Comparison of Steepest Descent and Newton's Type Optimisation Methods

Optimisation
Project Report

Serkan Burak Örs[*]

[*]orss19@itu.edu.tr

Contents

1 Introduction

In this project report, both steepest descent method and Newton type method implemented in order to solve given optimisation problem at Eq.[1.1]. Then the solution is visualised and results are discussed.

$$\text{Minimize } F\left(x_1, x_2\right) = x_1^3 - 2x_1x_2^2 + 3x_1x_2 + 2x_2^3 \tag{1.1}$$

$$\text{Subject to: } x_1 + 0.5x_2 = 7 \tag{1.2}$$

2 Solution

By using equation[1.2], x_2 can be written in terms of x_1 as:

$$x_1 + 0.5x_2 = 7 \tag{2.1}$$

$$x_2 = 14 - 2x_1 \tag{2.2}$$

And this obtained x_1 value can be substituted in equation[1.1] as:

$$F(x_1, x_2) = x_1^3 - 2x_1\left(14 - 2x_1\right)^2 + 3x_1\left(14 - 2x_1\right) + 2\left(14 - 2x_1\right)^3 \tag{2.3}$$

Termination criteria for applying the methods is chosen as

$$\text{Error} = \frac{x_{i+1} - x_i}{x_{i+1}} \times 100 \leq 10^{-5} \tag{2.4}$$

Steepest descent method):
At first, gradient descent diverged from solution for this problem, hence a line search algorithm for optimal step size is implemented and initial point is tuned in order to converge the minimum with using gradient descent method.

$$\alpha_0 = 0.01 \tag{2.5}$$

$$\alpha_k = arg\left[\min_{\alpha}\ \left(x_k - \alpha_0 F'(x_k)\right)\right] \tag{2.6}$$

$$x_{i+1} = x_i - \alpha_k F'(x_i) \tag{2.7}$$

Newton's type method:
In order to converge same optimal point with previous method, initial point is tuned.

$$x_{i+1} = x_i - \frac{F'(x_i)}{F''(x_i)} \tag{2.8}$$

By solving the problem using these methods with arrangements explained, we can obtain results below as:

- Due to steepest descent method
  - $x_1 = 5.036458583$
  - $x_2 = 3.927082834$
  - $F = 152.872610441$
- Due to Newton's method
  - $x_1 = 5.036557818$
  - $x_2 = 3.926884364$
  - $F = 152.872609510$

We can cross-check the results by solving the given constrained nonlinear optimisation problem using Lagrange multipliers

$$\text{Minimize } F(x_1, x_2) = x_1^3 - 2x_1x_2^2 + 3x_1x_2 + 2x_2^3 \tag{2.9}$$

$$g(x_1, x_2) = x_1 + 0.5x_2 - 7 = 0 \tag{2.10}$$

We can form the Lagrangian as

$$L = F + \lambda g = x_1^3 - 2x_1x_2^2 + 3x_1x_2 + 2x_2^3 + \lambda(x_1 + 0.5x_2 - 7) \tag{2.11}$$

Necessary conditions can be determined as:

$$\frac{\partial F}{\partial x_1} = 3x_1^2 - 2x_2^2 + 3x_2 + \lambda = 0 \tag{2.12}$$

$$\frac{\partial F}{\partial x_2} = -4x_1x_2 + 3x_1 + 6x_2^2 + 0.5\lambda = 0 \tag{2.13}$$

$$\frac{\partial F}{\partial \lambda} = x_1 + 0.5x_2 - 7 = 0 \tag{2.14}$$

By solving Eq.[2.12-2.14], we obtain minimum point as

$$F(x_1, x_2) = F((5.036557818, 3.926884364) = 152.872609510 \tag{2.15}$$

Now, we can compare optimisation methods with metrics in the table below

Table 1: Comparison of optimisation methods

Optimisation Methods	**Termination Time (sec)**	**Iteration Number**	**Error Percentage (%)**
Steepest descent	0.000000000e+00	5	2.042620951e-9
Newton's method	3.499507904e-03	5	3.136801832e-10

Discussions:

- Gradient descent method converges to optimal point faster than Newton's method if the initial conditions and step size are tuned appropriately
- Despite of its fast convergence, gradient descent has more error percentage, in order to reduce this error gradient descent and keep its fast convergence, gradient descent methods and Newton's type methods can be combined

And we can visualize the cost function, constraint and minimum point in Fig.[1]:

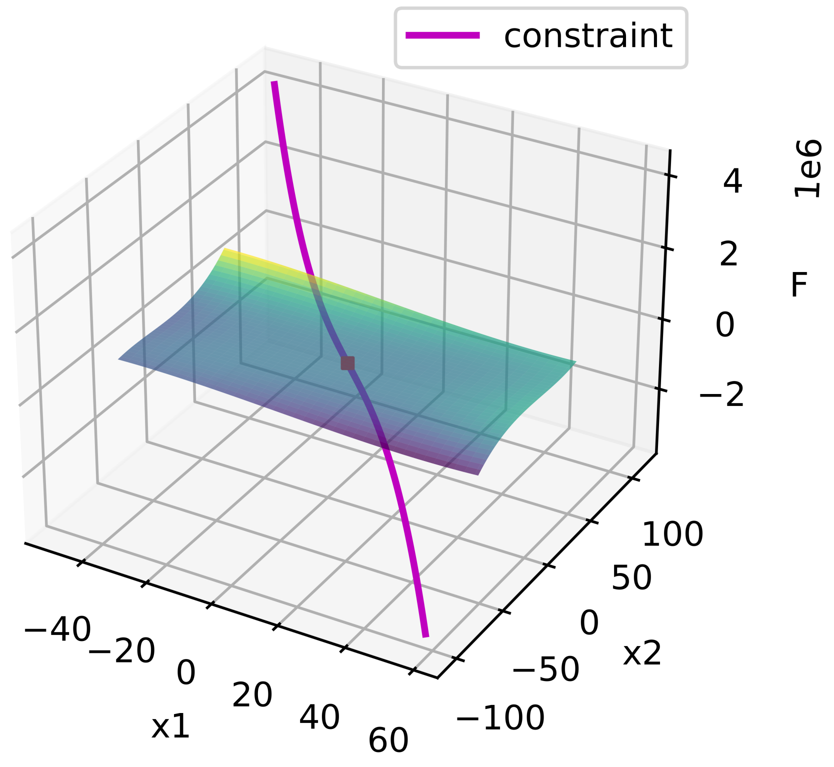

Figure 1: Graphical representation of objective function and the constraint

3 Appendix A: Python Code

```
import numpy as np
import sympy as sym
import matplotlib.pyplot as plt
import time
# Define symbolic variables
x = sym.symbols('x')
y = sym.symbols('y')
# Initialize parameters
x1 = np.ones(10)
x1[0] = 4.5
x2 = np.ones(10)
x2[0] = 4.5
F = x**3 - 2*x*y**2 + 3*x*y + 2*y**3
F = F.subs(y,14-2*x)
Fprime = sym.diff(F)
FdoublePrime = sym.diff(Fprime)
# Gradient descent with line search
start_time = time.time()
for i in range(9):
# determine step size using line search
alpha = 1e-2
while F.subs(x, x1[i] - alpha*Fprime.subs(x, x1[i])) ≥ F.subs(x, x1[i]) ...
    - 0.5*alpha*(Fprime.subs(x, x1[i])**2):
alpha *= 0.5 # minimising is performed by dividing half in order to ...
    converge faster
# update x1
x1_new = x1[i] - alpha*Fprime.subs(x, x1[i])
if abs(((x1_new - x1[i])/x1_new)*100) < 1e-5:
break
x1[i+1] = x1_new
funGD = F.subs(x,x1[i])
end_time_1 = time.time()
# Newton-type
start_time = time.time()
for i in range(9):
x2_new = x2[i] - (Fprime.subs(x, x2[i])/FdoublePrime.subs(x, x2[i]))
if abs(((x2_new - x2[i])/x2_new)*100) < 1e-5:
break
x2[i+1] = x2_new
```

```
funN = F.subs(x,x2[i])
end_time_2 = time.time()
# Print results
print(f'Due to gradient descent method with line search:\n x1 = ...
    {x1[i]:.9f}\n x2 = {14-2*x1[i]:.9f}\n F = {F.subs(x, x1[i]):.9f}\n')
print(f'Gradient descent converged in {i+1} iterations and took ...
    {end_time_1 - start_time:.9e} seconds\n')
print(f'Due to Newton\'s type method:\n x1 = {x2[i]:.9f}\n x2 = ...
    {14-2*x2[i]:.9f}\n F = {F.subs(x, x2[i]):.9f}\n')
print(f'Newton\'s type converged in {i+1} iterations and took ...
    {end_time_2 - start_time:.9e} seconds\n')
# Plot function
xx = np.arange(-50, 60)
X, Y = np.meshgrid(xx, xx)
FF = X**3-2*X*Y**2+3*X*Y+2*Y**3
fig = plt.figure(dpi=1200)
ax = fig.add_subplot(projection='3d')
ax.plot_surface(X, Y, FF, cmap='viridis',alpha=0.707)
ax.plot(xx, 14-2*xx, ...
    xx**3-2*xx*(14-2*xx)**2+3*xx*(14-2*xx)+2*(14-2*xx)**3, color='m', ...
    linewidth=2,label='constraint')
ax.set_xlabel('x1')
ax.set_ylabel('x2')
ax.set_zlabel('F')
ax.legend()
# Mark point on plot
ax.scatter(x2[i], 14-2*x2[i], F.subs(x, x2[i]), marker='s', s=10, ...
    facecolors='r')
plt.show()
# Results obtained from lagrange multipliers method
# in order to analyse and compare error percentage of the algorithms
x = [5.036557818,3.926884364]
fun = 152.872609510
# Print the results
print('\nResults obtained from lagrange multipliers method \nin order ...
    to analyse and compare error percentage of the algorithms:')
print(f'Real optimal solution:, ({x[0]:.9f},{x[1]:.9f})')
print(f'Real minimum value:, {fun:.9f}\n')
# Error calculations
errGD = ((abs(funGD - fun))/fun)*100
errN = ((abs(funN - fun))/fun)*100
# Print results
print(f'Error percent of Gradient Descent method: {errGD:.9e} %')
print(f'Error percent of Newton\'s type method: {errN:.9e} %\n')
```

4 References

[1] Kirk D.E. *Optimal Control Theory*. Dover Publications. 1998.

[2] Chong, Edwin K. P., and Stanislaw H. Zak. *Introduction to Optimization*. John Wiley & Sons. 2013.