Segundo Examen Parcial

Luis Eduardo Robles Jiménez.
0224969

1. *Resuelve el siguiente sistema de ecuaciones que se representa en su forma matricial, usando el método de Jacobi o Gauss-Seidel, escribe la fórmula para tus variables, y realiza al menos 3 iteraciones.*

$$\begin{pmatrix} -5 & 5 & 3 \\ 5 & 6 & 1 \\ 3 & 1 & 7 \end{pmatrix}\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

$$\begin{pmatrix} -5x & 5y & 3z \\ 5x & 6y & z \\ 3x & y & 7z \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

$$\begin{pmatrix} x = \dfrac{1 - 5y - 3z}{-5} \\ y = \dfrac{2 - 5x - z}{6} \\ z = \dfrac{3 - 3x - y}{7} \end{pmatrix}$$

```
3x3 System:
        [-5 5 3 = 1]
        [5 6 1 = 2]
        [3 1 7 = 3]

x[0] = [-0.2, 0.5, 0.44285714285714284]
x[1] = [0.5657142857142856, -0.21190476190476185, 0.21639455782312927]
x[2] = [-0.2820680272108843, 0.5323242630385487, 0.4734114026563006]
x[3] = [0.616371104632329, -0.259211154302991, 0.20144254862942915]
x[4] = [-0.33834562512533356, 0.5817142628328732, 0.4904746589347325]
x[5] = [0.6759990581937128, -0.3117449916505494, 0.18339254529563015]
x[6] = [-0.40170946447317135, 0.6375257961783711, 0.5096575138915919]
x[7] = [0.7433203045133262, -0.37104317274303716, 0.16301175131472262]
x[8] = [-0.4732361219542036, 0.7005281430760492, 0.5313114603980802]
x[9] = [0.8193150193148973, -0.4379810928287612, 0.14000514784058135]
x[10] = [-0.5539780041244124, 0.7716474787969135, 0.5557552190823319]
x[11] = [0.9051006102463126, -0.5135430450523158, 0.11403445918762545]
x[12] = [-0.6451223695397407, 0.8519295647518463, 0.5833482205524823]
x[13] = [1.0019384970833358, -0.598840117661527, 0.0847178037730742]
```

¡EL MÉTODO DIVERGE!

2. *Construye polinomio de aproximación de Hermite para los siguientes datos.*

x	f(x)	f'(x)
8.3	17.5649	3.1162
8.6	18.5051	3.1517

```
2 points:
	f(8.3) = 17.5649		f'(8.3) = 3.1162
	f(8.6) = 18.5051		f'(8.6) = 3.1517

Polynomial
17.5649 + 3.1162*(x-8.3) + 0.05933333333334033*(x-8.3)*(x-8.3) + -0.0011111111111578535*(x-8.3)*(x-8.3)*(x-8.6)

Simplified
-0.0011111111111578535*x**3 + 0.08733333333451824*x**2 + 1.8960999999899908*x - 3.5538044444162686

By Powers
-0.0011111111111578535*x**3 + 0.0873333333334518238*x**2 + 1.8960999999899908*x - 3.5538044444162685
```

3. *Considera la siguiente ecuación:* $2 + \frac{x}{2} - \frac{x^2}{4} = 0$.
   a. *Manipula algebraicamente y escribe al menos dos funciones para usar la iteración de punto fijo.*
      i. $2 + \frac{3x}{2} - \frac{x^2}{4} = x$
      ii. $\frac{x}{2} = \frac{x^2}{4} - 2; \qquad x = \frac{x^2}{2} - 4$
   b. *Usa* $g(x) = 2 + \frac{x}{2} - \frac{x^2}{4}$ *para aproximar una raíz tomando* $x_0 = 1$.

```
x = -x**2/4 + x/2 + 2

1. P = 2.25000000000000 Er = 55.5555555555556
2. P = 1.85937500000000 Er = 21.0084033613445
3. P = 2.06536865234375 Er = 9.97369898637666
4. P = 1.96624740865082 Er = 5.04113791869909
5. P = 2.01659148631890 Er = 2.49649361358650
6. P = 1.99163543748598 Er = 1.25304301998225
7. P = 2.00416478978049 Er = 0.625165772714831
8. P = 1.99791326874127 Er = 0.312902523699613
9. P = 2.00104227701753 Er = 0.156368923944784
10. P = 1.99947858990589       Er = 0.0782047439531863
11. P = 2.00026063707993       Er = 0.0390972636037272
12. P = 1.99986966447711       Er = 0.0195499041644994
13. P = 2.00006516351461       Er = 0.00977463339998605
14. P = 1.99996741718113       Er = 0.00488739629660346
15. P = 2.00001629114403       Er = 0.00244367823990332
16. P = 1.99999185436164       Er = 0.00122184409590849
17. P = 2.00000407280259       Er = 0.000610920803836767
```

2.00000407280259

4. *Obtén el polinomio interpolante de diferencias divididas de Newton usando los siguientes datos: (2, -15); (-3, 15); (5, -153); (-7, 291)*
   Úsala para aproximar el valor si $x = -4$
   Nota: *Escribe el polinomio interpolante y el valor cuando* $x = -4$

```
4 points:
        f(2) = -15
        f(-3) = 15
        f(5) = -153
        f(-7) = 291

Polynomial
-15 + -6.0*(x-2) + -5.0*(x-2)*(x--3) + -1.0*(x-2)*(x--3)*(x-5)

Simplified
-1.0*x**3 - 1.0*x**2 - 3.0

By Powers
-1.0*x**3 - 1.0*x**2 - 3.0

f(-4) ≈ 45.0000000000000
```

5. *Obtén el polinomio de interpolación de Lagrange usando los siguientes datos:(1, 10); (-4, 10); (-7, 34).*
   Y úsalo para aproximar $p(-3)$.
   ***Nota**: Escribe el polinomio y la interpolación cuando* $x = -3$.

```
3 points:
        f(1) = 10
        f(-4) = 10
        f(-7) = 34

Polynomial
10*(x - -4)/(1 - -4)*(x - -7)/(1 - -7) + 10*(x - 1)/(-4 - 1)*(x - -7)/(-4 - -7) + 34*(x - 1)/(-7 - 1)*(x - -4)/(-7 - -4)

Simplified
x**2 + 3*x + 6

By Powers
x**2 + 3*x + 6

g(-3) ≈ 6.00000000000000
```

C O D I G O S

Aproximaciones

Luis Eduardo Robles Jimenez

0224969

Function

In [1]:
```
#expression = "sqrt(1 + 1/x) - x"
#expression = "x**3+5*x**2+2"
#expression = "2*sin(sqrt(x))-x"
expression = "2 - x/2 -x**2/4"
```

Method

In [ ]:
```
NewtonRaphson(-5, 0.001, 50)
```

In [ ]:
```
BinarySearch(0, 3, 0.1, 50)
```

In [ ]:
```
Secant(0, 1, 0.01, 10)
```

In [6]:
```
FixedP(1, 0.001, 100)
```

```
        f(x) = -x**2/4 - x/2 + 2
        x = -x**2/4 + x/2 + 2

1. P = 2.25000000000000 Er = 55.5555555555556
2. P = 1.85937500000000 Er = 21.0084033613445
3. P = 2.06536865234375 Er = 9.97369898637666
4. P = 1.96624740865082 Er = 5.04113791869909
5. P = 2.01659148631890 Er = 2.49649361358650
6. P = 1.99163543748598 Er = 1.25304301998225
7. P = 2.00416478978049 Er = 0.625165772714831
8. P = 1.99791326874127 Er = 0.312902523699613
9. P = 2.00104227701753 Er = 0.156368923944784
10. P = 1.99947858990589        Er = 0.0782047439531863
11. P = 2.00026063707993        Er = 0.0390972636037272
12. P = 1.99986966447711        Er = 0.0195499041644994
13. P = 2.00006516351461        Er = 0.00977463339998605
14. P = 1.99996741718113        Er = 0.00488739629660346
15. P = 2.00001629114403        Er = 0.00244367823990332
16. P = 1.99999185436164        Er = 0.00122184409590849
17. P = 2.00000407280259        Er = 0.000610920803836767
```

Out[6]: 2.00000407280259

Newton Raphson

Newton Raphson

In [ ]:
```python
def NewtonRaphson(p0, e, n):
    f = parse_expr(expression)
    d = diff(f, x)
    print("\tf(x) =", f, "\n\tf'(x) =", d, "\n")
    for i in range(n):
        p = p0 - N(f.subs(x, p0))/N(d.subs(x, p0))
        error = abs(N((p - p0)/p))*100
        print(i + 1, ". ", sep = '', end = '')
        print("P =", p, "\tEr =", error)
        if error < e: return p
        p0 = p
    return p
```

Binary Search

In [ ]:
```python
def BinarySearch(a, b, e, n):
    f = parse_expr(expression)
    print("\tf(x) = ", f, "\n\t[", a, ", ", b, "]", "\n", sep = "")
    fp0, p0 = N(f.subs(x, a)), a
    for i in range(n):
        p = a + (b - a)/2
        fp = N(f.subs(x, p))
        error = abs((p - p0)/p)*100
        print(i + 1, ". ", sep = '', end = '')
        print("P = ", p, "\tEr = ", error, " %", sep = '')
        if error < e: return p
        if fp * fp0 > 0: a, fp0 = p, fp
        else: b = p
        p0 = p
    return p
```

Secant

In [ ]:
```python
def Secant(pa, pb, e, n):
    f = parse_expr(expression)
    print("\tf(x) =", f, "\n")
    for i in range(n):
        qa, qb = N(f.subs(x, pa)), N(f.subs(x, pb))
        pc = pb - qb*(pa - pb)/(qa - qb)
        error = abs(N((pc - pb)/pc))*100
        print(i + 1, ". ", sep = '', end = '')
        print("P =", pc, "\tEr =", error)
        if error < e: return pc
        pa, pb = pb, pc
    return p
```

Fixed Point

In [4]:
```
def FixedP(pa, e, n):
    f = parse_expr(expression)
    print("\tf(x) =", f)
    f = parse_expr(expression + " + x")
    print("\tx =", f, "\n")
    for i in range(n):
        pb = N(f.subs(x, pa))
        if not pb: return pa
        error = abs((pb - pa)/pb)*100
        print(i + 1, ". ", sep = '', end = '')
        print("P = ", pb, "\tEr = ", error, sep = '')
        if error < e: return pb
        pa = pb
    return pb
```

Run First

In [3]:
```
from sympy import *
x = symbols("x")
```

Aproximaciones

Luis Eduardo Robles Jimenez

0224969

Input

In [11]:
```
#xInput = (1, 4, 6, 5)
#yInput = "ln(x)"
#xInput = (1, 4, 6)
#yInput = "ln(x)"
#xInput = (1.0, 1.3, 1.6, 1.9, 2.2)
#yInput = (0.765197, 0.6200860, 0.4554022, 0.2818186, 0.1103623)
#xInput = (1.0, 1.3, 1.6)
#yInput = (0.7651977, 0.6200860, 0.4554022)
#xInput = (8.1, 8.3, 8.6, 8.7)
#yInput = (16.94410, 17.56492, 18.50515, 18.82091)
#xInput = (1.3, 1.6, 1.9)
#yInput = (0.6200860, 0.4554022, 0.2818186)
#dInput = (-0.5220232, -0.5698959, -0.5811571)
#xInput = (8.3, 8.6)
#yInput = (17.56492, 18.50515)
#dInput = (3.116256, 3.151762)
#xInput = (0, 0.6, 0.9)
#yInput = "ln(x+1)"
#xInput = (8, 9, 11)
#yInput = "log(x, 10)"
#xInput = (8, 9, 11)
#yInput = Cloud(xInput, "log(x, 10)")
#dInput = Cloud(xInput, "1/(x*log(10))")

#HERMITE SEGUNDO EXAMEN
#xInput = (8.3, 8.6)
#yInput = (17.5649, 18.5051)
#dInput = (3.1162, 3.1517)

#NEWTON SEGUNDO EXAMEN
#xInput = (2, -3, 5, -7)
#yInput = (-15, 15, -153, 291)

#LAGRANGE SEGUNDO EXAMEN
xInput = (1, -4, -7)
yInput = (10, 10, 34)
```

Method

In [14]:
```
f, e = Lagrange(xInput, yInput), -3
print("\ng(", e, ") ≈ ", N(f.subs(x, e)), sep = "")
```

```
3 points:
        f(1) = 10
        f(-4) = 10
        f(-7) = 34

Polynomial
10*(x - -4)/(1 - -4)*(x - -7)/(1 - -7) + 10*(x - 1)/(-4 - 1)*(x - -7)/(-4 -
-7) + 34*(x - 1)/(-7 - 1)*(x - -4)/(-7 - -4)

Simplified
x**2 + 3*x + 6

By Powers
x**2 + 3*x + 6

g(-3) ≈ 6.00000000000000
```

In [8]:
```
f, e = Newton(xInput, yInput), -4
print("\nf(", e, ") ≈ ", N(f.subs(x, e)), sep = "")
```

```
4 points:
        f(2) = -15
        f(-3) = 15
        f(5) = -153
        f(-7) = 291

Polynomial
-15 + -6.0*(x-2) + -5.0*(x-2)*(x--3) + -1.0*(x-2)*(x--3)*(x-5)

Simplified
-1.0*x**3 - 1.0*x**2 - 3.0

By Powers
-1.0*x**3 - 1.0*x**2 - 3.0

f(-4) ≈ 45.0000000000000
```

In [5]:
```
f, e = Hermite(xInput, yInput, dInput), 10
#print("\nf(", e, ") ≈ ", N(f.subs(x, e)), sep = "")
```

```
2 points:
        f(8.3) = 17.5649        f'(8.3) = 3.1162
        f(8.6) = 18.5051        f'(8.6) = 3.1517

Polynomial
17.5649 + 3.1162*(x-8.3) + 0.05933333333334033*(x-8.3)*(x-8.3) + -0.00111111
11111578535*(x-8.3)*(x-8.3)*(x-8.6)

Simplified
-0.00111111111111578535*x**3 + 0.0873333333451824*x**2 + 1.8960999999899908*
x - 3.5538044444162686

By Powers
-0.00111111111111578535*x**3 + 0.0873333333334518238*x**2 + 1.8960999999899908
*x - 3.5538044444162685
```

Lagrange

In [13]:
```
def Lagrange(xInput, yInput, p = None):
    yInput, n, s = Cloud(xInput, yInput), len(xInput), ""
    print(n, "points:")
    for i in range(n): print("\tf(", xInput[i], ") = ", yInput[i], sep = "")
    for i in range(n):
        p = str(yInput[i])
        for j in range(n):
            if i != j:
                p += "*(x - " + str(xInput[j]) + ")/(" + str(xInput[i]) + " -
        s += (" + " if i else "") + p
    return showPoly(s)
```

Newton's Polynomial

In [7]:
```
def Newton(xInput, yInput):
    yInput = Cloud(xInput, yInput)
    n = len(xInput)
    print(n, "points:")
    for i in range(n): print("\tf(", xInput[i], ") = ", yInput[i], sep = "")
    m = [[0 for i in range(n)] for j in range(n)]
    for i in range(n): m[i][0] = yInput[i]
    for j in range(1, n):
        for i in range(n - j):
            m[i][j] = (m[i+1][j-1] - m[i][j-1])/(xInput[i+j] - xInput[i])
    r, a = str(m[0][0]), ""
    for i in range(1, n):
        a += "*" + "(x-" + str(xInput[i - 1]) + ")"
        r += " + " + str(m[0][i]) + a
    return showPoly(r)
```

Hermite

In [4]:
```
def Hermite(xInput, yInput, dInput):
    n = len(xInput)
    print(n, "points:")
    for i in range(n):
        print("\tf(", xInput[i], ") = ", yInput[i], "\tf'(", xInput[i], ") =
    m = [[0 for i in range(2*n)] for j in range(2*n)]
    for i in range(n):
        m[2*i][0] = m[2*i+1][0] = yInput[i]
        m[2*i][1] = dInput[i]
        if i: m[2*i-1][1] = (m[2*i][0]-m[2*i-1][0])/(xInput[i]-xInput[i-1])
    for j in range(2, 2*n):
        for i in range(2*n-j):
            m[i][j] = (m[i+1][j-1] - m[i][j-1])/(xInput[int((i+j)/2)] - xInpu
    r, a = str(m[0][0]), ""
    for i in range(1, 2*n):
        a += "*" + "(x-" + str(xInput[int((i - 1)/2)]) + ")"
        r += " + " + str(m[0][i]) + a
    return showPoly(r)
```

AuxFucnt

In [3]:
```
def Cloud(xI, yI):
    if isinstance(yI, str):
        a, yI = list(), parse_expr(yI)
        for xVal in xI: a.append(N(yI.subs(x, xVal)))
        yI = tuple(a)
    return yI
def showPoly(s):
    print("\nPolynomial", s, sep = "\n")
    print("\nSimplified", simplify(parse_expr(s)), sep = "\n")
    print("\nBy Powers", r := collect(expand(parse_expr(s)), x), sep = "\n")
    return r
```

Run First

In [2]:
```
from sympy import *
x = symbols("x")
```

Aproximaciones

Luis Eduardo Robles Jimenez

0224969

Input

In [67]:
```
Matrix = [
    [-5, 5, 3],
    [5, 6, 1],
    [3, 1, 7]
]
Independent = [1, 2, 3]
```

Method

In [68]:
```
x = GaussSeidel(Matrix, Independent, 2000, 0.00000000001)
print(x)
```

```
3x3 System:
        [-5 5 3 = 1]
        [5 6 1 = 2]
        [3 1 7 = 3]

x[0] = [-0.2, 0.5, 0.44285714285714284]
x[1] = [0.5657142857142856, -0.21190476190476185, 0.21639455782312927]
x[2] = [-0.2820680272108843, 0.5323242630385487, 0.4734114026563006]
x[3] = [0.616371104632329, -0.259211154302991, 0.20144254862942915]
x[4] = [-0.3383456251253356, 0.5817142628328732, 0.4904746589347325]
x[5] = [0.6759990581937128, -0.3117449916505494, 0.18339254529563015]
x[6] = [-0.40170946447317135, 0.6375257961783711, 0.5096575138915919]
x[7] = [0.7433203045133262, -0.37104317274303716, 0.16301175131472262]
x[8] = [-0.4732361219542036, 0.7005281430760492, 0.5313114603980802]
x[9] = [0.8193150193148973, -0.4379810928287612, 0.14000514784058135]
x[10] = [-0.5539780041244124, 0.7716474787969135, 0.5557552190823319]
x[11] = [0.9051006102463126, -0.5135430450523158, 0.11403445918762545]
x[12] = [-0.6451223695397407, 0.8519295647518463, 0.5833482205524823]
x[13] = [1.0019384970833358, -0.598840117661527, 0.0847178037730742]
```

Gauss-Seidel

In [32]:
```
def GaussSeidel(m, it, n, e):
    print(len(m), "x", len(m), " System:\n", sep = "", end = "")
    for i in range(len(m)):
        print("\t[", end = "")
        for j in range(len(m[i])):
            print(m[i][j], end = " ")
        print("= ", it[i], "]", sep = "")
    print()
    x = [0 for _ in range(len(m))]
    for i in range(len(m)):
        d = m[i][i]
        for j in range(len(m[i])):
            m[i][j] /= d
        it[i] /= d

    for i in range(len(m)):
        s = it[i]
        for j in range(len(m[i])):
            if i != j:
                s -= m[i][j]*x[j]
        x[i] = s
    for _ in range(n):
        print("x[", _, "] = ", x, sep = "")
        c = 1
        for i in range(len(m)):
            o = x[i]
            s = it[i]
            for j in range(len(m[i])):
                if i != j:
                    s -= m[i][j]*x[j]
            x[i] = s
            if c and x[i]:
                error = 100*abs((x[i]-o)/x[i])
                if error > e:
                    c = 0
        if c: break;
    return x
```