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 [ ]:
```
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 [ ]:
```
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 [ ]:
```
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")
```