Scientific Research Publishing

American Journal of Operations Research, 2026, 16(4), 157-170
https://www.scirp.org/journal/ajor
ISSN Online: 2160-8849
ISSN Print: 2160-8830

A Novel Method for Solving Equality and Inequality Constrained Nonlinear Optimization Problems with Artificial Neural Networks

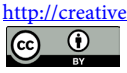
Chinelo R. Anokwute, Ben I. Oruh, Olaniyi S. Maliki

Department of Mathematics, Michael Okpara University of Agriculture, Umudike, Nigeria
Email: chineloanokwute897@gmail.com, oruhben@gmail.com, somaliki@gmail.com

How to cite this paper: Anokwute, C.R., Oruh, B.I. and Maliki, O.S. (2026) A Novel Method for Solving Equality and Inequality Constrained Nonlinear Optimization Problems with Artificial Neural Networks. *American Journal of Operations Research*, **16**, 157-170.
https://doi.org/10.4236/ajor.2026.164008

Received: April 20, 2026
Accepted: July 19, 2026
Published: July 22, 2026

Copyright © 2026 by author(s) and Scientific Research Publishing Inc.
This work is licensed under the Creative Commons Attribution International License (CC BY 4.0).
http://creativecommons.org/licenses/by/4.0/



Open Access

Abstract

In this research, we study how artificial neural network can be used in solving both equality and inequality constrained non-linear optimization problems. The procedure involves generating a system of differential equations which must correspond to the given number of intrinsic variables. We proceed to apply this technique to solve some constrained non-linear optimization problems. Using MathCAD14, we were able to generate the corresponding system of differential equations developed using the neural network gradient scheme. We provide the resulting graphical profiles of the solutions and compared it with the analytic solution. Our steps and results indicate that, for the problems tested, the neural network approach can be more straightforward to implement and avoids some of the iterative assumptions of classical methods.

Keywords

ANN, ODE, Network Training, Convergence, MathCAD14

1. Introduction

The process of minimizing or maximizing a problem is called optimization or mathematical programming as it is termed in standard texts. Optimization is all about finding the best possible solution in a given set of circumstances. Most real life problems always appear as non-linear multivariate problems when expressed mathematically [1]. Sometimes, these problems may have some conditions which its decision variables must satisfy and in this case, we say the problem is constrained, otherwise the problem is unconstrained. Optimization involves choices informed by the study of the situations and parameters affecting those situations

in order to either minimize cost/efforts or maximize benefits. According to Chong and Zak [2], optimization is central to any problem involving decision making, whether in engineering or in economics. The task of decision making entails choosing between various alternatives. This choice is governed by our desire to make the *best* decision. The measure of goodness of the alternatives is informed by an objective function or performance index. In this research article, we are concerned with solving some non-linear constrained optimization problems using artificial neural network approach.

While penalty-based neural network methods for constrained optimization have been established in the literature [3] [4], the practical implementation of these methods often remains abstract, with limited step-by-step numerical demonstrations, particularly for inequality-constrained nonlinear problems. The novelty of the present work is threefold:

1) We provide a fully transparent, step-by-step derivation from the penalty function to the explicit ODE system for two concrete test problems;

2) We demonstrate, using MathCAD14, how the choice of penalty parameters k_i and learning rates μ_j critically affects convergence to the true constrained optimum—a pragmatic issue often underreported; and

3) We present comparative graphical evidence that, for the problems considered, the neural ODE approach converges to the analytic solution from multiple initial conditions. Thus, unlike prior works that focus on theoretical architecture or general algorithms, this paper offers a reproducible numerical recipe and a candid discussion of parameter tuning.

2. Neural Network Structure

A neural network is an inter-connection of processing elements, units or nodes, whose functionality is similar to that of the human neurons [5] [6]. The processing ability of the network is stored in the connection strengths, simply called weights, which can be obtained by a process of adaptation to, a set of training patterns. Neural network methods can solve both linear and non-linear optimization problems. This is possible due to the function approximation property [7] of feed forward neural networks which converts the optimization problem into an ordinary differential equation. This in turn can be solved by conventional methods.

3. Mathematical Model of Artificial Neural Network (ANN)

ANNs are constructed with layers of units [8], which are termed *multilayer* ANNs. A layer of units in an ANN is made up of units that perform similar tasks. First layer of a multilayer ANN consists of input units. These units perform similar tasks as independent variables in statistical literature. Last layer contains output units known as dependent or response variables in statistical nomenclature. Between input and output units in the model are other units called hidden units and constitute hidden layers. There are two functions that govern the behaviour of a unit in a particular layer, which normally are the same for all units within the

whole ANN, *i.e.*

- the input function and
- the output/activation function

Input into a node is a weighted sum of output from nodes connected to it.

A neuron receives a set of n inputs $S=\{x_j \mid j=1,2,\cdots,n\}$. In Figure 1, each input is weighted before reaching the main body of a neuron N_i by connection strength or weight factor w_{ij} for $j=1,2,\cdots,n$. In addition, it has a bias b or w_0, a threshold value θ_k, which has to be reached or exceeded for the neuron to produce an output signal. A function $\varphi(s)$ acts on the produced weighted signal. This function is called an *activation function*. Mathematically, the output of the i^{th} neuron N_i is given by;

$$O_i=\varphi\left[w_0+\sum_{j=1}^{n} w_{ij}x_j\right] \tag{1}$$

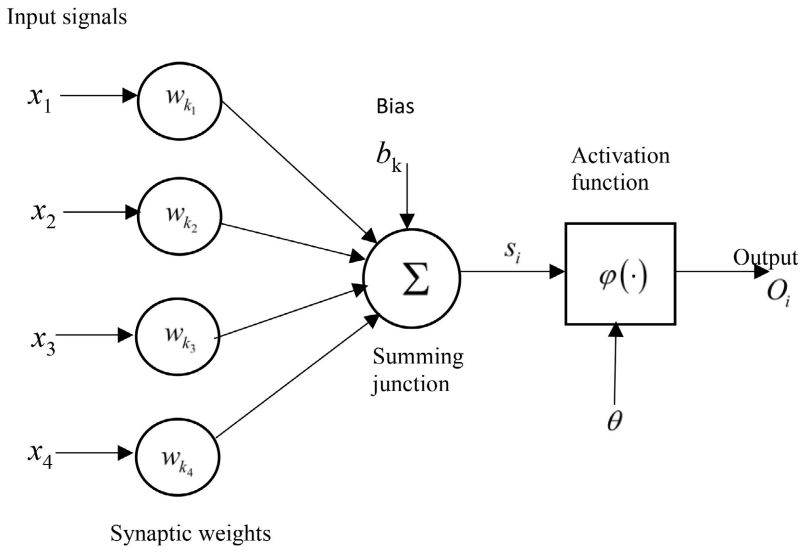


Figure 1. Mathematical model of artificial neural network.

And the neuron's firing condition is,

$$w_0+\sum_{j=1}^{n} w_{ij}x_j \geq \theta \tag{2}$$

Figure 1 shows detailed computational steps of the working principle of an artificial neuron (AN) in a neural network. Now the input signal for the i^{th} neuron N_i is,

$$s_i=w_0+\sum_{j=1}^{n} w_{ij}x_j \tag{3}$$

4. Neural Networks as Universal Approximators

Artificial neural network can make a nonlinear mapping from the inputs to the outputs of the corresponding system of neurons, which is suitable for analyzing

the problem defined by initial/boundary value problems that have no analytical solutions or which cannot be easily computed. One of the applications of the multilayer feed forward neural network is the global approximation of real valued multivariable function in a closed analytic form. Namely such neural networks are universal approximators. It is discovered in the literature that multilayer feed forward neural networks with one hidden layer using arbitrary squashing functions are capable of approximating any Borel measurable function from one finite dimensional space to another with any desired degree of accuracy. This is made clear in the following theorem.

5. Universal Approximation Theorem

The universal approximation theorem for MLP was proved by Cybenko [7] and Hornik *et al.* [8] in 1989. Let $\boldsymbol{I}_n$ represent an n-dimensional unit cube containing all possible input samples $\boldsymbol{x}=\left(x_1,x_2,\cdots,x_n\right)$ with $x_i\in\left[0,1\right]$, $i=1,2,\cdots,n$. Let $C\left(\boldsymbol{I}_n\right)$ be the space of continuous functions on $\boldsymbol{I}_n$, given a continuous sigmoid function $\varphi(\cdot)$, then the universal approximation theorem states that the finite sums of the form

$$y_k=y_k\left(\boldsymbol{x},\boldsymbol{w}\right)=\sum_{i=1}^{N_2}w_{ki}^3\varphi\left(\sum_{j=0}^{n}w_{ki}^2x_j\right)\quad k=1,2,\cdots,m \tag{4}$$

are dense in $C\left(\boldsymbol{I}_n\right)$. This simply means that given any function $f\in C\left(\boldsymbol{I}_n\right)$ and $\varepsilon>0$, there is a sum $y\left(\boldsymbol{x},\boldsymbol{w}\right)$ of the above form that satisfies

$$\left|y\left(\boldsymbol{x},\boldsymbol{w}\right)-f\left(\boldsymbol{x}\right)\right|<\varepsilon\quad\forall\boldsymbol{x}\in\boldsymbol{I}_n$$

6. The Importance of ANN in Optimization

The neural networks follow a different paradigm for computing. The conventional methods to solution are good for fast arithmetic and does what programmer programs. But the conventional methods to solution are not so good for interacting with noisy data or data from the environment massive parallelism, fault tolerance and adopting to circumstances.

The neural network systems help where we cannot formulate an algorithmic solution.

Neutral networks are a form of multiprocessor computer system with.

- Learning ability
- Fault tolerance
- Simple processing elements
- A high degree of interconnection
- A simple scalar messages and
- Adaptive interaction between elements
- Low energy consumption.

Most importantly neutral network converts the non-linear optimization problem together with the constraints to a set of ordinary differential equation which can easily be solved either analytically or numerically.

7. Statement of the Problem

Many constraint optimization problems have been solved using Khun-Trucker and Lagrange multiplier methods [9]. Most of these methods still give concern because of their vigorous steps and assumptions. To solve constrained non-linear optimization problems using artificial neural networks, we employ penalty functions that transform the constrained problem into a single unconstrained problem. The constraints are placed into the objective function via a penalty parameter in such a way that it penalizes any violation of the constraints. The penalty function constructed is actually an energy function for the neural network. Assuming differentiability, a local minimum of the penalty function is found by using a dynamic gradient scheme which provides a system of differential equations for the input neurons corresponding to the variables of the given optimization problem.

8. Optimization with ANN

We now consider a brief overview of artificial neural network structures employed in the solution of constrained optimization problems with inequality constraints. The general form of the problem to be solved is

$$\text{Min } f(\boldsymbol{x}) \tag{5}$$

$$\text{subject to } g_i(\boldsymbol{x}) \geq 0,\ i = 1,2,\cdots,m$$

where $\boldsymbol{x} = (x_1, x_2, \cdots, x_n)^{\mathrm{T}}$. In Equation (2) $f(\boldsymbol{x})$ is the objective function which has to be minimized and $g_i(\boldsymbol{x}), i = 1,2,\cdots,m$ are inequality constraints. It should be noted that any equality constraint $h_i(\boldsymbol{x})$, could be transformed into two inequality constraints $h_i(\boldsymbol{x}) \geq 0$, $-h_i(\boldsymbol{x}) \geq 0$, *i.e.* the problem (5) is sufficiently general. All functions involved are assumed to be continuously differentiable. We have the following important theorem.

Theorem 1. Let (x_k) be a sequence generated by the penalty method with $\{s_k\}$, $k = 1,2,\cdots$, being a nonnegative and strictly increasing sequence tending to infinity. Then, any limit point of the sequence (x_k) is a solution to the problem. The proof can be found in Luenberger [10].

The penalty function method consists of transforming the constrained optimization problem into an unconstrained one based on a penalty function defined as follows:

$$E(\boldsymbol{x}, k) = f(\boldsymbol{x}) + \sum_{i=1}^{m} k_i P(g_i(\boldsymbol{x})) \tag{6}$$

where the penalty term $P(g_i(\boldsymbol{x}))$ should satisfy

$$\begin{cases} P(g_i(\boldsymbol{x})) > 0, \text{ if } g_i(\boldsymbol{x}) \geq 0 \\ P(g_i(\boldsymbol{x})) = 0, \text{ if } g_i(\boldsymbol{x}) < 0 \end{cases} \tag{7}$$

The unconstrained problems are solved by adding a term, called a penalty function to the objective function that consists of a penalty parameter multiplied by a measure of violation of the constraint. In the neural network literature the penalty

function $E(\boldsymbol{x},k)$ is often considered as an energy function for the corresponding neural network [11]. $k_i = k$, $i = 1,2,\cdots,n$. Here, two types of penalty terms are used

$$P(g_i(\boldsymbol{x})) = \left[\min\{0, g_i(\boldsymbol{x})\}\right]^2 \tag{8}$$

$$P(g_i(\boldsymbol{x})) = -\min\{0, g_i(\boldsymbol{x})\} \tag{9}$$

Based on the penalty terms (8), the energy function obtained is

$$E(\boldsymbol{x},k) = f(\boldsymbol{x}) + \frac{1}{2}\sum_{i=1}^{m} k_i \left[\min\{0, g_i(\boldsymbol{x})\}\right]^2 \tag{10}$$

Assuming that $g_i(\boldsymbol{x}), i = 1,2,\cdots,m$ have continuous first derivatives, the same is true for $P(g_i(\boldsymbol{x}))$ as defined by (8), therefore $E(\boldsymbol{x},k)$ in (10) is continuously differentiable. The gradient of $P(g_i(\boldsymbol{x}))$ in (8) is:

$$\nabla\left[\min\{0, g_i(\boldsymbol{x})\}\right]^2 = 2\min\{0, g_i(\boldsymbol{x})\}\nabla g_i(\boldsymbol{x}) \tag{11}$$

By using the gradient strategy for minimizing $E(\boldsymbol{x},k)$, the following system of ordinary differential equations is obtained

$$\frac{\mathrm{d}x_j}{\mathrm{d}t} = -\mu_j\left(\frac{\partial f(\boldsymbol{x})}{\partial x_j} + \sum_{i=1}^{m} k_i S_i g_i(\boldsymbol{x})\frac{\partial g_i(\boldsymbol{x})}{\partial x_j}\right), \quad j = 1,2,\cdots,n.$$

In extended form, the above can be written

$$\begin{aligned}
&\frac{dx_1}{dt} = -\mu_1\left(\frac{\partial f(\boldsymbol{x})}{\partial x_1} + \sum_{i=1}^{m} k_i S_i g_i(\boldsymbol{x})\frac{\partial g_i(\boldsymbol{x})}{\partial x_1}\right) && x_1(0) = x_1^{(0)}\\
&\frac{\mathrm{d}x_2}{\mathrm{d}t} = -\mu_2\left(\frac{\partial f(\boldsymbol{x})}{\partial x_2} + \sum_{i=1}^{m} k_i S_i g_i(\boldsymbol{x})\frac{\partial g_i(\boldsymbol{x})}{\partial x_2}\right) && x_2(0) = x_2^{(0)}\\
&\quad\vdots && \quad\vdots\\
&\frac{\mathrm{d}x_n}{\mathrm{d}t} = -\mu_n\left(\frac{\partial f(\boldsymbol{x})}{\partial x_n} + \sum_{i=1}^{m} k_i S_i g_i(\boldsymbol{x})\frac{\partial g_i(\boldsymbol{x})}{\partial x_n}\right) && x_n(0) = x_n^{(0)}
\end{aligned} \tag{12}$$

where $\mu_j > 0 \quad j = 1,2,\cdots,n$, represent the learning rate parameters, $k_i > 0$, $i = 1,2,\cdots,m$, and t is the neural network time, with

$$S_i = \begin{cases} 1 & \text{if} \quad g_i(\boldsymbol{x}) \le 0 \\ 0 & \text{if} \quad g_i(\boldsymbol{x}) > 0 \end{cases} \tag{13}$$

and, obviously,

$$S_i g_i(\boldsymbol{x}) = \min\{0, g_i(\boldsymbol{x})\} \tag{14}$$

The functional scheme for simulating the above equations could be considered as a Neural Network, where the integrators represent the neurons (basic units) and functional nonlinear generators build up the connections between them. This approach, described by Cichocki *et al.* [12], essentially replaces the constrained problem (5) by an unconstrained minimization of the differentiable penalty function (6). However, theoretical results on the penalty function method, show that equivalence of the problems (5) and $\min_x\{E(\boldsymbol{x},k)\}$ is only obtained in the limit,

as $\min_{i=1,\cdots,m}\{k_i\} \to \infty$.

The quality of the approximation improves provided larger values are chosen for the k_i's, however, the true solution of (5) will not be obtained unless $\min_{i=1,\cdots,m}\{k_i\} \to \infty$.

Let us now consider the penalty terms (7). Using these in (10) gives the following penalty function:

$$E(\boldsymbol{x},k) = f(\boldsymbol{x}) - \sum_{i=1}^{m} k_i \min\{0, g_i(\boldsymbol{x})\} \tag{15}$$

Due to the non-differentiability of the penalty term (7), this penalty function is non-differentiable, even though the functions $g_i(\boldsymbol{x})$ are assumed differentiable. When the penalty terms (7) are used, the corresponding system of ordinary differential equations is:

$$\frac{\mathrm{d}x_j}{\mathrm{d}t} = -\mu_j \left(\frac{\partial f(\boldsymbol{x})}{\partial x_j} - \sum_{i=1}^{m} k_i S_i \frac{\partial g_i(\boldsymbol{x})}{\partial x_j} \right), \quad j = 1,2,\cdots,n \tag{16}$$

where $\mu_j > 0$ and

$$S_i = \begin{cases} 1 & \text{if} \quad g_i(x) \le 0 \\ 0 & \text{if} \quad g_i(\boldsymbol{x}) > 0 \end{cases}$$

This is a simpler neural network than the previous one. However because of the non-smooth nature of the penalty function, the first derivative discontinuities could slow up the solution speed [13].

9. Step-by-Step Algorithm for Solving Constrained NLP via Neural ODEs

Given: Min $f(\boldsymbol{x})$ s.t $g_i(\boldsymbol{x}) \ge 0, i = 1,2,\cdots,m$

1) Form the penalty function as given by Equation (10):

$$E(\boldsymbol{x},k) = f(\boldsymbol{x}) + \frac{1}{2}\sum_{i=1}^{m} k_i \left[\min\{0, g_i(\boldsymbol{x})\}\right]^2$$

2) Derive the ODE system via gradient descent:

$$\frac{\mathrm{d}x_j}{\mathrm{d}t} = -\mu_j \left(\frac{\partial f(\boldsymbol{x})}{\partial x_j} - \sum_{i=1}^{m} k_i S_i \frac{\partial g_i(\boldsymbol{x})}{\partial x_j} \right), \quad j = 1,2,\cdots,n$$

where $S_i = 1$ if $g_i(x) \le 0$ (active/violated), else $S_i = 0$.

3) Choose penalty parameters $k_i > 0$ (start moderate, e.g., 2 - 20) and learning rates $\mu_j > 0$ (e.g., 10 - 100).

4) Set initial conditions $\mathbf{x}(0)$—preferably inside the feasible region or slightly outside.

5) Solve the ODE system numerically (e.g., Runge-Kutta adaptive) over an interval $t \in [0,T]$ until $\mathrm{d}\boldsymbol{x}/\mathrm{d}t \approx 0$.

6) Check feasibility: verify $g_i(\boldsymbol{x}^*) \ge 0$ within tolerance. Check optimality: compare with known analytic solution or verify KKT conditions.

7) If constraints are violated, increase k_i and/or adjust μ_j and resolve.

Remark:

We now address the issue of the choice of parameters and tuning rule.

In all simulations, learning rates μ_j were set equal and chosen to ensure stable integration without oscillation (typical range 10 - 100). Penalty parameters k_i were initially set to 2. If the steady-state solution violated any constraint, k_i was increased multiplicatively (e.g., from 2 to 20) until all constraints were satisfied within a tolerance of 10^{-3}. Initial conditions were selected to test convergence from different basins; when convergence led to infeasibility, we re-initialized closer to the expected feasible region (as in Problem 2, moving from (1, 0, 1) to (15, 10, 5)). The integration interval $[0,T]$ was chosen as $T=2$ because all trajectories reached steady state by $t \approx 1$. No automatic stopping criterion was used; steady state was identified visually from the graphical profiles.

10. Limitations of the Neural ODE Penalty Approach

The penalty method transforms the constrained problem into an unconstrained one, but equivalence is only attained in the limit as $\min_i \{k_i\} \to \infty$. In practice, finite k_i yield approximate solutions. Convergence depends sensitively on the choice of penalty parameters, learning rates, and initial conditions, as demonstrated in Problem 2. Poor choices can lead to infeasible steady states or slow convergence. Additionally, the claim that the neural network method is *always* easier and faster than classical approaches (e.g., Lagrange multiplier or sequential quadratic programming) is not supported by direct computational time comparisons here. For small-scale problems, classical methods may be equally efficient and offer exact constraint satisfaction without parameter tuning. Therefore, the advantages of the neural ODE method are most evident when an analytic gradient is available, the ODE system can be solved with standard integrators, and the user is willing to tune penalty parameters for feasibility.

11. Numerical Results

In this section, we present the results for the neural network system considered. We present the simulation of the ODE models of the Neural Network obtained from solving our various problems. The graphical profile of the solution is also shown. We will then go on to discuss our results and then compare with the analytic solution of the constraint optimization problem. We first consider the following simple example in other to test the network.

Problem 1

$$\underset{x_1,x_2}{\text{Min}}\ f(\boldsymbol{x}) = 2x_1^2 + x_2^2$$

Subject to

$$\begin{cases} g_1(\boldsymbol{x}) = -x_1 + x_2 - 2 \geq 0 \\ g_2(\boldsymbol{x}) = \ x_1 + x_2 - 2 \geq 0 \end{cases} \tag{17}$$

The solution space for problem of Equation (14) is defined by the functions $g_1(\boldsymbol{x})$ and $g_2(\boldsymbol{x})$ depicted below in Figure 2.

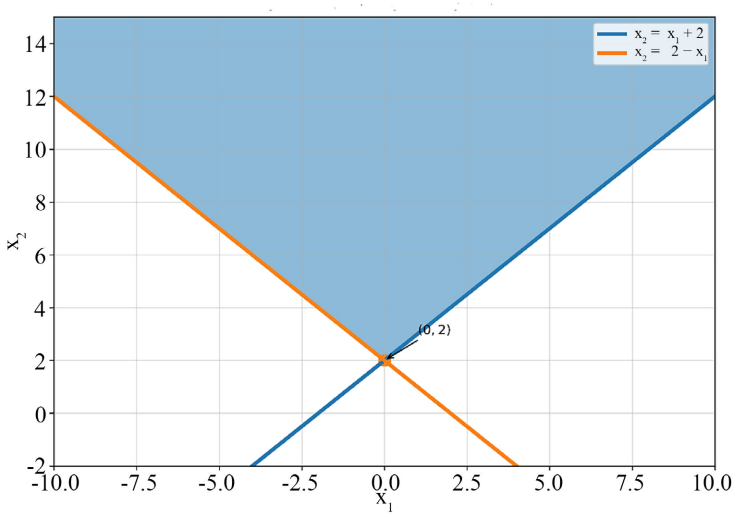


Figure 2. Feasible region of problem (17) with solution $\boldsymbol{x}^* = (x_1, x_2) = (0,2)$.

Following the algorithm of the penalty function term (16) associated with the Equation (2), we first develop the differential equation for the neurons x_1, x_2 whose solution we expect will converge to the solution $\boldsymbol{x}^* = (x_1, x_2) = (0,2)$. We then have;

$$\frac{\mathrm{d}x_j}{\mathrm{d}t} = -\mu_j \left(\frac{\partial f(x)}{\partial x_j} - \sum_{i=1}^{m} k_i S_i \frac{\partial g_i(x)}{\partial x_j} \right), \quad j = 1,2,\cdots,n$$

$$\Rightarrow \quad \frac{\mathrm{d}x_1}{\mathrm{d}t} = -\mu_1 \left(\frac{\partial f(x)}{\partial x_1} - \sum_{i=1}^{2} k_i S_i \frac{\partial g_i(x)}{\partial x_1} \right)$$

$$= -\mu_1 \left(\frac{\partial f(x)}{\partial x_1} - k_1 S_1 \frac{\partial g_1(x)}{\partial x_1} - k_2 S_2 \frac{\partial g_2(x)}{\partial x_1} \right)$$

$$\therefore \quad \frac{\mathrm{d}x_1}{\mathrm{d}t} = -\mu_1 (4x_1 - k_1 S_1 + k_2 S) \quad \Rightarrow \quad \frac{\mathrm{d}x_1}{\mathrm{d}t} = -\mu_1 (4x_1 - k_1 + k_2) \tag{18}$$

Here $S_1 = S_2 = 1$, since our constraints is not strictly greater than zero. Similarly,

$$\frac{\mathrm{d}x_2}{\mathrm{d}t} = -\mu_2 (2x_1 - k_1 - k_2) \tag{19}$$

Thus the neural network for the given optimization problem is

$$\dot{x}_1 = -\mu_1 (4x_1 - k_1 + k_2), \quad \dot{x}_2 = -\mu_2 (2x_1 - k_1 - k_2) \tag{20}$$

We employed MathCAD14 [14] for stimulation of the ODE model of the neural network. We started by choosing an initial condition;
$\boldsymbol{x}(0) = (x_1(0), x_2(0)) = (1,-1)$ and $\boldsymbol{x}(0) = (x_1(0), x_2(0)) = (-1,1)$. Furthermore we employ the following parameters $k_1 = 2$, $k_2 = 2$, $\mu_1 = 10$, $\mu_2 = 10$. Solving the ODEs, we obtain the following graphical profiles for their solutions.

In **Figure 3(a)**, we can see that $x_1(t)$ starting from its initial point of 1, converges to 0, while $x_2(t)$ started from its initial point of −1 converges to 2 which is the exact analytic solution to the optimization problem.

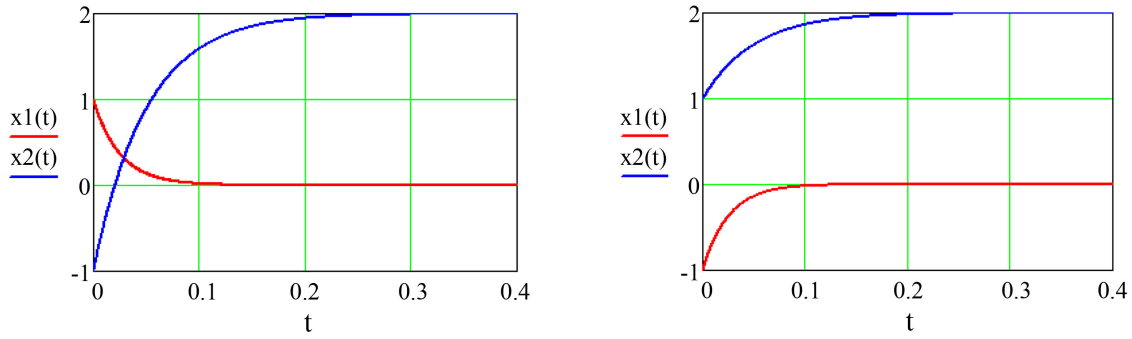


Figure 3. (a) $x_1(t)=e^{-40t}$, $x_2(t)=2-3e^{-20t}$; (b) $x_1(t)=-e^{-40t}$, $x_2(t)=2-e^{-20t}$.

Similarly in **Figure 3(b)** with different initial conditions the system converges again to the exact solution.

Next, we consider higher dimensional non-linear constrained optimization problems.

Problem 2

$$\text{Min } f(\boldsymbol{x})=x_1^2+x_2^2+x_3^2+40x_1+20x_2 \tag{21}$$

Subject to

$$\begin{aligned} g_1(\boldsymbol{x}) &= x_1-50\geq 0\\ g_2(\boldsymbol{x}) &= x_1+x_2-100\geq 0\\ g_3(\boldsymbol{x}) &= x_1+x_2+x_3-150\geq 0 \end{aligned}$$

Here $m=3$, $j=1,2,3$.

The neural network system of equations for the given optimization problem is

$$\begin{cases} \dot{x}_1=-\mu_1\left(2x_1+40-k_1S_1g_1(\boldsymbol{x})-k_2S_2g_2(\boldsymbol{x})-k_3S_3g_3(\boldsymbol{x})\right)\\ \dot{x}_2=-\mu_2\left(x_2+20-k_1S_1g_1(\boldsymbol{x})-k_2S_2g_2(\boldsymbol{x})-k_3S_3g_3(\boldsymbol{x})\right)\\ \dot{x}_3=-\mu_3\left(x_1-k_1S_1g_1(\boldsymbol{x})-k_2S_2g_2(\boldsymbol{x})-k_3S_3g_3(\boldsymbol{x})\right) \end{cases} \tag{22}$$

To solve the ODEs we first choose the following values $k_1=2$, $k_2=2$, $k_3=2$, $\mu_1=10$, $\mu_2=10$, $\mu_3=10$. Also we maintain $S_1=S_2=S_3=1$.

$$\begin{cases} \dot{x}_1=-80x_1-40x_2-20x_3+5600\\ \dot{x}_2=-60x_1-60x_2-20x_3+5800\\ \dot{x}_3=-60x_1-40x_2-40x_3+6000 \end{cases} \tag{23}$$

The above system is solved using MathCAD 14. The graphical profiles are given below.

Remark

1) In **Figure 4(a)**, from the initial point of 1, the neural network rises sharply above 30 and then converged from $t=0$ to a steady state solution of $y_0=x_1=37.526$.

2) In **Figure 4(b)**, from the initial point of 0, the neural network rises sharply above 40 and ten converged from $t=0$ to a steady state solution of $y_1=x_2=$

44.286.

3) In Figure 4(c) from the initial point of 1, the neural network rises sharply above 50 and ten converged from $t = 0$ to a steady state solution of $y_2 = x_3 = 54.286$.

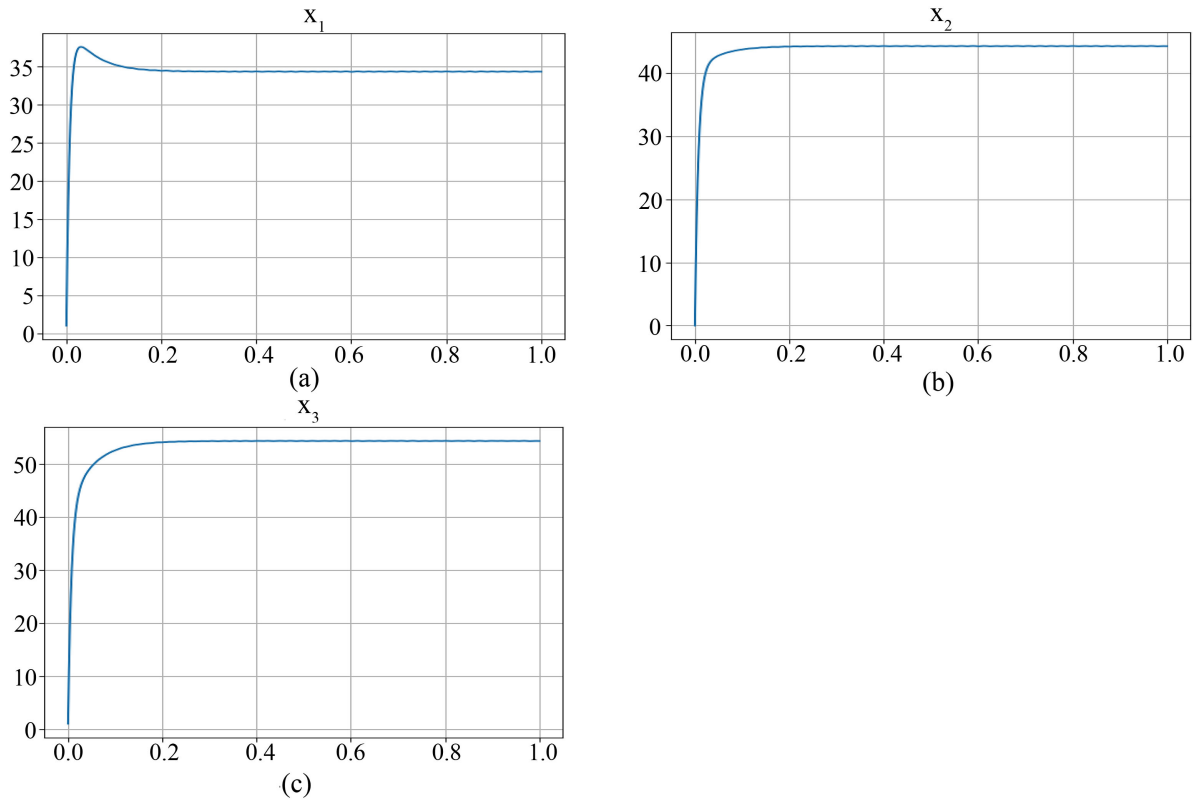


Figure 4. Graphical profile of the solutions to Problem 2.

From the values of our y, we can see that $y_0 = x_1 = 37.526$ did not satisfy the first constraint of the problem 2. We then adjusted our parameters upwards to have values; $\mu = 10$, $k_1 = 20$, $k_2 = 20$, $k_3 = 20$. This in turn gave us a new system of neural network ODE, viz;

$$\begin{cases} \dot{x}_1 = -620x_1 - 400x_2 - 200x_3 + 59600 \\ \dot{x}_2 = -600x_1 - 420x_2 - 200x_3 + 59800 \\ \dot{x}_3 = -600x_1 - 400x_2 - 220x_3 + 60000 \end{cases} \tag{24}$$

The steady state solution in this case is $x_1 = 48.515$, $x_2 = 52.295$ and $x_3 = 62.295$. We can see that at $y_0 = x_1 = 48.515$, did not still satisfy the first constraint of the problem 2. we then review the initial condition from (1, 0, 1) to (15, 10, 5) with the same parameters of $\mu = 10$, $k_1 = 20$, $k_2 = 20$, $k_3 = 20$. The set of neural network ODE is the same as before, *i.e.* Equations (24), but with initial conditions (15, 10, 5). The steady state solutions are as follows:

From Figure 5, the steady state solutions are now $x_1 = 50.243$, $x_2 = 52.295$ and $x_3 = 62.295$. In this case we can see that all the values have converged to the optimal solution of the equation and satisfying all the constraints of the problem.

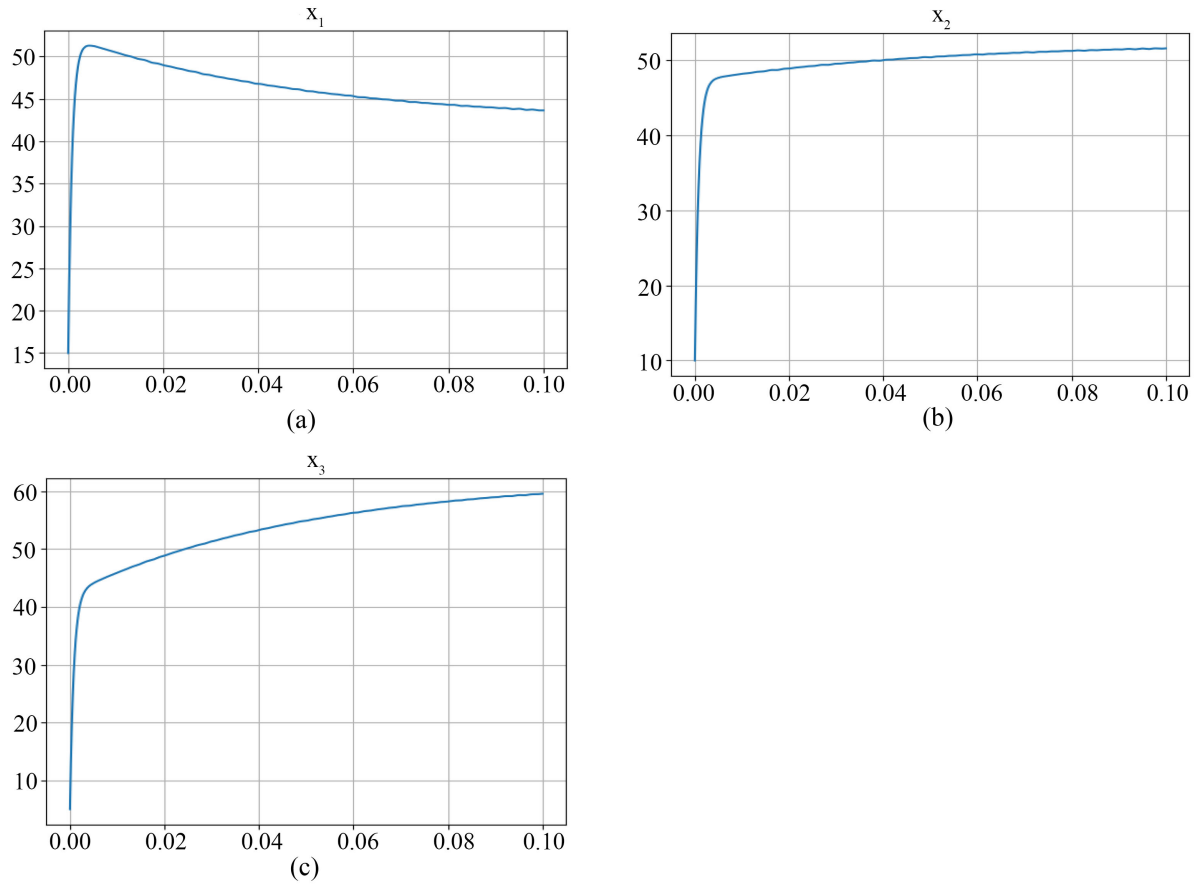


Figure 5. Graphical profile of the solutions to Problem 2 with adjusted parameters.

Problem 3 (Equality constraint)

Consider the equality constraint minimization problem

$$\text{Min } f(\boldsymbol{x}) = x_1^2 + x_2^2 \tag{25}$$

$$\text{Subject to } h(\boldsymbol{x}) = x_1 + x_2 - 1 = 0$$

According to section 8, the equivalent inequality-constrained problem becomes:

$$\text{Min } f(\boldsymbol{x}) = x_1^2 + x_2^2 \tag{26}$$

$$\text{Subject to } \begin{cases} g_1(\boldsymbol{x}) = x_1 + x_2 - 1 \geq 0 \\ g_2(\boldsymbol{x}) = -x_1 - x_2 + 1 \geq 0 \end{cases}$$

Following the differentiable penalty form given by Equation (10), we have

$$E(\boldsymbol{x}, k) = f(\boldsymbol{x}) + \frac{1}{2}\sum_{i=1}^{2} k_i \left[\min\left\{0, g_i(\boldsymbol{x})\right\}\right]^2 \tag{27}$$

here, $g_1(\boldsymbol{x}) = x_1 + x_2 - 1 \geq 0$ and $g_2(\boldsymbol{x}) = -x_1 - x_2 + 1 \geq 0$.

Since both constraints must be nonnegative, the penalty activates only when either becomes negative. The resulting ODE system can then be derived using Equation (12) with $m = 2$, S_1 and S_2 defined by Equation (13).

12. Conclusion

In this paper, we have been able to demonstrate how the neural network approach works in solving both equality and inequality non-linear programming problems as well as how to generate the system of equations that must correspond to the given number of variables. With the knowledge and understanding, we applied them in solving some constrained non-linear optimization problems. Using MathCAD, we were able to solve the system of equations developed by using the neural network gradient scheme. We presented our results graphically and compared it with the analytic solution. Our steps and results have shown that, for the problems tested, the neural network approach is more straightforward to implement and avoids some of the iterative assumptions of classical methods.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Cichockih, A. (1993) Introduction to Linear and Non-Linear Programming. Addison-Valley.

[2] Chong, E.K.P. and Zak, S.H. (2001) An Introduction to Optimization. 2nd Edition, John Wiley & Sons, Inc. https://onlinelibrary.wiley.com/doi/book/10.1002/9781118033340

[3] Orponen, P. (2000) An Overview of the Computational Power of Recurrent Neural Networks. In: Hyötniemi, H., Ed., Proceedings of the 9th Finnish AI Conference (STeP 2000—Millennium of AI): AI of Tomorrow, Symposium on Theory (Vol. 3, pp. 89-96), Espoo, 28-30 August 2000, Finnish Artificial Intelligence Society (Suomen Tekoälyseura), Vaasa.

[4] Richard, P.L. (1987) An Introduction to Computing with Neural Nets. *IEEE ASSP Magazine*, **4**, 4-22. https://ieeexplore.ieee.org/document/1165576

[5] Salchenberger, L.M., Cinar, E.M. and Lash, N.A. (1992) Neural Networks: A New Tool for Predicting Thrift Failures. *Decision Sciences*, **23**, 899-916. https://doi.org/10.1111/j.1540-5915.1992.tb00425.x

[6] Eckmiller, R. and von der Malsburg, C. (Eds.) (1989) Neural Computers. NATO ASI Series F: Computer and Systems Sciences, Vol. 41. Springer-Verlag.

[7] Cybenko, G. (1989) Approximation by Superpositions of a Sigmoidal Function. *Mathematics of Control, Signals, and Systems*, **2**, 303-314. https://doi.org/10.1007/bf02551274

[8] Hornik, K., Stinchcombe, M. and White, H. (1989) Multilayer Feedforward Networks Are Universal Approximators. *Neural Networks*, **2**, 359-366. https://doi.org/10.1016/0893-6080(89)90020-8

[9] Luenberger, D.G. (1973) Introduction to Linear and Non-Linear Programming. Addison-Wesley.

[10] Luenberger, D.G. (1984) Linear and Non-Linear Programming. Addison-Wesley.

[11] Cichocki, A. and Unbehauen, R. (1993) Neural Networks for Optimization and Signal Processing. John Wiley & Sons, Inc.

[12] Cichocki, A., Unbehauen, R., Weinzierl, K. and Hölzel, R. (1996) A New Neural Net-

work for Solving Linear Programming Problems. *European Journal of Operational Research*, **93**, 244-256. https://doi.org/10.1016/0377-2217(96)00044-6

[13] Valeri, M.M. and Nicholas, G.M. (2000) Neural Networks for Solving Constrained Optimization Problems. 4*th WSEAS Multi-Conference on Circuits*, *Systems*, *Communications and Computers* (*CSCC*'2000), Vouliagmeni (Athens), July 10-15, 1351-1359.

[14] Mathcad Version 14 (2007) PTC (Parametric Technology Corporation) Software Products. http://communications@ptc.com