

rosenbrock

May 19, 2024

1 Rosenbrock

1.0.1 Should converge to 1.0, 1.0, 1.0, 1.0, 1.0

```
[1]: import numpy as np
      from scipy.optimize import minimize

      def rosen(x):
          """The Rosenbrock function"""
          return sum(100.0*(x[1:]-x[:-1]**2.0)**2.0 + (1-x[:-1])**2.0)

      x0 = np.array([1.3, 0.7, 0.8, 1.9, 1.2])
```

1.0.2 Nelder-Mead

```
[2]: res = minimize(rosen, x0, method='nelder-mead', options={'xatol': 1e-8, 'disp':
      ↪True})
      print(res.x)
```

```
Optimization terminated successfully.
      Current function value: 0.000000
      Iterations: 339
      Function evaluations: 571
[1.  1.  1.  1.  1.]
```

1.0.3 Powell

```
[3]: res = minimize(rosen, x0, method='Powell', options={'disp': True})
      print(res.x)
```

```
Optimization terminated successfully.
      Current function value: 0.000000
      Iterations: 18
      Function evaluations: 988
[1.  1.  1.  1.  1.]
```

1.0.4 Rosenbrock derivative function

```
[4]: def rosen_der(x):  
    xm = x[1:-1]  
    xm_m1 = x[:-2]  
    xm_p1 = x[2:]  
    der = np.zeros_like(x)  
    der[1:-1] = 200*(xm-xm_m1**2) - 400*(xm_p1 - xm**2)*xm - 2*(1-xm)  
    der[0] = -400*x[0]*(x[1]-x[0]**2) - 2*(1-x[0])  
    der[-1] = 200*(x[-1]-x[-2]**2)  
    return der
```

1.0.5 Congruent Gradient

```
[5]: res = minimize(rosen, x0, method='CG', options={'disp': True})  
print(res.x)
```

Optimization terminated successfully.
Current function value: 0.000000
Iterations: 65
Function evaluations: 804
Gradient evaluations: 134
[0.99999826 0.99999652 0.99999303 0.99998604 0.99997204]

1.0.6 Congruent Gradient with derivative

```
[6]: res = minimize(rosen, x0, method='CG', jac=rosen_der, options={'disp': True})  
print(res.x)
```

Optimization terminated successfully.
Current function value: 0.000000
Iterations: 47
Function evaluations: 97
Gradient evaluations: 97
[0.99999977 0.99999954 0.99999909 0.99999819 0.99999636]

1.0.7 BFGS - Broyden-Fletcher-Goldfarb-Shannon

```
[7]: res = minimize(rosen, x0, method='BFGS', options={'disp': True})  
print(res.x)
```

Optimization terminated successfully.
Current function value: 0.000000
Iterations: 25
Function evaluations: 180
Gradient evaluations: 30
[0.99999925 0.99999852 0.99999706 0.99999416 0.99998833]

1.0.8 BFGS with derivative

```
[8]: res = minimize(rosen, x0, method='BFGS', jac=rosen_der, options={'disp': True})
print(res.x)
```

Optimization terminated successfully.

Current function value: 0.000000

Iterations: 25

Function evaluations: 30

Gradient evaluations: 30

[1.00000004 1.0000001 1.00000021 1.00000044 1.00000092]

1.0.9 L-BFGS-B, doesn't get the message Optimization terminated successfully

```
[9]: res = minimize(rosen, x0, method='L-BFGS-B', options={'disp': True})
print(res) # The result is different because it gets an error message
```

message: CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH

success: True

status: 0

fun: 1.5040593675501344e-11

x: [1.000e+00 1.000e+00 1.000e+00 1.000e+00 1.000e+00]

nit: 24

jac: [3.726e-08 -1.265e-05 1.180e-05 -1.227e-05 5.970e-06]

nfev: 156

njev: 26

hess_inv: <5x5 LbfgsInvHessProduct with dtype=float64>

1.0.10 L-BFGS-B with derivative

```
[10]: res = minimize(rosen, x0, method='L-BFGS-B', jac=rosen_der, options={'disp':
↳True, 'ftol':1e-7, 'gtol':1e-7})
print(res)
```

message: CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH

success: True

status: 0

fun: 5.458172231314972e-09

x: [1.000e+00 1.000e+00 1.000e+00 1.000e+00 1.000e+00]

nit: 22

jac: [-2.126e-03 -4.086e-04 1.104e-03 -1.355e-03 5.613e-04]

nfev: 24

njev: 24

hess_inv: <5x5 LbfgsInvHessProduct with dtype=float64>

```
[11]: x = x0
x
```

```
[11]: array([1.3, 0.7, 0.8, 1.9, 1.2])
```

1.0.11 Testing the rosen_der derivative function

```
[12]: rosen_der(x0)
```

```
[12]: array([ 515.4, -285.4, -341.6, 2085.4, -482. ])
```

1.0.12 Gradient Descent - converges slowly if at all

```
[13]: alpha = 1e-3                                # learning rate, 1e-2 is too high
      beta = 0.4                                    # momentum
      mse_tol = 1e-10                               # mean squared error goal
      x = x0                                         # find the values of x that minimize
      ↪ rosen(x)
      i = 0
      step = np.zeros_like(x)
      mse = rosen(x)
      print(f"Initial MSE = {mse:14.10f}      x = {x}")    # print initial values
      while mse > mse_tol:
          grad = rosen_der(x)
          gnorm = np.linalg.norm(grad)                # gradient norm
          step = -(1-beta)*alpha*grad+beta*step
          x += step
          mse = rosen(x)                              # update the measn squared error
          i += 1
      print(f"iterations = {i:6d}")
      print(f"Final MSE = {mse:14.10f}      x = {x}")
      print(f"gradient = {grad}")
      # [ print(f"grad[{str(j)}] = {grad[j]:8.6f}      ",end="") for j in
      ↪ range(len(grad))]
      print(f"gradient norm = {gnorm}")
```

```
Initial MSE = 848.2200000000      x = [1.3 0.7 0.8 1.9 1.2]
```

```
iterations = 19068
```

```
Final MSE = 0.0000000001      x = [0.99999892 0.99999784 0.99999568 0.99999133
0.99998262]
```

```
gradient = [-5.35689449e-07 -1.07339002e-06 -2.15181410e-06 -4.31421250e-06
-8.64985930e-06]
```

```
gradient norm = 9.975065339353984e-06
```

```
[ ]:
```