Ayudantía N°2

Magíster Economía PUC

Teoría Macroeconomómica I

Estudiante: Valentina Andrade de la Horra Profesor: Alexandre Janiak

1.Operaciones con variables aleatorias

Modificamos la matrix xx.m de la ayudantía pasada y creamos una matriz_2.m que contiene como *inputs* el tamaño del vector (fila y columna) y un parámetro i = {1,2,3,4} que determina cuál de las siguientes distribuciones será la generadora de dichos números, donde los *outputs* son

1. $X_1 = X \sim U\,(1,4)$
2. $X_1 = X \sim X^2_3$
3. $X_1 = X \sim \exp(1.1)$
4. $X_1 = X \sim t - \text{student}\ 2\ \text{gl}$

1.La cantidad de números aleatorios que necesita para llegar a sumar 20.

Debemos construir una funcion que me devuelva "vector" (r,c) para que para cada modelo me indique cuantas veces se debe sumar para sumar 20. Partimos de valores auxiliares

```
[vector, modelos] = matriz_2(100,1,2);
```

```
Modelo 2
```

```
A=1:4;
```

Explicación: La programación dice que para todo modelo A = [1:4] se alcanza la suma == 20 a partir de la cantidad de filas (cantidad (i)) que corresponde al vector j (dinámico). Entonces se van sumando j+1 elementos hasta alcanzar 20 para cada modelo que se estima (en i y j =1 agarra el primer elemento de la primera columna)

```
for i=1:length(A)% Para todo modelo (variable aleatoria)
        suma=0;
        j=0;
        while suma<=20 % hasta 20
           j=j+1; % En j + elemento (dinámico)
           suma=suma+modelos(j,i);  %suma 0 + modelos(j,i)
           cantidad(i)=j; %sumas(i)=suma
                break;
        end
      disp(['Usando el modelo ',num2str(A(i)),' se requiere sumar ',num2str(cantidad(i)),' vece
end
```

```
Usando el modelo 1 se requiere sumar 1 veces para alcanzar 20.
Usando el modelo 2 se requiere sumar 1 veces para alcanzar 20.
Usando el modelo 3 se requiere sumar 1 veces para alcanzar 20.
Usando el modelo 4 se requiere sumar 1 veces para alcanzar 20.
```

2. La cantidad de números aleatorios que necesita para alcanzar un número mayor que uno.

```
for i=1:length(A)
       j=1;
        while modelos(j,i)<1
            elemento=modelos(j,i);
            columna=find(modelos(:,i)==elemento);
            j=j+1;
        end
        disp(['Usando el modelo ',num2str(A(i)),' se requiere tomar ',num2str(j),' veces para a
end
```

```
Usando el modelo 1 se requiere tomar 1 veces para alcanzar un valor mayor que uno.
Usando el modelo 2 se requiere tomar 2 veces para alcanzar un valor mayor que uno.
Usando el modelo 3 se requiere tomar 21 veces para alcanzar un valor mayor que uno.
Usando el modelo 4 se requiere tomar 1 veces para alcanzar un valor mayor que uno.
```

3.La cantidad de números aleatorios que necesita para que la media del vector aleatorio sea al menor que 0.7

```
for i=1:length(A)
       j=1;
       media=0;
        while media < 0.7 % Hasta media menor 0.7
           media=mean(modelos(1:j,i));
           j=j+1;
            if j>100
                break;
            end
        end
       disp(['Usando el modelo ', num2str(A(i)),' requerimos tomar ',num2str(j),' numeros para
end
```

```
Usando el modelo 1 requerimos tomar 2 numeros para la media requerida
Usando el modelo 2 requerimos tomar 5 numeros para la media requerida
Usando el modelo 3 requerimos tomar 101 numeros para la media requerida
Usando el modelo 4 requerimos tomar 2 numeros para la media requerida
```

```
clc
clear
```

2.Solución sistemas no lineales

(a) Ingresamos el vector $[a_1, a_2, b_1, b_2, s_1, s_2] = [2, 1, 1, 2, 1, 1]$ y un par de valores iniciales en un vector Ini_{1x2}.

Reescribimos la función que se presenta acontinuación como $f(x) = 0$ para los parámetros solicitados

$$a_1 x - a_2 y = e^{-s_1 x}$$

$$b_1 x - b_2 y = e^{-s_1 y}$$

Resolvemos

$$f(x) = \begin{cases} a_1 x - a_2 y - e^{-s_1 x} = 0 \quad (1) \\ b_1 x - b_2 y - e^{-s_1 y} = 0 \quad (2) \end{cases}$$

```
x_i = [2 1 1 2 1 1] % [a1 a2 b1 b2 s1 s2]
```

```
x_i = 1×6
     2     1     1     2     1     1
```

```
ini=[5,5];

% Parametros (indexacion)
a1=x_i(1,1);
a2=x_i(1,2);
b1=x_i(1,3);
b2=x_i(1,4);
s1=x_i(1,5);
s2=x_i(1,6);

xa=linspace(-10,20,1000);
ya=linspace(-10,20,1000);
[x,y]=meshgrid(xa,ya);

%Funciones
f1=a1*x - a2*y -exp(-s1*x);
f2=-b1*x + b2*y -exp(-s2*y);
```

(b) Gráfico

```
figure(1)
contour(x,y,f1,[1,1],'red');
hold on;
grid on;
contour(x,y,f2,[1,1],'blue');
set(gcf,'color','w');
title('Sistema 1')
hold off;
```

(c) Resolvemos el sistema de ecuaciones y mostraremos **iterativamente** cómo llega al resultado mediante programación de ciclos (método de iteración). Indicaremos cuantas iteraciones nos toma resolver el sistema

```
tol=0.0003;
x0=ini(1);
y0=ini(2);
x1=10;
y1=10;
diff=10;
i=0;
while diff>tol
i=i+1;
x1=x0-(a1*x0 - b1*y0 -exp(-s1*x0));
```

```
y1=y0-(-a2*x0 + b2*y0 -exp(-s2*y0));
diff=abs(max((x1-x0),(y1-y0)));
XX(i,1)=x1;
YY(i,1)=y1;
x0=x1;
y0=y1;
end

F=['Solucion: ',num2str(x0),' ',num2str(y0), 'Iteraciones', i ]
figure(2)
plot(XX);
grid on;
title('Convergencia del sistema al equilibrio')
set(gcf,'color','w');
```

(d) Repetimos (b) empleando las opciones de la función *fsolve*

```
x0 = [5;5]; options = optimoptions('fsolve','Display','iter');
[x,fval] = fsolve(@myfunction,x0,options)
```

Iteration	Func-count	f(x)	Norm of step	First-order optimality	Trust-region radius
0	3	146.081		16.9	1
1	6	109.73	1	13.6	1
2	9	43.8189	2.5	8.04	2.5
3	12	3.81035	4.35451	3.62	6.25
4	13	3.81035	10.8863	3.62	10.9
5	14	3.81035	2.72157	3.62	2.72
6	17	3.20678	0.680393	1.42	0.68
7	20	3.08172	0.680393	1.49	0.68
8	21	3.08172	0.680393	1.49	0.68
9	24	2.65449	0.170098	0.467	0.17
10	27	2.64828	0.170098	0.562	0.17
11	28	2.64828	0.170098	0.562	0.17
12	31	2.61035	0.0425246	0.182	0.0425
13	32	2.61035	0.106311	0.182	0.106
14	35	2.6046	0.0265778	0.0407	0.0266
15	36	2.6046	0.0265778	0.0407	0.0266
16	39	2.60423	0.00664446	0.0249	0.00664
17	40	2.60423	0.0166112	0.0249	0.0166
18	43	2.6041	0.00415279	0.0124	0.00415
19	46	2.60406	0.00415279	0.00782	0.00415
20	47	2.60406	0.00415279	0.00782	0.00415
21	50	2.60405	0.0010382	0.001	0.00104
22	51	2.60405	0.0010382	0.001	0.00104
23	54	2.60405	0.000259549	0.000692	0.00026
24	57	2.60405	0.000259549	0.00071	0.00026
25	58	2.60405	0.000259549	0.00071	0.00026
26	61	2.60405	6.48873e-05	0.000184	6.49e-05
27	62	2.60405	0.000162218	0.000184	0.000162
28	65	2.60405	4.05546e-05	0.00011	4.06e-05
29	66	2.60405	4.05546e-05	0.00011	4.06e-05
30	69	2.60405	1.01386e-05	2.07e-05	1.01e-05
31	70	2.60405	2.53466e-05	2.07e-05	2.53e-05
32	73	2.60405	6.33665e-06	1.41e-05	6.34e-06
33	74	2.60405	6.33665e-06	1.41e-05	6.34e-06
34	77	2.60405	1.58416e-06	6.06e-06	1.58e-06
35	80	2.60405	1.58416e-06	4.41e-06	1.58e-06
36	81	2.60405	1.58416e-06	4.41e-06	1.58e-06
37	84	2.60405	3.96041e-07	9.7e-07	3.96e-07

```
No solution found.

fsolve stopped because the problem appears regular as measured by the gradient,
but the vector of function values is not near zero as measured by the
value of the function tolerance.

<stopping criteria details>
x = 2×1
    0.1040
   -0.6931
fval = 2×1
   -0.0000
   -1.6137
```

(e) Empleamos las funciones *fzero* y *fsolve* para resolver $f(x) = x^5 - 5x^2 + 8x - 5x^{\frac{1}{2}} = 2$

fzero es una función que nos dice las raices de la función no lineal, es decir donde la función cruza el eje x.

```
fun = @(x) x.^5 -5*x + 8*x -5*x.^0.5 - 2;
xo = 2; % punto inicial
z = fzero(@f, xo)
```

```
z = 1.3058
```

También cree la función @f (en f.m) pero para algo tan pequeño siento que se "pierde" más de lo que se gana

Como $f(x)$ es un polinomio, podemos encontrar el mismo cero de la función y un par de ceros complejos usando el comando *roots*

```
roots([1 -5 8 -5 -2])
```

```
ans = 4×1 complex
   2.9328 + 0.0000i
   1.1668 + 1.0944i
   1.1668 - 1.0944i
  -0.2665 + 0.0000i
```

También los podemos graficar

```
options = optimset('PlotFcns',{@optimplotx,@optimplotfval});
z = fzero(@f, xo, options)
```

fsolve es una función que permite conocer las raices de una solución a partir de la iteración de variables

```
options = optimoptions('fsolve','Display','iter');
[x,fval] = fsolve(@f,x0,options)
fimplicit(@f)
```

Como podemos notar **fsolve** tiene dos soluciones, mientras que **fzero** tiene solo una. Esto es básicamente pues las raices de la solución en **fsolve** pueden estar cerca del cero pero no ser cero. Entonces, podemos notar que **fsolve** es un algoritmo que soluciona un sistema de ecuaciones o una ecuaciín con un grado de tolerancia, y que se basa su modo de estimación en una iteración de parámetros mientras que **fzero** no.

```
clear all
```

3.Aproximación del numero e

En este caso se pide estimar mediante el procedimiento de **simulación de Montecarlo** basado en *Estimating the Value of e by Simulation, Russell* *el número e* (*número de Euler*). La idea general es que $E\left(Z\right)=e$ donde *Z* es una variable aleatoria que se define de la siguiente manera:

$$Z=\min\left\{n\mid\sum_{i=1}X_i>1\right\},X_i\sim U\left(0,1\right)$$

Entonces, Z representa el mínimo de *n* que alcanza cuando la suma de las variables aleatorias es mayor que 1. En resumidas palabras, se tiene una secuencia infinita de variables aleatorias *X*1, *X*2…, con distribución uniforme en [0,1]. Sea Z el menor entero *n* tal que la suma de las primeras *n* observaciones es mayor que 1. Este resultado permite estimar el valor de la constante por medio de simulaciones aleatorias (Russel, 1991)

Generaremos un programa que aproxime el numero *e* según lo planteado, es decir, comprararemos lo obtenido en una simulación con respecto al número *e* (en términos reltivos) donde debe utilizar una tolerancia de 10^{-6}.

```
% 1. Elegir numero de "runs" y parametros
N = 10000000;
n = N;
z = 0;
i = 0;
maxl = 0;
f = 0;
%2. Simulacion
while n > 0 % tolerancia no aparece en paper (10^-6) en ese caso da 1.
    z = z + rand;
    i = i + 1;
    if z > 1
        if i > maxl
            f(i) = 1;
            maxl = i;
        else
            f(i) = f(i) + 1;
        end
        i = 0;
       z = 0;
        n = n - 1;
    end
end

%Grafico
disp ((1:maxl)*f'/sum(f))
bar(f/sum(f))
grid on

%Comparacion intervalo de confianza
f/sum(f)
```

Entonces veremos que el valor es aproximadamente 2.7

Otras formas más eficientes puede ser como las siguientes, inspirados en este libro. Ocuparé sobolset que estima una "quasirandom" point set de Sobol

```
z = 2; %valor esperado
for i=2:10
    p=sobolset(i);
    N = 10000;
    X=net(p,N)';
    z = z + (sum(sum(X)<1)/N);
end
disp(z)
```

El valor es 2.7117

```
clear all
```

4. Optimización y simulación Montecarlo

Suponga una función de producción

$$F(L,K) = A * L^{0.4} K^{0.4}$$

Donde *A* es un parámetro fijo, *L* es la cantidad de horas trabajadas en cada puesto y *K* es un indicador del nivel de capital de la firma

(a) Graficaremos las **isocuantas** de esta función de producción, con A = 100. Para hacerlo considere la función G(L,K). Emplearemos vectores con valores equidistantes de intervalo [0,10] en R^{100} para cada variable K y L.

```
% Numero de grillas
A = 100;
x1 = linspace(0,10,1000);
x2 = linspace(0,10,1000);
alpha = 0.4;
beta = 1- alpha;
% This creates all possible combinations of the x1 and x2 vectors, fills up the grid
[L,K] = meshgrid(x1, x2);
% Evaluate the utility function at all x1 and x2 combination points
G =A*L.^alpha.*K.^beta;

% Graph curve
close all;
figure();
mesh(L,K,G);
% Labeling
xlabel('Labor');
ylabel('Capital');
zlabel('Cobb Douglas Utility');
title('Utility Function Along Capital and Labor')
```

(b) Compute una función de costos C(X), donde X es un vector conformado por valores de (L,K), y una función de producción $f(A, X)$.

Cada puesto de trabajo paga *w = 50.000* por hora trabajada normal (hasta 160 horas por periodo), y paga *w*1.5* por hora extra. Supondremos que la tasa de interés es *r = 0.05*

La elección por las firmas se demora de la siguiente forma

$B = \{(L, K) : L \geq 0, K \geq 0, \mathrm{wL} + \mathrm{rK} \leq M\}$

Donde M son los recursos totales que tiene la firma. Además la empresa enfrenta una función de demanda dada por p = 3z donde z es un parámetro de escala de la demanda que recibe la empresa. (c) Solución el problema de la empresa para estos parámetros, con z = 1. Calcularemos los beneficios en el óptimo

```
w = 50000;
if L < 160
    wage = w
else
    wage = w*1.5;
end
r=0.05;
% Evaluar los costos
C = L*wage + K*r;
% Graph
figure();
contour = contourf(L, K, 3*C,5);
clabel(contour);
% Labeling
xlabel('Labor');
ylabel('Capital');
zlabel('Cost');
title('Contour Plot of Budget Set over Labor and Capital')
```

(d) Supondremos que z es una variable aleatoria con distribución $\frac{\exp(5)}{100+1}$. Primero queremos conocer su distribución pues esto define el *estado* que luego será utilizado en nuestra simulación Montecarlo.

Las $X_i \sim$ exponencial$(\alpha), F(X_i) = \alpha \cdot e^{-x_i \cdot \alpha}, E(X_i) = \alpha^{-1}$ y $V(X_i) = \alpha^{-2}$ donde $\alpha = 5$ y $n = 1000$. Obtendremos X_i que distribuye exponencial a partir de la simulación para cada *i* en *n = 1000* simulaciones

$\alpha = 5, \mathrm{runs} = 1000$

```
runs=1000;
lambda=5;
x_i=[lambda * (exp(-lambda * randn(runs, 1))/101)];

edges = linspace(0, 1, 21); % Create 20 bins.
% Plot the histogram.
histogram(x_i, 'BinEdges',edges);
% Fancy up the graph.
grid on;
xlim([0, 1]);
```

```
xlabel('X', 'FontSize', 14);
ylabel('Simulations', 'FontSize', 14);
title('Histogram of random variable', 'FontSize', 14);
```

Ahora, estimamos empleado el método de **Montecarlo** con 1000 simulaciones los beneficios promedios, el beneficio en el percentil 90 y el 10

```
options = optimset('PlotFcns',@optimplotfval)
for i=1:runs
    z=x_i(i)/100+1;
profit=@(x)-(cobbdouglas(A,x)*3*z-costs(w,r,x));
x0=[10,10,100];
x=fminsearch(profit,x0, options);
profits(i,1)=-profit(x);
end

display('Firm profit expectation is in:');
mean(profits)
prctile(profits,90)
prctile(profits,10)

sort(beneficios);
display('Minimo ultimo quintil beneficios')
val80=benefordenados(round(simulaciones*0.8),1)
```

(e) ¿Cuál es el mínimo valor que tendrá el conjunto del 20% de mayores beneficios?

```
benefordenados =sort(profits);
benefordenados(round(simulaciones*0.8),1)
```

(f) Adicional: Indifference Curves

```
figure();
% Contour plot, the fourth parameter are at what utility values we want to see the contour line
% All consumption bundle along the same contour line gives the same utility, hence they are: In
contour_u = contourf(L, K, G);
clabel(contour_u);
% Labeling
colormap('white')
xlabel('Labor');
ylabel('Capital');
zlabel('Cobb Douglas Utility');
title('Utility Function Along Labor and Capital Dimensions and Budget')
```

```
clear all
```

Seguimiento II: MATLAB

Para el siguiente sistema de ecuaciones en (x,y,z):

$$f(x) = \begin{cases} a_1x + b_1\text{ry} + c_1\text{rz} = s_1 & (1) \\ a_2x + b_2y + c_2(2r-1)z = s_2 & (2) \\ a_3x + b_3(r-1)y + c_3\text{rz} = s_3 & (3) \end{cases}$$

Donde $A = \begin{bmatrix} 1 & -4 & 6 \\ -4 & 5 & 18 \\ 1 & 2 & 2 \end{bmatrix}, S = [1 \quad 2 \quad 3]$ y $r = [1 \quad 2 \quad 3 \quad 4]$

(a) Ingresar las matrices A_{3x3} de coeficientes r_{1x4} de parámetros y s_{1x3} de soluciones

```
A = [1 -4 6
    -4 5 18
    1 2 2];
s = [1
    2
    3];
r = [1 2 3 4];
A = @(r) [1,-4*r,6*r;-4, 5, 18*(2*r-1); 1, 2*(r-1), 2*r];
```

(b) Dados los valores de A y S, elaboramos un programa que determina el sistema de ecuaciones (1) tiene solución para cada valor de r.

Para ello crearemos una matriz aumentada, de la cual evaluaremos si A es singular, en cuyo caso la solución para Ax = s o bien no existe o no es única. Para ello verificaremos si el rango de la matriz A el rango de esta matriz aumentada (As)

```
for r = 1:4
    A(r);
    As = [A(r) s]; % Crearemos una matriz aumentada
% checking the ranks
if rank(A(r)) == rank(As)
    disp(['Cuando r =',num2str(r), 'La solución única existe'])
else
    disp(['Cuando r =',num2str(r), 'La solución única no existe'])
end
end
```

(c) Armar una interfaz que comuniqué al usuario qué tipo de sistemas está analizando: indeterminado, determinado, para cada valor de r, y la matriz A resultados. y (d) Resolver el sistema para r

```
for r = 1:4
    A(r);
    As = [A(r) s]; % Crearemos una matriz aumentada
    x_inv = inv(A(r))*s; % Podemos dar las soluciones con
% checking the ranks
if rank(A(r)) == rank(As)
    disp(['Cuando r =',num2str(r), ' el sistema es determinado. Para la matriz'])
```

```
    disp(A(r))
    disp('Las soluciones son')
    disp( x_inv)
else
    disp(['Cuando r =',num2str(r), 'sistema indeterminado para la matriz'])
    disp(A(r))
end
end
```

(e) Agregamos el comando que evalúa si (d) es correcto empleando el comando linsolve e informamos al usuario el resultado o que hay errores

```
for r = 1:4
    A(r);
    As = [A(r) s]; % Crearemos una matriz aumentada
    x_inv = inv(A(r))*s; % Podemos dar las soluciones con
    x_lin = linsolve(A(r), s);
% checking the ranks
if rank(A(r)) == rank(As)
    disp(['Cuando r =',num2str(r), ' el sistema es determinado. Para la matriz'])
    disp(A(r))
    disp('Las soluciones son')
    disp( x_inv)
    if x_inv == x_lin
        disp('Resultado consistente')
    else
        disp('Resultado no consistente')
    end
elseif rank(A(r)) ~= rank(As)
    disp(['Cuando r =',num2str(r), 'sistema indeterminado para la matriz'])
    disp(A(r))
end
end
```

Bibliografía

Ma, Dan (12 de diciembre de 2015). «A randomized definition of the natural logarithm constant». *Probability and Stats*.

Russell, K. G. (febrero de 1991). «Estimating the Value of e by Simulation». *The American Statistician* **45** (1): 66-68.

Referencias adicionales

Se realiza demostración de Euler en archivo aqui