Aproximaciones

Luis Eduardo Robles Jimenez

0224969

Input

In [6]:
```
#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 = (8, 9, 11)
#yInput = Cloud(xInput, "log(x)")
#dInput = Cloud(xInput, "1/x")
#yInput = (-15, 15, -153, 291)

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

#TERCER PARCIAL
#Lagrange
```

```
xInput = (8.1, 8.3, 8.6)
yInput = (16.944410, 17.56492, 18.50515)
```

Method

In [7]:
```
f, e = Lagrange(xInput, yInput), 8.4
print("\ng(", e, ") ≈ ", N(f.subs(x, e)), sep = "")
```

```
3 points:
        f(8.1) = 16.94441
        f(8.3) = 17.56492
        f(8.6) = 18.50515

Polynomial
16.94441*(x - 8.3)/(8.1 - 8.3)*(x - 8.6)/(8.1 - 8.6) + 17.56492*(x - 8.1)/(8.3 - 8.1)*(x - 8.6)/(8.3 - 8.6) + 18.5
0515*(x - 8.1)/(8.6 - 8.1)*(x - 8.3)/(8.6 - 8.3)

Simplified
0.0631000000000768*x**2 + 2.06770999999844*x - 3.94403199999579

By Powers
0.0631000000000768*x**2 + 2.06770999999844*x - 3.94403199999579

g(8.4) ≈ 17.8770679999965
```

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

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

Lagrange

In [5]:
```
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]) + " - " + str(xInput[j]) + ")"
        s += (" + " if i else "") + p
    return showPoly(s)
```

Newton's Polynomial

In [4]:
```
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 [3]:
```
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], ") = ", dInput[i], sep = "")
    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)] - xInput[int(i/2)])
    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 [2]:
```
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 [1]:
```
from sympy import *
x = symbols("x")
```