Solving two-side numerical optimization Problems using new approach of Gradient Descent Algorithm

Ammar Imad Nadhim
Iraq's University of Babylon's College for Pure Science Education in Hilla, amar.math89@gmail.com

Dr. Ahmed Sabah Al-Jilawi
Department of Mathematics, College of Education for Pure Science, University of Babylon, Hilla, Iraq, ahmed.aljelawy@uobabylon.edu.iq

Abstract

In this study, we present a new gradient descent algorithm method to find first-order iterative optimization with one variable as well as a larger number of variables. The main objective of the optimization is to improve the value of the objective function taking into account a variety of constraints. The new approach algorithm depends on the process of determining the local minimum and maximum values of the function. This approach is an effective method using this strategy in applied mathematics to reach the optimal solution. The new approach algorithm is used for constraint solving and unconstrained numerical optimization.

Keywords: *Gradient descent algorithm, Numerical Optimization, dynamic and control Optimization, Python.*

Introduction

Optimizing processes is one of the most powerful approaches in process integration. "Best" is a term used in optimization to describe the most advantageous option among a set of feasible alternatives mathematical modeling and numerical simulation. A mathematical model is a representation of physical reality that can be analyzed and calculated. We can compute the using numerical simulation, [1,2,3]

calculate a model's solution on a computer in order to make a virtual duplicate of physical reality.

PDEs (partial differential equations) or multivariable differential equations will be our major modeling tool in this inquiry (time and space, for example).

Applied mathematics has a third fundamental feature: the mathematical study of models.

Mathematical analysis is a necessary step It is possible to get some severe shocks from numerical solutions to physical models.

A detailed understanding of the underlying mathematical ideas is required to fully appreciate them. and

Nonlinear problems and applications are the driving force behind applied mathematics.

difficulties that do not have any random or stochastic aspects. Finally, something must be done in order for this to work.

In our efforts to be simple and understandable, we may occasionally use ambiguous language.

in our use of mathematics. We may ensure the more discerning reader that

an example of modeling that leads to the equation for heat flow.

Numerical algorithms must be utilized. This goal is to show and analyze several algorithms that help us to better understand the world around us.

To tackle real-world problems, all of the algorithms covered here may be put to work

computer-aided to specific optimization issues.

All of these algorithms are iterative in nature, beginning with a predetermined initial u_0condition.

Each approach creates a sequence (U_n) n ∈N that converges under certain conditions,[4,5,6,19]

Methodology

In this section, we talk about how to solve both constrained and unconstrained problems using the gradient descent algorithm of numerical optimization and solving dynamic and control optimization

Gradient Descent algorithm

Gradient descent initialize the gradient and learning rate at the beginning of each iteration to provide new points for each iteration.

Vector of objective function that contains partial derivatives with respect to points' dimensions or coordinates.

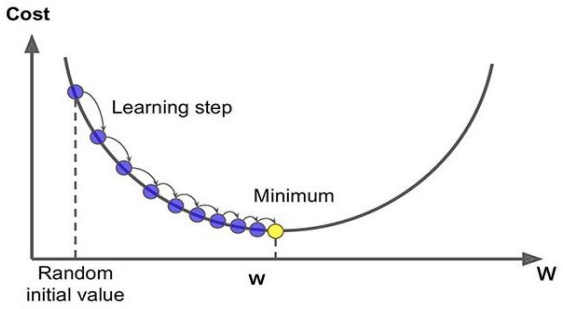
In other words, the learning rate tells us how close we want to the optimal point and how much time it will take us to get there. In minimization problems, the gradient is negative, and in maximization problems, the gradient is positive.

The Gradient Descent algorithm can be visualized as a hill that is gradually descending if we are briefed on its principle,[10,11,12]

goal of descending the mountain in the quickest feasible time by meticulous planning and adherence to detailed instructions. Our starting point, progress rate, and general direction are all determined by it. We can get a head start on achieving our objective if we use these factors as a guide. When the gradient is zero, we know we have reached the lowest since the function's derivatives are all zero at either the local or global minimum/maximum. According to the function's nature and location, it might be either global or local. In the case of convex functions, gradient descent can be used to ensure that the function under study converges to a single minimum. In more complex functions, most algorithms ultimately stabilize (and may even be optimum). The gradient is zero) points, based on the step size. In order to pick the precise step size, there are several criteria and approaches to consider, such as. A simple linear regression model will be used to demonstrate how the gradient descent works to reduce costs by optimizing the cost function using intercept and slope.

$$\boldsymbol{x^{k+1}} = \boldsymbol{x^{k}} - \Delta \boldsymbol{f}(\boldsymbol{x^{k}})\boldsymbol{\tau}$$

Where, ($\boldsymbol{x^{k+1}}$) is the next point we want to go, $\boldsymbol{x^{k}}$ is our current location, the gradient of the function at our current location, and " τ" is the learning rate,[16,17,18]



$$X = X - \alpha \frac{d}{dx} f(x)$$

Where X = input, F(x) = output based on X, α = learning rate

Data Analysis

Python Code in Gradient Descent with constraint

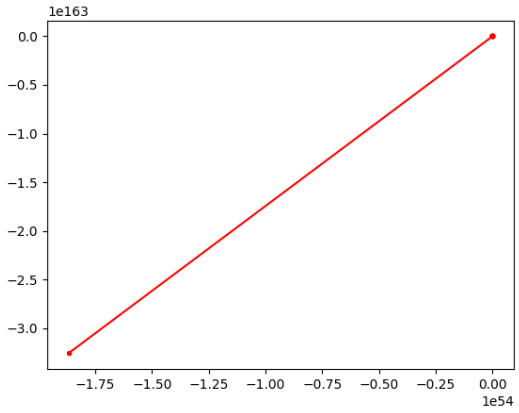
```
from scipy.optimize import minimize
def objective(x):
  x1 = x[0]
  x2 = x[1]
  return 5* (x1 ** 2) + x2 ** 2
def constraint1(x):
  return x[0] + x[1] - 1
# Try an initial condition of x1>2 and x2>=1
# Our initial condition satisfies the constraint already
x0 = [0.3, 0.7]
print(objective(x0))
xnew = [0.25, 0.75]
print(objective(xnew))
# Since we have already calculated on paper we know that x1 and x2 fall between 0 and 1
# We can set our bounds for both variables as being between 0 and 1
b = (0, 1)
bounds = (b, b)
# Lets make note of the type of constraint we have for out optimizer
con1 = {'type': 'eq', 'fun': constraint1}
cons = [con1]
sol_gradient = minimize(objective, x0, method='SLSQP', bounds=bounds, constraints=cons)
print(sol_gradient)
```

objective at x_0	0.94
objective at x_1	0.875
jac	array([1.66666673, 1.66666668])
nfev	7
nit	2
njev	2
status	0
x	array([0.16666667, 0.83333333])

Python Code in Gradient Descent without constraint

```
from numpy import asarray
from numpy import arange
from numpy.random import rand
from matplotlib import pyplot
# objective function
def objective(x):
  return 5 * x ** 3 + x ** 2 + 45
# derivative of objective function
def derivative(x):
  return  15 * x ** 2 + 2 * x
# gradient descent algorithm
def gradient_descent(objective, derivative, bounds, n_iter, step_size):
  # track all solutions
  solutions, scores = list(), list()
  # generate an initial point
  solution = bounds[:, 0] + rand(len(bounds)) * (bounds[:, 1] - bounds[:, 0])
  # run the gradient descent
  for i in range(n_iter):
    # calculate gradient
    gradient = derivative(solution)
    # take a step
    solution = solution - step_size * gradient
    # evaluate candidate point
    solution_eval = objective(solution)
    # store solution
    solutions.append(solution)
  scores.append(solution_eval)
  # report progress
  print('>%d f(%s) = %.5f' % (i, solution, solution_eval))
 return [solutions, scores]
# define range for input
bounds = asarray([[-1.0, 1.0]])
# define the total iterations
n_iter = 10
# define the step size
step_size = 0.1
# perform the gradient descent search
solutions, scores = gradient_descent(objective, derivative, bounds, n_iter, step_size)
# sample input range uniformly at 0.1 increments
inputs = arange(bounds[0, 0], bounds[0, 1] + 0.1, 0.1)
# compute targets
results = objective(inputs)
# create a line plot of input vs result
pyplot.plot(inputs, results)
# plot the solutions found
pyplot.plot(solutions, scores, '.-', color='red')
# show the plot
pyplot.show()
```

Iteration	x	f (x)
1	-0.27005033	44.97446
2	-0.32543103	44.93358
3	0.41920286	44.80740
4	-0.59895884	44.28436
5	-1.01729461	40.77096
6	-2.36616818	-15.63919
7	-10.29106235	-5298.51844
8	-167.09179617	-23297772.99643
9	-42013.1759599	-370786980175207.31250
10	-2.64769404e+09	-928054324384043537560324014 08.00000



this figure above compute the solutions by create a line of input vs result

dynamic and control optimization

Optimization problems for both linear and nonlinear dynamic programming (DP) are a common mathematical approach. The word "dynamic" was coined because the method is typically used to create a series of optimal judgments that may adapt dynamically to changes in conditions over time.[26]

Dynamic programming differs from linear programming, and Do not assume between variables that there is a linear relationship. This means that a wide variety of issues can be addressed with this technology. It's great that there is some difference. However, this comes at a cost because it requires a very special problem - framing the optimization problem as a dynamic program. Therefore, dynamic programming structures are usually considered works of art.

As a matter of control, system modeling is a challenge. The goal is to provide a mathematical description so simple that it accurately predicts the reaction.

A great variety of constraints can be imposed on the problem of optimal control. These constraints limit the range of values that the control and state variables can assume. One usually distinguishes between point constraints and path constraints; Optimum control problems may also have equal limitations. All these restrictions can be of equal or unequal kind. Point restrictions. These constraints are routinely used in optimal control problems, especially final constraints[23,27]

python code in dynamic and control optimization

$$\boldsymbol{max}\ x_2(t_f)$$

$$\boldsymbol{subject\ to} \quad \frac{dx_1}{dt} = -(u + 0.3u^2)x_1$$

$$\frac{dx_2}{dt} = u\,x_1$$

$$x(0) = [1 \quad 0]^T$$

$$0 \le u \le 7$$

$$t_f = \{0, 0.1, 0.2, \dots, 0.9, 1\}$$

```
import numpy as np
import matplotlib.pyplot as plt
from gekko import GEKKO
m = GEKKO()
nt = 101
m.time = np.linspace(0,1,nt)
# Parameters
u = m.MV(value=1,ub=7,lb=0)
u.STATUS = 1
# Variables
x1 = m.Var(value=1)
x2 = m.Var(value=0)
p = np.zeros(nt)
p[-1] = 1.0
final = m.Param(value=p)
# Equations
m.Equation(x1.dt()==-(u+0.3*u**2)*x1)
m.Equation(x2.dt()==u*x1)
# Objective Function
m.Obj(-x2*final)
m.options.IMODE = 9
m.solve()
print('Objective: ' + str(x2[-1]))
plt.figure(1)
plt.plot(m.time,x1.value,'k:',lw=2,label=r'$x_1$')
plt.plot(m.time,x2.value,'b-',lw=2,label=r'$x_2$')
plt.plot(m.time,u.value,'r--',lw=2,label=r'$u$')
plt.legend(loc='best')
plt.xlabel('Time')
plt.ylabel('Value')
show()
```

Numerical results for dynamic and control optimization problem with large scale variables

final value CPU sec in IPOPT	**0.012**
The objective	**- 0.557740968627238**

function final value	
Solution time	**8.040000000255532E-002 sec**
Objective	**-0.558157637615552**
Solver	**IPOPT**

iteration	**Objective**	INF_pr	INF_du	LG(mu)	$\|\|D\|\|$	ALPha_du	ALPha_pr
0	1.1899988e-04	1.30e+00	1.00e+00	0.0	0.00e+00	0.00e+00	0.00e+00
1	-5.5770830e-01	2.22e-16	1.00e-02	-8.0	1.28e+00	9.90e-01	1.00e+00h
2	-5.5770874e-01	2.22e-16	9.50e-05	-10.0	5.42e-05	9.91e-01	1.00e+00f
3	5.5774097e-01	1.11e-16	1.11e-16	-11.0	3.63e-03	1.00e+00	1.00e+00f
4	-5.5815068e-01	2.22e-16	6.30e-02	-8.0	1.06e-01	9.40e-01	1.00e+00h
5	-5.5815075e-01	2.22e-16	1.09e-03	-9.2	9.09e-06	9.90e-01	1.00e+00h
6	-5.5815764e-01	1.11e-16	1.00e-16	-11.0	8.40e-04	1.00e+00	1.00e+00f

iteration: the number of iteration and regular repetition during the recovery phase.

objective: The value of the objective function at the current point during the recovery phase.

INF_pr: Violation of constraint at the current non-scaled point. This quantity is the infinity (maximum) of the (unscaled) constraints (gL $\leqslant$ g(x) $\leqslant$ gU in (NLP))..

inf_du: The scaled dual infeasibility at the current point.

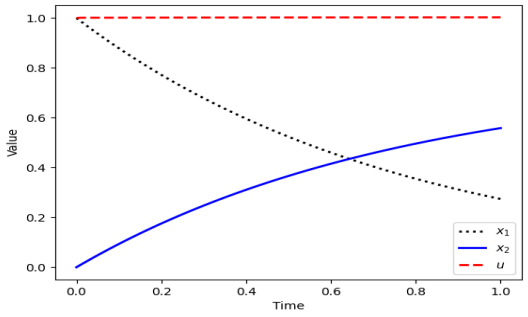
LG (mu): log10 value of the barrier coefficient, μ.

|| D ||: Infinity (maximum) criterion for the initial step (internal slack variables s and original variables x)

lg (rg): log10 for the value of the Hessian settlement term for the Lagrangian in

alpha_du: the size of the scores for the paired variables

alpha_pr: The size of the scores for the initial variables



RESULTS AND DISCUSSION

In this article, we check how the decent gradient algorithm works, it is used in solving problems that contain restricted and unrestricted optimization using Python programming and when solving problems that have an objective function and constraints when choosing a starting value of x_(0)the following iterations are found and other results as well In this article, we used Dynamics and Optimization Control and how to solve problems

As for problems a by the gradient decent algorithm that contains a objective function and without constraints when choosing the value of $x_1 = (-0.27005033)$, and $f(x_1) = (44.97446)$ then the next iterations were calculated until we reached the tenth iteration

and its value $f(x_{10}) = (-928054324384043537560324014 08.00000)$

CONCLUSION

In this paper, we used a new approach in one of the most important numerical optimization algorithms in applied mathematics, which is called gradient (maximum gradient) and it is an iterative first-order numerical optimization algorithm to find the local minimum or local maxima.

The idea of this algorithm is to take iterative steps in the opposite direction of the gradient (approximate gradient) of the function at the current point because that is the direction of the maximum gradient. In this study, we used a new approach to solve constraint and unconstrained numerical optimization problems in a way that the results of the new approach improved through the algorithm that the results are more accurate and in less time to find the optimal solution. Python language was used in all tests.

Reference

[1] Chen, X., Lin, Q., Kim, S., Carbonell, J. G., & Xing, E. P. (2012). Smoothing proximal gradient method for general structured sparse regression. The Annals of Applied Statistics, 6(2), 719-752.

[2] Courant, R., & Hilbert, D. (1989). Methods of mathematical physics, republished by John Wiley and Sons. New York.

[3] Lions, J. L. (1969). Quelques méthodes de résolution de problemes aux limites non linéaires.

[4] Ahuja, R. K., Magnanti, T. L., & Orlin, J. B. (1988). Network flows.

[5] Godlewski, E., & Raviart, P. A. (2013). Numerical approximation of hyperbolic systems of conservation laws (Vol. 118). Springer Science & Business Media.

[6] Dautray, R., & Lions, J. L. (2012). Mathematical analysis and numerical methods for science and technology: volume 1 physical origins and classical methods. Springer Science & Business Media.

[7] Nocedal, J., & Wright, S. J. (Eds.). (1999). Numerical optimization. New York, NY: Springer New York.

[8] Andersen, E. D., & Andersen, K. D. (2000). The MOSEK interior point optimizer for linear programming: an implementation of the homogeneous algorithm. In High performance optimization (pp. 197-232). Springer, Boston, MA.

[9] Frenk, H., Roos, K., Terlaky, T., & Zhang, S. (2000). Self-dual embedding technique. In High Performance Optimization (pp. 93-127). Springer, Boston, MA.

[10] Ruder, S. (2016). An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747.

[11] Andersen, E. D., Gondzio, J., Mészáros, C., & Xu, X. (1996). Implementation of interior point methods for large scale linear programming. HEC/Université de Geneve. [12] for large scale linear programming, in Interior Point Methods in Mathematical Programming,

[13] Chis, V. (2007). Factoring Symmetric Indefinite Matrices.

[14] Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., ... & Sorensen, D. (1999). Linear Algebra PACKage.

[15] Schwartz, J., & Dockery, D. W. (1992). Increased mortality in Philadelphia associated with daily air pollution

concentrations. Am Rev Respir Dis, 145(3), 600-604.

[16] 638 R EFERENCES Baldi, P. (1995). Gradient descent learning algorithm overview: A general dynamical systems perspective. IEEE Transactions on neural networks, 6(1), 182-195.

[17] Anitescu, M. (2000). On solving mathematical programs with complementarity constraints as nonlinear programs. Preprint ANL/MCS-P864-1200, Argonne National Laboratory, Argonne, IL, 3.

[18] Anitescu, M. (2005). On using the elastic mode in nonlinear programming approaches to mathematical programs with complementarity constraints. SIAM Journal on Optimization, 15(4), 1203-1236.

[19] Sand, G., Barkmann, S., Tylko, M., Engell, S., & Schembecker, G. (2004). Robust and efficient MINLP optimization of reactive distillation columns. In FOCAPD, Conference on Foundations of Computer-Aided Process Design (pp. 319-322).

[20] Stoer, J., & Baptist, P. (1977). On the Relation between Quadratic Termination and Convergence Properties of Minimization Algorithms. Part II. Applications. Numerische Mathematik, 28, 367-392.

[21] Kadhim, M. K., Wahbi, F. A., & Hasan Alridha, A. (2022). Mathematical optimization modeling for estimating the incidence of clinical diseases. International Journal of Nonlinear Analysis and Applications, 13(1), 185-195.

[22] Alridha, A., Salman, A. M., & Al-Jilawi, A. S. (2021, March). The Applications of NP-hardness optimizations problem. In Journal of Physics: Conference Series (Vol. 1818, No. 1, p. 012179). IOP Publishing..

[23] Salman, A. M., Alridha, A., & Hussain, A. H. (2021, March). Some Topics on Convex Optimization. In Journal of Physics: Conference Series (Vol. 1818, No. 1, p. 012171). IOP Publishing.

[24] Alridha, A., & Al-Jilawi, A. S. (2021, March). Mathematical Programming Computational for Solving NP-Hardness Problem. In Journal of Physics: Conference Series (Vol. 1818, No. 1, p. 012137). IOP Publishing.

[25] Alridha, A. H., & Al-Jilawi, A. S. (2022). Solving NP-hard problem using a new relaxation of approximate methods. International Journal of Health Sciences, 6, 523-536.

[26] Cutler, C. R., & Ramaker, B. L. (1980). Dynamic matrix control?? A computer control algorithm. In joint automatic control conference (No. 17, p. 72).

[27] Alridha, A., & Al-Jilawi, A. S. (2022, October). K-cluster combinatorial optimization problems is NP_Hardness problem in graph clustering. In AIP Conference Proceedings (Vol. 2398, No. 1, p. 060034). AIP Publishing LLC.