Programación Numérica en Geofísica

PNG 2020-1

Andrés Sepúlveda

Departamento de Geofísica
Universidad de Concepción

04 Agosto 2020

Anuncios

- Hoy: **Resolver Sistemas de Ecuaciones**

Sistemas de Ecuaciones

Lineales

- Estudiamos el caso general

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ \cdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= b_m \end{aligned}$$

  Donde a_{ij} son las constantes que multiplican las variables, $x_1, \cdots, x_n$ son los valores que buscamos obtener, y b_n son las valores finales de cada ecuación.

- Este puede ser representado de forma genérica como

$$A \cdot \overrightarrow{x} = \overrightarrow{b}$$

Sistemas de Ecuaciones

Lineales

- Donde
$$A \cdot \overrightarrow{x} = \overrightarrow{b}$$
- A es la matriz de coeficientes
$$A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix}$$
- $\overrightarrow{b}$ contiene el lado derecho de la ecuación,
- y $\overrightarrow{x}$ son las soluciones buscadas.

Sistemas de Ecuaciones

Introducción

- Para transformar un sistema lineal de ecuaciones

$$
\begin{aligned}
2x + y + z &= 2 \\
-x + y - z &= 3 \\
x + 2y + 3z &= -10
\end{aligned}
$$

- a la forma

$$A \cdot \overrightarrow{x} = \overrightarrow{b}$$

Sistemas de Ecuaciones

Lineales

linsolve

- Podemos construir la matriz explícitamente

```
A = [2 1 1; -1 1 -1; 1 2 3]
b = [2 3 -10]'
```

- Y usar *linsolve(A,b)* para encontrar m de $Am = b$

```
m = linsolve(A,b)

m =
  3
  1
 -5
```

- Donde los componentes de m son $x = 3, y = 1, z = -5$

Sistemas de Ecuaciones

Lineales

equationToMatrix

- También podemos usar la función *equationsToMatrix* (toolbox **symbolic** Matlab / Octave -> Python)
- Primero declaramos las ecuaciones

```
syms x y z
eqn1 = 2*x + y + z == 2;
eqn2 = -x + y - z ==3;
eqn3 = x + 2*y + 3*z == -10;
```

- Aplicamos la función

```
[A,b] = equationsToMatrix([eqn1, eqn2, eqn3],[x, y, z])
```

- Obteniendo

```
A =                 b = [ 2 3 -10]
[  2, 1,  1]
[ -1, 1, -1]
[  1, 2,  3]
```

Sistemas de Ecuaciones

Lineales

solve

- Alternativamente podemos usar la función **solve**.
- Primero declaramos el sistema de ecuaciones

```
syms x y z
eqn1 = 2*x + y + z == 2;
eqn2 = -x + y - z ==3;
eqn3 = x + 2*y + 3*z == -10;
```

- Después aplicamos la función

```
sol = solve([eqn1, eqn2, eqn3],[x, y, z])
```

- Obteniendo una **estructura** con las soluciones

```
sol =
  scalar structure containing the fields:
x = (sym) 3
z = (sym) -5
y = (sym) 1
```

Sistemas de Ecuaciones

Lineales

División de Matrices

- Alternativamente, la versión matricial

$$A \cdot \overrightarrow{x} = \overrightarrow{b}$$

  del problema

```
syms x y z
eqn1 = 2*x + y + z == 2;
eqn2 = -x + y - z ==3;
eqn3 = x + 2*y + 3*z == -10;
```

- obtenida mediante

```
[A,b] = equationsToMatrix([eqn1, eqn2, eqn3],[x, y, z])
```

(¡o a mano!)

- puede ser resuelta usando la llamada "división de matrices"

```
x = A\b
```

Sistemas de Ecuaciones

Desigualdades

solve

- **solve** sirve para desigualdades

$$\begin{aligned} -\infty &< x \\ x &< 10 \end{aligned}$$

- mediante

```
syms x
solve(x^2 - 1 > 0, x < 10)
```

- obteniendo

$$ans = (sym)(-\infty < x \wedge x < -1) \vee (1 < x \wedge x < 10)$$

Sistemas de Ecuaciones

Algebraicas

solve

- **solve** sirve para ecuaciones algebraicas

$$
\begin{aligned}
x^2y^2 &= 0 \\
x - \frac{y}{2} &= \alpha
\end{aligned}
$$

- mediante

```
syms x y a
[solx, soly] = solve(x^2*y^2 == 0, x-y/2 == a,[x y])
```

- obteniendo

```
solx = 0 a
soly = -2*a 0
```

Sistemas de Ecuaciones

Algebraicas

solve

- inclusive cuando hay múltiples soluciones

$$\begin{aligned} x^2 y^2 &= 1 \\ x - \frac{y}{2} &= \alpha \end{aligned}$$

- mediante

```
syms x y a
[solx, soly] = solve(x^2*y^2 == 1, x-y/2 == a, [x y])
```

- obtenemos

```
solx =                              soly
 a/2 - (a^2 - 2)^(1/2)/2            - a - (a^2 - 2)^(1/2)
 a/2 - (a^2 + 2)^(1/2)/2            - a - (a^2 + 2)^(1/2)
 a/2 + (a^2 - 2)^(1/2)/2             (a^2 - 2)^(1/2) - a
 a/2 + (a^2 + 2)^(1/2)/2             (a^2 + 2)^(1/2) - a
```

Sistemas de Ecuaciones

Inconsistentes

Sistemas subdeterminados

$$\begin{aligned} 6x_1 + 2x_2 + 3x_3 &= 0 \\ 4x_1 + x_2 - 2x_3 &= 0 \\ 2x_1 + x_2 + 5x_3 &= 0 \end{aligned}$$

¿Qué tiene de especial este sistema de ecuaciones?
¿Cuál es el rango de esta matriz?
¿Qué significa eso?

Sistemas de Ecuaciones

Infinitas Soluciones

```
 octave:6> x= A\b
warning: matrix singular to machine precision,
  rcond = 6.72862e-18
x =
   0
   0
   0
```

En estos casos hay que aplicar el proceso de eliminación Gaussiana, incorporado en la función **rref** (*reduced row echelon form*).

Sistemas de Ecuaciones

Inconsistentes

Infinitas Soluciones

```
>> rref([A b])

ans =

    1.0000         0    -3.5000         0
         0    1.0000    12.0000         0
         0         0          0         0
```

Es decir:

$$
\begin{aligned}
x_1 &= 3.5x_3 \\
x_2 &= -12x_3
\end{aligned}
$$

Sistemas de Ecuaciones

No lineales

fsolve

1. Consideremos las ecuaciones

$$
\begin{aligned}
e^{-e^{x_1+x_2}} &= x_2(1+x_1^2) \\
x_1\cos(x_2) + x_2\sin(x_1) &= \frac{1}{2}
\end{aligned}
$$

2. convertimos a la forma $F(x) = 0$ Consideremos las ecuaciones

$$
\begin{aligned}
e^{-e^{x_1+x_2}} - x_2(1+x_1^2) &= 0 \\
x_1\cos(x_2) + x_2\sin(x_1) - \frac{1}{2} &= 0
\end{aligned}
$$

3. Escribimos una función llamada **root2d**

```
 function F = root2d(x)

F(1) = exp(-exp(-(x(1)+x(2)))) - x(2)*(1+x(1)^2);
F(2) = x(1)*cos(x(2)) + x(2)*sin(x(1)) - 0.5;
```

Sistemas de Ecuaciones

No lineales

fsolve

1.
```
 function F = root2d(x)

F(1) = exp(-exp(-(x(1)+x(2)))) - x(2)*(1+x(1)^2);
F(2) = x(1)*cos(x(2)) + x(2)*sin(x(1)) - 0.5;
```

2. La solución se obtiene mediante

```
 fun = @root2d;
x0 = [0,0];
x = fsolve(fun,x0)

x =

    0.3532    0.6061
```

Sistemas de Ecuaciones

No lineales

fsolve

1. **fsolve** es una caja negra que contiene varios algoritmos. Consideremos

$$
\begin{aligned}
F_1(x) &= (x_1+1)(10-x_1)\frac{1+x_2^2}{1+x_2^2+x_2} \\
F_2(x) &= (x_2+2)(20-x_2)\frac{1+x_1^2}{1+x_1^2+x_1}
\end{aligned}
$$

```
function F = fbnd(x)

F(1) = (x(1)+1)*(10-x(1))*(1+x(2)^2)/(1+x(2)^2+x(2));
F(2) = (x(2)+2)*(20-x(2))*(1+x(1)^2)/(1+x(1)^2+x(1));
```

2. Las soluciones son:
(-1,-2), (10,-2), (-1, 20), (10,20)

Sistemas de Ecuaciones

No lineales

fsolve

1. Definimos un punto inicial para la búsqueda de soluciones $x0 = [1, 9]$
2. ```
   x0 = [1,9];
   opts = optimoptions(@fsolve,'Display','off',...
       'Algorithm','trust-region-dogleg');
   x1 = fsolve(@fbnd,x0,opts)
   x1 =   -1.0000   -2.0000

   opts.Algorithm = 'trust-region';
   x2 = fsolve(@fbnd,x0,opts)
   x2 =   -1.0000   20.0000

   opts.Algorithm = 'levenberg-marquardt';
   x3 = fsolve(@fbnd,x0,opts)
   x3 =    0.9523    8.9941             <---- ¡Malo!
   ```
   ```

Otras funciones

Raíces

roots

Un polinomio puede ser descrito como

$$f(x) = \sum_{0}^{n} a_i x^i$$

La función **roots**, usando los a_i en orden *descendiente*, entrega las soluciones.

$$y = 3x^3 + x^2 - 5$$

fzero

Llamamos esta función con
fzero(fun,x_o), donde *fun* es la función que describe la ecuación
$x_\circ$ es el punto de inicio. Es importante escoger $x_\circ$ de forma adecuada.
Ejemplo:

$$f(x) = \sin(10x)e^{-\sqrt{x}}$$

y $x_\circ$ 0.8, 1.0, 1.2.
Grafíque la función.

Newton-Raphson

Consideremos la expansión en series de Taylor de una función alrededor de $x = x_\circ$

$$f(x) = f(x_\circ) + f'(x_\circ)(x - x_\circ) + (f''(x_\circ)/2!)(x - x_\circ)^2 + \cdots$$

Usando sólo los dos primeros términos de la ecuación para buscar la raiz de $f(x)$

$$f(x) = 0$$

es decir

$$f(x) = 0 \approx f(x_\circ) + f'(x_\circ)(x_1 - x_\circ)$$

o

$$x_1 = x_\circ - f(x_\circ)/f'(x_\circ)$$

e iterativamente

$$x_2 = x_1 - f(x_1)/f'(x_1)$$

$$x_{i+1} = x_i - f(x_i)/f'(x_i)$$

hasta que converja

$$|f(x_{i+1}| < \epsilon$$

¡Programar!

```
function [x,iter]=newton(x0,f,fp)  % newton-raphson algorithm
N = 100; eps = 1.e-5; %  define max. no. iterations and error
maxval = 10000.0;     %  define value for divergence
xx = x0;
while (N>0)
   xn = xx-f(xx)/fp(xx);
if abs(f(xn))<eps
      x=xn;iter=100-N;
return;
end;
if abs(f(xx))>maxval
      disp(['iterations = ',num2str(iter)]);
      error('Solution diverges');
break;
end;
   N = N - 1;
   xx = xn;
end;
error('No convergence');
break;
% end function
```