Find minamun of f(x) using Newton's

① Fail if start guess close to max

② $f''(x_i)=0$ ③ Require two func call.

④ hot same rate as golden

Compare converage

① Bisection has linear if f(x) contionus. $f(a)f(b)<0$

② Newton fastest ③ Secent superlinear, low cost

④ Newton quardic convergence when close to root

```python
def hessian(f, X):
    x0, y0 = X[0, 0], X[1,0]
    x, y = Symbol('x'), Symbol('y')
    A = Matrix([f])
    B = Matrix([x, y])
    Hessian = A.jacobian(B).jacobian(B).subs({x:x0, y:y0})
    return np.array(Hessian).astype(np.float64)

def gradient(f, X):
    x0, y0 = X[0, 0], X[1,0]
    x, y = Symbol('x'), Symbol('y')
    A = Matrix([f])
    B = Matrix([x, y])
    Gradient = A.jacobian(B).subs({x:x0, y:y0})
    return np.array(Gradient).astype(np.float64).T

def newtons_method(f, x_init, tol):
    x_new = x_init
    x_prev = np.random.randn(x_init.shape[0])
    cnt = 0
    while(la.norm(gradient(f, x_new)) > tol):
        x_prev = x_new
        print(x_prev)
        s = -la.solve(hessian(f, x_prev), gradient(f, x_prev))
        x_new = x_prev + s
        print(x_new)
        cnt += 1
    return x_new, cnt
```

S=-grad

Jacob $O(n^2)$ solving non-linear $O(n^3)$

Hessian ≠ $O(n^3)$

OPtND Compare Method.

① Linear search $\nabla f(x_{k+1}) \perp \nabla f(x_k)$ ✓

② Steepest Descent move next $-\nabla f$ ✓

③ NT opt $O(n^2)$ ✗

④ Can eva without $f''(x)$ ✗

⑤ line search = NT method ✗

Newton opti

$x_0 = x_0 - \frac{f'(x_0)}{f''(x_0)}$ 一维.

求 root

$x_1 = x_0 - \frac{f(x_0)}{f(x)'}$

找 f(x,y) 根. Netwon.

f(x,y) J(x,y)

S= la.solve (J, -f)

$x_1 = x_0$ + S.flatten( )

grad

$[f(x+h) - y]/h$

h=np.zeros(n)

h[i] = 0

compare computational cost

① Secant require previous

② Secant = biscent

③ Iter ↑ Netwon most cost.

④ bis didn't use 2

⑤ Netwon doesn't use prev.

One step Secant

$x_1 = 0$, $x_0 = 2$

$d = \frac{f(x_1)-f(x_0)}{x_1-x_0} = f(x_1)'$

$x_2 = x_1 - \frac{f(x_1)}{f(x_1)'}$

```python
import numpy as np
m1 = a
m2 = b
iteration_cts = []
ints = []

for i in taus:
    m1 = a
    m2 = b
    count = 0
    for j in range(max_iter):
        count+=1
        if(abs(m1-m2)<1e-5):
            break


        m11 = m1 + (1-i)*(m2-m1)
        m22 = m1 + (i)*(m2-m1)
        f1 = f(m11)
        f2 = f(m22)
        if f1 > f2:
            m1 = m11
            m2 = m2
        else:
            m1 =m1
            m2 = m22
    ints.append((m1,m2))
    iteration_cts.append(count-1)
minc = min(iteration_cts)
index = iteration_cts.index(minc)
best_tau = taus[index]
best_interval = ints[index]
iteration_cts = np.array(iteration_cts)
print(iteration_cts)
```

```
# Sample Carrying out Newton steps (n-dimensional)
x, y  = symbols('x y')
f = exp(8*x) + 6 * cos(y)
X0 = np.array([[0], [np.pi]])
H = hessian(f, X0)
g = gradient(f, X0)
s = -la.solve(H, g)
X_new = X0 + s
X_new
```

给x1,x2求最小.

```
# Sample: Determine the length of the interval after one iteration
L = -8
R = 10
length_0 = R - L
length_1 = length_0 * (np.sqrt(5) - 1)/2
length_1
```

bracket len

```
# Sample Newton Solve 2
x, y = symbols('x y')
x0 = np.array([-1, 1])
X = Matrix([4*x*y+3, x**3 + y**2 + 6])
Y = Matrix([x, y])
J = X.jacobian(Y)
J_ = np.array(J.subs({x:x0[0], y:x0[1]})).astype(np.float64)
X_ = np.array(X.subs({x:x0[0], y:x0[1]})).astype(np.float64)
S = la.solve(J_, -X_)
S.flatten()+x0
```

```
# Sample Perform Two Steps of Bisection
f = lambda x: (x-2.7)**5
L = -3
R = 8
f_L = f(L)
f_R = f(R)

for i in range(2):
    mid = (L + R)/2
    f_mid = f(mid)
    if sign(f_L) == sign(f_mid):
        L, f_L = mid, f_mid
        print(L, R)
    else:
        R, f_R = mid, f_mid
        print(L, R)
```

```
# Sample Perform One Step of Golden Section Search
def f(x):
    return (x-6.7)**2
gs = (np.sqrt(5) - 1) / 2
a = -3
b = 8
m1 = a + (1 - gs) * (b - a)
m2 = a + gs * (b - a)

# Begin your modifications below here
f1, f2 = f(m1), f(m2)
if f1>= f2:
    a = m1
else:
    b = m2
m1 = a + (1 - gs) * (b - a)
m2 = a + gs * (b - a)
if f1>= f2:
    f1 = f2
    f2 = f(m2)
else:
    f2 = f1
    f1 = f(m1)
print(a, b)
```

```
# Sample N-Dimension Optimization using Steepest Descent
x, y = symbols('x y')
f = 13*x**2 + 7*x*y + 13*y**2 + 13*sin(y)**2 + 7*cos(x*y)
X0 = np.array([[-3], [3]])
alpha = 0.05
s0 = -gradient(f, X0)
print(s0)
X1 = X0 + alpha*s0
X1
```

```
# Sample Carrying out Newton steps
x0 = 0.3
x = Symbol('x')
f = -exp(-x**2)
df = diff(f,x,1)
d2f = diff(f, x, 2)
x1 = x0 - (df/d2f).subs(x, x0)
x1
```

给一个x,求最小f(x).

```
# Finite Difference: Calculation R -> R
x = Symbol('x')
y = -log(x)
x0 = 0.1
h = 0.01
(y.subs(x, x0+h) - y.subs(x, x0-h))/2/h
```

小step 用 central

→ central

```
# Sample Determine the length of the interval
x = Symbol('x')
f = (x - 5) ** 3
L = -12
R = 12
iteration = 3

for i in range(iteration):
    mid = (L + R) /2
    fL = f.subs(x,L)
    fR = f.subs(x,R)
    fmid = f.subs(x,mid)
    if fL * fmid <= 0:
        R = mid
    else:
        L = mid
R - L
```

Interval length.

$\frac{(R-L)}{2^k}$

```
# Sample Newton Solve
x, y = symbols('x y')
A = np.array([[3, -2, 2, 1, 3, -3],
              [-1, 3, -1, 0, -1, 1]])
X = np.array([x**2, x, 1, y**2, y, x*y])
X0 = np.array([-2, -1])

f = Matrix(A @ X)
Y = Matrix([x, y])
J = (f.jacobian(Y))

J_ = np.array(J.subs({x:X0[0], y:X0[1]})).astype(np.float64)
print(J_)
f_ = np.array(f.subs({x:X0[0], y:X0[1]})).astype(np.float64)

S = la.solve(J_, -f_)
S.flatten()+X0
```

```
x = Symbol('x')
f = x**3 - 4*x -7
x0 = 0
df = diff(f)
for i in range(2):
    # x_new = x0 - (f/df).subs(x,x0)
    # x0 = np.float(x_new)
    # print(x0)
    x0 -= float((f/df).subs(x, x0))
    print(x0)
```

两步牛顿求根

Finite Difference grad

```
# Sample Finite Difference Gradient
x, y, z = symbols('x y z')
X0 = np.array([1, 1, 1])
phi = Matrix([x**2 * y + x + y * z**2])
h = 0.1

phi0 = np.array(phi.subs({x:X0[0], y:X0[1], z:X0[2]})).astype(np.float64)
for i in range(3):
    a = np.zeros(3)
    a[i] = h
    X = X0 + a
    phix = np.array(phi.subs({x:X[0], y:X[1], z:X[2]})).astype(np.float64)
    print((phi0-phix)/h)
```

backward.

Forward: $X = X_0 - a$

F: - res.