triangulación de gauss y la matriz L se obtiene poniendo 1 en la diagonal principal y con $m_{i,j}$ el coeficiente cambiado de signo que se usó en el método de gauss para eliminar ese elemento.

Ejercicio: comprueba multiplicando si la factorización LU de la matriz del sistema (4.1.1) está dada por las matrices:

$$\begin{pmatrix} 1, & 0, & 0 \\ 2, & 1, & 0 \\ 1, & -2, & 1 \end{pmatrix} \begin{pmatrix} 1, & -1, & 2 \\ 0, & -1, & -1 \\ 0, & 0, & 3 \end{pmatrix};$$

y compara los elementos de L y U con la aplicación del método de gauss que realizamos entonces. Resuelve el sistema (4.1.1) a partir de la descomposición LU.

En general es posible que haya que cambiar filas para elegir pivotes lo que equivale a que la descomposición LU haya que escribirla en la forma $PA = LU$ donde P es una *matriz de permutación*- que contiene sólo un elemento distinto de cero e igual a uno en cada fila y columna- y que cambia el orden de las filas de A al hacer PA; en cambio al hacer AP se cambia el orden de las columnas de A. Si se tiene que $PA = LU$ para resolver el sistema $Ax = b$ multiplicando por P a la izquierda se tiene que $PAx = Pb$ con lo que calculando $b' = Pb$ se resuelve $LUx = b'$ como explicamos antes.

Ejercicio Comprueba que la matriz

$$P = \begin{pmatrix} 0, & 1, & 0 \\ 1, & 0, & 0 \\ 0, & 0, & 1 \end{pmatrix}$$

cambia las filas primera y segunda de A al hacer PA, además AP cambia las columnas primera y segunda de A con A una matriz cuadrada de dimensión tres. Comprueba que P tiene determinante distinto de cero y que $P^{-1} = P^T$.

En el paquete *linearalgebra* de Maxima con *get_lu_factors(lu_factor(A))* se obtienen la descomposición en factores LU de una matriz A. Recuerda que la matriz debe estar en forma matricial, no como una lista de listas.

Ejercicio: Halla con Maxima la descomposición LU de la matriz de los coeficientes del sistema

$$\begin{aligned} 5x_1 + 2x_2 + x_3 &= 1 \\ 5x_1 - 6x_2 + 2x_3 &= 2 \\ -4x_1 + x_2 + x_3 &= -1 \end{aligned},$$

resuélvelo usando $LUx = b$ y comprueba la solución usando solve.

indicación: carga el paquete con *load(linearalgebra)*

introduce la matriz con *A:matrix([fila1],[fila2],...)*,

haz a continuación *get_lu_factors(lu_factor(A))*,....

Si la matriz es simétrica y definida positiva la factorización $A = LU$ puede tomar la forma $A = LL^T$ es decir la matriz U es la transpuesta de L. Esta factorización se llama de Cholesky. En Maxima con el paquete linearalgebra se obtiene haciendo *cholesky(matriz)*

Ejercicio: Halla con Maxima la descomposición de Cholesky de la matriz

$$\begin{pmatrix} 4, & -1, & 1 \\ -1, & 5, & 2 \\ 1, & 2, & 3 \end{pmatrix}.$$

Comprueba que es LL^T. Indicación: escribe la matriz en *A*, haz *B:cholesky(A)* y luego *B.transpose(B)*.

Lee en la ayuda de Maxima que más puedes hacer hacer con el paquete linearalgebra.

Hay veces que la matriz del sistema $Ax = B$ es excesivamente grande y la aplicación de métodos directos puede llegar a ser incluso imposible por el gran número de operaciones, lo que incrementa los errores de redondeo y la excesiva ocupación de la memoria del ordenador. Un ejemplo de lo complejas que pueden ser las matrices correspondientes a los sistemas, puede ser resolver las ecuaciones correspondientes a la previsión de tiempo atmosférico en España con una cuadrícula de 1km -interesante porque previene fenómenos locales tipo gota fría-; como la superficie de la península es del orden de 500.000 km^2, que estudiar por ejemplo la temperatura, presión, humedad en unos 500.000 puntos. Esto da origen a $1.500.000$ variables y un sistema lineal con éstas variables tendría matrices con $1.500.000^2 = 2.25\,10^{12}$ elementos (afortunadamente muchos de ellos cero ya que puntos geográficamente lejanos se influyen poco en un instante dado). De hecho muchos de los ordenadores más potentes del mundo se dedican a predecir el tiempo. Por lo anterior es conveniente estudiar otros métodos: los métodos iterativos.

2 Métodos iterativos

Los métodos iterativos para la resolución numérica de sistemas lineales surgen por la necesidad de resolver este tipo de problemas cuando la matriz del sistema es muy grande y cuesta mucho aplicar un método directo, debido al gran número de operaciones y la excesiva ocupación de memoria del ordenador.

La mayoría de las técnicas iterativas sustituyen el sistema $Ax = B$ por otro sistema equivalente de la forma $x = Tx + C$ y, a partir de un vector inicial cualquiera, generan mediante iteraciones una sucesión de vectores que son soluciones aproximadas. Estos métodos, ya que pueden ser más rápidos, son preferibles a los directos cuando la matriz de los coeficiente tiene muchos ceros o es muy grande. También pueden ser más económicos en cuanto a requisitos de memoria de una computadora. Además en cálculos manuales poseen la ventaja, en caso de que se cometa algún error, de que son autocorregibles.

La base fundamental de los métodos iterativos para resolver un sistema lineal $Ax = B$ es la siguiente: se empieza por una aproximación inicial $x^{(0)}$ de la solución x, y generamos una sucesión $\{x^{(k)}\}_{k=0}^{\infty}$ que converge a x. Convertimos el sistema $Ax = B$ en otro sistema de la forma $x = Cx + D$. Una vez seleccionado la aproximación inicial, la sucesión de soluciones aproximadas se genera calculando

$$x^{(k)} = Cx^{(k-1)} + D$$

para cada $k = 1, 2, 3, \ldots$. Lo importante es garantizar que la sucesión $x^{(k)}$ converja a la solución del sistema. Si indicamos con $||A|| = max_{||x||=1}||Ax||$ la norma de una matriz asociada a una norma en los vectores $||x||$, se puede probar que $||Ax|| \leq ||A||\,||x||$ y si la norma de la matriz C es menor estricto que 1, la sucesión

$x^{(k)}$ converge independientemente del valor inicial $x^{(0)}$ para esa norma vectorial. Esto es importante porque entonces la convergencia no depende del valor inicial y los errores de redondeo que aparezcan afectan poco al resultado.

Como caso particular de los métodos iterativos, introducimos el método de Jacobi y el método de Gauss-Seidel.

Se descompone la matriz A en la suma de tres matrices $A = L + D + U$, donde D es diagonal, L triangular inferior y U triangular superior, formadas con los elementos respectivos de A.

4.2.1 Método de Jacobi

En el método de Jacobi las soluciones aproximadas del sistema $Ax = B$ se obtienen de la expresión

$$Dx^{(k)} = (-L - U)x^{(k-1)} + B.$$

Esto corresponde a despejar x_1 de la primera ecuación, x_2 de la segunda, etc..., por ejemplo para resolver el sistema

$$\begin{aligned} 5x_1 + 2x_2 + x_3 &= 1, \\ 3x_1 - 6x_2 + 2x_3 &= 2, \\ -x_1 + x_2 + 4x_3 &= -1, \end{aligned} \tag{4.2.1}$$

lo escribimos como

$$\begin{aligned} x_1 &= \tfrac{1}{5}(1 - 2x_2 - x_3), \\ x_2 &= \tfrac{-1}{6}(-3x_1 - 2x_3 + 2), \\ x_3 &= \tfrac{1}{4}(x_1 - x_2 - 1) \end{aligned}$$

y la iteración de Jacobi está dada por

$$\begin{aligned} x_1^{(k+1)} &= \tfrac{1}{5}(1 - 2x_2^{(k)} - x_3^{(k)}), \\ x_2^{(k+1)} &= \tfrac{-1}{6}(-3x_1^{(k)} - 2x_3^{(k)} + 2), \\ x_3^{(k+1)} &= \tfrac{1}{4}(x_1^{(k)} - x_2^{(k)} - 1) \end{aligned}$$

Si elegimos $x^{(0)} = (0, 0, 0)$; se obtiene que $x^{(1)} = (\frac{1}{5}, \frac{-1}{3}, \frac{-1}{4})$; $x^{(2)} = (\frac{23}{60}, \frac{-31}{90}, \frac{-7}{60})$; $x^{(3)} = (\frac{13}{36}, \frac{-277}{1080}, \frac{-49}{720})$; $x^{(4)} = (\frac{683}{2160}, \frac{-193}{810}, \frac{-413}{4320})$;... que converge a la solución $x_1 = \frac{50}{161}, x_2 = \frac{-5}{23}, x_3 = \frac{-19}{161}$.

Ejercicio: Programa en Maxima la iteración de Jacobi de este sistema, y comprueba si los resultados anteriores son correctos. Resuelve el sistema con solve, obtén los resultados en punto flotante, repite la iteración 10 veces y comprueba si convergen a la solución. Arranca la iteración a partir de otro valor inicial, por ejemplo $x^{(0)} = (1, 1, 1)$ y comprueba si la iteración también converge a la solución.

Indicación: para hacerlo con listas prueba algo así como

```
jacobi(arg):=block([x1:arg[1],x2:arg[2],x3:arg[3]],
[(1-2*x2-x3)/5,-(-3*x1-2*x2+2)/6,(x1-x2-1)/4] );
```

ini:[0,0,0]; for i:1 thru 10 do (print(ini),ini:jacobi(ini));

Observa que si algún elemento de la diagonal es cero el método de Jacobi no puede aplicarse tal cual. En este caso cambiamos de orden las ecuaciones del sistema. Por ejemplo, para aplicar el método de Jacobi para resolver el sistema

$$\begin{aligned} 2x_2 + x_3 &= 1, \\ 3x_1 - x_2 + x_3 &= 2, \\ -x_1 + x_2 + 4x_3 &= -1, \end{aligned}$$

podemos escribirlo en varias formas, por ejemplo

$$\begin{aligned} 3x_1 - x_2 + x_3 &= 2, \\ 2x_2 + x_3 &= 1, \\ -x_1 + x_2 + 4x_3 &= -1 \end{aligned} \quad o\ bien \quad \begin{aligned} -x_1 + x_2 + 4x_3 &= -1 \\ 2x_2 + x_3 &= 1, \\ 3x_1 - x_2 + x_3 &= 2, \end{aligned} \tag{4.2.2}$$

Ejercicio: Aplica el método de Jacobi a estas ecuaciones, resuelve el sistema y observa si converge a la solución o no.

4.2.2 Método de Gauss-Seidel

En el método de Gauss-Seidel las soluciones aproximadas se obtienen de la expresión

$$(L + D)x^{(k)} = (-U)x^{(k-1)} + B.$$

para cada $k = 1, 2, 3, \ldots$. Esto corresponde a despejar x_1 de la primera ecuación, x_2 de la segunda, etc. *pero usando los nuevos valores tan pronto como se obtengan* por ejemplo para resolver el sistema (4.2.1) igual que para Jacobi hacemos

$$\begin{aligned} x_1 &= \tfrac{1}{5}(1 - 2x_2 - x_3), \\ x_2 &= \tfrac{-1}{6}(-3x_1 - 2x_3 + 2), \\ x_3 &= \tfrac{1}{4}(x_1 - x_2 - 1) \end{aligned}$$

y la iteración de Gauss Seidel está dada por

$$\begin{aligned} x_1^{(k+1)} &= \tfrac{1}{5}(1 - 2x_2^{(k)} - x_3^{(k)}), \\ x_2^{(k+1)} &= \tfrac{-1}{6}(-3x_1^{(k+1)} - 2x_3^{(k)} + 2), \\ x_3^{(k+1)} &= \tfrac{1}{4}(x_1^{(k+1)} - x_2^{(k+1)} - 1) \end{aligned}$$

Observa que en el cálculo de $x_2^{(k+1)}$ usamos $x_1^{(k+1)}$ que ya está disponible y en el cálculo de $x_3^{(k+1)}$ usamos $x_1^{(k+1)}, x_2^{(k+1)}$.

Si elegimos $x^{(0)} = (0, 0, 0)$; se obtiene ahora que $x^{(1)} = (\frac{1}{5}, \frac{-7}{30}, \frac{-17}{120})$; $x^{(2)} = (\frac{193}{600}, \frac{-791}{3600}, \frac{-1651}{14400})$; $x^{(3)} = (\frac{22379}{72000}, \frac{-93373}{432000}, \frac{-204353}{1728000})$; ... que converge a la solución

Ejercicio: Programa en Maxima la iteración de Gauss Seidel de este sistema, y comprueba que los resultados anteriores son correctos, repite la iteración hasta 10 veces. Resuelve el sistema con solve, obtén los resultados en punto flotante y comprueba si la iteración converge a la solución. Arranca la iteración a partir de otro valor inicial, por ejemplo $x^{(0)} = (1, 1, 1)$ y comprueba si la iteración también converge a la solución.

Indicación: para hacerlo con listas prueba algo así como

```
gaussei(arg):=block([x1:arg[1],x2:arg[2],x3:arg[3],xn1,xn2,xn3],
xn1:(1-2*x2-x3)/5,xn2:-(-3*xn1-2*x2+2)/6,xn3:(xn1-xn2-1)/4,
[xn1,nx2,xn3] );
ini:[0,0,0]; for i:1 thru 10 do (print(ini),ini:gaussei(ini));
```

Igual que ocurre con el método de Jacobi si algún elemento de la diagonal es cero el método de Gauss Seidel no puede aplicarse tal cual. En este caso también cambiamos de orden las ecuaciones del sistema.

¿Cuando podemos garantizar que el método de Jacobi o de Gauss - Seidel converge a la solución para cualquier valor inicial? Existen varias condiciones que lo garantizan, la más sencilla es que la matriz $A = (a_{ij})$ del sistema sea *estrictamente diagonalmente dominante* lo que ocurre cuando $|a_{ii}| > \sum_{j \neq i} |a_{ij}|$ para todo i es decir si el valor absoluto de cada elemento de la diagonal es mayor que la suma de los valores absolutos de los demás elementos de su fila.

Por ejemplo la matriz del sistema (4.2.1) es diagonalmente dominante estrictamente porque en la primera fila $5 > 2 + 1$, en la segunda $6 > 3 + 2$ y en la tercera $4 > 1 + 1$. La matriz del primer sistema que aparece en (4.2.2) es diagonalmente dominante estrictamente porque $3 > 1 + 1$, $2 > 1$ y $4 > 1 + 1$ pero la matriz del segundo sistema NO lo es porque en la primera ecuación $1 < 1 + 4$ y en la tercera $1 < 1 + 3$, por tanto ni el método de Jacobi ni el de Gauss Seidel tienen por que converger para el segundo sistema de (4.2.2), sin embargo ambos métodos deben converger para el primer sistema de (4.2.2) y el sistema (4.2.1) (si no nos equivocamos al programar o al escribir las ecuaciones, claro).

Así pues para resolver un sistema por Jacobi o Gauss Seidel con garantías de convergencia es conveniente ponerlo en forma estrictamente diagonalmente dominante, cambiando de posición las ecuaciones.

Ejercicio: Escribe en forma diagonalmente dominante estrictamente el sistema

$$\begin{aligned} 3x_1 + 3x_2 + 7x_3 &= 1, \\ 3x_1 + 6x_2 + 2x_3 &= 2, \\ 3x_1 - x_2 + x_3 &= -1, \end{aligned}$$

y aplica 5 iteraciones por Jacobi y Gauss-Seidel a ver si convergen a la solución antes y después de escribirlo en forma diagonalmente dominante estrictamente.

Se puede probar también que si un sistema tiene la matriz de coeficientes estrictamente diagonalmente dominante se puede realizar el método de Gauss sin tener que permutar filas.

Como en cualquier método iterativo al resolver un sistema por los métodos de Jacobi o Gauss-Seidel tenemos que plantearnos cuantas iteraciones realizar y que tan cerca de la solución están las sucesivas aproximaciones.

Ejercicio: Plantea un programa en Maxima que haga a lo más un número de iteraciones $nmax$ por Jacobi y que utilice un test de parada, por ejemplo que $\frac{||x^n - x^{n-1}||}{||x^n||} < err$, (Indicación: hicimos algo parecido con los métodos de bisección o Newton.)

3 Ejercicios

1. Convierte a forma triangular por el método de Gauss y resuelve después los sistemas siguientes:

$$A).\quad \begin{array}{rl} 2x-y & =1;\\ -x+2y-z & =0;\\ -y+2z & =2; \end{array} \qquad B).\quad \begin{array}{rl} 4x+y+z+t & =1;\\ x+3y-z & =2;\\ x-y+2z & =3;\\ x+y+2t & =5; \end{array} \qquad C).\quad \begin{array}{rl} 6x+2y+z-t & =0\\ x+5y+z-t & =2;\\ x+y+4z-t & =0;\\ -x-z+3t & =10; \end{array}$$

Comprueba la solución.

2. Convierte a forma triangular por el método de Gauss usando pivotes y resuelve después los sistemas siguientes:

$$A).\quad \begin{array}{rl} -y+2z & =2;\\ -x+2y-z & =0;\\ 2x-y & =1; \end{array} \qquad B).\quad \begin{array}{rl} x+y+2t & =5;\\ x-y+2z & =3;\\ x+3y-z & =2;\\ 4x+y+z+t & =1; \end{array} \qquad C).\quad \begin{array}{rl} -x-z+3t & =10;\\ x+y+4z-t & =0;\\ x+5y+z-t & =2;\\ 6x+2y+z-t & =0 \end{array}$$

Comprueba la solución. Halla la inversa de la matriz de los coeficientes del sistema B). anterior resolviendo los cuatro sistemas obtenidos poniendo como término independiente las columnas de la matriz unidad. Comprueba que es la inversa.

3. Pon en forma adecuada del sistema A). del problema anterior y aplica 5 iteraciones por los métodos de Jacobi y Gauss Seidel a partir del vector nulo (0,0,0). Comprueba que tan cerca de la solución estás. Haz lo mismo para el sistema C).

4. Considera la matriz de Hilbert A dada por $a_{i,j} = \frac{1}{i+j-1}$ de orden 10 (en Maxima, en el paquete *linearalgebra*, se puede generar con la instrucción A:*hilbert_matrix(10)*, también puedes hacer h[i,j]:=1/(1+i+j); A:*genmatrix*(h,10,10);).

   Resuelve el sistema $Ax = b$ donde $b_1 = 1, b_j = 0$ si $j \neq 1$. (En Maxima puedes crear la lista de términos independientes con B:create_list(if i=1 then 1 else 0,i,1,10); y después hacer invert(A).B;). Para analizar como se afectan las soluciones del sistema por los errores en b resuelve el sistema $Ax = b$ donde $b_1 = 1.001, b_j = 0$ si $j \neq 1$. Compara los resultados y observa como un error relativo de 10^{-3} en B se transmite a las soluciones del sistema. Simula trabajar con 4 dígitos usando bfloat con 4 dígitos, por ejemplo en Maxima haz fpprec:4; bfloat(invert(bfloat(A)).B; repite los cálculos anteriores y observa si las diferencias son muy grandes. La razón es que una matriz de Hilbert con orden grande esta muy mal condicionada y por tanto resolver un sistema con ella como matriz de coeficientes es muy sensible a los errores.

5. Resuelve el sistema siguiente, que ya está dado en forma triangular y tiene solución exacta $x_1 = x_2 = x_3 = x_4 = 1$,

$$\begin{array}{ll} 0.9143 * 10^{-4} x_1 & = 0.9143 * 10^{-4},\\ 0.8762 x_1 + 0.7156 * 10^{-4} x_2 & = 0.87627156,\\ 0.7943 x_1 + 0.8143 x_2 + 0.9504 * 10^{-4} x_3 & = 1.60869504,\\ 0.8017 x_1 + 0.6123 x_2 + 0.7165 x_3 + 0.7123 * 10^{-4} x_4 & = 2.13057123, \end{array}$$

   directamente, y también usando bfloat con $fpprec : 4;$ y $fpprec : 8;$ para simular trabajar con 4 dígitos y con 8 dígitos y compara los resultados.

CAPÍTULO 5

Interpolación y aproximación.

Índice del Tema

Si tenemos una serie de puntos obtenidos por ejemplo de un experimento podemos plantearnos buscar funciones, por ejemplo polinomios, que pasen exactamente por esos puntos, a estas funciones se dice que *interpolan* los puntos y sirven para aproximar los valores de experimento en puntos intermedios.

Otra alternativa es suponer que los puntos tienen errores y buscar una función de una forma dada que sea la que más se aproxime a estos puntos, este enfoque da origen a la *aproximación por mínimos cuadrados*, por ejemplo la recta de regresión.

Los **Objetivos** de éste tema son:

- Separar el problema de *interpolar* un conjunto de puntos del de *aproximar* un conjunto de puntos por una función.
- Ver las ventajas e inconvenientes de la interpolación de Lagrange frente a las funciones ranura.
- Ser capaz de calcular el polinomio de interpolación de Lagrange y la interpolación por funciones ranura de un conjunto de puntos, a través de las herramientas que existen en Maxima y los programas que se describen.
- Entender el concepto de aproximación por mínimos cuadrados. Ser capaz de utilizar las herramientas que existen en Maxima y los programas que se describen para ajustar funciones sencillas por mínimos cuadrados.

1 Polinomio de interpolación.

Tenemos una colección de puntos y queremos hallar un polinomio del menor grado posible que pase por ellos, a éste polinomio se le llama *polinomio de interpolación de Lagrange*; por ejemplo es conocido que la recta que pasa por dos puntos $(x_1, y_1), (x_2, y_2)$ está dada por $y = \frac{y_2-y_1}{x_2-x_1}(x - x_1)$. Si tenemos 3 puntos el polinomio de menor grado sería un polinomio de grado 2 cuya fórmula es $y = a_2x^2 + a_1x + a_0$ y podemos plantear resolver el sistema $a_2x_1^2 + a_1x_1 + a_0 = y_1$, $a_2x_2^2 + a_1x_2 + a_0 = y_2$, $a_2x_3^2 + a_1x_3 + a_0 = y_3$, este es un sistema lineal de 3 ecuaciones con 3 incógnitas que permite hallar a_0, a_1, a_2 y obtener el polinomio que pasa por los 3 puntos $(x_1, y_1), (x_2, y_2), (x_3, y_3)$.

Si tenemos n puntos distintos $(x_1, y_1), (x_2, y_2), ...(x_n, y_n)$ podemos encontrar un polinomio de grado $n-1$, $y = a_{n-1}x^{n-1} + ...a_2x^2 + a_1x + a_0$ resolviendo el sistema lineal

$$\begin{array}{ll} a_{n-1}x_1^{n-1} + ... + a_1x_1 + a_0 = & y_1, \\ a_{n-1}x_2^{n-1} + ... + a_1x_2 + a_0 = & y_2, \\ ... & \\ a_{n-1}x_{n-1}^{n-1} + ... + a_1x_{n-1} + a_0 = & y_{n-1}, \end{array}$$

para hallar los coeficientes $a_0, a_1, ...a_{n-1}$. Se puede probar que el sistema admite solución única si los puntos son distintos y para cada valor de x hay un único valor de y.

Ejercicio: Halla el polinomio de interpolación que pasa por los puntos $(0, 1), (1, 2), (2, 5)$ resolviendo el sistema. Comprueba que el polinomio obtenido pasa realmente por los puntos. (indicación: como hay 3 puntos debemos buscar un polinomio de grado 2 es decir de la forma $y = ax^2 + bx + c$).

El polinomio de interpolación de Lagrange se puede obtener también sin resolver un sistema. Observa que el producto $(x-x_2)(x-x_3)...(x-x_n)$ vale cero cuando $x=x_2, x=x_3, ..., x=x_n$ y que $\frac{(x-x_2)...(x-x_n)}{(x_1-x_2)...(x_1-x_n)}$ valdrá cero cuando $x=x_2, ...x=x_n$ y vale 1 cuando $x=x_1$. Entonces, la expresión

$$y_1\frac{(x-x_2)...(x-x_n)}{(x_1-x_2)...(x_1-x_n)}+y_2\frac{(x-x_1)(x-x_3)...(x-x_n)}{(x_2-x_1)(x_2-x_3)...(x_2-x_n)}+...+y_n\frac{(x-x_1)(x-x_3)...(x-x_{n-1})}{(x_n-x_1)(x_n-x_3)...(x_n-x_{n-1})} \quad (5.1.1)$$

es un polinomio de grado $n-1$ porque todos los factores lo son y vale y_1 cuando $x=x_1$, y_2 cuando $x=x_2$,... , y_n cuando $x=x_n$ por lo que nos da el polinomio de interpolación de Lagrange que pasa por esos puntos. Observa también que en el primer sumando falta el factor $(x-x_1)$ arriba y el factor (x_1-x_1) abajo, en el segundo falta $(x-x_2)$ ... y en el último falta $(x-x_n)$. Usando $\sum$ para indicar suma y $\prod$ para indicar producto la expresión anterior se puede escribir de forma simplificada como $\sum_{i=1}^{n} y_i \prod_{j=1,j\neq i}^{n}\frac{x-x_j}{x_i-x_j}$.

Ejercicio: Calcula el polinomio de interpolación de lagrange directamente simplificando en Maxima la expresión (5.1.1) para los puntos $(0,1),(1,2),(2,5)$ y comprueba si da el mismo resultado. Indicación: simplifica algo de la forma $1*\frac{(x-1)(x-2)}{(0-1)(0-2)}+2*\frac{(x-0)(x-2)}{(1-0)(1-2)}+5*\frac{(x-0)(x-1)}{(2-0)(2-1)}$.

Ejercicio: Haz un programa en Maxima que calcule el polinomio de interpolación. Indicación: observa que delete delete(li[i],li); quita el i-esimo elemento de una lista y prueba algo así como

/*polinomio de interpolación de Lagrange que pasa por (x1,y1), (x2,y2), etc si escribimos los puntos como lista x, lista y es decir

lix:[x1,x2,x3,x4,x5]; liy:[y1,y2,y3,y4,y5]; */

interpol(lix,liy):=

sum(liy[j]*product(ev((x-delete(lix[j],lix)[i])/(lix[j]-delete(lix[j],lix)[i])),i,1,length(lix)-1), j,1,length(lix));

Ejercicio calcula el polinomio de interpolación que pasa por los puntos $(0,sen(0)),(1,sen(1)),....(10,sen(10))$. Dibújalo.

Existe una forma sencilla de calcular el polinomio de interpolación a través de *diferencias divididas* para ello escribimos los puntos en dos columnas x_i y y_i en la tercera columna escribimos $y_{i+1}^{(1)}=\frac{y_{i+1}-y_i}{x_{i+1}-x_i}$, en la cuarta $y_{i+2}^{(2)}=\frac{y_{i+1}^{(1)}-y_i^{(1)}}{x_{i+2}-x_i}$, en la quinta $y_{i+3}^{(3)}=\frac{y_{i+1}^{(2)}-y_i^{(2)}}{x_{i+3}-x_i}$, y así sucesivamente queda una disposición triangular de números.

Por ejemplo las diferencias divididas que corresponden a los puntos $(1,1),(2,3),(4,7),(6,9)$ se obtienen de la siguiente forma (entre paréntesis ponemos los cálculos)

$$\begin{array}{llll} x & y & y^{(1)} & y^{(2)} & y^{(3)} \\ 1, & 1, & & & \\ 2, & 3, & 2=(\frac{3-1}{2-1}), & & \\ 4, & 7, & 2=(\frac{7-3}{4-2}), & 0=(\frac{2-2}{4-1}), & \\ 6, & 9, & 1=(\frac{9-7}{6-4}), & \frac{-1}{4}=(\frac{1-2}{6-2}), & \frac{-1}{20}=(\frac{\frac{-1}{4}-0}{6-1}), \end{array}$$

Ahora el polinomio de interpolación es $y_1+y_2^{(1)}(x-x_1)+y_3^{(2)}(x-x_1)(x-x_2)+y_4^{(3)}(x-x_1)(x-x_2)(x-x_3)+...$ Observa que los coeficientes del polinomio son los últimos elementos de cada fila de las diferencias divididas. En nuestro caso el polinomio de interpolación es $1+2(x-1)+0(x-1)(x-2)-\frac{1}{20}(x-1)(x-2)(x-4)$.

Ejercicio: Comprueba los cálculos y si coincide con el polinomio de interpolación que pasa por los puntos $(1,1),(2,3),(4,7),(6,9)$.

En Maxima el paquete *interpol* permite calcular el polinomio de interpolación de Lagrange a través de la función *lagrange*.

Por ejemplo haz:

load("interpol"); pun:[[0,1],[1,2],[2,5]];lagrange(pun);

El polinomio de interpolación tiene el problema de que, si hay muchos puntos es de grado muy alto y *oscila mucho* por ejemplo: Claramente el polinomio $p(x) = \prod_{i=-3}^{3}(x-i) = (x-3)(x-2)(x-1)x(x+1)(x+2)(x+3)$ toma el valor cero en $x = -3, -2, -1, 0, 1, 2, 3$ y puede servir como polinomio que interpola estos puntos, si lo dibujamos en Maxima con la instrucción

plot2d(product(x-i,i,-3,3),[x,-3,3]);

observamos que es mas o menos plano cerca del cero (en medio de donde están los puntos que interpola) pero cerca de los extremos del intervalo toma valores muy grandes $\simeq 95$ lo que no resulta aceptable, por tanto no es razonable interpolar muchos puntos porque el polinomio de interpolación oscila mucho. Existe una alternativa: no tomar los puntos igualmente espaciados sino elegir más puntos cerca de los extremos y menos en el centro del intervalo (Si estás interesado investiga que relación tiene los polinomios de Chebyshev con dónde elegir los puntos a interpolar para minimizar el error).

La alternativa si hay muchos puntos es interpolar con funciones que estén definidas a trozos.

2 Interpolación por funciones ranura

Igual que en la sección anterior tenemos una familia de puntos $(x_1, y_1), (x_2, y_2), ...(x_n, y_n)$, con $x_1 < x_2 < x_3 < ... < x_n$ y deseamos hallar una función sencilla que pase por ellos, pero que no tiene por qué estar definida de igual forma en todo el intervalo $[x_1, x_n]$. A esta función de interpolación se llama *interpolación por funciones ranura* (spline en inglés).

La primera opción es definir las funciones constantes en cada intervalo

$$\begin{array}{ll} f(x) = y_1, & x \in [x_1, x2), \\ f(x) = y_2, & x \in [x_2, x3), \\ \quad ... & \\ f(x) = y_{n-1}, & x \in [x_{n-1}, x_n]. \end{array}$$

Esta forma tiene el inconveniente de que la función f no es ni siquiera continua.

Si tenemos los puntos $(1,1), (2,3), (4,7), (6,9)$, la gráfica de la función será en Maxima:

plot2d(if (x>=1 and x<2) then 1 else (if x<4 then 3 else (if x<6 then 7)),[x,1,6],[y,0,10]);

La siguiente opción es elegir en cada intervalo el polinomio de interpolación de grado 1 que pasa por los extremos, es decir

$$\begin{array}{ll} f(x) = y_1 + \frac{y_2-y_1}{x_2-x_1}(x-x_1), & x \in [x_1, x2), \\ f(x) = y_2 + \frac{y_3-y_2}{x_3-x_2}(x-x_2), & x \in [x_2, x3), \\ \quad ... & \\ f(x) = y_{n-1} + \frac{y_n-y_{n-1}}{x_n-x_{n-1}}(x-x_{n-1}), & x \in [x_{n-1}, x_n]. \end{array}$$

La función de interpolación es ahora continua pero puede dejar de ser derivable en los puntos que interpolamos.

Si tenemos los puntos $(1,1),(2,3),(4,7),(6,9)$, la gráfica de la función será en Maxima:

plot2d(if (x>=1 and x<2) then 1+(3-1)/(2-1)*(x-1) else (if x<4 then 3+(7-3)/(4-2)*(x-2) else (if x<6 then 7+(9-7)/(6-4)*(x-4))),[x,1,6],[y,0,10]);

En Maxima la instrucción *linearinterpol* del paquete *interpol* calcula la interpolación por funciones ranura lineales.

Si pedimos que la función sea en cada intervalo *un polinomio de grado 2*, en cada intervalo será de la forma $a_ix^2+b_ix+c_i$, por lo que tiene tres incógnitas a determinar que son a_i,b_i,c_i como hay que imponer que pase por (x_i,y_i) y por (x_{i+1},y_{i+1}) hay dos ecuaciones y nos queda la opción de pedir que la derivada coincida en x_{i+1} con la del polinomio del siguiente intervalo. De esta forma la función general $f(x)$ tendrá derivada continua en el intervalo $[x_1,x_n]$.

Por ejemplo, supongamos que tenemos los puntos $(1,1),(2,3),(4,7),(6,9)$, vamos a calcular la función. En primer lugar podemos poner cada polinomio $a_ix^2+b_ix+c_i$, en la forma $a_i(x-x_i)^2+b_i(x-x_i)+y_i$, de esta forma garantizamos que pase por el punto de la izquierda y nos ahorramos una incógnita.

Por tanto tenemos que hallar: el polinomio $a_1(x-1)^2+b_1(x-1)+1$ definido en [1,2]; el polinomio $a_2(x-2)^2+b_2(x-2)+3$ definido en [2,4]; y el polinomio $a_3(x-4)^2+b_3(x-4)+7$ definido en [4,6].

Estos polinomios deben cumplir que cada uno pase por el punto de la derecha del intervalo es decir $a_1(2-1)^2+b_1(2-1)+1=3$; $a_2(4-2)^2+b_2(4-2)+3=7$; y que $a_3(6-4)^2+b_3(6-4)+7=9$ y además que en los puntos (2,3) y (4,7) coincidan las derivadas. Como $(a_i(x-x_i)^2+b_i(x-x_i)+c_i)'=2a_i(x-x_i)+b_i$ esta condición da origen a las ecuaciones $2a_1(2-1)+b_1=b_2$; $2a_2(4-2)+b_2=b_3$; tenemos 6 incógnitas y 5 ecuaciones por lo que podemos imponer una condición todavía, por ejemplo un valor de la derivada en uno de los extremos. Ahora resolvemos el sistema.

Ejercicio: Impón la condición adicional que $f'(6)=0$, resuelve el sistema con Maxima y dibuja la función.

La interpolación por funciones ranura de orden 2 tiene el problema de que sólo hay continuidad en la derivada, en muchas aplicaciones esto no es suficiente (por ejemplo al tomar una curva con un automóvil el giro del volante depende de la derivada segunda de la trayectoria y se notaría si hay discontinuidades en la derivada segunda de función que sigue la carretera). Además puede ser conveniente poder especificar los valores de la derivada ó de la derivada segunda en los extremos. Por ello lo más usual es utilizar como funciones ranura polinomios de orden 3 también llamados *trazadores cúbicos*.

Ahora usamos las funciones:

$$\begin{array}{ll} p_1(x)=a_1(x-x_1)^3+b_1(x-x_1)^2+c_1(x-x_1)+y_1, & x\in[x_1,x2),\\ p_(x)=a_2(x-x_2)^3+b_2(x-x_2)^2+c_2(x-x_2)+y_2, & x\in[x_2,x3),\\ \quad\ldots & \\ p_{n-1}(x)=a_{n-1}(x-x_{n-1})^3+b_{n-1}(x-x_{n-1})^2+c_{n-1}(x-x_{n-1})+y_{n-1}, & x\in[x_{n-1},x_n]. \end{array}$$

Tenemos $3(n-1)$ incógnitas y las siguientes condiciones:

- Que cada función pase por el punto de la derecha: $p_1(x_2)=y_2$, $p_2(x_3)=y_3$, ... $p_{n-1}(x_n)=y_n$; que son $n-1$ condiciones.
- Que las primeras derivadas coincidan en los puntos intermedios: $p_1'(x_2)=p_2'(x_2)$; $p_2'(x_3)=p_3'(x_3)$;

... $p'_{n-2}(x_{n-1}) = p'_{n-1}(x_{n-1})$; que son $n-2$ condiciones

- Que las segundas derivadas coincidan en los puntos intermedios: $p''_1(x_2) = p''_2(x_2)$; $p''_2(x_3) = p''_3(x_3)$; ... $p''_{n-2}(x_{n-1}) = p''_{n-1}(x_{n-1})$; que son $n-2$ condiciones

Por tanto quedan dos condiciones libre y podemos imponer cuanto vale la derivada (o la derivada segunda) en los extremos x_1, x_n. Además la función de interpolación tendrá la segunda derivada continua en todo el intervalo.

Si imponemos que $f''(x_1) = 0, f''(x_n) = 0$ se dice que tenemos interpolación con *frontera libre o natural*, si no, imponemos los valores de $f'(x_1)$ y de $f'(x_n)$. Se puede probar que el sistema de ecuaciones necesario para hallar a_i, b_i, c_i puede ponerse en forma estrictamente diagonalmente dominante con lo que puede obtenerse la solución por iteración.

Por ejemplo, para los puntos $(1,1), (2,3), (4,7), (6,9)$ y condiciones de frontera natural buscamos polinomios $p_1(x) = a_1(x-1)^3 + b_1(x-1)^2 + c_1(x-1) + 1$ en el intervalo [1,2), $p_2(x) = a_2(x-2)^3 + b_2(x-2)^2 + c_2(x-2) + 3$ en el intervalo [2,3) y el polinomio $p_3(x) = a_3(x-4)^3 + b_3(x-4)^2 + c_3(x-4) + 7$ en el intervalo [4,6].

Las condiciones de continuidad son $a_1(2-1)^3 + b_1(2-1)^2 + c_1(2-1) + 1 = 3$, $a_2(4-2)^3 + b_2(4-2)^2 + c_2(4-2) + 3 = 7$, y $a_3(6-4)^3 + b_3(6-4)^2 + c_3(6-4) + 7 = 9$.

Las condiciones de continuidad de la derivada primera son $3a_1(2-1)^2 + 2b_1(2-1) + c_1 = c_2$, $3a_2(4-2) + 2b_2(4-2) + c_2 = c_3$; (observa que estas ecuaciones permiten sustituir c_2, c_3).

Las condiciones de continuidad de la derivada segunda son $6a_1(2-1) + 2b_1 = 2b_2$, $6a_2(4-2) + 2b_2 = 2b_3$; (observa que estas ecuaciones permiten sustituir b_2, b_3).

Finalmente las condiciones de frontera natural son $b_1 = 0, 6a_3(6-4) + 3b_3 = 0$. Hay 9 ecuaciones y 9 incógnitas que permiten calcular $a_1, a_2, a_2, b_1, b_2, b_3, c_1, c_2, c_3$.

Ejercicio: Resuelve con Maxima las ecuaciones y representa la función. Observa que la gráfica de la función es suave.

Ejercicio: Escribe el sistema para las condiciones de frontera $y'(1) = 1, y'(6) = -1$. Resuelve con Maxima las ecuaciones y representa la función. Observa que la gráfica de la función es suave y mira como es la tangente en los extremos.

En el paquete *interpol* de Maxima la instrucción *cspline* calcula las interpolación por trazadores cúbicos. Mira en la ayuda como usar cspline y comprueba si los resultados de los dos ejercicios anteriores coinciden con los que genera directamente Maxima.

3 Aproximación por mínimos cuadrados

Dada una colección de puntos, otra forma de platear el problema es hallar una función de un tipo determinado que sea la que más se acerque a los puntos. Este enfoque es válido si suponemos que los puntos corresponden a medidas que tienen errores.

Por ejemplo, si tenemos una colección de puntos $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$, y suponemos que las medidas y_i están afectadas por errores, podemos buscar cual es la ecuación de la recta $y = a\,x + b$ tal que las suma

de las distancias verticales de los puntos a la recta sea mínima. A esta recta se le llama *recta de regresión* y corresponde a hacer mínima la expresión $\sum_{i=1}^{n}(y_i-(a\,x_i+b))^2$.

Para obtener a, b igualamos a cero las derivadas parciales y tenemos

$\frac{\partial}{\partial a}\sum_{i=1}^{n}(y_i-(a\,x_i+b))^2 = 2\sum_{i=1}^{n}(-x_i)(y_i-(a\,x_i+b)) = 0;$

$\frac{\partial}{\partial b}\sum_{i=1}^{n}(y_i-(a\,x_i+b))^2 = 2\sum_{i=1}^{n}(-1)(y_i-(a\,x_i+b)) = 0;$

simplificando tenemos que

$a\,\sum_{i=1}^{n}x_i^2 + b\,\sum_{i=1}^{n}x_i = \sum_{i=1}^{n}x_iy_i;$

$a\,\sum_{i=1}^{n}x_i + n\,b = \sum_{i=1}^{n}y_i;$

Resolviendo el sistema se obtiene que

$$a=\frac{n\,(\sum_{i=1}^{n}x_iy_i)-(\sum_{i=1}^{n}x_i)(\sum_{i=1}^{n}y_i)}{n\,(\sum_{i=1}^{n}x_i^2)-(\sum_{i=1}^{n}x_i)^2};\quad b=\frac{(\sum_{i=1}^{n}x_i^2)(\sum_{i=1}^{n}y_i)-(\sum_{i=1}^{n}x_iy_i)(\sum_{i=1}^{n}x_i)}{n\,(\sum_{i=1}^{n}x_i^2)-(\sum_{i=1}^{n}x_i)^2};$$

En Maxima para unos datos dados como lispun:[[1,0],[2,1],[3,3],[4,2],[5,4],[6,5]];

recreg(lis):=block([n:length(lis),sx,sy,sxy,sx2],sx:sum(lis[i][1],i,1,n),sy:sum(lis[i][2],i,1,n),

sxy:sum(lis[i][1]*lis[i][2],i,1,n),sx2:sum(lis[i][1]^2,i,1,n),

(n*sxy-sx*sy)/(n*sx2-sx^2)*x+(sx2*sy-sxy*sx)/(n*sx2-sx^2));

ahora hacemos fun:recreg(lispun); y nos da la ecuación de la recta de regresión para esos puntos. Para pintarla podemos hacer

plot2d([fun,[discrete,lispun]],[x,0,6],[style,[lines,1,3],[points,3,2]]); y se observa como la recta esta entre los puntos.

Si queremos aproximar por un polinomio de mayor grado y no queremos deducir la fórmula general podemos utilizar Maxima para hallarlo, por ejemplo, para aproximar los puntos por un polinomio de grado 2 con coeficientes ax^2+bx+c la función a minimizar será $\sum_{i=1}^{n}(y_i-(a\,x_i^2+b\,x_i+c))^2$ así pues hacemos en Maxima para los puntos

lispun:[[1,0],[2,3],[3,7],[4,12],[5,20],[6,25]];

fm:sum((lispun[i][2]-(a*lispun[i][1]^2+b*lispun[i][1]+c))^2,i,1,length(lispun));

solve([diff(fm,a),diff(fm,b),diff(fm,c)]);

nos da $c=\frac{-13}{5}, b=\frac{269}{140}, a=\frac{13}{28}$

y con los valores obtenidos definimos el polinomio, por ejemplo

pol:$\frac{13}{28}x^2+\frac{269}{140}x-\frac{13}{5}$;

Ejercicio: comprueba los cálculos y pinta los puntos y el polinomio.

Ejercicio: Calcula el polinomio de mínimos cuadrados de grado 3 que mejor aproxima a esos puntos, pinta los puntos y el polinomio. (Ayuda: el polinomio será de la forma $a\,x^3+b\,x^2+c\,x+d$, y la función a minimizar será $\sum_{i=1}^{n}(y_i-(a\,x_i^3+b\,x_i^2+c\,x_i+d))^2$.)

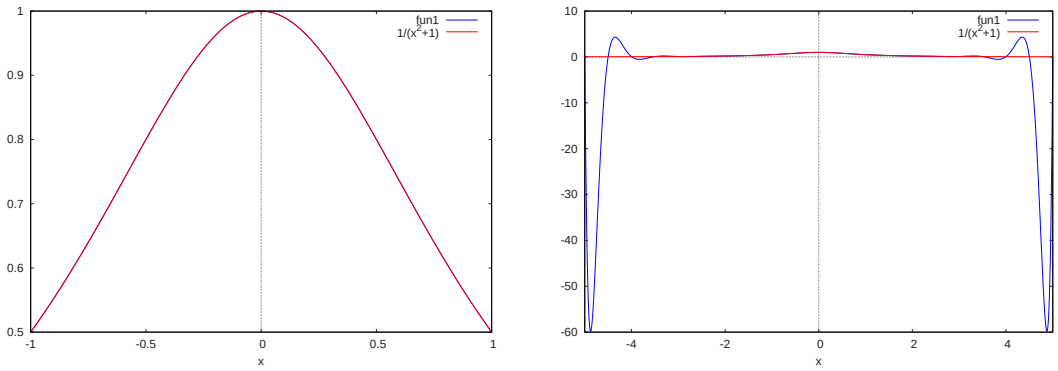

Figura 5.3.1: Polinomio de interpolación, 25 puntos de $\frac{1}{1+x^2}$ en [-1,1] y en [-5,5]

Ejercicio: Escribe una subrutina en Maxima que calcule el polinomio de regresión de grado n de una lista de puntos. Ayuda: Prueba

polreg(lis,n):=block([m:length(lis),a,pol,funob,camb],

if n<m then (

pol:sum(concat(a,i)*x^ i,i,0,n), funob:sum((lis[j][2]-ev(pol,x=lis[j][1]))^ 2,j,1,m),

camb:flatten(solve(makelist(diff(funob,ev(concat(a,i))),i,0,n))), ev(pol,camb)) else print("Número de puntos insuficiente"));

/*las variables locales son: m la longitud de la lista, a para generar los coeficientes a0,a1,a2,...an del polinomio pol, funob la distancia a minimizar, camb los valores de a0,a1,..an. Busca en la ayuda la función concat */

Si la función no es un polinomio, las ecuaciones que resultan de hacer cero las derivadas parciales no tienen por que ser lineales y hay que resolverlas numéricamente por un algoritmo de Newton. Es mucho más difícil. Por ejemplo si ajustamos por $\exp(a\,x+b)$ la función a minimizar es $\sum_{i=1}^{n}(y_i-(\exp(a\,x_i+b))^2$. Calcula las derivadas respecto a y b y comprueba que no dan origen a ecuaciones lineales en a, b. En algunos casos cambiando las variables se puede simplificar, por ejemplo si $y=\exp(a\,x+b)$ entonces tomando logaritmos neperianos $ln(y)=a\,x+b$ por lo que puede hacerse la recta de regresión.

En Maxima el paquete $lsquares$ permite ajustar funciones no lineales a una tabla de datos dada como un matriz por mínimos cuadrados.

4 Problemas

1. Considera 21 puntos igualmente distribuidos de la función $\frac{1}{1+x^2}$ en el intervalo [-1,1]. Halla el polinomio de interpolación de Lagrange que pasa por estos puntos y dibújalo junto con la función. Ayuda: puedes crear la lista de puntos en Maxima con pun:create_list([i/10,1/(1+(i/10)^ 2)],i,-10,10); después carga el paquete con load(interpol); y haz pol:lagrange(pun); para guardar en la variable pol el polinomio de interpolación. Para dibujarlo puedes hacer plot2d([pol,1/(1+x^ 2)],x,-1,1);. Interpreta el resultado y observa si la interpolación es buena (Puedes verlo en la primera de las figuras 5.3.1).

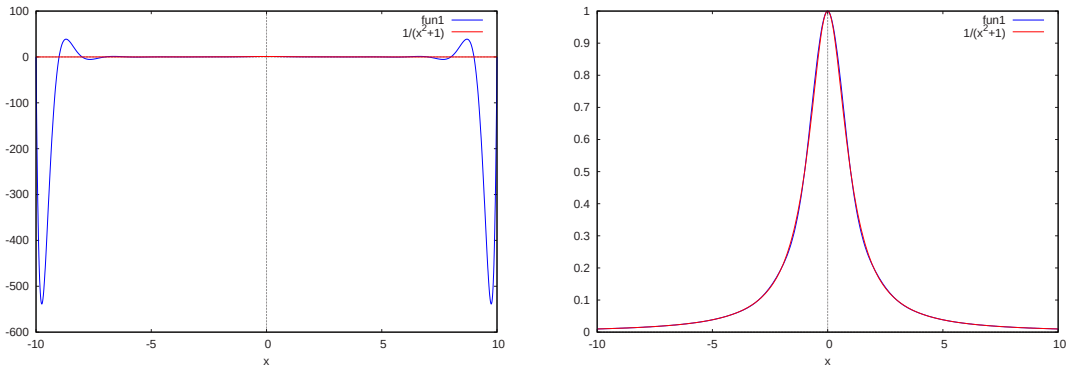

Figura 5.4.1: Polinomio de interpolación y funciones ranura con 25 puntos de $\frac{1}{1+x^2}$ en [-10,10]

Elige ahora los 21 puntos en el intervalo [-5,5] calcula el polinomio de interpolación y dibújalo junto con la función en [-5,5]. Interpreta de nuevo el resultado y observa si la interpolación es buena. Ayuda: Haz ahora pun:create_list([i/2,1/(1+(i/2)^2)],i,-10,10); etc.. (Puedes verlo en la segunda de las figuras 5.3.1).

Finalmente elige ahora los 21 puntos en el intervalo [-10,10] calcula el polinomio de interpolación y dibújalo junto con la función en $\frac{1}{1+x^2}$ [-10,10]. Interpreta de nuevo el resultado y observa que la interpolación es muy mala cerca de ± 10. Ayuda: Haz ahora pun:create_list([i,1/(1+i^2)],i,-10,10); etc.. (Puedes verlo en la primera de las figuras 5.4.1)

2. Repite el Problema anterior calculando la interpolación con funciones ranura con condiciones de frontera libre (f"=0) en los extremos en los tres casos [-1,1],[-5,5] y [-10,10], dibuja la función $\frac{1}{1+x^2}$ y la función de interpolación. Ayuda: usa funr:cspline(pun); para guardar en la variable funr la interpolación por funciones ranura. Observa que la interpolación se asemeja a la función en todos los casos. El caso correspondiente al intervalo [-10,10] puedes verlo en la segunda de las figuras 5.4.1.

3. Halla la recta de regresión de los 21 puntos del problema primero a través de la rutina polreg que hemos definido (Ayuda, copia la definición de polreg y luego haz polreg(pun,1); suponiendo que tienes los puntos en la variable pun. Alternativamente a través de la función lsquares Ayuda: Haz load(lsquares); si tienes definidos los puntos en pun para definir la matriz puedes hacer

   A:matrix(pun[1]); for ii in rest(pun) do A:addrow(A,ii); /* y luego */ lsquares(A,[x,y],y=a*x+b,[a,b]);.

   Compara si los resultados son iguales. Dibuja la recta de regresión y la función $\frac{1}{1+x^2}$. Calcula los polinomios de mínimos cuadrados de grado 2 y 3 y dibújalos con la función original.

4. Para simular errores experimentales calcula una lista de 21 puntos en [-2,2] con los valores de $x^2 - x + 1$ mas un número aleatorio de valor absoluto < 0.1, ajusta por mínimos cuadrados un polinomio $ax^2 + bx + c$ a los puntos y dibuja los puntos, $x^2 - x + 1$ y el polinomio que obtengas. Observa si el polinomio ajustado por mínimos cuadrados se parece al polinomio original. Repite los cálculos sumando a los datos un error aleatorio de valor absoluto menor que 0.5; Ayuda, puedes hacer pun:create_list([i/5,(i/5)^2-(i/5)+1+random(0.2)-0.1],i,-10,10); para que el error esté centrado, después define $polreg$ y haz polr:polreg(pun,2); (o usa $lsquares$ de Maxima) finalmente haz plot2d([x^2-x+1,polr,[discrete,pun]],[x,-2,2]);

CAPÍTULO 6

Aproximación de funciones.

Índice del Tema

En el capítulo anterior hemos tratado de aproximar una familia de puntos por una función. Ahora vamos a tratar de aproximar una función a través una familia de funciones sencillas: polinomios o funciones trigonométricas. Hay dos posibilidades:

- que la aproximación sea buena cerca de un punto - aunque mas lejos no lo sea tanto.
- que la aproximación sea buena en un intervalo prefijado.

Los **Objetivos** de este tema son:

- Separar el problema de hallar una buena aproximación local de una función (taylor) del problema de hallar una buena aproximación de una función en todo un intervalo.
- Saber calcular el polinomio de Taylor de una función.
- Entender que $d(f,g)=\sqrt{\int_a^b (f(x)-g(x))^2 dx}$ está relacionada con el área entre las funciones f, g.
- Ser capaz de calcular los coeficientes del desarrollo de una función en polinomios de Legendre y en serie de Fourier.
- Conocer la existencia de la transformada de Fourier rápida (FFT) y ser capaz de aplicarla para eliminar el ruido de un conjunto de datos, utilizando las herramientas de Maxima.

1 Polinomio de Taylor

Si una función f es suficientemente derivable la forma de aproximarla cerca de una punto x_0 por un polinomio de grado n está dada por

$$\sum_{i=0}^{n} \frac{f^{(i)}(x_0)}{i!}(x-x_0)^i$$

a esta expresión se le llama *Polinomio de Taylor de grado n de f en x_0*.

En Maxima podemos hacer

/* funciona para una función de x*/

kill(mitaylor); mitaylor(fun,x0,n):=block([yy,tem],

tem:ev(fun,x=x0)+sum(diff(fun,x,i)/i!*(yy-x)^i,i,1,n),ev(tem,x=x0,yy=x));

/* luego haz mitaylor(sin(x),0,5); */

Maxima la calcula con la instrucción *taylor*(fun,x,x0,n); pero pone puntos suspensivos detrás para indicar que aproxima a la función, para quitarlos haz taytorat(taylor(fun,x,x0,n));

con

plot2d([sin(x),taytorat(taylor(sin(x),x,0,3)),taytorat(taylor(sin(x),x,0,5)), taytorat(taylor(sin(x),x,0,7))],[x, -10,10],[y,-20,20]);

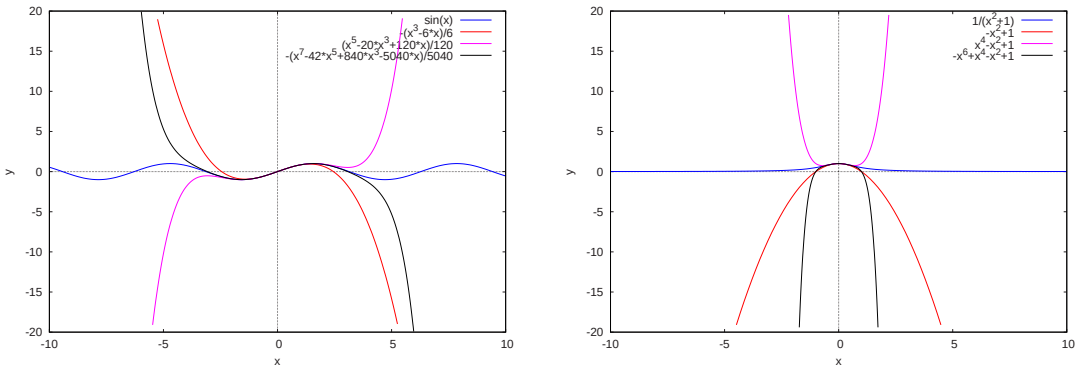

Figura 6.1.1: Polinomios de Taylor para $sen(x)$ y $\frac{1}{1+x^2}$ de grados 1,3,5,7

puedes ver como los polinomios de taylor de orden 3,5 y 7 aproximan a la función seno cerca de cero. Observa que la aproximación es muy buena cerca de cero pero luego se estropea. Dibuja polinomios de taylor de grado más alto a ver si mejora la aproximación (Observa la primera de las figuras de 6.1.1).

Repite el proceso para la función $\frac{1}{1+x^2}$ para ver si mejora la aproximación que ofrecen los polinomios de taylor cuando sube el grado del polinomio en particular fuera del intervalo [-1,1] (Observa la segunda de las figuras de 6.1.1).

2 Aproximación de funciones en intervalos

¿Cuál es la distancia entre dos funciones en un intervalo $f, g : [a, b] \in \mathbb{R} \to \mathbb{R}$? Hay varias alternativas entre ellas:

- $d(f, g) = sup\{|f(x) - g(x)|, x \in [a, b|\}$; si f, g son continuas entonces $d(f, g)$ es el máximo de $\{|f(x) - g(x)|, x \in [a, b|\}$. Que $d(f, g) < \varepsilon$ equivale a que en todos los puntos del intervalo [a,b] f dista de g menos de ε. Esta distancia da origen a la convergencia uniforme y es muy útil en matemáticas.
- Si las funciones son integrables podemos definir de otra forma la distancia a partir de $d(f, g) = \sqrt{\int_a^b (f(x) - g(x))^2 dx}$. Ya que la integral corresponde al área, esta distancia está relacionada con el área que está entre las gráficas de las funciones f, g, por lo tanto puede haber puntos en los que las funciones disten bastante pero en media las funciones distarán poco. A esta distancia se le suele llamar distancia en media cuadrática. En lo que sigue de éste capítulo usaremos esta definición de distancia.

La idea es aproximar, de acuerdo con esta definición de distancia una función por una familia en principio infinita de funciones más sencillas $\mathbf{F} = \{p_1, p_2, p_3, ...p_n....\}$, por ejemplo polinomios o funciones trigonométricas, para ello queremos encontrar unas constantes $a_1, a_2, ..., a_n, ...$ tal que la distancia entre f y $\sum_{i=1}^{\infty} a_i p_i = a_1 p_1 + a_2 p_2 + a_3 p_3 + ...$ se haga cada vez mas pequeña. Se puede probar que esta distancia se deduce del producto escalar entre funciones $< f, g >= \int_a^b f(x)g(x)dx$ que induce la norma para funciones $||f|| = \sqrt{\int_a^b f^2(x)dx}$; observa que la norma es la distancia entre la función y cero. Si estás un

poco familiarizado con el tema se tiene que el espacio de las funciones reales definidas en un intervalo es un espacio vectorial y dada una familia de vectores (que en este caso son funciones) linealmente independiente se puede obtener una familia ortonormal por un proceso llamado método de Gram-Schmidt.

La forma más cómoda de hallar los coeficientes es en el caso en que la familia **F** es ortogonal es decir $< p_i, p_j >= 0, i \neq j$; porque, si queremos que $f = \sum_{i=1}^{\infty} a_i p_i$ entonces, calculando el producto escalar con p_n, tenemos que $< f, p_n >=< \sum_{i=1}^{\infty} a_i p_i, p_n >= a_n < p_n, p_n >$ con lo que $a_n = \frac{<f,p_n>}{<p_n,p_n>}$ es decir

$$a_n = \frac{\int_a^b f(x)p_n(x)dx}{\int_a^b p_n^2(x)dx}; \qquad f = \sum_{n=1}^{\infty} a_n p_n; \tag{6.2.1}$$

6.2.1 Polinomios de Legendre

Si consideramos el intervalo [-1,1] y se busca una familia ortogonal a partir de $1, x, x^2, x^3, ...$ se obtiene los polinomios de Legendre que pueden definirse a partir de $P_n(x) = \frac{1}{n!2^n}\frac{d^n}{dx^n}(x^2-1)^n$.

Ejercicio: calcula los 5 primeros polinomios de Legendre.

Ayuda: prueba makelist(diff((x^2-1)^n,x,n)/(n!*2^n),n,0,4);

En Maxima cargando el paquete *orthopoly* la instrucción *legendre_ p(n,x)* devuelve el polinomio de Legendre de grado n. Puedes ver los 5 primeros haciendo makelist(legendre_ p(n,x),n,0,4); Comprueba si te salen los mismos polinomios.

Ejercicio: Comprueba que los polinomios de Legendre P_n para $n = 0, 1, 2, 3, 4$ forman un sistema ortogonal en [-1,1]. Ayuda: prueba create_ list(integrate(legendre_ p(i,x)*legendre_ p(j,x),x,-1,1),i,0,4,j,0,4);

Dibuja los 5 primeros polinomios de Legendre en [-1,1].

Veamos como se puede aproximar una función, por ejemplo $exp(x)$ en el intervalo [-1,1] a través de los primeros 5 polinomios de Legendre. Como hemos visto que los polinomios de Legendre son ortogonales en [-1,1] lo que hemos de hacer es calcular los coeficientes definidos en (6.2.1) y formar la suma $\sum_i a_i P_i(x)$ definiendo por ejemplo la función

aprleg(fun,n):=expand(float(sum(integrate(legendre_ p(i,x)*fun,x,-1,1)

/integrate(legendre_ p(i,x)*legendre_ p(i,x),x, -1,1)*legendre_ p(i,x),i,0,n))); y podemos dibujar la función $exp(x)$ y las aproximaciones en el intervalo [-1,1] con la instrucción

plot2d([exp(x),aprleg(exp(x),0),aprleg(exp(x),1),aprleg(exp(x),2),aprleg(exp(x),3)],[x,-1,1]);

Observa como las áreas entre la función $exp(x)$ y las sucesivas aproximaciones se hacen cada vez más pequeñas, en particular la última se acerca mucho. Lo puedes ver dibujado en la primera gráfica de 6.2.1. Esta es la idea de la distancia $d(f,g) = \sqrt{\int_a^b (f(x)-g(x))^2 dx}$, medir el area entre las curvas.

Si quieres verlo mejor, dibuja las diferencias entre las aproximaciones segunda y tercera por polinomios de Legendre y la función exponencial con plot2d([exp(x)-aprleg(exp(x),2),exp(x)-aprleg(exp(x),3)],[x,-1,1]); y verás que se van acercando en todo el intervalo, en el caso de la tercera difieren en menos de 0.012 en todo el intervalo. Lo puedes ver dibujado en la segunda gráfica de 6.2.1.

Ejercicio: Comprueba que lo mismo ocurre para otras funciones, por ejemplo $\frac{1}{1+x^2}$ ó $sen(x)$ es decir, calcula

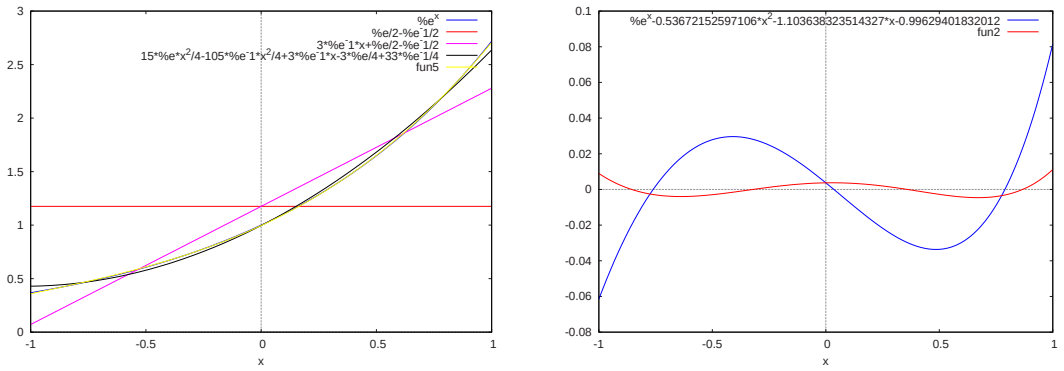

Figura 6.2.1: Aproximacion por polinomios de Legendre a exp(x) en [-1,1] y diferencias con la función

4 aproximaciones por polinomios de Legendre de $\frac{1}{1+x^2}$, dibuja la función junto con las aproximaciones y comprueba si se acercan cada vez más en el intervalo [-1,1]. Observa si, para $\frac{1}{1+x^2}$ hay menos aproximaciones por polinomios de Legender dibujados de los que debería. ¿Puede ocurrir que los coeficientes a_n pares o impares son cero? Compruébalo. Dibuja más aproximaciones de $\frac{1}{1+x^2}$.

6.2.2 Serie de Fourier

Si consideramos el intervalo [-π, π] tenemos que la familia

F={1,cos(x),cos(2x),cos(3x),....,sen(x),sen(2x),sen(2x),sen(3x)....} es ortogonal.

Ejercicio, comprueba con varios miembros si es cierto que la familia **F** es ortogonal en [-π, π]. Ayuda: prueba

create_ list(integrate(sin(i*x)*cos(j*x),x,-%pi,%pi),i,1,4,j,0,4);

create_ list(integrate(cos(i*x)*cos(j*x),x,-%pi,%pi),i,0,4,j,0,4);

create_ list(integrate(sin(i*x)*sin(j*x),x,-%pi,%pi),i,1,4,j,1,4);

para comprobar las integrales de los senos entre si, los cosenos entre si y senos y cosenos entre ellos. Observa que cos(0)=1.

Dibuja algunas funciones de la familia trigonométrica en [-π, π].

Ejemplo plot2d([sin(x),sin(2*x),sin(3*x),cos(x),cos(2*x),cos(3*x)],[x,-%pi,%pi]);

Dada una función f definida en [-π, π] se llama *desarrollo en serie de Fourier de f* a

$$a_0 + \sum_{n=1}^{\infty} a_n cos(nx) + \sum_{n=1}^{\infty} b_n sen(nx) \tag{6.2.2}$$

con

$$a_n = \frac{\int_{-\pi}^{\pi} f(x)cos(nx)dx}{\int_{-\pi}^{\pi} cos^2(nx)dx}, \qquad b_n = \frac{\int_{-\pi}^{\pi} f(x)sen(nx)dx}{\int_{-\pi}^{\pi} sen^2(nx)dx};$$

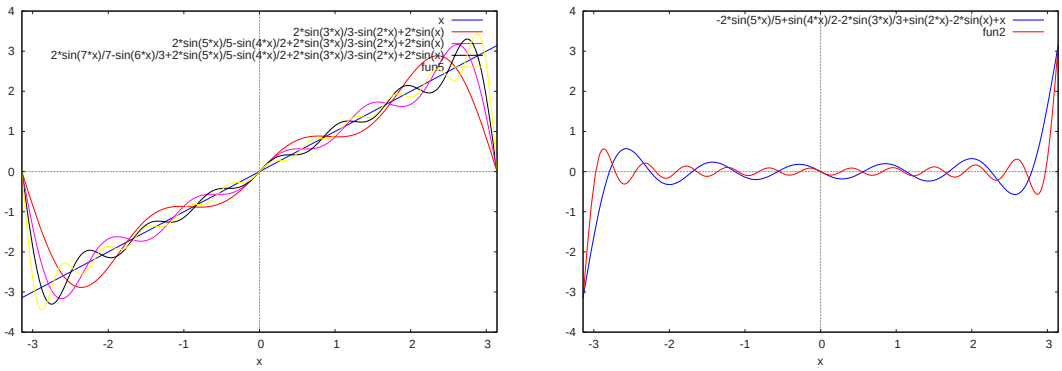

Figura 6.2.2: Aproximación por Fourier a x en $[-\pi,\pi]$ y diferencias con la función

Debido a que las funciones $sen(n\,x)$ son impares, es decir que $f(-x)=f(x)$ para todo x y las funciones 1, $cos(n\,x)$ son pares, es decir que $f(-x)=f(x)$ para todo x se puede probar que si una función es impar su desarrollo en serie de Fourier se puede escribir sólo en senos y si la función es par su desarrollo en serie de Fourier se puede escribir sólo en 1 y en cosenos.

Ejercicio: Halla los coeficientes a_0, a_1, a_2, b_1, b_2 del desarrollo de fourier de la función x en el intervalo [-π, π], comprueba si los coeficientes a_i son cero, dibuja la función y la aproximación. Dibuja la suma de más términos de la serie de fourier y mira si se aproximan mejor a la función. Ayuda puedes definir algo como

```
aprfou(fun,n):= expand(sum(integrate(cos(i*x)*fun,x,-%pi,%pi)/integrate(cos(i*x)^2,x,-%pi,%pi)*cos(i*x),i,0,n)
+sum(integrate(sin(i*x)*fun,x,-%pi,%pi)/integrate(sin(i*x)^2,x,-%pi,%pi)*sin(i*x),i,1,n));
```

y luego hacer aprfou(x,3) para calcular la aproximación sumando hasta $n=3$. Para dibujar puedes hacer

```
plot2d([x,aprfou(x,3),aprfou(x,5),aprfou(x,7),aprfou(x,11)],[x,-%pi,%pi]);
```

Lo puedes ver en la primera figura de 6.2.2.

Dibuja la diferencia entre x y las aproximaciones de fourier para $n=5$, $n=11$ haciendo

```
plot2d([x-aprfou(x,5),x-aprfou(x,11)],[x,-%pi,%pi]);
```

Lo puedes ver en la segunda figura de 6.2.2. Fíjate que las áreas entre la función y las aproximaciones se hacen más pequeñas.

Dibuja también la diferencia entre x y su aproximación por Fourier para $n=31$ y observa que es menor que la correspondiente a $n=11$. Observa que, como las funciones $sen(n\,x)$, $cos(n\,x)$ son periódicas y x toma valores distintos en $x=\pi, x=pi$ la aproximación por fourier se porta mal en estos puntos. En realidad ocurre que, cuando la función f tiene una discontinuidad de salto, su aproximación por Fourier tiene a magnificar el salto alrededor de un 18 por ciento, esto se llama fenómeno de Gibbs.

Ejercicio: comprueba que x^2 tiene la parte en senos de su desarrollo de fourier nula comprobando varios coeficientes b_n. Dibuja varias sumas parciales de la serie de Fourier que aproximen x^2 junto con la función. Como x^2 toma los mismos valores en $x=\pi, x=pi$ el fenómeno de Gibbs no aparecerá ahora.

Dado que los senos y cosenos aparecen como soluciones de la ecuación de ondas y las ondas son omnipresentes en la naturaleza, en gran cantidad de parcelas de la física, química e ingeniería se utilizan las

series de Fourier.

En Maxima en el paquete *fourie* existen funciones que calculan los coeficientes de la serie de Fourier. Investiga totalfourier. Comprueba si los resultados de Maxima son correctos.

3 Transformada de Fourier Rápida FFT

Existe un análogo a la serie de Fourier para la interpolación de grandes cantidades de datos discretos. Suponemos que tenemos una colección de $2m$ puntos $(x_0, y_0), ...(x_{2m-1}, y_{2m-1})$ y que los valores de x_i están igualmente distribuidos en [-π, π] es decir $x_j = -\pi + \frac{j}{m}\pi$. Queremos hallar un polinomio trigonométrico de mínimos cuadrados

$$S_n(x) = \frac{a_0}{2} + \sum_{k=1}^{n} a_k cos(kx) + \sum_{k=1}^{n-1} b_k sen(kx)$$

tal que sea mínimo $\sum_{j=0}^{2m-1}(y_j - S_n(x_j))^2$; observa que es lo que minimizábamos en el capítulo anterior cuando estudiamos mínimos cuadrados. Tenemos que hallar las constantes a_k, b_k que aparecen en S_n. El cálculo se simplifica porque resulta que, si los puntos x_j están igualmente distribuidos en $[-\pi, \pi]$, las funciones $1, cos(x), cos(2x), ..., sen(x), sen(2x)...$ son también ortogonales respecto al producto escalar $< f, g >= \sum_{j=1}^{2m-1} f(x_j)g(x_j)$ (que es un análogo de la integral que aparece en la serie de fourier). Se deduce que

$$a_k = \frac{1}{m}\sum_{j=0}^{2m-1} y_j\, cos(k\, x_j); \quad b_k = \frac{1}{m}\sum_{j=0}^{2m-1} y_j\, sen(k\, x_j);$$

Ejercicio: Obtener el polinomio $S_3(x)$ que corresponde a 20 puntos de la forma (x_j, e^{x_j}) Ayuda: para generar los puntos puedes hacer

```
pun:makelist([-%pi+j/10*%pi,exp(-%pi+j/10*%pi)],j,0,19);
```

y luego hacer kill(poltrig);poltrig(lis,n):=block([mm:length(lis)],

```
2/mm*(sum(lis[j][2],j,1,mm)/2+sum(sum(lis[j][2]*cos(k*lis[j][1]),j,1,mm)*cos(k*x),k,1,n)
+sum(sum(lis[j][2]*sin(k*lis[j][1]),j,1,mm)*sin(k*x),k,1,n-1) ) );
poli:float(poltrig(pun,3));
```

Ejercicio: dibuja los puntos y el polinomio S_3, Calcula y dibuja S_5, Idem S_7 y S_9. Finalmente hazlo con S_{10}

Ejercicio: Usa ahora 40 puntos de la función exponencial y observa si S_{20} se aproxima mejor.

Observa que si $m = n$ el número de puntos coincide con el número de incógnitas a_k, b_k y por tanto el polinomio S_n es un polinomio (trigonométrico) de interpolación ya que el polinomio de interpolación pasa por los puntos y la distancia $\sum_{j=0}^{2m-1}(y_j - S_n(x_j))^2$ se reduce a cero.

Si m es muy grande el cálculo de los coeficientes es muy laborioso pero, si m es una potencia de 2 existe un algoritmo, llamado FFT o algoritmo de Cooley-Tukey que reduce mucho el cálculo. Para expresarlo es

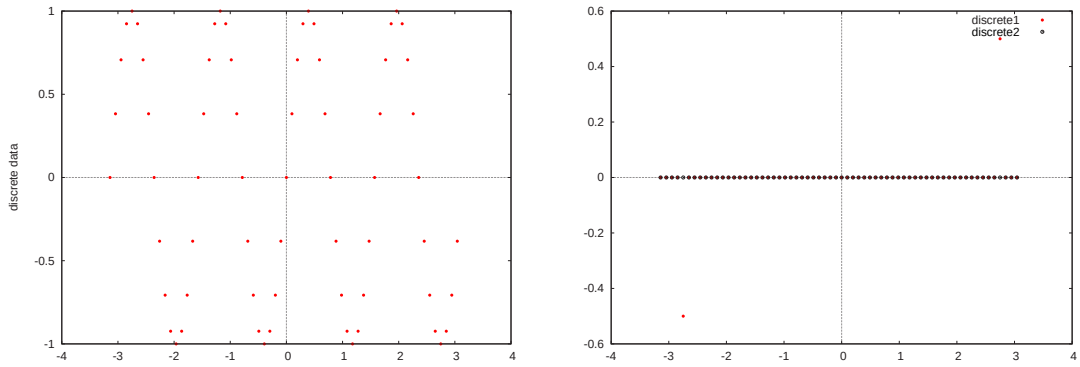

Figura 6.3.1: 64 puntos de sen(4x) y valores reales e imaginarios del espectro de los puntos

más simple usar números complejos. Ya que $e^{iz} = cos(z) + i\, sen(z)$ con $i = \sqrt{-1}$ la unidad imaginaria y que $e^{-iz} = cos(z) - i\, sen(z)$ se tiene que $sen(z) = \frac{e^{iz}-e^{-iz}}{2i}, cos(z) = \frac{e^{iz}+e^{-iz}}{2}$ por lo que podemos escribir (con algún cambio en los coeficientes)

$$S_n(x) = \frac{1}{m}\sum_{k=0}^{2m-1} c_k e^{ikx}, \quad con \quad c_k = \sum_{j=0}^{2m-1} y_j e^{ikx_j},$$

y $a_k + ib_k = \frac{(-1)^k}{m} c_k$

Maxima en el paquete *fft* incluye instrucciones para calcular los coeficientes de polinomios trigonométricos por la transformada de Fourier rápida, los datos deben estar en arrays, no en listas.

Los datos (complejos) deben estar en un array las partes reales y en otro las partes imaginarias; la instrucción *fft* devuelve las partes reales e imaginarias de los coeficientes del polinomio trigonométrico en las mismas arrays en las que estaban los datos originales. La instrucción *ift* es la transformada inversa y devuelve los datos originales a partir de los coeficientes del polinomio trigonométrico.

la transformada de fourier esta dada por

y[k]: (1/n) sum (x[j] exp (-2 %i %pi j k / n), j, 0, n-1) donde x[j] es array_real[j]+%i array_imaginaria[j] (en nuestro caso las hemos llamado punr,puni)

Los coeficientes a_j, b_j equivalentemente c_j están relacionados con el *espectro* de la función que representan los datos y su estudio es muy útil en electromagnetismo, física, etc. En particular podemos utilizar la FFT para eliminar parte del ruido de fondo de una señal que usualmente corresponde a errores en los datos. La idea básica es eliminar aquellos coeficientes del desarrollo de Fourier que son más pequeños.

Por ejemplo creamos una array con 64 datos de la función $sen(4x)$

/* declaramos dos arrays, una para las partes reales y otra para las imaginarias y las llenamos con los valores numéricos, que han de ser float */

array([punr,puni],63);

fillarray(punr,makelist(ev(sin(4*(-%pi+j/32*%pi)),numer),j,0,63)); fillarray(puni,makelist(0.0,j,0,63));

/* puedes pintarlas haciendo plot2d([discrete,makelist([-%pi+j/32*%pi,punr[j]],j,0,63)],[style,[points,1,2]]);

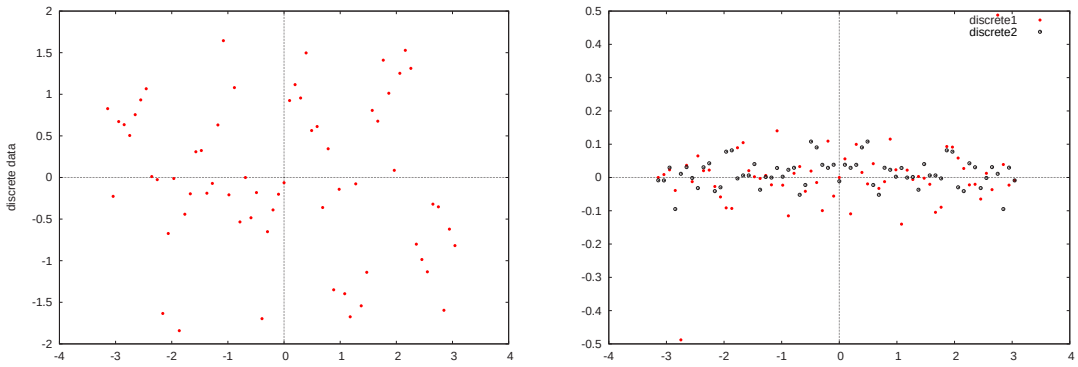

Figura 6.3.2: Puntos de sen(4x) contaminados con ruido y espectro

está dibujado en la primera figura de 6.3.1*/

/* cargamos el paquete y calculamos la transformada de Fourier*/

load(fft);

fft(punr,puni);

/* ahora los datos de punr, puni han pasado a ser las partes reales e imaginarias de c_j puedes pintarlas haciendo, de nuevo, plot2d([discrete,makelist([-%pi+j/32*%pi,punr[j]],j,0,63)],[style,[points,1,2]]);

o verlas con listarray(punr); listarray(puni)*/

/* Observa que los datos de punx son pequeños, mira los de puni ahora hay dos valores grandes, los que corresponden a $sen(4x) = \frac{exp(4ix)-exp(-4ix)}{2i}$ por lo que hay uno que corresponde a 4 y otro a -4*/

/* los valores de punx,puni restantes son errores de redondeo.*/ Los valores de las partes reales e imaginarias del espectro están dibujadas en la segunda figura de 6.3.1.

La FFT se puede usar para eliminar posibles errores de una lista de datos, generamos una lista de valores de la función $sen(4x)$ contaminados con un error aleatorio. Recuerda que random(float(2)) devuelve un número real entre 0 y 2, para que los errores estén centrados en la medida le restamos 1. Observa en la primera de las figuras de 6.3.2 que la función sen(4x) es irreconocible.

/* llenamos las arrays de datos contaminados de ruido (recuerda que si no están creadas hay que crearlas)*/

fillarray(punr,makelist(ev(sin(4*(-%pi+j/32*%pi))+random(float(2))-1,numer),j,0,63));

fillarray(puni,makelist(ev(0,numer),j,0,63));

/* los pintamos para ver que la funcion $sen(4x)$ está disimulada por el ruido (Ojo, como el ruido es aleatorio el dibujo en tu ordenador puede ser distinto) */

plot2d([discrete,makelist([-%pi+j/32*%pi,punr[j]],j,0,63)],[style,[points,1,2]]);

/* aplicamos la fft */

fft(punr,puni);

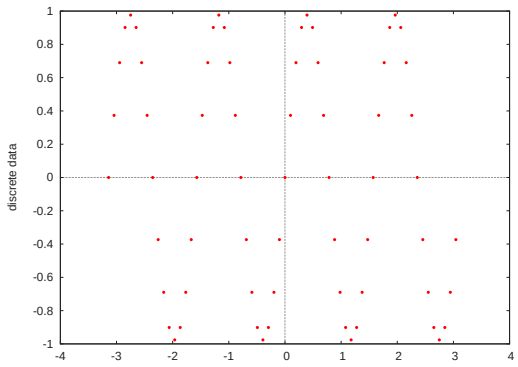

Figura 6.3.3: Reconstrucción de los datos sin ruido

/* si pintas ahora los valores de puni verás que dos están en valor absoluto por encima de los demás, son los que corresponden a sen(4x). Observa en la segunda de las figuras de 6.3.2, pese al ruido, cómo destacan los coeficientes que corresponde a sen(4x).*/

/* Si queremos podemos filtrar los datos pequeños, veamos cuales son los valores mayores /*

sort(listarray(punr)); sort(listarray(puni));

/*el mayor valor es ~ 0.5, para filtrar el ruido hacemos cero todos los que sean menores, por ejemplo, que 0.2 en valor absoluto; creamos dos nuevas arrays para los datos del espectro filtrado*/

array([punrfil,punifil],63);

fillarray(punrfil,makelist(if abs(punr[j])<.2 then float(0) else punr[j],j,0,63));

fillarray(punifil,makelist(if abs(puni[j])<.2 then float(0) else puni[j],j,0,63));

/*ahora restauramos los datos con la transformada inversa */

ift(punrfil,punifil);

/*pintamos los datos */

plot2d([discrete,makelist([-%pi+j/32*%pi,punrfil[j]],j,0,63)],[style,[points,1,2]]);

/* ahi tienes los datos sin ruido. Evidentemente si el filtro es muy fuerte podemos perder información */

Los datos con el ruido filtrado están pintados en 6.3.3.

Ejercicio: Crea una lista con 128 datos basados en la función $2sen(4x) + cos(8x)$ contaminada con un ruido aleatorio de amplitud 1. Dibuja la lista. Aplica la FFT para filtrar los datos. Dibuja la función sin ruido. Calcula el tiempo que ha tardado Maxima en hacer las cuentas.

Ejercicio: Crea ahora una lista con 512 datos. Repite los cálculos y compara los tiempos que tarda Maxima ahora.

CAPÍTULO 7

Derivación e integración numérica.

Índice del Tema

Con frecuencia tenemos una serie de datos experimentales y nos interesa estimar los valores de la derivada o la integral de la función a la que corresponden. Además puede ocurrir que la integral de una función sea difícil de hacer analíticamente y sea conveniente conocer su valor aproximado.

Los **Objetivos** de éste tema son:

- Conocer y aplicar fórmulas de derivación numérica.
- Conocer y aplicar fórmulas de integración numérica, entre ellas fórmulas de integración compuesta.
- Saber utilizar las herramientas de derivación e integración de Maxima.
- Entender el concepto de integración adaptativa.

1 Derivación numérica

Ya que la definición de derivada de f en x es $f'(x) = \lim_{h\to 0} \frac{f(x+h)-f(x)}{h}$ la primera opción para aproximar la derivada de f en x es fijar h y tomar

$$f'(x) = \frac{f(x+h) - f(x)}{h} + O(h), \tag{7.1.1}$$

con $O(h)$ indicamos que el error que se comete es del orden de h ya que se deduce del polinomio de Taylor el error es $\frac{h}{2}f''(\xi)$. Sin embargo si escribimos el polinomio de Taylor de $f(x+h) \sim f(x)+f'(x)h+f''(x)\frac{h^2}{2}+f'''(x)\frac{h^3}{6}$ y le restamos el de $f(x-h)$ tenemos que

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} + O(h^2); \tag{7.1.2}$$

Se puede deducir las dos expresiones siguientes:

$$f'(x) = \frac{-3f(x) + 4f(x+h) - f(x+2h)}{2h} + O(h^2); \tag{7.1.3}$$

$$f'(x) = \frac{f(x-2h) - 8f(x-h) + 8f(x+h) - f(x+2h)}{12h} + O(h^4); \tag{7.1.4}$$

vamos a estudiar el comportamiento de las fórmulas anteriores para, por ejemplo, la función $exp(x)$ en $x = 1$ para distintos valores de h. El valor exacto de la derivada es $e^1 \sim 2.7182818284590452354$

Hacemos en Maxima una función que nos de una lista con los distintos valores

```
kill(der);der(f,x0,h):=[float((ev(f,x=x0+h)-ev(f,x=x0))/h),
float((ev(f,x=x0+h)-ev(f,x=x0-h))/(2*h)),
float((-3*ev(f,x=x0)+4*ev(f,x=x0+h)-ev(f,x=x0+2*h))/(2*h)),
float((ev(f,x=x0-2*h)-8*ev(f,x=x0-h)+8*ev(f,x=x0+h)-ev(f,x=x0+2*h))/(12*h))];
```

se obtiene que der(exp(x),1,.1); devuelve

[2.858841954873883, 2.722814563947418, 2.708508438360253, 2.71827275672649]

der(exp(x),1,.01); devuelve

[2.731918655787125, 2.718327133382714, 2.718190536311571, 2.718281827552956]

y der(exp(x),1,.001); devuelve

[2.719641422533226, 2.718282281505724, 2.718280921684801, 2.718281828458643]

Observa que el valor que mejor se aproxima al valor real es el que da la fórmula (7.1.4) que ya tiene un error del orden de $O(h^4)$, en contra hay que evaluar 4 veces la función; después las dos formulas (7.1.2) y (7.1.3) y de ellas suele ser mejor la primera, en general las fórmulas centradas son mejores; finalmente la peor es, con diferencias, la que sale de la definición de derivada es decir (7.1.1).

Si hacemos h muy pequeño no mejora la aproximación por ejemplo der(exp(x),1,10^-15); devuelve

[3.108624468950438,2.886579864025407,2.664535259100376,3.034609600642094]

y der(exp(x),1,10^-20); nos da

[0.0,0.0,0.0,-3700.743415417188]. Estos resultados son inaceptables y se deben a que el método está mal condicionado. Si h es muy pequeño los valores de $f(x), f(x+h), f(x-h), ...$ son muy próximos lo que genera errores por cancelación de cifras significativas (excepto que $f(x) \sim 0$), además dividimos por h. En general los métodos de derivación numérica están mal condicionados y h no debe ser pequeño en exceso. Recuerda la gráfica 2.6.1 donde comparábamos los errores de truncamiento y redondeo al calcular la derivada de $exp(x)$ en $x = 1$ para diversos valores de h.

Ejercicio: Comprueba como se comporta para distintos valores de h la derivada de $sen(x)$ en $x = 1$ y en $x = 0$. Idem de $cos(x)$ en $x = 0$ y en $x = \pi$.

Para la derivada segunda se puede deducir que

$$f''(x) = \frac{f(x-h) - 2f(x) + f(x+h)}{h^2} + O(h^2); \tag{7.1.5}$$

y para la derivada tercera

$$f'''(x) = \frac{f(x+2h) - 2f(x+h) + 2f(x-h) - f(x-2h)}{2h^3} + O(h^2); \tag{7.1.6}$$

Ejercicio: Comprueba si la fórmula (7.1.5) aproxima bien a la derivada segunda de $exp(x)$ en $x = 1$ para diversos valores de h.

2 Integración numérica

Con frecuencia es necesario evaluar la integral definida $\int_a^b f(x)\,dx$ que no tiene primitiva explícita o que no es fácil de calcular.

La primera opción es integrar un polinomio que interpole a $f(x)$ en [a,b].

Usando un punto, en este caso el punto medio, tenemos la *regla del punto medio*. La regla del punto medio es interesante porque no usa los extremos del intervalo (se dice que es una fórmula abierta) lo que permite usarla si el integrando no es continuo en los extremos del intervalo.

$$\int_a^b f(x)\,dx = (b-a)f(\frac{a+b}{2}) \tag{7.2.1}$$

Si usamos dos puntos, interpolamos la función por una recta y se tiene la *fórmula del trapecio*.

$$\int_a^b f(x)\,dx = \frac{b-a}{2}\,(f(a)+f(b)) \tag{7.2.2}$$

Si usamos tres puntos, interpolamos la función por una parábola y se tiene la *fórmula del Simpson*. La fórmula de Simpson es exacta para polinomios de grado 3 o menos.

$$\int_a^b f(x)\,dx = \frac{b-a}{6}\,(f(a)+3f(\frac{a+b}{2})+f(b)) \tag{7.2.3}$$

Como ejemplo podemos hacemos en Maxima una función que nos de una lista con los distintos valores obtenidos por los tres métodos anteriores para diversas funciones e intervalos:

```
kill(inte);inte(f,a,b):=[float(ev(f,x=(a+b)/2)*(b-a)),
float((ev(f,x=a)+ev(f,x=b))*(b-a)),
float((ev(f,x=a)+3*ev(f,x=(a+b)/2)+ev(f,x=b))*(b-a)/6)];
```

Para aproximar $\int_0^\pi sen(x)\,dx$ con valor exacto 2, inte(sin(x),0,%pi); devuelve

[3.141592653589793, 0.0, 1.570796326794897].

El valor exacto de $\int_0^1 exp(x)\,dx$ es $e-1 \sim 1.71828$ y para inte(exp(x),0,1); se obtiene

[1.648721270700128, 3.718281828459045, 1.444074273426572].

El valor exacto de $\int_0^1 \frac{dx}{1+x^2}$ es $\frac{\pi}{4} \sim 0.785398$ y para inte(1/(1+x^2),0,1); se obtiene [0.8, 1.5, 0.65].

Los errores son todos grandes porque el intervalo es grande.

Aunque podemos plantear fórmulas de integración basadas en polinomios de interpolación con más puntos, éstas no son demasiado interesantes porque, ya que los polinomios de interpolación de grado alto oscilan mucho, son propensas a errores en intervalos grandes. Es más interesante usar puntos que no estén igualmente espaciados (busca qué es la cuadratura gaussiana) o descomponer el intervalo [a,b] en intervalos más pequeños.

7.2.1 Integración compuesta

La idea básica es dividir en intervalo [a,b] en muchos intervalos más pequeños de igual longitud y aplicar a cada uno de ellos las fórmulas anteriores. Sea n el número de subdivisiones y $h = \frac{b-a}{n}$, indicamos por $x_i = a + ih$ los extremos de los intervalos [a,a+h],[a+h,a+2h],...[a+(n-1)h,b], aplicando las fórmulas anteriores y sumando los resultados (compruébalo) tenemos: Para la regla del punto medio

$$\int_a^b f(x)\,dx = h\sum_{i=0}^{n-1} f(\frac{x_i + x_{i+1}}{2}) + O(h^2) \tag{7.2.4}$$

Para el método del trapecio tenemos

$$\int_a^b f(x)\,dx = \frac{h}{2}\,(f(a) + f(b) + 2\sum_{i=1}^{n-1} f(x_i)\,) + O(h^2) \tag{7.2.5}$$

Observa que muchos puntos (todos menos el primero y el último) se repiten.

Para el método de Simpson haciendo $h = \frac{b-a}{2n}$, los puntos son $a = x_0, x_1, ..., x_{2n-1}, x_{2n} = b$ y tenemos que

$$\int_a^b f(x)\,dx = \frac{h}{3}\,(f(a) + f(b) + 2\sum_{i=1}^{n-1} f(x_{2i}) + 3\sum_{i=1}^{n} f(x_{2i-1})\,) + O(h^4) \tag{7.2.6}$$

Observa también que muchos puntos se repiten. Podemos programar en Maxima los tres métodos

kill(ipm); ipm(f,a,b,n):=

block([h:(b-a)/n],ev(h*sum(ev(f,x=a+h*(i+1/2)),i,0,n-1),numer));

kill(trapecio); trapecio(f,a,b,n):=

block([h:(b-a)/n],ev((ev(f,x=a)+ev(f,x=b)+2*sum(ev(f,x=a+h*i),i,1,n-1))*h/2,numer));

kill(simpson); simpson(f,a,b,n):= block([h:(b-a)/(2*n)],

ev((ev(f,x=a)+ev(f,x=b) +2*sum(ev(f,x=a+2*h*i),i,1,n-1) +4*sum(ev(f,x=a+(2*i+1)*h),i,0,n-1))*h/3,numer));

Veamos como se comportan para calcular $\int_0^1 exp(x)dx = e - 1 \sim 1.718281828459045$, haciendo

[ipm(exp(x),0,1,5),trapecio(exp(x),0,1,5),simpson(exp(x),0,1,5)];

tenemos [1.715421362995842, 1.724005619782788, 1.718282781924824];

[ipm(exp(x),0,1,10),trapecio(exp(x),0,1,10),simpson(exp(x),0,1,10)];

devuelve [1.717566086461128, 1.719713491389315, 1.718281888103857] y

[ipm(exp(x),0,1,20),trapecio(exp(x),0,1,20),simpson(exp(x),0,1,20)];

da como resultado

[1.718102853818907, 1.718639788925221, 1.718281832187678].

Observa que los métodos del punto medio y del trapecio son del mismo orden de exactitud y el método de Simpson genera resultados más exactos que los otros dos (aunque debes comparar simpson con $n = 10$ con trapecio y punto medio con $n = 20$ para usar el mismo h).

Ejercicio: Compara el comportamiento de los tres métodos para $\int_0^\pi sen(x)dx$ y diversos valores de n.

Una mejora de estos métodos es la *integración adaptativa* que consiste en elegir un método, por ejemplo Simpson, y un valor inicial de h, se calcula $\int_a^{a+h} f(x)\,dx$ por el método para éste valor de h y también para $\frac{h}{2}$ si los dos valores calculados están suficientemente cerca se acepta el valor de la integral en el intervalo, se acumula en el futuro resultado y se calcula la integral en el intervalo siguiente, si no se divide por 2 el valor de h y se repite el proceso. Si el valor de h llega a ser excesivamente pequeño se da un mensaje de error. Si después de varios intervalos todo va bien se prueba a ver si es aceptable multiplicar por 2 el valor de h. De esta forma se destinan más puntos a evaluar los intervalos en que es más necesario y no se tienen por que repartir los puntos por igual.

En Maxima *integrate*(expresion,x,a,b), el paquete *romberg* y las funciones *quad_qag*, *quad_qags* permiten calcular numéricamente integrales, las dos últimas de forma adaptativa; mira en la ayuda de Maxima para más información.

3 Ejercicios:

1. Utiliza las fórmulas de diferenciación numérica dadas en (7.1.1), (7.1.2), (7.1.3), (7.1.4) para estimar el valor de la derivada de $ln(x)$ en $x = 1$ con $h = 0.1, h = 0.01, h = 0.001$ y compara con el valor real $ln'(1) = 1$. Para simular errores en los datos perturba los valores de la función sumando o restando un valor aleatorio de a lo más $0.1, 0.01$ y repite el cálculo de la derivada para $h = 0.1, h = 0.01$. Repite varias veces el cálculo de la derivada aproximada, compara con el resultado exacto y deduce el efecto de errores en los datos para la estimación de la derivada. Ayuda, para sumar o restar un valor aleatorio de a lo más 0.1 puedes hacer random(0.2)-0.1; por tanto puedes hacer, para la fórmula (7.1.2), ev((log(1+h)+random(0.2)-0.1-(log(1-h)+random(0.2)-0.1))/(2*h),h=0.1,numer);

2. Utiliza los métodos de integración que hemos programado (ipm, simpson, trapecio) para $n = 10, 20$ para calcular $\int_0^1 sen(x^2)\,dx \sim 0.3102683017$. Para simular errores en los datos perturba los valores de la función sumando o restando un valor aleatorio de a lo más $0.1, 0.01$ y repite el cálculo de la integral. Deduce el efecto de errores en los datos para la estimación de la integral. Ayuda: en lugar de la función $sin(x^2)$ utiliza $sin(x^2) + random(0.2) - 0.1$ para simular un error a lo mas de 0.1.

3. Utiliza los métodos de integración que hemos programado (ipm, simpson, trapecio) para $n = 10, 20$ para calcular $\int_5^{10} sen(x^2)\,dx \sim 0.05575361876$. Compara los resultados ahora. Ayuda: La función es mucho más difícil de integrar numéricamente porque oscila mucho en este intervalo, dibújala para verlo.

4. Utiliza las rutinas del Maxima $romberg$, $quad\_qag$ y $quad\_qags$ para estimar $\int_5^{10} sen(x^2)\,dx \sim 0.0557536187643009$. Compara los resultados. Ayuda: para $quad\_qag$ debes indicar el orden de la regla, por ejemplo haciendo quad_qag(sin(x^2),x,5,10,3);. Observa que un procedimiento adaptativo da mayor exactitud.

CAPÍTULO 8

Ecuaciones diferenciales.

Índice del Tema

Casi todos los modelos científicos usan relaciones entre las funciones y sus derivadas: el crecimiento de una población, la competencia entre dos poblaciones, la resistencia de un avión, la forma de un barco, las reacciones químicas, el enfriamiento de los cuerpos, el comportamiento de una viga son modelados por ecuaciones diferenciales es decir ecuaciones en las que aparecen derivadas de las variables dependientes respecto de las independientes.

Los **Objetivos** de éste tema son:

- Entender que es una ecuación diferencial y una ecuación en derivadas parciales.
- Separar lo que son problemas de valor inicial de problemas de contorno.
- Utilizar las herramientas de Maxima para resolver algunas ecuaciones diferenciales y hallar, si es posible, soluciones a problemas de valor inicial o de contorno sencillos.

- Ser capaz de utilizar los métodos de Euler, punto medio, Runge-Kutta para resolver problemas de valor inicial, comprender la importancia del paso h y el orden del método.
- Conocer la existencia de métodos multipaso y entender los métodos predictor-corrector.
- Saber la existencia de ecuaciones rígidas y saber aplicar el método del trapecio.
- Aplicar métodos numéricos a problemas de valor inicial para sistemas de ecuaciones diferenciales y ecuaciones de orden mayor que 1.
- Conocer y aplicar el método del disparo para problemas de contorno.
- Saber aplicar el método de diferencias finitas a problemas de contorno en recintos rectangulares.

Si en la ecuación diferencial aparecen derivadas respecto de dos o mas variables independientes decimos que es una *ecuación en derivadas parciales*, por ejemplo la ecuación de ondas $\frac{\partial^2 u(x,t)}{\partial t^2} = k\frac{\partial^2 u(x,t)}{\partial x^2}$ con $k > 0$ constante o la ecuación del calor $\frac{\partial u(x,t)}{\partial t} = k\frac{\partial^2 u(x,t)}{\partial x^2}$.

Si en la ecuación sólo aparecen derivadas respecto de una variable independiente decimos que es una *ecuación diferencial ordinaria* por ejemplo la ecuación de desintegración radioactiva $y'(t) = -k\,y(t)$ con $k > 0$ constante o la ecuación logística $y'(t) = a\,y(t) - b\,y^2(t)$.

Maxima tiene la instrucción *ode2* que permite resolver algunas (pocas) ecuaciones diferenciales ordinarias. Por ejemplo para resolver $y' = y + t$ en Maxima hacemos depends(y,t); ode2(diff(y,t)=y+t,y,t);.

También puedes escribir las derivadas sin evaluar, por ejemplo ode2(x*'diff(y,x)+3*y*x=cos(x)/x,y,x). Las instrucciones de Maxima *ic1, ic2,* dada la solución general de una ecuación diferencial de primero ó segundo orden permiten hallar la solución que cumple unas determinadas condiciones iniciales (también puedes resolver el sistema obtenido sustituyendo las condiciones iniciales para hallar los valores de las constantes). *dsolve* resuelve sistemas de ecuaciones diferenciales ordinarias lineales.

De otro lado, la mayor parte de las funciones matemáticas se pueden definir a partir de ecuaciones diferenciales, por ejemplo la función $exp(t)$ satisface $y' = y$; las funciones $sen(t), cos(t)$ satisfacen $y'' - y' = 0$ así es frecuente que no podamos (o sepamos) resolver en forma explícita una ecuación diferencial.

En general, al resolver una ecuación diferencial aparecen constantes arbitrarias por lo que si queremos que la solución esté determinada es preciso dar condiciones adicionales, por ejemplo $y = exp(t + c)$ es solución de $y' = y$ para todo c y tenemos que pedir, por ejemplo que $y(0) = 1$ para que la solución de $y' = y$ sea solamente $y = exp(t)$. La solución de $y'' = -y$ es $c_1 cos(t) + c_2 sen(t)$ para toda constante c_1, c_2 y si queremos que la solución sea $sen(t)$ podemos fijar por ejemplo que $y(0) = 0, y'(0) = 1$. Desde el punto de vista científico es bueno saber cual va a ser el comportamiento del fenómeno modelado por la ecuación diferencial a partir de unas condiciones. Si todas las condiciones que se imponen aparecen para el mismo valor de la variables independiente decimos que estamos en un *problema de valores iniciales*. Si las condiciones que se imponen aparecen para distintos valores de la variable independiente decimos que estamos en un *problema de contorno*.

1 Valores iniciales

Vamos a estudiar métodos numéricos para resolver la ecuación $y' = f(y, t)$ con la condición inicial $y(t_0) = y_0$. Para garantizar que este problema tiene solución única basta que f y $\frac{\partial f}{\partial t}$ sean continuas en un entorno

del punto (t_0, y_0) (aunque hay condiciones más generales).

Es costumbre indicar por h el paso, $t_i = t_0 + i\,h$ y escribir los valores aproximados que se obtienen de la ecuación para t_i como w_i para distinguirlos de los valores exactos y_i.

8.1.1 Método de Euler

El método más sencillo es el *método de Euler*. Ya que por el polinomio de Taylor $y(t+h) \sim y(t) + y'(t)\,h$ sustituyendo por la ecuación diferencial $y' = f(t,y)$ tenemos que $y(t+h) \sim y(t) + h\,f(t,y)$. Partiendo de la condición inicial $w_0 = y(t_0) = y_0$ el método de Euler está dado por

$$w_{i+1} = w_i + h\,f(t_i, w_i); \tag{8.1.1}$$

Por ejemplo la solución de $y' = y + t$ con $y(0) = 1$ es $y = 2\,exp(t) - t - 1$ (compruébalo). Si elegimos $h = 0.1$ y aplicamos el método de Euler a $y' = y + t$ con $y(0) = 1$ entonces $t_0 = 0$, $w_0 = y_0 = 1$ aplicando la fórmula tenemos que $w_1 = 1 + 0.1(1 + 0) = 1.1$; $w_2 = 1.1 + 0.1(1.1 + .1) = 1.22$ etc. asi que la solución aproximada pasa por $(0.1, 1.1)$, $(0.2, 1.22)$ etc. los valores exactos son $y(0.1) \sim 1.110341836151295$, $y(0.2) \sim 1.24280551632034$.

Podemos programar en Maxima el método de euler para $y' = f(t,y)$ definiendo donde el argumento es una lista formada por $t_i, w_i, f(y,t), h$ haciendo

eu1(arg):=block([ti:arg[1],wi:arg[2],ff:arg[3],hh:arg[4]], [ti+hh,wi+hh*ev(ff,y=wi,t=ti,numer),ff,hh]);

para aplicarlo a la ecuación $y' = y + t$, partiendo de $y(0) = 1$ con paso $h = .1$ hacemos sucesivamente

eu1([0,1,y+t,.1]); que da [0.1,1.1,y+t,0.1] después eu1(%) da

[0.2,1.22,y+t,0.1]; después eu1(%) que da [0.3,1.362,y+t,0.1] etc. así que las estimaciones por el método de euler de la solución que pasa por $y(0) = 1$ son $y(0.1) = 1.1$, $y(0.2) = 1.22$ e $y(0.3) = 1.362$;

Si queremos repetir el método de euler varias veces (pongamos 5) y guardar en una lista sólo los valores de $[t_i, w_i]$ por ejemplo para pintarlos, recordando que endcons añade un elemento al final de una lista podemos hacer

valor:[0.,1.,y+t,.1];listaeu:[[0.0,1.0]]; for i:1 thru 10 do (valor:eu1(valor),listaeu:endcons(valor,listaeu));

y en la variable listaeu queda [[0,1],[0.1,1.1],[0.2,1.22],[0.3,1.362],[0.4,1.5282],[0.5,1.72102]]

Por supuesto podemos definir f, h fuera y hacer que el programa lleve la cuenta sólo de t_i, w_i, por ejemplo,

ff:y+t; hh:0.1; kill(eu2); eu2(arg):=block([ti:arg[1],wi:arg[2]], [ti+hh,wi+hh*ev(ff,y=wi,t=ti,numer)]); /* y se empieza con*/ eu2([0.,1.]);

Ejercicio: Calcula 25 aproximaciones por el método de Euler $w_1, w_2, ..., w_{25}$ de la solución de la ecuación $y' = y + t$ con $h = 0.1$, a partir de $y(0) = 1$. Dibuja la solución exacta y los puntos obtenidos. Observa que los errores crecen pero de forma controlada, desafortunadamente es lo que suele ocurrir. Ayuda: una vez calculados los valores, prueba algo así como

listaeu:[[0.0,1.0]];valor:[0.0,1.0]; for i:1 thru 25 do (valor:eu2(valor),listaeu:endcons(valor,listaeu));

plot2d([2*exp(t)-t-1,[discrete,listaeu]],[t,0,2.5],[style,[lines,1,3],[points,1,2]]);

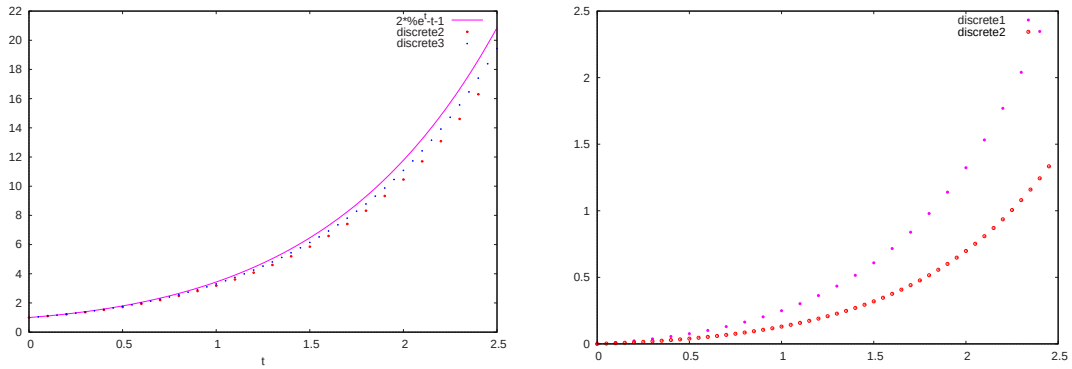

Figura 8.1.1: Aproximaciones por Euler y errores con paso $h = 0.1, 0.05$

Ejercicio: Calcula ahora 50 aproximaciones por el método de Euler $w_1, w_2, ..., w_{50}$ de la solución de la ecuación $y' = y + t$ con $h = 0.05$, a partir de $y(0) = 1$. Dibuja la solución exacta y los puntos obtenidos en el ejercicio anterior y ahora. Debes generar la primera de las gráficas de 8.1.1. Observa gráficamente cuando son los errores mayores. Calcula los errores para los pasos $h = 0.1$ y $h = 0.05$. Ayuda: prueba algo así como

```
listerr1:makelist([listaeu[j][1],ev(abs(2*exp(t)-t-1-listaeu[j][2]),t=listaeu[j][1])],j,1,25);
```

para ver los errores que corresponden a $h = .1$ Dibuja los errores en cada punto para $h = 0.1$ y $h = 0.05$. Debes generar la segunda de las gráficas de 8.1.1. Observa que los errores dependen del valor de h.

8.1.2 Métodos de orden 2

Para medir el error que se comete en cada paso se introduce el *error de truncamiento local* que mide hasta que punto la solución exacta de la ecuación diferencial satisface la formula con que se obtiene la aproximación w_{i+1}. Se puede probar que el error de truncamiento local del método de euler es de orden h por lo que en el ejercicio anterior debiste observar que si reduces el paso de h a $\frac{h}{2}$ el error se reduce aproximadamente a la mitad. Es muy conveniente hallar métodos en que el error de truncamiento local sea de orden h^2 o incluso de orden h^4 para que con h mas grande (y por tanto menos puntos de cálculo) se tengan resultados más exactos. El precio a pagar es que estos métodos necesitan más evaluaciones de la función en cada paso.

Partiendo de la condición inicial $w_0 = y(t_0) = y_0$ existen varios métodos de orden 2 entre ellos:

el método del *punto medio* está dado por

$$w_{i+1} = w_i + h\, f(t_i + \frac{h}{2}, w_i + \frac{h}{2} f(t_i, w_i)); \tag{8.1.2}$$

el método del *Euler modificado* está dado por

$$w_{i+1} = w_i + \frac{h}{2}\,(\, f(t_i, w_i) + f(t_{i+1}, w_i + h\, f(t_i, w_i))\,); \tag{8.1.3}$$

el método del *Heun* está dado por

$$w_{i+1} = w_i + \frac{h}{4}\,(\, f(t_i, w_i) + 3f(t_i + \frac{2}{3}h, w_i + \frac{2}{3}h\, f(t_i, w_i))\,); \tag{8.1.4}$$

Un programa en Maxima del método del punto medio para $y' = f(t, y)$ con argumento una lista formada por $t_i, w_i, f(y, t), h$ puede ser

kill(odepm);odepm(arg):=block([ti:arg[1],wi:arg[2],ff:arg[3],hh:arg[4],tem],

tem:ev(ff,y=wi,t=ti,numer), [ti+hh,wi+hh*ev(ff,y=wi+hh/2*tem,t=ti+hh/2,numer),ff,hh]);

De nuevo

valor:[0.,1.,y+t,.1];listapm:[[0.0,1.0]];

for i:1 thru 5 do (valor:odepm(valor),listapm:endcons([valor[1],valor[2]],listapm));

nos devuelve una lista con los valores de $w_1, w_2, ...w_5$ por el método del punto medio para la ecuación $y' = y + t$. Observa que si queremos pintar los puntos tenemos que guardar en la lista sólo los pares $[t_i, w_i]$ y no los valores de f, h. Esto puede hacerse, si los valores están en una lista llamada listapm, con

lispun1:[];for i:1 thru length(listapm) do lispun1:endcons([listapm[i][1],listapm[i][2]],lispun1);

para guardar sólo los dos primeros elementos.

De forma alternativa podemos sacar fuera el valor de f, h y transmitir sólo los valores de t_i, w_i como hicimos en $eu2$, por ejemplo definimos

kill(odepm2);odepm2(arg):=block([ti:arg[1],wi:arg[2],tem],

tem:ev(ff,y=wi,t=ti,numer), [ti+hh,wi+hh*ev(ff,y=wi+hh/2*tem,t=ti+hh/2,numer)]);

y hay que guardar en la variable hh el valor del paso y en la variable ff la función f(t,y) antes de empezar. Por ejemplo, se puede hacer

hh:0.05;ff:y+t; valor:[0.,1.];listapm:[[0.0,1.0]];

for i:1 thru 50 do (valor:odepm2(valor),listapm:endcons(valor,listapm));

Ejercicio: Calcula con $h = 0.05$ un total de 50 valores de w_i por el método del punto medio para la ecuación $y' = y + t$ con $y(0) = 1$, dibuja la solución y las aproximaciones por los métodos de Euler y del punto medio y observa cuál método es mejor. La gráfica que obtengas debe ser la primera de 8.1.2.

Un programa para el método de Euler modificado puede ser

kill(odeem);odeem(arg):=block([ti:arg[1],wi:arg[2],ff:arg[3],hh:arg[4],tem], tem:ev(ff,y=wi,t=ti,numer),

[ti+hh,wi+hh/2*(tem+ev(ff,y=wi+hh*tem,t=ti+hh,numer)),ff,hh]);

y otro para el método de Heun

kill(odeheun);odeheun(arg):=block([ti:arg[1],wi:arg[2],ff:arg[3],hh:arg[4],tem], tem:ev(ff,y=wi,t=ti,numer),

[ti+hh,wi+hh/4*(tem+3*ev(ff,y=wi+hh*2/3*tem,t=ti+hh*2/3,numer)),ff,hh]);

Ejercicio: Crea una lista con 25 valores aproximados entre $t = 0, t = 2.5$ de la solución de $y' = y + t$ con $y(0) = 1$ tomando $h = .1$ por los métodos de punto medio, euler modificado y Heun y comprueba si los errores son similares. Dibuja la solución exacta y las aproximaciones halladas. Calcula listas con los errores que corresponden a cada método y dibuja los errores en cada punto. Observa que para dibujar los puntos tienes que guardar en la lista sólo los valores de t_i, w_i o definir fuera f, h y usar una lista con sólo los valores

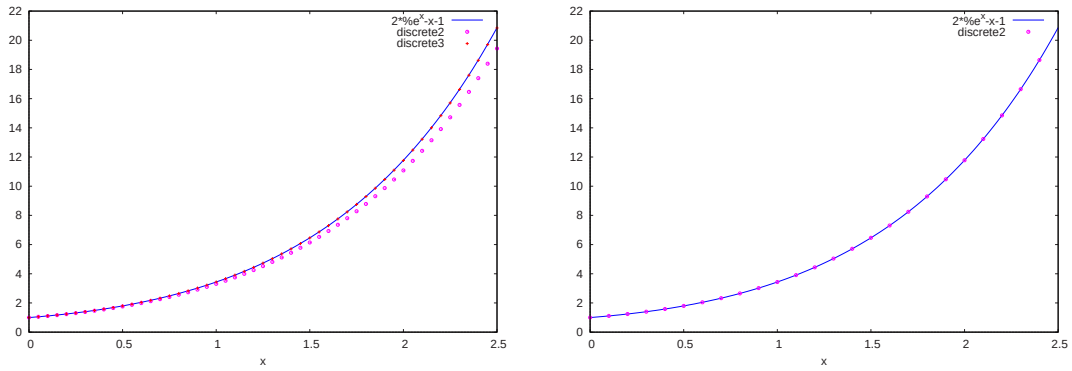

Figura 8.1.2: Método de Euler y punto medio con h=0.05; Runge-Kutta con h=0.1

de t_k, w_k.

Ejercicio: Repite los cálculos del ejercicio anterior con paso $h = 0.05$ y estudia si los errores de Heun, Euler modificado y punto medio son aproximadamente la mitad o la cuarta parte del ejercicio anterior.

Ejercicio: Comprueba si la ecuación $y' = t\,y + t$ con la condición $y(1) = 2$ tiene solución $3\,exp(\frac{1}{2}(t^2 - 1)) - 1$. Halla soluciones numéricas por el método de Euler y uno de los métodos anteriores y estudia el comportamiento de los errores.

8.1.3 Runge-Kutta de orden 4 y Runge-Kutta-Fehlberg

Es posible y deseable tener métodos de orden más alto, en concreto el siguiente método es de orden 4, observa que en su definición usamos 4 variables temporales

$$\begin{array}{ll} k_1 = h\,f(t_i, w_i); & k_2 = h\,f(t_i + \frac{h}{2}, w_i + \frac{k_1}{2}); \\ k_3 = h\,f(t_i + \frac{h}{2}, w_i + \frac{k_2}{2}); & k_4 = h\,f(t_i + h, w_i + k_3); \end{array} \qquad (8.1.5)$$

$$w_{i+1} = w_i + \frac{1}{6}\,(\,k_1 + 2k_2 + 2k_3 + k_4\,);$$

Un programa para el método de Runge-Kutta de orden 4 puede ser

```
kill(rk4);

rk4(arg):=block([ti:arg[1],wi:arg[2],ff:arg[3],hh:arg[4],k1,k2,k3,k4],

k1:hh*ev(ff,y=wi,t=ti,numer),k2:hh*ev(ff,y=wi+k1/2,t=ti+hh/2,numer),

k3:hh*ev(ff,y=wi+k2/2,t=ti+hh/2,numer),k4:hh*ev(ff,y=wi+k3,t=ti+hh,numer),

[ti+hh,wi+(k1+2*k2+2*k3+k4)/6,ff,hh]);
```

De nuevo puedes definir fuera ff, hh donde están los valores de $f(t, y), h$ y hacer que el método sólo lleve dos valores, t_k, w_k con lo cual la salida es buena para dibujar.

Ejercicio: Crea una lista con 25 valores aproximados entre $t = 0, t = 2.5$ de la solución de $y' = y + t$ con $y(0) = 1$ tomando $h = 0.1$ por el método de Runge-Kutta de orden 4 $rk4$ y comprueba con los valores

exactos de la solución. Dibuja la solución exacta y las aproximaciones halladas. Debes obtener la segunda de las gráficas de la figura 8.1.2. Observa que la aproximación es muy buena. Calcula una lista con los errores y dibuja los errores.

Ejercicio: Repite los cálculos del ejercicio anterior con paso $h = 0.05$ y estudia cómo son los errores ahora. Repite el proceso con otra ecuación, por ejemplo $y' = t\,y + t$ con la condición $y(1) = 2$ tiene solución $3\,exp(\frac{1}{2}(t^2-1)) - 1$.

En el paquete *dynamics* de Maxima se incorpora la función *rk* que resuelve numéricamente una ecuación diferencial (o un sistema de ecuaciones diferenciales) de primer orden por el método de Runge-Kutta de orden 4. Recuerda que si la ecuación diferencial está escrita en la forma $y' = f(y,t)$ con la condición inicial $y(t_0) = y_0$ para calcular los valores de w_i desde t_0 hasta $t = b$ con paso h es $rk(f(y,t), y, y_0, [t, t_0, b, h])$; y devuelve una lista de los puntos $[t_i, w_i]$. Por ejemplo para resolver numéricamente $y' = cos(y) + t + 1$ con $y(1) = 2$ desde $t = 1$ hasta $t = 4$ con $h = .05$ se escribe $rk(cos(y) + t + 1, y, 2, [t, 1, 4, .05])$;

La idea del método de Runge- Kutta- Fehlberg es, como en integración adaptativa, usar paso variable. Se usan a la vez dos métodos de Runge-Kutta uno de cuarto orden y otro de quinto, si las valores obtenidos son suficientemente próximos aceptamos el valor y buscamos el siguiente, si no reducimos el paso h y probamos de nuevo. Si después de varios puntos con el mismo paso todo va bien probamos a aumentar el paso. Si tenemos que reducir mucho el paso damos un mensaje de error y paramos.

La importancia de Runge- Kutta- Fehlberg radica en que se usan evaluaciones comunes de f, en concreto sólo 6, para calcular los valores de Runge-Kutta de cuarto orden y de quinto, por lo que ahorra mucho cálculo. La definición del método es:

$$\begin{array}{ll} k_1 = h\,f(t_i, w_i); & k_2 = h\,f(t_i + \frac{h}{4}, w_i + \frac{1}{4}k_1); \\ k_3 = h\,f(t_i + \frac{3h}{8}, w_i + \frac{3}{32}k_1 + \frac{9}{32}k_2); & k_4 = h\,f(t_i + \frac{12h}{13}, w_i + \frac{1932}{2197}k_1 - \frac{7200}{2197}k_2 + \frac{7296}{2197}k_3); \\ \multicolumn{2}{c}{k_5 = h\,f(t_i + h, w_i + \frac{439}{216}k_1 - 8k_2 + \frac{3680}{513}k_3 - \frac{845}{4104}k_4);} \\ \multicolumn{2}{c}{k_6 = h\,f(t_i + \frac{h}{2}, w_i - \frac{8}{27}k_1 + 2k_2 - \frac{3544}{2565}k_3 + \frac{1859}{4104}k_4 - \frac{11}{40}k_5);} \\ \multicolumn{2}{c}{w^{(4)}_{i+1} = w_i + \frac{25}{216}k_1 + \frac{1408}{2565}k_3 + \frac{2197}{4104}k_4 - \frac{1}{5}k_5;} \\ \multicolumn{2}{c}{w^{(5)}_{i+1} = w_i + \frac{16}{135}k_1 + \frac{6656}{12825}k_3 + \frac{28561}{56430}k_4 - \frac{9}{50}k_5 + \frac{2}{55}k_6;} \end{array} \qquad (8.1.6)$$

donde representamos con $w^{(4)}_{i+1}, w^{(5)}_{i+1}$ los valores obtenidos por Runge-Kutta de orden 4 y 5 respectivamente. Observa que sólo hemos evaluado la función f en 6 puntos que sirven para los dos métodos.

Aunque la implantación del programa adaptativo es complicada (hay que decidir cuando se cambia h), la programación en Maxima de una subrutina que devuelva sólo una lista con los dos valores de Runge-Kutta 4 y 5 es sencilla, suponemos que $hh = h$, $ti = t_i$ y $ff = f(t,y)$ están definidos fuera de la rutina y el único argumento es el valor de $wi = w_i$. Haciendo

```
kill(rkf);

rkf(wi):=block([k1,k2,k3,k4,k5,k6], k1:hh*ev(ff,t=ti,y=wi,numer),

k2:hh*ev(ff,t=ti+hh/4,y=wi+1/4*k1,numer),

k3:hh*ev(ff,t=ti+3*hh/8,y=wi+3/32*k1+9/32*k2,numer), k4:hh*ev(ff,t=ti+12*hh/13,y=wi+1932/2197*k1-
7200/2197*k2+7296/2197*k3,numer),

k5:hh*ev(ff,t=ti+hh,y=wi+439/216*k1-8*k2+3680/513*k3-845/4104*k4,numer),

k6:hh*ev(ff,t=ti+hh/2,y=wi-8/27*k1+2*k2-3544/2565*k3+1859/4104*k4-11/40*k5,numer),
```

[wi+25/216*k1+1408/2565*k3+2197/4104*k4-k5/5,wi+16/135*k1+6656/12825*k3+28561/56430*k4

-9/50*k5+2/55*k6]);

nos falta definir, por ejemplo $hh = 0.1, ti = 0, ff = y+t$, para $y(0) = 1$ la solución es $y = 2exp(t) - t - 1$, el valor exacto es $y(.1) \sim 1.1103418361512952496$ y rkf(1.); devuelve la lista con los valores de R-K-4 y R-K-5 [1.110341858974359,1.110341834294872] que difieren en $\sim 2 * 10^{-8}$, si aceptamos que la estimación del error es buena hacemos que ti=ti+hh, guardamos el valor de ti y el segundo valor (que debería ser el más exacto), y llamamos a rkf con el valor aceptado, si no aceptamos el valor hacemos $hh = hh/2$, si este valor de h es demasiado pequeño damos un mensaje de error y paramos, si el valor de h está dentro de lo aceptable llamamos de nuevo a rkf(1.) y sigue el proceso. Así hasta que lleguemos a b si queremos una aproximación numérica de la solución en el intervalo $[t_0, b]$.

8.1.4 Métodos multipaso

Todos los métodos anteriores se basan sólo en el valor de w_i para estimar el de w_{i+1}. Existen también los *métodos multipaso* que usan los valores de $w_i, w_{i-1}, ...$ para estimar el valor de w_{i+1}. Existen métodos multipaso explícitos, en los que y_{k+1} esta despejada, llamados de *Adams-Bashforth;* los de orden 1 (que es el método de Euler), 2 y 3 son:

$$\begin{array}{c} w_{k+1} = w_k + h\, f(t_k, w_k); \\ w_{k+1} = w_k + \frac{h}{2}(3f(t_k, w_k) - f(t_{k-1}, w_{k-1})); \\ w_{k+1} = w_k + \frac{h}{12}(23f(t_k, w_k) - 16f(t_{k-1}, w_{k-1}) + 5f(t_{k-2}, w_{k-2})); \end{array} \tag{8.1.7}$$

y métodos multipaso implícitos, en los que y_{k+1} no está despejada, llamados de *Adams-Moulton;* los de orden 1, 2 y 3 son:

$$\begin{array}{c} w_{k+1} = w_k + f(t_{k+1}, w_{k+1}); \\ w_{k+1} = w_k + \frac{h}{2}(f(t_{k+1}, w_{k+1}) + f(t_k, w_k)); \\ w_{k+1} = w_k + \frac{h}{12}(5f(t_{k+1}, w_{k+1}) + 8f(t_k, w_k) - f(t_{k-1}, w_{k-1})); \end{array} \tag{8.1.8}$$

Al método de orden 1 se le llama *método de Euler hacia atrás (backward)* y al método de orden 2 se le llama *método del trapecio*. Los métodos de Adams-Moulton requieren la solución de una ecuación no lineal en cada caso, sin embargo, si h es suficientemente pequeño usualmente un par de iteraciones por punto fijo convergen a la solución. Lo usual es seguir un esquema *predictor-corrector* eligiendo un método implícito y otro explícito del mismo orden. Dado h, se usa un método explícito para obtener una aproximación de w_{i+1} (la predicción), con ella se usa un esquema de punto fijo con el método implícito unas cuantas veces (ahora se corrige el valor). La diferencia entre las sucesivas iteraciones sirve para estimar el error y poder usar, si queremos, un esquema adaptativo, si el error es grande tomamos $h = h/2$ etc..

Por ejemplo vamos a aplicar a la ecuación $y' = y + t$ con las condiciones iniciales $y(0) = 1$ y $h = .1$ un método predictor corrector de orden 2. Para calcular w_{k+1} necesitamos saber w_k y w_{k-1} por lo que no nos basta la condición inicial sino que necesitamos un valor más, así que, para poder arrancar, en primer lugar necesitamos calculamos con buena precisión, por ejemplo con Runge-Kutta de orden 4, el valor en $t = 0.1$

Usando el programa Maxima que implantamos en (8.1.5) rk4([0,1,y+t,.1]); nos da una valor aproximado de $y(.1) = 1.110341666666667$, que tiene un error de $\sim 1.7\,10^{-7}$; a continuación hacemos en Maxima $hh : .1; ff : y + t; tk : 0.1; w0 : 1; w1 : 1.110341666666667$ por la primera fórmula de (8.1.7) calculamos w1+hh/2*(3*ev(ff,t=tk+hh,y=w1,numer)-ev(ff,t=tk,y=w0,numer)); que nos da un valor estimado

de $y(.2) = 1.251892916666667$ ésta es la predicción. El valor exacto es ~ 1.24280551632034 y el error es del orden de 10^{-2}.

Para la primera fórmula de (8.1.8) definimos

```
kill(am);

am(wkm1,wk,tk,hh,ff):=wk+hh/2*(ev(ff,y=wkm1,t=tk+hh,numer)+

ev(ff,y=wk,t=tk,numer));
```

Con las definiciones anteriores de wk,tk,hh,ff en Maxima tenemos que

vcorregido:am(1.251892916666667,w1,tk,hh,ff); devuelve 1.243453395833334; esta es la primera corrección, observa que está más cerca del valor exacto.

vcorregido:am(vcorregido,w1,tk,hh,ff); devuelve 1.243031419791667; ésta es la segunda correción.

Aplicando de nuevo vcorregido:am(vcorregido,w1,tk,hh,ff); tenemos 1.243010320989584; (el error de este valor es $\sim 2.3\,10^{-4}$). Observa que estamos usando métodos de orden 2 y con un valor de $h = .1$ tenemos errores del orden de 10^{-4}. Ahora podemos repetir el proceso hasta que, por ejemplo, la distancia entre dos valores corregidos sucesivos sea suficientemente pequeña. Por supuesto, si no se alcanza la distancia prescrita en un número máximo de veces se devuelve un mensaje de error y se para; también puede implementarse un algoritmo adaptativo y reducir el valor de h.

Para implementar en Maxima una versión simple de predictor corrector de orden 2 podemos calcular el primer punto con Runge Kutta para poder arrancar con los puntos w_k, w_{k-1} y luego usamos un predictor y 2 veces corrector siempre (sin controlar el error que sería lo deseable).

Ahora hacemos en Maxima

```
/*los argumentos son t_k el t del último punto calculado, w_k el último punto calculado, w_{k-1} el penúltimo
punto calculado, la función f(t,y), h */

kill(pc2); pc2(arg):=block([tk:arg[1],wk:arg[2],wk1:arg[3],ff:arg[4],hh:arg[5],wfin],

wfin:wk+hh/2*(3*ev(ff,t=tk,y=wk,numer)-ev(ff,t=tk-hh,y=wk1,numer)),

/* la variable temporal wfin ahora tiene el valor del predictor */

wfin:wk+hh/2*(ev(ff,y=wfin,t=tk+hh,numer)+ev(ff,y=wk,t=tk,numer)),

/* la variable temporal wfin ahora tiene el valor corregido una vez */

wfin:wk+hh/2*(ev(ff,y=wfin,t=tk+hh,numer)+ev(ff,y=wk,t=tk,numer)),

/* la variable temporal wfin ahora tiene el valor corregido dos veces, damos la salida observa que ahora wk
se desplaza a wk1 */

[tk+hh,wfin,wk,ff,hh]  );
```

Una alternativa más sencilla es usar un predictor de orden n-1 y un corrector de orden n, nuestro caso podemos usar como predictor el método de Euler (que solo necesita un punto para arrancar) y luego el método del trapecio como corrector, de esta forma nos ahorramos usar Runge - Kutta al principio.

Ejercicio: Para la ecuación $y' = y + t$ con las condiciones iniciales $y(0) = 1$ y $h = .1$ utiliza el método

predictor corrector de orden 2 para calcular el valor de $y(1)$.

Ejercicio: Implementa en Maxima el algoritmo para aplicar el método de Euler como predictor y el del trapecio como corrector, usando dos veces corrector.

Ejercicio: Haz una subrutina en Maxima que repita el corrector hasta que el valor absoluto de una estimación del error absoluto (¿Que tal la diferencia entre dos valores corregidos sucesivos?) en cada paso sea menor que una tolerancia.

Además pueden usarse en el algoritmo métodos de diverso orden, por ejemplo al principio un método predictor- corrector de orden 1, cuando se conocen dos puntos método predictor- corrector de orden 2, luego de orden 3, hasta por ejemplo de orden 4, si se baja el paso h se baja también el orden del método.

Los métodos predictor - corrector de paso variable se usan mucho en la práctica ya que las diferencias entre sucesivas aplicaciones del corrector permiten estimar el error en cada punto.

Los métodos multipaso son más estables cuanto más bajo es el orden y los métodos implícitos son más estables que los explícitos.

8.1.5 Ecuaciones rígidas

Vamos a aplicar dos métodos de orden 2: punto medio y trapecio y el método Runge-Kutta de orden 4 a la ecuación $y' = \frac{1-y\,t-(y\,t)^2}{t^2}$ con la condición inicial $y(1) = 1$ y solución exacta $y = \frac{1}{t}$ con diversos valores de h y comparar el valor obtenido por cada método en $t = 2$. el valor exacto es $\frac{1}{2} = 0.5$.

Recuerda anteriormente hemos implementado $odepm$, hacemos

```
val:[1,1,(1-y*t-y*y*t*t)/(t*t),.1]; /*y después*/

for i:1 thru 10 do val:odepm(val);

/*resulta que el valor aproximado para t = 2 es 0.50102959416826*/

/* si hacemos h=0.01 */

val:[1,1,(1-y*t-y*y*t*t)/(t*t),.01];

/*y después*/ for i:1 thru 100 do val:odepm(val);

/*resulta que el valor aproximado para t = 2 es 0.50000880698681*/
```

Repetimos el proceso para $rk4$.

Para predictor corrector calculamos el primer punto con Runge Kutta y luego usamos un predictor y 2 veces corrector siempre (sin controlar el error). Para $h = .1$ rk4 nos da una estimación para $w(1.1) = 0.90909313081282$

usando en Maxima la subrutina $pc2$ que definimos antes, para arrancar hacemos

```
val:[1.1,0.90909313081282,1,(1-y*t-y*y*t*t)/(t*t),.1];

for i:1 thru 9 do val:pc2(val);
```

y el segundo valor de val es 0.49965457241174.

Para $h = 0.01$ el valor estimado en 1.1 por Runge Kutta es 0.99009900992936; arrancamos con

val:[1.01,0.99009900992936,1,(1-y*t-y*y*t*t)/(t*t),.01];

y hacemos for i:1 thru 99 do val:pc2(val);

el valor obtenido es ahora 0.49999573301848.

En resumen para la ecuación $y' = \frac{1-y\,t-(y\,t)^2}{t^2}$ tenemos:

$Metodo$	h	$w(2)$	$error$
$Punto\,medio$	0.1	0.50102959416826	$\sim 10^{-3}$
$Predictor\,Corrector\,2$	0.1	0.49965457241174	$\sim 3.4 * 10^{-4}$
$Runge\,Kutta\,4$	0.1	0.50000155409217	$\sim 1.6 * 10^{-6}$
$Punto\,medio$	0.01	0.50000880698681	$\sim 10^{-5}$
$Predictor\,Corrector\,2$	0.01	0.49999573301848	$\sim 4 * 10^{-6}$
$Runge\,Kutta\,4$	0.01	0.50000000013855	$\sim 1.4 * 10^{-10}$

Observa que todos los métodos funcionan bien, el método de orden 4 es mucho más exacto que los métodos de orden 2, dentro de ellos predictor-corrector funciona mejor que punto medio y dividir h por 10 casi duplica los dígitos exactos en base 10. La ecuación diferencial es buena.

Aplicamos ahora los mismos métodos a la ecuación $y' = 3y - 4exp(-t)$ con la condición $y(0) = 1$ que tiene solución exacta $y = exp(-t)$ pero la solución general es $y = exp(-t) + c_1\,exp(3t)$, los errores de redondeo activan el término $exp(3t)$ con resultados ruinosos.

Se tiene que y(3)=exp(-3)=0.049787068367863942979;

Ahora hacemos val:[0,1,3*y-4*exp(-t),.1]; y repetimos el bucle de $odepm$ 30 veces y luego val:[0,1,3*y-4*exp(-t),.01]; y repetimos el bucle de $odepm$ 300 veces. Igual hacemos con $rk4$.

Para predictor corrector con $h = 0.1$ después de estimar el valor en .1 por Runge-Kutta empezamos con val:[0.1,0.90483429367721,1,3*y-4*exp(-t),.1]; y hacemos 29 iteraciones de pc2 obtenemos -1.023911982760934

Para predictor corrector con $h = 0.01$ después de estimar el valor en .01 por Runge-Kutta empezamos con

val:[0.01,0.99004983371742,1,3*y-4*exp(-t),.01];

hacemos 299 iteraciones de pc2 y obtenemos 0.033578345952815

En resumen para la ecuación $y' = 3y - 4exp(-t)$ tenemos:

$Metodo$	h	$w(3)$	$error$
$Punto\,medio$	0.1	-5.388926856180671	~ 5.5
$Predictor\,Corrector\,2$	0.1	-1.023911982760934	~ 1
$Runge\,Kutta\,4$	0.1	-0.0070776390244746	~ 0.06
$Punto\,medio$	0.01	-0.016897763210128	~ 0.07
$Predictor\,Corrector\,2$	0.01	0.033578345952815	~ 0.02
$Runge\,Kutta\,4$	0.01	0.04978070126978	$\sim 10^{-5}$

Observa que los métodos funcionan en general mal, sólo Runge Kutta de orden 4 con paso pequeño da resultados aceptables, pero mucho peores que en la ecuación anterior. De nuevo predictor corrector funciona mejor que punto medio. La razón es que la ecuación diferencial está mal condicionada. Si cambiamos ligeramente la condición inicial la solución cambiará mucho por la parte en $exp(3t)$. Cualquier error hace

que el coeficiente c_1 de la solución general $y = exp(-t) + c_1\, exp(3t)$ se haga distinto de cero con lo que el término $exp(3t)$ domina al término $exp(-t)$ y hace crecer los errores exponencialmente.

La definición de ecuación *rígida* (en inglés stiff) es bastante técnica. Si la ecuación $y' = f(t, y)$ se linealiza es decir se escribe como $f(t, y) = cy + g(t, y)$ entonces la ecuación es rígida si c es negativo y grande, en el caso de sistemas de ecuaciones diferenciales c es la matriz jacobiana de f y tiene algún autovalor con parte real negativa y grande. Esta definición intuitivamente refleja el hecho de que alguna de las soluciones cambia rápidamente respecto de las otras o bien alguna solución cambia rápidamente en un intervalo y no en otro. Físicamente corresponde a la existencia de soluciones transitorias, que toman valores grandes en un intervalo pequeño y luego prácticamente desaparecen.

Las ecuaciones rígidas son difíciles de resolver numéricamente y exigen un paso h muy pequeño o algoritmos especiales, por ejemplo el método del trapecio $w_{k+1} = w_k + \frac{h}{2}(f(t_{k+1}, w_{k+1}) + f(t_k, w_k))$ que es bueno en cuanto a estabilidad. A partir de t_k, t_{k+1}, w_k se calcula w_{k+1} por el método de Newton o de la secante a partir, por ejemplo, de la estimación dada por w_k o bien por un método explícito. Usualmente un par de iteraciones bastan, aunque es conveniente repetir el proceso hasta que las sucesivas aproximaciones sean menores que una tolerancia.

Recuerda que el método de Newton para $g(x) = 0$ estaba dado por $x_{n+1} = x_n - g(x_n)/g'(x_n)$. Escribiendo la fórmula del método del trapecio como $F(w_{k+1}) = w_{k+1} - w_k - \frac{h}{2}(f(t_{k+1}, w_{k+1}) + f(t_k, w_k)) = 0$ tenemos que $\frac{\partial F}{\partial w_{k+1}} = 1 - \frac{h}{2} f_y(t_{k+1} w_{k+1}))$ y las sucesivas iteraciones $w_{k+1}^{(n)}$ deducidas por la fórmula de Newton son

$$w_{k+1}^{(n)} = w_{k+1}^{(n-1)} - \frac{w_{k+1}^{(n-1)} - w_k - \frac{h}{2}(f(t_{k+1}, w_{k+1}^{(n-1)}) + f(t_k, w_k))}{1 - \frac{h}{2} f_y(t_{k+1}, w_{k+1}^{(n-1)}))} \tag{8.1.9}$$

Ya que la parte de la fórmula $w_k + \frac{h}{2} f(t_k, w_k)$ depende sólo de t_k, w_k y se va a calcular varias veces podemos guardarla en una variable y también usarla como arranque del método de Newton. Observa también que la fórmula puede simplificarse sacando denominador común y usando una variable temporal para guardar el valor de $f(t_{k+1}, w_{k+1}^{(n-1)})$ y no calcularlo 2 veces, además se puede repetir Newton hasta que la diferencia entre dos valores sucesivos sea menor en valor absoluto que una tolerancia, etc..

En Maxima podemos hacer

```
kill(mettrap);mettrap(arg):= block([tk:arg[1],wk:arg[2],ff:arg[3],hh:arg[4],yder,wfin,fwk],
yder:diff(ff,y),fwk:wk+hh/2*ev(ff,t=tk,y=wk,numer),wfin:fwk,
/* observa que tomamos como valor inicial fwk y usamos siempre 2 veces Newton*/
wfin:wfin-(wfin-fwk-hh/2*ev(ff,t=tk+hh,y=wfin,numer) )
/(1-hh/2*ev(yder,t=tk+hh,y=wfin,numer) ),
wfin:wfin-(wfin-fwk-hh/2*ev(ff,t=tk+hh,y=wfin,numer) )
/(1-hh/2*ev(yder,t=tk+hh,y=wfin,numer) ),
[tk+hh,wfin,ff,hh] );
```

Lo aplicamos a la ecuación $y' = 5exp(5t)*(y-t)^2+1$ que tiene como solución exacta $y = t + \frac{1}{c_1 - exp(5t)}$; para

$y(0) = -1$ la solución es $y = t - exp(-5t.)$ Para $h = 0.25$ hacemos val:[0,-1,5*exp(5*t)*(y-t)**2+1,.25]; y obtenemos los valores sucesivos haciendo

for i: 1 thru 5 do (val:mettrap(val),print([val[1],val[2]])); nos da

[0.25,0.003524949682058], [0.5,0.42129182263298], [0.75,0.7144119523626],

[1.0,0.97184039709192], [1.25,1.233416144031785];

Aplicando Runge-Kutta de orden 4 a las mismas condiciones iniciales tenemos:

$[0.25, 0.40143153646034], [0.5, 3.437475304647394], [0.75, 1.4463916150265733 * 10^{+23}]$ y un error de Maxima por sobrepaso en la estimación de $y(1)$. Estos resultados no son aceptables ya que los valores exactos son

[0.25,-0.03650479686019], [0.5,0.4179150013761], [0.75,0.72648225414399],

[1.0,0.99326205300091], [1.25,1.248069545863772].

Ejercicio: Realiza los cálculos para la ecuación $y' = 5exp(5t) * (y - t)^2 + 1$ con $y(0) = -1$ y paso más pequeño. Observa si mejora el método de Runge-Kutta.

Ejercicio: Mejora el programa dando una tolerancia y repitiendo Newton hasta que la diferencia en valor absoluto de dos iteraciones sucesivas sea menor que la tolerancia. Mejora el programa usando un contador de repeticiones de Newton y haciendo que escriba un mensaje de error si hay que hacer demasiadas.

Para resolver numéricamente una ecuación diferencial es muy conveniente conocer su interpretación física para poder analizar los resultados, si la solución numérica crece excesivamente u oscila mucho puede haber inestabilidades. Es conveniente repetir el cálculo con dos valores del paso h distintos y aceptar sólo los dígitos comunes.

8.1.6 Sistemas de ecuaciones diferenciales.

La mayor parte de los métodos numéricos que existen para resolver un problema de valor inicial para una ecuación de primer orden se extienden sin dificultad al caso de un sistema de ecuaciones de primer orden con una condición inicial, sin más que interpretar la fórmula considerando que y, y' son vectores.

Por ejemplo, el método de Euler para $y = f(t, y)$ con $y(t_0) = y_0$ está dado por la fórmula $w_{i+1} = w_i + h\, f(t_i, w_i)$, su aplicación a un sistema $Y' = F(t, Y)$ con $Y(t_0) = Y_0$ donde Y, F, Y_0 son vectores está dado por $W_{i+1} = W_i + h\, F(t_i, W_i)$, donde W_i es el vector de los aproximaciones.

Por ejemplo apliquemos el método de Euler al sistema

$y' = y + 2z + exp(t), z' = 3y + 2z$ con las condiciones iniciales $y(0) = 1, z(0) = 0$ que tiene como solución $y = 1/30(9exp(-t) + 5exp(t) + 16exp(4t))$, $z = 1/10(-3exp(-t) - 5exp(t) + 8exp(4t))$.

Escribiendo wy, wz para las aproximaciones de y, z

$$Y = \begin{pmatrix} y \\ z \end{pmatrix};\ Y' = \begin{pmatrix} y' \\ z' \end{pmatrix};\ F(t, Y) = \begin{pmatrix} y + 2z + exp(t) \\ 3y + 2z \end{pmatrix};\ W = \begin{pmatrix} wy \\ wz \end{pmatrix}.$$

Las fórmulas que corresponden al método de Euler son:

$$wy_{i+1} = wy_i + h(wy_i + 2z_i + exp(t_i)); \qquad wz_{i+1} = wz_i + h(3wy_i + 2wz_i);$$

con valor inicial $t_0 = 0, wy_0 = 1, wz_0 = 0$;

En Maxima usando una lista formada por $[t_i, y_i, z_i, fy, fz, h]$ podemos hacer

kill(eulersis);eulersis(val):=block([ti:val[1],yi:val[2],zi:val[3],fy:val[4],fz:val[5],hh:val[6],yin,zin],

yin:yi+hh*ev(fy,t=ti,y=yi,z=zi,numer),zin:zi+hh*ev(fz,t=ti,y=yi,z=zi,numer), [ti+hh,yin,zin,fy,fz,hh]);

para $h = .1$ partimos de val:[0,1,0,y+2*z+exp(t),3*y+2*z,.1] los valores de t, y, z que obtenemos por

for i:1 thru 3 do (val:eulersis(val),print([val[1],val[2],val[3]]));

son:

$[0.1, 1.2, 0.3], [0.2, 1.490517091807565, 0.72], [0.3, 1.905709076804338, 1.31115512754227],$

y los valores exactos son:

$[0.1, 1.251286217165407, 0.3694230736644], [0.2, 1.636141514146073, 0.92411213779049],$

$[0.3, 2.217950959593341, 1.758918668196722].$

Los valores aproximados no son excesivamente buenos pero el método de Euler con paso $h = .1$ no es muy exacto.

Ejercicio: Aplica el método de Euler al sistema siguiente:

y' = 0.1 y - 0.005 /60 y z , z'= 0.00004 z y - 0.04 z, con la condición inicial y(0) = 600, z(0) = 2000;

Este sistema está relacionado con la variación de poblaciones de conejos y zorros (predador-presa) donde los conejos crecen de forma proporcional a su número y son comidos por los zorros; los zorros crecen y mueren de forma proporcional a su número y necesitan conejos que comer para que su población crezca.

Ejercicio: La instrucción $map(" = ", lista1, lista2)$ de Maxima devuelve la lista formada por las igualdades

lista1[1]=lista2[1],....,lista1[n]=lista2[n] así que para evaluar f numéricamente haciendo que las variables listavar:[x1,x2,..xn] tomen el valor valor:[v1,v2,...vn] se puede hacer ev(f,map("=",listavar,valor),numer);

Programa el método de euler para sistemas con un número arbitrario de ecuaciones, la variable de entrada puede ser $[t_i, listavariablesindependientes, listaecuaciones, listavaloresiniciales, h]$, con todas las listas de igual longitud y después usa map para las evaluaciones de f.

El método del punto medio (8.1.2) dado por $w_{i+1} = w_i + h\, f(t_i + \frac{h}{2}, w_i + \frac{h}{2} f(t_i, w_i))$; puede programarse en Maxima para un sistema de dos ecuaciones

kill(pmsis);pmsis(val):=block([ti:val[1],yi:val[2],zi:val[3],fy:val[4],fz:val[5],hh:val[6],yin,zin],

/* Observa que guardamos en yin el valor correspondiente a la primera variable de $w_i + \frac{h}{2} f(t_i, w_i)$ y en zin el correspondiente a la segunda */

yin:yi+hh/2*ev(fy,t=ti,y=yi,z=zi,numer),zin:zi+hh/2*ev(fz,t=ti,y=yi,z=zi,numer),

[ti+hh,yi+hh*ev(fy,t=ti+hh/2,y=yin,z=zin,numer),

zi+hh*ev(fz,t=ti+hh/2,y=yin,z=zin,numer),fy,fz,hh]);

Aplicándolo al sistema $y' = y + 2z + exp(t), z' = 3y + 2z$ partiendo de val:[0,1,0,y+2*z+exp(t),3*y+2*z,.1]

obtenemos [0.1,1.245127109637602,0.36], [0.2,1.617728548301885,0.89614641659611],

[0.3,2.17674513359932,1.696620411281688]. Como el método del punto medio es de orden 2, los valores que se obtiene son más exactos que los de euler, como es de esperar.

Ejercicio: aplica el programa del punto medio al sistema predador - presa.

Recuerda en el paquete *dynamics* de Maxima, la instrucción rk sirve para resolver numéricamente sistemas de ecuaciones diferenciales de primer orden $y' = f1(t,y,z)$, $z' = f2(t,y,z)$ con condiciones iniciales $y(t_0) = y_0, z(t_0) = z_0$ desde t_0 hasta $t = b$ con paso h se escribe $rk([f1(t,y,z), f2(t,y,z)], [y,z], [y_0,z_0], [t,t_0,b,h])$; y devuelve una lista de valores $[t_i, y_i, z_i]$. Ejemplo, para resolver $y' = y*z+t, z' = y+z-t$ con $y(0) = 1, z(0) = 2$ desde $t = 0$ hasta $t = 3$ con paso $h = 0.1$ se escribe $rk([y*z+t, y+z-t], [y,z], [1,2], [t,0,3,.1])$;

Ejercicio: Calcula y dibuja soluciones del sistema predador - presa anterior usando la instrucción rk de Maxima.

Una fuente de inestabilidad para sistema es que algunas variables sean grandes y otras pequeñas, por ejemplo al modelizar el movimiento de un cohete a la luna la distancia a la tierra varía mucho y lentamente en el tiempo pero los ángulos de rotación del cohete sobre si mismo varían muy rápidamente. De igual modo al refinar el petróleo para producir gasolinas aparecen reacciones químicas complejas algunas de las cuales son muy rápidas (centésimas de segundo) mientras que otras tardan minutos en realizarse. Resolver ecuaciones diferenciales que modelen estos procesos es un problema porque para reflejar bien alguna de las ecuaciones el paso h ha de ser muy pequeño pero otras ecuaciones hacen que el rango $t \in [a,b]$ deba ser muy grande.

8.1.7 Ecuaciones de orden mayor que 1.

Dada una ecuación explícita de orden mayor que 1, $y^{(n)} = f(t,y,y',y'',...y^{(n-1)})$ es posible reducirla a un sistema de ecuaciones diferenciales de primer orden sin mas que introducir variables auxiliares para las derivadas.

Llamando $y = y_0, y' = y_1, y'' = y_2, ...y^{(n-1)} = y_{n-1}$ tenemos el sistema de primer orden

$$\begin{array}{lcl} y_0' & = & y_1, \\ y_1' & = & y_2, \\ y_2' & = & y_3, \\ & ... & \\ y_{n-1}' & = & f(t,y_0,y_1,...y_{n-1}); \end{array}$$

que podemos resolver con las técnicas estudiadas en la sección anterior.

Por ejemplo la ecuación de Airy $y'' - ty = 0$; cuyas soluciones pueden representarse en Maxima con las funciones $airy\_ai(x), airy\_bi(x)$ se transforma, llamando $y' = z$ en el sistema $y' = z, z' = t\,y$.

La ecuación de Bessel $t^2y'' + ty' + (t^2 - n^2)y = 0$; sirve para definir las funciones de Bessel J_n, Y_n que en Maxima se escriben $bessel\_j(n,t), bessel\_y(n,t)$. Escribiendo la ecuación de Bessel como $y'' = -\frac{1}{t}y' + (\frac{n^2}{t^2} - 1)y$; si llamamos $z = y'$ la ecuación se transforma en el sistema de dos ecuaciones de primer orden $y' = z, z' = -\frac{1}{t}z + (\frac{n^2}{t^2} - 1)y$.

La ecuación $y''' + \lambda y = at^2 + bt + c$, llamando $u = y', v = y''$ se transforma en el sistema $y' = u, u' = v, v' = at^2 + bt + c - \lambda y$. También podíamos haber llamado $y_0 = y, y_1 = y', y_2 = y''$ con lo que el sistema es ahora $y_0' = y_1, y_1' = y_2, y_2' = at^2 + bt + c - \lambda y_0$.

Ejercicio: Utiliza los métodos $eulersis, pmsis$ de Euler y punto medio para sistemas de dos ecuaciones diferenciales de primer orden que programamos en la sección anterior para hallar soluciones numéricas de la ecuación de airy con $y(0) = 0.35502805388782, y'(0) = -0.25881940379281$, compara con los valores de $airy\_ai$ en Maxima. Calcula ahora las soluciones numéricas con la instrucción rk del paquete $dynamics$. El error debe ser menor porque rk utiliza un Runge-Kutta de orden 4.

Ejercicio: Aplica rk a la ecuación $y''' + 3y = -t^2 + 1$, con las condiciones $y(0) = 1, y'(0) = 2, y''(0) = 3$. Dibuja la solución en el intervalo $[0, 1]$.

2 Problemas de contorno.

Decimos que hay un problema de contorno si tenemos una ecuación diferencial de orden mayor que 1 e imponemos condiciones en dos puntos distintos. Por ejemplo si buscamos soluciones de la ecuación $y'' = -y$ con la condición $y(0) = 0, y(\pi/2) = 1$.

Los problemas de contorno aparecen en la naturaleza, por ejemplo para estudiar un puente que tiene los extremos fijos (si no, se cae) o un cable eléctrico que cuelga entre dos postes.

Existen problemas de contorno que tiene una solución por ejemplo $y'' = -y$ con $y(0) = 0, y(\pi/2) = 1$ tiene solución $y = sen(t)$; que no tienen ninguna por ejemplo $y'' = -y$ con $y(0) = 0, y(\pi) = 1$ ó que tienen infinitas como $y'' = -y$ con $y(0) = 0, y(\pi) = 0$ que admite $y = c_1 sen(t)$.

Ejercicio: Ya que la solución general de $y'' = -y$ es $y = c_1 sen(t) + c_2 cos(t)$ comprueba lo anterior.

Si Maxima es capaz de hallar la solución general de la ecuación resolver un problema de contorno es muy fácil, basta imponer las condiciones de contorno y calcular los valores de las constantes arbitrarias, por ejemplo para resolver $y'' + \frac{y'}{x} + y = 0$, $y(1) = 1, y(2) = 1$; hacemos

```
sol:ode2('diff(y,x,2)+'diff(y,x)/x+y,y,x); /* que devuelve */

y=bessel_y(0,x)*%k2+bessel_j(0,x)*%k1

/*tomamos el lado derecho haciendo */

sol:rhs(sol); /* resolvemos las condiciones de contorno haciendo */

ec1:ev(sol,y=1); ec2:ev(sol,y=2);

solve([ec1=0,ec2=0],[%k1,%k2]),numer;

/* que devuelve */

[[%k1=-0.23803159219318,%k2=2.063760353560585]]

/* la solución del problema de contorno la podemos calcular con */

ev(sol,%);

/* que devuelve */

2.063760353560585*bessel_y(0,x)-0.23803159219318*bessel_j(0,x).
```

Ejercicio: Resuelve el problema de contorno $y'' - y = x$, $y(0) = 1, y(2) = 2$.

8.2.1 Reducción a problemas de valores iniciales

Por simplificar estudiamos ecuaciones de segundo orden.

Si la ecuación diferencial es lineal es decir puede ponerse como $y'' + p(t)y' + q(t)y = r(t)$, ya que la derivada de la suma de dos funciones es la suma de las derivadas $(f+g)' = f' + g'$ y la derivada de una constante por una función cf es cf' se cumple que si y_1, y_2 son soluciones de la ecuación lineal homogénea $y'' + p(t)y' + q(t)y = 0$ también lo es $y = c_1y_1 + c_2y_2$ para toda constante c_1, c_2. Además si y_p es una solución cualquiera de la ecuación completa $y'' + p(t)y' + q(t)y = r(t)$ entonces todas las soluciones de la ecuación completa pueden escribirse como $y = c_1y_1 + c_2y_2 + y_p$.

Si tenemos una ecuación lineal homogénea $y'' + p(t)y' + q(t)y = 0$, y las condiciones de contorno son de la forma $y(a) = \alpha, y(b) = \beta$ podemos resolver los *dos problemas de valor inicial*

$y'' + p(t)y' + q(t)y = 0,\ y(a) = 1,\ y'(a) = 0;$

$y'' + p(t)y' + q(t)y = 0,\ y(b) = 1,\ y'(b) = 0;$

Sean sus soluciones (posiblemente numéricas) y_1, y_2 entonces $\alpha\, y_1 + \beta\, y_2$ es solución (posiblemente numérica) del problema de contorno ya que por linealidad es solución de la ecuación diferencial y satisface las condiciones $y(a) = \alpha, y(b) = \beta$

Ejercicio: Para resolver la ecuación de Airy $y'' - ty = 0$ con $y(0) = 2, y(1) = 3$, sea w_1 la solución numérica de $y'' - ty = 0,\ y(0) = 1, y'(0) = 0$ y w_2 la solución numérica de $y'' - ty = 0,\ y(1) = 1, y'(1) = 0$ (toma h negativo ahora). Calcula numéricamente w_1, w_2 por ejemplo con rk de Maxima (recuerda que tienes que escribir la ecuación como un sistema de primer orden) y dibuja $2w_1 + 3w_2$ para ver si satisface que $y(0) = 2, y(1) = 3$.

Si la ecuación lineal es no homogénea $y'' + p(t)y' + q(t)y = r(t)$, y las dos condiciones de contorno son también lineales es decir son de la forma

$$\begin{array}{c} a_1y(a) + a_2y'(a) + a_3y(b) + a_4y'(b) = A; \\ b_1y(a) + b_2y'(a) + b_3y(b) + b_4y'(b) = B \end{array} \tag{8.2.1}$$

podemos resolver los *tres problemas de valor inicial*

$y'' + p(t)y' + q(t)y = 0,\ \ y(a) = 1,\ y'(a) = 0;$ con solución $y_1(t);$

$y'' + p(t)y' + q(t)y = 0,\ \ y(b) = 1,\ y'(b) = 0;$ con solución $y_2(t);$

$y'' + p(t)y' + q(t)y = q(t),\ \ y(a) = 0,\ y'(a) = 0;$ con solución $y_3(t);$

y tomando como solución $y = c_1y_1 + c_2y_2 + y_3$ elegimos c_1, c_2 de forma que se satisfagan las condiciones de contorno (8.2.1).

Por ejemplo para resolver el problema de contorno $y'' - y = x^2,\ y(0) = 1, y'(1) = 4$ resolvemos:

$y'' - y = 0, y(0) = 1, y'(0) = 0$ que tiene como solución $y_1 = cosh(t);$

$y'' - y = 0, y(1) = 1, y'(1) = 0$ que tiene como solución $y_2 = cosh(1-t);$

$y'' - y = x^2, y(0) = 0, y'(0) = 0$ que tiene como solución $y_3 = 2cosh(t) - t^2 - 2;$

Ahora formamos $y = c_1y_1 + c_2y_2 + y_3 = c_1\, cosh(t) + c_2\, cosh(1-t) + 2\, cosh(t) - t^2 - 2$ y resolvemos

el sistema $y(0) = 1 = c_1 + c_2 cosh(1)$, $y'(1) = -2 + 2\,senh(1) + c_1\,senh(1) = 4$, y se tiene que $c_1 \sim 3.1055, c_2 \sim -1.36448$

así que la solución es $y \sim 3\,cosh(t) + 1.60354\,senh(t) - t^2 - 2$.

Ejercicio: Comprueba con Maxima los cálculos del resultado anterior (de hecho exactamente $c_1 = -2 + 6\,csch(1)$, $c_2 = 3(1 - 2\,csch(1))sech(1)$, con $senh, cosh, csch, sech$ las funciones seno hiperbólico, coseno hiperbólico, cosecante hiperbólica y secante hiperbólica) y comprueba que satisfacen la ecuacion diferencial y las condiciones de contorno.

Método del disparo El método del disparo o de la manguera se llama así porque es similar a intentar regar una maceta lejana con una manguera o que un barco pirata le acierte con un cañón a otro barco. Se hace un primer disparo de prueba, si la bala no llega al barco se sube el ángulo del cañón y si la bala cae más lejos se baja el ángulo del tiro, así hasta que acierta.

Si queremos resolver el problema de contorno $y'' = f(t, y, y')$, $y(a) = \alpha$, $y(b) = \beta$, elegimos un valor y_1 para la derivada y resolvemos numéricamente $y'' = f(t, y, y')$, $y(a) = \alpha$, $y'(a) = y_1$, sea la solución $w_1(t)$; elegimos un valor y_2 y resolvemos numéricamente $y'' = f(t, y, y')$, $y(a) = \alpha$, $y'(a) = y_2$, sea la solución $w_2(t)$, ahora podemos aplicar el método de la secante a partir de los datos $y_1, w_1(b), y_2, w_2(b)$, para estimar cual es y_3, resolvemos de nuevo numéricamente $y'' = f(t, y, y')$, $y(a) = \alpha$, $y'(a) = y_3$, sea la solución $w_3(t)$, aplicamos de nuevo secante con $y_2, w_2(b)$, $y_3, w_3(b)$, para obtener y_4 y así sucesivamente.

Como puede probarse que, bajo determinadas condiciones, hay dependencia continua entre la solución y las condiciones iniciales el proceso convergerá en general.

Por ejemplo resolvamos el problema de contorno

$$y'' + \frac{y'}{t} + y = 0,\ y(1) = 1, y(2) = 1.$$

La solución exacta es $\sim 1.13847\,bessel\_j(0,t) + 1.45992\,bessel\_y(0,t)$. Lo escribimos como un sistema haciendo $y' = z, z' = -y - z/t$. En Maxima cargamos el paquete que incluye rk con

```
load(dynamics)
```

/* resolvemos numéricamente con paso $h = 1$ y la condición inicial $y(1) = 1, y'(1) = 1$ (por ejemplo) lo que equivale a $y(1) = 1, z(1) = 1$ desde t=1 hasta t=2*/

```
rk([z,-y-z/t],[y,z],[1,2],[t,1,2,.1]);
```

/* resulta que $w_1(2) \sim 1.209947203669865$ (el cañón está alto) tomamos la condición inicial $y(1) = 1, y'(1) = 0$ */

```
rk([z,-y-z/t],[y,z],[1,0],[t,1,2,.1]);
```

/* resulta que $w_2(2) \sim 0.62752948966996$ (ahora el barco, que vale 1, está entre los dos disparos se podía usar bisección pero secante debe ir mejor)*/

/* el método de la secante estaba dado en (3.5.1), para los puntos $(x_1, y_1), (x_2, y_2)$ corresponde $x_3 = \frac{x_1 y_2 - x_2 y_1}{y_2 - y_1}$ en nuestro caso es $(y'(1) = 1, w_1(2) - 1), (y'(1) = 0, w_2(2) - 1)$ porque queremos que $w(2) = 1$, así que hacemos */

```
(1*(0.62752948966996-1)-0*(1.209947203669865-1))/((0.62752948966996-1)-(1.209947203669865-1));
```

/*que nos da un valor aproximado de $y'(1) \sim 0.63952469400699$ por tanto probamos */

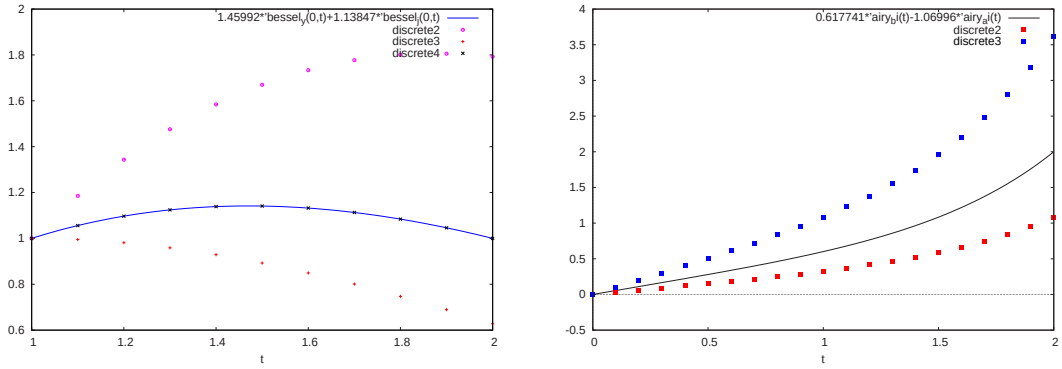

Figura 8.2.1: Método del disparo

rk([z,-y-z/t],[y,z],[1,0.63952469400699],[t,1,2,.1]);

/* que da $w_3(2) = 1.000000000000002$; Hemos terminado, ya que la solución del problema de valor inicial $y(1) = 1, y'(1) = 0.63952469400699$ dentro de los errores de rk satisface que $y(1) = 1, y(2) = 1$ que era nuestra condición de contorno. No siempre será así de bueno porque puede haber inestabilidad y además la ecuación de partida es lineal.*/

Ejercicio: Dibuja la solución exacta y las tres listas de soluciones aproximadas que hemos obtenido. Debes obtener algo parecido al primer dibujo de 8.2.1.

Observa que rk nos da una lista de valores con lo que podemos interpolar, dibujar la solución y si hace falta partir de ésta para hallar otra mas exacta con paso h menor.

Ejercicio: Comprueba los cálculos anteriores, ya que la solución exacta es $\sim 1.13847\, bessel\_j(0,t) + 1.45992\, bessel\_y(0,t)$. Dibuja los valores de la derivada de la solución. Estima cuanto vale $w(0.12)$ (tienes dos opciones, recalcular la solución del problema de contorno con $h = 0.02$ o interpolar el valor de la solución aproximada en $t = 0.12$ con los valores obtenidos para $h = 0.1$).

Ejercicio: Resuelve por el método del disparo el problema de contorno $y'' = ty, y(0) = 0, y(2) = 2$; elige $h = 0.1$ La solución exacta es $y \sim 0.617741\, airy\_bi(t) - 1.06996\, airy\_ai(t)$. Ayuda: la ecuación se escribe como un sistema $z = y'$, $z' = t*y$. Las dos primeras aproximaciones partiendo de $y(0) = 0, z(0) = y'(0) = 1$ y de $y(0) = 0, z(0) = y'(0) = 0.3$ y la solución exacta se dibujan en la segunda figura de 8.2.1.

8.2.2 Diferencias finitas para ecuaciones diferenciales ordinarias

Los métodos anteriores suelen ser inestables. El método de las diferencias finitas para el problema de contorno $y'' = f(t, y, y'), y(a) = \alpha, y(b) = \beta$ consiste en tomar $n + 1 = (b - a)/h$ puntos igualmente espaciados $t_0 = a, t_1 = a + h, t_2 = a + 2h, ...t_n = a + nh, t_{n+1} = a + (n + 1)h = b$ en el intervalo [a,b] y aplicar en cada punto la ecuación diferencial con las derivadas reemplazadas por las fórmulas de derivación numérica que vimos en un Capítulo anterior. Esto da origen a n ecuaciones con n incógnitas que se resuelven. Si la ecuación diferencial es lineal las ecuaciones son lineales y se pueden resolver por métodos matriciales (Gauss, Jacobi, Gauss-Seidel,...). Si la ecuación diferencial no es lineal se usa, por ejemplo un método de Newton (en Maxima existe la función $mnewton$) para sistemas no lineales.

Es importante que los errores de las fórmulas de derivación numérica que se utilicen sean del mismo orden y

que h no sea demasiado pequeño por los problemas de estabilidad de las fórmulas de derivación numérica.

Por ejemplo, consideremos el problema de contorno lineal $y''+y'+ty=0$, $y(0)=2$, $y(1)=3$ con $h=0.25$ usamos para la derivada primera la fórmula (7.1.2) $f'(x)=\frac{f(x+h)-f(x-h)}{2h}$ y para la derivada segunda la fórmula (7.1.5) $f''(x)=\frac{f(x+h)-2f(x)+f(x-h)}{h^2}$ que son ambas de orden h^2. Como hemos elegido $h=0.25$ tomamos los puntos $t_0=a=0$, $t_1=.25, t_2=.5, t_3=.75$, $t_4=b=1$; y los valores correspondientes que tenemos que hallar son $w_1=w(t_1), w_2=w(t_2), w_3=w(t_3)$, los valores $w(0)=2, w(1)=3$ salen de las condiciones de contorno. En la primera gráfica de 8.2.2 puedes ver los tres puntos.

La fórmula (7.1.2) puede escribirse $w'(t_k)=\frac{w(t_{k+1})-w(t_{k-1})}{2h}$ y la fórmula (7.1.5) se puede escribir como $w''(t_k)=\frac{w(t_{k+1})-2w(t_k)+w(t_{k-1})}{h^2}$ así que en cada punto (t_k,w_k) la ecuación diferencial $y''+y'+ty=0$ se escribe

$$\frac{w_{k+1}-2w_k+w_{k-1}}{h^2}+\frac{w_{k+1}-w_{k-1}}{2h}+t_kw_k=0;$$

En el punto $(t_1=0.25,\ w_1)$ queda la ecuación (recuerda que $w_0=2$ es una de las condiciones de contorno)

$$\frac{w_2-2w_1+2}{.25^2}+\frac{w_2-2}{2*0.25}+0.25w_1=0;$$

En el punto $(t_2=0.5,\ w_2)$ queda la ecuación

$$\frac{w_3-2w_2+w_1}{0.25^2}+\frac{w_3-w_1}{2*0.25}+0.5w_2=0;$$

Finalmente en el punto $(t_1=0.75,\ w_3)$ queda la ecuación (recuerda que $w_4=3$ es una de las condiciones de contorno)

$$\frac{3-2w_3+w_2}{0.25^2}+\frac{3-w_2}{2*0.25}+0.75w_3=0;$$

Haciendo $solve([(w2-2*w1+2)/0.25^2+(w2-2)/0.5+0.25*w1=0,(w3-2*w2+w1)/0.25^2+(w3-w1)/0.5+0.5*w2=0,(3-2*w3+w2)/0.25^2+(3-w2)/0.5+0.75*w3=0],[w1,w2,w3]),numer;$

Maxima nos devuelve

$w1=2.471240192402656, w2=2.803437561599129, w3=2.98394002759641$

Los valores exactos son $w1\sim 2.47231, w_2\sim 2.80438, w_3\sim 2.98427$ así que el resultado no está mal.

Apliquemos el método a una ecuación diferencial no lineal por ejemplo $y''=y^2$ con las condiciones $y(0)=6, y(1)=1.6$, por simplificar hacemos $h=\frac{1}{3}$, esto nos da los puntos $t_0=a=0$, $t_1=\frac{1}{3}, t_2=\frac{2}{3}$, $t_3=b=1$; los valores que tenemos que hallar son w_1, w_2. Aplicando las fórmulas (7.1.2),(7.1.5) en cada punto (t_k,w_k) la ecuación diferencial nos queda $\frac{w_{k+1}-2w_k+w_{k-1}}{h^2}=w_k^2$ y considerando que $w_0=6, w_3=1.6$, las ecuaciones que corresponden a (t_1,w_1) y (t_2,w_2) son

$$\frac{w_2-2w_1+6}{\frac{1}{9}}=w_1^2,\ \frac{1.6-2w_2+w_1}{\frac{1}{9}}=w_2^2;$$

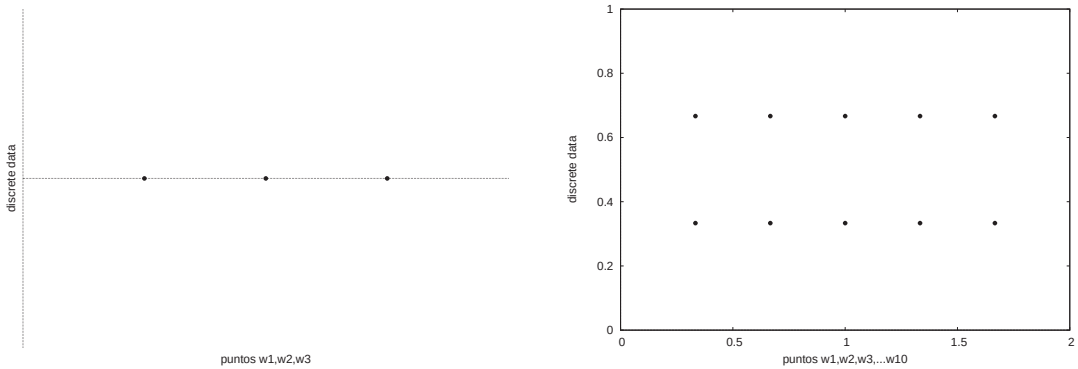

Figura 8.2.2: Puntos para diferencias finitas

que son no lineales (aunque si polinómicas). Resolviendo tenemos que

solve([w2-2*w1+6=w1^2/9,1.6-2*w2+w1=w2^2/9],[w1,w2]);

nos devuelve dos soluciones complejas y las soluciones

[w1=3.459502806736167,w2=2.248800959232614],

[w1=-7.111114186851211,w2=-14.60356652949246]; La segunda solución no es coherente con los valores en el contorno. Los valores aproximados de la solución exacta son $y(1/3) \sim 3.39759, y(2/3) \sim 2.21552$. Para el lector interesado la solución exacta se puede dar a partir de la función elíptica de P de Weierstrass, de hecho es $\sqrt[3]{6}P(\frac{t+1}{\sqrt[3]{6}}, 0, 1.2)$ donde $0, 1.2$ son parámetros y las condiciones exactas de contorno son $y(0) \sim 6.00714$, $y(1) \sim 1.61496$.

Ejercicio: Aplica diferencias finitas al problema de contorno $y'' + t\,y' + t^2\,y = 0, y(0) = 1, y(2) = 4$ con $h = 0.5$. Los valores exactos son $y(0.5) \sim 4.1991$, $y(1) \sim 6.38549$, $y(1.5) \sim 6.40682$.

8.2.3 Diferencias finitas para ecuaciones en derivadas parciales

El método de diferencias finitas puede aplicarse también a ecuaciones en derivadas parciales. Supongamos que queremos estudiar la ecuación

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0. \tag{8.2.2}$$

en el rectángulo $x \in [0,2], y \in [0,1]$ con las condiciones de contorno $u(x,0) = u(0,y) = 0, u(2,y) = 4y, u(x,1) = 2x$ con paso correspondiente a x, $h = \frac{1}{3}$ y paso correspondiente a y, $k = \frac{1}{3}$. Para estos valores de h, k enumeramos, por ejemplo, los puntos que están dentro del rectángulo [0,2]x[0,1] de izquierda a derecha y de abajo arriba, así hay 10 puntos

$P_1 = (x_1, y_1) = (\frac{1}{3}, \frac{1}{3}); w(P_1) = w_1$. $P_2 = (x_2, y_2) = (\frac{2}{3}, \frac{1}{3})$; $w(P_2) = w_2$. $P_3 = (x_3, y_3) = (1, \frac{1}{3})$; $w(P_3) = w_3$. $P_4 = (x_4, y_4) = (\frac{4}{3}, \frac{1}{3})$; $w(P_4) = w_4$. $P_5 = (x_5, y_5) = (\frac{5}{3}, \frac{1}{3})$; $w(P_5) = w_5$. $P_6 = (x_6, y_6) = (\frac{1}{3}, \frac{2}{3})$; $w(P_6) = w_6$. $P_7 = (x_7, y_7) = (\frac{2}{3}, \frac{2}{3})$; $w(P_7) = w_7$. $P_8 = (x_8, y_8) = (1, \frac{2}{3})$; $w(P_8) = w_8$. $P_9 = (x_9, y_9) = (\frac{4}{3}, \frac{2}{3})$; $w(P_9) = w_9$. $P_{10} = (x_{10}, y_{10}) = (\frac{5}{3}, \frac{2}{3})$; $w(P_{10}) = w_{10}$.

En la segunda gráfica de 8.2.2 puedes ver los 10 puntos.

la fórmula (7.1.5) para la derivada segunda $f''(z)=\frac{f(z+h)-2f(z)+f(z-h)}{h^2}$ se convierte para $\frac{\partial^2 u}{\partial x^2}$ en $\frac{u(x+h,y)-2u(x,y)+u(x-h,y)}{h^2}$ y para $\frac{\partial^2 u}{\partial y^2}$ en $\frac{u(x,y+k)-2u(x,y)+u(x,y-k)}{k^2}$ así que la ecuación correspondiente a la discretización de la ecuación diferencial (8.2.2) en un punto $P=(x,y)$ es

$$\frac{u(x+h,y)-2u(x,y)+u(x-h,y)}{h^2}+\frac{u(x,y+k)-2u(x,y)+u(x,y-k)}{k^2}=0.$$

Para el punto $P_1=(\frac{1}{3},\frac{1}{3})$ es $\frac{u(\frac{2}{3},\frac{1}{3})-2u(\frac{1}{3},\frac{1}{3})+u(0,\frac{1}{3})}{(\frac{1}{3})^2}+\frac{u(\frac{1}{3},\frac{2}{3})-2u(\frac{1}{3},\frac{1}{3})+u(\frac{1}{3},0)}{(\frac{1}{3})^2}=0$;

Considerando que $u(0,\frac{1}{3})=u(\frac{1}{3},0)=0$ por las condiciones de contorno y como hemos llamado a $w_1,w_1,...w_{10}$ la ecuación que corresponde al punto P_1 nos queda quitando el denominador común $w_2-2w_1+w_6-2w_1=0$; que puede escribirse $4w_1=w_2+w_6$.

La ecuación correspondiente a P_2, usando la condición de contorno $u(\frac{2}{3},0)=0$ es $w_3-2w_2+w_1+w_7-2w_2=0$; que puede escribirse $4w_2=w_1+w_3+w_7$.

La ecuación correspondiente a P_8, usando la condición de contorno $u(1,1)=2$ es $w_7-2w_8+w_9+w_3-2w_8+2=0$; que puede escribirse $4w_8=w_3+w_7+w_9+2$. Observa que podemos escribir las 10 ecuaciones con 10 incógnitas $w_1,w_2,...w_{10}$ en forma matricial de forma diagonalmente dominante luego el sistema puede resolverse de forma iterativa.

Ejercicio: Halla las ecuaciones correspondientes a los restantes puntos, resuelve el sistema y halla los valores aproximados $w_1,w_2,...w_{10}$. Compara con la solución exacta $u(x,y)=2xy$.

Apliquemos el método de diferencias finitas a la ecuación en derivadas parciales

$$\frac{\partial u}{\partial t}-\frac{\partial^2 u}{\partial x^2}=0. \tag{8.2.3}$$

Igual que antes la fórmula (7.1.5) para la derivada segunda $f''(z)=\frac{f(z+h)-2f(z)+f(z-h)}{h^2}$ se convierte para $\frac{\partial^2 u}{\partial x^2}$ en $\frac{u(t,x+h)-2u(t,x)+u(t,x-h)}{h^2}$ con un error del orden de h^2 y al discretizar la derivada primera $\frac{\partial u}{\partial t}$ si usamos la fórmula (7.1.1) $f'(z)=\frac{f(z+k)-f(z)}{k}$ el error que se comete es del orden de k con lo que para que sea coherente con el error de discretizar $\frac{\partial^2 u}{\partial x^2}$ nos obliga a tomar $k\sim h^2$ que puede ser demasiado pequeño y obligarnos a tomar muchos puntos. Es más interesante usar para $\frac{\partial u}{\partial t}$ la fórmula (7.1.2) $f'(z)=\frac{f(z+k)-f(z-k)}{2k}$ que tiene un error del orden de k^2 con lo que podemos tomar $h=k$.

Así que la ecuación correspondiente a la discretización de la ecuación diferencial (8.2.3) en un punto cualquiera $P=(x,y)$ es

$$\frac{u(t+k,x)-u(t-k,x)}{2k}-\frac{u(t,x+h)-2u(t,x)+u(t,x-h)}{h^2}=0. \tag{8.2.4}$$

Ejercicio: Aplica diferencias finitas a la ecuación (8.2.3) en el rectángulo $t\in[0,2],x\in[0,1]$ con las condiciones de contorno $u(t,0)=0,u(0,x)=sen(x),u(2,x)=exp(-2)sen(x),u(t,1)=0.8414exp(-t)$ con paso $h=k=\frac{1}{2}$. Resuelve el sistema y compara con la solución $u=exp(-t)sen(x)$. (Ayuda: sólo hay 3 puntos que estudiar $(\frac{1}{2},\frac{1}{2}),(1,\frac{1}{2}),(\frac{3}{2},\frac{1}{2})$).

Dependiendo de las fórmulas de discretización utilizadas y de la ecuación el método de las diferencias finitas puede ser inestable.

3 Problemas

1. Comprueba que $y = c\,exp(3x)$ es solución de la ecuación diferencial $y' = 3y$ para cualquier valor de c. Dibuja soluciones para varios valores de c. Ayuda: prueba plot2d(create_list((c/4)*exp(3*x),c,-8,8),[x,-2,2],[y,-5,5]); Observa que por cada punto del plano pasa sólo una solución (si no te fías haz que las c estén más próximas).

2. Comprueba que $y = (x+c)^3$ es solución de la ecuación diferencial $y' = 3\sqrt[3]{y^2}$ para cualquier valor de c. Comprueba que $y = 0$ también es solución de la ecuación diferencial. Dibuja la solución $y = 0$ y también $y = (x+c)^3$ para varios valores de c. Ayuda: prueba lisfun:cons(0,create_list((x+c/2)^ 3,c,-6,6)); para crear una lista que tenga el cero junto con $(x-3)^2, (x-5/2)^2, (x-2)^2, ..., (x+3)^2$ y luego haz plot2d(lisfun,[x,-4,4],[y,-20,20]);. Observa que por cada punto del plano excepto en el eje de las x pasa sólo una solución (si no te fías haz que las c estén más próximas).

3. Un modelo sencillo de crecimiento de población es el modelo exponencial $y' = a\,y$ con a constante que refleja que el crecimiento es proporcional a la población en cada momento. Otro modelo es el modelo logístico $y' = a\,y - by^2$ que refleja el hecho de tener que competir por los recursos. Resuelve con la instrucción $ode2$ de Maxima las dos ecuaciones para distintos valores iniciales y representa las soluciones. Dibuja varias soluciones y observa si los modelos reflejan situaciones reales.

4. El paquete de Maxima *plotdf* crea un dibujo del campo de direcciones (es decir si la ecuación es $y' = f(t,y)$ en los puntos se pinta la dirección que tiene la solución que pasa por el punto que será $f(x,y)$) de una ecuación diferencial. La gráfica es interactiva, picando con el ratón en un punto se dibuja la solución numérica de la ecuación diferencial que pasa por él. Por ejemplo haz

   load(plotdf); plotdf(exp(-x)+y,[trajectory_at,2,-0.1]);

   para dibujar el campo de direcciones de la ecuación $y' = y + exp(-x)$ y la solución que pasa por $(2,-.1)$. Pica con el ratón en diversos puntos de la gráfica para pintar la trayectoria de otras soluciones. Halla el campo de direcciones de la ecuación $y' = 3y$ y dibuja varias trayectorias (puedes hacer plotdf(3*y);). Observa que las soluciones se apartan del eje horizontal. Halla el campo de direcciones de la ecuación $y' = y^2$ y dibuja varias trayectorias. Observa como las soluciones del semiplano inferior se acercan al eje real y las del semiplano superior se alejan. Halla el campo de direcciones de la ecuación $y' = exp(y)$ y dibuja varias trayectorias. Observa que algunas soluciones crecen muy rápido.

   Algunas ecuaciones pueden tener trayectorias cerradas por ejemplo el sistema $y' = z, z' = -3y$ se dibuja con plotdf([z,-3*y],[y,z]); dibuja varias trayectorias y observa que se cierran. Prueba también el sistema $y' = -z, z' = -3y$. Juega poniendo otras ecuaciones y sistemas. Lee la ayuda de Maxima para ver más opciones. El sistema debe ser autónomo es decir no debe aparecer la variable independiente. La sintaxis de la definición de las funciones no cubre todas las funciones de Maxima.

5. Comprueba que $y' = y + 4t$ admite la solución $y = f(t) = 3exp(t) - 4t - 3$. Halla con paso $h = 0.1$ un total de 25 puntos de cada solución aproximada por el método de Euler de la ecuación $y' = y + 4t$ que satisfaga las condiciones iniciales $y(0) = 0, y(0.1) = f(0.1) \sim -0.0844872, y(0.2) = f(0.2) \sim -0.135792, y(0.3) = f(0.3) \sim -0.150424, y(0.4) = f(0.4) \sim -0.124526$ Dibuja las 5 soluciones aproximadas y la solución exacta. Ayuda usa $eu2$ definido antes haciendo

ff:y+4*t; hh:0.1; kill(eu2); eu2(arg):=block([ti:arg[1],wi:arg[2]], [ti+hh,wi+hh*ev(ff,y=wi,t=ti,numer)]);

valor:[0.,0.];lista1:[valor];

for i:1 thru 25 do (valor:eu2(valor),lista1:endcons(valor,lista1)); /*ya tienes la primera lista de puntos */

valor:[0.1,-0.0844872];lista2:[valor];

for i:1 thru 25 do (valor:eu2(valor),lista2:endcons(valor,lista2)); /*ya tienes la segunda lista de puntos */

calcula ahora lista3 partiendo de valor:[0.2,-0.135792]; y sigue con lista4 y lista5 finalmente dibuja

plot2d([3*exp(t)-4*t-3,[discrete,lista1],[discrete,lista2],...],[t,0,2.5]);

Observa que las soluciones numéricas se van alejando de la solución exacta pero no demasiado deprisa. Repite el proceso usando métodos de mayor orden, por ejemplo euler modificado o Runge Kutta de orden 4 y comprueba si las soluciones numéricas se alejan de la solución exacta también.

6. Repite el problema anterior con la ecuación $y' = 4y - 5exp(-t)$ que admite la solución $y = f(t) = exp(-t)$ (compruébalo) hallando con paso $h = .1$ un total de 25 puntos por el método de Euler con las condiciones iniciales $y(0) = 1, y(0.1) = f(0.1) \sim 0.904837, y(0.2) = f(0.2) \sim 0.818731, y(0.3) = f(0.3) \sim 0.740818, y(0.4) = f(0.4) \sim 0.67032$, Dibuja también la solución exacta y las 5 listas de soluciones y observa si las soluciones aproximadas se alejan más o menos que con la ecuación $y' = y + 4t$. Ayuda: la ecuación actual está peor condicionada, su solución completa es $y = exp(-t) + c * exp(4t)$ y, aunque la condición inicial corresponde a $c = 0$ los errores hacen no nulo el coeficiente de $exp(4t)$ y la solución aproximada se aleja rápidamente. Repite el proceso con el método del trapecio (lo definimos antes, $mettrap$) y observa si las soluciones se alejan más despacio o no.

7. La ecuación $y' = y^2 - 10\,exp(-100(t-2)^2)$ tiene el término $10exp(-100(t-2)^2)$ que es muy pequeño excepto que t sea muy próximo a 2 (dibuja la función). La interpretación física de una función de este tipo es un *impulso*, una fuerza que actúa en un tiempo muy corto. Calcula con Runge-Kutta de orden 4 con $h = 0.1$ la solución que satisface $y(1) = -1$ hasta $t = 7$ y, si es posible, dibuja la solución. Ayuda: después de la definición de $rk4$ usa valor:[1.,-1,y^2-exp(-100*(t-2)^2),.1];listark4:[[1.,.2]]; for i:1 thru 60 do (valor:rk4(valor),listark4:endcons([valor[1],valor[2]],listark4)); Prueba con h más pequeño. Dibuja a la vez las dos soluciones obtenidas. Compara con una solución obtenida con la rutina de Maxima rk (Ayuda: después de load (dynamics); haz rk(y^2-10*exp(-100*(t-2)^2),y,-1,[t,1,7,.1]);). Comprueba que $y = \frac{-1}{t}$ es solución de $y' = y^2$ con $y(1) = -1$ y que $y = \frac{1}{1.5-t}$ es solución de $y' = y^2$ con $y(2.5) = -1$. Dibuja $y = \frac{-1}{t}, y = \frac{1}{1.5-t}$ junto con la solución que has obtenido para la ecuación con el impulso y observa como afecta el impulso a las soluciones de ésta ecuación.

8. Para las siguientes ecuaciones halla la solución aproximada en $t = 1$ por los métodos de Euler, Euler modificado, Runge- Kutta de orden 4 y predictor-corrector para $h = 0.05$ y $h = 0.01$. Compara los errores entre los métodos. Observa para cada método como afecta al error dividir h por cinco.

   (a) $(1 + t^2)y' + ty = 0, y(0) = 1$ solución exacta $y = \frac{1}{\sqrt{1+t^2}}$.

   (b) $y' = 2t\,y, y(0) = 1$ solución exacta $y = exp(t^2)$.

   (c) $y'\,tan(t) - y = 1, y(\frac{\pi}{6}) = -0.5$ solución exacta $y = sen(t) - 1$.

   (d) $\sqrt{1 + t^2}\,y' + ty = 0, y(0) = 1$ solución exacta $y = exp(1 - \sqrt{1 + t^2})$.

   (e) $t\,y' - (t + y) = 0, y(\frac{1}{e}) = 0$ solución exacta $y = t(1 + log(t))$.

9. Puede ser interesante estudiar hasta que punto un programa de resolución numérica de ecuaciones diferenciales $y' = f(t, y)$ (por ejemplo rk de Maxima) gasta tiempo en calcular los valores de $f(t, y)$, para ello podemos hacer $showtime : true$; y calcular 1000 puntos de una ecuación muy simple $y' = 1$ (si no quieres ver el resultado, termina la orden con $). Después calcula 1000 puntos de una ecuación diferencial con una función f complicada por ejemplo $y' = exp(cos(t)) + cos(exp(t)) + log(1 + exp(cos(t * t)))$. Deduce en que gasta el tiempo rk. Si está interesado, Maxima tiene la función $compile$ que permite escribir las funciones de forma que se ejecuten más deprisa, busca en la ayuda y prueba si se ejecuta más deprisa compilando f.

10. Para simular los efectos del redondeo en las ecuaciones del problema anterior calcula la solución aproximada sumando un número aleatorio del orden de 10^{-4} al valor de $f(t, y)$ en cada uno de los cálculos con $h = 0.05$ y $h = 0.01$ (Ayuda: en lugar de usar en la ecuación (a) $f(t, y) = \frac{-t\,y}{1+t^2}$ usa $\frac{-t\,y}{1+t^2} + random(0.0002) - 0.0001$). Dibuja la solución exacta y la solución aproximada. Observa como afectan los errores a cada ecuación y cada método.

11. El método de Milne es un método predictor corrector dado por el predictor

$$w_{k+1} = w_{k-3} + \frac{4h}{3}(2f(t_k, w_k) - f(t_{k-1}, w_{k-1}) + 2f(t_{k-2}, w_{k-2}));$$

y el corrector

$$w_{k+1} = w_{k-1} + \frac{h}{3}(f(t_{k+1}, w_{k+1}) + 4f(t_k, w_k) + f(t_{k-1}, w_{k-1}));$$

Programa en Maxima el método usando siempre 2 veces el corrector y aplícalo a resolver alguna ecuación. Comprueba si el error es similar al de Runge-Kutta de orden 4 con el mismo paso.

12. La ecuación de Lorentz que corresponde a un sistema simplificado de predicción del tiempo puede expresarse por el sistema $x' = a(x - y); y' = -xz + bx - y; z' = xy - cz$ donde a, b, c son constantes. Fija el valor de las constantes $a = -3, b = 26.5, c = 1$; y halla la solución aproximada hasta $t = 30$ a partir de $x(0) = 0, y(0) = 1, z(0) = 0$ con la instrucción rk de Maxima. Halla también la solución aproximada hasta $t = 30$ a partir de $x(0) = 0, y(0) = 1.1, z(0) = 0$. Dibuja las dos soluciones correspondiente a x en la misma gráfica y observa como son prácticamente iguales al principio y luego difieren mucho. Dibuja las diferencias entre las dos soluciones. Prueba con otros valores de a, b, c para ver si ocurre igual.

13. Llamamos $x(t), y(t)$ al número de conejos y zorros. Para que la escala sea similar el número de conejos está dividido por 100. Un modelo más ajustado del problema predador presa se puede obtener de la siguiente forma: sea m la cantidad de conejos mínima que come un zorro por unidad de tiempo, $exp(\frac{-d\,x}{m\,y})$ el porcentaje de zorros que mueren, sea de hambre o de vejez, a, b miden la reproducción de conejos y zorros, c refleja que hasta que no hay suficientes conejos los zorros cazan otro animal, m regula la mortalidad de los zorros. Las ecuaciones son

$$x' = ax - m * min(1, x/c)y; \qquad y' = by - exp(-\frac{d\,x}{m\,y})y;$$

Elige valores para los parámetros, por ejemplo $a = 0.2, b = 0.2, m = 110, c = 30, d = 3.4$ y toma $x(0) = 12, y(0) = 0.4$; Resuelve las ecuaciones con rk de Maxima hasta $t = 100$ y dibuja los resultados. Observa que la población de conejos decae y luego evoluciona a un estado estacionario.

Elige ahora los valores de los parámetros $a = 0.26, b = 0.2, m = 110, c = 30, d = 3.4$ y toma $x(0) = 120, y(0) = 3$; resuelve y dibuja los resultados. Observa que al haber muchos conejos al principio la población prácticamente desaparece.

Prueba con otros valores (por ejemplo pocos zorros al principio $x(0) = 12, y(0) = 0.003$), m más pequeño o grande o cambia el valor de d para ver como cambia el modelo. Analiza si los resultados son coherentes con la realidad.

14. Escribe como un sistema la ecuación $y''' = -\frac{yy'}{2}$. Resuelve numéricamente el problema de valor inicial $y(0) = 0, y'(0) = 0, y''(0) = 1$ para el intervalo [0,10]. Dibuja la solución. Resuélvelo también para el intervalo [-5,0]. Ayuda: toma h negativo.

15. Aplica el método del disparo para resolver el problema de contorno

    $x' = y; \quad y' = z; \quad z' = -\frac{x\,y}{2}; \quad x(0) = 0, \quad y(0) = 0, \quad z(10) = 1;$

    siendo $x(t), y(t), z(t)$. Utiliza rk de Maxima en [0,10] con las condiciones iniciales $x(0) = 0, \quad y(0) = 0, \quad z(0) = a$; por ejemplo prueba con valores $a = 0.2$, $a = 0.5$ para arrancar el proceso. Dibuja la solución.

16. Escribe la ecuación en diferencias que corresponde a sustituir las fórmulas para las derivadas (7.1.2) y (7.1.5) en la ecuación $y'' + y' - y = t$ con paso h. Resuelve por diferencias finitas el problema de contorno $y'' + y' - y = t, y(0) = 0, y(3) = 1$ en [0,3] con $h = 1$. Compara con la solución exacta $y \sim -1 - t + 0.22exp(-1.61t) + 0.78exp(0.62t)$. Toma ahora $h = 0.5$, resuelve el problema de contorno y compara los errores.

17. Escribe la ecuación en diferencias que corresponde a sustituir la fórmula para la derivada (7.1.5) en la ecuación $y'' - t\,y = t$ con paso h. Resuelve por diferencias finitas el problema de contorno $y'' - t\,y = t, y(0) = 0, y(3) = 3$ en [0,3] con $h = 1$. Compara con la solución exacta $y \sim (0.284 + \pi Ai'(t))Bi(t) + Ai(t)(2.33 - \pi Bi'(t))$; con Ai, Bi las funciones de Airy (en Maxima *airy_ai,airy_bi*) y Ai', Bi' las derivadas de las funciones de Airy (en Maxima *airy_dai,airy_dbi*). Toma ahora $h = 0.5$, resuelve el problema de contorno y compara los errores.

18. Escribe la ecuación en diferencias que corresponde a sustituir las fórmulas para las derivadas (7.1.2) y (7.1.5) en la ecuación

$$\frac{\partial^2 u}{\partial x^2} - (x+t)\frac{\partial u}{\partial x} + \frac{\partial^2 u}{\partial t^2} = (-t^2 + 2x - t(x^2 + tx - x - 2))exp(x+t)$$

    con pasos $h = k$. Resuelve el problema de contorno dado por la ecuación y las condiciones $u(0, x) = 0, u(t, 0) = 0, u(2, x) \sim 14.78\,x\,exp(x)$, $u(t, 3) \sim 60.26\,t\,exp(t)$, en el rectángulo [0,2]x[0,3] con $h = k = 1$. Compara con la solución exacta $u(t, x) = t\,x\,exp(x + t)$. Toma ahora $h = k = 0.5$, resuelve el problema de contorno y compara los errores.

Bibliografía

[1] Abramowitz M., Stegun I.A. *Handbook of Mathematical Functions* (Dover, New York 1972). 0.0.1

[2] Atkinson K.E. *An introduction to Numerical Analysis*, 2 edición, John Whiley & Sons, New York, 1989. 0.0.1

[3] Burden R.L., Douglas Faires J., *Análisis Numérico*, 6 edición, Internacional Thomson Editores, Méjico, 1998. 0.0.1

[4] Chapra S.C., Canale R.P., *Métodos Numéricos para Ingenieros*, McGraw-Hill, Méjico, 1988. 0.0.1

[5] B.P. Demidovich, I.A. Maron, *Cálculo Numérico Fundamental*, Paraninfo, Madrid, 1977. 0.0.1

[6] C. García, J.L. Romero, *Modelos y Sistemas Dinámicos*, Serv. Pub. Universidad de Cádiz, Cádiz, 1998. 0.0.1

[7] *Manual de Maxima*, http://www.maxima.sourceforge.net/docs/manual/es/ 1

[8] J.R. Rice, *Numerical Methods, Software, and Analysis*, McGraw-Hill, Auckland, 1983. 0.0.1

[9] M. Rodríguez, Primeros pasos en Maxima, http://www.telefonica.net/web2/biomates/maxima/ 1

[10] R. Rodríguez, Maxima: una herramienta de cálculo, http://www.uca.es/softwarelibre/publicaciones/guia-wxmaxima 1

[11] F. Scheid, R.E. Di Constanzo *Métodos Numéricos*, 2 edición McGraw-Hill, Méjico, 1991. 0.0.1