Aproximaciones

Luis Eduardo Robles Jimenez

0224969

Function

In [7]:
```
#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"
#expression = "2*x**3 - 11.7*x**2 + 17.7*x - 5"
expression = "x**2-6"
```

Method

In [8]:
```
NewtonRaphson(1, 0.001, 50)
```

```
        f(x) = x**2 - 6
        f'(x) = 2*x

1. P = 3.50000000000000         Er = 71.4285714285714
2. P = 2.60714285714286         Er = 34.2465753424658
3. P = 2.45425636007828         Er = 6.22944283863252
4. P = 2.44949437160697         Er = 0.194406997889468
5. P = 2.44948974278755         Er = 0.000188970761244172
```

Out[8]: 2.44948974278755

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

In [ ]:
```
Secant(3, 4, 0.01, 100)
```

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

Newton Raphson

In [5]:
```
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 [4]:
```
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 [3]:
```
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 [2]:
```
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 [1]:
```
from sympy import *
x = symbols("x")
```