Report Lab 4

Nguyen Tien Duc - ITITIU18029

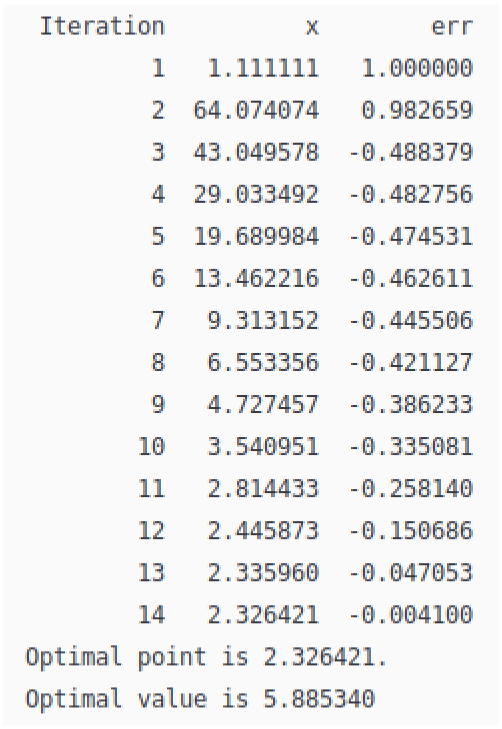
December 22, 2019

Dijkstra and Floy

Code

```python
def f(x): return 4 * x - 1.8 * (x ** 2) + 1.2 * (x ** 3) - 0.3 * (x ** 4)


# derivative of f(x) is g(x)
def g(x): return 4 - 3.6 * x + 3.6 * (x ** 2) - 1.2 * (x ** 3)


def dg(x): return -3.6 + 7.2 * x - 3.6 * (x ** 2)


def newton(f, df, x_initial, e_s, max_iter):
  i = 0
  e_a = 100
  x0 = x_initial
  global x
  print("%10s %10s %10s" % ("Iteration", "x", "err"))
  while i < max_iter and abs(e_a) > e_s:
    i = i + 1
    x = x0 - (f(x0) / df(x0))
    e_a = (x - x0) / x
    x0 = x
    print("%10d %10f %10f" % (i, x, e_a))
  return x


optimal_point = newton(g, dg, 0, 0.01, 100)

print("Optimal point is %f." % optimal_point)

print("Optimal value is %f" % f(optimal_point))
```

Running



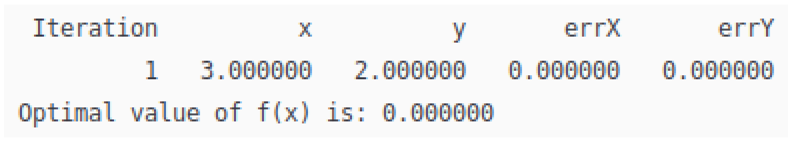
```
 Iteration          x        err
         1   1.111111   1.000000
         2  64.074074   0.982659
         3  43.049578  -0.488379
         4  29.033492  -0.482756
         5  19.689984  -0.474531
         6  13.462216  -0.462611
         7   9.313152  -0.445506
         8   6.553356  -0.421127
         9   4.727457  -0.386233
        10   3.540951  -0.335081
        11   2.814433  -0.258140
        12   2.445873  -0.150686
        13   2.335960  -0.047053
        14   2.326421  -0.004100
Optimal point is 2.326421.
Optimal value is 5.885340
```

2

Code

```python
import sympy as sp

X, Y = sp.symbols('X, Y', real=True)

f = (X - 3) ** 2 + (Y - 2) ** 2


def steepest_descent(function, x_i, y_i, Es, max_iter, true_x, true_y):
  global x_res, y_res
  i = 0
  err_x = 100
  err_y = 100

  x0, y0 = x_i, y_i

  H = sp.Symbol('H', real=True)
  dfdx = sp.diff(f, X)
  dfdy = sp.diff(f, Y)

  print("%10s %10s %10s %10s %10s" % ("Iteration", "x", "y", "errX", "errY"))

  while i < max_iter and (abs(err_x) > Es or abs(err_y) > Es):
    i = i + 1

    x = x0 + dfdx.subs([(X, x0), (Y, y0)]) * H
    y = y0 + dfdy.subs([(X, x0), (Y, y0)]) * H

    g = f.subs([(X, x), (Y, y)])
    dg = sp.diff(g, H)

    if sp.solve(dg, H):
      h = sp.solve(dg, H)[0]

    x_res = x.subs(H, h)
    y_res = y.subs(H, h)

    err_x = (true_x - x_res) / true_x
    err_y = (true_y - y_res) / true_y

    x0, y0 = x_res, y_res
    print("%10d %10f %10f %10f %10f" % (i, x_res, y_res, err_x, err_y))
  return f.subs([(X, x_res), (Y, y_res)])


print("Optimal value of f(x) is: %f" % steepest_descent(f, 1, 1, 0.01, 100, 3, 2))
```

Running



```
 Iteration          x          y       errX       errY
         1   3.000000   2.000000   0.000000   0.000000
Optimal value of f(x) is: 0.000000
```

Analysis and Comparation

Although both of the 2 methods can help us find the optimal points and values of a function, there are some differences:

- Newton method
  This method is originally a root-finding method.
  Here we just use Newton method to find the root of the derivative function, which is also the optimal point.

  ⇒ Therefore, this method is simple in its principle but it is not the best method for optimization.
  This can be proved by looking at the 1st problem: a simple 4th degree single variable non-linear function but it requires up to 14 iterations to find out the optimal value.

- Steepest descent method
  This method, from the result in problem 2, is unarguably faster than Newton method with only 1 iteration needed.
  This is due to it being designed to solve optimization problems at first.
  However, the main disadvantage of this method that I encounter while applying it is the complexity in its principles.
  The first step of finding out the gradient is simple, but the fact that we have to recalculate the value of the gradient in every iteration makes it tedious.
  The second step of find out the optimal point in the gradient direction, on the other hand, is a whole root-finding problem of the function's derivative. Thus, it can be very tedious in cases when it is hard

to find the derivatives or when the derivative function is hard to solve.

$\Rightarrow$ Therefore, this method is faster, better for optimization but is much more complex and tedious.

Conclusion

Newton method is much simpler, thus, it should be prioritized in hand-on calculation with no help from computer.

Reversely, steepest descent method should be prioritized as an algorithm in optimization programs because the computer can help us do all the tedious jobs and we can benefit from the advantages of the method.