

Universidad Nacional Hermilio Valdizán
Facultad de Ingeniería Industrial y de Sistemas
Escuela Profesional De Ingeniería De Sistemas



En este informe se está presentando los ejercicios del libro de Chapra del capítulo 1 al 16

Integrante:
Rios Nolasco, Angelo Patrick

Docente:
Ing. Jorge Chávez Estrada

Curso:
Métodos numéricos

Huánuco – Perú
2022

Índice

Capítulo I Modelos matemáticos y solución de problemas en ingeniería.....	2
Capítulo II Programación y software.....	2
Capítulo III Aproximaciones y errores de redondeo	15
Capítulo IV Errores de truncamiento y la serie de Taylor	20
Capítulo V Métodos cerrados.....	28
Capítulo VI Métodos abiertos.....	44
Capítulo VII Raíces de polinomios.....	49
Capítulo VIII Estudio de casos: raíces de ecuaciones	54
Capítulo IX Eliminación de Gauss.....	63
Capítulo X Descomposición LU e inversa de matrices.....	72
Capítulo XI Matrices especiales y el método de Gauss-Seidel.....	82
Capítulo XII Estudio de casos: ecuaciones algebraicas lineales	87
Capítulo XIII Optimización unidimensional no restringida	91
Capítulo XIV Optimización multidimensional no restringida	95
Capítulo XV Optimización restringida	102
Capítulo XVI Aplicaciones en ingeniería: optimización.....	115

Capítulo I

Modelos matemáticos y solución de problemas en ingeniería

1.1

```
% EJERCICIO 1.1
Pesoagua = 60/100;           Pesocorporal = 100/100;
plasma_agua = 4.5/100;       plasma_corporal = 7.5/100;
tejido_agua = 4.5/100;       tejido_corporal = 7.5/100;
linfa_agua = 12/100;         linfa_corporal = 20/100;
huesos_agua = 4.5/100;       huesos_corporal = 7.5/100;
intracelular_agua = 33/100;  transcelular_corporal = 2.5/100;

disp('El porcentaje del peso total corporal debe corresponder al agua transcelular es');
x=60-(4.5 + 4.5 + 12 + 4.5 + 33);
disp(strcat(num2str(x),'%'));
disp('El porcentaje del total de agua del cuerpo debe corresponder al agua intracelular es');
y=100-(7.5 + 7.5 + 20 + 7.5 + 2.5);
disp(strcat(num2str(y),'%'));
```

1.2

```
% EJERCICIO 1.2
salon = [240 256000];
total_estudiantes = [2.25 2400]; %30 2.25 VOLUMEN 2400 POTENCIA= W/t
T1 = 20; %C
TIK = T1+273;
T2 = 0;
MwT = 28.97;
P = 101.325;
V = salon([1]);
R = 8.314;
t = 15; % 15 min a s
%Capacidad calorifica
Cv_aire = 718; % 0,718 KJ A J

%ley de gas ideal
m = P*V*MwT/(8.314*293);

%Calor sensible a volumen constante
Q = m*Cv_aire*(T2-T1);
%despejando
for i=1:1:t
    T3(i) = (total_estudiantes([2])*i*60)/(m*Cv_aire);
    disp('tiempo min');
    disp(i);
    disp(['La temperatura varia en C°']);
    disp(T3(i));
    tiempo = [1:i];
    plot(tiempo,T3);
    xlabel('Tiempo min');
    ylabel('Variación de temperatura °C');
end
```

1.3

```
% EJERCICIO 1.3
balance_inicial = 1512.33; %01/05
%Fecha respectiva : 01/06 01/07 01/08 01/09
depositos = [220.13 216.80 450.25 127.31];
retiros = [327.26 378.61 106.88 350.61];
cant_fecha = 4;
for i=1:1:4

    balance_anterior = balance_inicial;
    balance_final(i) = depositos(i)+ balance_anterior - retiros(i);
    balance_inicial = balance_final(i);
    tt = [1:i];
    fprintf('Balance de cada mes %8.3f\n',balance_final(i));
    plot(tt,balance_final);
end

if balance_inicial == balance_anterior;
    disp('El cálculo es de estado estacionario');
else
    disp('El cálculo es de estado transitorio');
end
```

1.4

```
% EJERCICIO 1.4
%Teorema de la continuidad volumetrica
% V1.A1=V2.A2=V3.A3=0  V1.A1=V2.A2+V3.A3
%Flujo volumétrico   Q = V.A;
Flujo_ent = 40;
Flujo_sal2 = 20;
Flujo_sal3 = (Flujo_ent- Flujo_sal2)/6;
fprintf('El área requerida en el tubo 3 es %.2f\n', Flujo_sal3);
```

1.5

```
% EJERCICIO 1.5
%{
GANAGUA EL CUERPO
1L 0.3L 0.05
PIERDE AGUA EL CUERPO
0.4L 0.2L 1.4L 0.2L 0.35L
CONDICION ESTADO ESTACIONARIO
suma_GANAGUA = suma_PIERDE AGUA
%}

Gana = [1 0.3 0.05 0 0.0];
Pierde = [0.4 0.2 1.4 0.2 0.35];
Gana([5]) = sum(Pierde) - sum(Gana);

disp('La cantidad de agua que se debe tomar por dia (L)');
disp(sum(Gana));
```

1.6

```
% EJERCICIO 1.6
%{
Paracaista en caida libre
V(ti+1) = V(ti) + [g-(c/m)*V(ti)](ti+1-ti)
%}

masa_prcaidal = 70;
coef_arrastrel = 12;
g = 9.81;
t_1 = 10;
t = [1:20];
v1_1 = 0;
v1_2 = 0;
masa_prcaida2 = 75;
coef_arrastre2 = 15;
%hallar t2 para que alcance la misma velocidad del paracaída 1

for i=
    %velocidad del paracaída 1 en 10 s
    vf1(i) = v1_1 + ((g - (coef_arrastrel/masa_prcaidal)*v1_1)*(t(i)-(t(i)-1)));
    v1_1 = vf1(i);
    fprintf('Paracaidista 1 -> Tiempo y velocidad %d %.3f \n',t(i),vf1(i));
    %velocidad del paracaída 2 en 10 s
    vf2(i) = v1_2 + ((g - (coef_arrastre2/masa_prcaida2)*v1_2)*(t(i)-(t(i)-1)));
    v1_2 = vf2(i);
    fprintf('Paracaidista 2 -> Tiempo y velocidad %d %.3f \n',t(i),vf2(i));
if(t_1==t(i))
    disp('Tiempo dado para el paracaidista 1');
end
end
```

1.8

```
% EJERCICIO 1.8
% ejemplo 1.2
%
c=12.5
m=68.1
t=10
tamaño de paso 1s y 0.5
%
t=[1:10];

g=9.81;
vi_1=0;
vi_2=0;
coef_arrastre = 12.5;
masa_prcaida = 68.1;
for i=t

    vi_1 = vi_1 + ((g - (coef_arrastre/masa_prcaida)*vi_1)*(t(i)-(t(i)-1)));
    disp('Tiempo de paso 1s');
    disp(vi_1);
    vi_2 = vi_2 + ((g - (coef_arrastre/masa_prcaida)*vi_2)*(t(i)-(t(i)-0.5)));
    disp('Tiempo de paso 0.5s');
    disp(vi_2);

end
```

1.10

```
% EJERCICIO 1.10
m = 80;
c = 10;
T = [1:20];
g = 9.81;
vi = 20;

for i=T
    vi = vi + ((g - (c/m)*vi)*(T(i)-(T(i)-1)));
    if(10 == T(i))
        c = 50;
    end
    fprintf('Paracaidista tiempo y velocidad %d %.3f\n',T(i),vi);
end
```

1.16

```
% EJERCICIO 1.16
%
Ley de enfriamiento de Newton
dT/dt=-k(T-Ta)
Metodo de euler
dy/dx = f(x;y) x(i+1) = x(i)+h ; y(i+1)=y(i)+h(f(x;y));
%
T_café = 68;
Ta = 21;
t = [];
k = 0.017;
tt = -1;
h = 1;

for i=1:11
    t(i) = tt + 1;
    tt = t(i);
    T(i) = T_café + h*(-k*(T_café-Ta));
    T_café = T(i);
    fprintf('Iteracion, Tiempo(min), Temperatura(°C) %d %d %.2f\n',i,t(i),T(i));

    plot(t,T);
    xlabel('tiempo (min)');
    ylabel('Temperatura (°C)');
end
```

1.18

```
%% EJERCICIO 1.18

Q = [];
f = [1:10];
Q(2) = 0.6;
Q(3) = 0.4;
Q(7) = 0.2;
Q(8) = 0.3;
% PIDEN HALLAR Q1 Q4 Q5 Q6 Q9 Q10

Q(1) = Q(2) + Q(3);
Q(6) = Q(8) - Q(7);
Q(5) = Q(6) + Q(7);
Q(4) = Q(3) - Q(5);
Q(9) = Q(4) + Q(8);
Q(10) = Q(9) + Q(2);

plot(f,Q);
xlabel('cant. flujo');
ylabel('Flujo');
```

Capítulo II

Programación y software

2.1.

```
IF x>=10 THEN
    DO
        x = x - 5
        IF x<50 EXIT
    ENDDO
ELSE
    IF x<5 THEN
        x = 5
    ELSE
        x = 7.5
ENDIF
```

2.2

```
DO
    i = i + 1
    IF z > 50 EXIT
    x = x + 5
    IF x > 5 THEN
        y = x
    ELSE
        y = 0
    ENDIF
    z = x + y
ENDDO
```

2.3

```
# algoritmo hecho con python
def algoritmoCartas():
    import random;
    tarjetas = [];

    # 1. Ingresar los valores para la concentración de un contaminante
    for i in range(0, 52):
        valueTarjeta = random.randrange(0, 100);
        tarjetas.append(valueTarjeta);

    # 2. Identificamos los datos que necesitamos calcular
    suma = 0;
    promedio = 0;
    max = 0;

    for i in tarjetas:
        # 3. Sumamos todos los valores que tenemos
        suma = suma + i;
        # 4. Calcular el valor máximo de todos los valores
        if(i>max):
            max = i;
    # 5. Calculamos el promedio dividiendo la suma entre la cantidad de valores
    promedio = suma / 52;
    # 6. Mostrar los valores obtenidos
    print(f"Suma {suma} -- Promedio {promedio} -- valor máximo {max}");
algoritmoCartas();
```

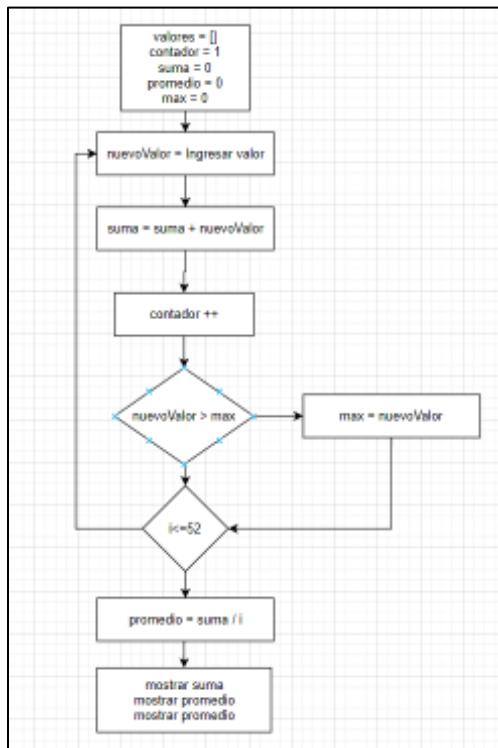
2.3 En cada una de las tarjetas de un conjunto de cartas índice, se registra un valor para la concentración de un contaminante en un lago. Al final del conjunto, se coloca una carta marcada como "fin de los datos". Escriba un algoritmo para determinar la suma, el promedio y el máximo de dichos valores.



suma = a1+a2+...+a52
promedio = suma/52
max = a1
if valor > max
 max = valor

para calcular el max

2.4



2.5

```

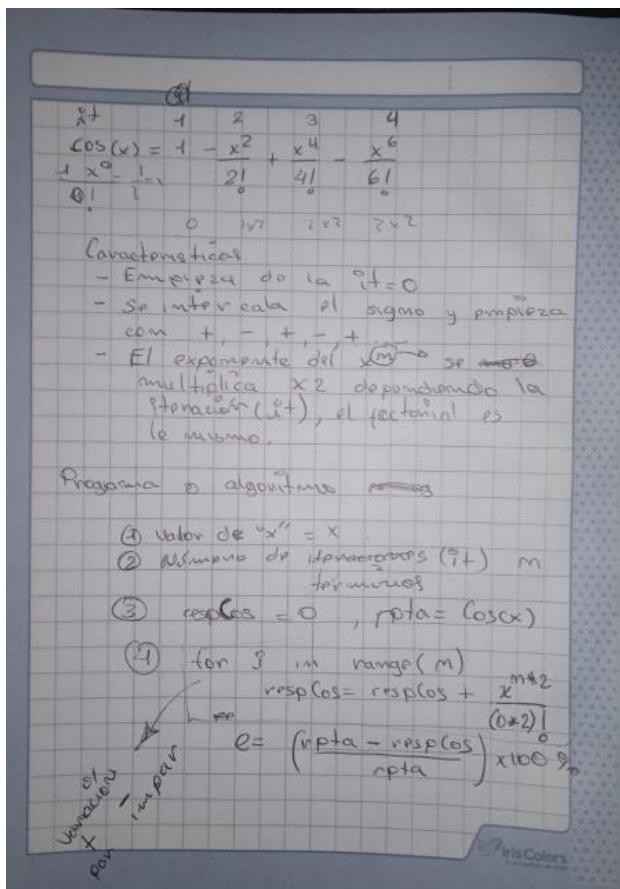
# ejercicio para calcular las raices de una ecuación cuadrática
def ejercicioCuadratica():
    import math;
    import cmath;
    # 1. Solicitar los valores al usuario
    a = float(input("Ingrese el valor de 'a': "));
    b = float(input("Ingrese el valor de 'b': "));
    c = float(input("Ingrese el valor de 'c': "));

    def calcularDiscriminante(a,b,c):
        return math.pow(b,2) - 4*a*c;

    discriminante = calcularDiscriminante(a,b,c);

    def calcularRaices(a,b,discriminante):
        if discriminante > 0:
            return print(f"x1 = {(-b+math.sqrt(discriminante))/(2*a)} ; x2 = {(-b-math.sqrt(discriminante))/(2*a)}");
        if discriminante == 0:
            return print(f"x(1,2) = {-b/(2*a)}");
        if discriminante < 0:
            return print(f"x1 = {(-b+cmath.sqrt(discriminante))/(2*a)} ; x2 = {(-b-cmath.sqrt(discriminante))/(2*a)}")
    try:
        calcularRaices(a,b,discriminante);
    except:
        print("Existe división entre cero");
ejercicioCuadratica();
    
```

2.6



Paso 1: Guardar número del coseno ('x') y número de iteraciones ('n')

Paso 2: Definir la variable 'respCos' y el contador 'contador'

Paso 3: verificamos si el 'contador' es par o impar

Paso 4: Si el 'contador' es par entonces, respCos va ser igual a 'respCos' más $(x^{(\text{contador}*2)}) / (\text{contador}*2)$!

Paso 5: Si el 'contador' es impar entonces, respCos va ser igual a 'respCos' menos $(x^{(\text{contador}*2)}) / (\text{contador}*2)$!

Paso 6: El error en cada iteración es $(\cos(x) - \text{respCos}) / \cos(x)$

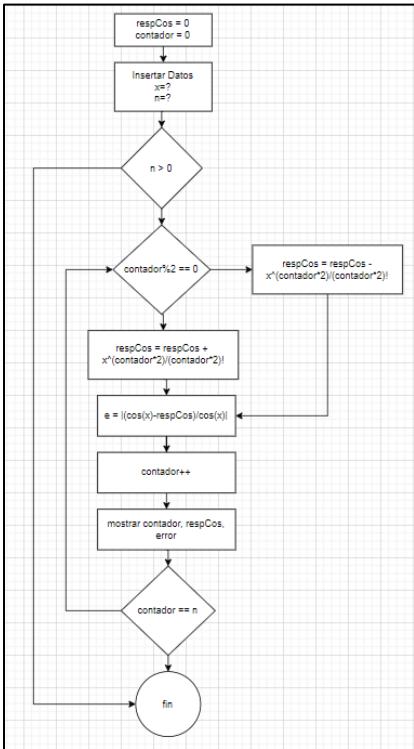
Paso 7: Mostrar los resultados (contador -- respCos -- error)

Paso 8: Aumentamos el 'contador' en 1

Paso 9: Si contador no es igual a 'n' entonces, volvemos al paso 3°

Paso 10: Fin del programa

2.7



```

READ X
READ N

RESPCOS = 0
CONTADOR = 0
E = 0
DO WHILE(CONTADOR != N)
    IF CONTADOR % 2 == 0 THEN
        RESPCOS = RESPCOS + X^(CONTADOR * 2)/(CONTADOR * 2)!
    ELSE
        RESPCOS = RESPCOS - X^(CONTADOR * 2)/(CONTADOR * 2)!
    E = |(COS(X)-RESPCOS)/COS(X)|
    WRITE (CONTADOR -- RESPCOS -- E)
    CONTADOR ++
END DO

```

2.8

```

def calcularSerieCoseno():
    import math;
    from decimal import Decimal;

    # 1. Solicitar datos como el valor de x y el número de términos
    x = float(input("Cos(x) -> ingrese el valor de 'x': "));
    n = int(input("Ingrese el número de términos: "));

    # 2. Calculamos el oficial del cos(x)
    rpta = math.cos(x);
    print(f"El valor de cos({x}) = {round(Decimal(rpta), 6)}");

    # 3. Calculamos los valores de las iteraciones
    respCos = 0;
    print("it          itCos          error");
    for i in range(n):
        if(i%2 == 0):
            respCos = respCos + math.pow(x,i*2)/math.factorial(i*2);
        else:
            respCos = respCos - (math.pow(x,i*2))/(math.factorial(i*2));

    # 4. Calculamos el error
    e = abs((rpta - respCos)/rpta);

    # 5. imprimir los valores resueltos
    print(f"{i}          {round(Decimal(respCos),6)}          {round(Decimal(e),6)*100} %");
calcularSerieCoseno();

```

2.9

```
# pseudocódigo hecho en pseint
function value<-solicitarNotas(cantNotas)
    suma<-0;
    Para i<-0 Hasta cantNotas Con Paso 1 Hacer
        Escribir "Digite la nota ",(i+1);
        Leer nota;
        suma<-suma + nota;
    FinPara
    value<-suma / cantNotas;
FinFunction

Proceso Principal
    Escribir "Nombre del curso: ";
    Leer nombreCurso;
    Escribir "Clave de curso: ";
    Leer ClaveCurso;

    Escribir "Peso de los cuestionarios: ";
    Leer c;
    Escribir "Peso de las tareas: ";
    Leer t;

    pc<-0;
    pt<-0;
    nFinal<-0;
    promedioFinal<-0;
    cantNotas<-0;

    Escribir "Cant. de notas de los cuestionarios: ";
    Leer cantNotas;
    Leer pc<-solicitarNotas(cantNotas);
    Escribir "Cant. de notas de las tareas: ";
    Leer cantNotas;
    Leer pt<-solicitarNotas(cantNotas);

    Escribir "El curso tiene una nota final? si(1) - no(2): ";
    Leer pregunta;

    Si pregunta == 1 Entonces
        Escribir "Peso de la nota final: ";
        Leer ef;
        Escribir "Nota final: ";
        Leer nFinal;
        promedioFinal<-((c*pc + t*pt + ef*nFinal)/(c + t + ef))*100;
    SiNo
        promedioFinal<-((c*pc + t*pt)/(c + t))*100;
    Fin Si
    Escribir "Clave: ",claveCurso,"Curso: ",nombreCurso,"Promedio Final: ",promedioFinal;
FinProceso
```

```

def ejercicioCalificaciones():
    def solicitarNotas(cantNotas):
        suma = 0;
        for i in range(cantNotas):
            nota = int(input(f"Nota {i+1}: "));
            suma = suma + nota;
        return suma / cantNotas;

    nombreCurso = input("Ingrese el nombre del curso: ");
    claveCurso = input("Ingrese la clave del curso: ");
    c = int(input("Peso de la nota de los cuestionarios: "));
    t = int(input("Peso de la nota de las tareas: "));

    pc = 0;
    pt = 0;
    nFinal = 0;
    promedioFinal = 0;

    cantNotas = int(input("Cant. de notas de los cuestionarios: "));
    pc = solicitarNotas(cantNotas);

    cantNotas = int(input("Cant. de notas de la tarea: "));
    pt = solicitarNotas(cantNotas);

    pregunta = int(input("El curso tiene una nota final? si(1) - no(2): "));
    if pregunta == 1:
        ef = int(input("Peso de la nota final: "));
        nFinal = int(input("Ingrese la nota final del curso: "));
        promedioFinal = ((c*pc + t*pt + ef*nFinal)/(c + t + ef))*100;
    else:
        promedioFinal = ((c*pc + t*pt)/(c + t))*100;
    print(f"Clave: {claveCurso} -- Curso: {nombreCurso} -- Calificación Promedio: {promedioFinal}%");

```

2.10

```

READ A
TOL = 0
IF A>0 THEN
    DO WHILE(TRUE)
        Y = (X+(A/X))/2;
        E = |(Y-X)/Y|;
        X = Y;
        IF E<TOL THEN
            BREAK
        WRITE ("LA RAIZ APROXIMA ES: ",X)
    END DO
ELSE
    WRITE ("LA RAIZ CUADRADA ES 0");

```

```

def aproximacionRaiz():
    from decimal import Decimal
    a = float(input("Ingrese el valor de la raiz: "));
    tol = 0.00001;
    if a > 0:
        x = a/2;
        while True:
            y = (x+(a/x))/2;
            e = round(Decimal(abs((y-x)/y)),6);
            x = y;
            print(f"Aproximación de sqrt({a}) = {x} con un e = {e}");
            if(e<tol):
                break;
        else:
            print("Raiz cuadrada = 0");
    aproximacionRaiz();

```

2.11

```
# ejercicio 2.11 calcular el valor futuro
def calcularValorFuturo():
    p = float(input("Ingrese la inversión inicial: "));
    tasa = float(input("Ingrese la tasa de interés (decimal): "));
    n = int(input("Número de años: "));

    print(f"Periodo          Valor futuro");
    for i in range(n):
        f = p*math.pow((1+tasa),i+1);
        print(f"{i+1}           {round(Decimal(f),4)} $");
calcularValorFuturo();
```

2.12

```
def calcularPagoAnual():
    from decimal import Decimal;
    p = float(input("Ingrese el préstamo realizado: "));
    ti = float(input("Ingrese la tasa de interés (decimal): "));
    n = int(input("Ingrese la cantidad de pagos: "));
    a = (p*ti*math.pow((1+ti), n))/(math.pow((1+ti),n)-1);
    print("n      capital      interes      amotiz      cuotaAnual");
    interesTotal = 0;
    amortizacionTotal = 0;
    for i in range(n):
        itc = p * ti;
        interesTotal = interesTotal + itc;
        amortizacion = a - itc;
        amortizacionTotal = amortizacionTotal + amortizacion;
        print(f"{i+1}  {round(Decimal(p),4)}  {round(Decimal(itc),4)}  {round(Decimal(amortizacion),4)}  {round(Decimal(a),4)}");
        p = p - amortizacion;
    print("Final-> {round(Decimal(p),4)}  {round(Decimal(interesTotal),4)}  {round(Decimal(amortizacionTotal),4)}  {round(Decimal(a*n),4)}");
calcularPagoAnual();
```

2.14

```
def paracaidas():
    ca = float(input("Ingrese el coeficiente de rozamiento: "));
    m = float(input("Ingrese la masa del paracaídas: "));

    #número de iteraciones
    contador = int(input("Cuántos recorridos desea?: "))

    #velocidad inicial
    v = 0;

    #tamaño de pasos
    pasos = 1;

    #gravedad
    g = 9.8;
    t1 = 0;
    t2 = 0;

    print("t(s)  v(m/s)");
    for i in range(contador+1):
        v = v+(g-(ca*v)/(m))*(t2-t1);

        # imprimir valores
        print(f"{t2}  {round(Decimal(v),5)}");
        t1 = t2;
        t2 = t2 + pasos;
```

2.15

```
# 2.15 ejercicio de la burbuja

def metodoBurbuja():
    import random;
    numeros = [];

    # insertar números aleatorios
    for i in range(20):
        numeros.append(random.randint(0, 100));

    aux = 0;

    # método totalmente ineficiente del ordenamiento de la burbuja
    for i in range(len(numeros)):
        for j in range(len(numeros)-1):
            if numeros[j] > numeros[j+1]:
                aux = numeros[j];
                numeros[j] = numeros[j+1];
                numeros[j+1] = aux;
```

2.17

```
# ejercicio 2.17 calcular polares
def calcularPolares():
    # 1. Saber la coordenada

    while(True):
        x = float(input("Ingrese el valor de x: "));
        y = float(input("Ingrese el valor y: "));

        # 2. radio (r)
        r = math.sqrt(math.pow(x,2) + math.pow(y,2));

        angulo = 0;

        if x<0 and y>0:
            angulo = math.degrees(math.atan(y/x)) + 180;
        if x<0 and y<0:
            angulo = math.degrees(math.atan(y/x)) - 180;
        if x<0 and y==0:
            angulo = 180;
        if x==0 and y>0:
            angulo = 90;
        if x==0 and y<0:
            angulo = -90;
        if x==0 and y==0:
            angulo = 0;
        if x>0:
            angulo = math.degrees(math.atan(y/x));

        if angulo < 0:
            angulo = angulo + 360;
        print(f"r = {r} ; angulo={angulo}")
```

2.18

```
# ejercicio 2.18 condicionales

def procedimientosCondicionales():
    num = int(input("Ingrese un número: "));

    if 90<=num<=100:
        print("A");
    if 80<=num<90:
        print("B");
    if 70<=num<80:
        print("C");
    if 60<=num<70:
        print("D");
    if num<60:
        print("F");
    if num>100:
        print("No tiene clasificación numérica.");
```

Capítulo III

Aproximaciones y errores de redondeo

3.2

```
base= input("ingrese base ");
bits= input("ingrese bits ");
ep = 1;
for i=1:(bits-1)

    ep = ep * (1 / base) ;

end

disp(ep)
```

3.3

```
function xmin = getmin
x = 1;

while(1)
    if x <= 0, break, end

    xmin = x;
    x = x / 2;

end
```

3.4

```

vd = double(pi ^ 4 / 90);
sum = 0;
n = 10000;

for i=1:n
sum = sum + 1 / i ^ 4;

end

vr=100* abs((vd-sum)/vd);
fprintf('suma\n');
disp(sum)
fprintf('muestra el error relativo porcentual verdadero\n');
disp(vr)

```

3.5

3.5			
A	$f(x)=e^{-5}$		
TERMINOS	RESULTADO	E _t	E _{a%}
0	1	-147.4131591	0
1	6	-889.4789545	83.33333333
2	18.5000000	-2744.6434430	67.56756757
3	39.3333333	-5836.5842572	52.96610169
4	65.3750000	-9701.5102750	39.83428936
5	91.4166667	-13566.4362928	28.48678213
6	113.1180556	-16787.2079743	19.18472589
7	128.6190476	-19087.7591753	12.05186351
8	138.3071677	-20525.6036760	7.004785221
9	143.6894566	-21324.4061763	3.745778597
10	146.3806010	-21723.8074265	1.838457034
11	147.6038485	-21905.3534493	0.828736847
12	148.1135350	-21980.9976255	0.344118753
13	148.3095682	-22010.0915394	0.132178424
14	148.3795801	-22020.4822230	0.047184306
15	148.4029174	-22023.9457841	0.015725629
16	148.4102103	-22025.0281470	0.004914017
17	148.4123552	-22025.3464890	0.001445278
18	148.4129511	-22025.4349174	0.000401465
19	148.4131079	-22025.4581880	0.000105648
20	148.4131471	-22025.4640056	2.64121E-05

B			
TERMINOS	R.EVALUADO	E _t	E _{a%}
0	1	-147.4131591	0
1	0.1666667	-23.7355265	-500
2	0.0540541	-7.0223329	-208.3333333
3	0.0254237	-2.7732159	-112.6126126
4	0.0152964	-1.2701822	-66.20762712
5	0.0109389	-0.6234803	-39.83428936
6	0.0088403	-0.3120201	-23.73898511
7	0.0077749	-0.1538972	-13.70337563
8	0.0072303	-0.0730692	-7.532414692
9	0.0069595	-0.0328744	-3.891547345
10	0.0068315	-0.0138854	-1.872889299
11	0.0067749	-0.0054830	-0.835662288
12	0.0067516	-0.0020229	-0.345307019
13	0.0067427	-0.0006985	-0.132353367
14	0.0067395	-0.0002263	-0.04720658
15	0.0067384	-0.0000690	-0.015728102
16	0.0067381	-0.0000199	-0.004914259
17	0.0067380	-0.0000054	-0.001445299
18	0.0067380	-0.0000014	-0.000401466
19	0.0067379	-0.0000003	-0.000105649
20	0.0067379	-0.0000001	-2.64121E-05

3.6

3.6			
F(X)	$\frac{6x}{(1 - 3x^2)^2}$		0.577
RPTA:	2352910.793		

3.7

3.7		
X=	1.37	
Y=	0.043053	
EA% =	7.09125961	
X=	1.37	
Y=	0.743053	
EA% =	0.410872441	

3.8

3.8	
	$\cos x = 1 - \frac{x^2}{2} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \dots$
X= 0.3*PI	
VALORES	
0	1
1	0.444132198
2	0.032875568
3	0.000973407
4	1.544E-05
COS(0.3*PI)	1.477996613

3.9

```
x=0.3*pi;
es=0.5e-08;
cosi=1;
j=1;
fprintf("j=%2.0f cos(x)=%0.10f\n",j,cosi)
fact = 1;
while(1)
    j=j+1;
    i=2*j-2;
    fact= fact*i*(i-1);
    cosn=cosi+((-1)^(j+1))*((x)^i)/fact;
    ea = abs((cosn-cosi)/cosn);
    fprintf("j=%2.0f cos(x)=%0.10f ea=%0.1e\n",j,cosn,ea);
    if ea<es,break,end
    cosi=cosn;
end
```

3.10

3.10 $x^2 - 5000.002x + 10$

$$x = \frac{5000.002 \pm \sqrt{(5000.002)^2 - (4)(1)(10)}}{2}$$

X1=	0.0020
X2=	5000.0

CALCULAMOS CON 5 DIGITOS

$$x = \frac{5000.0 \pm \sqrt{25000000 - (4)(1)(10)}}{2}$$

X1=	0.05
X2=	4999.95

ER% DE X1 0.0020 %
 ER% DE X2 2400.0 %

CALCULAMOS NUEVAMENTE LAS RAICES

$$\frac{-2c}{b \pm \sqrt{b^2 - 4ac}} = \frac{-2(10)}{-5000.0 \pm \sqrt{25000000 - (4)(1)(10)}}$$

X1=	200
X2=	0.00200002

ER% DE X2 96
 ER% DE X1 0.001

3.11

```
a = 1;
b = -5000.002;
c = 10;
% este bloque se trabajo con la primera ecuacion (3.12)

x1 = (-b + sqrt(b ^ 2 - 4 * a * c)) / (2 * a);
x2 = (-b - sqrt(b ^ 2 - 4 * a * c)) / (2 * a);
% este bloque se trabajo con la segunda ecuacion (3.13)
x3 = (-2 * c) / (b + sqrt(b ^ 2 - 4 * a * c));
x4 = (-2 * c) / (b - sqrt(b ^ 2 - 4 * a * c));

er1 = ((x1 - x3) / x1) * 100
er2 = ((x2 - x4) / x2) * 100
%los errores relativos porcentuales entre (x1-x3) y (x2-x4)
fprintf('er1 = \n',er1);
fprintf('er2 = \n',er2);
```

Capítulo IV

Errores de truncamiento y la serie de Taylor

1.1

```
from cmath import exp
import math
import numpy

from sympy import ln
#4.1 La serie infinita
#e^x=1+x+(x^2)/2!+(x^3)/3!+...+(x^n)/n!
#Se utiliza para aproximar e^x

print("a) Muestre que la expansión en serie de Maclaurin es un caso especial de la expansión en la serie de Taylor con xi= 0 y h = x.")
print("")
x = 0
print("Primera iteracion")
e_i_1 = x**0/math.factorial(0)
print(e_i_1)
print("Segunda iteracion")
e_i_2 = x**0/math.factorial(0) + x**1/math.factorial(1)
print(e_i_2)
print("Tercera iteracion")
e_i_3 = x**0/math.factorial(0) + x**1/math.factorial(1) + x**2/math.factorial(2)
print(e_i_3)
print("Cuarta iteracion")
e_i_4 = x**0/math.factorial(0) + x**1/math.factorial(1) + x**2/math.factorial(2) + x**3/math.factorial(3)
print(e_i_4)
print("Quinta iteracion")
e_i_5 = x**0/math.factorial(0) + x**1/math.factorial(1) + x**2/math.factorial(2) + x**3/math.factorial(3) + x**4/math.factorial(4)
print(e_i_5)
print("")
print("Siempre resulta 1 eso lo hace un caso particular y se muestra una proximidad a la función e^x")
print("")
print("b) Use la serie de Taylor para estimar f(x) = e^-x en xi+1 = 1 para xi = 0.25.")
print("Emplee versiones de cero, primero, segundo y tercer orden, y calcule |et| para cada caso.")
print("")
print("Valor verdadero f(x)=e^-x en xi+1=1 para xi=0.25")
valor_v=exp(-1)
h=1-0.25
print("El valor verdadero es = ",numpy.round(valor_v,5))

print("Orden cero")
x=-0.25
e_0 = exp(x)/math.factorial(0)
print("El valor en el orden cero es = ",numpy.round(e_0,5))
print("El error |et| porcentual es = %",numpy.round(abs((valor_v-e_0)*100/valor_v),5))

print("Primer orden")
suma=e_0 #acumulamos los valores
x=-e_0
e_1 = suma + x*(h**1)/math.factorial(1)
print("El valor en el primer orden es = ",numpy.round(e_1,5))
print("El error |et| porcentual es = %",numpy.round(abs((valor_v-e_1)*100/valor_v),5))

print("Segundo orden")
suma=e_1 #acumulamos los valores
x=-e_0
e_2 = suma + x*(h**2)/math.factorial(2)
print("El valor en el segundo orden es = ",numpy.round(e_2,5))
print("El error |et| porcentual es = %",numpy.round(abs((valor_v-e_2)*100/valor_v),5))

print("Tercer orden")
suma=e_2 #acumulamos los valores
x=-e_0
e_3 = suma + x*(h**3)/math.factorial(3)
print("El valor en el tercer orden es = ",numpy.round(e_3,5))
print("El error |et| porcentual es = %",numpy.round(abs((valor_v-e_3)*100/valor_v),5))
```

1.2

```

from cmath import exp, pi
import math
import numpy

from sympy import ln
#4.2 La expansión en serie de Maclaurin para cos x es
#cos(x)=1-(x^2)/2!+(x^4)/4!-(x^6)/6!+(x^8)/8!-...
#Se utiliza para aproximar e^x
print("Iniciando con el primer término cos x = 1, agregue los términos uno a uno para estimar cos (pi/4).")
print("Después de que agregue cada uno de los términos, calcule los errores relativos porcentuales exactos y aproximados. ")
print("")
print("Valor verdadero f(x)=cos(pi/4) y como primer término se tiene cos(x) = 1")
xi=pi/4
valor_v=math.cos(xi) #valor verdadero
print("El valor verdadero es = ",numpy.round(valor_v,5))

print("Orden cero")
suma=1 #El primer valor es 1
e_0 = suma
print("El valor de cos(x) en el orden cero es = ",numpy.round(e_0,5))
print("El error |et| porcentual es = %",numpy.round(abs((valor_v-e_0)*100/valor_v),5))

print("Primer orden")
suma=e_0 #acumulamos los valores
e_1 = suma - (xi**2)/math.factorial(2)
print("El valor de cos(x) en el primer orden es = ",numpy.round(e_1,5))
print("El error |et| porcentual es = %",numpy.round(abs((valor_v-e_1)*100/valor_v),5))

print("Segundo orden")
suma=e_1 #acumulamos los valores
e_2 = suma + (xi**4)/math.factorial(4)
print("El valor de cos(x) en el segundo orden es = ",numpy.round(e_2,5))
print("El error |et| porcentual es = %",numpy.round(abs((valor_v-e_2)*100/valor_v),5))

```

1.3

```

from cmath import exp, pi
import math
import numpy

from sympy import ln
#4.3 Repita los cálculos del problema 4.2, pero ahora usando la expansión de la serie de Maclaurin para sen(x)
#sen(x)=x-(x^3)/3!+(x^5)/5!-(x^7)/7!+...
#para evaluar el sen (pi/4).
print("Aplicando lo del ejercicio anterior pero para el sen(pi/4).")
print("")
print("Valor verdadero f(x)=sen(pi/4)")
xi=pi/4
valor_v=math.sin(xi) #valor verdadero
print("El valor verdadero es = ",numpy.round(valor_v,5))

print("Orden cero")
e_0 = xi
print("El valor de sen(x) en el orden cero es = ",numpy.round(e_0,5))
print("El error |et| porcentual es = %",numpy.round(abs((valor_v-e_0)*100/valor_v),5))

print("Primer orden")
suma=e_0 #acumulamos los valores
e_1 = suma - (xi**3)/math.factorial(3)
print("El valor de sen(x) en el primer orden es = ",numpy.round(e_1,5))
print("El error |et| porcentual es = %",numpy.round(abs((valor_v-e_1)*100/valor_v),5))

print("Segundo orden")
suma=e_1 #acumulamos los valores
e_2 = suma + (xi**5)/math.factorial(5)
print("El valor de sen(x) en el segundo orden es = ",numpy.round(e_2,5))
print("El error |et| porcentual es = %",numpy.round(abs((valor_v-e_2)*100/valor_v),5))

```

1.4

```

from cmath import exp, pi
import math
import numpy
from sympy import *

from sympy import ln
#4.4 Emplee la expansión de la serie de Taylor de cero hasta tercer orden para predecir f(2) si
#f(x)=25x^3-6x^2+7x-88
#usando como punto base x = 1. Calcule el error relativo porcentual verdadero et para cada aproximación.
print("")
print("Se tiene que f(x)=25x^3-6x^2+7x-88 usar el teorema de taylor para predecir f(2)")
print("y calcule el error porcentual")
print("")
print("Valor verdadero f(2)=25(2)^3-6(2)^2+7(2)-88 y se tiene que xi+1=2 para xi=1")
h=2-1
x=2
valor_v=25*x**3-6*x**2+7*x-88 #valor verdadero
print("El valor verdadero es = ",valor_v)

# Funcion y derivadas
xi=1
fun=25*x**3-6*x**2+7*x-88
# 1 derivada
d1=75*x**2-12*x**1+7
# 2 derivada
d2=150*x**1-12
# 3 derivada
d3=150

print("Orden cero")
e_0 = fun
print("El valor de f(x) en el orden cero es = ",numpy.round(e_0,5))
print("El error |et| porcentual es = %",numpy.round(abs((valor_v-e_0)*100/valor_v),5))

print("Primer orden")
suma=e_0 #acumulamos los valores
e_1 = suma + d1*(h**1)/math.factorial(1)
print("El valor de f(x) en el primer orden es = ",numpy.round(e_1,5))
print("El error |et| porcentual es = %",numpy.round(abs((valor_v-e_1)*100/valor_v),5))

print("Segundo orden")
suma=e_1 #acumulamos los valores
e_2 = suma + d2*(h**2)/math.factorial(2)
print("El valor de f(x) en el segundo orden es = ",numpy.round(e_2,5))
print("El error |et| porcentual es = %",numpy.round(abs((valor_v-e_2)*100/valor_v),5))

print("Tercer orden")
suma=e_2 #acumulamos los valores
e_3 = suma + d3*(h**3)/math.factorial(3)
print("El valor de f(x) en el tercer orden es = ",numpy.round(e_3,5))
print("El error |et| porcentual es = %",numpy.round(abs((valor_v-e_3)*100/valor_v),5))
print("")
print("Como el error nos dio 0 llegamos al valor exacto")

```

1.5

```

from cmath import exp, pi
import math
import numpy
from math import log

from sympy import ln
#4.5 Use la expansión de la serie de Taylor de cero al cuarto orden para estimar f(3) si f(x) = ln x utilizando x = 1 como punto base.
# Calcule el error relativo porcentual et para cada aproximación.
# Analice los resultados.
print("")
print("Se tiene que f(x)=ln(x) usar el teorema de taylor para predecir f(3)")
print("y calcule el error porcentual")
print("")
print("Valor verdadero f(3)=ln(3) y se tiene que xi+1=3 para xi=1")
h=3-1
x=3
valor_v=log(x,numpy.exp(1)) #valor verdadero
print("El valor verdadero es = ",valor_v)

# Funcion y derivadas
xi=1
fun=log(xi,numpy.exp(1))
# 1 derivada
d1=1/xi
# 2 derivada
d2=-1/(xi**2)
# 3 derivada
d3=2/(xi**3)
# 4 derivada
d4=-6/(xi**4)

print("Orden cero")
e_0 = fun
print("El valor de Ln(x) en el orden cero es = ",e_0)
print("El error |et| porcentual es = %",numpy.round(abs((valor_v-e_0)*100/valor_v),5))

print("Primer orden")
suma=e_0 #acumulamos los valores
e_1 = suma + d1*(h**1)/math.factorial(1)
print("El valor de Ln(x) en el primer orden es = ",numpy.round(e_1,5))
print("El error |et| porcentual es = %",numpy.round(abs((valor_v-e_1)*100/valor_v),5))

print("Segundo orden")
suma=e_1 #acumulamos los valores
e_2 = suma + d2*(h**2)/math.factorial(2)
print("El valor de Ln(x) en el segundo orden es = ",numpy.round(e_2,5))
print("El error |et| porcentual es = %",numpy.round(abs((valor_v-e_2)*100/valor_v),5))

print("Tercer orden")
suma=e_2 #acumulamos los valores
e_3 = suma + d3*(h**3)/math.factorial(3)
print("El valor de Ln(x) en el tercer orden es = ",numpy.round(e_3,5))
print("El error |et| porcentual es = %",numpy.round(abs((valor_v-e_3)*100/valor_v),5))

print("Cuarto orden")
suma=e_3 #acumulamos los valores
e_4 = suma + d4*(h**4)/math.factorial(4)
print("El valor de Ln(x) en el cuarto orden es = ",numpy.round(e_4,5))
print("El error |et| porcentual es = %",numpy.round(abs((valor_v-e_4)*100/valor_v),5))
print("")
print("en este caso el error va aumentasndo porque nos alejamos del valor verdadero")

```

1.6

```

from cmath import exp, pi
import math
import numpy
from sympy import *

from sympy import ln
#4.6 Utilice aproximaciones en diferencias de 0(h) hacia atrás y hacia adelante y una aproximación de diferencia central de 0(h2)
#para estimar la primera derivada de la función mencionada en el problema 4.4. Evalúe la derivada en x = 2 usando un tamaño del
#incremento 0.2. Compare los resultados con el valor exacto de las derivadas. Interprete los resultados considerando el término
#residual de la expansión en la serie de Taylor.

#f(x)=25x^3-6x^2+7x-88
#usando como punto base x = 2, incremento=0.2
print("")
print("Se tiene que f(x)=25x^3-6x^2+7x-88")
print("")
print("Valor verdadero x=2 en f'(2)=75(2)^2-12(2)^1+7")

x=2
incremento=0.2
valor_v=75*x**2-12*x+7 #valor verdadero de la derivada
print("El valor verdadero es =",valor_v)

#valores a partir del incremento
ximenos=x-incremento
xicentro=x
ximas=x+incremento

# Funcion y derivadas
fundec=25*ximenos**3-6*ximenos**2+7*ximenos-88
funcen=25*xicentro**3-6*xicentro**2+7*xicentro-88
funinc=25*ximas**3-6*ximas**2+7*ximas-88

print("Hacia adelante")
a = (funinc-funcen)/incremento
print("El valor de f'(x) hacia adelante es =",numpy.round(a,5))
print("El error |εt| porcentual es %",numpy.round(abs((valor_v-a)*100/valor_v),5))

print("Hacia atras")
b = (funcen-fundec)/incremento
print("El valor de de f'(x) hacia atras es =",numpy.round(b,5))
print("El error |εt| porcentual es %",numpy.round(abs((valor_v-b)*100/valor_v),5))

print("Central")
c= (funinc-fundec)/(2*incremento)
print("El valor de de f'(x) en el centro es =",numpy.round(c,5))
print("El error |εt| porcentual es %",numpy.round(abs((valor_v-c)*100/valor_v),5))

```

1.7

```

from cmath import exp, pi
import math
import numpy
from sympy import *

from sympy import ln
#M4.7 Con la aproximación en diferencias centrales de O(h^2) estime la segunda derivada de la función examinada en el problema 4.4.
# Realiza la evaluación para x = 2 usando un tamaño de incremento 0.25 y 0.125 y compara lo estimado con el valor exacto de
# PROBLEMAS la segunda derivada. Interprete sus resultados considerando el término residual de la expansión en la serie de Taylor.

#f(x)=25x^3-6x^2+7x-88
#usando como punto base x = 2, incremento 0.25 y 0.125
print("")
print("Se tiene que f(x)=25x^3-6x^2+7x-88")
print("")
print("Valor verdadero x=2 en f''(2)=150x-12")

x=2
valor_v=150*x-12 #valor verdadero de la derivada
print("El valor verdadero es = ",valor_v)

#Tanto la función como el valor central serán el mismo
xicentro=x
funcen=25*xicentro**3-6*xicentro**2+7*xicentro-88

#valores a partir del incremento 0.25
inc1=0.25
ximenos=x-inc1
ximas=x+inc1
# Función y derivadas para el inc 0.25
fundec=25*ximenos**3-6*ximenos**2+7*ximenos-88
funinc=25*ximas**3-6*ximas**2+7*ximas-88

#valores a partir del incremento 0.125
inc2=0.125
ximenos=x-inc2
ximas=x+inc2
# Función y derivadas para el inc 0.125
fundatras=25*ximenos**3-6*ximenos**2+7*ximenos-88
funadelante=25*ximas**3-6*ximas**2+7*ximas-88

print("Para el incremento 0.25 los valores en las funciones son ")
print("atras = ",fundec," centro = ",funcen," adelante = ",funinc)
a = (funinc-2*funcen+fundec)/(inc1**2)
print("El valor de f''(x) para el incremento 0.25 = ",numpy.round(a,5))
print("El error [et] porcentual es = %",numpy.round(abs((valor_v-a)*100/valor_v),5))

print("Para el incremento 0.125 los valores en las funciones son ")
print("atras = ",fundatras," centro = ",funcen," adelante = ",funadelante)
b = (funadelante-2*funcen+fundatras)/(inc2**2)
print("El valor de f''(x) para el incremento 0.125 = ",numpy.round(b,5))
print("El error [et] porcentual es = %",numpy.round(abs((valor_v-b)*100/valor_v),5))

print("Ambos incrementos llegan al valor exacto ya que su error es 0%")

```

1.8

```

from xml.etree.ElementTree import C14NWriterTarget
import sympy as sp
import numpy as np
#4.8 Recuerde que la velocidad de caída del paracaidista puede
# calcularse con [ecuación (1.10)]
# Use un análisis de error de primer orden para estimar el error de
# v para t = 6, si g = 9.8 y m = 50, pero c = 12.5 ± 2
print("")
print("Dicha ecuacion es V(t)=(gm)(1-e^(-ct/m))/c")
print("")
print("Sacamos la derivada en funcion de c y remplazamos datos")
g=9.8
m=50
t=6
e=np.exp(1)
c=sp.symbols('c') #define la variable simbolica c
f=(g*m)*(1-e**(-c*t/m))/c
dfk=sp.diff(f,c)
print("La derivada con respecto a c es ",dfk)
c=12.5
derivada= -(490.0 - 490.0/2.71828182845905**(3*c/25))/c**2 + 58.8/(2.71828182845905**(3*c/25)*c)
print("y con el c=12.5 el valor de la funcion V'(t)= ",derivada)
Δv=np.abs(derivada)**2 #La variacion ± 2
print("Sacamos Δv(c) = ",Δv)
fun=(g*m)*(1-e**(-c*t/m))/c
print("c=12.5 en la funcion original la funcion V(t)= ",fun)
print("")
print("Entonces nos queda que V = ",fun," ± ",Δv)

```

1.9

```

from xml.etree.ElementTree import C14NWriterTarget
import sympy as sp
import numpy as np
#4.9 Repita el problema 4.8 con g = 9.8, t = 6, c = 12.5 ± 1.5 y m= 50 ± 2.
print("")
print("Dicha ecuacion es V(t)=(gm)(1-e^(-ct/m))/c")
print("")
print("Sacamos la derivada en funcion de m y remplazamos datos")
g=9.8
t=6
c=12.5
e=np.exp(1)
Δv = 2.7734 #La variacion Δv(c) que obtuvimos en el ejercicio anterior

m=sp.symbols('m') #define la variable simbolica m
f=(g*m)*(1-e**(-c*t/m))/c
dfk=sp.diff(f,m)
print("La derivada con respecto a m es ",dfk)
m=50
derivada=0.784 - 0.784/2.71828182845905**(75.0/m) - 58.8/(2.71828182845905**(75.0/m)*m)
print("y con el m=50 el valor de la funcion V'(t)= ",derivada)
Δm=np.abs(derivada)*1.5 #La variacion ± 1.5
print("Sacamos Δv(m) = ",Δm)
fun=(g*m)*(1-e**(-c*t/m))/c
print("m=50 en la funcion original la funcion V(t)= ",fun)
print("")
print("Entonces nos queda que V = ",fun," ± ",Δm+Δv)

```

1.10

```
from math import *
import sympy as sp
from sympy.plotting import plot

def PolTaylor(a,n):
    x=sp.symbols('x') #define la variable simbolica x
    f=sp.exp(x)
    T=f.subs(x,a)
    for k in range(1,n+1):
        dfk=sp.diff(f,x)
        T=T+dfk.subs(x,a)*((x-a)**k)/factorial(k)
        f=dfk
    print(sp.expand(T))
a=float(input('Digite alrededor de cual punto desea el polinomio '))
n=int(input('Digite el orden del polinomio de Taylor '))
PolTaylor(a,n)
```

Capítulo V

Métodos cerrados

5.1

```
import numpy as np
import matplotlib.pyplot as plt
import math

def Bisec(fun, x_a, x_b, steps=10):
    n=0
    x_ant=0
    error_a=0
    valor_v=6.405124837953327
    error_rpv=0
    for n in range(steps+1):
        x_m = (x_a + x_b) / 2
        error_a=(x_m-x_ant)*100 / x_m
        error_v=valor_v-x_m
        error_rpv=(error_v/valor_v)

        if f(x_m) == 0:
            return x_m

        if f(x_a) * f(x_m) < 0:
            x_b = x_m
        else:
            x_a = x_m
        x_ant=x_m

    return (x_a + x_b) / 2,n,error_a,error_rpv

def raicesEcuacion_2(a,b,c):
    determinante=caso=((b**2)-(4*c*a))**(1/2)
    if(caso>=0):caso="Reales"
    else:caso="Imaginarios"
    x1= (-b+determinante)/(2*a)
    x2= (-b-determinante)/(2*a)
    return x1,x2,determinate,caso

def Graficar_funcion(fx,inicio, parada,semilla=500):
    xi=np.linspace(inicio,parada,semilla)
    fi=fx(xi)
    plt.plot(xi,fi, label='f(x)')
    plt.axhline(0, color='k')
    plt.title('Grafica de Funcion')
    plt.legend()
    plt.show()

print("la ecuacion cuadratica tiene esta forma : ")
print("f(x) = ax^2 + bx + c")
a = -0.5
b = 2.50
c = 4.5
print("____")
print(f"teniendo una forma de : {a}x^2 + {b}x +{c} \n\n")
x1,x2,determinate,caso=raicesEcuacion_2(a,b,c)
print(f" - las raices de la ecuacion son ",caso, " , teniendo una determinante de : ",determinate)
print(f" - las raices son : x1 -> {x1} x2 -> {x2}")

f = lambda x: (a*x**2)+(b*x)+c
Graficar_funcion(f,-50,50)
print("____")
print(" - por metodo de biseccion :")
x_max,n,error_a,error_rpv=Bisec(f,5,10)
print(f" - para en la iteracion : {n}")
print(f" - x max : {x_max}")
print(f" - error aproximado : {error_a}")
print(f" - error rpv : {error_rpv}")
```

5.2

```
import numpy as np
import matplotlib.pyplot as plt
import math

def Bisec(fun, x_a, x_b, steps=100):
    n=0
    x_ant=0
    error_a=1
    error_s=0.1
    for n in range(steps+1):
        x_m = (x_a + x_b) / 2
        print(" - x_m: ", x_m)
        print(" - x_ant : ", x_ant)

        if (error_a<error_s):
            return x_m,n,error_a

        if(x_m!=0):
            error_a=abs((x_b-x_a)/(x_b+x_a))
        print("error actual : ",error_a)
        print("error s : ",error_s)

        if f(x_m) == 0:
            return x_m,n,error_a
        if f(x_a) * f(x_m) < 0:
            x_b = x_m
        else:
            x_a = x_m
        x_ant=x_m
    return (x_a + x_b) / 2,n,error_a

def Graficar_funcion(fx,inicio, parada,semilla=500):
    xi=np.linspace(inicio,parada,semilla)
    fi=fx(xi)
    plt.plot(xi,fi, label='f(x)')
    plt.axhline(0, color='k')
    plt.title('Grafica de Funcion')
    plt.legend()
    plt.show()

print("la ecuacion cuadratica tiene esta forma : ")
print("f(x) = ax^3 + bx^2 + cx + d")
a = 5
b = -5
c = 6
d = -2
print("____")
print(f"teniendo una forma de : {a}x^3 + {b}x^2 +{c}x + {d} \n\n")

f = lambda x: (a*x**3)+(b*x**2)+(c*x)+d
Graficar_funcion(f,-50,50)
print("____")
print(" - por metodo de biseccion :")
x_min,n,error_a=Bisec(f,0,1)
print(f" - para en la iteracion : {n}")
print(f" - x min : {x_min}")
```

5.3

```
from ast import Lambda
import numpy as np
import matplotlib.pyplot as plt
import math

def Bisecc(fun, x_a, x_b, eps=None, steps=100):
    for n in range(steps + 1):
        xr=(x_a+x_b)/2
        if(xr!=0):
            ea=abs((x_b-x_a)/(x_b+x_a))
        if (ea<eps):
            return xr
        if f(xr) == 0:
            return xr

        test=f(x_a)*f(xr)
        if test < 0:
            x_b = xr
        elif test>0 :
            x_a = xr
        else:
            ea=0

    return xr

def falsaPosicion(funcion, x_a, x_b,error_r=0.001, iteraciones=100):
    # Se inicializan las variables
    solucion= None
    contador = 0
    error_calculado = 101
    while contador <= iteraciones and error_calculado>error_r:
        contador+=1
        solucion = x_b-((funcion(x_b)*(x_b - x_a))/(funcion(x_b) - funcion(x_a)))
        error_calculado = abs((solucion - x_a)/solucion)*100
        #Se redefine el nuevo intervalo con los signos
        if funcion(x_a) * funcion(solucion)>=0:
            x_a = solucion
        else:
            x_b = solucion

    print('la solucion aproximada es: {:.3f}'.format(solucion))
    print('encontrada en: {:.0f}'.format(contador) + ' iteraciones')
    print('con un error de: {:.3f}'.format(error_calculado) + '%')

def Graficar_funcion(fx,inicio, parada,semilla=1000):
    xi=np.linspace(inicio,parada,semilla)
    fi=fx(xi)
    plt.plot(xi,fi, label='f(x)')
    plt.axhline(0, color='k')
    plt.title('Grafica de Funcion')
    plt.legend()
    plt.show()

print("la ecuacion cuadratica tiene esta forma : ")
print("f(x) = ax^5 + bx^4 + cx^3 + dx^2 + ex + f")
a = 0.7
b = -8
c = 44
d = -98
e = 82
f_ = -25
print("____")
print(f"teniendo una forma de : {a}x^5 + {b}x^4 + {c}x^3 + {d}x^2 + {e}x + {f_} \n\n")
f = lambda x: (a*x**5)+(b*x**4)+(c*x**3)+(d*x**2)+(e*x)+f_
Graficar_funcion(f,-30,30)
print("____")
print(" - por metodo de biseccion :")
x=Bisecc(f,0.5,1,0.1)
# print(f" - para en la iteracion : {n}")
print(f" - x : {x}")

print("____")
print(" - por metodo de la falsa posicion : ")
falsaPosicion(f,0.5,1,0.02)
```

5.4

```
from ast import Lambda
import numpy as np
import matplotlib.pyplot as plt
import math

def falsaPosicion(funcion, x_a, x_b, iteraciones=100, error_r=0.001):
    # Se inicializan las variables
    solucion= None
    contador = 0
    error_calculado = 101
    #Se evalua si la raiz esta dentro del intervalo
    if funcion(x_a) * funcion(x_b) <= 0:
        #Se procede a calcular la funcion
        while contador <= iteraciones and error_calculado>error_r:
            contador+=1
            solucion = x_b-((funcion(x_b)*(x_b - x_a))/(funcion(x_b) - funcion(x_a)))
            error_calculado = abs((solucion - x_a)/solucion)*100
            #Se redefine el nuevo intervalo con los signos
            if funcion(x_a) * funcion(solucion)>=0:
                x_a = solucion
            else:
                x_b = solucion

        print('la solucion aproximada es: {:.3f}'.format(solucion))
        print('encontrada en: {:.0f}'.format(contador) + 'iteraciones')
        print('con un error de:{:.3f}'.format(error_calculado) + '%')
    else:
        print('no existe solucion en ese intervalo')

def Graficar_funcion(fx,inicio, parada,semilla=1000):
    xi=np.linspace(inicio,parada,semilla)
    fi=fx(xi)
    plt.plot(xi,fi, label='f(x)')
    plt.axhline(0, color='k')
    plt.title('Grafica de Funcion')
    plt.legend()
    plt.show()

print("la ecuacion cuadratica tiene esta forma : ")
print("f(x) =cx^3 + dx^2 + ex + f")
a = -2.75
b = 18
c = -21
d = -12
print("____")
print(f"teniendo una forma de : {a}x^3 + {b}x^2 + {c}x + {d} \n\n")
f = lambda x: (a*x**3)+(b*x**2)+(c*x)+(d)
Graficar_funcion(f,-100,100)

print("____")
print(" - por metodo de la falsa posicion : ")
falsaPosicion(f,-1,1)
```

5.5

```
from ast import Lambda
import numpy as np
import matplotlib.pyplot as plt
import math

def Bisecc(fun, x_a, x_b, eps=None, steps=100):
    for n in range(steps + 1):
        xr=(x_a+x_b)/2
        if(xr!=0):
            ea=abs((x_b-x_a)/(x_b+x_a))
        if (ea<eps):
            return xr
        if f(xr) == 0:
            return xr

        test=f(x_a)*f(xr)
        if test < 0:
            x_b = xr
        elif test>0 :
            x_a = xr
        else:
            ea=0

    return xr

def Graficar_funcion(fx,inicio, parada,semilla=1000):
    xi=np.linspace(inicio,parada,semilla)
    fi=fx(xi)
    plt.plot(xi,fi, label='f(x)')
    plt.axhline(0, color='k')
    plt.title('Grafica de Funcion')
    plt.legend()
    plt.show()

print("la ecuacion cuadratica tiene esta forma : ")
print("f(x) =sen(x) + x^2")
print("_____")
f = lambda x: np.sin(x)-x**2
Graficar_funcion(f,-100,100)

print("_____")
print(" - por metodo de Biseccion : ")
x=Bisecc(f,0.5,1,0.02)
print(" - x : ",x)
print("_____")
print(" - replasamos el x encontrado : ")
print(" - ",f(x))
```

5.6

```
from ast import Lambda
import numpy as np
import matplotlib.pyplot as plt
import math

def Bisecc(fun, x_a, x_b, eps=None, steps=100):
    for n in range(steps + 1):
        xr=(x_a+x_b)/2
        if(xr!=0):
            ea=abs((x_b-x_a)/(x_b+x_a))
            if (ea<eps):
                return xr
            if f(xr) == 0:
                return xr
            fa=f(x_a)
            fr=f(xr)
            test=fa*fr
            if test < 0:
                x_b = xr
            elif test>0 :
                x_a = xr
            else:
                ea=0
        return xr

def falsaPosicion(funcion, x_a, x_b,error_r=0.001, iteraciones=100):
    # Se inicializan las variables
    solucion= None
    contador = 0
    error_calculado = 101
    while contador <= iteraciones and error_calculado>error_r:
        contador+=1
        solucion = x_b-((funcion(x_b)*(x_b - x_a))/(funcion(x_b) - funcion(x_a)))
        error_calculado = abs((solucion - x_a)/solucion)*100
        #Se redefine el nuevo intervalo con los signos
        if funcion(x_a) * funcion(solucion)>=0:
            x_a = solucion
        else:
            x_b = solucion
    print('la solucion aproximada es: {:.3f}'.format(solucion))
    print('encontrada en: {:.0f}'.format(contador) + ' iteraciones')
    print('con un error de:{:.3f}'.format(error_calculado) + '%')

def Graficar_funcion(fx,inicio, parada,semilla=1000):
    xi=np.linspace(inicio,parada,semilla)
    fi=fx(xi)
    plt.plot(xi,fi, label='f(x)')
    plt.axhline(0, color='k')
    plt.title('Grafica de Funcion')
    plt.legend()
    plt.show()

print("la ecuacion cuadratica tiene esta forma : ")
print("f(x) = Ln x^2 c")
c = -0.7
print("____")
print(f"teniendo una forma de : Ln x^2 + {c} \n\n")

f = lambda x: (np.log(x**2))+c
Graficar_funcion(f,-30,30)
print("____")
print(" - por metodo de biseccion :")
x=Bisecc(f,0.5,2,0.004,3)
# print(f" - para en la iteracion : {n}")
print(f" - x : {x}")

print("____")
print(" - por metodo de la falsa posicion : ")
falsaPosicion(f,0.5,2,0.03,3)
```

5.7

```
from ast import Lambda
import numpy as np
import matplotlib.pyplot as plt
import math

def falsaPosicion(funcion, x_a, x_b,error_r=0.001, iteraciones=100):
    x_real=2.666666
    solucion= None
    contador = 0
    error_calculado = 101
    error_rpv=0
    while contador <= iteraciones and error_calculado>error_r:
        contador+=1
        solucion = x_b-((funcion(x_b)*(x_b - x_a))/(funcion(x_b) - funcion(x_a)))
        error_rpv=abs((x_real-solucion)/x_real)
        error_calculado = abs((solucion - x_a)/solucion)
        print("f" - error_a : {error_calculado}")
        print("f" - error_t : {error_rpv}")
        #Se redefine el nuevo intervalo con los signos
        if funcion(x_a) * funcion(solucion)>0:
            x_a = solucion
        else:
            x_b = solucion

    print('la solucion aproximada es: {:.3f}'.format(solucion))
    print('encontrada en: {:.0f}'.format(contador) + ' iteraciones')

def Graficar_funcion(fx,inicio, parada,semilla=1000):
    xi=np.linspace(inicio,parada,semilla)
    fi=fx(xi)
    plt.plot(xi,fi, label='f(x)')
    plt.axhline(0, color='k')
    plt.title('Grafica de Funcion')
    plt.legend()
    plt.show()

print("la ecuacion cuadratica tiene esta forma : ")
print("f(x) = (ax+b)/x")
a=-0.3
b=0.8
print("_____")
print(f"teniendo una forma de : ({a}x+{b})/x \n\n")

print("_____")
print(" METODO ANALITICO")
print(" (-0.3x+0.8)/x = 0 ; x != 0")
print(" -0.3x = -0.8")
print(" x = 2.666 \n\n")

f = lambda x: ((a*x) + b)/x
Graficar_funcion(f,-30,30)

print("_____")
print(" - por metodo de la falsa posicion : ")
falsaPosicion(f,1,3,0.03)
```

5.8

```

from ast import Lambda
import numpy as np
import matplotlib.pyplot as plt

def falsaposicion(funcion, x_a, x_b,error_r=0.001, iteraciones=100):
    # Se inicializan las variables
    solucion= None
    contador = 0
    error_calculado = 101
    while contador <= iteraciones and error_calculado>error_r:
        contador+=1
        solucion = x_b-((funcion(x_b)*(x_b - x_a))/(funcion(x_b) - funcion(x_a)))
        error_calculado = abs((solucion - x_a)/solucion)*100
        #Se redefine el nuevo intervalo con los signos
        if funcion(x_a) * funcion(solucion)>0:
            x_a = solucion
        else:
            x_b = solucion

    print('la solucion aproximada es: {:.3f}'.format(solucion))
    print('encontrada en: {:.0f}'.format(contador) + ' iteraciones')
    print('con un error de: {:.3f}'.format(error_calculado))

def Graficar_funcion(fx,inicio, parada,semilla=1000):
    xi=np.linspace(inicio,parada,semilla)
    fi=fx(xi)
    plt.plot(xi,fi, label='f(x)')
    plt.axhline(0, color='k')
    plt.title('Grafica de Funcion')
    plt.legend()
    plt.show()

print("la ecuacion cuadratica tiene esta forma : ")
print("f(x) = x^2 + b")
a=1
b=-18
print("____")
print(f"teniendo una forma de : {a}x^2 + {b} \n\n")

f = lambda x: (a*x**2) + b
Graficar_funcion(f,-30,30)

print("____")
print(" - por metodo de la falsa posicion : ")
falsaPosicion(f,4,5,0.05)

```

5.9

```

from ast import Lambda
import numpy as np
import matplotlib.pyplot as plt

def falsaPosicion(funcion, x_a, x_b,error_r=0.001, iteraciones=100):
    # Se inicializan las variables
    solucion= None
    contador = 0
    error_calculado = 101
    while contador <= iteraciones and error_calculado>error_r:
        contador+=1
        solucion = x_b-((funcion(x_b)*(x_b - x_a))/(funcion(x_b) - funcion(x_a)))
        error_calculado = abs((solucion - x_a)/solucion)*100
        #Se redefine el nuevo intervalo con los signos
        if funcion(x_a) * funcion(solucion)>0:
            x_a = solucion
        else:
            x_b = solucion

    print('la solucion aproximada es: {:.3f}'.format(solucion))
    print('encontrada en: {:.0f}'.format(contador) + ' iteraciones')
    print('con un error de: {:.3f}'.format(error_calculado) )
    return solucion

def Graficar_funcion(fx,inicio, parada,semilla=1000):
    xi=np.linspace(inicio,parada,semilla)
    fi=fx(xi)
    plt.plot(xi,fi, label='f(x)')
    plt.axhline(0, color='k')
    plt.title('Grafica de Funcion')
    plt.legend()
    plt.show()

print("la ecuacion cuadratica tiene esta forma : ")
print("f(x) = x^2*|cos((x)^(1/2))|")
a=1
b=-18
print("____")
print(f"teniendo una forma de : \n\n")

f = lambda x: ((x**2)*np.absolute(np.cos(np.sqrt(x))))-5
Graficar_funcion(f,-30,30)

print("____")
print(" - por metodo de la falsa posicion : ")
solucion = falsaPosicion(f,0,5,0.01)
print("____")
print(" - el resultado reemplazado en la funcion es : ",f(solucion))
print(" - el error es minimo :) ")

```

5.10

```
from ast import Lambda
import numpy as np
import matplotlib.pyplot as plt
import math

def falsaPosicion(funcion, x_a, x_b, iteraciones=100, error_r=0.001):
    # Se inicializan las variables
    x_real = 5.60979
    solucion = None
    contador = 0
    error_calculado = 101
    #Se evalua si la raiz esta dentro del intervalo
    if funcion(x_a) * funcion(x_b) <= 0:
        while contador <= iteraciones and error_calculado>error_r:
            contador+=1
            solucion = x_b-((funcion(x_b)*(x_b - x_a))/(funcion(x_b) - funcion(x_a)))
            error_rpv=abs((x_real-solucion)/x_real)
            error_calculado = abs((solucion - x_a)/solucion)
            print(f" - error_a : {error_calculado}")
            print(f" - error_t : {error_rpv}")
            #Se redefine el nuevo intervalo con los signos
            if funcion(x_a) * funcion(solucion)>=0:
                x_a = solucion
            else:
                x_b = solucion

        print('la solucion aproximada es: {:.3f}'.format(solucion))
        print('encontrada en: {:.0f}'.format(contador) + ' iteraciones')
        print('con un error de:{:.3f}'.format(error_calculado) + '%')
    else:
        print('no existe solucion en ese intervalo')

def Graficar_funcion(fx,inicio, parada,semilla=1000):
    xi=np.linspace(inicio,parada,semilla)
    fi=fx(xi)
    plt.plot(xi,fi, label='f(x)')
    plt.axhline(0, color='k')
    plt.title('Grafica de Funcion')
    plt.legend()
    plt.show()

print("la ecuacion cuadratica tiene esta forma : ")
print("f(x) =ax^4 + bx^3 + cx^2 + dx + e")
a = 1
b = -8
c = -35
d = 450
e = -1001
print("____")
print(f"teniendo una forma de : {a}x^4 + {b}x^3 + {c}x^2 + {d}x + {e} \n\n")
f = lambda x: (a*x**4)+(b*x**3)+(c*x**2)+(d*x)+(e)
Graficar_funcion(f,-100,100)

print("____")
print(" - por metodo de la falsa posicion : ")
falsaPosicion(f,4.5,6.5,0.01)|
```

5.11

```

from ast import Lambda
import numpy as np
import matplotlib.pyplot as plt
import math

def falsaPosicion(funcion, x_a, x_b, error_r=0.001 , iteraciones=100):
    # Se inicializan las variables
    x_real = 5.60979
    solucion= None
    contador = 0
    error_calculado = 101
    #Se evalua si la raiz esta dentro del intervalo
    if funcion(x_a) * funcion(x_b) <= 0:
        while contador <= iteraciones and error_calculado>error_r:
            contador+=1
            solucion = x_b-((funcion(x_b)*(x_b - x_a))/(funcion(x_b) - funcion(x_a)))
            error_rpv=abs((x_real-solucion)/x_real)
            error_calculado = abs((solucion - x_a)/solucion)
            print(f" - error_a : {error_calculado}")
            print(f" - error_t : {error_rpv}")
            #Se redefine el nuevo intervalo con los signos
            if funcion(x_a) * funcion(solucion)>=0:
                x_a = solucion
            else:
                x_b = solucion

            print('la solucion aproximada es: {:.3f}'.format(solucion))
            print('encontrada en: {:.0f}'.format(contador) + ' iteraciones')
            print('con un error de:{:.3f}'.format(error_calculado) + '%' )
    else:
        print('no existe solucion en ese intervalo')

def Graficar_funcion(fx,inicio, parada,semilla=1000):
    xi=np.linspace(inicio,parada,semilla)
    fi=fx(xi)
    plt.plot(xi,fi, label='f(x)')
    plt.axhline(0, color='k')
    plt.title('Grafica de Funcion')
    plt.legend()
    plt.show()

print("la ecuacion cuadratica tiene esta forma : ")
print("f(x) = ax^3.5 + b = 0")
a = 1
b = -80
print("____________________________________")
print(f"teniendo una forma de : {a}x^3.5 + {b} \n\n")
print("____________________________________")
print(" - Solucion Analitica : ")
print(" - x^3.5 - 80 = 0")
print(" - x^3.5 = 80 ")
print(" - x^(7/2) = 80 ")
print(" - x = 80 ^ (2/7)")
print(" - x = 6400^(1/7)")
print(" - x = 3.497")
f = lambda x: (a*x**3.5) + b
Graficar_funcion(f,-100,100)

print("____________________________________")
print(" - por metodo de la falsa posicion : ")
falsaPosicion(f,2.5,0.025)

```

5.12

```

from ast import Lambda
import numpy as np
import matplotlib.pyplot as plt
import sympy as sp

def Bisecc(fun, x_a, x_b, eps=None, steps=100):
    for n in range(steps + 1):
        xr=(x_a+x_b)/2
        if(xr!=0):
            ea=abs((x_b-x_a)/(x_b+x_a))
        if (ea<eps):
            return xr
        if f(xr) == 0:
            return xr
        fa=f(x_a)
        fr=f(xr)
        test=fa*fr
        if test < 0:
            x_b = xr
        elif test>0 :
            x_a = xr
        else:
            ea=0
    return xr

def Graficar_funcion(fx,inicio, parada,semilla=1000):
    xi=np.linspace(inicio,parada,semilla)
    fi=fx(xi)
    plt.plot(xi,fi, label='f(x)')
    plt.axhline(0, color='k')
    plt.title('Grafica de Funcion')
    plt.legend()
    plt.show()

print("la ecuacion tiene esta forma : ")
print("f(x) = ax^6 + bx^4 + cx + d")
a= -2
b= -1.5
c= 10
d= 2
print("____")
print("teniendo una forma de : {a}x^6 + {b}x^4 + {c}x + {d} \n\n")

f = lambda x: a*x**5 + b*x**3 + c
print(" - nos piden el punto maximo entre 0 - 1, para \n lo cual derivamos")
print(" dx : ")
print("f'(x) = -12 x^(5)-1.5*4 x^(3)+10")
print("____")
Graficar_funcion(f,-30,30)
print("____")
print(" - por metodo de biseccion :")
x=Bisecc(f,0,1,0.05)
# print(f" - para en la iteracion : {n}")
print(f" - x : {x}")

```

5.13

```

from ast import Lambda
import numpy as np
import matplotlib.pyplot as plt
import sympy as sp

def falsaPosicion(funcion, x_a, x_b,error_r=0.001, iteraciones=100):
    # Se inicializan las variables
    solucion= None
    contador = 0
    error_calculado = 101
    while contador <= iteraciones and error_calculado>error_r:
        contador+=1
        solucion = x_b-((funcion(x_b)*(x_b - x_a))/(funcion(x_b) - funcion(x_a)))
        error_calculado = abs((solucion - x_a)/solucion)*100
        #Se redefine el nuevo intervalo con los signos
        if funcion(x_a) * funcion(solucion)>=0:
            x_a = solucion
        else:
            x_b = solucion

    print('la solucion aproximada es: {:.3f}'.format(solucion))
    print('encontrada en: {:.0f}'.format(contador) + ' iteraciones')
    print('con un error de:{:.3f}'.format(error_calculado) )

def Graficar_funcion(fx,inicio, parada,semilla=1000):
    xi=np.linspace(inicio,parada,semilla)
    fi=fx(xi)
    plt.plot(xi,fi, label='f(x)')
    plt.axhline(0, color='k')
    plt.title('Grafica de Funcion')
    plt.legend()
    plt.show()

print("la ecuacion tiene esta forma : ")
print("v = g*m/cr*(1-exp(-cr/m*tf))")
g=9.8
cr=15
v=35
tf=9
print("_____")

f = lambda m: (g*m/cr)*(1-np.e**((-cr/m)*tf))-v
print(" - tomaremos puntos entre 50 y 70")
print("_____")
Graficar_funcion(f,-30,30)

print("_____")
print(" - por metodo de la falsa posicion : ")
falsaPosicion(f,50,70,0.01)
print(" - reemplazando : ")
print(f(59.841)+v)

```

5.15

```

from ast import Lambda
import numpy as np
import matplotlib.pyplot as plt
import math

def Bisecc(fun, x_a, x_b, eps=None, steps=100):
    for n in range(steps + 1):
        xr=(x_a+x_b)/2
        if(xr==0):
            ea=abs((x_b-x_a)/(x_b+x_a))
        if (ea<eps):
            return xr
        if f(xr) == 0:
            return xr
        fa=f(x_a)
        fr=f(xr)
        test=fa*fr
        if test < 0:
            x_b = xr
        elif test>0 :
            x_a = xr
        else:
            ea=0
    return xr

def falsaPosicion(funcion, x_a, x_b,error_r=0.001, iteraciones=100):
    # Se inicializan las variables
    solucion = None
    contador = 0
    error_calculado = 101
    while contador <= iteraciones and error_calculado>error_r:
        contador+=1
        solucion = x_b-((funcion(x_b)*(x_b - x_a))/(funcion(x_b) - funcion(x_a)))
        error_calculado = abs((solucion - x_a)/solucion)*100
        #Se redefine el nuevo intervalo con los signos
        if funcion(x_a) * funcion(solucion)>0:
            x_a = solucion
        else:
            x_b = solucion
    print('la solucion aproximada es: {:.3f}'.format(solucion))
    print('encontrada en: {:.0f}'.format(contador) + ' iteraciones')
    print('con un error de: {:.3f}'.format(error_calculado) + '%')

def Graficar_funcion(fx,inicio,parada,sevilla=1000):
    xi=np.linspace(inicio,parada,sevilla)
    fi=fx(xi)
    plt.plot(xi,fi, label='f(x)')
    plt.axhline(0, color='k')
    plt.title('Grafica de Funcion')
    plt.legend()
    plt.show()

print("la ecuacion tiene esta forma : ")
print("f(y) = 1-((q**2)/(g*(3*y+((y**2)/2)**3))**((3+y))")
q=20
g=9.81
print("      ")
f = lambda y: 1-((q**2)/(g*(3*y+((y**2)/2)**3))**((3+y)))
Graficar_funcion(f,-30,30)
print("      ")
print(" - por metodo de biseccion")
x=Bisecc(f,0.5,2.5,0.01,10)
# print("f" - para en la iteracion : {n})
print("f" - x : {x})

print("      ")
print(" - por metodo de la falsa posicion : ")
falsaPosicion(f,0.5,2.5,0.01,9)

```

5.16

```

from ast import Lambda
import numpy as np
import matplotlib.pyplot as plt
import math

def falsaPosicion(funcion, x_a, x_b,error_r=0.001, iteraciones=100):
    # Se inicializan las variables
    solucion= None
    contador = 0
    error_calculado = 101
    while contador <= iteraciones and error_calculado>error_r:
        contador+=1
        solucion = x_b-((funcion(x_b)*(x_b - x_a))/(funcion(x_b) - funcion(x_a)))
        error_calculado = abs((solucion - x_a)/solucion)*100
        #Se redefine el nuevo intervalo con los signos
        if funcion(x_a) * funcion(solucion)>=0:
            x_a = solucion
        else:
            x_b = solucion

    print('la solucion aproximada es: {:.3f}'.format(solucion))
    print('encontrada en: {:.0f}'.format(contador) + ' iteraciones')
    print('con un error de:{:.3f}'.format(error_calculado) + '%')

def Graficar_funcion(fx,inicio, parada,semilla=1000):
    xi=np.linspace(inicio,parada,semilla)
    fi=fx(xi)
    plt.plot(xi,fi, label='f(x)')
    plt.axhline(0, color='k')
    plt.title('Grafica de Funcion')
    plt.legend()
    plt.show()

print("la ecuacion tiene esta forma : ")
print("f(y) = 1-((q**2)/(g*(3*y+((y**2)/2))**3))*((3+y))")
r=3
v=30

print("____")
f = lambda h: (np.pi*r*h**2)-((np.pi*h**3)/3)-v
Graficar_funcion(f,-30,30)

print("____")
print(" - por metodo de la falsa posicion : ")
falsaPosicion(f,0,2,0.01,2)

```

5.18

```

from ast import Lambda
import numpy as np
import matplotlib.pyplot as plt
import math

#parametros de entra de la funcion con: la ecuacion, x inicial y final, error , cantidad de iteraciones
def Bisec(fun, x_a, x_b, eps=None, steps=100):
    for n in range(steps + 1):
        xr=(x_a+x_b)/2
        if(xr!=0):
            ea=abs((x_b-x_a)/(x_b+x_a))
        if (ea<eps):
            return xr
        if f(xr) == 0:
            return xr
        fa=f(x_a)
        fr=f(xr)
        test=fa*fr
        if test < 0:
            x_b = xr
        elif test>0 :
            x_a = xr
        else:
            ea=0
    return xr

def Graficar_funcion(fx,inicio, parada,sevilla=1000):
    xi=np.linspace(inicio,parada,sevilla)
    fi=fx(xi)
    plt.plot(xi,fi, label='f(x)')
    plt.axhline(0, color='k')
    plt.title('Grafica de Funcion')
    plt.legend()
    plt.show()

print("la ecuacion tiene esta forma : ")
print("f(y) = 1-((q**2)/(g*(3*y+(y**2)/2))**3))**((3+y))")
q=20
g=9.81
print("____")
f = lambda y: 1-((q**2)/(g*(3*y+(y**2)/2))**3))**((3+y))
Graficar_funcion(f,-30,30)
print("____")
print(" - por metodo de biseccion :")
x=Bisec(f,0.5,2.5,0.01,10)
print(f" - x : {x}")
#se comprueba :
print(f(x))

```

5.19

```

from ast import Lambda
import numpy as np
import matplotlib.pyplot as plt
import math
#parametros de entra de la funcion con: la ecuacion, x inicial y final, error , cantidad de iteraciones
def Bisec(fun, x_a, x_b, eps=None, steps=100):
    for n in range(steps + 1):
        xr=(x_a+x_b)/2
        if(xr!=0):
            ea=abs((x_b-x_a)/(x_b+x_a))
        if (ea<eps):
            print(f" - paro en la iteracion (n)")
            return xr
        if f(xr) == 0:
            print(f" - paro en la iteracion (n)")
            return xr
        fa=f(x_a)
        fr=f(xr)
        test=fa*fr
        if test < 0:
            x_b = xr
        elif test>0 :
            x_a = xr
            fa=fr#Cambio aqui
        else:
            ea=0
    return xr

def Graficar_funcion(fx,inicio, parada,sevilla=1000):
    xi=np.linspace(inicio,parada,sevilla)
    fi=fx(xi)
    plt.plot(xi,fi, label='f(x)')
    plt.axhline(0, color='k')
    plt.title('Grafica de Funcion')
    plt.legend()
    plt.show()

print("la ecuacion tiene esta forma : ")
print("v = g*m/cr*(1-exp(-cr/m*tf))")
g=0.8
m=68.1
v=40
tf=10
print("____")
f = lambda c: (g*m/c)*(1-np.e**((-c/m)*tf))-v
Graficar_funcion(f,-30,30)
print("____")
print(" - por metodo de biseccion :")
x=Bisec(f,12,16,0.002)
print(f" - x : {x}")

```

5.20, 5.21, 5.22

```
from ast import Lambda
import numpy as np
import matplotlib.pyplot as plt
import math

def Bisecc(fun, x_a, x_b, eps=None, steps=100):
    for n in range(steps + 1):
        xr=(x_a+x_b)/2
        if(xr!=0):
            ea=abs((x_b-x_a)/(x_b+x_a))
        if (ea<eps):
            return xr
        if f(xr) == 0:
            return xr
        fa=f(x_a)
        fr=f(xr)
        test=fa*fr
        if test < 0:
            x_b = xr
        elif test>0 :
            x_a = xr
        else:
            ea=0
    return xr

def falsaPosicion(funcion, x_a, x_b,error_r=0.001, iteraciones=100):
    # Se inicializan las variables
    solucion= None
    contador = 0
    error_calculado = 101
    while contador <= iteraciones and error_calculado>error_r:
        contador+=1
        solucion = x_b-((funcion(x_b)*(x_b - x_a))/(funcion(x_b) - funcion(x_a)))
        error_calculado = abs((solucion - x_a)/solucion)*100
        #se redefine el nuevo intervalo con los signos
        if funcion(x_a) * funcion(solucion)>0:
            x_a = solucion
        else:
            x_b = solucion

    print('la solucion aproximada es: {:.5f}'.format(solucion))
    print('encontrada en: {:.0f}'.format(contador) + ' iteraciones')
    print('con un error de: {:.3f}'.format(error_calculado) + '%')

def Graficar_funcion(fx,inicio, parada,semilla=1000):
    xi=np.linspace(inicio,parada,semilla)
    fi=fx(xi)
    plt.plot(xi,fi, label='f(x)')
    plt.axhline(0, color='k')
    plt.title('Grafica de Funcion')
    plt.legend()
    plt.show()

f = lambda x: (x**10)-1
Graficar_funcion(f,-30,30)
print("_____")
print(" - por metodo de bisección :")
x=Bisecc(f,0,1.3,0.01)
# print(" - para en la iteracion : {n}")
print(" - x : {x}")

print("_____")
print(" - por metodo de la falsa posicion : ")
falsaPosicion(f,0,1.3,0.01)
print(" - comprobando la raiz : ")
print(f( 0.99969))
```

Capítulo VI

Métodos abiertos

6.1

```
import math
from decimal import Decimal
from sympy import *

# ejercicio 6.1
def metodoPuntoFijo():

    xInicial = float(input("Ingrese el valor inicial: "));
    eInicial = float(input("Ingrese el error E(a): "));
    it = 1;

    print("it      xi      g(x)      error      error(%)");
    while(True):
        # aquí tenemos el g(x)
        xr = 2*math.sin(math.sqrt(xInicial));
        e = abs((xr - xInicial)/xr);

        # imprimir la lista de valores
        print(f"{it} --- {round(Decimal(xInicial),6)} --- {round(Decimal(xr),6)} --- {round(Decimal(e),6)} --- {round(Decimal(e/e)*100,2)}%");

        xInicial = xr;
        it = it + 1;

        # termina cuando el error es menor igual que el error del problema

        if round(Decimal(e),5) <= eInicial:
            print(f"Raiz aproximada: {round(Decimal(xInicial),6)} con un error de {round(Decimal(e),6)*100}%");
            break;

# 6.2 en la seccante
```

```
def metodoSecante():

    xInicial0 = float(input("Ingrese el valor de x1-1: "));
    xInicial1 = float(input("Ingrese el valor de x1: "));

    # Como nos da el número de iteraciones entonces no es necesario tener el error
    # eInicial = float(input("Ingrese el error: "));

    it = 1;
    print("it      x1-1      x1      x1+1      f(x1)      f(x1-1)      error      error(%)");
    while(True):
        f1 = 2*math.pow(xInicial1,3)-11.7*math.pow(xInicial1,2)+17.7*xInicial1-5;
        f2 = 2*math.pow(xInicial0,3)-11.7*math.pow(xInicial0,2)+17.7*xInicial0-5;

        xr = xInicial1 - ((f1*(xInicial1-xInicial0))/(f1 - f2));
        e = abs((xInicial1 - xInicial0)/xInicial1);

        # Imprime la lista de valores
        print(f"{it} --- {round(Decimal(xInicial0),6)} --- {round(Decimal(xInicial1),6)} --- {round(Decimal(xr),6)} --- {round(Decimal(f1),6)} --- {round(Decimal(f2),6)} --- {round(Decimal(e),6)} --- {round(Decimal(e/e)*100,2)}%");

        xInicial0 = xInicial1;
        xInicial1 = xr;
        it = it + 1;

        # termina cuando el error es menor que el error del problema

        if it>4:
            print(f"Raiz aproximada: {round(Decimal(xr),6)} con un error de {round(Decimal(e),6)*100}%");
            break;
```

6.3

PUNTO FIJO					
it	xi	g(x)	error	e%	0.05%
1	5	3.39116499	0.47441956	47.441956%	
2	3.39116499	2.93327411	0.15610232	15.610232%	
3	2.93327411	2.78924603	0.05163692	5.163692%	
4	2.78924603	2.74237905	0.0170899	1.708990%	
5	2.74237905	2.72695476	0.00565623	0.565623%	
6	2.72695476	2.7218594	0.00187202	0.187202%	
7	2.7218594	2.72017406	0.00061957	0.061957%	
8	2.72017406	2.71961639	0.00020505	0.020505%	
9	2.71961639	2.71943183	6.7865E-05	0.006787%	
10	2.71943183	2.71937075	2.2461E-05	0.002246%	

Newton raphson						
it	xi	xi+1	f(x)	f'(x)	error	e%
1	5.00000	3.35366	-13.50000	-8.20000	0.49091	49.090909%
2	3.35366	2.80133	-2.71044	-4.90732	0.19717	19.716558%
3	2.80133	2.72111	-0.30506	-3.80266	0.02948	2.948204%
4	2.72111	2.71934	-0.00644	-3.64222	0.00065	0.064980%
5	2.71934	2.71934	0.00000	-3.63868	0.00000	0.000000%
6	2.71934	2.71934	0.00000	-3.63868	0.00000	0.000000%
7	2.71934	2.71934	0.00000	-3.63868	0.00000	0.000000%
8	2.71934	2.71934	0.00000	-3.63868	0.00000	0.000000%

6.4

```
# ejercicio 6.4
def metodoNewtonRaphson():
    x = Symbol('x');
    fPrincipal = input("función: ");
    xInicial = float(input("Ingrese el valor inicial: "));
    eInicial = float(input("Ingrese el error E(s): "));

    it = 1;

    print("it           xi           xi+1           f(x)           f'(x)           e(s)");
    while(True):

        # f(x)
        # f1 = -1+5.5*xInicial-4*(math.pow(xInicial,2))+0.5*(math.pow(xInicial,3))
        f1 = sympify(fPrincipal).subs(x,xInicial);
        # f'(x)
        # f2 = 5.5 - 8*xInicial+1.5*(math.pow(xInicial,2));
        f2 = sympify(diff(fPrincipal),x).subs(x,xInicial);

        xr = xInicial - (f1/f2);
        e = abs((xr - xInicial)/xr)*100;

        # imprimir los valores
        print(f'{it} --- {round(xInicial,6)} --- {round(xr,6)} --- {round(f1,6)} --- {round(f2,6)} --- {round(e,6)}%');

        xInicial = xr;
        it = it + 1;

        # termina cuando el error es menor que el error del problema
        if round((e),6) < eInicial:
            print(f"Raíz aproximada: {round(xInicial,6)} con un error de {round(e,6)}%");
            break;
```

6.5

```

# ejercicio 6.4
def metodoNewtonRaphson():
    x = Symbol("x");
    fPrincipal = input("función: ");
    xInicial = float(input("Ingrese el valor inicial: "));
    eInicial = float(input("Ingrese el error E(s): "));

    it = 1;

    print("it      xi      xi+1      f(x)      f'(x)      e(s)");
    while(True):
        # f(x)
        # f1 = -1+5.5*xInicial-4*(math.pow(xInicial,2))+0.5*(math.pow(xInicial,3))
        f1 = sympify(fPrincipal).subs(x,xInicial); ← Ingresar la función requerida
        # f'(x)
        # f2 = 5.5 - 8*xInicial+1.5*(math.pow(xInicial,2));
        f2 = sympify(diff(fPrincipal),x).subs(x,xInicial);

        xr = xInicial - (f1/f2);
        e = abs((xr - xInicial)/xr)*100;

        # imprimir los valores
        print(f"{{it}} --- {{round({{xInicial}},6)}} --- {{round({{xr}},6)}} --- {{round({{f1}},6)}} --- {{round({{f2}},6)}} --- {{round({{e}},6)}%}");

        xInicial = xr;
        it = it + 1;

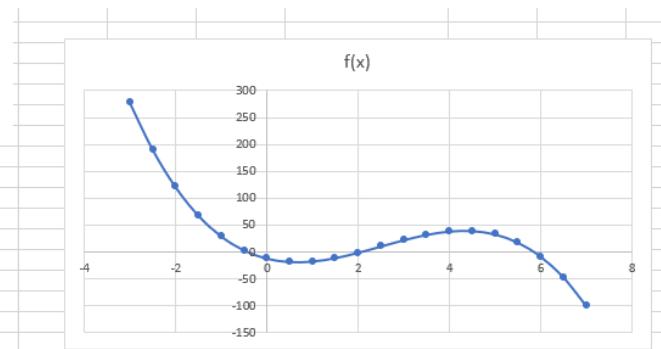
        # termina cuando el error es menor que el error del problema
        if round((e),5) < eInicial:
            print(f"Raíz aproximada: {{round({{xInicial}},6)}} con un error de {{round({{e}},6)}%}");
            break;

```

6.6

6.6 Determine la raíz real más pequeña de $f(x) = -12 - 21x + 18x^2 - 2.4x^3$: a) en forma gráfica, y b) con el empleo del método de la secante para un valor de ε_s que corresponda a tres cifras significativas.

x	f(x)
-3	277.8
-2.5	190.5
-2	121.2
-1.5	68.1
-1	29.4
-0.5	3.3
0	-12
0.5	-18.3
1	-17.4
1.5	-11.1
2	-1.2
2.5	10.5
3	22.2
3.5	32.1
4	38.4
4.5	39.3
5	33
5.5	17.7
6	-8.4
6.5	-47.1
7	-100.2



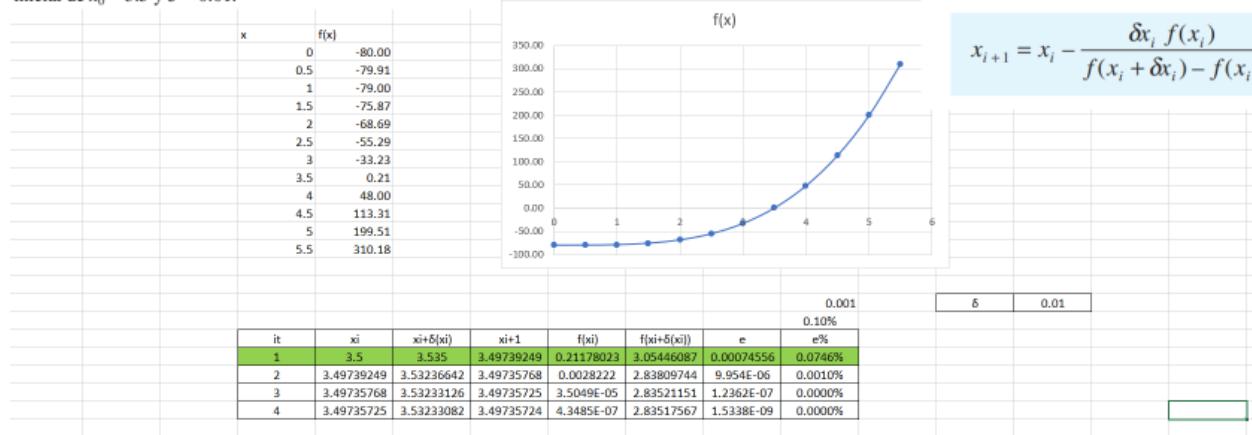
it	xi-1	xi	xi+1	f(xi)	f(xi-1)	xi - (xi-1)	error	error %
1	0.000000	3.000000	1.052632	22.200000	-12.000000	3.000000	1.000000	100.000000%
2	3.000000	1.052632	1.896024	-16.959907	22.200000	-1.947368	1.850000	185.000000%
3	1.052632	1.896024	2.112706	-3.466640	-16.959907	0.843393	0.444822	44.482173%
4	1.896024	2.112706	2.052157	1.344363	-3.466640	0.216681	0.102561	10.256106%
5	2.112706	2.052157	2.053593	-0.032651	1.344363	-0.060548	0.029505	2.950473%
6	2.052157	2.053593	2.053605	-0.000261	-0.032651	0.001436	0.000699	0.069912%
7	2.053593	2.053605	2.053605	0.000000	-0.000261	0.000012	0.000006	0.000563%
8	2.053605	2.053605	2.053605	0.000000	0.000000	0.000000	0.000000	0.000000%

6.7

it	xi-1	xi	xi+1	f(xi)	f(xi-1)	xi - (xi-1)	error	error %
1	1.000	3.000	-0.023	-1.698	-0.575	2.000	0.667	66.67%
2	3.000	-0.023	-1.226	-0.483	-1.698	-3.023	130.231	13023.08%
3	-0.023	-1.226	0.234	-2.745	-0.483	-1.203	0.981	98.11%
4	-1.226	0.234	0.396	-0.275	-2.745	1.460	6.242	624.19%
it	xi-1	xi	xi+1	f(xi)	f(xi-1)	xi - (xi-1)	error	error %
1	1.500	2.500	2.357	0.166	-0.997	1.000	0.400	40.000%
2	2.500	2.357	2.547	0.670	0.166	-0.143	0.061	6.070%
3	2.357	2.547	2.526	-0.083	0.670	0.190	0.075	7.473%
4	2.547	2.526	2.532	0.031	-0.083	-0.021	0.008	0.829%
it	xi-1	xi	xi+1	f(xi)	f(xi-1)	xi - (xi-1)	error	error %
1	1.500	2.250	1.927	0.754	-0.997	0.750	0.333	33.333%
2	2.250	1.927	1.951	-0.062	0.754	-0.323	0.168	16.761%
3	1.927	1.951	1.945	0.024	-0.062	0.024	0.013	1.253%
4	1.951	1.945	1.945	0.000	0.024	-0.007	0.004	0.354%

6.8

6.8 Determine la raíz real de $x^{3.5} = 80$, con el método de la seante modificada dentro de $\varepsilon_s = 0.1\%$, con el uso de una elección inicial de $x_0 = 3.5$ y $\delta = 0.01$.



6.9

método de newton raphson							
it	xi	xi+1	f(x)	f'(x)	e	e%	
1	3.5	3.365651	0.60625	4.5125	0.0399177	3.99%	
2	3.365651	3.3451123	0.0712487	3.4689969	0.0061399	0.61%	
3	3.3451123	3.3446454	0.0015492	3.3185371	0.0001396	0.01%	

método de la secante							
it	xi-1	xi	xi+1	f(xi)	f(xi-1)	xi-(xi-1)	e
1	2.5	3.5	3.0630631	0.60625	-0.78125	1	0.2857143
2	3.5	3.0630631	3.2919062	-0.6667003	0.60625	-0.4369369	0.1426471
3	3.0630631	3.2919062	3.3670921	-0.1648739	-0.6667003	0.2288432	0.0695169

método de la secante modificado							
it	xi	xi+δ(xi)	xi+1	f(xi)	f(xi+δ(xi))	e	e%
1	3.5	3.535	3.3697997	0.60625	0.7692201	0.0386374	3.864%
2	3.3697997	3.4034977	3.3461609	0.0857044	0.207879	0.0070645	0.706%
3	3.3461609	3.3796225	3.3447016	0.0050332	0.1204392	0.0004363	0.044%

6.10

método newton raphson							
it	xi	xi+1	f(x)	f'(x)	e	e%	
1	0.3	0.1078437	0.75141403	3.91043139	1.78180367	178%	
2	0.1078437	0.14348661	-0.22695159	6.3673702	0.24840582	25%	
3	0.14348661	0.14501208	-0.00895208	5.86838801	0.01051965	1%	

método de la secante							
it	xi-1	xi	xi+1	f(x)	f(xi-1)	xi-(xi-1)	e
1	0.5	0.3	0.03858235	0.75141403	1.32629031	-0.2	0.66666667
2	0.3	0.03858235	0.16494918	-0.70309686	0.75141403	-0.26141765	6.77557638
3	0.03858235	0.16494918	0.1473367	0.11386469	-0.70309686	0.12636683	0.76609554

método de la secante modificado							
it	xi	xi+δ(xi)	xi+1	f(xi)	f(xi+δ(xi))	e	e%
1	0.3	0.303	0.1070065	0.75141403	0.76309443	1.83160366	183.160%
2	0.1070065	0.10807657	0.14346268	-0.23228735	-0.22546923	0.24665727	24.666%
3	0.14346268	0.14489731	0.1450146	-0.00909251	-0.0006872	0.00080883	0.081%
4	0.1450146	0.14646475	0.14501481	-1.2472E-06	0.00846401	0.00999851	1.000%
5	0.14501481	0.14646496	0.14501481	2.1201E-09	0.00846527	0.01	1.000%

Capítulo VII

Raíces de polinomios

7.1

(7.1) $x^4 - 7.5x^3 + 14.5x^2 + 3x - 20 \quad | \quad x-2$

$$\begin{array}{c} x-2=0 \\ x=2 \end{array} \left| \begin{array}{ccccc} 1 & -7.5 & 14.5 & 3 & 20 \\ & 2 & =11 & 7 & 20 \\ \hline 1x^3 & -5.5x^2 & 3.5x+10 & & 0=r(x) \end{array} \right.$$

$\left\{ \begin{array}{l} x=2 \text{ si } r(x)=0 \\ \text{raiz porque } r(2)=0 \\ r(x)=0 \end{array} \right.$

7.2

(7.2) $x^5 - 5x^4 + x^3 - 6x^2 - 7x + 10 \quad | \quad x-2$

$$\begin{array}{c} x-2=0 \\ x=2 \end{array} \left| \begin{array}{cccccc} 1 & -5 & 1 & -6 & -7 & 10 \\ & 2 & -6 & -10 & -32 & -78 \\ \hline 1x^4 & -3x^3 & -5x^2 & -16x & -39 & -68=r(x) \end{array} \right.$$

$\left\{ \begin{array}{l} x=2, \text{ no es una} \\ \text{raiz porque } r(x) \neq 0 \end{array} \right.$

7.3

```

# ejercicio 7.3 método de Muller
def metodoMuller():
    # declaramos las variables que utilizamos
    x = Symbol("x")

    # función que calcule los valores de las funciones fx0, fx1, fx2
    def valueFunction(number):
        value = simplify(Principal).subs(x,number)
        return complex(value)

    # ingresamos la función principal a evaluar
    PPrincipal = input("Función: ")

    # tenemos que ingresar 3 valores que vienen a ser x0 x1 x2 y luego calculamos x3
    x0 = float(input("Valor x0: "));
    x1 = float(input("Valor x1: "));
    x2 = float(input("Valor x2: "));

    # también necesitamos el error máximo para que pueda para la iteración
    eMax = float(input("Error máximo: "));
    print("x3 --- error");
    it = 0;
    while(True):
        # necesitamos la función principal de donde queremos calcular la raíz aproximada
        f0 = valueFunction(x0);
        f1 = valueFunction(x1);
        f2 = valueFunction(x2);

        h0 = x1 - x0;
        h1 = x2 - x1;

        d0 = (f1-f0)/(h0);
        d1 = (f2-f1)/(h1);

        a = (d1-d0)/(h1+h0);
        b = a**2/(h1)+d1;
        c = f2;

        denominador = a;
        discr1 = b**2-4*a*c;

        if abs(b+cmath.sqrt(discr1)) > abs(b-cmath.sqrt(discr1)):
            denominador = b+cmath.sqrt(discr1);
        else:
            denominador = b-cmath.sqrt(discr1);
        x3 = x2 + (-2*c)/denominador;

        # error
        e = abs((x3-x2)/(x3))*100;
        it = it + 1;
        print(it,round(x3.real,5),round(Decimal(x3.imag),6),"i",round(e,3));
        if e<eMax:
            break;

        # siguiente iteración
        x0 = x1;
        x1 = x2;
        x2 = x3;
metodoMuller();

```

7.4

```

def raicesim():
    # tenemos que ingresar 3 valores que vienen a ser x0 x1 x2 y luego calculamos x3
    x0 = float(input("Valor x0: "));
    x1 = float(input("Valor x1: "));
    x2 = float(input("Valor x2: "));

    # también necesitamos el error máximo para que pueda para la iteración
    eMax = float(input("Error máximo: "));
    print("x3 --- error");
    while(True):
        # necesitamos la función principal de donde queremos calcular la raíz aproximada
        # f0 = 2*x0**4+6*x0**2+10;
        # f1 = 2*x1**4+6*x1**2+10;
        # f2 = 2*x2**4+6*x2**2+10;
        f0 = x0**4+x0**2+3*x0-2;
        f1 = x1**4+x1**2+3*x1-2;
        f2 = x2**4+x2**2+3*x2-2

        h0 = x1 - x0;
        h1 = x2 - x1;

        d0 = (f1-f0)/(h0);
        d1 = (f2-f1)/(h1);

        a = (d1-d0)/(h1+h0);
        b = a**2/(h1)+d1;
        c = f2;

        denominador = a;
        discr1 = b**2-4*a*c;
        print(discr1);
        if abs(b+cmath.sqrt(discr1)) > abs(b-cmath.sqrt(discr1)):
            denominador = b+cmath.sqrt(discr1);
        else:
            denominador = b-cmath.sqrt(discr1);
        print(denominador);

        x3 = x2 + (-2*c)/denominador;

        # error
        e = abs((x3-x2)/(x3));
        if e<eMax:
            break;

        # siguiente iteración
        x0 = x1;
        x1 = x2;
        x2 = x3;
raicesim()

```

7.5

```
def metodoBeirtow():
    x = Symbol("x");
    valuesA = [];
    valuesB = [];
    valuesC = [];

    deltaR = 0;
    deltaS = 0;

    er = 1;
    es = 1;
    eInicial = 0.01;
    # 1% = 0.01
    # estos valores deben ingresarse por consola (ojo: en caso de que un error se deben cambiar los valores de r y s)
    r = -1;
    s = -1;
    fPrincipal = input("Función: ");

    # los coeficientes del polinomio
    valuesA = Poly(fPrincipal, x).all_coeffs();
    # grado mayor del polinomio por ejemplo x2 +x +1 + x gradoMax = cantidad - 1
    n = len(valuesA)-1;
    # contador de iteraciones
    contador = 1;
    print("RAÍCES CON EL CÁLCULO BEIRSTOW")
    while(true):
        # calcular los valore de b
        for i in range(n+1):
            if i == 0:
                valuesB.append(valuesA[i]);
            elif i == 1:
                valuesB.append(valuesA[i]+valuesB[i-1]*r);
            else:
                valuesB.append(valuesA[i]+r*valuesB[i-1]+s*valuesB[i-2]);
        for i in range(n):
            if i == 0:
                valuesC.append(valuesB[i]);
            elif i == 1:
                valuesC.append(valuesB[i]+valuesC[i-1]*r);
            else:
                valuesC.append(valuesB[i]+r*valuesC[i-1]+s*valuesC[i-2]);
        # ahora calculamos los deltas
        deltaR = float(-valuesB[n-1]*valuesC[n-2]+valuesB[n]*valuesC[n-3])/float((valuesC[n-2])**2-valuesC[n-1]*valuesC[n-3]);
        deltaS = float(-valuesB[n]*valuesC[n-2]+valuesB[n-1]*valuesC[n-1])/float((valuesC[n-2])**2-valuesC[n-1]*valuesC[n-3]);
        # calculamos los errores
        er = abs(deltaR/(deltaR+r));
        es = abs(deltaS/(deltaS+s));
        # nuevos valores de r y s
        r = deltaR + r;
        s = deltaS + s;

        # con los nuevos valores podemos calcular el valor de la raíces
        # x^2-rx-s

        if er < eInicial and es < eInicial:
            contador = 0;
            # raices de la ecuación
            x1 = (-(-r)+sqrt((-r)**2-4*(-s)))/2;
            x2 = (-(-r)-sqrt((-r)**2-4*(-s)))/2;
            print(round(x1,1));
            print(round(x2,1));
            valuesA = valuesB[:-2];
            n = len(valuesA)-1;

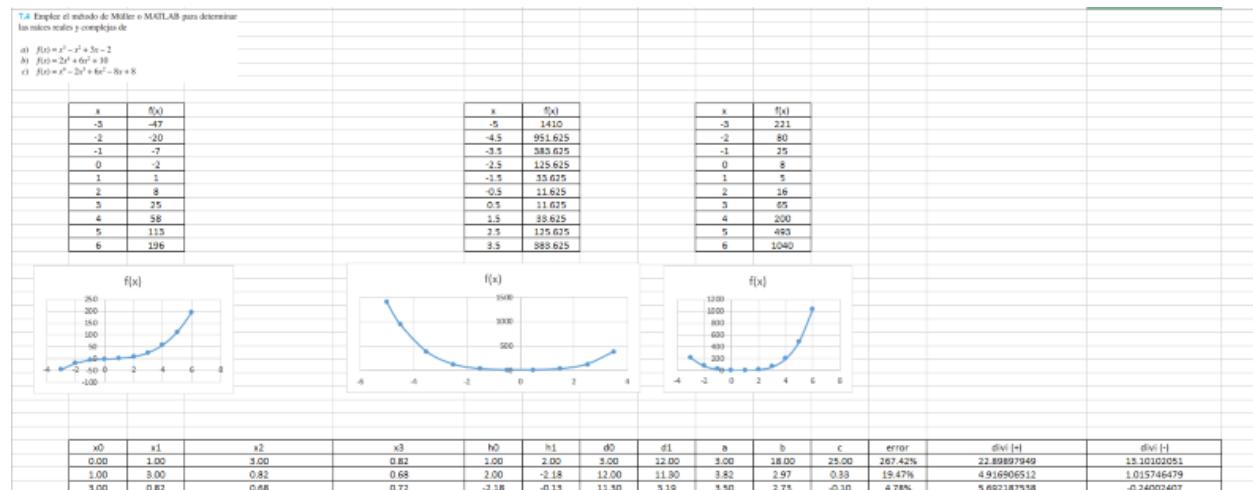
            # si es grado dos entonces calculamos las 2 últimas raíces y se hace un brak
            if n==2:
                print(round((-valuesA[1]+sqrt(valuesA[1]**2-4*valuesA[0]*valuesA[2]))/(2*valuesA[0]),1));
                print(round((-valuesA[1]-sqrt(valuesA[1]**2-4*valuesA[0]*valuesA[2]))/(2*valuesA[0]),1));
                break;
            elif n==1:
                print(round(-valuesA[1]/valuesA[0],1));
                break;
            # si es grado uno entonces calculamos la 1 última raíz
            # aumentar el contador
            contador = contador + 1;

    # reiniciar los valores de los arrays
    valuesB = [];
    valuesC = [];
metodoBeirtow();
```

7.6

El programa ha sido realizado con Python 3.9.8 y el algoritmo se encuentra implementado en el ejercicio 7.2.

7.7



7.8

El programa ha sido realizado con Python 3.9.8 y el algoritmo se encuentra implementado en el ejercicio 7.4.

7.9

a)

```
Función: -2+6.2*x-4*x**2+0.7*x**3
RAICES CON EL CÁLCULO BEIRSTOW
2.000000000000000
0.4
3.3
```

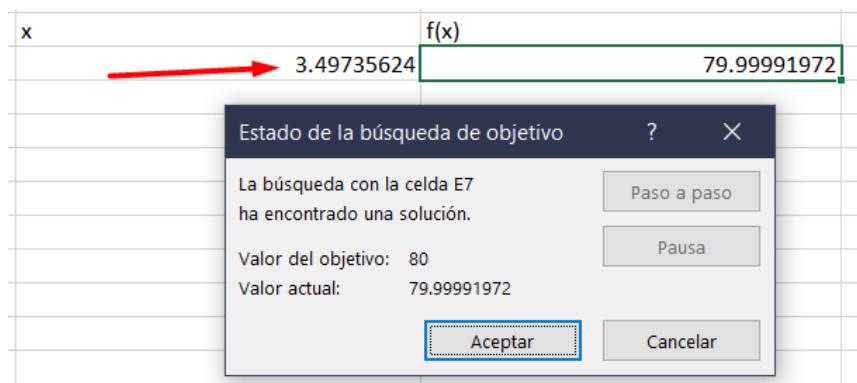
b)

```
Función: 9.34-21.97*x+16.3*x**2-3.704*x**3
RAICES CON EL CÁLCULO BEIRSTOW
1.2
1.000000000000000
2.3
```

c)

```
RAICES CON EL CÁLCULO BEIRSTOW
2.000000000000000
-1.000000000000000
1.0 + 2.0*I
1.0 - 2.0*I
```

7.10



Capítulo VIII

Estudio de casos: raíces de ecuaciones

8.1

```
# LIBRERIAS UTILIZADAS
from sympy import *

# ejercicios del cap 8
# ejercicio 1

def ejer1():
    v = Symbol("v")
    a = 12.02
    b = 0.08407
    n = 1
    R = 0.082054
    T = 400
    P = 2.5

    # error admitido
    ea = 0.001
    it = 1

    v0 = (n*R*T)/(P)
    fPrincipal = (P+(a/v**2))*(v-b)-R*T

    while(True):
        # función principal
        f1 = sympify(fPrincipal).subs(v,v0)

        # función principal derivada
        f2 = sympify(diff(fPrincipal),v).subs(v,v0)

        vr = v0 - (f1/f2)
        e = abs((vr-v0)/vr)*100

        v0 = vr

        print(f"it:{it} - valor: {vr} - error: {e}")

        if e < ea:
            break
        it = it + 1

ejer1()
```

```
it:1 - valor: 12.0009094560661 - error: 2.24073337576552
it:2 - valor: 12.0007278650101 - error: 0.00141418839465795
it:3 - valor: 12.0007278649345 - error: 5.88613106318072E-10
```

Demostración en excel

$v = \frac{RT}{P} = \frac{0.082054(400)}{2.5} = 13.12864$	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>I</th><th>XI</th><th>f(XI)</th><th>f(XI)</th><th>Ea</th></tr> </thead> <tbody> <tr> <td>0</td><td>13.12864</td><td>0.699518</td><td>2.431156</td><td></td></tr> <tr> <td>1</td><td>12.84091</td><td>0.000441</td><td>2.428057</td><td>2.2407%</td></tr> <tr> <td>2</td><td>12.84073</td><td>1.84E-10</td><td>2.428055</td><td>0.0014%</td></tr> <tr> <td>3</td><td>12.84073</td><td>0</td><td>2.428055</td><td>0.0000%</td></tr> </tbody> </table>	I	XI	f(XI)	f(XI)	Ea	0	13.12864	0.699518	2.431156		1	12.84091	0.000441	2.428057	2.2407%	2	12.84073	1.84E-10	2.428055	0.0014%	3	12.84073	0	2.428055	0.0000%
I	XI	f(XI)	f(XI)	Ea																						
0	13.12864	0.699518	2.431156																							
1	12.84091	0.000441	2.428057	2.2407%																						
2	12.84073	1.84E-10	2.428055	0.0014%																						
3	12.84073	0	2.428055	0.0000%																						
ecuación de van Waals:																										
Determine la raíz de																										
$f(v) = \left(2.5 + \frac{12.02}{v^2}\right)(v - 0.08407) - 0.082054(400)$																										
Cualquiera de las técnicas de los Capítulos 5 o 6 se puede utilizar para determinar la raíz como $v = 12.8407 \text{ L/mol}$. El método de Newton-Raphson sería una buena opción porque (a) la ecuación es relativamente simple de diferenciar y (b) la ley de los gases ideales proporciona una buena estimación inicial. El método de Newton-Raphson se puede formular como																										
El uso de la ley de los gases ideales para la estimación inicial da como resultado una determinación de la raíz precisa en unas pocas iteraciones:																										

Borrador del algoritmo

Notes on the derivation of the ideal gas law and its application to a real gas:

$$V = \left[\frac{4}{3} \cos^{-1} \left(\frac{R-h}{2} \right) - \left(R-h \sqrt{4h-h^2} \right)^2 \right] S$$

$$V = \left[4 \cos^{-1} \left(\frac{R-h}{2} \right) - 2 + h \sqrt{4h-h^2} \right] S$$

$$V = 20 \cos^{-1} \left(\frac{R-h}{2} \right) - 10 + 5h \sqrt{4h-h^2}$$

$$pV = nRT \rightarrow \text{primero gas}$$

$$m = 12 \text{ kg} = \frac{12000 \text{ g}}{2.56 \text{ g/mol}} = 298.15 \text{ mol}$$

$$m = \frac{M_m}{M_{\text{air}}}$$

$$P = 1 \text{ atm}$$

$$\textcircled{1} \text{ Determinar el volumen molar}$$

$$a = 12.02$$

$$b = 0.08403$$

$$T = 2400 \text{ K}$$

$$P = 1.8 \text{ atm}$$

$$R = 0.082044 \text{ L atm/molK}$$

$$V = \frac{M_m T}{P}$$

$$\frac{12.02}{x^2} = \frac{24.04 (x-0.08403)}{x^3}$$

$$12.02 = \frac{2.5 - 0.02 + 0.02404 (0.08403)}{x^3}$$

$$\frac{12.02}{x^2} = \frac{24.04 x + 1.9204 (0.08403)}{x^3}$$

$$\frac{12.02}{x^2} = \frac{24.04}{x^3} - \frac{24.04}{x^2}$$

$$\frac{-12.02}{x^2} + \frac{24.04 (0.08403)}{x^2}$$

$$= 3 + \frac{12.02}{x^2} - \frac{24.04 (0.08403)}{x^3}$$

$$100 + \frac{1.92}{(0.08403)^2} = \frac{2.52 (0.14 - 0.05)}{(0.08403)^2}$$

$$\frac{136}{0.0049} = \frac{0.11}{0.0049}$$

$$28 = 11$$

8.2

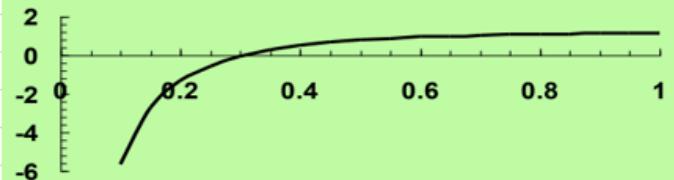
La función a resolver es

$$f(R) = \ln \frac{1+R(1-X_{Af})}{R(1-X_{Af})} - \frac{R+1}{R[1+R(1-X_{Af})]} = 0$$

o sustituyendo $X_{Af} = 0.95$

$$f(R) = \ln \frac{1+0.05R}{R(0.05)} - \frac{R+1}{R(1+0.05R)} = 0$$

Una gráfica de la función indica una raíz en aproximadamente $R = 0.3$



La bisección con estimaciones iniciales de 0.01 y 1 se puede usar para determinar una raíz de 0.30715 después 16 iteraciones con $\Delta a = 0,005\%$.

8.3

La función a resolver es:

$$f(x) = \frac{x}{1-x} \sqrt{\frac{7}{2+x}} - 0.04 = 0$$

Una gráfica de la función indica una raíz en aproximadamente $x = 0.02$.

Debido a que la función es tan lineal, la posición falsa es una buena opción. Usando estimaciones iniciales de 0.01 y 0.03, la primera iteración es

$$x_1 = 0.03 - \frac{0.017432(0.01-0.03)}{-0.02113-0.01743} = 0.020964$$

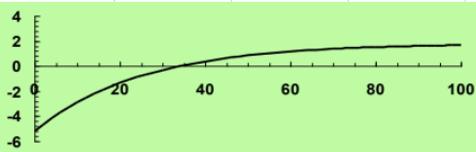


8.4

La función a resolver es:

$$f(t) = 12(1 - e^{-0.04t}) + 5e^{-0.04t} - 10.2 = 0$$

Una gráfica de la función indica una raíz en aproximadamente $t = 34$



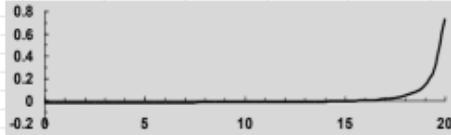
La bisección con estimaciones iniciales de 0 y 50 se puede usar para determinar una raíz de 33.95309 después de 16 iteraciones con $\text{Ra} = 0,002\%$.

8.5

La función a resolver es:

$$f(x) = \frac{4+x}{(42-2x)^2(28-x)} - 0.016 = 0$$

a) Una gráfica de la función indica una raíz en aproximadamente $x = 16$.



a) La forma de la función indica que la posición falsa sería una mala elección (recuerde la Fig. 5.14). La bisección con estimaciones iniciales de 0 y 20 se puede utilizar para determinar una raíz de 15.85938 después de 8 iteraciones con $\text{Ra} = 0.493\%$. Tenga en cuenta que la posición falsa habría requerido 68 iteraciones para lograr una precisión comparable.

i	x_l	x_u	x_c	$f(x_l)$	$f(x_u)$	$f(x_c)$	$ x_c - x_{l,u} $	Ra
1	0	20	10	-0.01592	-0.01439	0.000229	100.000%	
2	10	20	15	-0.01439	-0.00585	8.42E-05	33.333%	
3	15	20	17.5	-0.00585	0.025789	-0.00015	14.286%	
4	15	17.5	16.25	-0.00585	0.003096	-1.8E-05	7.692%	
5	15	16.25	15.625	-0.00585	-0.00228	1.33E-05	4.000%	
6	15.625	16.25	15.9375	-0.00228	0.000123	-2.8E-07	1.961%	
7	15.625	15.9375	15.78125	-0.000123	-0.00114	2.59E-06	0.990%	
8	15.78125	15.9375	15.85938	-0.00114	-0.00052	5.98E-07	0.493%	

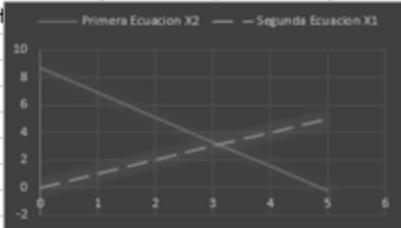
8.6

La función a resolver es:	$K_1 = \frac{(c_{c,0} + x_1 + x_2)}{(c_{a,0} - 2x_1 - x_2)^2(c_{b,0} - x_1)}$
	$K_2 = \frac{(c_{c,0} + x_1 + x_2)}{(c_{a,0} - 2x_1 - x_2)(c_{d,0} - x_2)}$
	$f_1(x_1, x_2) = \frac{5 + x_1 + x_2}{(50 - 2x_1 - x_2)^2(20 - x_1)} - 4 \times 10^{-4}$
	$f_2(x_1, x_2) = \frac{5 + x_1 + x_2}{(50 - 2x_1 - x_2)(10 - x_1)} - 3.7 \times 10^{-2}$

Los gráficos se pueden generar especificando valores de x_1 y resolviendo para x_2 usando un valor numérico método como bisección.

Primera Ecuación		Segunda Ecuación	
X1	X2	X1	X2
0	8.6672	0	4.4167
1	6.8618	1	3.9187
2	5.0649	2	3.4010
3	3.2709	3	2.8630
4	1.4984	4	2.3038
5	-0.2700	5	1.7227

Estos valores se pueden graficar:



Por lo tanto, la raíz parece estar aproximadamente en $x_1 = 3.3$ y $x_2 = 2.7$. Empleando estos valores como conjeturas iniciales para el método de Newton-Raphson de dos variables da

$$f_1(3.3, 2.7) = -2.36 \times 10^{-6}$$

$$f_2(3.3, 2.7) = 2.33 \times 10^{-5}$$

$$\frac{\partial f_1}{\partial x_1} = 9.9 \times 10^{-5}$$

$$\frac{\partial f_2}{\partial x_1} = 5.185 \times 10^{-3}$$

$$\frac{\partial f_1}{\partial x_2} = 5.57 \times 10^{-5}$$

$$\frac{\partial f_2}{\partial x_2} = 9.35 \times 10^{-3}$$

$$|J| = 6.37 \times 10^{-7}$$

$$x_1 = 3.3 - \frac{-2.36 \times 10^{-6}(9.35 \times 10^{-3}) - 2.33 \times 10^{-5}(5.57 \times 10^{-5})}{6.37 \times 10^{-7}} = 3.3367$$

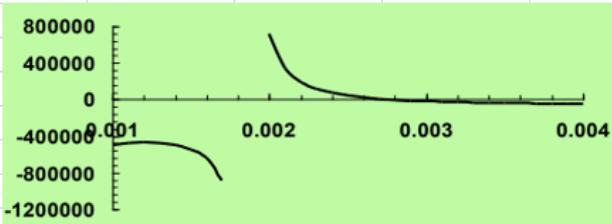
$$x_2 = 2.7 - \frac{2.33 \times 10^{-5}(9.9 \times 10^{-5}) - (-2.36 \times 10^{-6})(5.185 \times 10^{-3})}{6.37 \times 10^{-7}} = 2.677$$

La segunda iteración produce $x_1 = 3.3366$ y $x_2 = 2.677$, con un error máximo aproximado de 0,003%.

Usando los valores dados, $a = 12.6126$ y $b = 0.0018707$. Por lo tanto, el problema de las raíces es resuelto es:

$$f(v) = \frac{0.518(223)}{(v - 0.0018707)v(v + 0.0018708)\sqrt{223}} - 6500$$

Un gráfico indica una raíz en aproximadamente 0,0028.



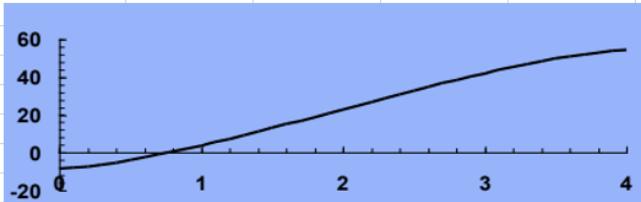
Usando estimaciones iniciales de 0.002 y 0.004, se puede emplear la bisección para determinar la raíz como 0.00275 después de 12 iteraciones con $\Delta a = 0.018\%$. La masa de metano contenida en el tanque puede calcularse como $3 / 0.00275 = 1091 \text{ kg}$

8.8

Usando los valores dados, el problema de raíces a resolver es:

$$f(h) = \left[4 \cos^{-1}\left(\frac{2-h}{2}\right) - (2-h)\sqrt{4n-h^2} \right] 5 - 8.5 = 0$$

Un gráfico indica una raíz en aproximadamente 0,8.



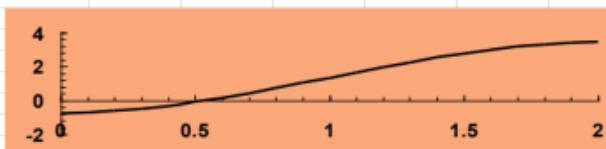
Se puede utilizar un método numérico para determinar que la raíz es 0,77194.

8.9

Usando los valores dados, el problema de raíces a resolver es

$$f(h) = \frac{\pi h^2(3-h)}{3} - 0.75 = 0$$

Un gráfico indica una raíz en aproximadamente 0,52



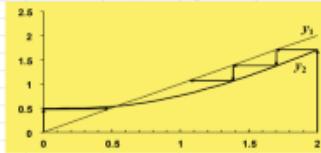
Se puede usar un método numérico para determinar que la raíz es 0.53952

8.10

La mejor manera de abordar este problema es utilizar el método gráfico que se muestra en la figura 6.3. Para la primera versión, trazamos

$$y_1 = h \quad y \quad y_2 = \frac{\sqrt{h^3 + 0.7162}}{3}$$

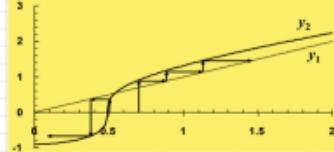
versus el rango de h . Tenga en cuenta que para la esfera, h varía de 0 a $2r$. Como se muestra a continuación, esta versión siempre convergerá.



Para la segunda versión, trazamos

$$y_1 = h \quad y \quad y_2 = 3\sqrt{3h^2 - 0.7162}$$

versus el rango de h . Como se muestra a continuación, esta versión no es convergente



8.11

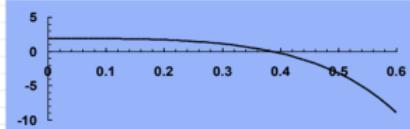
Sustituyendo los valores de los parámetros se obtiene

$$\frac{20}{1-\varepsilon} = 150 \frac{1-\varepsilon}{1000} + 1.75$$

Esto se puede reorganizar y expresar como un problema de raíz:

$$f(\varepsilon) = 0.15(1-\varepsilon) + 1.75 - 20 \frac{\varepsilon^3}{1-\varepsilon} = 0$$

Un gráfico de la función sugiere una raíz en aproximadamente 0,38.



Pero supongamos que no tenemos una trama. ¿Cómo llegamos a una buena suposición inicial? La fracción de vacíos (la fracción del volumen que no es sólido; es decir, consta de vacíos) varía entre 0 y 1. Como puede verse, un valor de 1 (que es físicamente irreal) provoca una división por cero. Por lo tanto, se pueden elegir dos conjecturas iniciales basadas en la física como 0 y 0,99.

Tenga en cuenta que el cero tampoco es físicamente realista, pero dado que no causa ninguna dificultad matemática, está bien. Al aplicar la biseción se obtiene un resultado de $\varepsilon = 0.384211$ en 15 iteraciones con un error relativo absoluto aproximado de $7.87 \times 10^{-3}\%$

8.12

El número de Reynolds se puede calcular como

$$Re = \frac{\rho V D}{\mu} = \frac{1.23(40)0.005}{1.79 \times 10^{-5}} = 13743$$

Para encontrar f , debemos determinar la raíz de la función $g(f)$

$$g(f) = -2.0 \log \left(\frac{0.0000015}{3.7(0.005)} + \frac{2.51}{13743\sqrt{f}} \right) - \frac{1}{\sqrt{f}} = 0$$

Como se mencionó en el problema, se puede obtener una buena suposición inicial a partir de la fórmula de Blasius

$$f = \frac{0.316}{13743^{0.25}} = 0.029185$$

Con esta suposición, se puede obtener una raíz de 0.028968 con un enfoque como el método de la secante modificado. Este resultado se puede utilizar para calcular la caída de presión como

$$\Delta p = 0.28968 \frac{0.2(1.23)(40)^2}{2(0.005)} = 1140.17$$

- a) Repita el cálculo pero para un tubo de acero comercial más rugoso ($\varepsilon = 0.045$ mm). Para la tubería de acero más rugosa, debemos determinar la raíz de

$$g(f) = -2.0 \log \left(\frac{0.000045}{3.7(0.005)} + \frac{2.51}{13743\sqrt{f}} \right) - \frac{1}{\sqrt{f}} = 0$$

Usando la misma estimación inicial que en (a), se puede obtener una raíz de 0.04076. Este resultado se puede utilizar para calcular la caída de presión como

$$\Delta p = 0.04076 \frac{0.2(1.23)(40)^2}{2(0.005)} = 1604.25$$

Por lo tanto, como era de esperar, la caída de presión es mayor para la tubería más rugosa.

8.13

Una forma es combinar las ecuaciones para producir un solo polinomio. Las ecuaciones 1 y 2 se pueden resolver para:

$$[CO_2] = \frac{M^+ [HCO_3^-]}{K_1} \quad [CO_3^{2-}] = \frac{K_2 [CO_3^{2-}]}{M^+}$$

Estos resultados se pueden sustituir en la ecuación 4, que se puede resolver para:

$$[CO_2] = F_0 c_p \quad [HCO_3^-] = F_1 c_p \quad [CO_3^{2-}] = F_2 c_p$$

donde F_0 , F_1 y F_2 son las fracciones del carbono inorgánico total en dióxido de carbono, bicarbonato y carbonato, respectivamente, donde:

$$F_0 = \frac{M^+}{[H^+]^2 + K_1[M^+] + K_1K_2} \quad F_1 = \frac{K_1[M^+]}{[H^+]^2 + K_1[M^+] + K_1K_2}$$

$$F_2 = \frac{K_2}{[H^+]^2 + K_1[H^+] + K_1K_2}$$

Ahora bien, estas ecuaciones, junto con la Eq. 3 se puede sustituir en la ecuación 5 para dar

$$0 = F_1 c_p + 2F_2 c_p + K_w/[H^+] - [H^+] - Alk$$

Aunque puede que no sea evidente, este resultado es un polinomio de cuarto orden en $[H^+]$

$$[H^+]^4 + (K_1 + Alk) [H^+]^3 + (K_1 K_2 + Alk K_1 - K_w - K_1 c_p) [H^+]^2 + (Alk K_1 K_2 - K_2 - K_w - 2K_1 K_2 c_p) [H^+] - K_1 K_2 K_w = 0$$

La sustitución de los valores de los parámetros da

$$[H^+]^4 + 2.001 \times 10^{-3} [H^+]^3 - 2.012 \times 10^{-19} [H^+]^2 - 1.255 \times 10^{-19} [H^+] - 2.512 \times 10^{-31} = 0$$

Esta ecuación se puede resolver para $[H^+] = 2.51 \times 10^{-19}$ ($\text{pH}=6.6$). Este valor se puede utilizar para calcular

$$[OH^-] = \frac{10^{-14}}{2.51 \times 10^{-19}} = 3.98 \times 10^{-5}$$

$$[CO_2] = \frac{(2.51 \times 10^{-7})^2 + 10^{-4} \times (2.51 \times 10^{-7}) + 10^{-4} \times 10^{-23} \times 3 \times 10^{-2}}{0.33304(3 \times 10^{-3})} = 0.001$$

$$[HCO_3^-] = \frac{10^{-6} \times (2.51 \times 10^{-7})^2}{(2.51 \times 10^{-7})^2 + 10^{-4} \times (2.51 \times 10^{-7}) + 10^{-4} \times 10^{-23} \times 3 \times 10^{-2}} = 0.666562(3 \times 10^{-3}) = 0.002$$

$$[CO_3^{2-}] = \frac{10^{-8} \times 10^{-23} \times 3 \times 10^{-2}}{(2.51 \times 10^{-7})^2 + 10^{-6} \times (2.51 \times 10^{-7}) + 10^{-6} \times 10^{-23} \times 3 \times 10^{-2}} = 0.000133(3 \times 10^{-3}) = 4 \times 10^{-7} M$$

8.14

Este problema utilizamos el método solve que viene integrado Excel para calcular el objetivo deseado.

La integral se puede evaluar como

$$-\int_{C_{in}}^{C_{out}} \frac{K}{k_{max} C} + \frac{1}{k_{max}} dC = -\frac{1}{k_{max}} \left[K \ln\left(\frac{C_{out}}{C_{in}}\right) + C_{out} - C_{in} \right]$$

Por lo tanto, el problema consiste en encontrar la raíz de

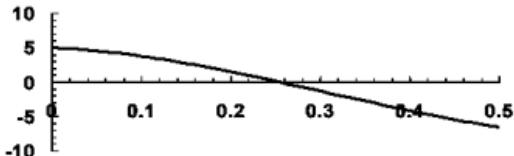
$$f(C_{out}) = \frac{V}{F} + \frac{1}{k_{max}} \left[K \ln\left(\frac{C_{out}}{C_{in}}\right) + C_{out} - C_{in} \right]$$

8.15

La función a resolver es

$$f(t) = 9e^{-0.7t} \cos(4t) - 3.5$$

Una gráfica de la función indica una raíz en aproximadamente $t = 0.25$



- a) Emplee el método de Newton-Raphson para determinar la raíz con $\epsilon_s = 0.01\%$.

El método de Newton-Raphson se puede configurar como

$$t_{i+1} = t_i - \frac{9e^{-0.7t_i} \cos(4t_i) - 3.5}{-36e^{-0.7t_i} \sin(4t_i) - 6.3 \cos(4t_i)e^{-0.7t_i}}$$

Usando una estimación inicial de 0.3

i	t	f(t)	f'(t)	Ea
0	0.3	-0.85651	-29.0483	
1	0.270514	-0.00335	-28.7496	10.899824%
2	0.270398	-1.20E-07	-28.7476	0.043136%
3	0.270398	0	-28.7476	0.000002%

- a) Use el método de la secante para determinar la raíz con $\epsilon_s = 0.01\%$.

El método de la secante se puede implementar con estimaciones iniciales de 0.3

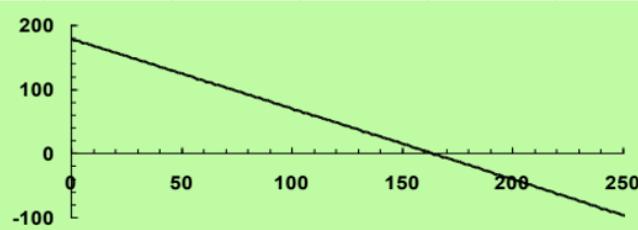
i	t_{i-1}	$f(t_{i-1})$	t_i	$f'(t_i)$	Ea
0	0.2	1.951189	4	-3.69862	
1	0.4	-3.69862	0.269071	0.038125	48.66%
2	0.269071	0.038125	0.270407	-0.00026	0.49%
3	0.270407	0.00026	0.270398	1.07E-07	0.0034%

8.16

La función a resolver es

$$f(P/A) = \frac{250}{1 + 0.4 / \cos[251\sqrt{(P/A)/200000}]} - \frac{P}{A}$$

Un gráfico de la función indica una raíz en aproximadamente $P/A = 163$.



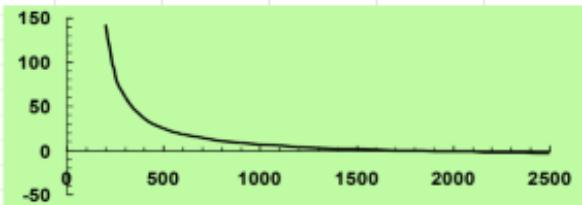
Se puede utilizar un método numérico para determinar que la raíz es 163,4429.

8.17

La función a resolver es

$$f(T_A) = \frac{T_A}{12} \cos h\left(\frac{600}{T_A}\right) + 6 - \frac{T_A}{12} - 15$$

Un gráfico de la función indica una raíz en aproximadamente $T_A = 1700$.



Se puede utilizar un método numérico para determinar que la raíz es 1684,365

8.18

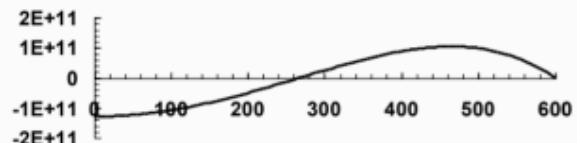
Este problema se puede resolver determinando la raíz de la derivada de la curva elástica

$$\frac{dy}{dx} = 0 = \frac{W_0}{120EI L} (-5x^4 + 6L^2x^2 - L^4)$$

$$f(x) = -5x^4 + 2,160,000x^2 - 1.296 \times 10^{11} = 0$$

Por lo tanto, después de sustituir los valores de los parámetros, debemos determinar la raíz de

Una gráfica de la función indica una raíz en aproximadamente $x = 270$



$$y = \frac{2.5}{120(50,000)(30,000)(600)} (-(-268.328)^5 + 720,000(-268.328)^3 - 1.296 \times 10^{11}(-268.328)) = -0.51519$$

Capítulo IX

Eliminación de Gauss

1.1

a)

$$\left[\begin{array}{ccc|c} 0 & 2 & 5 & x \\ 1 & 0 & 1 & 1 \\ 8 & 3 & 0 & \{ 3 \} \end{array} \right] \quad \begin{array}{l} x \\ 1 \\ x \\ 2 \\ x \\ 3 \end{array} = \left\{ \begin{array}{l} 50 \\ 10 \\ 20 \end{array} \right\}$$

b)

$$[A]^2 = \left[\begin{array}{ccc} 0 & 1 & 8 \\ 2 & 0 & 3 \\ 5 & 1 & 0 \end{array} \right]$$

1.2

a)

$$\begin{array}{llll} [A] = 3x2 & [B] = 3x3 & [C] = 3x1 & [D] = 2x4 \\ [E] = 3x3 & [F] = 2x3 & [G] = 1x3 & \end{array}$$

b)

$$\begin{array}{ll} \text{cuadrado:} & [B] \text{ y } [E] \\ \text{columna:} & [C] \\ \text{fila:} & [G] \end{array}$$

c)

$$\begin{array}{lll} a_{12} = 7 & b_{23} = 7 & d_{32} = \text{no existe} \\ \\ e_{22} = 2 & f_{12} = 0 & g_{12} = 6 \end{array}$$

d)

$$(1) [E] + [B] = \begin{bmatrix} 5 & 8 & 15 \\ 8 & 4 & 10 \\ 6 & 0 & 10 \end{bmatrix}$$

(2) $[A] + [F] =$ no es posible

$$(3) [B] - [E] = \begin{bmatrix} 3 & -2 & -1 \\ -6 & 0 & 4 \\ -2 & 0 & -2 \end{bmatrix}$$

$$(4) 7[B] = \begin{bmatrix} 28 & 21 & 49 \\ 7 & 14 & 49 \\ 14 & 0 & 28 \end{bmatrix}$$

$$\begin{bmatrix} 25 & 13 & 74 \\ 36 & 25 & 75 \\ 28 & 12 & 52 \end{bmatrix}$$

(5) $[E] \times [B] =$

(6) $|c| = [3 \ 6 \ 1]$

$$(7) [B] \times [A] = \begin{bmatrix} 5 \\ 4 & 76 \\ 4 \\ 9 & 12 & 53 \\ 4 & 2 & 38 \\ 3 & 8 & 38 \\ -6 & 5 \end{bmatrix}$$

(8) $|D|^T =$

(9) $[A] \times [C] =$ no es posible

(10) $[I] \times [B] = [B]$

$$(11) |E|^T \times [E] = \begin{bmatrix} 66 & 19 & 53 \\ 19 & 29 & 46 \\ 10 & 53 & 46 \\ 9 \end{bmatrix}$$

(12) $[C]^T \times [C] = 46$

9.3

a)

$$[A][B] = \begin{vmatrix} 4 & 15 \\ 8 & 29 \\ 9 & 29 \end{vmatrix}$$

$$[A] [C] = \begin{vmatrix} -16 & 4 \\ -24 & 4 \\ 2 & -10 \end{vmatrix}$$

$$[B] [C] = \begin{vmatrix} -7 & 1 \\ -5 & 1 \end{vmatrix}$$

$$[C] [B] = \begin{vmatrix} 1 & 2 \\ -25 & -7 \end{vmatrix}$$

$$[B] [B] = \begin{vmatrix} 25 & 9 \\ 1.5 & 5.5 \end{vmatrix}$$

$$[C] [C] = \begin{vmatrix} 10 & -6 \\ 10 & -6 \end{vmatrix}$$

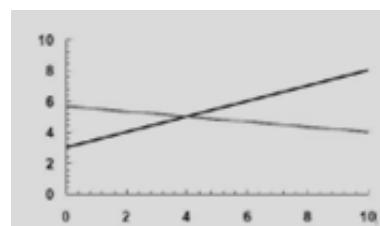
b)

[B] [A] y [C] [A] son imposibles porque las dimensiones internas no coinciden:
 $(2x2) * (3x2)$

c)

De acuerdo con (a), $[B] [C] \neq [C] [B]$

9.4

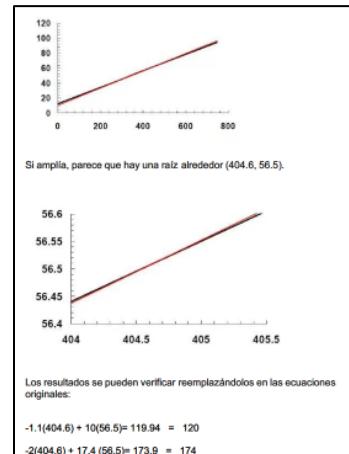


$$4(4)-8(5)=-24$$

$$4+8(5)=34$$

9.5

a)



b)

La trama sugiere que el sistema puede estar mal acondicionado porque las pendientes son muy similares.

c)

$$D = -1.1(17.4) - 10(-2) = 0.86$$

$$D = -0.11(1) - 1(-0.11494) = 0.00494$$

d)

$$x_1 = \frac{17.4(120) - 10(174)}{0.86} = 404.6512$$

$$x_2 = \frac{-1.1(174) - (-2)(120)}{0.86} = 56.51163$$

9.6

a)

$$A_1 = \begin{vmatrix} 1 & 1 \\ 1 & 0 \end{vmatrix} = 1(0) - 1(0) = -1$$

$$A_1 = \begin{vmatrix} 2 & 1 \\ 3 & 0 \end{vmatrix} = 2(0) - 1(3) = -3$$

$$A_1 = \begin{vmatrix} 2 & 1 \\ 3 & 1 \end{vmatrix} = 2(1) - 1(3) = -1$$

$$D = 0(-1) - 2(-3) + 5(-1) = 1$$

b)

$$x_1 = \frac{\begin{vmatrix} 0 & 9 & 5 \\ 2 & 9 & 1 \\ 3 & 3 & 0 \end{vmatrix}}{D} = \frac{-6}{1} = -6$$

$$x_2 = \frac{\begin{vmatrix} 0 & 9 & 5 \\ 2 & 9 & 1 \\ 3 & 3 & 0 \end{vmatrix}}{D} = \frac{-8}{1} = -8$$

$$x_3 = \frac{\begin{vmatrix} 0 & 9 & 5 \\ 2 & 9 & 1 \\ 3 & 3 & 0 \end{vmatrix}}{D} = \frac{5}{1} = 5$$

c)

$$2(-8) + 5(5) = 9$$

$$2(6) + (-8) + 5 = 9$$

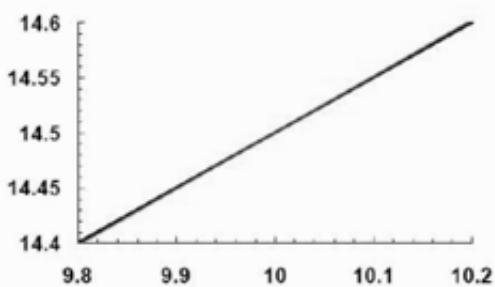
$$3(6) + (-8) = 10$$

9.7

a)

$$x_2 = 9.5 + 0.5x_1$$

$$x_2 = 9.4 + 0.5x_1$$



La solución es $x_1 = 10$, $x_2 = 14.5$. Tenga en cuenta que las líneas tienen pendientes muy similares.

b)

$$D = 0.5(-2) - (-1)1.02 = 0.02$$

c)

La trama y el bajo valor del determinante sugieren que el sistema no está condicionado.

d)

$$x_1 = \frac{-9.5(-2) - (-1)(-18.8)}{0.02} = 10$$

$$x_2 = \frac{0.5(-18.8) - (-9.5)1.02}{0.02} = 14.5$$

e)

$$x_1 = \frac{-9.5(-2) - (-1)(-18.8)}{-0.02} = -10$$

$$x_1 = \frac{0.52(-18.8) - (-9.5)1.02}{-0.02} = 4.3$$

9.8

a)

$$\begin{array}{ccc|c} 10 & 2 & -1 & 27 \\ -3 & -6 & 2 & 61.5 \\ 1 & 1 & 5 & 21.5 \end{array}$$

$$\begin{array}{ccc|c} 10 & 2 & -1 & 27 \\ 0 & 5.4 & 1.7 & -53 \\ 0 & 0 & 5.35 & 32.11 \end{array}$$

Sustitución posterior:

$$x_3 = \frac{-32.1111}{5.35185} = -6$$

$$x_2 = \frac{-53.4 - 1.7(-6)}{-5.4} = 8$$

$$x_1 = \frac{27 - (-1)(-6) - 2(8)}{10} = 0.5$$

b)

$$10(0.5) + 2(8) - (-6) = 27$$
$$-3(0.5) - 6(8) + 2(-6) = -61.5$$
$$0.5 + (8) + 5(-6) = -21.5$$

9.9

El sistema se expresa primero como una matriz aumentada

$$\begin{array}{ccc|c} 10 & 2 & 2 & 27 \\ 12 & 2 & 2 & 6 \end{array}$$

$$\begin{array}{ccc|c} 12 & 2 & 2 & 6 \\ 10 & 2 & 4 & 4 \\ 8 & 2 & -2 & -2 \end{array}$$

$$\begin{array}{ccc|c} 12 & 2 & 2 & 6 \\ 0 & 0.3 & 2.3 & -1 \\ 0 & 0.7 & -3.3 & -6 \end{array}$$

Pivot

$$\begin{array}{ccc|c} 12 & 2 & 2 & 6 \\ 0 & 0.66 & -3.3 & -6 \\ 0 & 0.33 & 2.3 & -1 \end{array}$$

$$\begin{array}{ccc|c} 12 & 2 & 2 & 6 \\ 0 & 0.66 & -3.33 & -6 \\ 0 & 0 & 4 & 2 \end{array}$$

$$x_3 = \frac{2}{4} = 0.5$$

$$x_2 = \frac{-6 - (-3.33)0.5}{0.66667} = -6.5$$

$$x_1 = \frac{6 - 2(0.5) - 2(-6.5)}{12} = 1.5$$

$$8(1.5) + 2(-6.5) - 2(0.5) = -2$$

$$10(1.5) + 2(-6.5) + 4(0.5) = 4$$

$$12(1.5) + 2(-6.5) + 2(0.5) = 6$$

9.10

$$\left| \begin{array}{cc} 2 & -1 \\ -2 & 0 \end{array} \right| = 2(0) - (-1)(-2) = -2$$

$$A2 = \left| \begin{array}{cc} 1 & -1 \\ 5 & 0 \end{array} \right| = 1(0) - (-1)(5) = 5$$

$$A3 = \left| \begin{array}{cc} 1 & 2 \\ 5 & -2 \end{array} \right| = 1(-2) - 2(5) = -12$$

$$D = 0(-2) - (-3)5 + 7(-12) = -69$$

b)

$$X_1 = \frac{\begin{vmatrix} 2 & -3 & 7 \\ 3 & 2 & -1 \\ 2 & -2 & 0 \end{vmatrix}}{D} = \frac{-68}{-69} = \frac{0.9855}{0.07}$$

$$X_2 = \frac{\begin{vmatrix} 0 & 2 & 7 \\ 1 & 3 & -1 \\ 5 & 2 & 0 \end{vmatrix}}{D} = \frac{-101}{-69} = \frac{1.46376}{8}$$

$$X_3 = \frac{\begin{vmatrix} 0 & -3 & 2 \\ 1 & 2 & 3 \\ 5 & -2 & 2 \end{vmatrix}}{D} = \frac{63}{-69} = \frac{0.9130}{-0.43}$$

c)

$$\left[\begin{array}{ccc|c} 0 & -3 & 7 & 2 \\ 1 & 2 & 1 & 3 \\ 5 & -2 & 0 & 2 \end{array} \right]$$

$$\left[\begin{array}{ccc|c} 0 & -2 & 0 & 2 \\ 1 & 2 & -1 & 3 \\ 0 & -3 & 7 & 2 \end{array} \right]$$

$$\left[\begin{array}{ccc|c} 5 & -2 & 0 & 2 \\ 0 & 2.4 & -1 & 2.6 \\ 0 & -3 & 7 & 2 \end{array} \right]$$

Pivot

$$\left[\begin{array}{ccc|c} 5 & -2 & 0 & 2 \\ 0 & -3 & 7 & 2 \\ 0 & 2.4 & -1 & 2.6 \end{array} \right]$$

$$\left[\begin{array}{ccc|c} 5 & -2 & 0 & 2 \\ 0 & -3 & 7 & 2 \\ 0 & 0 & 4.6 & 4.2 \end{array} \right]$$

$$x_3 = \frac{4.2}{4.6} = 0.913$$

$$x_2 = \frac{2.7(0.913)}{-3} = \frac{1.46}{3}$$

$$x_1 = \frac{2 - 0(0.913) - 2(1.4637)}{5} = 0.9855$$

d)

$$\begin{aligned}-3(1.4637) + 7(0.9130) &= 2 \\ (0.985507) + 2(1.463768) - (0.913043) &= 3 \\ 5(0.985507) - 2(1.463768) &= 2\end{aligned}$$

9.11

a)

$$\left[\begin{array}{ccc|c} 2 & -6 & -1 & -38 \\ -3 & -1 & 7 & -34 \\ -8 & 1 & -2 & -20 \end{array} \right]$$

$$\left[\begin{array}{ccc|c} -8 & 1 & -2 & -20 \\ -3 & -1 & 7 & -34 \\ 2 & -6 & -1 & -38 \end{array} \right]$$

$$\left[\begin{array}{ccc|c} -8 & 1 & -2 & -20 \\ 0 & -1.375 & 7.75 & -26.5 \\ 0 & -5.75 & -1.5 & -43 \end{array} \right]$$

Pivot:

$$\left[\begin{array}{ccc|c} -8 & 1 & -2 & -20 \\ 0 & -5.75 & -1.5 & -43 \\ 0 & 0 & 8.1086 & -16.21 \end{array} \right]$$

$$x_3 = \frac{-16.2174}{8.1086} = -2$$

$$x_2 = \frac{-43 - (-1.5)(-2)}{-5.75} = 8$$

$$x_1 = \frac{-20 + 2(-2) - 1(8)}{-} = 4$$

b)

$$\begin{aligned}2(4) - 6(8) - (-2) &= -38 \\ -3(4) - (8) + 7(-2) &= -34 \\ -8(4) + (8) - 2(-2) &= -20\end{aligned}$$

Capítulo X

Descomposición LU e inversa de matrices

1.1

```
# ejercicio 1
def ejer1():
    # Utilice las reglas de la multiplicación de matrices
    # para
    # demostrar que las ecuaciones (10.7) y (10.8) se obtienen
    # de la
    # (10.6).
    m1 = [
        [2,-3,5],
        [6,-1,3],
        [-4,1,-2]
    ]
    # m1 = [
    #     [3,-0.1,-0.2],
    #     [0.1,7,-0.3],
    #     [0.3,-0.2,10]
    # ]

    # sirve para la creación de una matriz que se adecue a L
def createMatriz(size):
    aux = []
    for i in range(size):
        a = [];
        for j in range(size):
            a.append(0)
        aux.append(a);
    return aux;
L = createMatriz(len(m1));

    # 1. calcular la u con el método de gaus
    # 2. rellenar los valores de L a la vez
```

```

# metodo de gaus
n = len(m1);

for j in range(n):
    for i in range(n):
        if i == j:
            pMatriz = m1[i]
            pivo = m1[i][j]
            # matriz L
            L[i][j] = 1;
        if i>j:
            L[i][j] = m1[i][j]/pivo
            newAList = list(map(lambda x: x*(m1[i][j]/pivo) ,
pMatriz))
            m1[i] = list(map(lambda x,y: x-y ,
m1[i],newAList));

# imprimir la matriz
print("EL VALOR DE L")
for i in range(n):
    print(L[i]);
print("EL VALOR DE U")
for i in range(n):
    print(m1[i]);

# el orden de la multiplicación es importante
# m1 * L != L * m1
print("MULTIPLICACIÓN")
multi = np.dot(L,m1)
print(multi)

# ejercicio 2
def ejer2():
    # m1 = [

```

```

#    [10,2,-1,27],
#    [-3,-6,2,-61.5],
#    [1,1,-5,-21.5]
# ]
m1 = [
[1,2,1,8],
[-1,3,-2,1],
[3,4,-7,10]
]
def createMatriz(size):
aux = []
for i in range(size):
a = [];
for j in range(size):
a.append(0)
aux.append(a);
return aux;

L = createMatriz(len(m1));

n = len(m1);

for j in range(n):
for i in range(n):
if i == j:
pMatriz = m1[i]
pivo = m1[i][j]
# matriz L
L[i][j] = 1;
if i>j:
L[i][j] = m1[i][j]/pivo
newAList = list(map(lambda x: x*(m1[i][j]/pivo) ,
pMatriz))
m1[i] = list(map(lambda x,y: x-y ,
m1[i],newAList));

```

```

print("EL VALOR DE U")
for i in range(n):
    print(m1[i]);

m1[::-1]

valEcuaciones = []
for i in reversed(range(n)):
    sum = m1[i][n];
    for j in reversed(range(i+1,n)):
        sum = sum - m1[i][j]*valEcuaciones[n-j-1];
    valEcuaciones = [*valEcuaciones, sum/m1[i][i]]

# insertar los valores en L
for i in range(len(valEcuaciones)):
    L[i].append(valEcuaciones[i])

print("EL VALOR DE L")
for i in range(n):
    print(L[i]);

valEcuaciones = []
for i in reversed(range(n)):
    sum = L[i][n];
    for j in reversed(range(i+1,n)):
        sum = sum - L[i][j]*valEcuaciones[n-j-1];
    valEcuaciones = [*valEcuaciones, sum/L[i][i]]
print("RESULTADO")
print(valEcuaciones)

```

1.3

```
# ejercicio 3
def ejer3():
    # descomposición LU sin pivoteo parcial
    def ejerA():

        # FUNCTIONS
        # Crear una matriz compuesta de ceros
        def createMatriz(size):
            # Se podría utilizar la librería numpy y evitar esta
            function
            # L = np.zeros(n,n)
            aux = []
            for i in range(size):
                a = [];
                for j in range(size):
                    if i == j:
                        a.append(1)
                    else:
                        a.append(0)
                aux.append(a);
            return aux;
        # calculate values of a matrix lower
        def calculateValuesLower(uniqueMatriz):
            valEcuaciones = []

            for i in range(n):
                sum = uniqueMatriz[i][n]
                for j in range(i):
                    sum = sum - uniqueMatriz[i][j] *
            valEcuaciones[j];
                valEcuaciones.append(sum/uniqueMatriz[i][i]);

            return valEcuaciones;
```

```

# calculate values of a matriz upper
def calculateValuesUpper(uniqueMatriz):
    valEcuaciones = []
    for i in reversed(range(n)):
        sum = uniqueMatriz[i][n];
        for j in reversed(range(i+1,n)):
            sum = sum - uniqueMatriz[i][j]*valEcuaciones[n-j-1];
        valEcuaciones = [*valEcuaciones,
sum/uniqueMatriz[i][i]]
    return valEcuaciones

# declaration of variables

A = [
    [1,2,1],
    [-1,3,-2],
    [3,4,-7]
]
A = [
    [0.5,-2.7,0,0],
    [1,-0.2,-3,0],
    [0,4.9,-7.8,4],
    [0, 0,2.8,-3.4]
]
equationsValues = [
    -5.925,-1.35,4.975,-0.1
]
n = len(A)
L = createMatriz(len(A))

# verificar que la matriz es cuadrada
if len(A) != len(A[0]):
    return print("La matriz no es una cuadrada")

# sin pivoteo (sin intercambio de filas)

```

```

for j in range(len(A)):
    for i in range(len(A)):
        if i == j:
            pMatriz = A[i]
            pivo = A[i][j]
        if i > j:
            L[i][j] = A[i][j]/pivo
            newAList = list(map(lambda x: x*(A[i][j]/pivo) ,
pMatriz))
            A[i] = list(map(lambda x,y: x-y ,
A[i],newAList));

# insertar los values de la equation en L
for i in range(len(equationsValues)):
    L[i].append(equationsValues[i])

# news values
equationsValues = calculateValuesLower(L);

# insert values in U
for i in range(len(equationsValues)):
    A[i].append(equationsValues[i])

# news values
equationsValues = calculateValuesUpper(A);
print(equationsValues)
ejerA();

def ejerB():
    # calcular la inversa de la matriz
    # 1. calcular la determinante de la matriz
    # 2. si la determinante es 0, entonces no tiene matriz
inversa
    # 3. luego la transpuesta de la matriz
    # 4. matriz adjunta
    # paso 1

```

```

A = [
    [4,1,3],
    [2,1,4],
    [0,1,2]
]
trans = np.zeros((3,3));
mInve = np.zeros((3,3));

# En este caso vamos a utilizar el try except para
detectar los errores
try:
    # hallar la determinante
    det = np.linalg.det(A)

    if det != 0:
        # calcular la trasnpuesta
        for i in range(len(A)):
            for j in range(len(A[0])):
                trans[j][i] = A[i][j];

    # matriz adjunta

    def matrixCof(i,j):
        mx = []
        for k in range(len(trans)):
            b = []
            for p in range(len(trans[0])):
                if (i != k and j != p):
                    b.append(trans[k][p])
            if len(b) != 0:
                mx.append(b)
        print(mx)
        return mx

    for i in range(len(trans)):

```

```

        for j in range(len(trans[0])):
            mInve[i][j] = round((((-1)**(i+j)) *
np.linalg.det(matrixCof(i,j))/det,2)
    else:
        print("No tiene inversa")
except:
    print("No tiene inversa")
print(mInve)

def multiplyMatrix(matrix1, matrix2):
    s = 0
    f1 = []
    for i in range(len(matrix1)):
        f2 = []
        for j in range(len(matrix2)):
            for k in range(len(matrix1[0])):
                s = s + (matrix1[i][k] * matrix2[k][j]);
            f2.append(round(s,1))
            s = 0
        f1.append(f2)
    return f1;
# demostración [A][A^-1] = [I]
print(multiplyMatrix(A,mInve))
ejerB();

ejer3();

```

1.4

```
# simple gaussian method
def methodGaussSimple(A):
    # tamaño de la matriz
    n = len(A)

    # eliminación hacia adelante
    # recorrido por columna
    for j in range(n):
        for i in range(n):
            if i == j:
                pivoMatriz = A[i];
                pivo = A[i][j];

            if i>j:
                newList = list(map(lambda x: x*(A[i][j]/pivo) ,
pivoMatriz))
                A[i] = list(map(lambda x,y: x-y , A[i],newList));
            # sustitución hacia atrás
            # calcular los valores del sistema de ecuaciones
            valEcuaciones = []
            for i in reversed(range(n)):
                sum = A[i][n];
                print(f"i")
                for j in reversed(range(i+1,n)):
                    print(n-j-1)
                    sum = sum - A[i][j]*valEcuaciones[n-j-1];
                valEcuaciones = [*valEcuaciones, sum/A[i][i]]

            # devolver el valor de las ecuaciones
    return valEcuaciones;
```

Capítulo XI

Matrices especiales y el método de Gauss-Seidel

1.1

```
# ejercicio 1 - método de thomas en matriz tridiagonal

from black import err
from sympy import false, true

def ejer1():
    # FUNCTIONS
    # Crear una matriz compuesta de ceros
    def createMatriz(size):
        # Se podría utilizar la librería numpy y evitar esta
    function
        # L = np.zeros(n,n)
        aux = []
        for i in range(size):
            a = [];
            for j in range(size):
                if i == j:
                    a.append(1)
                else:
                    a.append(0)
            aux.append(a);
        return aux;

    # calculate values of a matrix lower
    def calculateValuesLower(uniqueMatriz):
        valEcuaciones = []
        for i in range(n):
            sum = uniqueMatriz[i][n]
            for j in range(i):
                sum = sum - uniqueMatriz[i][j] * valEcuaciones[j];
            valEcuaciones.append(sum);
        return valEcuaciones;
```

```

    valEcuaciones.append(sum/uniqueMatriz[i][i]);
    return valEcuaciones;

# calculate values of a matriz upper
def calculateValuesUpper(uniqueMatriz):
    valEcuaciones = []
    for i in reversed(range(n)):
        sum = uniqueMatriz[i][n];
        for j in reversed(range(i+1,n)):
            sum = sum - uniqueMatriz[i][j]*valEcuaciones[n-j-1];
        valEcuaciones = [*valEcuaciones,
sum/uniqueMatriz[i][i]]
    return valEcuaciones

A = [
    [0.5,-2.7,0,0],
    [1,-0.2,-3,0],
    [0,4.9,-7.8,4],
    [0, 0,2.8,-3.4]
]
equationsValues = [
    -5.925,-1.35,4.975,-0.1
]

e = 0
f = 0

n = len(A);
L = createMatriz(n)

for i in range(n-1):
    if i == 0:
        e = A[i+1][i] / A[i][i]
    else:

```

```

e = A[i+1][i] / f

# formar la matriz u
A[i+1][i] = 0

# insert matrix L
L[i+1][i] = e

f = A[i+1][i+1] - e*A[i][i+1]

# insert matrix U
A[i+1][i+1] = f;

print("L")
for i in L:
    print(i)

# insertar los values de la equation en L
for i in range(len(equationsValues)):
    L[i].append(equationsValues[i])

# news values
equationsValues = calculateValuesLower(L);

print("U")
for i in A:
    print(i)

# insert values in U
for i in range(len(equationsValues)):
    A[i].append(equationsValues[i])

# news values
equationsValues = calculateValuesUpper(A);
print("VALORES")

```

```
    print(equationsValues)
```

método gaus seidel

```
# método de gauss seidel
def gausSeidel():

    # FUNCTIONS
    def diagonalDominante(matrix):
        for j in range(len(matrix)):
            for i in range(j, len(matrix)):
                if abs(matrix[j][j]) < abs(matrix[i][j]):
                    aux = matrix[j]
                    matrix[j] = matrix[i]
                    matrix[i] = aux
        return matrix

    def createArrayCeros(l):
        matrixAux = []
        for i in range(l):
            matrixAux.append(0)
        return matrixAux

    # VARIABLES
    M = [
        [3,-4,5,-7,19],
        [1,2,-4,5,31],
        [2,-4,5,-1,21],
        [3,-1,7,5,10]
    ];
    # tamaño de la matriz
    n = len(M)
    # error admitido
    ea = 0.01

    # diagonal dominante
```

```
M = diagonalDominante(M)

values = createArrayCeros(n)
errores = []
while(true):
    for i in range(n):
        sum = M[i][n]
        for k in range(n):
            if i != k:
                sum = sum - M[i][k]*values[k]
        aux = values[i]
        values[i] = sum / M[i][i]
        errores.append(abs(1-(aux/values[i])))

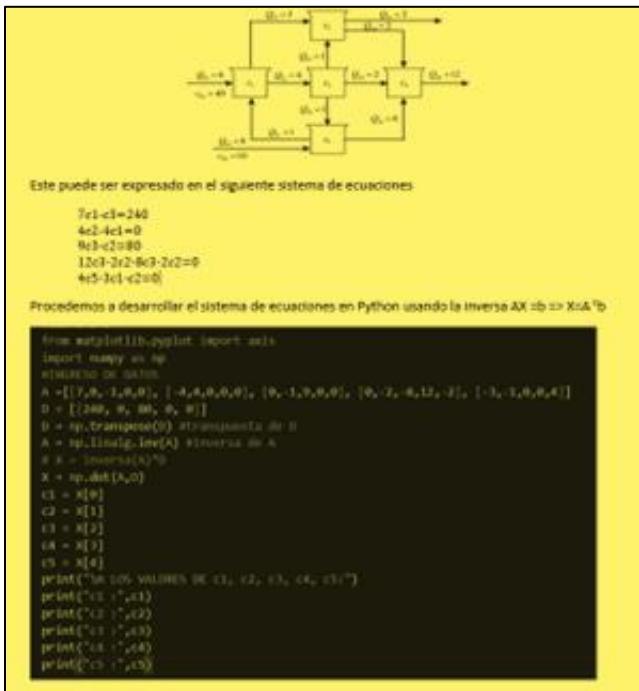
    if max(errores) < ea:
        break;
    errores = []

# mostrar los resultados
for i in range(len(values)):
    print(f"x{i} = {values[i]} - error = {errores[i]}")
gausSeidel()
```

Capítulo XII

Estudio de casos: ecuaciones algebraicas lineales

12.1



12.2

Modificando las restricciones al anterior problema tenemos

$$\begin{aligned}6e1-e3 &= 50 \\-3e1+3e2 &= 0 \\e2+9e3 &= 160 \\-e2-8e3-11e4+2e5 &= 0 \\-3e1-e2+4e5 &= 0\end{aligned}$$

Hallamos la solución en Python a través de la inversa

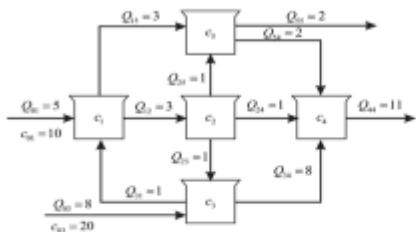
```

from matplotlib.pyplot import axis
import numpy as np
#INGRESO DE DATOS
A = [[1,0,-3,0,0], [-4,4,0,0,0], [0,-1,9,0,0], [0,-2,-8,12,-2], [-3,-1,0,0,4]]
A = np.linalg.inv(A) #inversa de A
#cambio de c1 se produce por la reducción del 25% del reactor 3
cambioC1 = A[1,2]**(0.25*100)
#cambio de c4 se produce por la reducción del 25% del reactor 3
cambioC4 = A[3,2]**(0.25*100)
#datos proporcionados en el texto
c1 = 11.51
c2 = 11.51
c3 = 19.06
c4 = 17.00
c5 = 11.51
#cálculo del cambio porcentual
cambio = (cambioC1/c1)*100
cambio2 = (cambioC4/c4)*100
# Muestra de resultado
print("\nCambio porcentual de C1 : ",cambio,"%")
print("Cambio porcentual de C4 : ",cambio2,"%")

```

12.3

Con respecto a la figura desarrollada en el problema 1 se tiene



Las unidades del caudal están expresadas en metro cúbico por minuto. Debido a que el sistema anterior está en estado estacionario, las entradas al sistema serán iguales a las salidas. Por lo tanto, la suma del caudal de entrada (Q_{in} and Q_{out}) será igual a la suma del caudal de salida (Q_{out} and Q_{in})

Por lo tanto, los cuatro flujos presentan la siguiente relación:

$$Q_1 + Q_2 = Q_3 + Q_4$$

12.4

Se obtiene el siguiente sistema lineal de ecuaciones

$$\begin{aligned} 8c_1 - 3c_3 &= 50 \\ 4c_2 - 4c_1 &= 0 \\ 10c_3 - 2c_2 &= 160 \\ 10c_4 - 7c_3 - 3c_5 &= 0 \\ 6c_5 - 4c_1 - 2c_2 &= 0 \end{aligned}$$

Tal y como se viene haciendo para los ejercicios anteriores usamos el mismo modelo en Python

```

from matplotlib.pyplot import axis
import numpy as np
#INGRESO DE DATOS
A = [[8,0,-3,0,0], [-4,4,0,0,0], [0,-2,10,0,0], [0,0,-7,10,-3], [-4,-2,0,0,6]]
D = [[50, 0, 160, 0, 0]]
D = np.transpose(D) #transpuesta de D
A = np.linalg.inv(A) #inversa de A
# X = inversa(A)*D
X = np.dot(A,D)
c1 = X[0]
c2 = X[1]
c3 = X[2]
c4 = X[3]
c5 = X[4]
print("\n LOS VALORES DE c1, c2, c3, c4, c5:")
print("c1 : ",c1)
print("c2 : ",c2)
print("c3 : ",c3)
print("c4 : ",c4)
print("c5 : ",c5)]

```

12.5

Modificando la figura del problema inicial, se obtiene el siguiente sistema de ecuaciones

$$\begin{aligned} 5c_1 &= 50 \\ 7c_2 - 7c_3 &= 0 \\ 10c_1 + 10c_3 &= 160 \\ 10c_4 - 7c_2 - 3c_3 &= 0 \\ 3c_5 - 3c_1 &= 0 \end{aligned}$$

Usando el mismo procedimiento se tiene

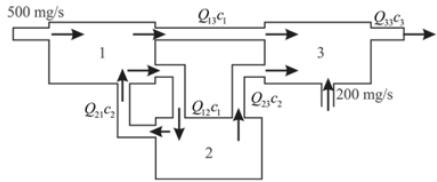
```

import numpy as np
#INGRESO DE DATOS
A = [[5,0,0,0,0], [0,7,-7,0,0], [-2,0,10,0,0], [0,-7,-3,10,0], [-3,3,0,0,3]
D = [[50, 0, 160, 0, 0]]
D = np.transpose(D) #transpuesta de D
A = np.linalg.inv(A) #Inversa de A
# X = inversa(A)*D
X = np.dot(A,D)
c1 = X[0]
c2 = X[1]
c3 = X[2]
c4 = X[3]
c5 = X[4]
print("\n LOS VALORES DE c1, c2, c3, c4, c5:")
print("c1 : ",c1)
print("c2 : ",c2)
print("c3 : ",c3)
print("c4 : ",c4)
print("c5 : ",c5)

```

12.6

Desarrollando el problema usando el diagrama especificado y agregando las incógnitas de la tasa de flujo se tiene



De este planteamos el siguiente sistema de inequaciones que se observa en el programa. La matriz está dada por A y el vector por D que corresponden a mg/s

```

import numpy as np
#INGRESO DE DATOS
A = [[130,-30,0], [-90,90,0], [-40,-60,120]]
D = [[500, 0, 200]]
D = np.transpose(D) #transpuesta de D
A = np.linalg.inv(A) #Inversa de A
# X = inversa(A)*D
X = np.dot(A,D)
c1 = X[0];c2 = X[1];c3 = X[2]

print("\n LOS VALORES DE c1, c2, c3:")
print("c1 : ",c1)
print("c2 : ",c2)
print("c3 : ",c3)

```

12.7

```

import numpy as np
#INGRESO DE DATOS
A = [[67,0,0,0,0], [0,36,0,0,0], [-67,-36,182,0,0], [0,0,-161,182,0], [0,0,0,-182,212]]
D = [[180, 710, 740, 3850, 4720]]
D = np.transpose(D) #transpuesta de D
A = np.linalg.inv(A) #inversa de A
# X = inversa(A)*D
X = np.dot(A,D)
cS = X[0]; cM = X[1]; cH = X[2]
cE = X[3]; cO = X[4]
print("\n LOS VALORES DE cS, cM, cH, cE, cO:")
print("cS : ",cS); print("cM : ",cM)
print("cH : ",cH); print("cE : ",cE)
print("cO : ",cO)

```

12.11

```

import numpy as np
#INGRESO DE DATOS
A = [[-2,1,2,0,0,0], [0,0,-2,1,2,0], [0,0,0,0,-2,3], [1,1,0,0,0,0], [0,1,-1,-1,0,0], [0,0,0,1,-1,-1]]
D = [[0, 0, 0, 1, 0, 0]]
D = np.transpose(D) #transpuesta de D
A = np.linalg.inv(A) #inversa de A
# X = inversa(A)*D
X = np.dot(A,D)
Q1 = X[0]
Q2 = X[1]
Q3 = X[2]
Q4 = X[3]
Q5 = X[4]
Q6 = X[5]
print("\n LOS VALORES DE Q1, Q2, Q3, Q4, Q5, Q6:")
print("Q1 : ",Q1)
print("Q2 : ",Q2)
print("Q3 : ",Q3)
print("Q4 : ",Q4)
print("Q5 : ",Q5)
print("Q6 : ",Q6)

```

12.13

```

import numpy as np
#INGRESO DE DATOS
A = [[0.55,0.25,0.25], [0.38,0.45,0.28], [0.15,0.30,0.55]]
D = [[4800, 5800, 5700]]
D = np.transpose(D) #transpuesta de D
A = np.linalg.inv(A) #inversa de A
# X = inversa(A)*D
X = np.dot(A,D)
X1 = X[0]
X2 = X[1]
X3 = X[2]

print("\n LOS VALORES DE X1, X2, X3:")
print("X1 : ",X1)
print("X2 : ",X2)
print("X3 : ",X3)

```

Capítulo XIII

Optimización unidimensional no restringida

13.1

a. Derivamos la función:
 $f(x) = -2x + 8$
 Igualamos a 0:
 $-2x + 8 = 0 \rightarrow x = 4$
 Usamos el criterio de la primera derivada, para esto evaluamos a $f'(x)$ en un punto a la izquierda y uno a la derecha de $x = 4$, entonces tomamos $x = 3, x = 5$:
 $f(3) = -2(3) + 8 = 2 > 0 \rightarrow$ la función crece por la izquierda
 $f(5) = -2(5) + 8 = -2 < 0 \rightarrow$ la función decrece por la derecha
 el valor de $f(x)$ en $x = 4$ es:

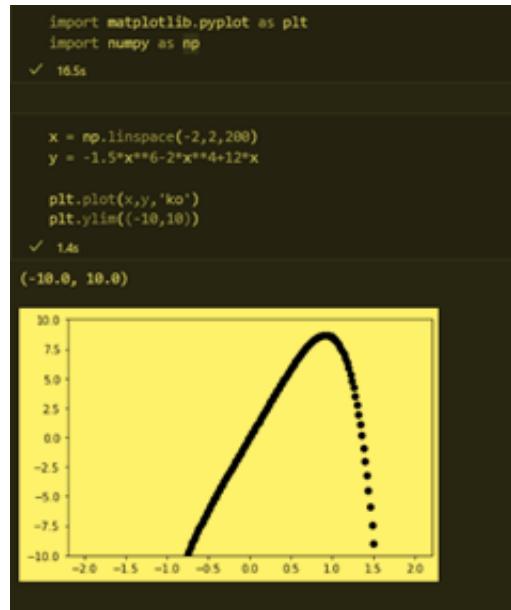
$$f(4) = -4^2 + 8(4) - 12 = 4$$

Entonces el punto máximo de $f(x)$ es (4,4)

x0	x1	x2	f(x0)	f(x1)	f(x2)	$(x1^2 - x2^2)$	x1-x2	$(x2^2 - x0^2)$	x2-x0	$(x0^2 - x1^2)$	x0-x1	x3
0	2	6	-12	0	0	-32	-4	36	6	-4	-2	4

Vemos que como resultado para $x_3 = 4$, entonces podemos indicar que mediante los dos métodos nos dan los mismos resultados

13.2



a)

Claramente, $f(x)$ se define como continua sobre la recta real \mathbb{R} siendo una función polinomial.

Calcular la primera y segunda derivada de $f(x)$

$$f(x) = 1.5x^6 - 2x^3 + 12x$$

$$f'(x) = 9x^5 - 8x^3 + 12$$

$$f''(x) = -45x^4 - 24x^2$$

$$= -3x^2(15x^2 + 8)$$

Es claro que en el intervalo $(-\infty, 0)$:

$f'(x) \geq 0 \rightarrow$ Entonces la función es cóncava hacia arriba en $(-\infty, 0)$.

Asimismo en el intervalo $(0, +\infty)$:

$f'(x) \leq 0 \rightarrow$ entonces la función es cóncava hacia abajo en $(0, +\infty)$.

b) Derivamos $f(x)$:

$$f'(x) = 9x^5 - 8x^3 + 12$$

El gráfico muestra que la raíz se encuentra entre 0 y 2.

Comienza con la primera iteración, aplica el método de ubicación de la raíz o el método de biseción de $[0,2]$.

i	x_l	x_u	x_r	$f(x_l)$	$f(x_r)$	$f(x_l)f(x_r)$	$\varepsilon_s (\%)$
1	0	2	1	12	-5	-60	
2	0	1	0.5	12	10.71875	128.6250	100
3	0.5	1	0.75	10.71875	6.489258	69.5567	33.33
4	0.75	1	0.875	6.489258	2.024445	13.1371	14.29
5	0.875	1	0.9375	2.024445	-1.10956	-2.2463	6.67

Calcular $f(x)$ para $x = 0.9375$

$f(x) = 1.5(0.9375)^6 - 2(0.9375)^4 + 12(0.9375) = 8.68664$, por lo tanto $f(x)$ es máximo en $x = 0.9375$ y el valor máximo aprox. Es 8.686.

13.3

```
for i in range(0,3):
    result = biseccion(x0,x1,epsilon)
    band_prm = 0
    print("str(i+1)*' ' + 'x = (0 : ,0f)'*3.format(np.around(x0,3))+' ,'
          'x = (0 : ,0f)'*3.format(np.around(x1,3))+' ,'
          'x = (0 : ,0f)'*3.format(np.around(result[0],3))+' ,'
          'x = (0 : ,0f)'*3.format(np.around(result[1],3))+' ,'
          'x = (0 : ,0f)'*3.format(np.around(result[2],3))+' ,'
          'x = (0 : ,0f)'*3.format(np.around(result[3],3))+' ,'
          'x = (0 : ,0f)'*3.format(np.around(result[4],3))+' ,'
          'err = '+str(err))
    if result[1]>result[2]:
        x = result[2]
        band_prm = 1
        xmp = result[0]
        err = (1-xmp)*abs((0-x)/result[0])*100
        xopt = result[0]
    else:
        x = result[0]
        band_prm = 2
        xmp = result[2]
        err = (1-xmp)*abs((0-x)/result[2])*100
        xopt = result[2]
    i += 1
print("x óptima = "+str(x_opt)+" evaluado en f(x) = "+str(function(x_opt)))
```

13.4

```
import numpy as np

x0 = 0
x1 = 1
x2 = 2

def function(x):
    y = -1.5*x**6 - 2*x**4 + 12*x
    return y

def InterpolationCubatica(x0,x1,x2):
    x3 = ((function(x0)*(x1**2-x2**2)+function(x1)*(x2**2-x0**2)+function(x2)*(x0**2-x1**2))/(2*(function(x0)*(x1-x2)+function(x1)*(x2-x0)+function(x2)*(x0-x1)))
    fx0 = function(x0)
    fx1 = function(x1)
    fx2 = function(x2)
    fx3 = function(x3)

    return x0, fx0, x1, fx1, x2, fx2, x3, fx3
```

```

for i in range(0,3):
    result = interpolate(result,(i,x1,x2))
print(str(i+1)+x0+"*exp(np.around(result[0],3))+", x0 = "+str(result[0]), result[1],"+", x0,
      x1 = "+str(np.around(result[1],3))+", "+str(np.around(result[2],3))+",
      "+x1+", x2 = "+str(np.around(result[2],3))+", "+str(np.around(result[3],3))+",
      "+x2+", "+str(np.around(result[3],3))+", "+x3+" = "+str(np.around(result[4],3)))
print("")

if result[0]==result[2]:
    x0 = x1
    x1 = result[4]
else:
    x1 = x2
    x2 = x0
print("El símbolo se encuentra en x = "+str(np.around(result[4],3))+" con un valor en la función de "+str(np.around(result[4],3)))

```

13.5

```

from sympy import Symbol
from scipy.misc import derivative
from sympy import simplify
import numpy as np

x = Symbol('x')
y = -1.5*x**6 + 2*x**4 + 12*x

x0 = 2

y_prima = y.diff(x)
y_segunda = y_prima.diff(x)

def Newton(x0):
    fx0pri = simplify(y_prima).subs(x,x0)
    fx0seg = simplify(y_segunda).subs(x,x0)
    x_1 = x0 - (fx0pri)/(fx0seg)

    return fx0pri, fx0seg, x_1

for i in range(0,3):
    fx0 = simplify(y).subs(x,x0)
    result = Newton(x0)
    print(str(i+1)+"*x0 = "+str(x0)+"+", fx0 = "+str(fx0)+"+", fx0 = "+str(result[0])+", fx0 = "+str(result[1]))
    x0 = result[2]
    if i==2:
        print("El símbolo se encuentra en x = "+str(x0)+" con un valor en la función de "+str(result[2]))

```

13.12

a)

```

from sympy import Symbol
from scipy.misc import derivative
from sympy import simplify
import numpy as np

x = Symbol('x')
y = 3 + 6*x + 5*x**2 + 3*x**3 + 4*x**4

x0 = -1.0
err = 100
x_ant=0

y_prima = y.diff(x)
y_segunda = y_prima.diff(x)
i = 1

def Newton(x0):
    fx0pri = simplify(y_prima).subs(x,x0)
    fx0seg = simplify(y_segunda).subs(x,x0)
    x_1 = x0 - (fx0pri)/(fx0seg)

    return fx0pri, fx0seg, x_1

while err>=1:
    fx0 = simplify(y).subs(x,x0);
    err = np.absolute((x0 - x_ant)/x0)*100
    result = Newton(x0)
    print(str(i)+"*x0 = "+str(x0)+"+", fx0 = "+str(fx0)+"+", fx0 = "+str(result[0])+", fx0 = "+str(result[1]), Err = "+str(err))
    x_ant = x0

```

b)

```

x = Symbol('x')

def #funcion(x):
    y = 3 + 6*x + 5*x**2 + 3*x**3 + 4*x**4
    return y

x0 = -1.0
err = 100
x_ant=0
d= 0.01

def #funcion_prima(x):
    y_prima = (funcion(x+d*x) - #funcion(x - d*x))/(2*d*x)
    return y_prima
def #funcion_segunda(x):
    y_segunda = (funcion(x+d*x) - 2*#funcion(x) + #funcion(x-d*x))/(d**2)
    return y_segunda
i = 1

def Newton(x0):
    fx0pri = #funcion_prima(x0)
    fx0seg = #funcion_segunda(x0)
    x_1 = x0 - (fx0pri)/(fx0seg)

    return fx0pri, fx0seg, x_1

while err>=1:
    fx0 = #funcion(x0)
    err = np.absolute((x0 - x_ant)/x0)*100

    result = Newton(x0)
    print(str(i)+" "+x0+" "+str(x0)+" "+str(result[0])+" "+str(result[1])+" "+str(err))
    x_ant = x0
    x0 = result[2]
    i = i + 1

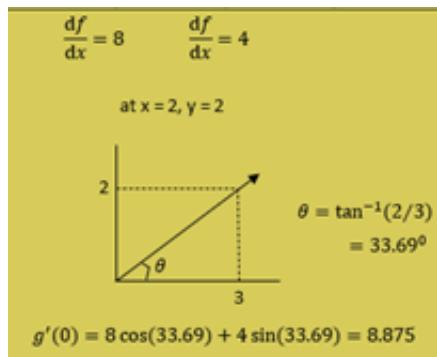
print("")
print("El minimo se encuentra en x = "+str(x0)+" con un valor en la #función de "+str(result[0]))

```

Capítulo XIV

Optimización multidimensional no restringida

14.1



14.2

$$a) \nabla f = \begin{bmatrix} 2y^2 + 3ye^{xy} \\ 4xy + 3xe^{xy} \end{bmatrix}$$

$$H = \begin{bmatrix} 3y^2e^{xy} & 4y + 3xye^{xy} + 3e^{xy} \\ 4y + 3xye^{xy} + 3e^{xy} & 4x + 3x^2e^{xy} \end{bmatrix}$$

$$b) \nabla f = \begin{bmatrix} 2x \\ 2y \\ 4z \end{bmatrix} \quad H = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 4 \end{bmatrix}$$

$$c) \nabla f = \begin{bmatrix} 2x + 2y \\ x^2 + 2xy + 3y^2 \\ 2x + 6y \\ x^2 + 2xy + 3y^2 \end{bmatrix}$$

$$H = \begin{bmatrix} -2x^2 + 2y^2 - 4xy & -2x^2 - 6y^2 - 12xy \\ 2x^2 - 6y^2 - 12xy & 2x^2 - 24y^2 - 12xy \end{bmatrix}_{(x^2 + 2xy + 3y^2)^2}$$

14.3

$$\frac{df}{dx} = \frac{df}{dy} = 0$$

$$-2.5x + 2y = 0$$

$$2x - 4y = -1.5$$

Resolviendo:

$$x = 0.5 \quad y = 0.625$$

14.4

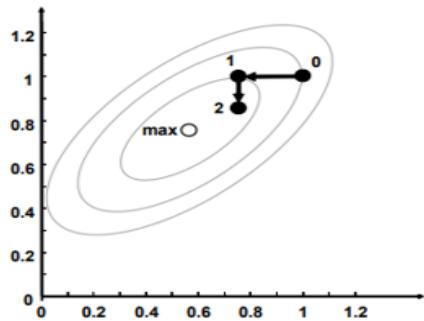
$\frac{df}{dx} = 2y - 2.5x$	
$\frac{df}{dy} = 2x + 1.5 - 4y$	
$x_0 = 1, y_0 = 1$	
$\frac{df}{dx} = -0.5$	$\frac{df}{dy} = -0.5$
Busca dirección:	$= -0.5i - 0.5j$
$f(1 - 0.5h, 1 - 0.5h) = 1 - 0.75h - 1.25(1 - h + 0.25h^2)$	
$g(h) = 0.25 + 0.5h - 0.3125h^2$	
Nos da:	$g'(h) = 0; h^* = 0.8$

14.5

Las derivadas parciales se pueden evaluar en las suposiciones iniciales, $x = 1, y = 1$
$>> f=2.25*x*y+1.75*y-1.5*x^2-2*y^2$
$>> primera=diff(f,x)$
$>> subs(primera,[x,y],[1,1])$
Da el siguiente resultado:
$\frac{\partial f}{\partial x} = -3x + 2.25y = -3(1) + 2.25(1) = -0.75$
$\frac{\partial f}{\partial y} = 2.25x - 4y + 1.75 = 2.25(1) - 4(1) + 1.75 = 0$
Por lo tanto, la dirección de búsqueda es $-0.75i$.
$f(1 - 0.75h, 1) = 0.5 + 0.5625h - 0.84375h^2$

Esto se puede diferenciar, igualar a cero y resolver para $h^* = 0.33333$. Por lo tanto, los
El resultado de la primera iteración es $x = 1 - 0.75$ $(0.3333) = 0.75$ e $y = 1 + 0 (0.3333) = 1$.
Para la segunda iteración, las derivadas parciales se pueden evaluar como
$>> subs(primera,[x,y],[0.75,1])$
$ans =$
0
$>> subs(segunda,[x,y],[0.75,1])$
$ans =$
$-9/16$
Por lo tanto, la dirección de búsqueda es $-0.5625j$.
$f(0.75, 1 - 0.5625h) = 0.59375 + 0.316406h - 0.63281h^2$

Esto puede diferenciarse y ponerse igual a cero y resolverse para $h^* = 0.25$. Por tanto, el resultado de la segunda iteración es $x = 0.75 + 0 (0.25) = 0.75$ e $y = 1 + (-0.5625) 0.25 = 0.859375$.



14.6

```
>> f=(x-3)^2+(y-2)^2
>> primera=diff(f,x)
>> subs(primera,[x,y],[1,1])
ans =
-4

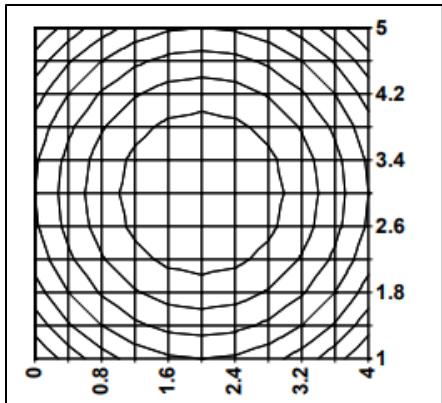
>> f=(x-3)^2+(y-2)^2
>> segunda=diff(f,y)
>> subs(segunda,[x,y],[1,1])
ans =
-2

>> g=subs(f,[x,y],[1-4*h,1-2*h])
>> diff(g)
>> diff(g)

ans =
```

$40h + 20$
Igualando g' a 0 tendríamos que $h^*=-0.5$, luego
 $x = 1 - 4(-0.5) = 3$
 $y = 1 - 2(-0.5) = 2$

Por lo tanto, para este caso especial, el enfoque converge en la respuesta correcta después de una sola iteración. Esto ocurre porque la función es esférica como se muestra a continuación. Por lo tanto, el gradiente para cualquier conjetura apunta directamente a la solución.



14.7

Las derivadas parciales se pueden evaluar en los valores siguientes iniciales, $x = 0$, $y = 0$

```

>> f=4*x+2*y+x^2-2*x^4+2*x*y-3*y^2
>> primera=diff(f,x)
>> subs(primera,[x,y],[0,0])
ans =
4
>> f=4*x+2*y+x^2-2*x^4+2*x*y-3*y^2
>> segunda=diff(f,y)
>> subs(segunda,[x,y],[0,0])
ans =
2
>> g=subs(f,[x,y],[0+4*h,0+2*h])
g =

```

```

g =
- 512*h^4 + 20*h^2 + 20*h
>> diff(g)
ans =
- 2048*h^3 + 40*h + 20

```

La raíz de esta ecuación se puede hallar mediante bisección usando el valor $h=0$

La raíz de esta ecuación se puede determinar mediante bisección. El uso de estimaciones iniciales de $h = 0$ y 1 produce una raíz de $h^* = 0.244$ después de 13 iteraciones con $a = 0.05\%$

Luego

$$x = 0 + 4(0.244) = 0.976$$

$$y = 0 + 2(0.244) = 0.488$$

Interaccion	xa	xb	Xr	f(xa)	f(xr)	f(xa)*f(xr)	Ea
1	0	1	0.5	20	-216	-4320	
2	0	0.5	0.25	20	-2	-40	100%
3	0	0.25	0.125	20	21	420	100
4	0.125	0.25	0.1875	21	14	294	33.3333333
5	0.1875	0.25	0.21875	14	7.3125	102.375	14.2857143
6	0.21875	0.25	0.234375	7.3125	3.0078125	21.9946289	6.66666667
7	0.234375	0.25	0.2421875	3.0078125	0.59472656	1.78882599	3.22580645
8	0.2421875	0.25	0.24609375	0.59472656	-0.67956543	-0.40415561	1.58730159
9	0.2421875	0.24609375	0.24414063	0.59472656	-0.03669739	-0.02182491	0.8
10	0.2421875	0.24414063	0.24316406	0.59472656	0.28043938	0.16678475	0.40160643
11	0.24316406	0.24414063	0.24365234	0.28043938	0.12222791	0.03427752	0.2004008
12	0.24365234	0.24414063	0.24389648	0.12222791	0.04285458	0.00523803	0.1001001
13	0.24389648	0.24414063	0.24401855	0.04285458	0.00010094	0.00013289	0.05002501

14.8

```
>> f=-8*x+x^2+12*y+4*y^2-2*x*y
>> primera=diff(f,x)
>> subs(primera,[x,y],[0,0])
ans =
-8
>> segunda=diff(f,y)
>> subs(segunda,[x,y],[0,0])
ans =
12
>> g=subs(f,[x,y],[0-8*h,00+12*h])
g =
832*h^2 + 208*h
>> diff(g)
```

```
ans =
1664*h + 208
El cual h*=-0.125
Luego
x = 0 - 8(-0.125) = 1
y = 0 + 12(-0.125) = -1.5
```

14.10

```

Option Explicit
Sub GridSearch()
Dim nx As Long, ny As Long
Dim xmin As Double, xmax As Double, ymin As Double, ymax As Double
Dim maxf As Double, maxx As Double, maxy As Double
xmin = -2: xmax = 2: ymin = 1: ymax = 3
nx = 1000
ny = 1000
Call GridSrch(nx, ny, xmin, xmax, ymin, ymax, maxx, maxf)
MsgBox maxf
MsgBox maxx
MsgBox maxy
End Sub
Sub GridSrch(nx, ny, xmin, xmax, ymin, ymax, maxy, maxx, maxf)
Dim i As Long, j As Long
Dim x As Double, y As Double, fn As Double
Dim xinc As Double, yinc As Double
xinc = (xmax - xmin) / nx
yinc = (ymax - ymin) / ny

```

```

maxf = -100000000#
x = xmin
For i = 0 To nx
    y = ymin
    For j = 0 To ny
        fn = f(x, y)
        If fn > maxf Then
            maxf = fn
            maxx = x
            maxy = y
        End If
        y = y + yinc
    Next j
    x = x + xinc
Next i
End Sub
Function f(x, y)
f = y - x - 2 * x ^ 2 - 2 * x * y - y ^ 2

```

14.11

```

>>f=6*x^2*y-9*y^2-8*x^2
>>primera=diff(f,x)
>>subs(primera,[x,y],[2,4])

$$\frac{\partial f}{\partial x} = 12xy - 16x \Rightarrow 12(2)(4) - 16(4) = 32$$

>>segunda=diff(f,y)
>>subs(segunda,[x,y],[4,2])

$$\frac{\partial f}{\partial y} = 6x^2 - 18y \Rightarrow 6(4)^2 - 18(2) = 60$$


$$\nabla f = 32\hat{i} + 60\hat{j}$$

>>g=subs(f,[x,y],[4+32*h,2+60*h])
g(x)=368,640h^3 + 63,856h^2 + 4,624h + 28

```

14.12

```

>> f=2*x^3*y^2-7*y*x+x^2+3*y
>> primera=diff(f,x)
primera =
6*x^2*y^2 + 2*x - 7*y

>> subs(primera,[x,y],[1,1])
ans =

1

>> segundaa=diff(f,y)
segunda =
4*y*x^3 - 7*x + 3

>> subs(segunda,[x,y],[1,1])
ans =

```

```

0

 $\nabla f = 1\hat{i} + 0\hat{j}$ 

>> g=subs(f,[x,y],[1+h,1+0*h])
g =
(h + 1)^2 - 7*h + 2*(h + 1)^3 - 4

g(x) = 2h^3 + 7h^2 + h - 1

```

Capítulo XV

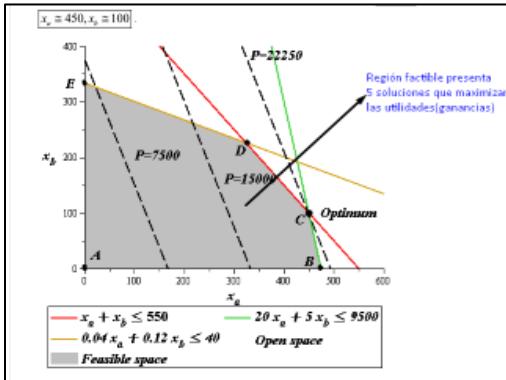
Optimización restringida

15.1

Maximizar $P = 45x_a + 20x_b$ Maximizar ganancias

Sujeto a:

$$\begin{array}{ll} 20x_a + 5x_b \leq 9500 & \text{(raw material constraint)} \\ 0.04x_a + 0.12x_b \leq 40 & \text{(production time constraint)} \\ x_a + x_b \leq 550 & \text{(storage constraint)} \\ x_a, x_b \geq 0 & \text{(positivity constraint)} \end{array}$$



```

import gurobipy as gp
from gurobipy import GRB
# Create a new model
model = gp.Model("simpleLP")
# Define decision variables
model.x1 = gp.Var(name="x1", vtype=GRB.CONTINUOUS)
model.x2 = gp.Var(name="x2", vtype=GRB.CONTINUOUS)
# Set objective: Maximize profit = 45*x1 + 20*x2
model.Obj = gp.LinExpr(45*model.x1 + 20*model.x2, constant=0)
# Constraints
model.Constraint1 = gp.Constraints(expr = 20*model.x1 + 5*model.x2 <= 9500)
model.Constraint2 = gp.Constraints(expr = 0.04*model.x1 + 0.12*model.x2 <= 40)
model.Constraint3 = gp.Constraints(expr = model.x1 + model.x2 <= 550)
# Solving the model
model.setObjective(gp.LinExpr(45*model.x1 + 20*model.x2))
model.optimize()
print("Status: ", model.Status)
print("Optimal value: ", model.ObjVal)
print("Optimal x1: ", model.x1)
print("Optimal x2: ", model.x2)

```

A	B	C	D	E	F	G	H
1	Microsoft Excel 12.0 Sensitivity Report						
2	Worksheet: [Book1]Sheet1						
3	Report Created: 13-Jun-15 5:48:41 PM						
4							
5							
6	Adjustable Cells						
7		Final	Reduced	Objective	Allowable	Allowable	
8	Cell	Name	Value	Cost	Coefficient	Increase	Decrease
9	\$B\$2	produced xa	450	0	45	35	25
10	\$C\$2	produced xb	100	0	20	25	8.75
11							
12	Constraints						
13		Final	Shadow	Constraint	Allowable	Allowable	
14	Cell	Name	Value	Price	R.H. Side	Increase	Decrease
15	\$D\$3	raw material total	9500	1.666666667	9500	1500	1875
16	\$D\$4	time total	30	0	40	1E+30	10
17	\$D\$5	storage total	550	11.66666667	550	68.18181818	75

15.2

$$\text{Maximizar } Z = 150x_1 + 175x_2 + 250x_3$$

Sujeto a:

$$7x_1 + 11x_2 + 15x_3 \leq 154 \text{ (material constraint)}$$

$$10x_1 + 8x_2 + 12x_3 \leq 80 \text{ (time constraint)}$$

$$x_1 \leq 9 \text{ (regular storage constraint)}$$

$$x_2 \leq 6 \text{ (premium storage constraint)}$$

$$x_3 \leq 5 \text{ (supreme storage constraint)}$$

$$x_1, x_2, x_3 \geq 0 \text{ (positivity constraints)}$$

```

# Importando la libreria
model = pulp.LpProblem("")

# Definiendo las variables de decisión y su tipo
model.x1 = pulp.LpVariable("x1", lowBound=0, upBound=9)
x1 = model.x1

model.x2 = pulp.LpVariable("x2", lowBound=0, upBound=6)
x2 = model.x2

model.x3 = pulp.LpVariable("x3", lowBound=0, upBound=5)
x3 = model.x3

# Poniendo restricciones
model.constr1 = pulp.Constraint(x1+11*x2+15*x3 <= 154)
model.constr2 = pulp.Constraint(x1+8*x2+12*x3 <= 80)
model.constr3 = pulp.Constraint(x1 <= 9)
model.constr4 = pulp.Constraint(x2 <= 6)
model.constr5 = pulp.Constraint(x3 <= 5)

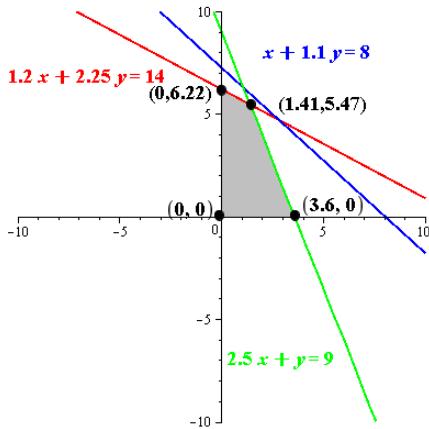
# Resolviendo el modelo
solver = pulp.SolverFactory('CPLEX')
results = solver.solve(model)

print(results)
print("Objetivo:", results.value(model.objective))
print("x1: ", model.x1.value())
print("x2: ", model.x2.value())
print("x3: ", model.x3.value())

```

A	B	C	D	E	F	G	H
1	Microsoft Excel 12.0 Sensitivity Report						
2	Worksheet: [46045-15-2P.xlsx]Sheet1						
3	Report Created: 15-Jun-15 4:13:01 PM						
4							
5							
6	Adjustable Cells						
7	Cell	Name	Final Value	Reduced Cost	Objective Coefficient	Allowable Increase	Allowable Decrease
9	\$B\$2	produced regular	0	-58.333333333	150	58.333333333	1E+30
10	\$C\$2	produced premium	6	0	175	1E+30	8.333333333
11	\$D\$2	produced supreme	2.666666667	0	250	12.5	70
12							
13	Constraints						
14	Cell	Name	Final Value	Shadow Price	Constraint R.H. Side	Allowable Increase	Allowable Decrease
16	\$E\$3	raw total	106	0	154	1E+30	48
17	\$E\$4	time total	80	20.833333333	80	28	32
18	\$E\$5	storage regular total	0	0	9	1E+30	9
19	\$E\$6	storage premium total	6	8.333333333	6	4	3.5
20	\$E\$7	storage supreme total	2.666666667	0	5	1E+30	2.333333333

15.3



Se observa que hay 4 soluciones posibles que pueden maximizar la función objetivo(el valor trivial no se considera, así que tendríamos 3). Hallamos la solución óptima hallando los puntos de intersección a través de solución de inecuaciones.

Para el punto (0, 6.22)

$$\begin{aligned} Z &= 1.75(0) + 1.25(6.22) \\ &= 0 + 7.775 \\ &= 7.775 \end{aligned}$$

Para el punto (1.42, 5.47)

$$\begin{aligned} Z &= 1.75(1.4124) + 1.25(5.4689) \\ &= 2.4717 + 6.8361 \\ &= 9.3078 \\ &\approx 9.308 \end{aligned}$$

Para el punto (3.6, 0)

$$\begin{aligned} Z &= 1.75(3.6) + 1.25(0) \\ &= 6.3 + 0 \\ &= 6.3 \end{aligned}$$

De ello, se desprende que el punto (1.42, 5.47) es el que contiene la solución que maximice las utilidades.

```
# Definimos el modelo
model = pyo.ConcreteModel()

# Inicializando las variables de decisión y restringiendo
model.x1 = pyo.Var(within=pyo.NonNegativeReals)
x1 = model.x1

model.x2 = pyo.Var(within=pyo.NonNegativeReals)
x2 = model.x2

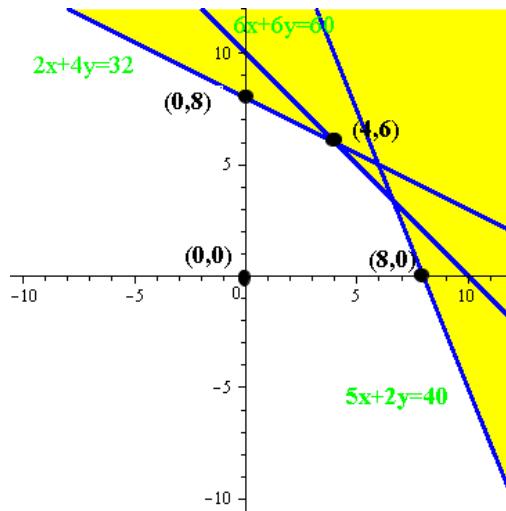
# Función objetivo
model.Obj = pyo.Objective(expr=1.75*x1 + 1.25*x2, sense=pyo.maximize)

# Constraints
model.Const1 = pyo.Constraint(expr=1.2*x1+2.25*x2<= 14)
model.Const2 = pyo.Constraint(expr=x1+1.1*x2 <= 8)
model.Const3 = pyo.Constraint(expr=2.5*x1 +x2<= 9)

# Solving the model
optm = SolverFactory('glpk')
results = optm.solve(model)

print(results)
print('obj function', model.Obj())
print('model x1 ', x1())
print('model x2 ', x2())
```

15.4



Sustituimos el punto $(0,8)$

$$\begin{aligned} Z &= 6(0) + 8(8) \\ &= 0 + 64 \\ &= 64 \end{aligned}$$

Sustituimos el punto $(4,6)$

$$\begin{aligned} Z &= 6(4) + 8(6) \\ &= 24 + 48 \\ &= 72 \end{aligned}$$

Sustituimos el punto $(8,0)$

$$\begin{aligned} Z &= 6(8) + 8(0) \\ &= 48 + 0 \\ &= 48 \end{aligned}$$

```
# Definimos el modelo
model = pyo.ConcreteModel()

# Initializando las variables de decisión y restringiendo
model.x1 = pyo.Var(within=pyo.NonNegativeReals)
x1 = model.x1

model.x2 = pyo.Var(within=pyo.NonNegativeReals)
x2 = model.x2

# Función objetivo
model.Obj = pyo.Objective(expr=6*x1 + 8*x2, sense=pyo.maximize)

# Constraints
model.Const1 = pyo.Constraint(expr=5*x1+2*x2<= 40)
model.Const2 = pyo.Constraint(expr=6*x1+4*x2<= 32)
model.Const3 = pyo.Constraint(expr=2*x1+4*x2<= 32)
model.Const4 = pyo.Constraint(expr=x1+2*x2<= 50)

# Solving the model
optm = SolverFactory('glpk')
results = optm.solve(model)

print(results)
print('obj function:', model.Obj())
print('model x1 ', x1())
print('model x2 ', x2())
print('x1 ', x1())
print('x2 ', x2())
```

15.5

```
...
File: pyomo/repn/plugins/cpxlp.pyx:159, in pyomo.repn.plugins.cpxlp.Pro
    bleminit_cpxlp._call_(...)
#file pyomo/repn/plugins/cpxlp.pyx:151, in pyomo.repn.plugins.cpxlp.Pro
    bleminit_cpxlp._print_model(P)
RuntimeError: Cannot write legal LP file. Objective 'Obj' has nonlinear
ar terms that are not quadratic.
```

En Excel, tipeamos la formula y las restricciones del problema en una tabla

A	B	C	D
1	x	y	f(x,y)
2	0	0	=1.2*A2+2*B2-B2^3 =2*A2+B2

Luego activamos el complemento solver en Excel y vamos a Datos y seleccionamos solver. Despues, ingresamos los argumentos del problema (restriccion y valores de x y Y, seleccionando las celdas y dando una referencia abosulta)



Presionamos el botón **Solve**.

A	B	C	D
1	x	y	f(x,y)
2	0.66	0.68	1.837588

Tenemos que en el punto (0.66, 0.68) se obtiene el punto que maximiza la función. Su valor es 1.837.

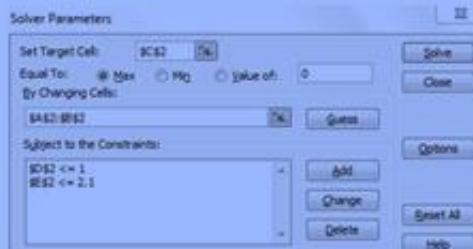
15.6

De la misma forma que el anterior problema usamos Excel para resolver el problema de programación.

Transcribimos las restricciones del problema y la función objetivo.

A	B	C	D	E
1	x	y	f(x,y)	Constraint:
2	0	0	=15*A2+15*B2	=A2^2+B2^2

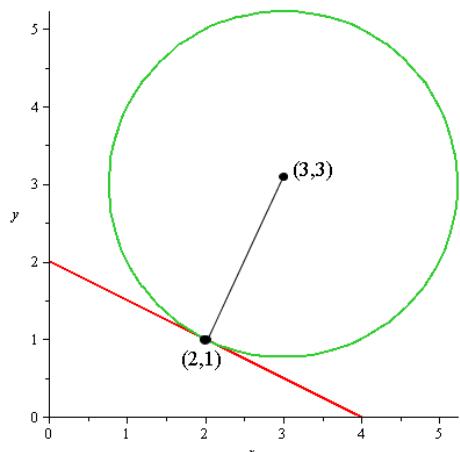
Luego procedemos a usar Solver agregando los respectivos argumentos al seleccionar las celdas y damos click en **Solve**.



La solución que obtenemos se encuentra especificado en el punto (0.73, 0.7) el cual maximiza la función objetivo. Su valor es 21.2

A	B	C	D	E
1	x	y	f(x,y)	Constraint:
2	0.73	0.7	21.204351	1.000001

15.7



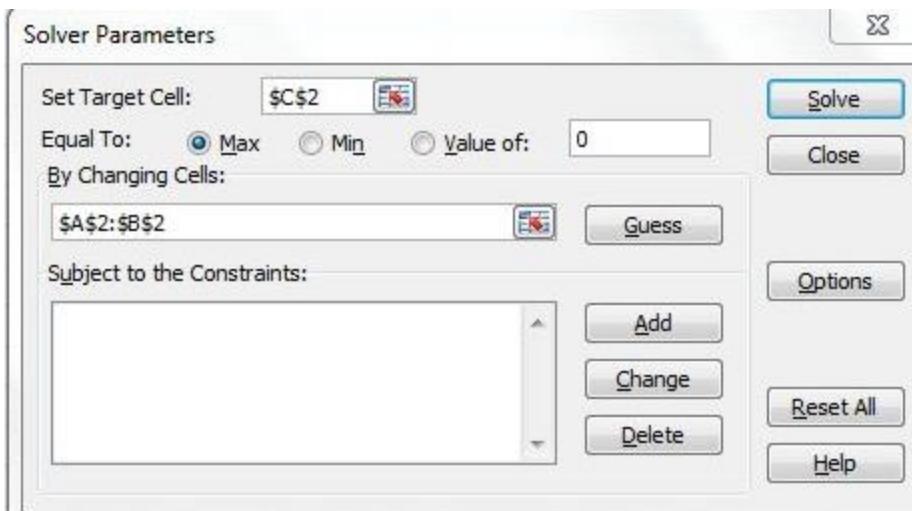
	A	B
1	x	0
2	y	0
3	Minimize	
4	$f(x,y)$	18
5	Subject to	
6	$x+2y$	0
7		



	A	B
1	x	1.999999992
2	y	1.000000004
3	Minimize	
4	$f(x,y)$	5
5	Subject to	
6	$x+2y$	4
7		

15.8

	A	B	C
1	x	y	$f(x,y)$
2	0	0	$=2.25*A2*B2+1.75*B2-1.5*A2^2-2*B2^2$



	A	B	C
1	x	y	f(x,y)
2	0.567568	0.756757	0.662162162

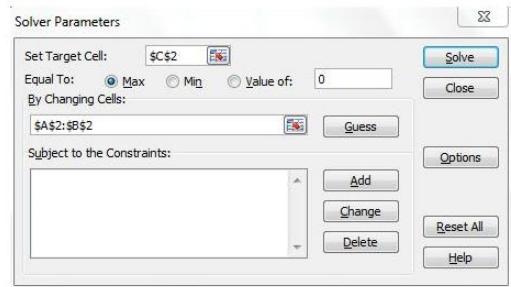
15.9

Este problema es parecido al anterior.

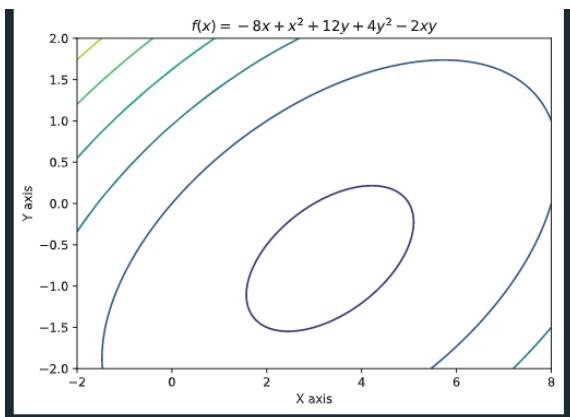
Desarrollamos la tabla

	A	B	C
1	x	y	f(x,y)
2	0	0	=4*A2+2*B2+A2^2-2*A2^4+2*A2*B2-3*B2^2

Asignamos el único argumento: la celda C2 que contiene la función objetivo.



15.10



Para responder el inciso b usamos Matlab puesto que tiene una built-in function que permite calcular el mínimo de una función multivariante.

```

untitled.mlx * x +
1
2
3
4
5
function f = func(x)
f = -8*x(1)+x(1)^2+12*x(2)+4*x(2)^2-2*x(1)*x(2)
end
x = fminsearch(@func,[0,0])

```

El valor que retorna la función `fminsearch` es

```

x =
3.3333 -0.6666

```

Por ende, la función tiene un mínimo en $x = 3.33$ y $y = -0.6666$

En ese sentido, usamos esos valores para hallar el valor de la función

$$\begin{aligned} f(3.33, -0.66) &= -8(3.33) + (3.33)^2 + 12(-0.66) + 4(-0.66)^2 - 2(3.33)(-0.66) \\ &= -17.3332 \end{aligned}$$

De manera analítica, para hallar el hessiano necesitamos hallar las derivadas parciales de x , y y su combinación. Esto nos da la solución de donde se encuentran los puntos mínimos.

Para x

$$\begin{aligned} \frac{\partial f(x,y)}{\partial x} &= \frac{\partial (-8x + x^2 + 12y + 4y^2 - 2xy)}{\partial x} \\ &= -8 + 2x - 2y \end{aligned}$$

Para "y" y x

$$\begin{aligned} \frac{\partial^2 f(x,y)}{\partial y \partial x} &= \frac{\partial (-8 + 2x - 2y)}{\partial y} \\ &= -2 \end{aligned}$$

Para x^2

$$\begin{aligned} \frac{\partial^2 f(x,y)}{\partial x^2} &= \frac{\partial (-8 + 2x - 2y)}{\partial x} \\ &= 2 \end{aligned}$$

Con respecto a "y"

$$\begin{aligned}\frac{\partial f(x,y)}{\partial y} &= \frac{\partial(-8x + x^2 + 12y + 4y^2 - 2xy)}{\partial y} \\ &= 12 + 8y - 2x\end{aligned}$$

Con respecto a x y "y"

$$\begin{aligned}\frac{\partial^2 f(x,y)}{\partial x \partial y} &= \frac{\partial(12 + 8y - 2x)}{\partial x} \\ &= -2\end{aligned}$$

Con respecto a y²

$$\begin{aligned}\frac{\partial^2 f}{\partial y^2} &= \frac{\partial(12 + 8y - 2x)}{\partial y} \\ &= 8\end{aligned}$$

Luego, hallamos el hessiano, que es la determinante de las derivadas parciales halladas

$$H = \begin{vmatrix} \frac{\partial^2 f(x,y)}{\partial x^2} & \frac{\partial^2 f(x,y)}{\partial x \partial y} \\ \frac{\partial^2 f(x,y)}{\partial y \partial x} & \frac{\partial^2 f(x,y)}{\partial y^2} \end{vmatrix}$$

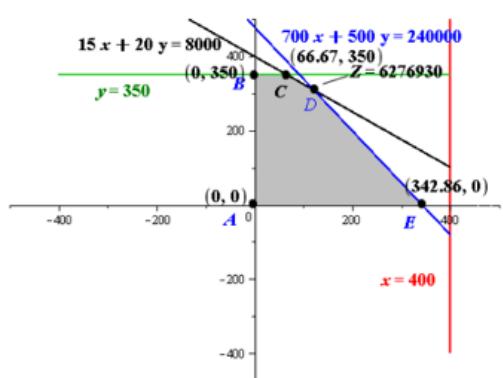
Sustituyendo los valores

$$H = \begin{bmatrix} 2 & -2 \\ -2 & 8 \end{bmatrix}$$

$$\begin{aligned}|H| &= 2 \times 8 - (-2)(-2) \\ &= 12\end{aligned}$$

Hay una concavidad hacia arriba. Esto significa que el punto en cuestión es un punto mínimo.

15.12



Hallamos los puntos de intersección para la región factible (feasible región)

A(0, 350), B(66.67, 350), C(342.86, 0), D(123.08, 307.69).

Reemplazamos en la función objetivo para elegir cual de todos los puntos maximiza a la función objetivo.

$$\begin{aligned} Z &= 13500(0) + 15000(350) \\ &= 0 + 5250000 \\ &= 5250000 \end{aligned}$$

$$\begin{aligned} Z &= 13500(66.67) + 15000(350) \\ &= 900045 + 5250000 \\ &= 6150045 \end{aligned}$$

$$\begin{aligned} Z &= 13500(123.08) + 15000(307.69) \\ &= 1661580 + 4615350 \\ &= 6276930 \end{aligned}$$

$$\begin{aligned} Z &= 13500(342.86) + 15000(0) \\ &= 4628610 + 0 \\ &= 4628610 \end{aligned}$$

Por ende, el punto que maximiza es D(123.08, 307.69) con un valor de 6276930.

Ahora, usamos Python para comprobar la correctitud de nuestro análisis. Desarrollamos el modelo.

```
# Definimos el modelo
model = pyo.ConcreteModel()

# Inicializando las variables de decisión y restringiendo
model.x1 = pyo.Var(within=pyo.NonNegativeReals)
x1 = model.x1

model.x2 = pyo.Var(within=pyo.NonNegativeReals)
x2 = model.x2

# Funcion objetivo
model.Obj = pyo.Objective(expr=13500*x1+15000*x2, sense=pyo.maximize)

# Constraints
model.Const1 = pyo.Constraint(expr=15*x1+20*x2<= 8000)
model.Const2 = pyo.Constraint(expr=700*x1+500*x2<= 240000)
model.Const3 = pyo.Constraint(expr=x1<=400)
model.Const4 = pyo.Constraint(expr=x2<=350)

# Solving the model
optm = SolverFactory('glpk')
results = optm.solve(model)

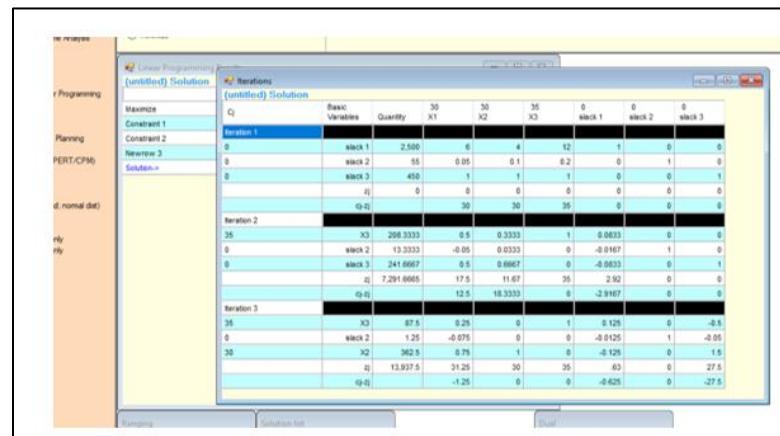
print(results)
print('obj function', model.Obj())
print('model x1 ', x1())
print('model x2 ', x2())
✓ 2.75
```

15.16

```

x_inferior = 0
x_superior = 5
error_sol = 0.01
error_abs = 0
main_f = lambda x: 2 * x / (4 + 0.8 * x + pow(x, 2) + 0.2 * pow(x, 3))
R = (pow(5, 0.5) - 1) / 2
xl = x_inferior
xu = x_superior
g_reason = R * (xu - xl)
x1 = xl + g_reason
x2 = xu - g_reason

```



c) Ahora procedemos a usar Python para desarrollar el modelo con la librería [scipy](#).

```

from scipy.optimize import linprog
c = [-30, -30, -35]
A_ub = [[[6, 4, 12], [0.05, 0.1, 0.2], [1, 1, 1]]]
b_ub = [2500, 55, 450]
res = linprog(c, A_ub, b_ub, bounds=(0, None))
print(res)
print(f"valor optimo: {res.fun} \nX: {res.x}")

```

Los resultados son

```

con: array([], dtype=float64)
fun: -13937.499998061907
message: 'Optimization terminated successfully.'
nit: 7
slack: array([3.76087883e-07, 1.25000001e+00, 6.16648208e-08])
status: 0
success: True
x: array([5.8047989e-09, 3.6250000e+02, 8.7500000e+01])
valor optimo: -13937.499998061907
X: [5.8047989e-09 3.6250000e+02 8.7500000e+01]

```

S1 = Restricción de materia prima
S3 = Restricción del almacenaje
También se tiene que en la tabla:
S1 = -0.63
S3 = -27.5
Lo que significa que si incremento una unidad más en el recurso S3, la función objetivo aumentara en 27.5.

15.10

```
from scipy.optimize import linprog
#variables de la función objetivo
c = [0.5, 1, 1.2]
#variables de las restricciones(15x1 + 10x2 + 75x3)
A_ub = [[-29, -10, -5], [135, 180, 75]]
#variables de los valores(>=100)
b_ub = [-10, 100]
#si la función es 28x1 + 10x2 + 5x3 *x10
#se multiplica por -1 a la función y se colocan esos valores
```

```
from scipy.optimize import linprog
#variables de la función objetivo
c = [0.5, 1, 1.2]
#variables de las restricciones(15x1 + 10x2 + 75x3)
A_ub = [[-29, -10, -5], [135, 180, 75]]
#variables de los valores(>=100)
b_ub = [-10, 100]
#si la función es 28x1 + 10x2 + 5x3 *x10
#se multiplica por -1 a la función y se colocan esos valores

#solución
x0 = np.array([0, 0, 0])
slack = np.array([4.2521064e-16, 3.3560000e+01])
status = 0
success = True
x1 = np.array([5.0000000e+00, 2.5848887e-31, 2.40200078e-11])
value_optimal = 0.15000000000000002
t1 = 15.0000000e+00 2.5848887e-11 2.40200078e-11
```

Capítulo XVI

Aplicaciones en ingeniería: optimización

16.1

Del problema mostrado se puede trabajar con la siguiente ecuación

$$Q = \frac{1}{n} A_c R^{2/3} S^{1/2}$$

Where:

n = Coefficient of Manning roughness

A_c = Cross section area of channel

R = Hydraulic radius

S = Channel slope

Parametrizamos el problema a un modelo de programación lineal para poder resolverlo en solver

Consider the following variables from the textbook,

$n = 0.03$

$S = 0.0004$

$Q = 1 \text{ m}^3/\text{s}$

Thus, the solution of the total perimeter,

$$P = B + 2 \times H$$

Subject to

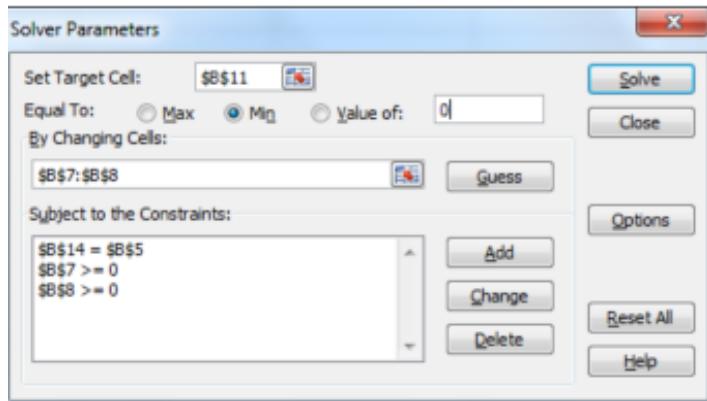
$$\frac{1}{n} BH \left(\frac{BH}{B + 2H} \right)^{2/3} S^{1/2} = Q$$

The problem can be set up and solved with the Excel Solver as in

Desarrollamos la tabla en Excel

	A	B
1		
2		
3	n	0.03
4	S	0.0004
5	Q	1
6		
7	B	1
8	H	1
9		
10	A_c	1
11	P	3
12	R	0.333333333
13		
14	Q_{manning}	0.320499905

Resolvemos usando el complemento en solver y seleccionamos las celdas pertinentes al problema



Este nos produce los siguientes valores

The resulting solution is

	A	B
1		
2		
3	n	0.03
4	S	0.0004
5	Q	1
6		
7	B	2.135251318
8	H	1.067559138
9		
10	Ac	2.279507056
11	P	4.270369594
12	R	0.533796199
13		
14	Qmanning	0.999999999

El valor que minimiza el perímetro son $B = 2.13$ y $H = 1.07$

16.19

Consideramos la afirmación del texto, el coeficiente de comprensión es ostentado por el barril cilíndrico. Este es $P = 3000 \text{ Kg}$.

Luego,

That is,

$$P < P_c \quad \text{and} \quad V = \pi r^2 L = 0.075 \text{ m}^3$$

And the critical load is,

$$P_c = \frac{\pi^2 EI}{L^2}$$

Thus, solve

$$P < P_c$$

$$P < \frac{\pi^2 EI}{L^2}$$

$$P < \frac{\pi^2 E}{L^2} \left(\frac{\pi r^4}{4} \right)$$

Resolvemos las restricciones y formulamos mejor el problema

$$P < \frac{\pi^3 Er^4}{4L^2}$$

$$P < \frac{\pi^3 E \left(\frac{V}{\pi L} \right)}{4L^2}$$

$$P < \frac{\pi EV^2}{4L^4}$$

Further solve,

$$L^4 < \frac{\pi EV^2}{4P}$$

$$L < \left(\frac{\pi EV^2}{4P} \right)^{1/4}$$

$$L < \left(\frac{\pi (200 \times 10^9) (0.075)^2}{4 \times 3000 \times 10^3} \right)^{1/4}$$

Por ende L es máximo cuando V es mínimo

En ese sentido hallamos el radio despejando la ecuación del volumen del cilindro

$$L < 4.14267$$

And the volume of beam,

$$V = \pi r^2 L \\ = 0.075 \text{ m}^3$$

And the value of r ,

$$r = \sqrt{\frac{0.075}{\pi (4.14267)}} \\ = 0.075913$$

Thus, the largest height that can be utilized is $L = 4.14267 \text{ m}$ and the corresponding radius is

$$r = 0.075913 \text{ m}$$

	G7	f_1	f_2		
	A	B	C	D	E
1	Variables to be determined:				
2	Flow rate of Channel 1:		700000 m ³ /d		
3	Flow rate of Channel 2:		600000 m ³ /d		

The Absolute difference is shown in cell D4,

	D4	f_x	=ABS(D3-D2)		
	A	B	C	D	E
1	Variables to be determined:				
2	Flow rate of Channel 1:		700000 m ³ /d		
3	Flow rate of Channel 2:		600000 m ³ /d		
4	Absolute difference:		100000 m ³ /d		

Step 2:

Now, the Annual Maintenance Cost is calculated. Costs per year (Channel 1) in cell D8 is shown below,

	D8	f_x	=1.1*D2		
	A	B	C	D	E
1	Variables to be determined:				
2	Flow rate of Channel 1:		700000 m ³ /d		
3	Flow rate of Channel 2:		600000 m ³ /d		
4	Absolute difference:		100000 m ³ /d		
5	Channel 1+Channel 2:		1300000		
6					
7	Annual maintenance cost:				
8	Costs per year (Channel 1):		770000		

Note: The formula used here shown in the formula bar is actually $C = \$1.1 \times \text{Channel 1}$.

And costs per year (Channel 2) contained in cell D9 shown below,

	D9	f_x	=1.4*D3		
	A	B	C	D	E
1	Variables to be determined:				
2	Flow rate of Channel 1:		700000 m ³ /d		
3	Flow rate of Channel 2:		600000 m ³ /d		
4	Absolute difference:		100000 m ³ /d		
5	Channel 1+Channel 2:		1300000		
6					
7	Annual maintenance cost:				
8	Costs per year (Channel 1):		770000		
9	Costs per year (Channel 2):		840000		

	A	B	C	D	E
1	Variables to be determined:				
2	Flow rate of Channel 1:	700000	m^3/d		
3	Flow rate of Channel 2:	600000	m^3/d		
4	Absolute difference:	100000	m^3/d		
5	Channel 1+Channel 2:	1300000			
6					
7	Annual maintenance cost:				
8	Costs per year (Channel 1):	770000			
9	Costs per year (Channel 2):	840000			
10	Electric Power (Channel 1):	2800000			

Ahora procedemos a resolver con **Solver**:

Step 4:

Once all the arrangements are complete, **Solver** can be called by clicking on **Data** and the conditions set as shown below:



Click on **Solve** and the optimized results obtained are shown below:

Esto nos da los siguientes resultados

	A	B	C	D	E
1	Variables to be determined:				
2	Flow rate of Channel 1:	300000	m^3/d		
3	Flow rate of Channel 2:	200000	m^3/d		
4	Absolute difference:	100000	m^3/d		
5	Channel 1+Channel 2:	500000			
6					
7	Annual maintenance cost:				
8	Costs per year (Channel 1):	330000			
9	Costs per year (Channel 2):	280000			
10	Electric Power (Channel 1):	1200000			
11	Electric Power (Channel 2):	600000			
12	Irrigation revenue (Channel 1):	672000			
13	Irrigation revenue (Channel 2):	512000			
14	Total Cost:	3594000			
15					
16	Profit:	-2E+06			

16.25

```

In [1]: # Import required packages
import numpy as np
from scipy.optimize import minimize
from scipy.optimize import differential_evolution

# Define the objective function
model = np.zeros(2)
def objective(x):
    model[0] = x[0]**2 + 2*x[1]**2 - 2*x[0]*x[1]
    model[1] = x[0] + x[1] - 12
    model[2] = x[0] - 4*x[1] - 4
    return model

# Define the constraints
cons1 = {'type': 'ineq', 'fun': lambda x: 12 - x[0] - x[1]}
cons2 = {'type': 'ineq', 'fun': lambda x: 4 - x[0] + 4*x[1]}

# Solving the model
optm = differential_evolution(objective, bounds=[(-10, 10), (-10, 10)])
results = optm.optimize()

print(results)
print(results.fun)
print(results.message)
print(results.success)
print(results.x)
print(results.x[0])
print(results.x[1])

```

16.26

Step 1 of 3:

Consider the statement from the textbook, the total drag on the airfoil is

$$D = 0.01\sigma V^2 + \frac{0.95}{\sigma} \left(\frac{W}{V}\right)^2$$

Where D is drag, σ is ratio of air density between the flight altitude and sea level, W is weight, and V is velocity.

Comment:

(a)

The minimum drag at $\frac{dD}{dV} = 0$. That is,

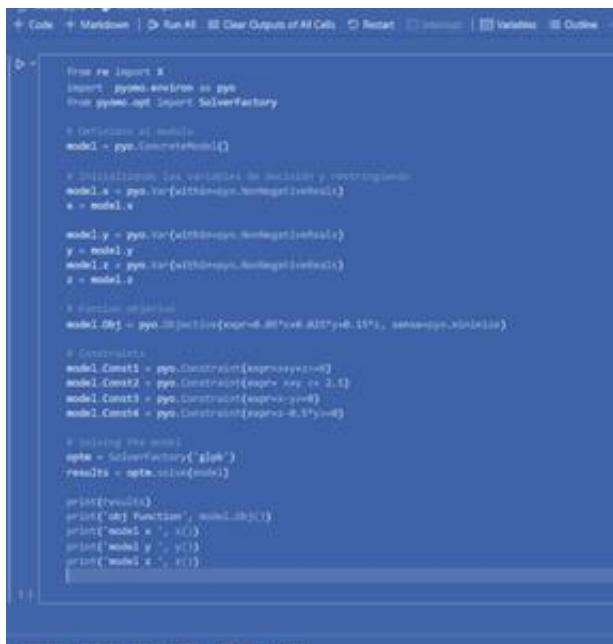
$$\begin{aligned} \frac{d}{dV} \left[0.01\sigma V^2 + \frac{0.95}{\sigma} \left(\frac{W}{V}\right)^2 \right] &= 0 \\ 0.02\sigma V + \frac{0.95W^2}{\sigma} \left(-2\frac{1}{V^3}\right) &= 0 \\ 0.02\sigma V - \frac{1.9W^2}{\sigma} \frac{1}{V^3} &= 0 \end{aligned}$$

Further solve,

$$\begin{aligned} V^4 &= \frac{1.9W^2}{0.02\sigma^2} \\ V &= \left(\frac{1.9W^2}{0.02\sigma^2}\right)^{1/4} \\ &= \left(\frac{1.9(16000)^2}{0.02(0.6)^2}\right)^{1/4} \\ &= 509.8181 \end{aligned}$$

	A	B	C
7			
8			
9	W	V	D
10	12000	441.5154	2339.231
11	13000	459.5438	2534.167
12	14000	476.8912	2729.102
13	15000	493.6293	2924.038
14	16000	509.8181	3118.974
15	17000	525.5085	3313.91
16	18000	540.7438	3508.846
17	19000	555.5614	3703.782
18	20000	569.994	3898.718

16.29



```

from pyomo.environ import *
from pyomo.environ import SolverFactory
model = pyo.ConcreteModel()

# Initialize the variables of decision y, representing
model.x = pyo.Var(within=NonNegativeReals)
x = model.x

model.y = pyo.Var(within=NonNegativeReals)
y = model.y
model.z = pyo.Var(within=NonNegativeReals)
z = model.z

# Linear objective
model.OBJ = pyo.Objective(expr=8.00*x+6.00*y+6.15*z, sense=minimize)

# Constraints
model.Constraint1 = pyo.Constraint(expr=x+y+z=1)
model.Constraint2 = pyo.Constraint(expr=x >= 2.0)
model.Constraint3 = pyo.Constraint(expr=y>z)
model.Constraint4 = pyo.Constraint(expr=x-0.3*y>0)

# Solving the model
opt = SolverFactory('glpk')
results = opt.solve(model)

print(results)
print("Optimal Number", model.OBJ())
print("Model x = ", x())
print("Model y = ", y())
print("Model z = ", z())

```