Ingeniería en Seguridad Informática y Redes

Ingeniería en Sistemas Computacionales

Algoritmos de Solución Numérica

Manual de funciones

IS727272 - Marco Ricardo Cordero Hernández

SI727576 - Edgar Guzmán Claustro

Jal., 21 de noviembre de 2023

Índice

Contexto

A lo largo del curso, se ha tenido la oportunidad de revisar múltiples métodos numéricos para facilitar la vida de todos aquellos que los hemos estudiado y consecuentemente implementado. Es bien sabido que la herramienta predilecta para llevar a cabo esta ardua y ocasionalmente extenuante tarea ha sido Matlab (2023), sin embargo, a pesar de las opiniones divididas que genera el planteamiento de su uso, la herramienta es capaz de realizar mucho más que solo aproximaciones de valores.

Aún cuando cuenta con un lenguaje de programación propio, extendible a grandes niveles de complejidad, Matlab no deja de ser una herramienta auxiliar científica; de hecho, se le puede ver como una calculadora con muchas teclas integradas y funciones alternas que atienden casi todo rincón que la matemática exige. Este hecho cuenta con cierto grado de ambivalencia, puesto que es posible que las necesidades programáticas de un problema no sean cubiertas de forma sencilla, pero, cuando un cálculo extensivo se presenta, es casi seguro que el acercamiento a su resolución pueda realizarse sucintamente con la simple invocación de un método integrado.

El documento actual precisamente tiene el objetivo de demostrar solo algunas de las amplias funciones integradas dentro de Matlab a través de código real, e incluso se harán algunas implementaciones en donde los mismos métodos numéricos previamente mencionados serán parcial o completamente sustituidos por solo un par de llamadas a las funciones revisadas.

También, cabe mencionar que lo presentado ha sido posible gracias a la búsqueda directa dentro de la misma documentación que Matlab ofrece, algo que nuevamente demuestra cómo no todo el tiempo es necesaria la enseñanza presencial emitida a través de una figura docente, especialmente en carreras relacionadas a la tecnología, porque al final resulta cierto aquello de que “todo se encuentra en internet”.

Exploración

interp1

Uso común → `interp1(x, v, xq)` → conjunto de puntos

`x` → Puntos de muestra (arreglo o rango)

`v` → Función donde se evalúan los puntos (v(x))

`xq` → Puntos a interpolar

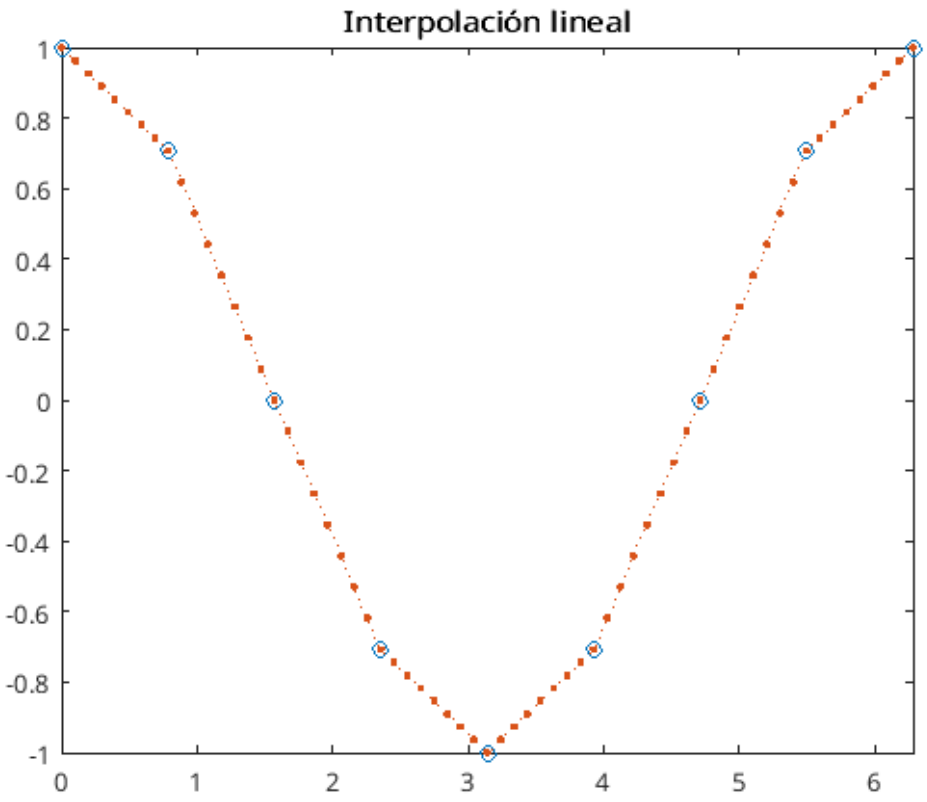
Esta función devuelve valores *interpolados* de una función dado un conjunto de muestra, la función a evaluar, y un rango en donde se deseen interpolar los datos para el mismo conjunto. La función cuenta con múltiples parámetros opcionales como `method`, el cual es útil para especificar el método de interpolación para los valores proporcionados, siendo un posible valor 'spline' (revisado más adelante como función independiente).

Demostración

```
clc, clear, close all

x = 0:pi/4:2*pi;
v = cos(x);
xq = 0:pi/32:2*pi;
vq = interp1(x,v,xq);

figure('Name','Demostración de interp1');
plot(x,v,'o',xq,vq,':.');
xlim([0 2*pi]);
title('Interpolación lineal');
```



spline

Uso común → `spline(x,y,xq)` → conjunto de puntos

`x` → Puntos de muestra (arreglo o rango)

`y` → Función donde se evalúan los puntos (y(x))

`xq` → Puntos a interpolar

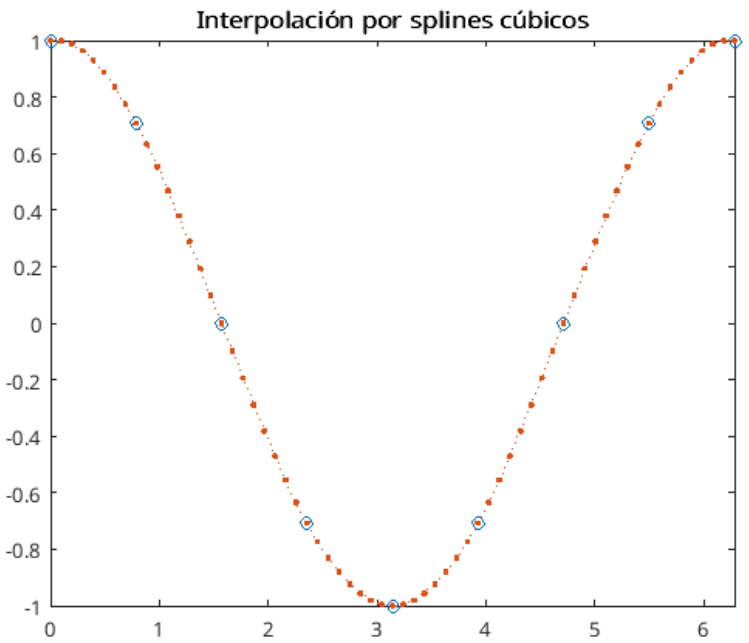
Como se mencionó anteriormente, *spline* puede ser utilizado como argumento para la función `interp1` o como método directo a través de la misma sintaxis revisada. La diferencia entre la anterior y la actual es que esta utiliza "splines cúbicos" como su nombre indica; esto hace referencia al uso de ecuaciones de tercer grado para aproximar los valores del conjunto dado, de hecho, es el método más común de interpolación, ya que otorga buenas aproximaciones sin necesidad de demasiada complejidad computacional (similar a RK4).

Demostración

```
clc, clear, close all

x = 0:pi/4:2*pi;
v = cos(x);
xq = 0:pi/32:2*pi;
vq = spline(x,v,xq);

figure('Name','Demostración de spline');
plot(x,v,'o',xq,vq,':.');
xlim([0 2*pi]);
title('Interpolación por splines cúbicos');
```



Como se puede ver, con los mismos valores de entrada pero con el cambio de función, se puede obtener una mejor interpolación para la función determinada.

polyfit

Uso común → `polyfit(x,y,n)` → coeficientes de un polinomio

`x` → Puntos de muestra (arreglo, rango o vector lineal/*linspace*)

`y` → Función

`n` → Grado del polinomio

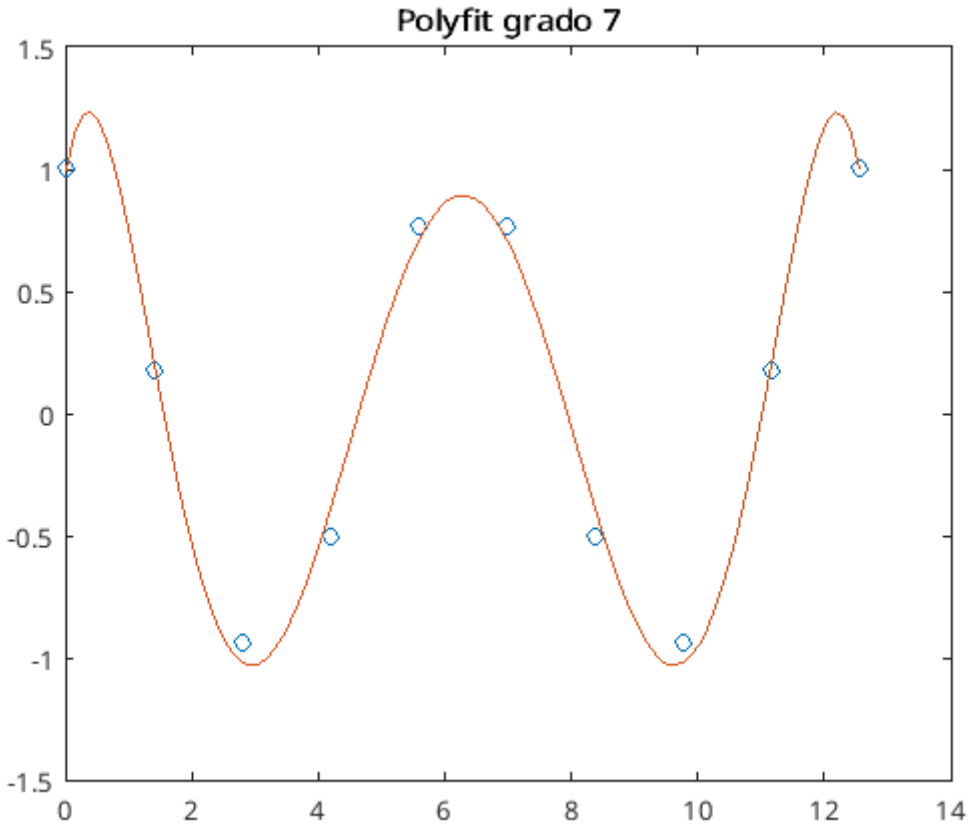
Esta función toma un rango de valores sobre el cual se evalúa una función determinada. El resultado será el conjunto de coeficientes correspondiente a un polinomio de grado *n* correspondiente a la forma $p_n x^n + p_{n-1} x^{n-1} + \ldots + p_0$, el cual se asimilaría a la función original.

Demostración

```
clc, clear, close all

x = linspace(0,4*pi,10);
y = cos(x);
n = 7;
p = polyfit(x,y,n);

x1 = linspace(0,4*pi);
y1 = polyval(p,x1);
figure
plot(x,y,'o');
hold on
plot(x1,y1);
title('Polyfit grado 7');
hold off
```



polyval

Uso común → `polyval(p,x)` → arreglo

`p` → Coeficientes de un polinomio; tamaño del arreglo ⇔ coeficiente del polinomio + 1

`x` → Puntos a evaluar dentro del polinomio

Esta función, aunque no se especifica como tal, opera de manera iterativa a través de un arreglo de valores, de forma que cada elemento del mismo evalúa un polinomio y devuelve el resultado al sustituir la variable independiente con cada elemento. Como se podrá haber percatado, esta función usualmente se usa en conjunto de `polyfit` para evaluar los coeficientes dados como resultado para posteriormente mostrar las evaluaciones finales en gráficas, tablas, archivos, etc.

Demostración

```
clc, clear

p = [12 13 8];
x = [1 1 2 3 5 8 13 21];
y = polyval(p, x)
```

```
y =

          33          33          82         155         373         880        2205        5573
```

plot

Uso común → `plot(x,y)` → gráfica bidimensional

`x` → Puntos de muestra (arreglo, rango o vector lineal/*linspace*)

`y` → Función *f*(*x*)

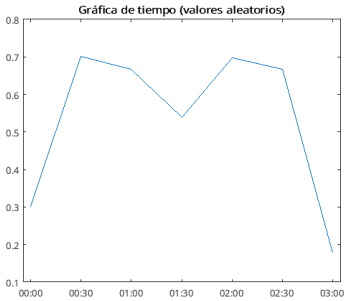
Una de las herramientas más poderosas de Matlab es la posibilidad de graficar conjuntos y evaluaciones de funciones determinadas sin mayor complicación, puesto que la actual se encarga del dimensionamiento, posición de los ejes, vista inicial y otros factores de manera automática.
Se muestra el uso más básico de la función, sin embargo, esta cuenta con un amplio espectro de customizaciones para modificar el comportamiento y apariencia de las gráficas generadas.
Su implementación usualmente viene acompañada de otras instrucciones integradas como `hold on/off` o `title` para alterar el flujo del programa donde se muestra y así dotar con mayores posibilidades al usuario.

Demostración

```
clc, clear, close all

t = 0:seconds(30):minutes(3);
y = rand(1,7);

plot(t,y,'DurationTickFormat','mm:ss');
title('Gráfica de tiempo (valores aleatorios)');
```



subplot

Uso común → `subplot(m,n,p)` → gráfica bidimensional múltiple

`m` → Unidades horizontal

`n` → Unidades verticales

`p` → Posición dentro de la cuadrícula

Por si no fuera suficiente, esta función extiende las características de `plot` al provisionar al usuario con la capacidad de posicionar dos gráficas posiblemente independientes dentro de un solo espacio, de forma que se pueden mostrar múltiples resultados dentro de una imagen compacta, lo cual resultaría útil en reportes o comparaciones dentro de un mismo contexto.
Su uso consta en la llamada a la función actual y luego a `plot`, de esta forma, Matlab comprende que se está intentando "apilar" múltiples gráficas a la vez y las distribuirá de acuerdo a los argumentos de entrada dados.

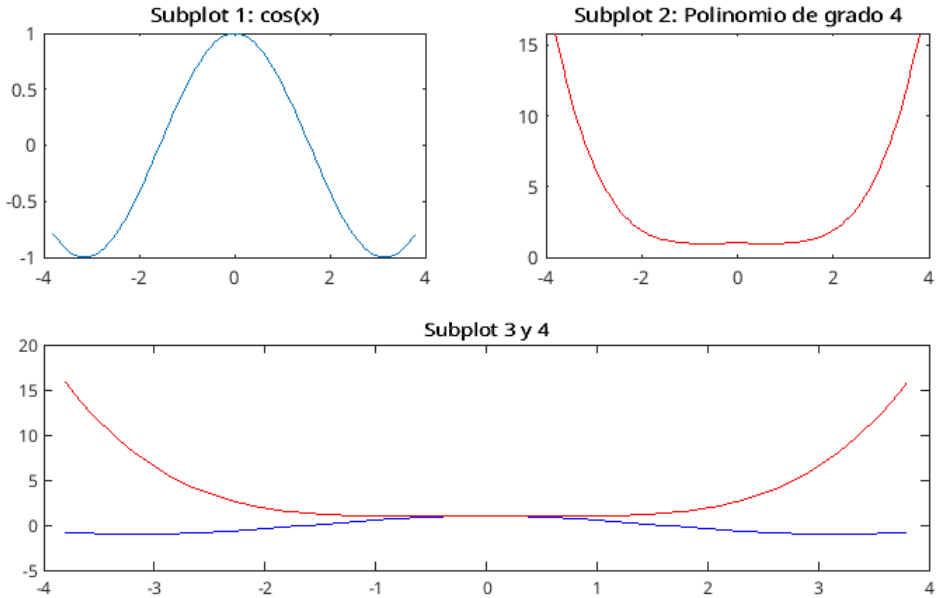
Demostración

```
clc, clear, close all

subplot(2,2,1);
x = linspace(-3.8,3.8);
y_cos = cos(x);
plot(x,y_cos);
title('Subplot 1: cos(x)');

subplot(2,2,2);
y_poly = 1 - x.^2./12 + x.^4./13;
plot(x,y_poly,'r');
title('Subplot 2: Polinomio de grado 4');

subplot(2,2,[3,4]);
plot(x,y_cos,'b',x,y_poly,'r');
title('Subplot 3 y 4');
```



integral

Uso común → `integral(@función,a,b)` → Valor numérico

`@función` → Función que se desea integrar

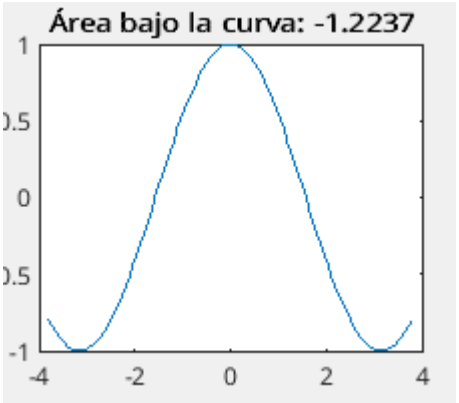
`a` → límite menor de integración

`b` → Límite mayor de integración

La función integral se puede entender como una herramienta para encontrar el área bajo una curva en un intervalo dado. La demostración a continuación ilustra cómo se puede usar la función integral para calcular y visualizar la integral definida de la función coseno en el intervalo [-3.8, 3.8].

Demostración

```
clc, clear, close all
f_cos = @(x) cos(x);
result_cos = integral(f_cos, -3.8, 3.8);
x = linspace(-3.8, 3.8);
y_cos = cos(x);subplot(2,2,1);
plot(x, y_cos);
title('Subplot 1: cos(x)');
subtitle = ['Área bajo la curva: ' num2str(result_cos)];
title(subtitle);
```



ode45

Uso común → `ode45(odefun,tspan,y0)` → matrices T y Y

`odefun` → Ecuación diferencial: ($y' = -y$)

`tspan` → Intervalo de tiempo [t_inicial, t_final]

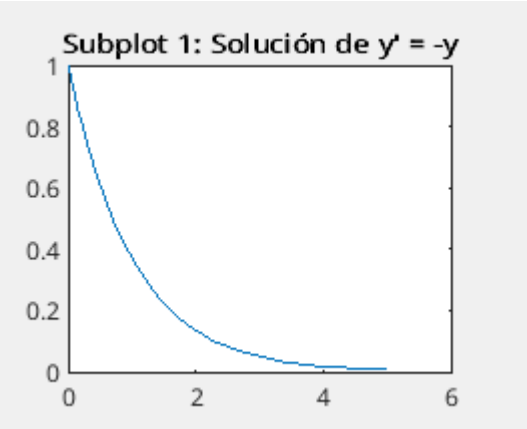
`y0` → Condición inicial ($y(t = 0) = 1$)

La función ode45 en MATLAB se utiliza para resolver ecuaciones diferenciales ordinarias (EDO) de primer orden. A continuación, se presenta un ejemplo de cómo usar ode45 para resolver una EDO y visualizar la solución.

Demostración

```
clc, clear, close all
ode = @(t, y) -y;
```

```
tspan = [0, 5];
y0 = 1;
[t, y] = ode45(ode, tspan, y0);
subplot(2,2,1);
plot(t, y);
title('Subplot 1: Solución de y'' = -y');
```



fsolve

Uso común → `fsolve(fun,x0,options)` → Valor numérico

`fun`	→ Una función que representa el sistema de ecuaciones no lineales que se desea resolver
`x0`	→ Vector que proporciona las suposiciones iniciales
`options`	→ Conjunto de opciones para personalizar el comportamiento

La función fsolve en MATLAB se utiliza para resolver sistemas de ecuaciones no lineales.

Demostración

```
clc, clear, close all
fun = @(x) x^2 - 4;
x0 = 2;
x_sol = fsolve(fun, x0);
subplot(2,2,1);
x_vals = linspace(0, 4, 100);
y_vals = fun(x_vals);
plot(x_vals, y_vals, 'b', x_sol, 0, 'ro');
title('Subplot 1: Función y su raíz');
```

cftool

Uso común → `cftool(x,y)` → Configuración para el ajuste de curvas

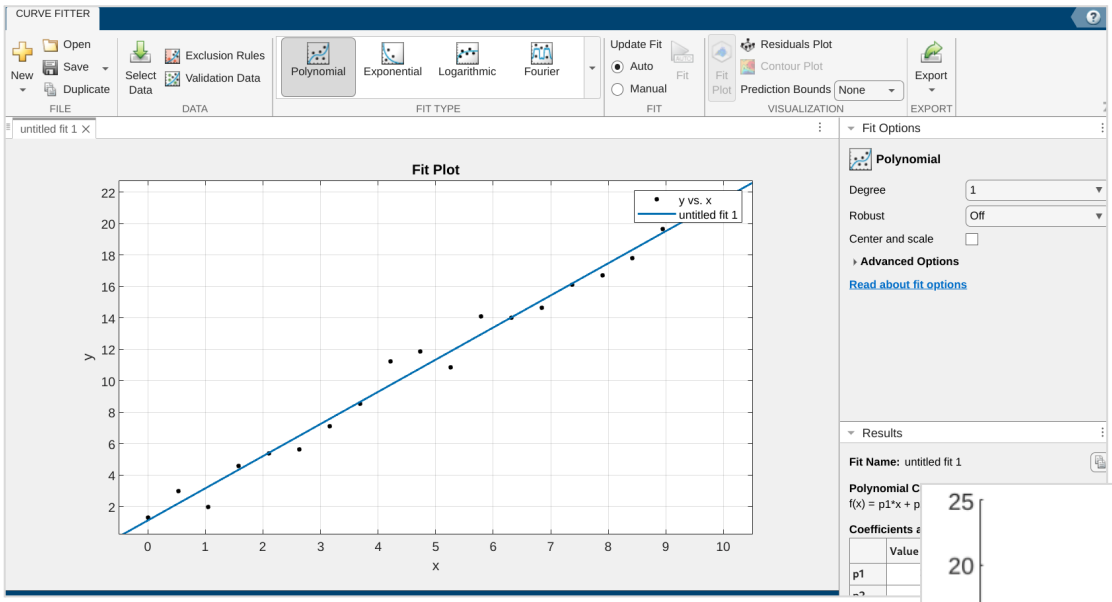
`x` → Es la variable independiente en el contexto de ajuste de curvas.

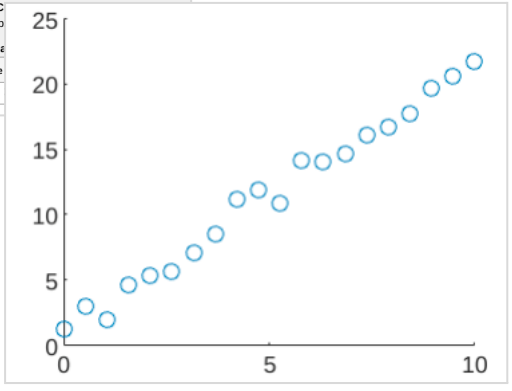
`y` → Es la variable dependiente. Representa los valores de salida que dependen de x.

Es una aplicación gráfica en MATLAB que facilita el ajuste de curvas a datos. Proporciona una interfaz interactiva para explorar diferentes ajustes de curvas y modelos.

Demostración

```
x = linspace(0, 10, 20)';
y = 2*x + 1 + 0.5*randn(size(x));
cftool(x, y);
fitresult = fit(x, y, 'poly1');
coefficients = coeffvalues(fitresult);
figure;
scatter(x, y, 'o', 'DisplayName', 'Datos');
hold on;
plot(fitresult, 'r-', 'DisplayName', 'Ajuste');
title('Ajuste de Curva con cftool');
xlabel('X');
ylabel('Y');
legend('show');
grid on;
```





robustfit

Uso común → `robustfit(x,y)` → Configuración para el ajuste de curvas

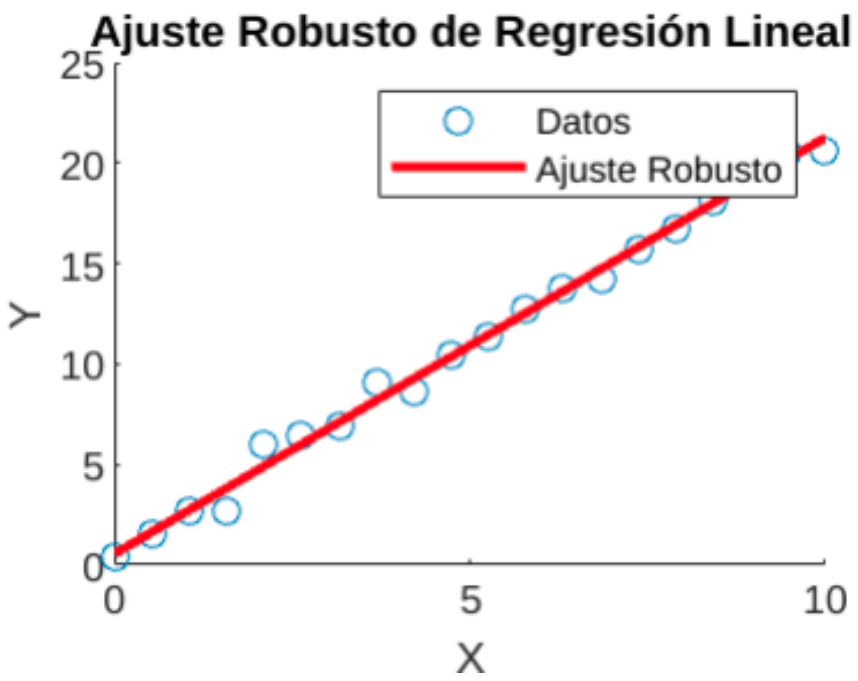
`x` → Vector de la variable independiente

`y` → Vector de la variable dependiente

La función robustfit realiza un ajuste de regresión lineal robusto a los datos, lo que significa que es menos sensible a los valores atípicos en comparación con el método de mínimos cuadrados ordinarios. La salida de la función incluye los coeficientes de regresión (b) y estadísticas adicionales (stats) que proporcionan información sobre la robustez del ajuste.

Demostración

```
x = linspace(0, 10, 20)';
y = 2*x + 1 + 0.5*randn(size(x));
[b, stats] = robustfit(x, y);
figure;
scatter(x, y, 'o');
hold on;
plot(x, b(1) + b(2)*x, 'r-', 'LineWidth', 2);
title('Ajuste Robusto de Regresión Lineal');
xlabel('X');
ylabel('Y');
legend('Datos', 'Ajuste Robusto');
```



Sustitución de método

En la vida, muchas situaciones parecen cíclicas, terminando un proceso exactamente donde empezó. A este concepto se le puede ver como una instancia de uróboros, el símbolo que representa la lucha eterna e incesable. Esto resulta apropiado para dar conclusión al desarrollo de la elaboración de este manual, puesto que el primer método numérico visto emerge del olvido para ser revisado una vez más. El mencionado no puede ser nada más y nada menos que *el **método de bisección*** (Cortés Rosas et al., 2019).

Este primitivo método utiliza un rango finito de valores y un error definido para *intentar* (puede que no converja) encontrar las raíces de una función. Revisando los métodos presentados en el documento actual, la función `fsolve` se encarga de hacer algo parecido. A continuación se presenta el código del método desarrollado y la invocación de la función integrada junto con la comparación de la cantidad de líneas.

```
% Restablecer entorno
clear, clc

% Variables comunes
f = @(x) x.^2 - 0.5;     % Función a evaluar
a = -10;                 % Límite por la izquierda
b = 10;                  % Límite por la derecha

% ---- Bisección ---- %
% Variables del método
le = 1;                  % Error absoluto actual
e = 0.0001;              % Tolerancia
bis = [];                % Puntos resultantes
helper = 0;              % Variable de ayuda
iters = 500;             % Iteraciones máximas antes de fallar

% Asegurar intervalo correcto
if (f(a)*f(b) > 0)       % Revisar convergencia inicial
   while (f(a) > 0)      % Modificar límite izquierdo
       a = a + 0.1; iters = iters - 1;
       if (iters == 0)
           fprintf('Límite inferior incorrecto.\n');
           return
       end
   end

   iters = 500;

   while (f(b) < 0)      % Modificar límite derecho
       b = b - 0.1; iters = iters - 1;
       if (iters == 0)
           fprintf('Límite superior incorrecto.\n');
           return
       end
   end

   if (a > b)            % Invertir límites si se requiere
       helper = a; a = b; b = helper;
   end
end

% Proceso de bisección
```

```
while (le > e)
   bis(end + 1) = (a + b) / 2;

   % Sustituir límites por nuevo valor
   if (f(bis(end)) < 0)
       a = bis(end);
   else
       b = bis(end);
   end

   % Evaluar error absoluto
   if (length(bis) > 1)
       le = abs(bis(end) - bis(end - 1));
   end
end
fprintf('Resultado de bisección\t-> %f\n', bis(end));

% ---- fsolve ---- %
xs = [a, b];                          % Puntos iniciales
opts = optimset('Display','off');     % Ocultar texto del resultado
easy_method = fsolve(f, xs, opts);    % Uso de la función
fprintf('Resultado de fsolve\t-> %f\n', easy_method);
```

```
Resultado de bisección  -> 0.707134
Resultado de fsolve     -> -0.707107
Resultado de fsolve     -> 0.707107
```

Cantidad de líneas para el método de bisección: 40
Cantidad de líneas para `fsolve`: 6 (reducibles a 1)

Como se puede apreciar, la función integrada reduce el número de líneas utilizadas casi 7 veces de la cantidad empleada para el método de bisección, eso contemplando que se hicieron algunos ajustes para reducir incluso más la cantidad de la implementación original.

Otro detalle de suma importancia es que en los resultados se muestran dos salidas para `fsolve`, esto no es un error, sino una capacidad agregada de la función, puesto que esta busca *todas las raíces* de la expresión matemática utilizada, mientras que el método de bisección solo es capaz de manejar una a la vez. En este caso, al ser una parábola el argumento utilizado, dos raíces son encontradas y posteriormente desplegadas.

Conclusiones

Guzmán Claustro, Edgar

Este ejercicio sirvió para recordar las funciones que más utilidad tienen en Matlab. Además de esto, este documento puede servir como futura referencia para nuevos trabajos donde se necesite realizar implementación con este lenguaje de programación. Sin duda es un lenguaje "sencillo" a manera que cuando profundizamos en él, podemos ser testigos de la complejidad que hay por detrás. Volver a programar esto, es volver a recordar, por así decirlo. Siempre es un placer aprender cosas nuevas y vivir nuevas experiencias. En este caso contar con el manual nos brinda una herramienta extra para el futuro.

Como comentario adicional, ajeno al documento. Quiero destacar el martirio que es utilizar Matlab dentro de plataformas cuyo soporte no es tan "basto". No quiero decir que deje de ser funcional, sin embargo, hace que personas como yo pierdan el interés. Nadie me obligó a utilizar un sistema operativo diferente a MacOS o Windows, fue por gusto propio y sufrimiento que yo mismo me busqué.

Cordero Hernández, Marco R.

La construcción de este manual ha servido para repasar gran parte del curso y su posible aplicación no solo en los métodos numéricos, sino en una amplia gama de problemas resolubles a través de la automatización, y, evidentemente, Matlab.

Ha sido de sumo interés analizar las funciones propuestas en el desarrollo de este trabajo, puesto que muchas de las funciones desconocidas hicieron sentido para muchos proyectos exógenos a la materia, y para las funciones que ya habían sido implementadas en otro momento, nuevas configuraciones útiles fueron descubiertas.

¿Qué es la esencia de la programación si no es el autodescubrimiento? Presentar las funciones actuales es un claro ejemplo de uno de los casos más clásicos: la reinvención de la rueda. Usualmente en proyectos arbitrarios se encuentra el lamentable caso en que se necesita cumplir con un objetivo que aparentemente luce complejo, empezando a obviar las soluciones esenciales y los métodos preexistentes. Sin indagar en su veracidad, el supuesto caso de la NASA y su problema con la escritura en el espacio demuestra esto claramente, en donde sus científicos se daban de topes por crear una pluma capaz de escribir en entornos sin gravedad, mientras que las agencias rusas usaban el lápiz y funcionaba a la perfección. Muchas veces las personas olvidan las raíces acompañadas del arduo trabajo que alguien más ya se tomó la molestia por hacer, y esa, es la enseñanza predominante que prevalece en este trabajo…

Bibliografía

The MathWorks Inc. (2023). MATLAB version: 9.14.0 (R2023a), Natick, Massachusetts: The MathWorks Inc. https://www.mathworks.com.

Cortés Rosas, J. J., González Cárdenas, M. E., Pinilla Morán, V. D., et al. (2019). Métodos cerrados para la solución numérica de ecuaciones algebraicas y trascendentes. Recuperado de https://www.ingenieria.unam.mx/pinilla/PE105117/pdfs/tema2/2-1_metodos_cerrados.pdf.