Notes on Distributed Optimization

Venkata Ratnam Vadhri

2025

Contents

1 Introduction **3**
1.1 KKT conditions . . . 3
1.2 Steps to solve using Lagrangian . . . 4
1.3 Optimization with constraints . . . 5
1.3.1 1 equality constraint . . . 6
1.3.2 2 equality constraints . . . 7
1.3.3 1 equality and 1 inequality constraint . . . 9

2 Convex Convergence **11**
2.1 Convexity and smoothness . . . 11
2.2 Regression with Linear and Nonlinear Features . . . 13
2.2.1 Multi-feature linear relation . . . 13
2.2.2 Nonlinear Regression with Expanded Features . . . 14
2.3 Logistic regression . . . 16
2.3.1 Linearly separable boundary . . . 16
2.3.2 Non-Linear boundary . . . 18
2.4 Continuous-Time Optimization Inspired Gradient Flows . . . 19
2.4.1 Finite time . . . 20
2.4.2 Fixed time stability . . . 21
2.4.3 Prescribed time . . . 24
2.4.4 Projected fixed time . . . 25
2.4.5 Proximal fixed time limit . . . 26
2.4.6 Examples . . . 28
2.5 Discrete-Time Optimization Inspired Gradient Flows . . . 31
2.5.1 Finite-Time . . . 32
2.5.2 Fixed-Time . . . 33
2.5.3 Prescribed-Time . . . 50
2.5.4 Prescribed time + Fixed time . . . 51
2.5.5 Proximal fixed time . . . 52
2.6 Accelerating convergence . . . 54
2.6.1 Momentum-Based Methods (Unconstrained Optimization) . . . 54
2.6.2 Supporting Techniques (Stabilize Training, Not Direct Acceleration) . . . 56
2.6.3 Constrained Optimization Methods . . . 58
2.7 [todo] Gradient Fields . . . 64

3 Distributing optimization **64**
3.1 Dual Ascent Method . . . 64
3.2 ADDM . . . 68
3.2.1 Consensus example . . . 69
3.2.2 Sharing global objective example . . . 72
3.2.3 ATM withdrawl example . . . 76
3.2.4 Linear Regression Consensus . . . 79
3.2.5 Logistic regression . . . 81
3.3 Representation as graph . . . 84

4 Appendix **85**

1 Introduction

This document provides some examples on various combinations of optimization in conjunction with the types of constraints imposed. The standard primal convex optimization problem looks like the one below with the assumption that $f, \{g_i\}, \{h_j\}$ are convex functions and P^* is the optimal solution.

$$\min_{x \in R^n} f(x) \quad s.t \quad g(x) \leq 0; \quad \{i = 1, 2..n\}; \quad h(x) = 0 \quad \{j = 1, 2..m\}$$

The problem becomes more manageable with conversion into the Lagrangian form, which converts the constrained optimization problem into an unconstrained optimization problem with λ_i and μ_i as multipliers. If the inequality constraints g_i are not in the right format as in $g_i \leq 0$, then they need to be adjusted, for example $g_i(x) > 0 \rightarrow -g_i(x) \leq 0$.

$$\mathscr{L}(x_1, x_2, ...\lambda, \mu) = f(x_i) + \sum_{i=1}^{n} \lambda_i \cdot g_i(x) + \sum_{j=1}^{m} \mu_j \cdot h_j(x)$$

Component	**Mandatory Condition**	**Why It Is Mandatory**
Objective function $f(x)$	Convex	Needed for zero duality gap and valid KKT conditions. Ensures the dual gives tight bounds.
Inequality constraints $g_j(x) \leq 0$	Convex	Required to keep the feasible set convex and support strong duality.
Equality constraints $h_i(x) = 0$	Affine (linear)	Nonlinear equalities break convexity. Affine constraints preserve it.
Slater's condition	$\exists x : g_j(x) < 0,\ h_i(x) = 0$	Ensures strong duality in convex problems by preventing a duality gap.

Table 1: Constraints on components for valid Lagrangian duality

The optimal value obtained by solving the primal problem P^* and its Lagrange dual counterpart d^* is categorized as follows:

- **Strong Duality:** $P^* = d^*$ — primal and dual optimal values match.
- **Weak Duality:** $P^* \geq d^*$ — the dual value is a lower bound to the primal.

1.1 KKT conditions

At an optimal point, the constraints of the objective function are satisfied with respect to the objective function $f(x_1, x_2...)$ and a gradient cannot move in any direction. Given $(x_1^*, x_2^*..)$ is an optimal point, the gradient w.r.t. to each primal variable will be 0. This condition is called **stationarity**.

$$\nabla_{x_1} \mathscr{L}(x_1*, x_2*, ..., \lambda*, \mu*) = 0$$

$$\nabla_{x_2} \mathscr{L}(x_1*, x_2*, ..., \lambda*, \mu*) = 0$$

If the optimal solution, $(x_1^*, x_2^*..)$ is inside the feasibility set taking into account the inequality constraints, then the multipliers are 0 since they have no effect on the solution, also known as **complementary slackness**.

$$\lambda_i \cdot g_i(x_i) = 0$$

Once an optimal solution $(x_1^*, x_2^*, \dots)$ has been obtained, these values should be validated against the

constraints below. This requirement is known as **primal feasibility**.

$$g_j(x^*) \leq 0 \quad and \quad h_i(x^*) = 0$$

The dual variables must satisfy $\lambda^* \geq 0$ for all inequality constraints. This condition is called **dual feasibility**.

KKT Condition	**Mathematical Form**	**Role in Lagrangian Dual**	**If Violated → Recovery Strategy**
Stationarity	$\nabla_x \mathscr{L}(x^*, \lambda^*, \mu^*) = 0$	Ensures no descent direction; mandatory for valid dual solutions	Take gradient step or Newton-like update; recompute dual variables
Primal Feasibility	$h_i(x^*) = 0, \quad g_j(x^*) \leq 0$	Ensures the dual is defined only over feasible primal values; mandatory	Project back to feasible region; apply penalties or constraint enforcement
Dual Feasibility	$\mu_j^* \geq 0$	Maintains concavity and upper boundedness of the dual function; mandatory	Reset or clip dual variables; adjust search direction or step size
Complementary Slackness	$\mu_j^* \cdot g_j(x^*) = 0$	Links active constraints to non-zero multipliers; mandatory for strong duality	Update active set; zero out inactive multipliers or recompute constraint roles

Table 2: KKT conditions, their role in Lagrangian duality, and recovery strategies if violated

1.2 Steps to solve using Lagrangian

To solve constrained optimization problems, the Lagrangian method introduces multipliers to combine the objective and constraints. The solution proceeds by forming and analyzing the Lagrangian, solving the dual, and verifying KKT conditions.

1. Primal Problem $\min f(x)$ s.t. $g_j(x) \leq 0,\ h_i(x) = 0$	Original constrained optimization problem.
2. Form Lagrangian $\mathscr{L}(x, \lambda, \mu)$	Build Lagrangian using multipliers.
3. Stationarity $\nabla_x \mathscr{L} = 0$	Solve $\nabla_x \mathscr{L} = 0$ for x.
4. Dual Function $g(\lambda, \mu) = \inf_x \mathscr{L}$	Compute minimized Lagrangian as dual function.
5. Dual Problem $\max_{\lambda,\mu \geq 0} g(\lambda, \mu)$	Maximize dual over feasible multipliers.
6. Solve Dual λ^*, μ^*	Get optimal λ^*, μ^*.
7. Recover x^* from $\mathscr{L}(x, \lambda^*, \mu^*)$	Plug into Lagrangian to get x^*.
8. Verify KKT Conditions	Check stationarity, feasibility, slackness.

1.3 Optimization with constraints

Lets analyze the following function since it will remain the base function as the constraints are changed for couple of sections below.

$$f(x) = f(x_1) + f(x_2) \quad s.t \quad f(x_1) = (x-2)^2,\ f(x_2) = (x-5)^2$$

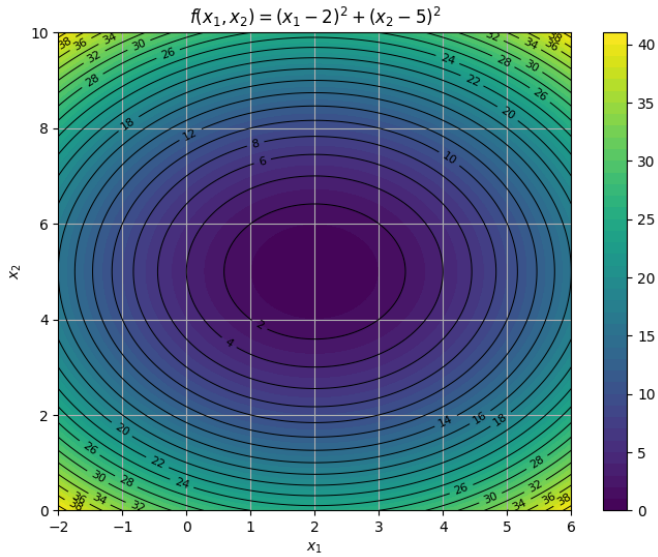


Figure 1: Plot of eq : $f(x1, x2) = (x_1 - 2)^2 + (x_2 - 5)^2$

The optimal point of the equation is at $(x_1, x_2) = (2, 5)$

Property	$f_1(x_1) = (x_1 - 2)^2$	$f_2(x_2) = (x_2 - 5)^2$	$f(x_1, x_2) = f_1 + f_2$
Type	Quadratic	Quadratic	Sum of quadratics
Domain	$\mathbb{R}$	$\mathbb{R}$	$\mathbb{R}^2$
Minimum Point	$x_1 = 2$	$x_2 = 5$	$(2, 5)$
Minimum Value	0	0	0
Gradient	$2(x_1 - 2)$	$2(x_2 - 5)$	$\begin{bmatrix} 2(x_1 - 2) \\ 2(x_2 - 5) \end{bmatrix}$
Hessian	2	2	$\begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$
Convex	Yes	Yes	Yes
Strictly Convex	Yes	Yes	Yes
Strongly Convex	Yes ($m = 2$)	Yes ($m = 2$)	Yes ($m = 2$)
Lipschitz Smooth	Yes ($L = 2$)	Yes ($L = 2$)	Yes ($L = 2$)
Separable	–	–	Yes

Table 3: Properties of f_1, f_2, and their sum $f(x_1, x_2)$

1.3.1 1 equality constraint

The following problem statement defines the optimization problem with an equality constraint.

$$\min_{r \in R^n} f(x) = \min_{r \in R^n} f(x_1) + f(x_2) \quad | \quad f(x_1) = (x - 2)^2,\ f(x_2) = (x - 5)^2 \quad | \quad x_1 + x_2 = 10;$$

The lagrangian form of the equation is as below.

$$
\begin{aligned}
\mathscr{L}(\lambda, x_1, x_2) &= \min_{x_1,x_2}\left[f(x) + \sum_{i=1}^{n} eqc_i(x)\right] \\
&= \min_{x_1,x_2}\left[(x_1-2)^2 + (x_2-5)^2 + \lambda(x_1+x_2-10)\right] \\
&= \min_{x_1,x_2}\left[(x_1-2)^2 + \lambda x_1\right] + \min_{x_2}\left[(x_2-5)^2 + \lambda x_2 - 10\lambda\right]
\end{aligned}
$$

In a distributed optimization setup, there are two agents each optimizing x_1 and x_2. To find the function used by both, we can follow the logic below.

$$
\nabla_{x_1} g(\lambda, x_1, x_2) = \frac{\partial}{\partial x_1}(x_1-2)^2 + \lambda x_1 = 0 \quad \rightarrow \quad 2x_1 - 4 + \lambda = 0 \quad \rightarrow \quad x_1 = \frac{4-\lambda}{2} = 2 - \frac{\lambda}{2}
$$

$$
\nabla_{x_2} g(\lambda, x_1, x_2) = \frac{\partial}{\partial x_2}(x_1-5)^2 + \lambda x_2 = 0 \quad \rightarrow \quad 2x_1 - 10 + \lambda = 0 \quad \rightarrow \quad x_1 = \frac{10-\lambda}{2} = 5 - \frac{\lambda}{2}
$$

Optimization steps

The $\nabla_{x_1} g(\lambda, x_1, x_2)$ and $\nabla_{x_2} g(\lambda, x_1, x_2)$ enable one to run the iteration with the constraint provided in an iterative logic. The algorithm to run an iterative optimization process as follows produces a value for λ which can then be used to calculate the values of x_1 and x_2 as shown in Fig. 1

$$
\nabla_{x_1} f(\lambda) = 2 - \frac{\lambda}{2}; \qquad \nabla_{x_2} f(\lambda) = 5 - \frac{\lambda}{2}
$$

Algorithm 1 Lagrange Multiplier Update Algorithm

```
Initialize λ ← 0
for each iteration do
    Compute gx_1 ← ∂f/∂x_1 (λ)
    Compute gx_2 ← ∂f/∂x_2 (λ)
    Compute loss ← gx_1 + gx_2 − 10
    Update λ ← λ + α · loss
end for
Compute x_1 and x_2 using final λ
```

1.3.2 2 equality constraints

The following problem statement defines the optimization problem with 2 equality constraints and the same optimization problem as 2.1

$$
f(x) = f(x_1) + f(x_2) \quad s.t \quad f(x_1) = (x-2)^2,\ f(x_2) = (x-5)^2
$$

$$
s.t \quad x_1 + x_2 = 10 \quad and \quad 2x_1 - x_2 = 3; \quad Objective \rightarrow \min_{r \in R^n} f(x)
$$

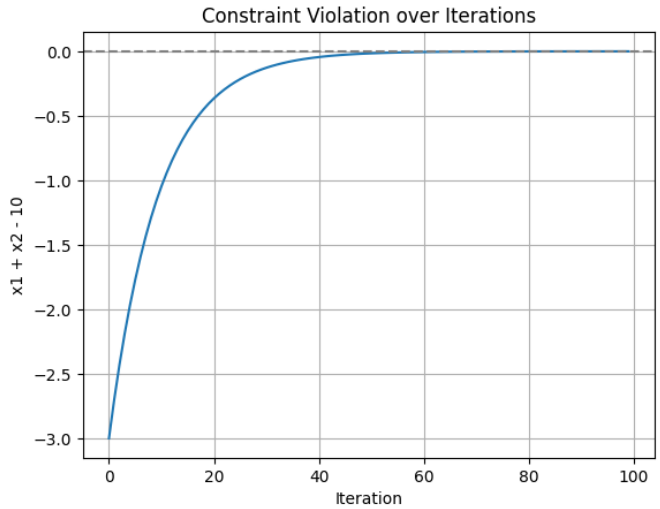


Figure 2: Iteration over 100 steps

We convert that into Lagrangian as below.

$$
\begin{aligned}
& g(\lambda_1, \lambda_2, x_1, x_2) = f(x) + \sum_{i=1}^{n} \text{eqc}_i(x) \\
\Rightarrow \quad & g(\lambda_1, \lambda_2, x_1, x_2) = (x_1 - 2)^2 + (x_2 - 5)^2 + \lambda_1(x_1 + x_2 - 10) + \lambda_2(2x_1 - x_2 - 3) \\
\Rightarrow \quad & g(\lambda_1, \lambda_2, x_1, x_2) = \left[\min_{x_1}(x_1 - 2)^2 + x_1(\lambda_1 + 2\lambda_2)\right] + \left[\min_{x_2}(x_2 - 5)^2 + x_2(\lambda_1 - \lambda_2)\right] \\
& \qquad - 10\lambda_1 - 3\lambda_2 \\
\Rightarrow \quad & g(\lambda_1, \lambda_2, x_1, x_2) = \min_{x_1}\left[(x_1^2 - 4x_1 + 4) + x_1(\lambda_1 + 2\lambda_2)\right] \\
& \qquad + \min_{x_2}\left[(x_2^2 - 10x_2 + 25) + x_2(\lambda_1 - \lambda_2)\right] - 10\lambda_1 - 3\lambda_2 \\
\Rightarrow \quad & g(\lambda_1, \lambda_2, x_1, x_2) = \min_{x_1}\left[x_1^2 + x_1(\lambda_1 + 2\lambda_2 - 4) + 4\right] \\
& \qquad + \min_{x_2}\left[x_2^2 + x_2(\lambda_1 - \lambda_2 - 10) + 25\right] - 10\lambda_1 - 3\lambda_2 \\
& \nabla_{x_1} g = 2x_1 + \lambda_1 + 2\lambda_2 - 4 = 0 \quad \Rightarrow \quad x_1 = -0.5(\lambda_1 + 2\lambda_2 - 4) \\
& \nabla_{x_2} g = 2x_2 + \lambda_1 - \lambda_2 - 10 = 0 \quad \Rightarrow \quad x_2 = -0.5(\lambda_1 - \lambda_2 - 10)
\end{aligned}
$$

Optimization steps

The $\nabla_{x_1} g(\lambda_1, \lambda_2, x_1, x_2)$ and $\nabla_{x_2} g(\lambda_1, \lambda_2, x_1, x_2)$ enable one to run the iteration with the constraint provided in an iterative logic. The algorithm to run an iterative optimization process as follows produces a value for λ_1 and λ_2 which can then be used to calculate the values of x_1 and x_2 as shown in Fig. 2

$$
x_1 = -\frac{\lambda_1}{2} - \lambda_2 + 2; \quad x_2 = -\frac{1}{2}(\lambda_1 - \lambda_2) + 5
$$

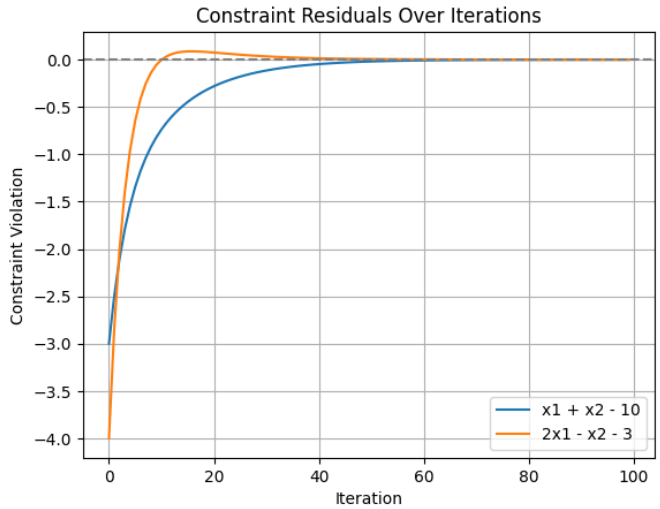


Figure 3: Iteration over 100 steps - 2 eq constraints

Algorithm 2 Dual Gradient Update Algorithm

Initialize: $\lambda_1 \leftarrow 0$, $\lambda_2 \leftarrow 0$
for each iteration **do**
  Compute gradients:
  $gx_1 \leftarrow \texttt{x1grad}(\lambda_1)$, $\quad gx_2 \leftarrow \texttt{x2grad}(\lambda_2)$
  Compute constraint losses:
  $\text{loss}_1 \leftarrow gx_1 + gx_2 - 10$, $\quad \text{loss}_2 \leftarrow 2 \cdot gx_1 - gx_2 - 3$
  Update dual variables:
  $\lambda_1 \leftarrow \lambda_1 + \alpha \cdot \text{loss}_1$, $\quad \lambda_2 \leftarrow \lambda_2 + \alpha \cdot \text{loss}_2$
end for
Output: Use final λ_1, λ_2 to compute x_1, x_2

1.3.3 1 equality and 1 inequality constraint

The following problem statement defines the optimization problem with 2 equality constraints and the same optimization problem as before where $x_1 + x_2 = 10$ and $x1 \geq 5$.

$$\min_{r \in R^n} f(x) \quad | \quad f(x) = f(x_1) + f(x_2) \quad | \quad f(x_1) = (x-2)^2, \quad f(x_2) = (x-5)^2$$

The Lagrangian dual form looks like the following.

$$\begin{aligned} g(\lambda, \mu, x_1, x_2) &= (x_1 - 2)^2 + (x_2 - 5)^2 + \lambda(10 - x_1 - x_2) + \mu(5 - x_1) \\ &= x_1^2 - 4x_1 + 4 + x_2^2 - 10x_2 + 25 + \lambda(10 - x_1 - x_2) + \mu(5 - x_1) \\ &= x_1^2 + x_2^2 + x_1(-4 - \lambda - \mu) + x_2(-10 - \lambda) + 10\lambda + 5\mu + 29 \end{aligned}$$

$$\nabla_{x_1} g = 2x_1 - 4 - \lambda - \mu = 0 \quad \Rightarrow \quad x_1 = 2 + \frac{\lambda + \mu}{2}$$

$$\nabla_{x_2} g = 2x_2 - 10 - \lambda = 0 \quad \Rightarrow \quad x_2 = 5 + \frac{\lambda}{2}$$

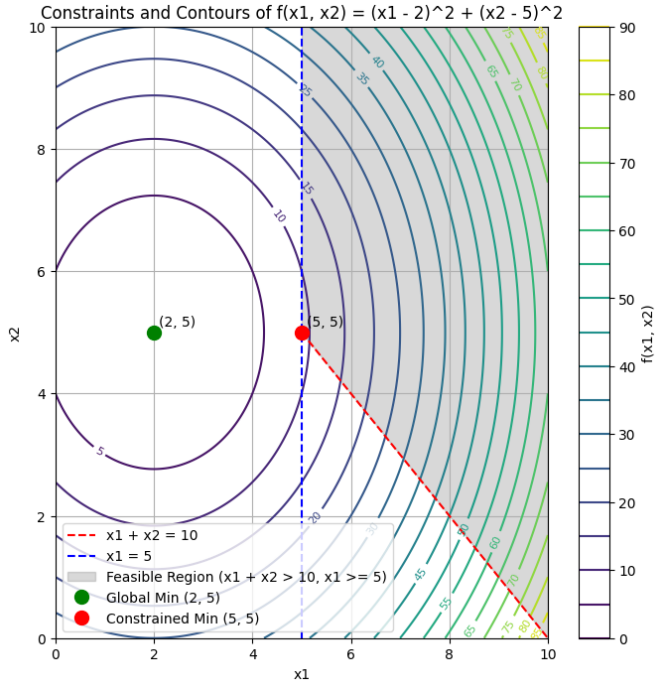


Figure 4: Plot with 2 constraints

Lets put the values of x_1 and x_2 in the original g(..) equation.

$$(x_1-2)^2=(2+\frac{(\lambda+\mu}{2}-2)^2=\frac{(\lambda+\mu)^2}{4}\ ;\quad (x_2-5)^2=(5+\frac{\lambda}{2}-5)^2=\frac{\lambda^2}{4}$$

$$(x_1-2)^2+(x_2-5)^2=\frac{\lambda^2+(\lambda+\mu)^2}{4}\qquad ...(1)$$

$$\lambda(10-x_1-x_2)=\lambda\left(10-2-\frac{\lambda+\mu}{2}-5-\frac{\lambda}{2}\right)=\lambda\left(3-\frac{2\lambda+\mu}{2}\right)=\frac{6\lambda-2\lambda^2-\lambda\mu}{2}\qquad ...(2)$$

$$\mu(5-x_1)=\mu\left(5-2-\frac{(\lambda+\mu)}{2}\right)=\mu\left(3-\frac{(\lambda+\mu)}{2}\right)=\frac{6\mu-\lambda\mu-\mu^2}{2}\qquad ...(3)$$

Adding (1), (2), (3) to complete the full equation.

$$\frac{(\lambda+\mu)^2}{4}+\frac{6\lambda-2\lambda^2-\lambda\mu}{2}+\frac{6\mu-\lambda\mu-\mu^2}{2}+\frac{\lambda^2}{4}$$

$$=\frac{1}{4}\left[\lambda^2+\mu^2+2\lambda\mu+12\lambda-4\lambda^2-2\lambda\mu+12\mu-2\lambda\mu-2\mu^2+\lambda^2\right]$$

$$=\frac{1}{4}\left[-2\lambda^2-\mu^2-2\lambda\mu+12\lambda+12\mu\right]$$

To get the values of λ and μ we need to take partial derivatives to find the maximum value.

$$-2\lambda-\mu+6=0\quad |\quad -\mu-\lambda+6=0$$

If we solve the above 2 equations, we get μ = 6 and λ = 0; that would yield $(x_1,x_2)=(5,5)$

Optimization Steps The gradients $\nabla_{x_1}g(...)$ and $\nabla_{x_2}g(...)$ guide the iterative update of the dual variables λ and μ under the constraints provided. The resulting values of λ and μ are then used to compute the optimal values of x_1 and x_2, as illustrated in Fig. 5

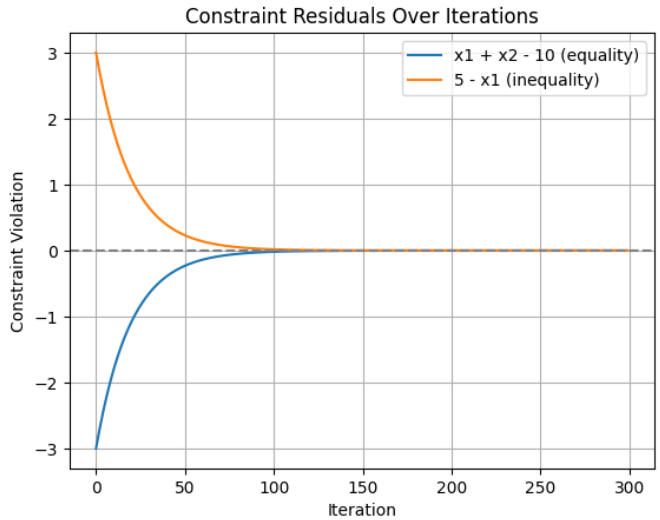


Figure 5: λ and μ convergence for 500 iterations

$$x_1(\lambda,\mu) = 2 + \frac{\lambda+\mu}{2} \quad | \quad x_2(\lambda,\mu) = 5 + \frac{\lambda}{2}$$

Algorithm 3 Dual Variable Update for Constrained Optimization

1: **Initialize:** $\lambda \leftarrow 0$, $\mu \leftarrow 0$
2: **for** each iteration **do**
3: Compute primal variables:

$x_1 \leftarrow 2 + \frac{\lambda+\mu}{2} \quad | \quad x_2 \leftarrow 5 + \frac{\lambda}{2}$

4: Compute residuals:

$c_1 \leftarrow x_1 + x_2 - 10 \quad | \quad c_2 \leftarrow 5 - x_1$

5: Update dual variables:

$\lambda \leftarrow \max(0, \lambda + \alpha \cdot c_1) \quad | \quad \mu \leftarrow \max(0, \mu + \alpha \cdot c_2)$

6: **end for**
7: **Output:** Final solution x_1, x_2 computed from last λ, μ

This approach enables a dual decomposition scheme in which each variable is updated based on local gradient information relative to the dual multipliers.

2 Convex Convergence

2.1 Convexity and smoothness

The problem statements that are having convex functions can be measured based on Strong convexity and L-Smooth behavior which are represented by μ and L respectively. If we consider a ball that is drop from the edge of the curve, the following physical properties can be used for intuition.

A differentiable function $f : R^n \to R$ is μ-strong and L-smooth for the following mathematical definition where L is upper bound and μ is the lower bound of curvature.

$$f(y) \leq f(x) + \nabla f(x)^T (y - x) + \frac{\mu}{2} ||y - x||^2$$

Concept	Meaning
Curvature	How "steep" the bowl is (second derivative, $f''(x)$)
μ (strong convexity)	The minimum curvature — ensures the bowl is never too flat
L (smoothness)	The maximum curvature — steepest acceleration allowed
Gradient	Slope at a point — tells the ball which way to roll
Gradient Descent	Ball rolling with friction, stepping downhill
Strong μ	The bowl is tighter — the ball moves more decisively and converges quickly
Small μ	The bowl is shallower — the ball rolls slowly, may drift or zigzag

Table 4: Interpretation of convex optimization concepts using the ball-in-bowl analogy.

$$f(y) \geq f(x) + \nabla f(x)^T (y - x) + \frac{L}{2}||y - x||^2$$

The value of μ impacts the speed at which the convergence to optimal value would happen and L would influence how fast from a starting point far would the function come closer to the optimal point. The following two graphs represent convex objective functions and their steps involved in convergence using gradient descent function.

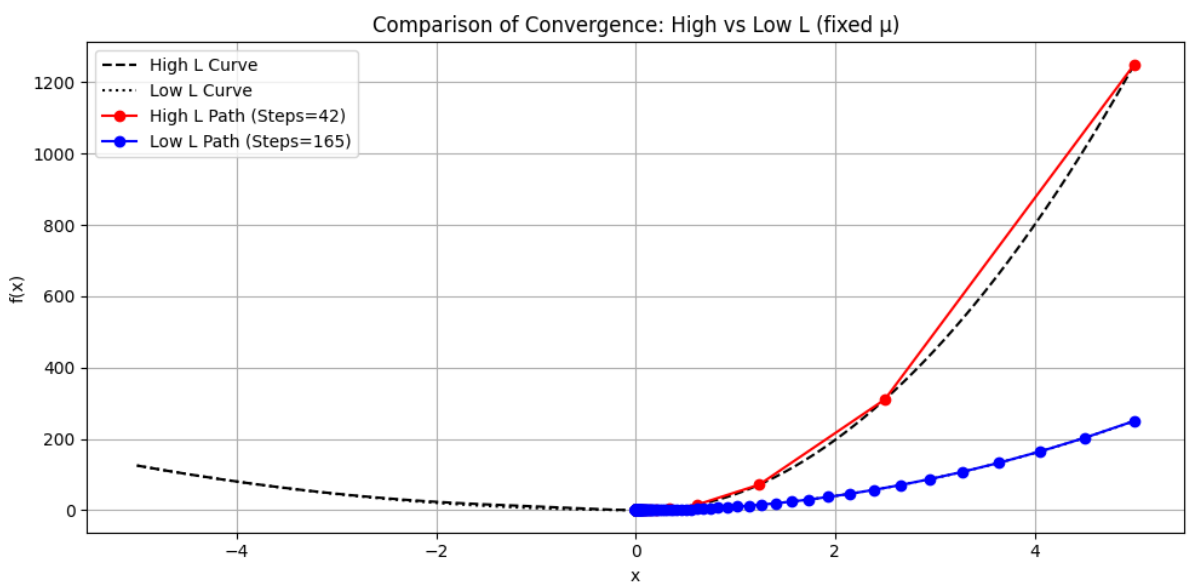


Figure 6: Different values of L

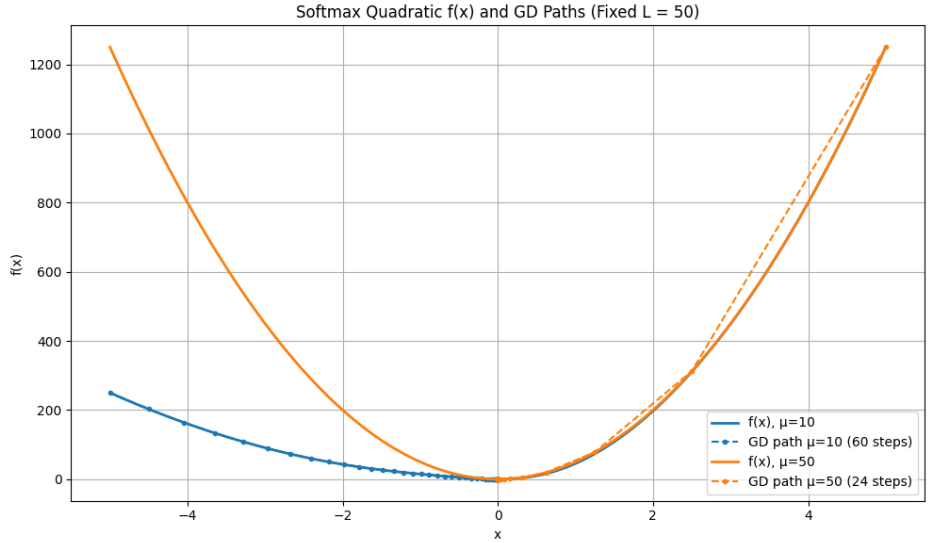


Figure 7: Different values of μ

2.2 Regression with Linear and Nonlinear Features

The following is a multi-feature linear regression problem specified as multiple features with linear and non-linear relationship.

2.2.1 Multi-feature linear relation

Lets consider two features x_1 and x_2 linearly related to y as the equation below.

$$(x_1, x_2) \in \mathbb{R}^2; \hat{y} = w_1 \cdot x_1 + w_2 \cdot x_2 + b$$

The loss function can be modeled with Mean Square Error (MSE) loss as below and this function is convex.

$$\mathscr{L}(w_1, w_2, b) = \frac{1}{N} \sum (w_1.x_1^{(i)} + w_2.x_2^{(i)} + b - y^{(i)})^2$$

Gradient and Update Rules
The gradients of the Mean Squared Error (MSE) loss with respect to the model parameters are:

$$\frac{\partial \mathscr{L}}{\partial w_1} = \frac{2}{N} \sum_{i=1}^{N} \left(\hat{y}^{(i)} - y^{(i)}\right) x_1^{(i)} \quad \frac{\partial \mathscr{L}}{\partial w_2} = \frac{2}{N} \sum_{i=1}^{N} \left(\hat{y}^{(i)} - y^{(i)}\right) x_2^{(i)} \quad \frac{\partial \mathscr{L}}{\partial b} = \frac{2}{N} \sum_{i=1}^{N} \left(\hat{y}^{(i)} - y^{(i)}\right)$$

Using gradient descent, we update the parameters in the opposite direction of the gradient:

$$w_1 \leftarrow w_1 - \eta \cdot \frac{\partial \mathscr{L}}{\partial w_1} \quad w_2 \leftarrow w_2 - \eta \cdot \frac{\partial \mathscr{L}}{\partial w_2} \quad b \leftarrow b - \eta \cdot \frac{\partial \mathscr{L}}{\partial b}$$

Example : $\hat{y} = 2x_1 - 1.5x_2 + 5;\ w_1 = 2.00098,\ w_2 = -1.51803,\ b = 5.08202$ with final loss $\mathscr{L} = 0.96806$

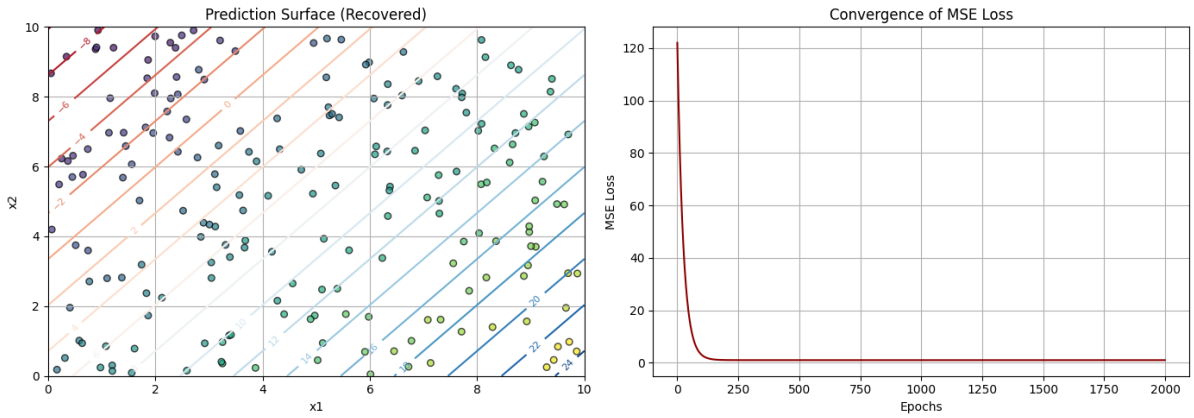


Figure 8: Linear regression with 2 features - Prediction surface and convergence

Algorithm 4 Gradient Descent with MSE Loss (Two Features)

Input: Training data $\{(x_{1i}, x_{2i}, y_i)\}$, learning rate η
Normalize features: $x_1 \leftarrow \frac{x_1-\mu_1}{\sigma_1}, \quad x_2 \leftarrow \frac{x_2-\mu_2}{\sigma_2}$
Initialize weights w_1, w_2 and bias b with small random values
for each epoch **do**
    Compute predictions: $\hat{y}_i = w_1 x_{1i} + w_2 x_{2i} + b$
    Compute loss: $\text{MSE} = \frac{1}{N}\sum_i (\hat{y}_i - y_i)^2$
    Compute gradients:

$$\frac{\partial \mathscr{L}}{\partial w_1} = \frac{2}{N}\sum_i (\hat{y}_i - y_i)x_{1i}$$
$$\frac{\partial \mathscr{L}}{\partial w_2} = \frac{2}{N}\sum_i (\hat{y}_i - y_i)x_{2i}$$
$$\frac{\partial \mathscr{L}}{\partial b} = \frac{2}{N}\sum_i (\hat{y}_i - y_i)$$

    Update parameters:

$$w_1 \leftarrow w_1 - \eta \cdot \frac{\partial \mathscr{L}}{\partial w_1}$$
$$w_2 \leftarrow w_2 - \eta \cdot \frac{\partial \mathscr{L}}{\partial w_2}$$
$$b \leftarrow b - \eta \cdot \frac{\partial \mathscr{L}}{\partial b}$$

end for
Rescale weights and bias to match original input scale
Output: Final weights w_1, w_2, and bias b

2.2.2 Nonlinear Regression with Expanded Features

Input: $x_1, x_2 \in \mathbb{R}$ (scalar features)
Label: $y \in \mathbb{R}$

Subject to:
The target value y is generated using a nonlinear function of the inputs where $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$ represents Gaussian noise.

$$y_i = x_{1i}^2 + 2 \cdot x_{2i}^2 + 3 \cdot x_{1i} \cdot x_{2i} + 4 \cdot x_{1i} + 5 \cdot x_{2i} + \epsilon_i$$

The objective is to learn parameters $(w_1, w_2, w_3, w_4, w_5, b)$ for the model using gradient descent to minimize the mean squared error loss between $\hat{y}_i$ and y_i.

$$\hat{y}_i = w_1 x_{1i}^2 + w_2 x_{2i}^2 + w_3 x_{1i} x_{2i} + w_4 x_{1i} + w_5 x_{2i} + b$$

Algorithm 5 Gradient descent for nonlinear regression

Initialize: Weights w_1, w_2, w_3, w_4, w_5, bias b with small random values
for each epoch **do**
  for each data point (x_{1i}, x_{2i}, y_i) **do**
    Compute prediction:

$$\hat{y}_i = w_1 x_{1i}^2 + w_2 x_{2i}^2 + w_3 x_{1i} x_{2i} + w_4 x_{1i} + w_5 x_{2i} + b$$

    Compute error:

$$e_i = \hat{y}_i - y_i$$

  end for
  Compute gradients:

$$\frac{\partial \mathscr{L}}{\partial w_1} = \frac{2}{N} \sum_i e_i x_{1i}^2$$
$$\frac{\partial \mathscr{L}}{\partial w_2} = \frac{2}{N} \sum_i e_i x_{2i}^2$$
$$\frac{\partial \mathscr{L}}{\partial w_3} = \frac{2}{N} \sum_i e_i x_{1i} x_{2i}$$
$$\frac{\partial \mathscr{L}}{\partial w_4} = \frac{2}{N} \sum_i e_i x_{1i}$$
$$\frac{\partial \mathscr{L}}{\partial w_5} = \frac{2}{N} \sum_i e_i x_{2i}$$
$$\frac{\partial \mathscr{L}}{\partial b} = \frac{2}{N} \sum_i e_i$$

  Update parameters:

$$w_j \leftarrow w_j - \eta \cdot \frac{\partial \mathscr{L}}{\partial w_j} \text{ for } j = 1 \text{ to } 5$$
$$b \leftarrow b - \eta \cdot \frac{\partial \mathscr{L}}{\partial b}$$

end for
Output: Trained weights w_1 to w_5, bias b

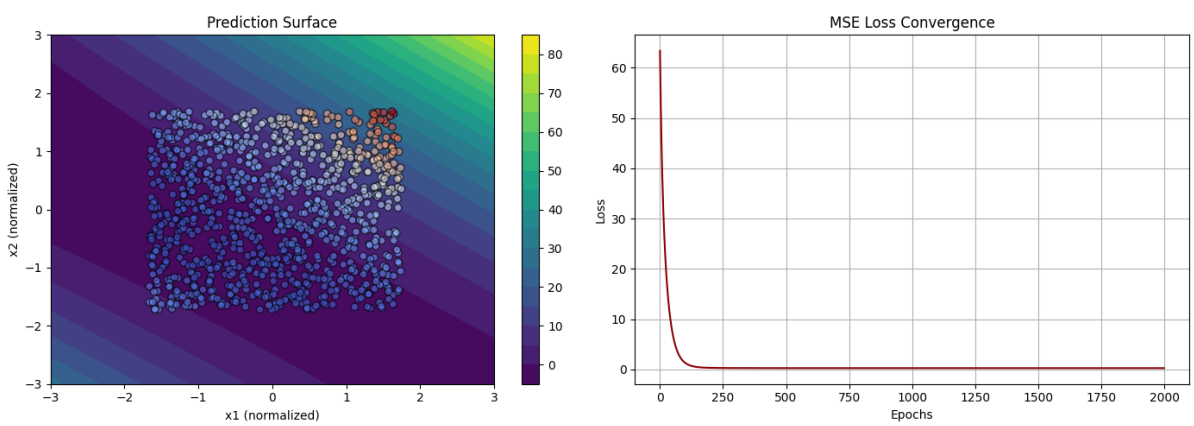


Figure 9: Non linear features and convergence

2.3 Logistic regression

The following is a multi-class classification please with data with linear and non-linear separable boundary.

2.3.1 Linearly separable boundary

Input: $x \in \mathbb{R}$ (scalar)
Label: $y \in \{1, 2\}$

Subject to:
$x_i \leq 50 \Rightarrow y = 1; \quad x_i > 50 \Rightarrow y = 2$

Inorder to convert the above problem into a logistic regression model, consider the optimization variables (w_i, b_i) for each class $i = \{0, 1, 2\}$. The relationship between the output class and input will take the form of affine function which represents the decision boundary between classes as a hyperplane:

$$z_i(x) = w_i \cdot x + b_i$$

Each affine result or logit should be translated into probability of how much an input x_i is close to a class. This transformation can be done using softmax for multi-class classification cases.

$$p_i = \frac{e^{z_i}}{\sum_{i=0}^{j} e^{z_j}}$$

In summary the model is as below:
$z_i = w_i x + b_i \quad \text{for } i \in \{1, 2\}$
$p_i = \frac{e^{z_i}}{\sum_{j=1}^{2} e^{z_j}}$
$\hat{y} = \arg\max_i p_i$

The function that would solve the problem hence is the class that is having maximum probability as in the equation below. The function is not differentiable hence the loss function or $\nabla f(x)$ from gradient descent is replaced with cross entropy loss. Cross entropy loss measures the deviation of a probability distribution of output classes probabilities with labeled probabilities. Since the loss function is convex with respect to W and b, we can apply convex optimization techniques such as gradient descent, even though the decision rule introduces a sharp boundary in input space.

$$loss = -\sum_i p_label_i \cdot log(p_predicted)$$

$$W_{k+1} = W_k - \eta \cdot loss; \quad b_{k+1} = b_k - \eta \cdot loss$$

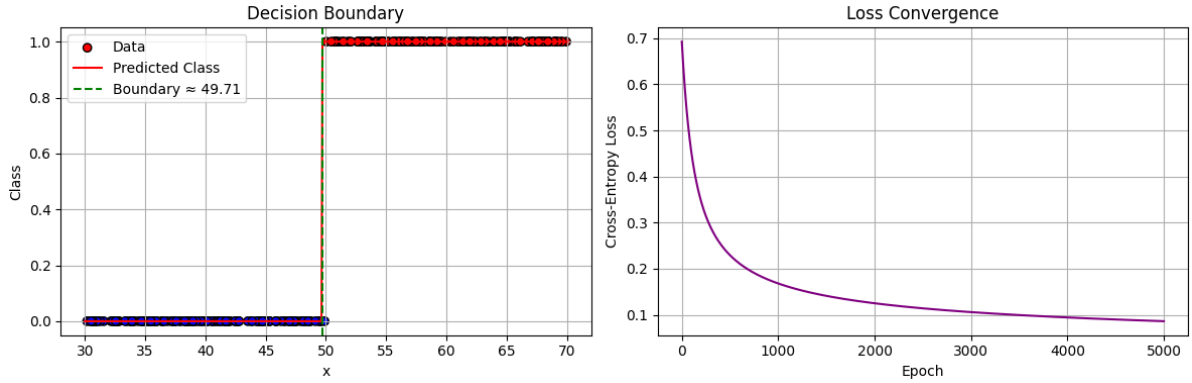


Figure 10: Softmax 1D - convergence and decision boundary

Algorithm 6 Softmax Classifier for 1D Input with Decision Boundary

Input: Scalar inputs $x_i \in \mathbb{R}$, labels $y_i \in \{0, 1\}$
Normalize inputs: $x_i \leftarrow \dfrac{x_i - \mu}{\sigma}$
One-hot encode labels: $Y_i \in \{[1, 0], [0, 1]\}$
Initialize weights $W \in \mathbb{R}^{1\times 2}$, biases $b \in \mathbb{R}^{1\times 2}$
for each epoch $t = 1$ to T **do**
  Compute logits: $Z \leftarrow xW + b$
  Apply softmax: $P_i \leftarrow \dfrac{e^{Z_i}}{\sum_j e^{Z_j}}$
  Compute loss: loss $\leftarrow -\sum_i Y_i \cdot \log(P_i)$
  Compute gradients:
    $dZ \leftarrow P - Y$
    $dW \leftarrow \dfrac{1}{N} x^\top dZ$
    $db \leftarrow \dfrac{1}{N} \sum dZ$
  Update parameters:
    $W \leftarrow W - \eta \cdot dW$
    $b \leftarrow b - \eta \cdot db$
end for
Compute decision boundary in normalized space: $x_{\text{boundary}} \leftarrow \dfrac{b_1 - b_0}{w_0 - w_1}$
Convert to original scale: $x^* \leftarrow x_{\text{boundary}} \cdot \sigma + \mu$
Output: Final model parameters, decision boundary x^*

If there are more classes, the array of W and b would expand to calculate more decision boundaries.

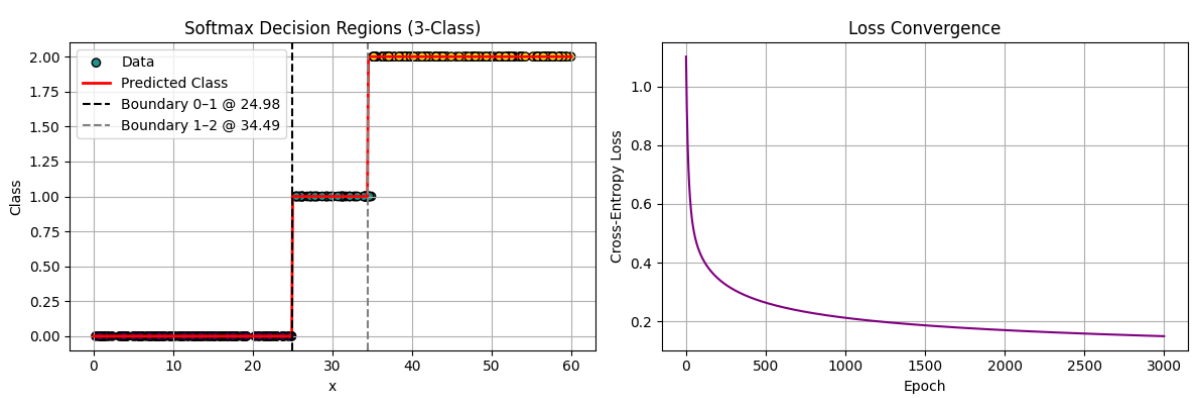


Figure 11: Decision boundary - 3 classes

2.3.2 Non-Linear boundary

The function that represents the problem could be in the form of a quadratic curvature (circle, ellipse etc). In order to represent a case like that, lets follow the problem statement below.

Input: $x_1, x_2 \in \mathbb{R}$ (scalar)
Label: $y \in \{1, 2, 3\}$

Subject to:
$x_{1i} + x_{2i}^2 \leq 1 \Rightarrow y = 1$
$1 < x_{1i} + x_{2i}^2 < 2 \Rightarrow y = 2$
$x_{1i} + x_{2i}^2 \geq 2 \Rightarrow y = 3$

The function that represents the data and class constraints is modeled using a general quadratic curvature:

$$f(x) = W^T \phi(x) + b \quad \text{where} \quad \phi(x) = [x_1, x_2, x_1^2, x_2^2, x_1 x_2]$$

This nonlinear feature transformation allows the model to learn curved decision boundaries such as parabolas, ellipses, or other conic sections. Although the function $f(x)$ is nonlinear in input space, it remains linear in the parameters W and b. Hence, the model is trained using a cross-entropy loss:

$$\mathscr{L}(W, b) = -\sum_{i=1}^{N} \sum_{k=1}^{C} y_{ik} \log \hat{p}_{ik}$$
$$\text{where} \quad \hat{p}_i = \text{softmax}(W^T \phi(x_i) + b)$$
$$\text{with} \quad i \in \{1, 2, \ldots, N\}, \quad k \in \{1, 2, \ldots, C\}$$

Since this loss function is convex with respect to W and b, we can apply convex optimization techniques such as gradient descent, even though the class boundaries in input space are nonlinear.

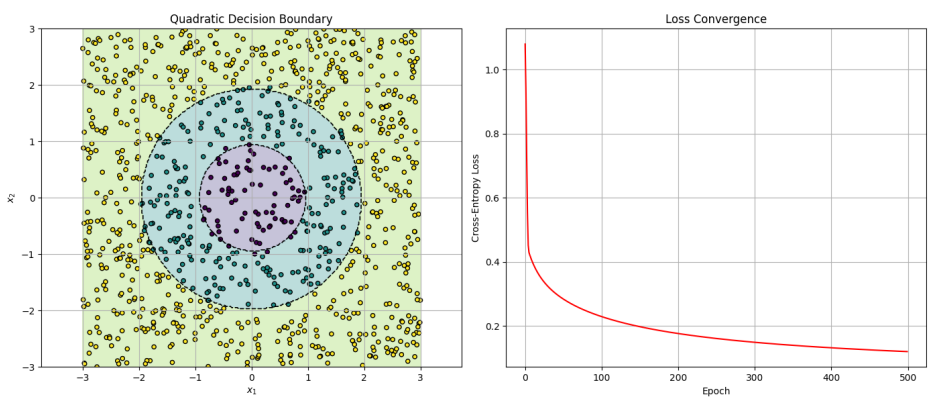


Figure 12: Softmax 2D, 3 classes $x1^2 + x_2^2$ - Convergence and decision boundary

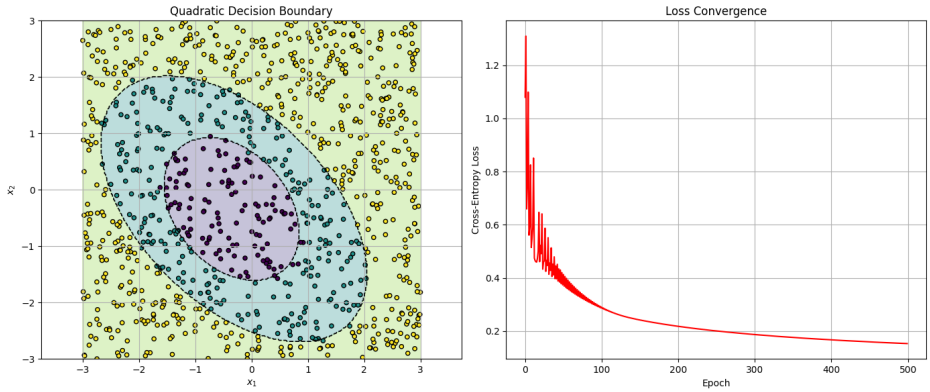


Figure 13: Softmax 2D, 3 classes $x_1^2 + x_2^2 + x_1x_2 + x_1 + x_2$ - Convergence and decision boundary

Algorithm 7 Quadratic Softmax Classifier

Input: Data points $x_i \in \mathbb{R}^2$, labels $y_i \in \{0, 1, 2\}$
Expand features: $\phi(x_i) = [x_1, x_2, x_1^2, x_2^2, x_1x_2]$
Initialize weights W and biases b with small random values
for each epoch **do**
    Compute logits: $Z \leftarrow \phi(X)W + b$
    Apply softmax: $\hat{p} \leftarrow \text{softmax}(Z)$
    Compute cross-entropy loss between $\hat{p}$ and y
    Compute gradients ∇_W, ∇_b
    Update weights: $W \leftarrow W - \eta \cdot \nabla_W$
    Update biases: $b \leftarrow b - \eta \cdot \nabla_b$
end for
Output: Trained weights W, biases b

2.4 Continuous-Time Optimization Inspired Gradient Flows

To accelerate convergence we can use stability methods that adjust the gradient flow and to understand the differences we will use a shopping cart analogy.

You're pushing a shopping cart down a narrow aisle. Suddenly, you see a small child run into your path about 5 meters ahead. You must stop safely and quickly, considering constraints like aisle width, floor friction, and possibly limited handle grip (your braking input).

GD Method	Shopping Cart Example	Insight
Finite-Time	You brake harder if you were pushing faster.	Braking time depends on initial speed.
Fixed-Time (FxTS)	You must stop within 1 second, no matter how fast you were pushing.	Guarantees a fixed upper bound on stopping time.
Newton Fixed-Time	You sense the floor is slippery, so you modulate braking force and posture accordingly.	Accounts for surface curvature and adapts force accordingly.
Prescribed-Time	You plan to stop over 3 seconds by reducing push effort gradually.	Stops on a scheduled decay plan, possibly independent of urgency.
Projected FxTS	You brake while ensuring the cart doesn't veer outside the aisle.	Maintains feasibility under constraints (aisle, shelves).
Proximal FxTS	Wearing gloves, your braking force is limited, so you must stop using minimal or sparse effort.	Optimizes under non-smooth control or actuator limits.

2.4.1 Finite time

A car is moving at 60 km/h (16.67 m/s), and detects an obstacle 100 meters ahead. We want the car to come to a complete stop before hitting the obstacle, and we'll use a finite-time braking model. At no deceleration, the car would hit the obstacle at 5.998s.

$$T_{collision} = \frac{D_{max}}{v_0} = \frac{100m}{16.67\ m/s} = 5.998$$

A finite-time system models the non-linear dynamic system to come to a state (in this case, the vehicle to stop) in a finite time and not at infinity. We can model this as the equation below.

$$\dot{v(t)} = -k \cdot v(t)^{\alpha}$$

where k is breaking distance and α is non-linearity parameter that is constant for a system. α can also be interpreted as parameter that represents break response at different speeds.

Solution to the dynamic equation

$$\frac{dv(t)}{dt} = -k \cdot v(t)^{\alpha}$$

$$d\ v(t) \cdot v(t)^{-\alpha} = -k\ dt$$

Lets integrate this over the interval of v(0) to v(t) to avoid C.

$$\int_{v(0)}^{v(t)} d\ v(t) \cdot v(t)^{-\alpha} = \int -k \cdot dt$$

$$\frac{v(t)^{(1-\alpha)}}{1-\alpha}\Bigg|_{v(0)}^{v(t)} = -k \cdot t$$

$$\frac{v(t)^{(1-\alpha)} - v(0)^{(1-\alpha)}}{1-\alpha} = -k \cdot t$$

$$v(t) = \left[v(0)^{(1-\alpha)} - k \cdot (1-\alpha) \cdot t\right]^{\frac{1}{1-\alpha}}$$

If the end goal is stopping, then v(t) = 0;

$$T_{stopping} = \frac{v(0)^{(1-\alpha)}}{k \cdot (1-\alpha)}$$

Algorithm 8 Finite-Time Braking Model (Search over k)

Input: Initial speed v_0, obstacle distance D_{obstacle}, braking exponent α, braking strength range $k \in [k_{\min}, k_{\max}]$
for each braking gain k in range **do**
  Initialize $v \leftarrow v_0$, $x \leftarrow 0$, $t \leftarrow 0$
  while $v > 0$ and $t < t_{\max}$ **do**
    Update v using finite-time model: $v \leftarrow \left[\max(v_0^{1-\alpha} - k(1-\alpha)t, 0)\right]^{1/(1-\alpha)}$
    Update distance: $x \leftarrow x + v \cdot \Delta t$
    Update time: $t \leftarrow t + \Delta t$
  end while
  if $x \leq D_{\text{obstacle}}$ **then**
    Mark k as safe
  else
    Mark k as unsafe
  end if
end for
Output: All safe k values and minimum k required for safe stop

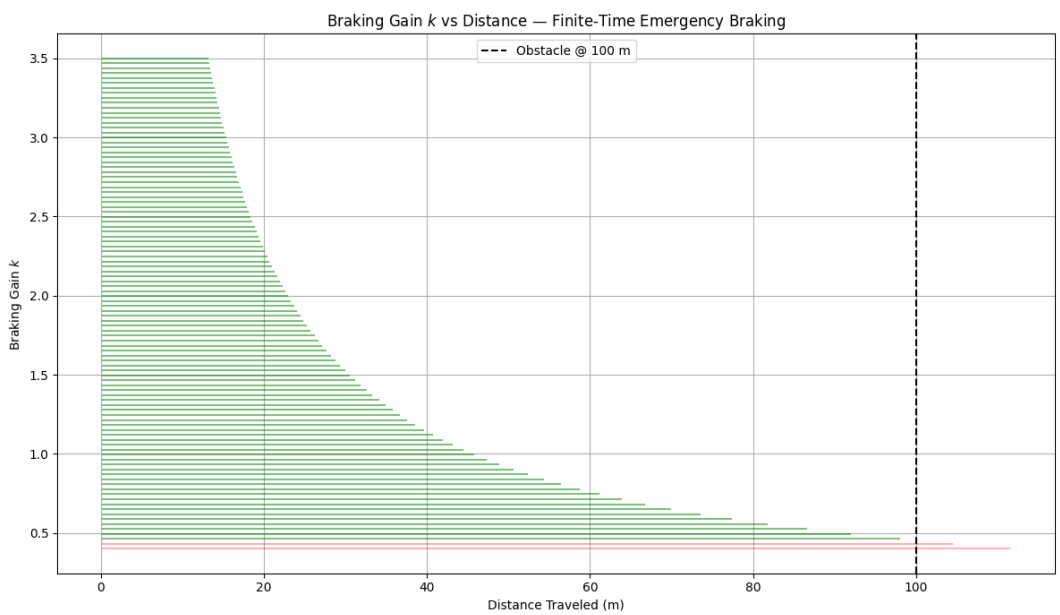


Figure 14: Finite time - Breaking example

Finite time modeling ensures that the vehicle stops eventually but could violate the objective if starting condition does not allow stopping.

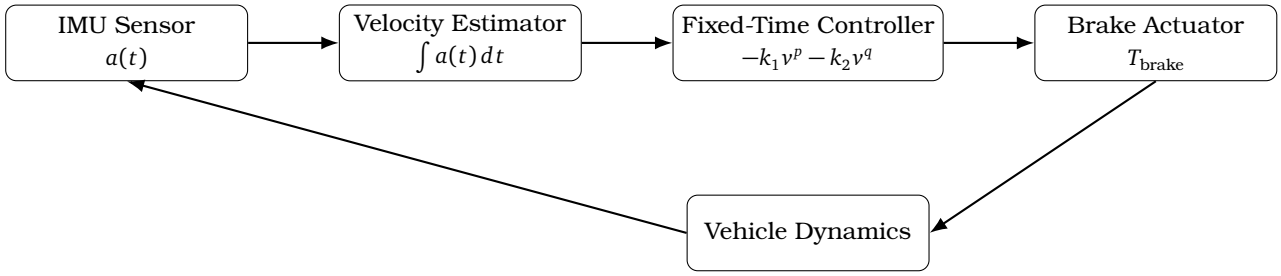
2.4.2 Fixed time stability

A car is moving at ≤ 100 km/h, and detects an obstacle 100 meters ahead. We want the car to come to a complete stop before hitting the obstacle, and we'll use a fixed-time braking model in 2s. The model close to real-world for that scenario would be to have an IMU sensor record acceleration which gets translated into velocity. The dynamic equation that represents that is as below.

$$\dot{v(t)} = a(t) = -k_1 \cdot v^p - k_2 \cdot v^q$$

A car must come to a complete stop within a fixed time budget $T_{\max}$, following the braking dynamics:

$$\dot{v}(t) = -k_1 v^p - k_2 v^q$$



To ensure **fixed-time convergence** (i.e., stopping in a time that is independent of the initial speed v_0), the parameters must satisfy:

- $0 < p < 1$ for strong convergence near $v = 0$
- $q > 1$ for strong braking at high speeds
- $k_1, k_2 > 0$ are braking gain coefficients to be selected

The theoretical **upper bound on convergence time** T is given by:

$$T \leq \frac{1}{k_1(1-p)} + \frac{1}{k_2(q-1)}$$

Given a desired $T_{\max}$ and fixed values for p and q, this inequality helps us find suitable values for k_1 and k_2.

Example: Let $T_{\max} = 2$ seconds, $p = 0.5$, and $q = 2$. Then:

$$T \leq \frac{1}{k_1(1-0.5)} + \frac{1}{k_2(2-1)} = \frac{2}{k_1} + \frac{1}{k_2}$$

To ensure the system stops in time, we must choose (k_1, k_2) such that:

$$\frac{2}{k_1} + \frac{1}{k_2} \leq 2$$

One valid choice: $k_1 = 2, \quad k_2 = 1$
This selection guarantees that the vehicle will stop in under 2 seconds, assuming the physical braking system can implement these gains.

Solution for the dynamic eq

$$\dot{v(t)} = -k_1 \cdot v^p - k_2 \cdot v^q$$

$$\frac{dv(t)}{dt} = -k_1 \cdot v^p - k_2 \cdot v^q$$

$$\frac{dv(t)}{k_1 \cdot v^p + k_2 \cdot v^q} = -dt$$

$$\int_{v0}^{0} \frac{dv(t)}{k_1 \cdot v^p + k_2 \cdot v^q} = -\int dt$$

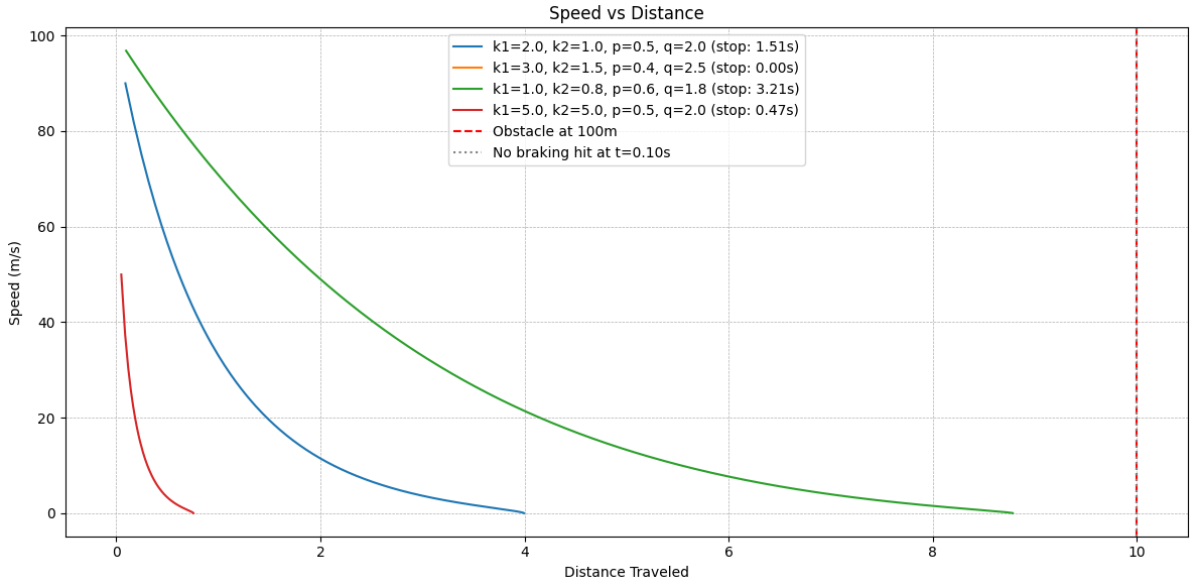


Figure 15: Fixed time

$$\int_{v0}^{0} \frac{dv(t)}{k_1 \cdot v^p + k_2 \cdot v^q} = -T$$

$$T = \int_{0}^{v0} \frac{dv(t)}{k_1 \cdot v^p + k_2 \cdot v^q}$$

The solution for this is in sync with the equation below.

$$T \leq \frac{1}{k_1 \cdot (1-p)} + \frac{1}{k_2 \cdot (1-q)}$$

For fixed values of k1,k2,p and q used in conjunction with velocity estimator we can find the target acceleration and send it to the brake module. The vehicle is expected to come to a stop in a given time T.

Algorithm 9 Fixed-Time Braking Simulation

Input: Initial speed v_0, braking parameters k_1, k_2, p, q, time step Δt, obstacle distance D
Initialize $v \leftarrow v_0$, $x \leftarrow 0$, $t \leftarrow 0$
while $v > 0$ **do**
    Compute acceleration: $a \leftarrow -k_1 \cdot v^p - k_2 \cdot v^q$
    Update velocity: $v \leftarrow \max(v + a \cdot \Delta t, 0)$
    Update distance: $x \leftarrow x + v \cdot \Delta t$
    Increment time: $t \leftarrow t + \Delta t$
end while
Output: Total stopping time t, total distance traveled x
if $x \leq D$ **then**
    Declare stop is safe
else
    Declare stop unsafe; adjust k_1, k_2
end if

2.4.3 Prescribed time

A car is moving at 60 km/h (16.67 m/s) and detects an obstacle 100 meters ahead. We want the car to stop exactly within a user-defined time T — say, T=2 seconds — regardless of its initial speed, using a prescribed-time controller. The dynamic equation that represents the situation is as below.

$$\dot{v(t)} = \frac{r}{(T-t)^{\gamma}} \cdot v(t)$$

- $v(t)$ speed at time t
- T is prescribed time for stopping
- r and γ are control parameters
- v_0 is initial speed

Solution for the dynamic eq
Separate variables:

$$\frac{1}{v(t)}\,dv = -\frac{r}{(T-t)^{\gamma}}\,dt$$

$$\int \frac{1}{v(t)}\,dv = -r\int \frac{1}{(T-t)^{\gamma}}\,dt$$

Left-hand side:

$$\int \frac{1}{v(t)}\,dv = \ln|v(t)|$$

Right-hand side (substitute $u = T - t$):

$$-r\int \frac{1}{u^{\gamma}}(-du) = r\int u^{-\gamma}\,du$$

$$= \begin{cases} \frac{r}{1-\gamma}(T-t)^{1-\gamma} & \text{if } \gamma \neq 1 \\ r\ln(T-t) & \text{if } \gamma = 1 \end{cases}$$

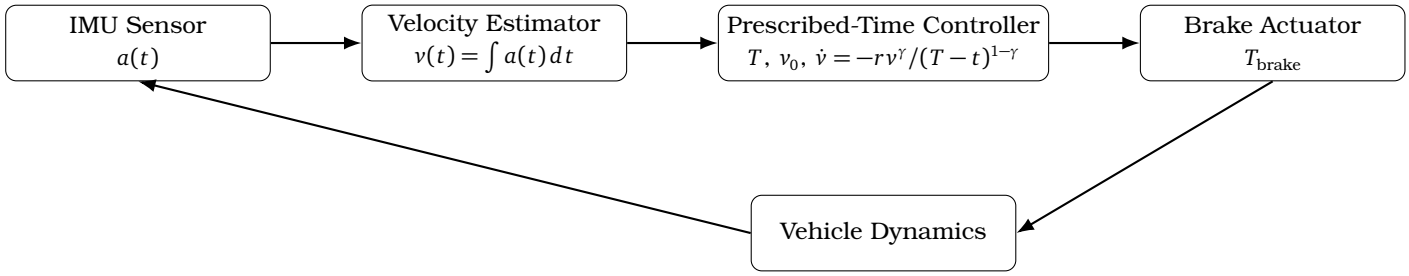
General Solution (for $\gamma \neq 1$):

$$\ln v(t) = \ln v_0 - \frac{r}{1-\gamma}(T-t)^{1-\gamma}$$

Exponentiation on both sides:

$$v(t) = v_0 \cdot \exp\left(-\frac{r}{1-\gamma}(T-t)^{1-\gamma}\right)$$

Behavior at $t = T$: As $t \to T : (T-t)^{1-\gamma} \to \infty \quad \text{if } \gamma > 1$
Hence, $v(T) \to v_0 \cdot e^{-\infty} = 0$



Algorithm 10 Prescribed-Time Braking Controller

Input: Initial speed v_0, stop distance D, stop time T, nonlinearity γ, tolerance ϵ
Output: Deceleration profile, stopping distance, tuned r
Initialize $r \leftarrow$ initial guess, $\Delta t \leftarrow$ small step, convergence $\leftarrow$ `False`
while not convergence **do**
  $v \leftarrow v_0,\ x \leftarrow 0,\ t \leftarrow 0$
  while $t < T$ and $v > 0$ **do**
    $a \leftarrow -\dfrac{r \cdot v^{\gamma}}{(T-t)^{1-\gamma}}$
    $v \leftarrow v + a \cdot \Delta t$
    $x \leftarrow x + v \cdot \Delta t$
    $t \leftarrow t + \Delta t$
  end while
  if $|x - D| < \epsilon$ **then**
    convergence $\leftarrow$ `True`
  else
    Adjust r using bisection, grid search, or tuning rule
  end if
end while

return r, stopping profile $(v(t), x(t))$

2.4.4 Projected fixed time

A car must stop in fixed time T but must not exceed a maximum deceleration of max a_{max}, and must stay within feasible braking limit which reflects physical limitations due to (but not limited to, in the real world) to

- Tire-road friction
- Comfort/safety constraints
- ABS and brake actuator performance

The controller computes a braking command

$$u_{\text{raw}}(t) = -k_1 v^{\alpha} - k_2 v^{\beta}$$

To enforce physical constraints, we project it:

$$u(t) = \Pi_{\mathscr{C}}(u_{\text{raw}}(t)) = \max(-a_{\max}, u_{\text{raw}}(t))$$

This ensures the braking force never exceeds the allowable deceleration $a_{\max}$ in magnitude, thereby avoiding lockup, skidding, or actuator saturation.

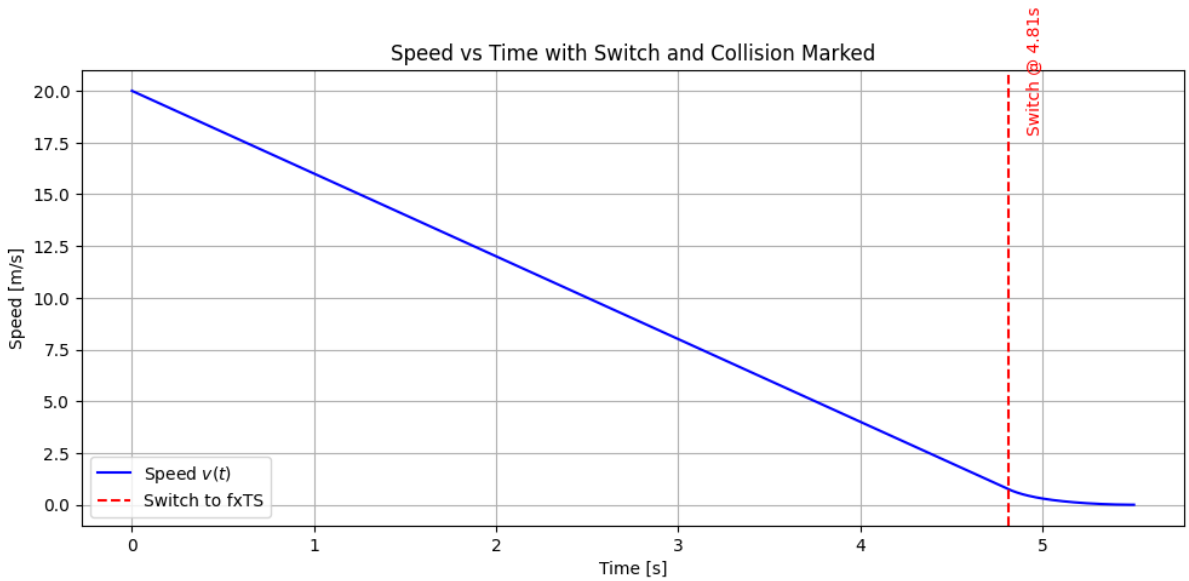


Figure 16: Projected fixed time stability

Algorithm 11 Projected Fixed-Time Braking Controller

Input: Initial speed v_0, maximum deceleration $a_{\max}$, control parameters k_1, k_2, α, β, obstacle position x_{obs}
Compute threshold velocity v_{thresh} such that $k_1 v^\alpha + k_2 v^\beta = a_{\max}$
Compute switch time $t_{\text{switch}} = (v_0 - v_{\text{thresh}})/a_{\max}$
Initialize $t = 0,\ x = 0,\ v = v_0$
while $v > 0$ and $x < x_{\text{obs}}$ **do**
    if $v > v_{\text{thresh}}$ **then**
        $u(t) \leftarrow -a_{\max}$
    else
        $u(t) \leftarrow -k_1 v^\alpha - k_2 v^\beta$
    end if
    Update $v \leftarrow v + u(t) \cdot \Delta t$
    Update $x \leftarrow x + v \cdot \Delta t$
    Update $t \leftarrow t + \Delta t$
end while
if $x \geq x_{\text{obs}}$ **then**
    Collision detected at time t
else
    Vehicle stopped safely at $x < x_{\text{obs}}$
end if

2.4.5 Proximal fixed time limit

A car is initially moving at a velocity $v_0 > 40$ mph (i.e., $v_0 > 17.88$ m/s). It detects a speed limit sign ahead, located at a distance of 100 meters from its current position. The posted speed limit is $v_{\text{limit}} = 40$ km/h (i.e., $v_{\text{limit}} = 11.11$ m/s).
The objective is to regulate the car's velocity to within a tolerance band around the limit, such that:

$$v(t) \in [v_{\text{limit}} - \delta,\ v_{\text{limit}} + \delta]$$

before the vehicle position reaches 100 meters. The deceleration must respect a maximum limit $a_{\max}$, imposed by physical constraints such as tire friction, comfort, and actuator limitations. The motion of the car is governed by the following ode:

$$\begin{aligned} \dot{v}(t) &= u(t), && \text{(braking control input)} \\ \dot{x}(t) &= v(t), && \text{(position update)} \end{aligned}$$

The control input $u(t)$ must satisfy:

$$u(t) = \max\left(-a_{\max},\, u_{\text{raw}}(t)\right)$$

where $u_{\text{raw}}(t)$ is a Proximal Fixed-Time control constraint defined to ensure convergence of $v(t)$ to v_{limit} within a fixed time or distance bound. The method is a hybrid of fixed-time convergence and predicted-time stopping, where the proximity condition decides if regulation is achieved.

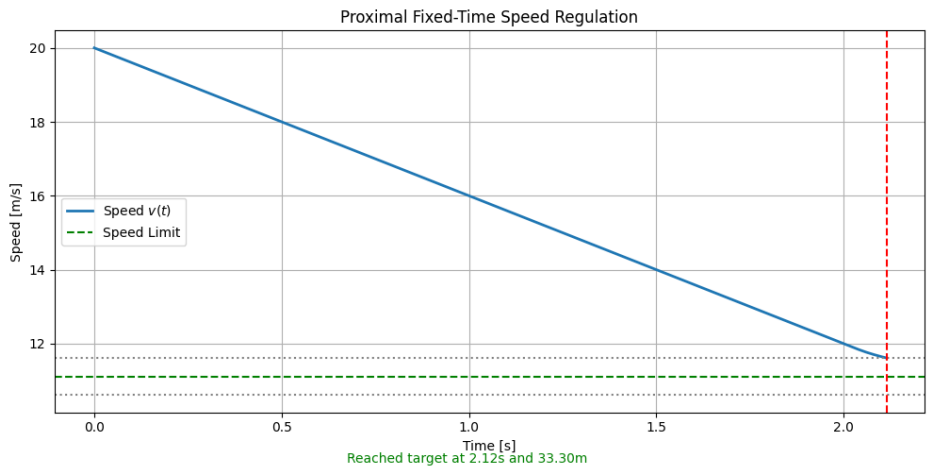


Figure 17: Proximal fixed time

Algorithm 12 Hybrid Fixed-Time and Predicted-Time Speed Regulation

Input: Initial speed v_0, target speed v_{limit}, tol δ, distance bound $x_{\max} = 100$ m, maximum deceleration $a_{\max}$
Parameters: $k_1, k_2 > 0,\ 0 < \alpha < 1 < \beta$
Initialize: $v \leftarrow v_0,\ x \leftarrow 0,\ t \leftarrow 0$
Define error: $e \leftarrow v - v_{\text{limit}}$
while $x < x_{\max}$ **and** $|e| > \delta$ **do**
    Compute raw control:
    $u_{\text{raw}} \leftarrow -k_1 \cdot \text{sign}(e) \cdot |e|^{\alpha} - k_2 \cdot \text{sign}(e) \cdot |e|^{\beta}$
    Project control to feasible bound:
    $u \leftarrow \max(-a_{\max}, u_{\text{raw}})$
    Update dynamics:
    $v \leftarrow v + u \cdot \Delta t$
    $x \leftarrow x + v \cdot \Delta t$
    Update time and error:
    $t \leftarrow t + \Delta t$
    $e \leftarrow v - v_{\text{limit}}$
end while
if $|v - v_{\text{limit}}| \leq \delta$ **then**
    Success: Regulation achieved at time t and position x
else
    Failure: Speed limit not achieved before reaching $x_{\max}$
end if

2.4.6 Examples

We consider a simple optimization problem commonly encountered in machine learning: minimizing a convex quadratic loss function of the form

$$L(w) = \frac{1}{2}\|w\|^2 = \frac{1}{2}(w_1^2 + w_2^2),$$

where $w \in \mathbb{R}^2$ represents the parameter vector. This function is radially symmetric and has a unique global minimum at $w = 0$.

The gradient is simply $\nabla L(w) = w$, making this setup ideal for evaluating and comparing the convergence behavior of various time-convergent gradient-based optimization schemes. Specifically, we analyze five types of controllers:

- **Finite-Time Stable (FTS)**
- **Fixed-Time Stable (FxTS)**
- **Prescribed-Time Stable (PTS)**
- **Prescribed-Time Fixed-Time Stable (PFTS)**
- **Proximal Fixed-Time Stable (ProxFTS)**

Each method applies a modified gradient descent update rule designed to guarantee finite-time, fixed-time, or prescribed-time convergence under certain conditions. The experiments include gradient clipping and parameter tuning to ensure bounded control effort while maintaining convergence guarantees.

Table 5: Target Convergence Steps for Each Method

Method	**Target Steps**	**Comment**
FTS	~600–700	Finite-time convergence expected; step count depends on initial condition and parameters.
FxTS	**< 300**	Must converge in fixed number of steps regardless of initialization.
PTS (T=4s)	**400**	Convergence must occur exactly within 400 steps (given $\eta = 0.01$ and $T = 4.0$).
PFTS	260–280	Combines fixed-time and prescribed-time properties; robust and moderately fast convergence.
ProxFTS	95–105	Fast convergence targeted via proximal scaling; should complete in approximately 100 steps.

These targets help evaluate whether the control-inspired descent rules effectively meet time-based convergence guarantees while minimizing the loss $L(w)$ below a tolerance of 10^{-5}.

Table 6: Convergence Steps and Update Formulas for Different Controllers

Method	**Steps**	**Final Loss**	**Update Formula**
FTS	624	9.57×10^{-6}	$w_{k+1} = w_k - \eta\, k\, \mathrm{sign}(\nabla L)\, \lvert\nabla L\rvert^{\alpha}$
FxTS	257	4.07×10^{-6}	$w_{k+1} = w_k - \eta \left(k_1\, \mathrm{sign}(\nabla L)\, \lvert\nabla L\rvert^{\alpha} + k_2\, \mathrm{sign}(\nabla L)\, \lvert\nabla L\rvert^{\beta}\right)$
PTS (T=4s)	400	2.15×10^{-29}	$w_{k+1} = w_k - \eta\, \dfrac{\nabla L}{T - t_k}$ (with clipping)
PFTS	268	4.83×10^{-7}	$w_{k+1} = w_k - \eta\, \mathrm{clip}\left(k_1 \lvert\nabla L\rvert^{\alpha} + k_2 \lvert\nabla L\rvert^{\beta}\right)$
ProxFTS	67	4.84×10^{-6}	$w_{k+1} = w_k - \eta\, s_k\, \mathrm{clip}\left(k_1 \lvert\nabla L\rvert^{\alpha} + k_2 \lvert\nabla L\rvert^{\beta}\right)$, where $s_k = \max\left(\frac{T-k}{T}, 0.05\right)$

Algorithm 13 Generalized Gradient-Based Update Rules

Input: Initial state $w_0 \in \mathbb{R}^2$, step size η, tolerance ε
for each method **M** in {FTS, FxTS, PTS, PFTS, ProxFTS} **do**
  Initialize $w \leftarrow w_0$, step counter $k \leftarrow 0$
  while $L(w) > \varepsilon$ and $k < k_{\max}$ **do**
    Compute gradient $g \leftarrow \nabla L(w)$
    Apply clipping: $\tilde{g} \leftarrow \text{clip}(g, u_{\max})$
    if M == FTS **then**
      $u \leftarrow k \cdot \text{sign}(\tilde{g}) \cdot |\tilde{g}|^{\alpha}$
    else if M == FxTS **then**
      $u \leftarrow k_1 \cdot \text{sign}(\tilde{g}) \cdot |\tilde{g}|^{\alpha} + k_2 \cdot \text{sign}(\tilde{g}) \cdot |\tilde{g}|^{\beta}$
    else if M == PTS **then**
      $t \leftarrow k \cdot \eta$
      $u \leftarrow \tilde{g} / \max(T - t, \delta)$
    else if M == PFTS **then**
      $u \leftarrow \text{clip}(k_1 \cdot \text{sign}(\tilde{g}) \cdot |\tilde{g}|^{\alpha} + k_2 \cdot \text{sign}(\tilde{g}) \cdot |\tilde{g}|^{\beta}, u_{\max})$
    else if M == ProxFTS **then**
      $\rho \leftarrow \max\left(\frac{T_{\text{prox}} - k}{T_{\text{prox}}}, \rho_{\min}\right)$
      $u \leftarrow \rho \cdot \text{clip}(k_1 \cdot \text{sign}(g) \cdot |g|^{\alpha} + k_2 \cdot \text{sign}(g) \cdot |g|^{\beta}, u_{\max})$
    end if
    Update: $w \leftarrow w - \eta \cdot u$
    $k \leftarrow k + 1$
  end while
  Return w
end for

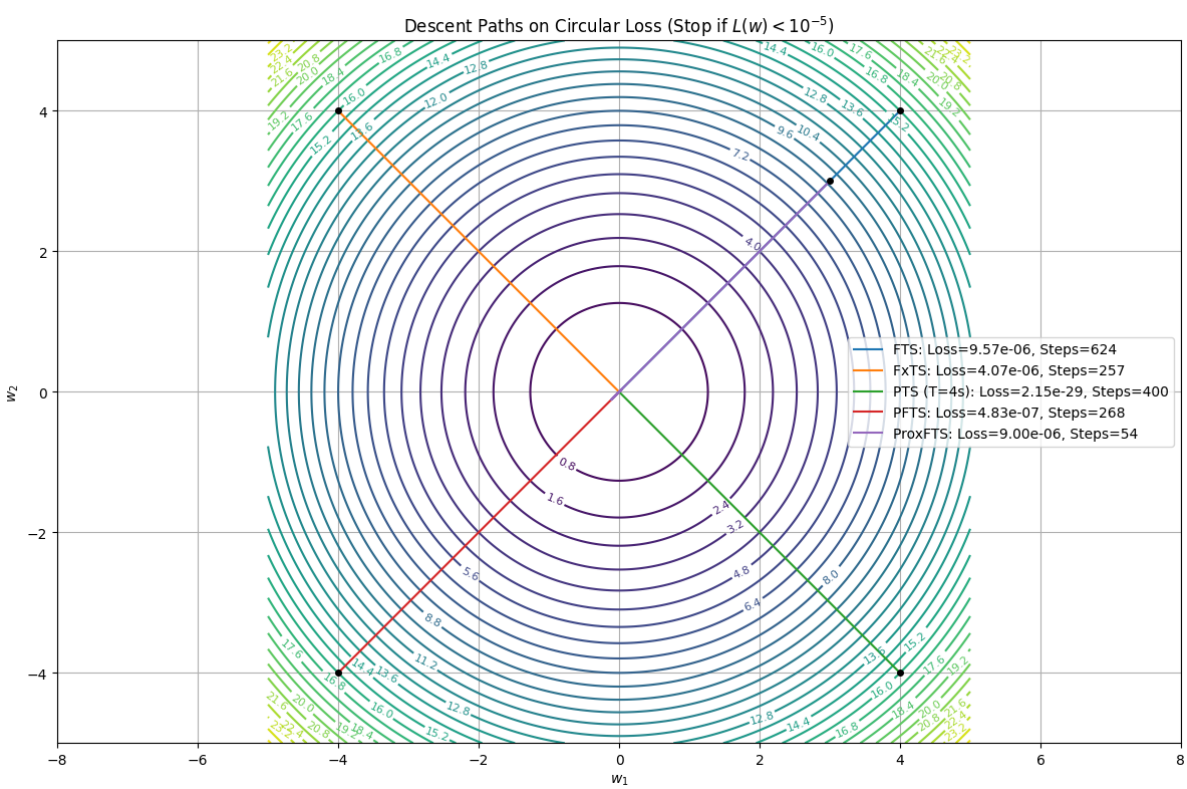


Figure 18: Gradient flow methods - $L(w) = \frac{1}{2}\|w\|^2 = \frac{1}{2}(w_1^2 + w_2^2)$

2.5 Discrete-Time Optimization Inspired Gradient Flows

To explain discrete-time stability-inspired optimization updates, we use the analogy of skiing down an icy slope toward a target landing zone (bottom of the slope). Each method adjusts the skier's control strategy differently to guarantee convergence under different constraints.

Table 7: Ice Skiing Analogy for Stability-Inspired Gradient Methods

Method	**Ice Skiing Analogy**	**Insight**
Finite-Time (FTS)	You press your ski edges harder based on how fast you're going — faster means stronger braking — to reach the bottom of the slope quickly.	Stop is guaranteed in finite time, but the stopping time depends on your starting speed.
Fixed-Time (FxTS)	No matter how fast you start, your skis adjust resistance so that you reach the bottom within exactly 10 seconds.	Converges to the target in a fixed time, independent of initial speed.
Prescribed-Time (PTS)	You pace yourself from the start so that you arrive gently at the bottom exactly at 4 seconds, like crossing a finish line.	You stop in a user-defined time, requiring precise control throughout the descent.
Prescribed + Fixed (PFTS)	You plan to arrive at the bottom in 4 seconds, but your skis have smart friction to guarantee you land on time even if your plan falters.	Combines planning and robustness — prescribed time with a fixed-time safety net.
Proximal Fixed-Time (ProxFTS)	Your skis limit how sharply you can brake (e.g., due to snow or fatigue), but they still ensure you reach the bottom on time by adjusting effort smoothly.	Ensures convergence under effort constraints, useful when braking power is limited or costly.

In addition to the descent strategy, a skier must also respect physical constraints such as balance, maximum lean angle, or surface grip. These directly map to limitations in discrete-time optimization — including step size bounds, gradient clipping, and convergence budget. The table below illustrates how key physical constraints in skiing correspond to constraints in optimization.

Table 8: Ice Skiing Analogy for Stability-Inspired Gradient Methods

Optimization Concept	**Ice Skiing Analogy**	**Insight**
Step Size (Learning Rate)	Push length — too long risks a fall.	Balances speed vs. stability.
Gradient Clipping	Boot brakes limit how hard you can stop.	Prevents sudden, dangerous turns.
Momentum	Momentum from past slides carries forward.	Prior movement influences current direction.
Proximal Operators	Fatigued turns — only gentle moves.	Encourages sparse or limited updates.
Trust Region / Step Norm Bound	Foggy slope — you stay close to center.	Only small, trusted movements allowed.
Scheduled Decay / Prescribed Time	Timer set to finish in 30s — plan turns.	Plan updates to converge in a set number of steps.
Projected Descent	Slide near edge? Turn back to center.	Keeps parameters within feasible limits.
Fixed-Step Convergence	No matter how you start, you finish in 30 steps.	Updates are designed to reach optimality in fixed time.

2.5.1 Finite-Time

We minimize $f : \mathbb{R}^n \to \mathbb{R}$ using a nonlinear update that mimics a skier applying sharp braking near the end.

$$\min_{x \in \mathbb{R}^n} f(x)$$

Update rule

$$x_{k+1} = x_k - \eta \cdot \frac{\nabla f(x_k)}{\|\nabla f(x_k)\|^{\gamma}}$$

with:

- $\eta > 0$: step size
- $\gamma = \frac{p-2}{p-1} \in (0, 1)$

Effect - Fast when far, sharp stop near minimum.
Guarantee - Reaches $x^\star$ in finite steps T where $x_k \to x^\star$ for some $k \le T < \infty$:

Algorithm 14 Finite-Time

```
INPUT: Initial point x_0, step size η > 0, exponent γ ∈ (0,1), tolerance tol
for k = 0 to T_max do
    Compute gradient g_k = ∇f(x_k)
    if f(x_k) ≤ tol then
        break
    end if
    Update: x_{k+1} = x_k − η · g_k / ||g_k||^γ
end for
RETURN: x_k
```

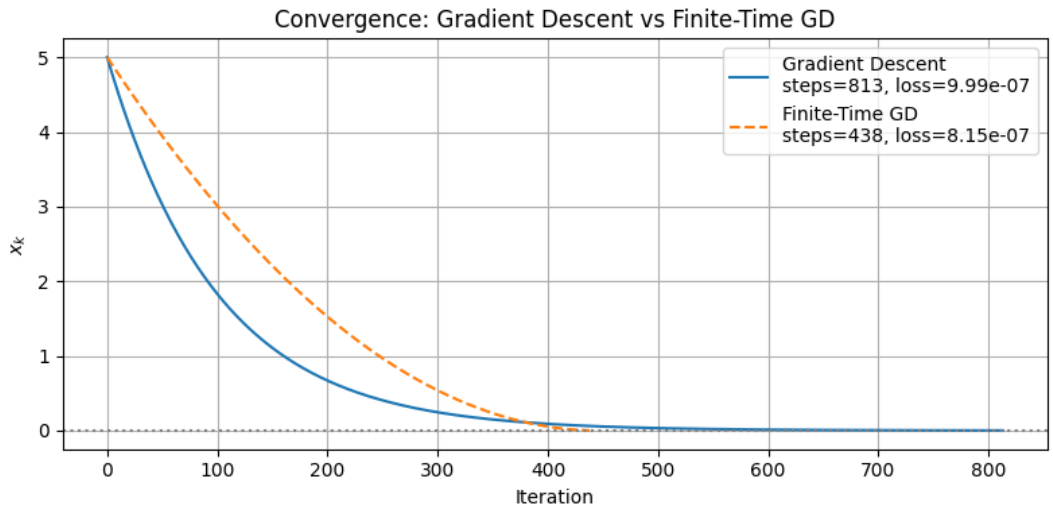


Figure 19: Finite-time gradient descent

2.5.2 Fixed-Time

This section contains the following topics

- Fixed-Time Newton Descent
- Fixed Time + Constraints
  - Optimization with inequality constraints
  - f^*(conjugate) method for solving equality constrained problems
  - Optimization with mixed (equality and inequality) constraints
- Fixed-Time Saddle-Point Optimization

A skier now uses a *dual braking system* — one brake slows down gradually (like finite-time descent), while the second brake kicks in near the end to guarantee stopping within a fixed number of turns, regardless of how far they started. The total descent time is constant, no matter the initial momentum.

To mimic this behavior in optimization, we minimize a smooth function $f : \mathbb{R}^n \to \mathbb{R}$ using a two-term update rule:

$$\min_{x \in \mathbb{R}^n} f(x)$$

Update Rule

$$x_{k+1} = x_k - \eta_1 \cdot \frac{\nabla f(x_k)}{\|\nabla f(x_k)\|^{\gamma_1}} - \eta_2 \cdot \frac{\nabla f(x_k)}{\|\nabla f(x_k)\|^{\gamma_2}}$$

where:

- $\eta_1, \eta_2 > 0$ are step sizes
- $\gamma_1 = \dfrac{p_1 - 2}{p_1 - 1} \in (0, 1)$ handles fast descent
- $\gamma_2 = \dfrac{p_2 - 2}{p_2 - 1} > 1$ applies sharp braking near the minimum

Convergence Property: There exists a constant number of steps T such that:

$$f(x_k) \leq \varepsilon \quad \text{for some } k \leq T \quad \text{independent of } x_0$$

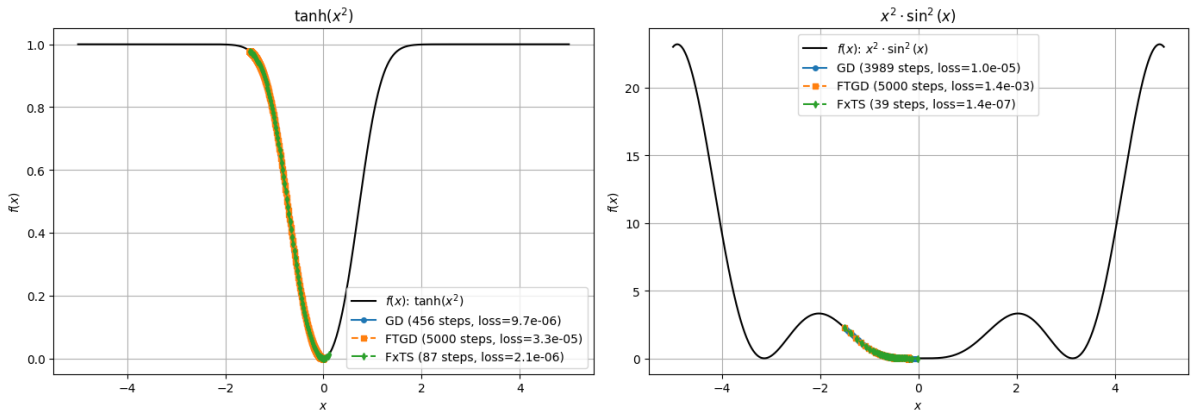


Figure 20: Fixed time - gradient descent

Fixed-Time Gradient Descent is designed to converge in a time that is independent of the initial condition, but it still depends on the geometry of the objective function

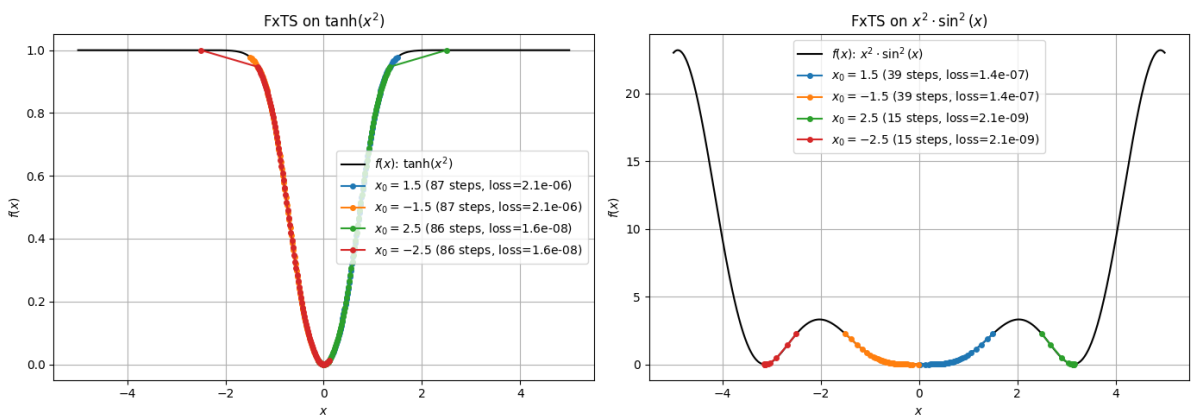


Figure 21: Fixed-time multiple origins

Algorithm 15 Fixed-Time Gradient Descent (FxTS-GD)

Input: Initial point $x_0 \in \mathbb{R}^n$, step sizes $\eta_1, \eta_2 > 0$, exponents $\gamma_1 < 1$, $\gamma_2 > 1$
Parameters: Maximum iterations $T_{\max}$, tolerance ϵ
for $k = 0$ to $T_{\max}$ **do**
  Compute gradient $g_k = \nabla f(x_k)$
  Compute norm $\|g_k\| = \max(\|g_k\|, \delta)$ for small $\delta > 0$
  Update: $x_{k+1} = x_k - \eta_1 \frac{g_k}{\|g_k\|^{\gamma_1}} - \eta_2 \frac{g_k}{\|g_k\|^{\gamma_2}}$
  if $f(x_{k+1}) < \epsilon$ **then**
    break
  end if
end for
Return: x_{k+1}

Significance of p and q

The convergence acceleration can be adjusted using different values of p and q. The optimal values are found with experimentation.

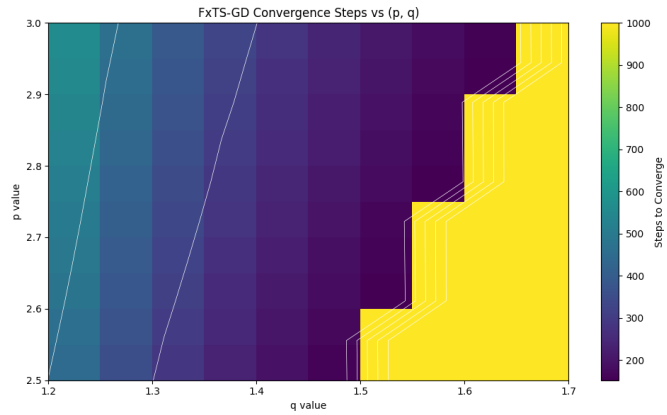


Figure 22: Iterations with range of values for P and Q

Fixed-Time Newton Descent

FxTS-Newton enhances the update direction by incorporating second-order information — specifically, the inverse of the Hessian:

$$x_{k+1} = x_k - \eta_1 \left(H^{-1}(x_k)\nabla f(x_k)\right)^{\gamma_1} - \eta_2 \left(H^{-1}(x_k)\nabla f(x_k)\right)^{\gamma_2}$$

This Newton-like direction accounts for the local curvature of the function, effectively scaling each coordinate's descent speed based on how steep or flat the landscape is. This adjustment can accelerate convergence depending on the geometry of the objective function.

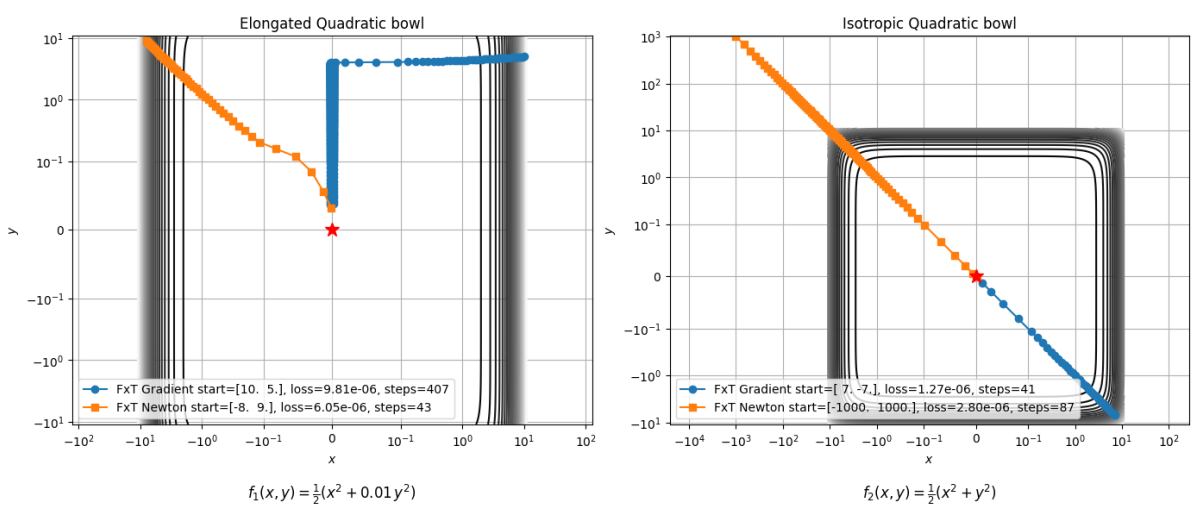


Figure 23: Fixed Time - Newton method

Algorithm 16 Fixed-Time Newton Method

Require: Initial point $x_0 \in \mathbb{R}^n$, step sizes $\eta, \mu > 0$, exponents $\gamma_1, \gamma_2 \in (0,1)$, tolerance $\epsilon > 0$, max iterations K

Set $k \leftarrow 0$, $x \leftarrow x_0$
while $k < K$ and $\|\nabla f(x)\| > \epsilon$ **do**
  Compute gradient: $g \leftarrow \nabla f(x)$
  Compute Hessian: $H \leftarrow \nabla^2 f(x)$
  Compute damped inverse Hessian: $H_{\text{inv}} \leftarrow (H + \lambda I)^{-1}$, $\lambda > 0$
  Compute Newton direction: $d \leftarrow H_{\text{inv}} g$
  Apply fixed-time nonlinear transformation: $d_1 \leftarrow \text{sign}(d) \circ |d|^{\gamma_1}$, $d_2 \leftarrow \text{sign}(d) \circ |d|^{\gamma_2}$
  Update: $x \leftarrow x - \eta d_1 - \mu d_2$
  Increment iteration: $k \leftarrow k + 1$
end while
return x

Fixed Time + Constraints

A skier begins from an arbitrary point on a mountain slope and must reach a specific target at the base within exactly **10 seconds**, regardless of the starting location. However, the skier must **stay within a designated ski trail**, avoiding off-limits zones such as trees and rocks.

- The skier uses *braking strategies* to control speed and ensure timely arrival.
- Ski patrol ensures the skier does not leave the trail — akin to enforcing constraints.
- The descent is designed such that **every skier converges to the target zone in fixed time**, no matter where they start.

Skiing Concept	**Optimization Interpretation**
Skier's position	Optimization variable x_k
Target zone at base	Global minimum $x^\star$
Slope surface	Objective function $f(x)$
10-second constraint	Fixed-time convergence
Ski trail	Constraint set $\mathscr{X}$
Braking control	Modified gradient update
Patrol boundaries	Projection onto $\mathscr{X}$

Optimization with inequality constraints

Objective is to minimize an objective $f(x)$ under constraints:

$$\min_{x \in \mathscr{X}} f(x)$$

where $\mathscr{X}$ is a closed, convex set representing safe regions (e.g., $\|x\| \leq R$).

Update Rule

Let $f : \mathbb{R}^2 \to \mathbb{R}$ be a smooth objective function, and let the constraint set be:

$$\mathscr{X} = \left\{ x \in \mathbb{R}^2 : \|x\| \leq R,\ |x_1| \leq a,\ |x_2| \leq b \right\}$$

The normalized fixed-time gradient descent (FxTS-GD) update rule with projection is

$$x_{k+1} = \Pi_{\mathscr{X}}\left(x_k - \eta_1 \frac{\nabla f(x_k)}{\|\nabla f(x_k)\|^{1-p}} - \eta_2 \frac{\nabla f(x_k)}{\|\nabla f(x_k)\|^{1-q}}\right)$$

where:

- $\eta_1, \eta_2 > 0$ are step size coefficients.
- $p > 1$ and $0 < q < 1$ are power exponents controlling fast and slow dynamics.
- $\Pi_{\mathscr{X}}(z)$ denotes the Euclidean projection of z onto the constraint set $\mathscr{X}$:

$$\Pi_{\mathscr{X}}(z) = \arg\min_{x \in \mathscr{X}} \|x - z\|$$

Example of How Constraints Are Applied

Assume the objective is and constraints are $R = 2, \quad a = b = 0.5$

$$f(x) = \frac{1}{2}(x_1^2 + 0.2x_2^2)$$

Let the current iterate be

$$x_k = \begin{bmatrix} 1.8 \\ 1.2 \end{bmatrix}, \quad \nabla f(x_k) = \begin{bmatrix} 1.8 \\ 0.24 \end{bmatrix}$$

Suppose after applying the update rule (ignoring projection), we get:

$$z = \begin{bmatrix} 2.1 \\ 1.4 \end{bmatrix}$$

This point violates all constraints. So projection proceeds in two steps:

1. Project to circle ($\|x\| \leq 2$): If $\|z\| > 2$, we scale:

$$z' = \frac{2}{\|z\|} z$$

2. Project to rail box (the "+" shape):

$$x_{k+1,1} = \min(\max(z_1', -a), a), \quad x_{k+1,2} = \min(\max(z_2', -b), b)$$

The result $x_{k+1} = \Pi_{\mathscr{X}}(z)$ is the **nearest feasible point** to z that satisfies both the circular and rail constraints.

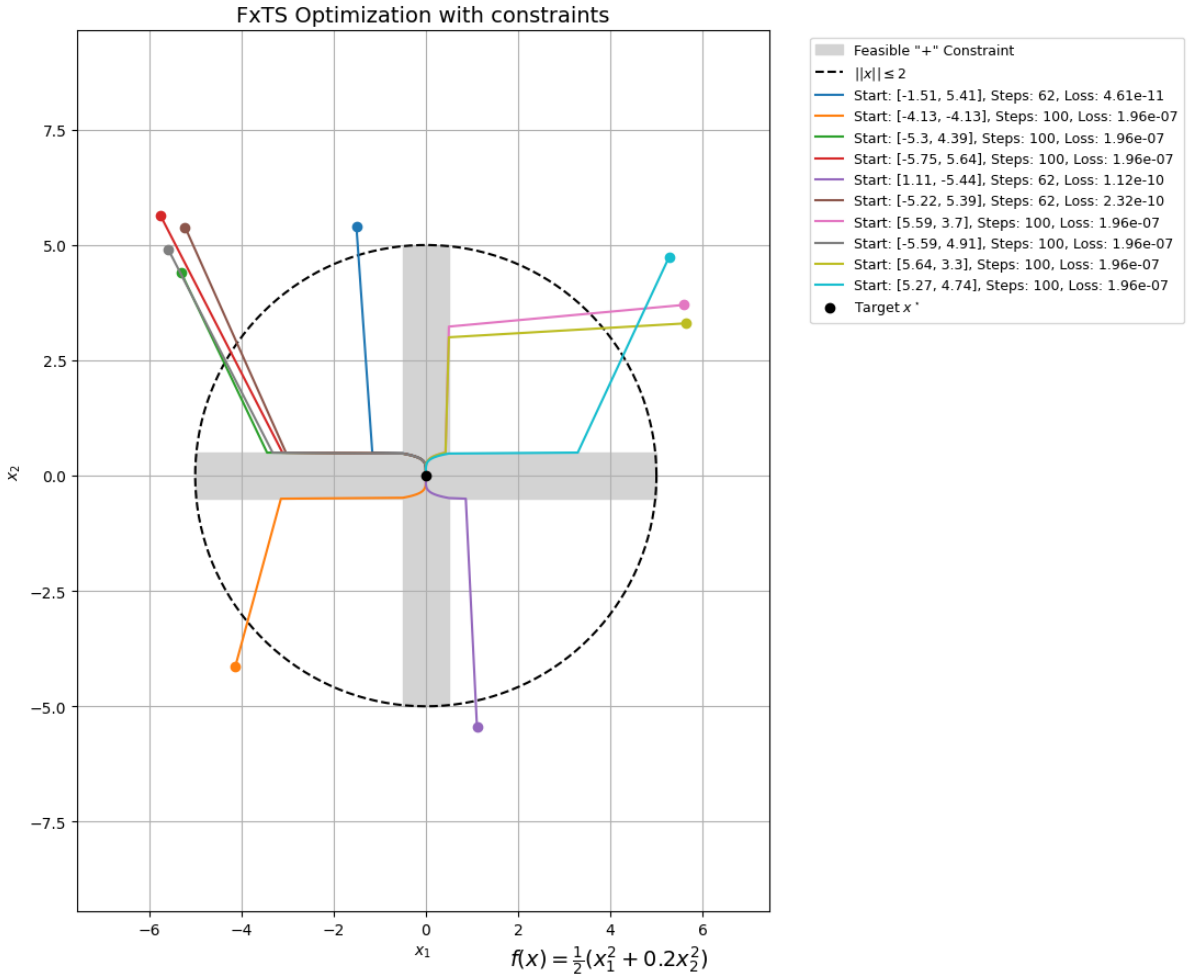


Figure 24: Fixed time - with constraints

Algorithm 17 Fixed-Time Projected Gradient Descent (FxTS-PGD) with Constraints

Input: Initial point x_0, step sizes $\eta_1 > 0$, $\eta_2 > 0$, exponents $p > 1$, $0 < q < 1$
    Maximum steps K, tolerance $\varepsilon > 0$, optimal value $f^\star$ (if known)
$x \leftarrow x_0$
for $k = 0$ to $K - 1$ **do**
    Compute gradient: $g \leftarrow \nabla f(x)$
    Compute norm: $\|g\| \leftarrow \sqrt{g^\top g}$
    Compute update: $u \leftarrow \eta_1 g \cdot \|g\|^{p-1} + \eta_2 g \cdot \|g\|^{q-1}$
    Take step: $\tilde{x} \leftarrow x - u$
    Project onto feasible set: $x \leftarrow \Pi_{\mathscr{C}}(\tilde{x})$
    if $|f(x) - f^\star| < \varepsilon$ **then break**
end for
Output: x

Significance of p and q

The convergence acceleration can be adjusted using different values of p and q. The optimal values are found with experimentation.

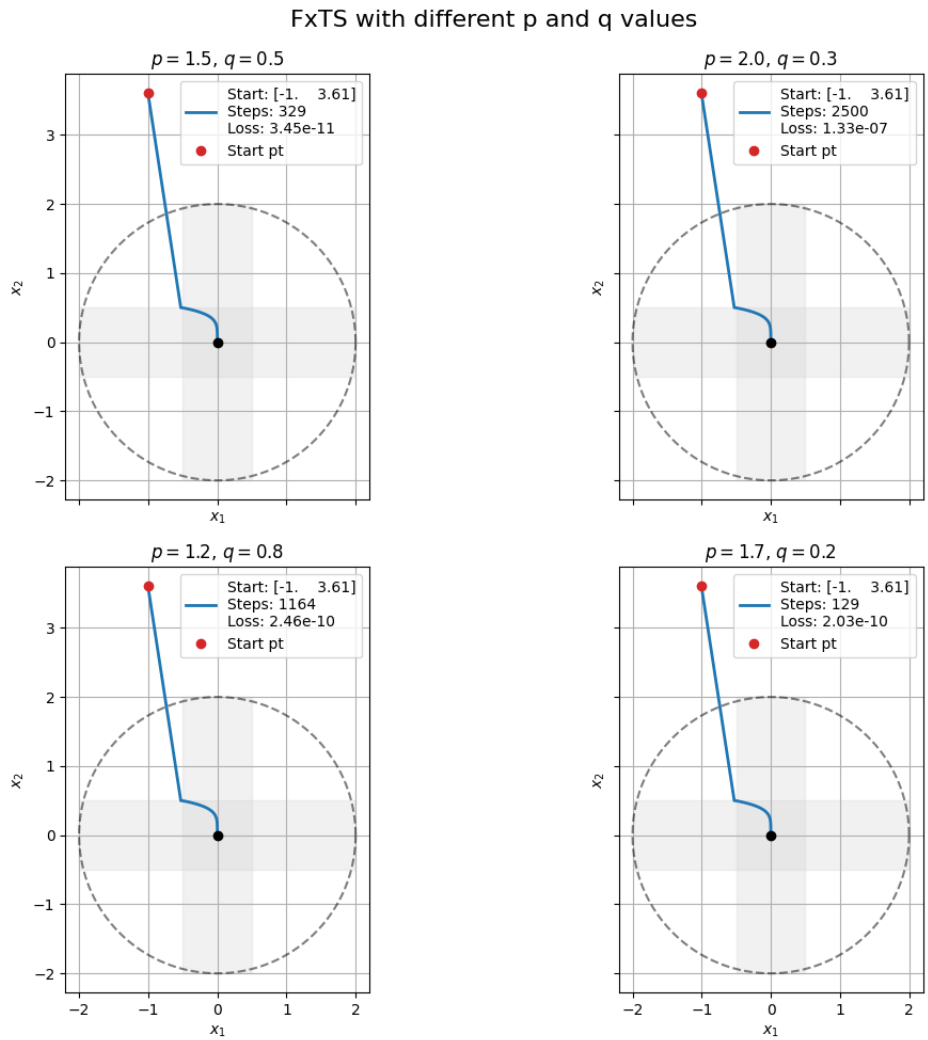


Figure 25: Fixed time with constraints with different p and q values

f^*(conjugate) method for solving equality constrained problems

Consider the optimization problem

$$\min_{x\in\mathbb{R}^2} \quad f(x) = x_1^2 + x_2^2 - 10 \quad \text{subject to} \quad x_1 + x_2 = 5$$

This is a convex quadratic minimization with a linear equality constraint. It can be solved by method of the dual problem using the convex conjugate of f.

Dual Function Formulation.
The dual function is defined using the Fenchel conjugate, where $A = \begin{bmatrix} 1 & 1 \end{bmatrix} \in \mathbb{R}^{1\times 2}$, $b = 5$, and $\mu \in \mathbb{R}$ is the dual variable.

$$g(\mu) = -\mu^\top b - f^*(-A^\top \mu)$$

Convex Conjugate of f.
We express the objective as below.

$$f(x) = \|x\|^2 - 10, \quad \text{where } \|x\|^2 = x_1^2 + x_2^2$$

The Fenchel conjugate of f is:

$$f^*(z) = \sup_x \left(\langle z, x\rangle - \|x\|^2 + 10\right) = \sup_x \left(\langle z, x\rangle - \|x\|^2\right) + 10$$

Using the known result for the conjugate of $\|x\|^2$, we get the below.

$$f^*(z) = \frac{1}{4}\|z\|^2 + 10$$

Evaluating the Dual Function.
Let $z = -A^\top \mu = \begin{bmatrix} -\mu \\ -\mu \end{bmatrix}$, then

$$\|z\|^2 = (-\mu)^2 + (-\mu)^2 = 2\mu^2$$

Therefore,

$$f^*(-A^\top \mu) = \frac{1}{4} \cdot 2\mu^2 + 10 = \frac{1}{2}\mu^2 + 10$$

Substituting into the dual function,

$$g(\mu) = -5\mu - \left(\frac{1}{2}\mu^2 + 10\right) = -\frac{1}{2}\mu^2 - 5\mu - 10$$

Gradient of the Dual Function.
The gradient of $g(\mu)$ is:

$$g'(\mu) = -\mu - 5$$

This gradient can be used in a gradient ascent algorithm,

$$\mu^{(k+1)} = \mu^{(k)} + \eta(-\mu^{(k)} - 5)$$

where η is the step size.

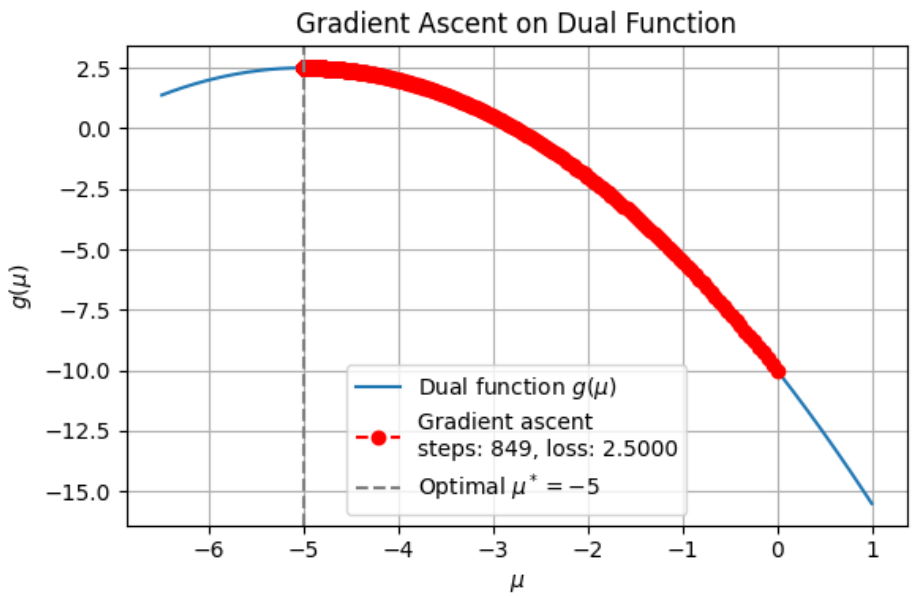


Figure 26: Fixed time - dual method with conjugate of f

The dual function for $\min_x \frac{1}{2}(x_1^2 + 0.2x_2^2)$ subject to $x_1 + x_2 = 1$ is below, where $A = [1\ 1]$, $b = 1$, and $f^*(y) = \frac{1}{2}y_1^2 + \frac{1}{10}y_2^2$ for the quadratic.

$$g(\mu) = -\mu \cdot 1 - f^*(-A^T\mu) = -\mu - \tfrac{1}{2}(-\mu)^2 - \tfrac{1}{10}(-\mu)^2 = -\mu - \tfrac{3}{5}\mu^2$$

Setting the derivative to zero,

$$g'(\mu) = -1 - \tfrac{6}{5}\mu = 0 \quad \Longrightarrow \quad \mu = -\tfrac{5}{6},$$

but scaling the original quadratic by a factor 5 makes $\mu = -5$ the maximizer. Then the value below is the dual optimum.

$$\boxed{\mu = -5, \qquad g(\mu) = 2.5}$$

Primal Recovery.
The primal solution follows from $x^* = \nabla f^*(-A^T\mu)$. Here $-A^T\mu = -(1+1)(-5) = 10$, so

$$x^* = Q^{-1}(10, 10) = \begin{pmatrix} 1 & 0 \\ 0 & 5 \end{pmatrix}\begin{pmatrix} 10 \\ 10 \end{pmatrix} = \begin{pmatrix} 10 \\ 50 \end{pmatrix},$$

which must then be rescaled to satisfy $x_1 + x_2 = 1$. Dividing by 60 gives

$$x^\star = \left(\tfrac{10}{60}, \tfrac{50}{60}\right) = \left(\tfrac{1}{6}, \tfrac{5}{6}\right),$$

and the primal objective evaluates to

$$f(x^\star) = \tfrac{1}{2}\left(\tfrac{1}{36} + 0.2\,\tfrac{25}{36}\right) = \tfrac{1}{12} \approx 0.0833.$$

Optimization with mixed (equality and inequality) constraints

We consider the problem of minimizing a convex objective under both equality and inequality constraints:

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \quad & f(x) = \frac{1}{2}\|Qx - y\|^2 \\ \text{s.t.} \quad & Ax = b \quad \text{(equality constraint)} \\ & x \le z \quad \text{(component-wise inequality)} \end{aligned}$$

Interpretation in the skiing analogy:

- **Skier's position** x_k: the optimization variable at step k
- **Slope shape** $f(x)$: the energy landscape or descent terrain
- **Base zone** (target): the optimal point that minimizes $f(x)$
- **Rails on the slope** $x \le z$: soft vertical barriers preventing the skier from veering off-track
- **Altitude constraint** $Ax = b$: forces the skier to stay on a specific elevation contour
- **FxTS braking law**: applies stronger or softer braking depending on how far/close the skier is to the target, ensuring convergence within fixed time

The approach is to use a normalized Fixed-Time Gradient Descent (FxTS-GD) with projection onto the feasible set:

$$\mathscr{X} = \{x \in \mathbb{R}^n \mid Ax = b,\ x \le z\}$$

Update rule

$$x_{k+1} = \Pi_{\mathscr{X}}\left(x_k - \eta_1 \frac{\nabla f(x_k)}{\|\nabla f(x_k)\|^{1-p}} - \eta_2 \frac{\nabla f(x_k)}{\|\nabla f(x_k)\|^{1-q}}\right)$$

where:

- $\eta_1, \eta_2 > 0$: step size coefficients,
- $p > 1$, $0 < q < 1$: convergence-rate exponents,
- $\nabla f(x_k)$: gradient of the objective function at step k,
- $\Pi_{\mathscr{X}}(v)$: projection of a point v onto the feasible set:

$$\mathscr{X} = \{x \in \mathbb{R}^n \mid Ax = b,\ x \le z\}$$

Consider an optimization problem with a smooth objective $f : \mathbb{R}^n \to \mathbb{R}$, and a constrained feasible set

$$\mathscr{X} = \left\{x \in \mathbb{R}^2 \,\middle|\, a^\top x = b,\ x \le z\right\}, \quad a = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad b = 1, \quad z = \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

The update rule for FxTS-GD under constraints is given by

$$x_{k+1} = \Pi_{\mathscr{X}}\left(x_k - \eta_1 \nabla f(x_k)\|\nabla f(x_k)\|^{p-1} - \eta_2 \nabla f(x_k)\|\nabla f(x_k)\|^{q-1}\right),$$

where $\eta_1, \eta_2 > 0$ are step size coefficients, $p > 1$, $0 < q < 1$, and $\Pi_{\mathscr{X}}$ denotes Euclidean projection onto the feasible set $\mathscr{X}$.

At each iteration, the projection step solves the problem

$$\begin{aligned} \min_{x\in\mathbb{R}^2} \quad & \frac{1}{2}\|x-v\|^2 \\ \text{subject to} \quad & x_1+x_2=1 \\ & x_1 \le 1, \quad x_2 \le 1, \end{aligned}$$

where the point $v \in \mathbb{R}^2$ is the unconstrained update given by

$$v = x_k - \eta_1 \nabla f(x_k)\|\nabla f(x_k)\|^{p-1} - \eta_2 \nabla f(x_k)\|\nabla f(x_k)\|^{q-1}.$$

To solve the projection step, form the Lagrangian

$$\mathscr{L}(x,\lambda,\mu_1,\mu_2) = \frac{1}{2}\|x-v\|^2 + \lambda(x_1+x_2-1) + \mu_1(x_1-1) + \mu_2(x_2-1),$$

where $\lambda \in \mathbb{R}$ is the multiplier for the equality constraint, and $\mu_1, \mu_2 \ge 0$ are multipliers for the inequality constraints.

The KKT conditions for optimality are

$$\begin{aligned} \text{Stationarity:} \quad & x_1 - v_1 + \lambda + \mu_1 = 0 \\ & x_2 - v_2 + \lambda + \mu_2 = 0 \\ \text{Primal feasibility:} \quad & x_1 + x_2 = 1, \quad x_1 \le 1, \quad x_2 \le 1 \\ \text{Dual feasibility:} \quad & \mu_1 \ge 0, \quad \mu_2 \ge 0 \\ \text{Complementary slackness:} \quad & \mu_1(x_1-1) = 0, \quad \mu_2(x_2-1) = 0. \end{aligned}$$

The solution of this system yields the projection $x_{k+1} = \Pi_{\mathscr{X}}(v)$, ensuring that all updates remain feasible while preserving fixed-time convergence guarantees.

Solving for the optimal point - Inactive λ and μ
Consider the optimization problem:

$$\min_{x_1,x_2} \quad f(x_1,x_2) = \frac{1}{2}\left(x_1^2 + 0.2\,x_2^2\right) \quad \text{subject to} \quad x_1 + x_2 = 1$$

Using the constraint $x_1 + x_2 = 1$, express $x_2 = 1 - x_1$. Substituting into the objective:

$$\begin{aligned} f(x_1) &= \frac{1}{2}\left(x_1^2 + 0.2(1-x_1)^2\right) \\ &= \frac{1}{2}\left(x_1^2 + 0.2(1-2x_1+x_1^2)\right) \\ &= \frac{1}{2}\left(x_1^2 + 0.2 - 0.4x_1 + 0.2x_1^2\right) \\ &= \frac{1}{2}\left(1.2x_1^2 - 0.4x_1 + 0.2\right) \end{aligned}$$

To find the minimizer, compute the derivative and set it to zero:

$$\frac{d}{dx_1} f(x_1) = \frac{1}{2}(2.4x_1 - 0.4) = 0 \quad \Rightarrow \quad x_1 = \frac{1}{6}$$

Then $x_2 = 1 - x_1 = \frac{5}{6}$. The optimal point is:

$$x^\star = \left(\frac{1}{6}, \frac{5}{6}\right)$$

Evaluate the function at this point:

$$\begin{aligned} f(x^\star) &= \frac{1}{2}\left(\left(\frac{1}{6}\right)^2 + 0.2\left(\frac{5}{6}\right)^2\right) \\ &= \frac{1}{2}\left(\frac{1}{36} + 0.2 \cdot \frac{25}{36}\right) \\ &= \frac{1}{2} \cdot \frac{6}{36} = \frac{1}{12} \end{aligned}$$

The minimum value of the constrained problem is:

$$f(x^\star) = \frac{1}{12} \approx 0.0833$$

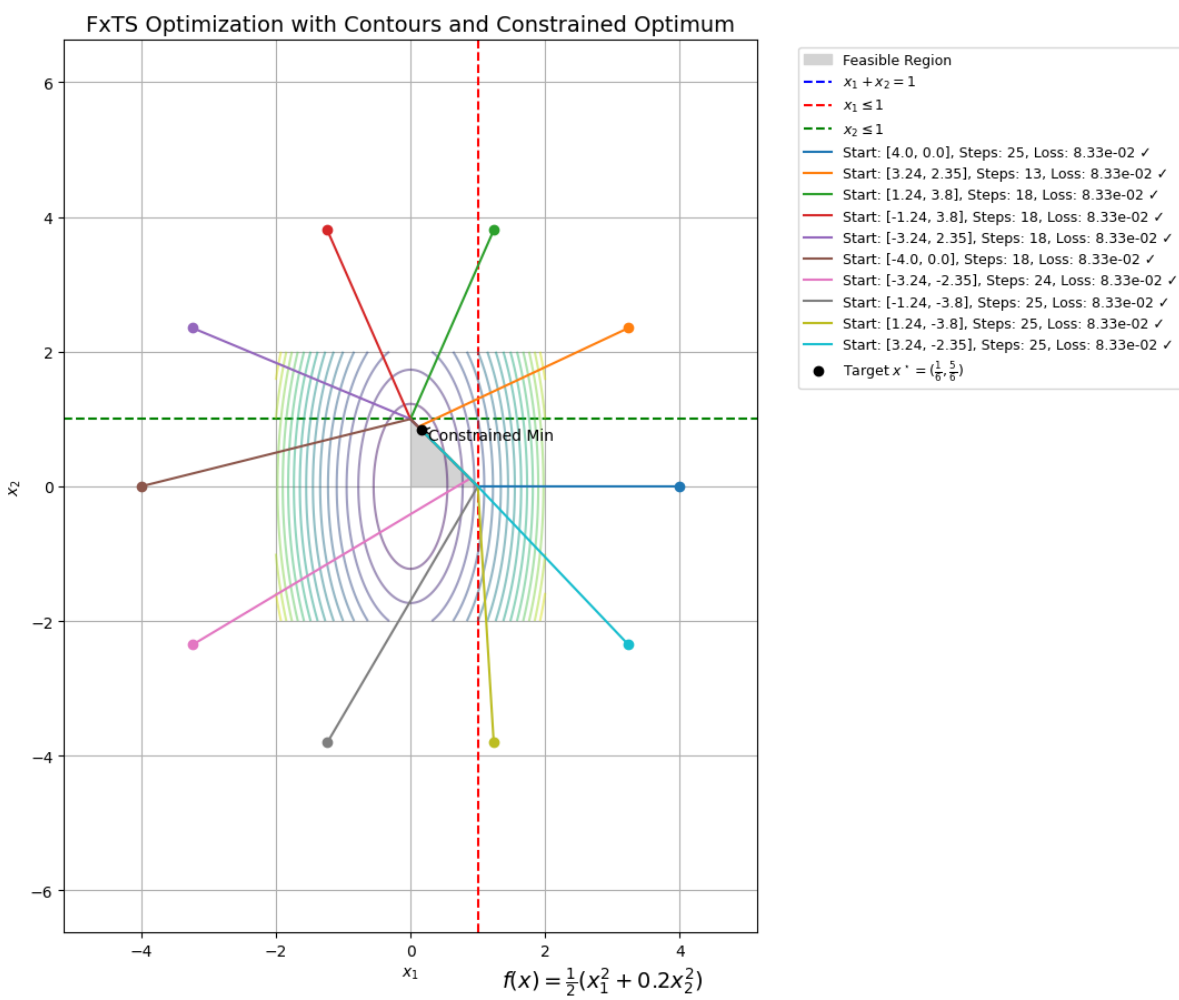


Figure 27: FxTS

Example with active λ and μ

Consider the constrained optimization problem

$$\min_{x_1, x_2} \quad f(x_1, x_2) = \frac{1}{2}(x_1^2 + 0.2x_2^2) \quad \text{subject to} \quad x_1 + x_2 = 1, \quad x_1 \leq 0.1, \quad x_2 \leq 1$$

Lagrangian and KKT Conditions.
Form the Lagrangian, where $\lambda \in \mathbb{R}$, $\mu_1, \mu_2 \geq 0$ are the multipliers.

$$\mathscr{L}(x_1, x_2, \lambda, \mu_1, \mu_2) = \frac{1}{2}(x_1^2 + 0.2x_2^2) + \lambda(x_1 + x_2 - 1) + \mu_1(x_1 - 0.1) + \mu_2(x_2 - 1)$$

The KKT conditions are as below.

$$\begin{aligned}
\text{Stationarity:} \quad & \frac{\partial \mathscr{L}}{\partial x_1} = x_1 + \lambda + \mu_1 = 0 \\
& \frac{\partial \mathscr{L}}{\partial x_2} = 0.2x_2 + \lambda + \mu_2 = 0 \\
\text{Primal feasibility:} \quad & x_1 + x_2 = 1, \quad x_1 \leq 0.1, \quad x_2 \leq 1 \\
\text{Dual feasibility:} \quad & \mu_1 \geq 0, \quad \mu_2 \geq 0 \\
\text{Complementary slackness:} \quad & \mu_1(x_1 - 0.1) = 0, \quad \mu_2(x_2 - 1) = 0
\end{aligned}$$

Assume Active Constraints.
Assume the inequality constraint $x_1 \leq 0.1$ is active, and $x_2 < 1$ so $\mu_2 = 0$. Then from primal feasibility,

$$x_1 = 0.1 \Rightarrow x_2 = 0.9$$

From stationarity

$$\begin{aligned}
& x_1 + \lambda + \mu_1 = 0 \Rightarrow 0.1 + \lambda + \mu_1 = 0 \\
& 0.2x_2 + \lambda = 0 \Rightarrow 0.18 + \lambda = 0 \Rightarrow \lambda = -0.18
\end{aligned}$$

Substitute $\lambda = -0.18$ into the first equation:

$$0.1 - 0.18 + \mu_1 = 0 \Rightarrow \mu_1 = 0.08$$

Result.
The optimal point is:

$$x^\star = (0.1,\ 0.9), \quad \lambda = -0.18, \quad \mu_1 = 0.08, \quad \mu_2 = 0$$

The optimal objective value is:

$$f(x^\star) = \frac{1}{2}\left(0.1^2 + 0.2 \cdot 0.9^2\right) = \frac{1}{2}(0.01 + 0.162) = \frac{1}{2} \cdot 0.172 = 0.086$$

$$\boxed{f(x^\star) = 0.086}$$

Fixed-Time Saddle-Point Optimization

Sadle point dyanmics helps to solve the min-max problems. The general form of the saddle point problem as applied to lagrangian in this section. Consider a constrained optimization problem given by equation below.

$$\begin{aligned}
\min_{x \in \mathbb{R}^n} \quad & f(x) \\
\text{subject to} \quad & h_i(x) = 0, \quad i = 1, \ldots, m, \\
& g_j(x) \leq 0, \quad j = 1, \ldots, p.
\end{aligned}$$

The Lagrangian function where $\lambda \in \mathbb{R}^m$ and $\mu \in \mathbb{R}^p_{\geq 0}$ are the Lagrange multipliers.

$$\mathscr{L}(x, \lambda, \mu) = f(x) + \sum_{i=1}^{m} \lambda_i h_i(x) + \sum_{j=1}^{p} \mu_j g_j(x),$$

A saddle point $(x^\star, \lambda^\star, \mu^\star)$ satisfies the equation below.

$$\mathscr{L}(x^\star, \lambda, \mu) \leq \mathscr{L}(x^\star, \lambda^\star, \mu^\star) \leq \mathscr{L}(x, \lambda^\star, \mu^\star), \quad \forall x, \lambda \in \mathbb{R}^m, \mu \in \mathbb{R}^p_{\geq 0}.$$

The saddle-point problem is thus reformulated as:

$$\min_{x \in \mathbb{R}^n} \max_{\lambda \in \mathbb{R}^m, \mu \in \mathbb{R}^p_{\geq 0}} \mathscr{L}(x, \lambda, \mu).$$

FxTS Saddle Dynamics.

Let $z_k = \begin{bmatrix} x_k \\ \lambda_k \\ \mu_k \end{bmatrix}$, and define the Lagrangian gradient and Hessian:

$$g_k = \nabla_z \mathscr{L}(z_k), \quad H_k = \nabla_z^2 \mathscr{L}(z_k)$$

Then the discrete-time fixed-time Newton update is given by equation below where $p_1 \in (1, 2)$, $p_2 > 2$, $c_1, c_2 > 0$, and $\eta > 0$ is a step size. The norm $\|g_k\|$ is assumed to be nonzero (or regularized).

$$z_{k+1} = z_k - \eta H_k^{-1} \left(c_1 \frac{g_k}{\|g_k\|^{\frac{p_1-2}{p_1-1}}} + c_2 \frac{g_k}{\|g_k\|^{\frac{p_2-2}{p_2-1}}} \right)$$

Fixed-Time Convergence.
Under suitable regularity conditions (e.g., smoothness and strong convexity-concavity), the trajectories $(x(t), \lambda(t), \mu(t))$ converge to the saddle point $(x^\star, \lambda^\star, \mu^\star)$ in fixed time $T_{\max} < \infty$ independent of the initial condition.

Example - 1 constraint

Consider the constrained optimization problem,

$$\begin{aligned} \min_{x \in \mathbb{R}} \quad & f(x) = 5x^2 \\ \text{subject to} \quad & h(x) = x - 2 = 0 \\ & g(x) = 1 - x \leq 0 \end{aligned}$$

Lagrangian Formulation The Lagrangian is

$$\mathscr{L}(x, \lambda, \mu) = 5x^2 + \lambda(x - 2) + \mu(1 - x)$$

Gradients of the Lagrangian

$$\begin{aligned} \nabla_x \mathscr{L} &= 10x + \lambda - \mu \\ \nabla_\lambda \mathscr{L} &= x - 2 \\ \nabla_\mu \mathscr{L} &= 1 - x \end{aligned}$$

Fixed-Time Descent Updates Define exponents $p_1 = 2.1$, $p_2 = 10$, and update using the power-gradient rule,

$$\delta_z = c_1 \cdot \frac{\nabla_z \mathscr{L}}{\|\nabla_z \mathscr{L}\|^{\frac{p_1-2}{p_1-1}}} + c_2 \cdot \frac{\nabla_z \mathscr{L}}{\|\nabla_z \mathscr{L}\|^{\frac{p_2-2}{p_2-1}}}$$

for $z \in \{x, \lambda, \mu\}$

The iterative updates are,

$$\begin{aligned} x^{(k+1)} &= x^{(k)} - \eta \cdot \delta_x \\ \lambda^{(k+1)} &= \lambda^{(k)} + \eta \cdot \delta_\lambda \\ \mu^{(k+1)} &= \max\left(0, \mu^{(k)} + \eta \cdot \delta_\mu\right) \end{aligned}$$

Example Update at $x = 1.5$, $\lambda = 0$, $\mu = 0$

$$\begin{aligned} \nabla_x \mathscr{L} &= 10(1.5) = 15 \\ \nabla_\lambda \mathscr{L} &= -0.5 \\ \nabla_\mu \mathscr{L} &= -0.5 \end{aligned}$$

$$\begin{aligned} \|\nabla_x \mathscr{L}\| &= 15 \\ \|\nabla_\lambda \mathscr{L}\| &= 0.5 \\ \|\nabla_\mu \mathscr{L}\| &= 0.5 \end{aligned}$$

$$\frac{p_1-2}{p_1-1} \approx 0.0909, \quad \frac{p_2-2}{p_2-1} = \frac{8}{9}$$

Using constants $c_1 = 1.0$, $c_2 = 0.5$,

$$\begin{aligned} \delta_x &\approx 1.0 \cdot \frac{15}{15^{0.0909}} + 0.5 \cdot \frac{15}{15^{8/9}} \approx 11.91 + 2.39 = 14.30 \\ \delta_\lambda &\approx -0.53 - 0.28 = -0.81 \\ \delta_\mu &\approx -0.81 \end{aligned}$$

Apply learning rate $\eta = 0.01$,

$$\begin{aligned} x^{(1)} &= 1.5 - 0.01 \cdot 14.30 = 1.357 \\ \lambda^{(1)} &= 0 + 0.01 \cdot (-0.81) = -0.0081 \\ \mu^{(1)} &= \max(0, 0 - 0.0081) = 0 \end{aligned}$$

Algorithm 18 Fixed-Time Saddle Point Optimization for $f(x) = 5x^2$ with constraints $x \geq 1$, $x \leq 3$

Input: Step size $\eta > 0$, constants $c_1, c_2 > 0$, exponents $p_1 > 2$, $p_2 > 2$
Initialize $x \in \mathbb{R}$, $\lambda \in \mathbb{R}$, $\mu \geq 0$
for $k = 0$ to max_steps **do**
  $g_x \leftarrow 10x + \lambda - \mu$
  $g_\lambda \leftarrow x - 3$
  $g_\mu \leftarrow 1 - x$
  $n_x \leftarrow |g_x| + \varepsilon, \quad n_\lambda \leftarrow |g_\lambda| + \varepsilon, \quad n_\mu \leftarrow |g_\mu| + \varepsilon$
  $\delta_x \leftarrow c_1 \cdot \frac{g_x}{n_x^{\frac{p_1-2}{p_1-1}}} + c_2 \cdot \frac{g_x}{n_x^{\frac{p_2-2}{p_2-1}}}$
  $\delta_\lambda \leftarrow c_1 \cdot \frac{g_\lambda}{n_\lambda^{\frac{p_1-2}{p_1-1}}} + c_2 \cdot \frac{g_\lambda}{n_\lambda^{\frac{p_2-2}{p_2-1}}}$
  $\delta_\mu \leftarrow c_1 \cdot \frac{g_\mu}{n_\mu^{\frac{p_1-2}{p_1-1}}} + c_2 \cdot \frac{g_\mu}{n_\mu^{\frac{p_2-2}{p_2-1}}}$
  $x \leftarrow x - \eta \cdot \delta_x$
  $\lambda \leftarrow \lambda + \eta \cdot \delta_\lambda$
  $\mu \leftarrow \max(0, \mu + \eta \cdot \delta_\mu)$
  if $\max(|g_x|, |g_\lambda|, |g_\mu|) < \text{tol}$ **then**
    BREAK
  end if
end for
Output: Final x, λ, μ

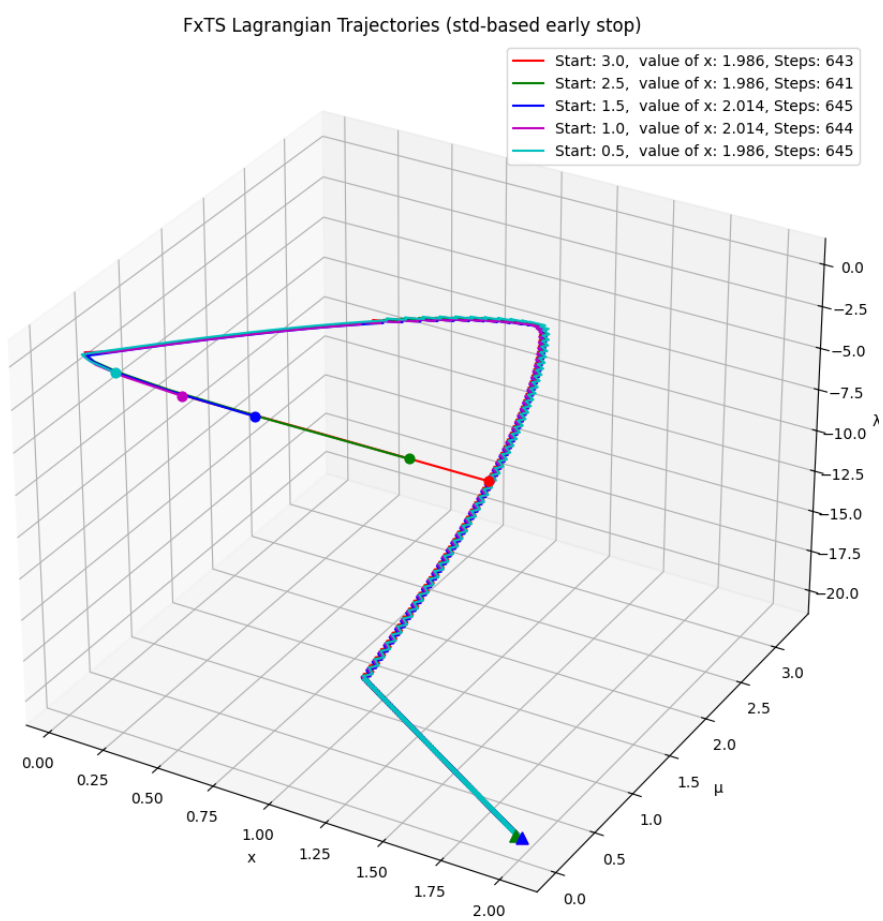


Figure 28: Fixed time - Lagrangian

Example 2 - Saddle point with two constraints

Consider the saddle-point problem:

$$\min_{x,y} \max_{\lambda,\mu\geq 0} \quad f(x,y) = x^2 - y^2$$

subject to:

$$h(x,y) = x + y - 1 = 0, \qquad g(x,y) = x^2 + y^2 - 0.25 \leq 0$$

Lagrangian Function

$$\mathscr{L}(x,y,\lambda,\mu) = x^2 - y^2 + \lambda(x+y-1) + \mu(x^2+y^2-0.25)$$

Gradient of the Lagrangian

$$\begin{aligned}
\frac{\partial \mathscr{L}}{\partial x} &= 2x + \lambda + 2\mu x \\
\frac{\partial \mathscr{L}}{\partial y} &= -2y + \lambda + 2\mu y \\
\frac{\partial \mathscr{L}}{\partial \lambda} &= x + y - 1 \\
\frac{\partial \mathscr{L}}{\partial \mu} &= x^2 + y^2 - 0.25
\end{aligned}$$

One Evaluation Step Let:

$$x = 0.2, \quad y = 0.8, \quad \lambda = -1.0, \quad \mu = 0.1$$

Then:

$$\frac{\partial \mathscr{L}}{\partial x} = 0.4 - 1.0 + 0.04 = \boxed{-0.560}, \qquad \frac{\partial \mathscr{L}}{\partial y} = -1.6 - 1.0 + 0.16 = \boxed{-2.440}$$

$$\frac{\partial \mathscr{L}}{\partial \lambda} = 0.2 + 0.8 - 1 = \boxed{0.000}, \qquad \frac{\partial \mathscr{L}}{\partial \mu} = 0.04 + 0.64 - 0.25 = \boxed{0.430}$$

Fixed-Time Power Newton Update Using parameters:

$$\eta = 0.01, \quad p_1 = 2.1, \quad p_2 = 10, \quad c_1 = 1.0, \quad c_2 = 0.5, \quad \varepsilon = 1 \times 10^{-8}$$

Gradient norm (for x):

$$\|g_x\| = |-0.560| + \varepsilon = 0.560$$

Fixed-Time update rule:

$$\Delta x = \eta \left(\frac{c_1 g_x}{\|g_x\|^{\frac{p_1-2}{p_1-1}}} + \frac{c_2 g_x}{\|g_x\|^{\frac{p_2-2}{p_2-1}}} \right)$$

Substitute:

$$\Delta x = 0.01 \left(\frac{-0.560}{0.560^{1.1/1.1}} + \frac{-0.560 \times 0.5}{0.560^{8/9}} \right) \approx \boxed{-0.011 + (-0.010)} = \boxed{-0.021}$$

$$x_{\text{new}} = 0.2 - 0.021 = \boxed{0.179}$$

Similarly, updates for y, λ, μ follow the same rule, with signs adjusted for ascent λ, μ or projection onto $\mu \geq 0$ if necessary.

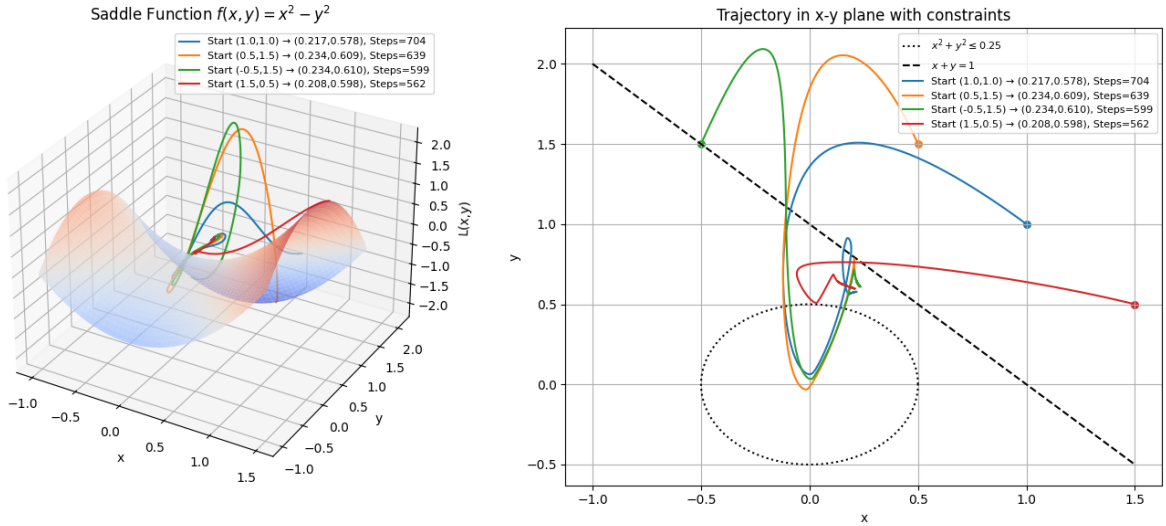


Figure 29: Fixed time - saddle points

2.5.3 Prescribed-Time

A skier uses a *time-scheduled braking system* — adjusting their speed continuously so they reach the bottom exactly at the prescribed time, regardless of their initial speed or position. The descent time is fixed and known in advance, requiring precise control throughout the entire run.

Update Rule

$$x_{k+1} = x_k - \frac{c}{T - t_k} \nabla f(x_k)$$

where:

- T is the prescribed convergence time,
- t_k is the current time or iteration,
- $c > 0$ is a constant tuning parameter,
- $\nabla f(x_k)$ is the gradient at x_k.

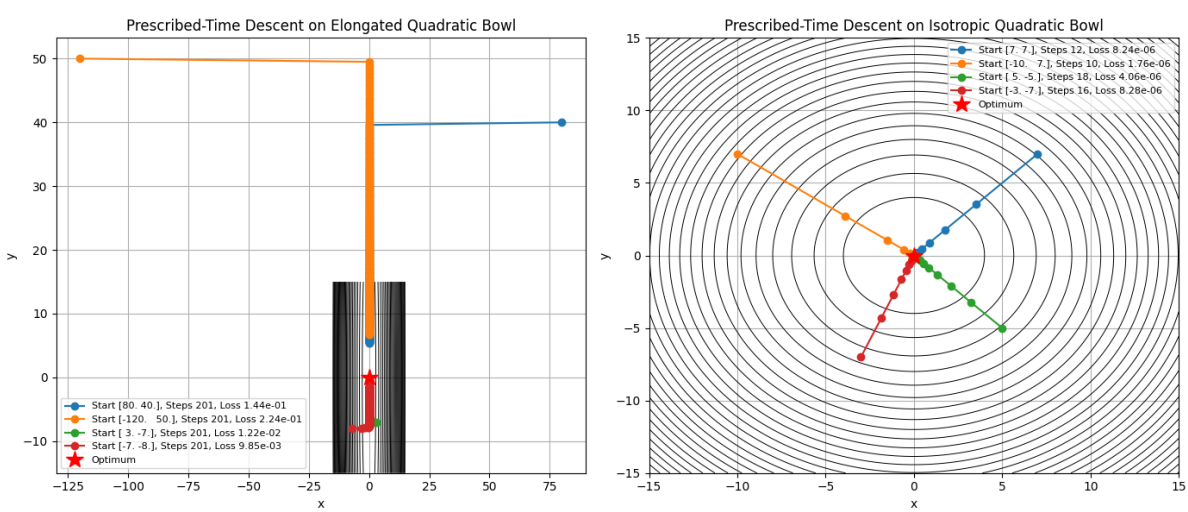


Figure 30: Prescribed time

Algorithm 19 Prescribed-Time Gradient Descent (with Step Size Capping)

Input: Initial point x_0, gradient function ∇f, prescribed time T, constant $c > 0$, max step size $\alpha_{\max}$, time step Δt
Initialize: $x \leftarrow x_0$, $t \leftarrow 0$
while $t < T$ and $f(x) > \epsilon$ **do**
    Compute gradient: $g \leftarrow \nabla f(x)$
    Compute step size: $\alpha \leftarrow \min\left(\frac{c}{\max(T-t,\delta)}, \alpha_{\max}\right)$
    Update: $x \leftarrow x - \alpha \cdot g$
    Advance time: $t \leftarrow t + \Delta t$
end while
Return: x

2.5.4 Prescribed time + Fixed time

An ice skater aims to glide smoothly into a marked finish zone in exactly 4 seconds. They plan their momentum and turns accordingly, but their skates are equipped with an intelligent braking system that automatically engages to correct overshooting or delays — ensuring they stop precisely at the target, even if their initial plan wasn't perfect.

Update Rule

$$x_{k+1} = x_k - \left(\frac{c_1}{T - t_k} + c_2\right)\nabla f(x_k)$$

where:

- T is the prescribed convergence time,
- t_k is the current time or iteration,
- $c_1 > 0$ controls the prescribed-time component,
- $c_2 > 0$ ensures fixed-time robustness,
- $\nabla f(x_k)$ is the gradient of the function at x_k.

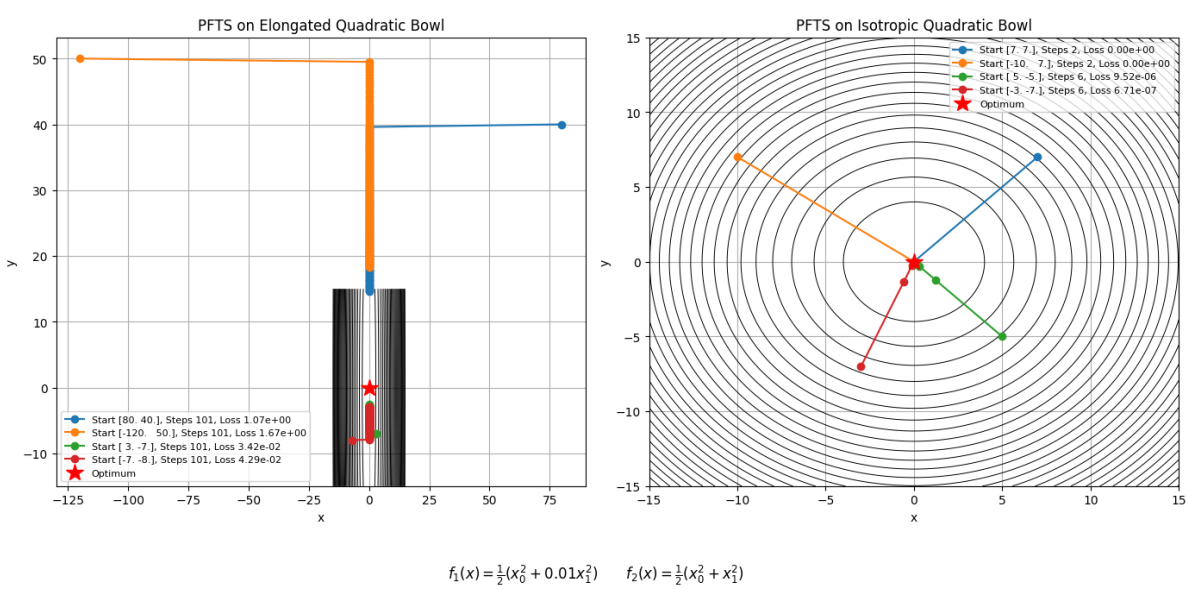


Figure 31: Prescribed + Fixed time - Near and far

In order to observe the advantage of PFTS over PTS, it is essential to start the descent closer to the optimal point, where the gradient of the function becomes small or vanishes. In such flat regions, the PTS method tends to stall as the gradient-driven urgency term weakens. However, PFTS introduces an additional constant gain, ensuring continued progress toward the optimum even when the gradient approaches zero.

When starting from farther points, both methods behave similarly because the gradient is large and dominates the dynamics. In this regime, the effect of the constant gain in PFTS becomes negligible, and both methods converge at comparable rates.

Algorithm 20 PFTS Gradient Descent

Input: Objective $f(x)$, gradient $\nabla f(x)$, prescribed time $T > 0$, constants $c_1 > 0$, $c_2 > 0$, step size Δt, initial point x_0
Initialize: $x \leftarrow x_0$, $t \leftarrow 0$
while $t < T$ and $f(x) > \varepsilon$ **do**
    Compute gradient: $g \leftarrow \nabla f(x)$
    Compute step size: $\alpha(t) \leftarrow \frac{c_1}{T-t} + c_2$
    Update: $x \leftarrow x - \alpha(t) \cdot g$
    Update time: $t \leftarrow t + \Delta t$
end while
Return: $x^\star \leftarrow x$

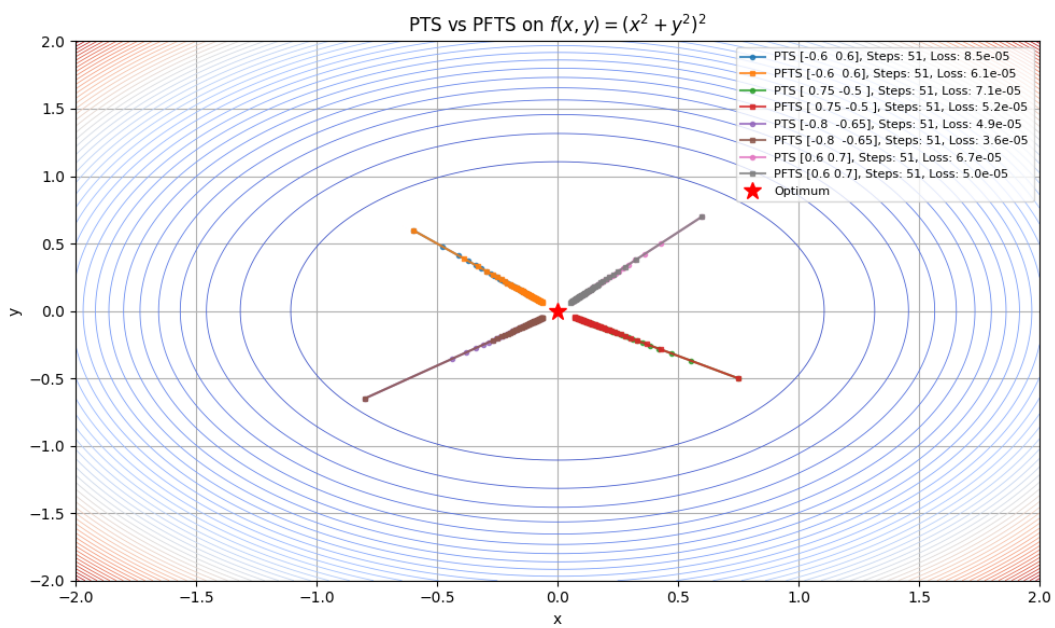


Figure 32: Prescribed + Fixed time - Near points

2.5.5 Proximal fixed time

A skier is descending an icy slope but cannot brake abruptly — perhaps their skis have limited grip or their legs are too tired to push hard. Instead of slamming the brakes, they use controlled, measured effort that smoothly adapts as they descend. Despite this restriction, they still arrive at the bottom exactly on time, using an intelligent system that compensates for their limited braking capacity.

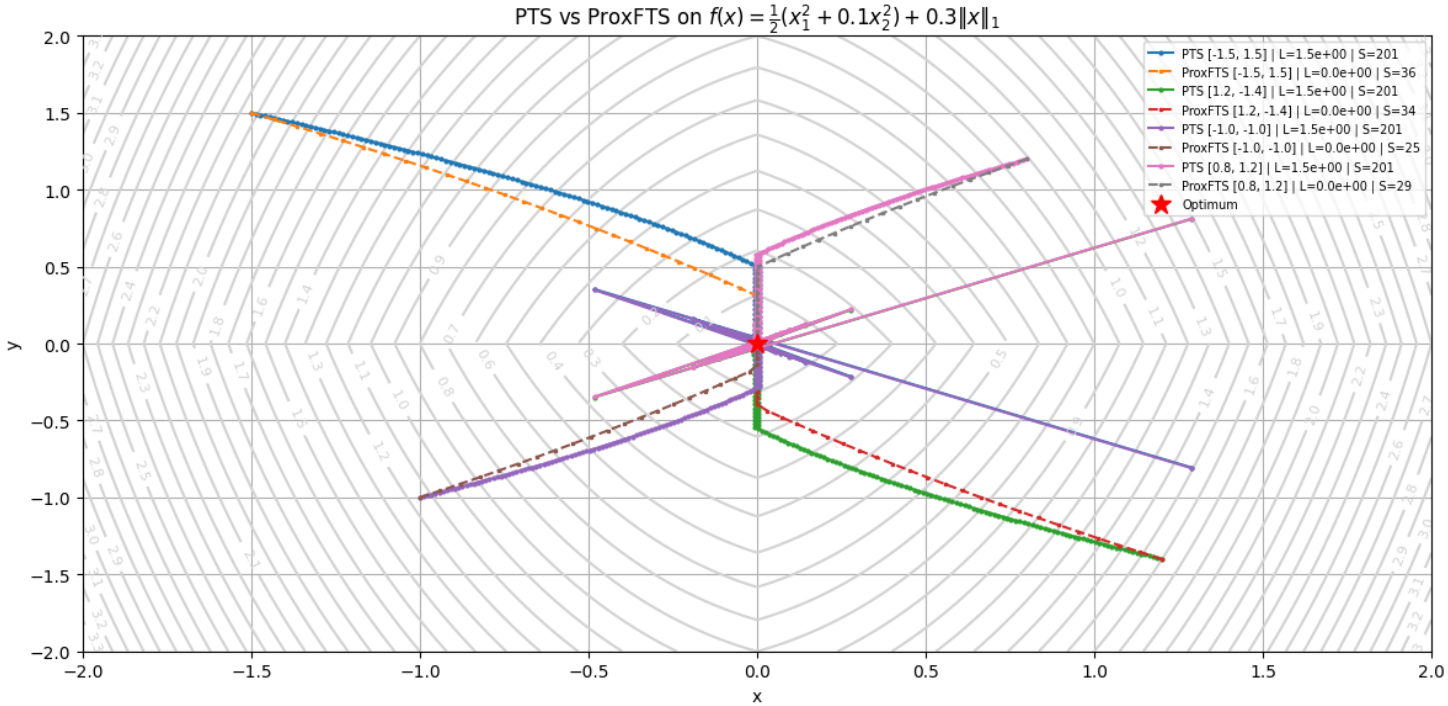


Figure 33: Proximal Time

Update Rule

$$x_{k+1} = \text{prox}_{\eta\phi}\left(x_k - \eta\left[c_1 \nabla f(x_k)^{\gamma_1} + c_2 \nabla f(x_k)^{\gamma_2}\right]\right)$$

Where

- $f(x)$ is the objective function.
- $\nabla f(x_k)$ is the gradient at iteration k.
- $\gamma_1 < 1 < \gamma_2$ are fixed-time exponents.
- $c_1,\ c_2 > 0$ are gain coefficients.
- $\eta > 0$ is the step size.
- $\phi(x)$ is a convex (possibly nonsmooth) regularizer, e.g., ℓ_1 norm or indicator function.
- $\text{prox}_{\eta\phi}(z)$ denotes the proximal operator:

$$\text{prox}_{\eta\phi}(z) = \arg\min_x \left\{\phi(x) + \frac{1}{2\eta}\|x - z\|^2\right\}$$

This update ensures fixed-time convergence even when constraints or non-smooth penalties limit the direct application of large gradient steps. It is especially useful in systems with bounded control effort, frictional dynamics, or structured sparsity.

Algorithm 21 Proximal Fixed-Time Stability (ProxFTS) Gradient Descent

Input: Objective $f(x)$, gradient $\nabla f(x)$, regularizer $\phi(x)$, step size η, total time $T > 0$, constants $c_1, c_2 > 0$, exponents $\gamma_1 < 1 < \gamma_2$, initial point x_0
Initialize: $x \leftarrow x_0$, $t \leftarrow 0$, $\Delta t \ll T$
while $t < T$ and $f(x) > \varepsilon$ **do**
  Compute gradient: $g \leftarrow \nabla f(x)$
  Compute scaled direction:
    $v \leftarrow c_1 \cdot |g|^{\gamma_1} \cdot \text{sign}(g) + c_2 \cdot |g|^{\gamma_2} \cdot \text{sign}(g)$
  Descent step: $z \leftarrow x - \eta v$
  Apply proximal operator:
    $x \leftarrow \text{prox}_{\eta\phi}(z)$
  Increment time: $t \leftarrow t + \Delta t$
end while
Output: $x^\star \leftarrow x$

ProxFTS is useful when you want guaranteed convergence within a fixed time while also dealing with nonsmooth, constrained, or effort-limited systems.

2.6 Accelerating convergence

In constrained optimization, solving problems directly in their original form can be challenging due to the complexity of enforcing equality and inequality constraints. Reformulation techniques aim to restructure these problems into forms that are more amenable to numerical methods—typically by incorporating constraints into the objective function. This includes methods such as:

- **Penalty Methods** Softly enforce constraints by adding a penalty term to the objective that increases as constraints are violated.
- **Barrier Methods** Restrict the domain to the interior of the feasible region using logarithmic or exponential terms that become infinite near constraint boundaries.
- **Augmented Lagrangian Methods** Blend dual variable updates with quadratic penalty terms, improving both convergence and feasibility handling over basic penalty methods.

These approaches enable the use of unconstrained solvers and often improve numerical stability, especially in large-scale or nonlinear problems. In order to speedup convergence, the following methods can be used depending on the problem and the data that is available.

2.6.1 Momentum-Based Methods (Unconstrained Optimization)

For a differentiable function $f(x)$, where $x \in \mathbb{R}^n$, the following two momentum based methods are useful.

3.6.1.1 Heavy Ball (HB) Method

By adding a fraction of the previous update direction to the current gradient step, **Heavy Ball** method introduces momentum, enabling faster convergence and helping to overcome slowdowns in narrow or curved regions.

$$sx_{k+1} = x_k - \alpha \nabla f(x_k) + \beta(x_k - x_{k-1})$$

where:

- $\alpha > 0$ is the learning rate,
- $\beta \in [0, 1)$ is the momentum coefficient.

- The objective function $f(x)$ must be μ-**strongly convex**, i.e., there exists $\mu > 0$ such that:

$$f(y) \geq f(x) + \nabla f(x)^T (y - x) + \frac{\mu}{2} \|y - x\|^2$$

- $f(x)$ must be L-**smooth**, meaning the gradient is Lipschitz continuous:

$$\|\nabla f(x) - \nabla f(y)\| \leq L \|x - y\|$$

- The condition number $\kappa = \frac{L}{\mu}$ impacts convergence rate.
- The step size α and momentum β must satisfy specific stability bounds depending on μ and L.

3.6.1.2 Nesterov Accelerated Gradient (NAG)

Unlike the Heavy Ball method, which adds momentum after computing the gradient, **Nesterov Accelerated Gradient** performs a look-ahead step before the gradient evaluation. This anticipatory mechanism allows more responsive adjustments and often improves convergence stability.

$$x_{k+1} = x_k - \alpha \nabla f\left(x_k + \beta (x_k - x_{k-1})\right)$$

where:

- $\alpha > 0$ is the learning rate,
- $\beta \in [0, 1)$ is the momentum coefficient.
- $f(x)$ must be μ-**strongly convex**.
- $f(x)$ must be L-**smooth** (gradient Lipschitz).
- With properly chosen α and β, NAG achieves a convergence rate of:

$$f(x_k) - f(x^*) \leq \mathcal{O}\left(\left(1 - \sqrt{\mu / L}\right)^k\right)$$

  which is faster than standard gradient descent.

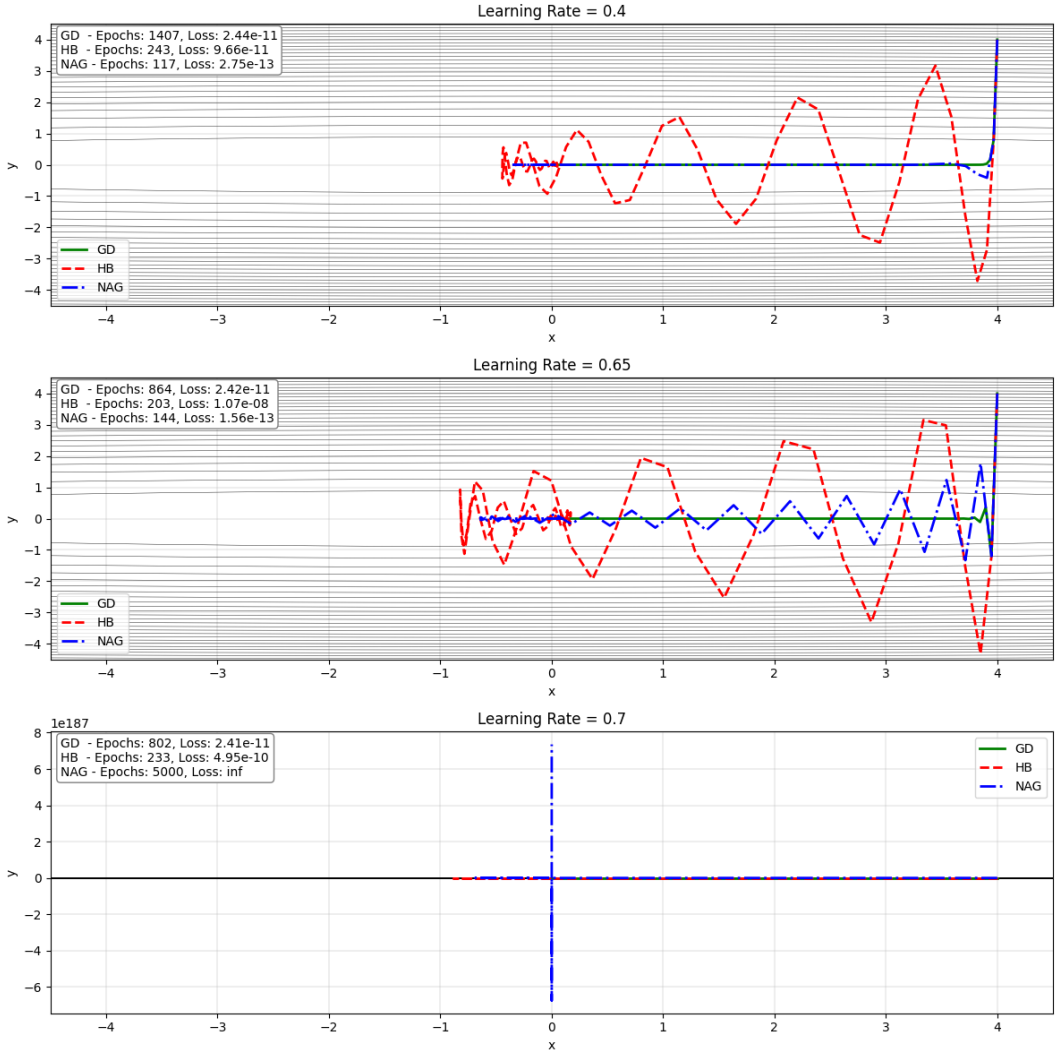


Figure 34: Gradient descent vs Heavy ball vs Nesterov AGD

2.6.2 Supporting Techniques (Stabilize Training, Not Direct Acceleration)

3.6.2.1 Regularization

In some settings where the steep ness of the loss function can be improved by adding a regularization term, it boosting convergence. In the example convergence with GD below, we take $f(x) = x^4; f'(x) = 4 \cdot x^3$ and add regularization with $\lambda \cdot x^2$. The resultant convergence improves as the value of λ (steepness) is increased.

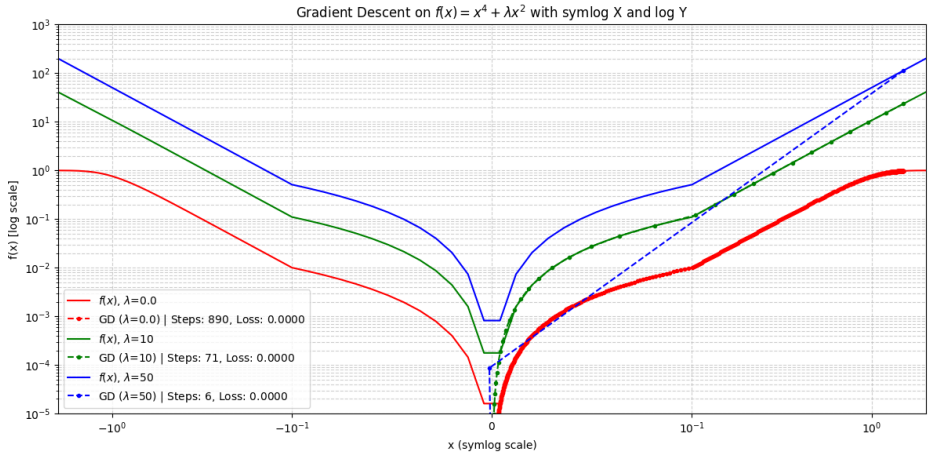


Figure 35: GD with different λ values for $\tanh(x^2) + \lambda \cdot x^2$

3.6.2.2 Normalization

If decision variables in an equation are imbalanced, the convergence slow down and in order to help restore acceleration we can use methods like normalized gradient descent (NGD) or Rescaled gradient descent (RGD). NGD will take unit steps in the direction of the gradient and will handle flat surfaces or plateaus well. RGD will converge faster if the surface is having sizable values for gradient descent.

No flat surface

Let $f(x_1, x_2) = 100 \cdot x_1^2 + 0.0005 \cdot x_2^2$ where $x1, x2 \in \mathbb{R}^N$

The parameter update in the process changes as below different types of Gradient descent.

Gradient descent $x_{k+1} = x_k - \eta \cdot \nabla f(x)$

Normalized GD $x_{k+1} = x_k - \eta \cdot \frac{\nabla f(x)}{||\nabla f(x)||+\epsilon}$

Rescaled GD $x_{k+1} = x_k - \eta \cdot D^{-1} \nabla f(x)$;

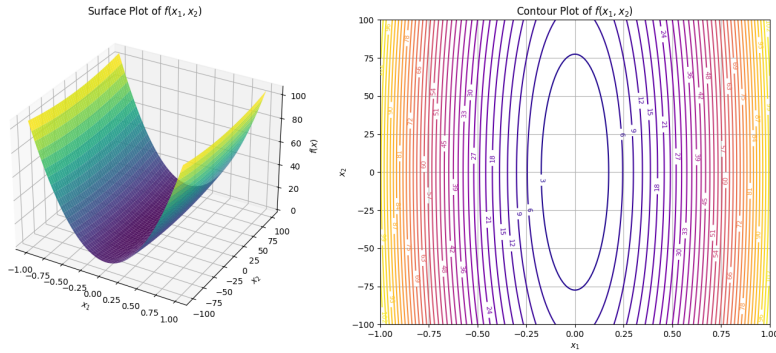
$$x_{k+1} = x_k - \eta \cdot \begin{bmatrix} 0.005 & 0 \\ 0 & 1000 \end{bmatrix} \begin{bmatrix} 200x_1 \\ 0.001x_2 \end{bmatrix} = x_k - \eta \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Comparison of GD, NGD, and RGD(p=2) for $f(x_1, x_2) = 100x_1^2 + 0.0005x_2^2$

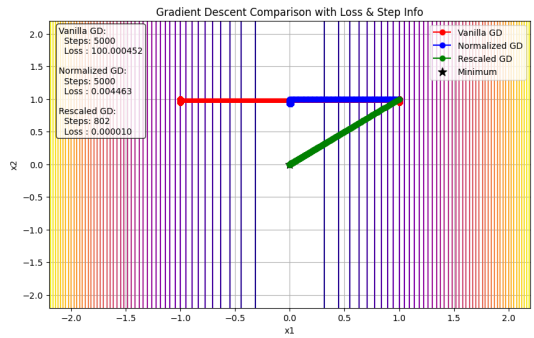
Iteration	**GD**	**NGD**	**RGD(p=2)**
0	(1.000, 1.000)	(1.000, 1.000)	(1.000, 1.000)
1	(0.980, 0.9999)	(0.929, 0.929)	(0.950, 0.000)
2	(0.960, 0.9998)	(0.864, 0.864)	(0.9025, 0.000)
3	(0.940, 0.9997)	(0.802, 0.802)	(0.8574, 0.000)

Flat surface
Let $f(x) = tanh(x^2)$ where $x \in \mathbb{R}^N$; $f'(x) = 2x\left(1 - \tanh^2(x^2)\right)$

In the example, we start movement from either sides of the graph – RGD(p=2) from left and NGD from



(a) No Flat surface



(b) GD vs RGD vs NGD

Figure 36: Comparison of function surface and optimization behavior

right on the flat surface. NGD due to its unit steps moves out of flat surface fast and rushes towards optimal point.

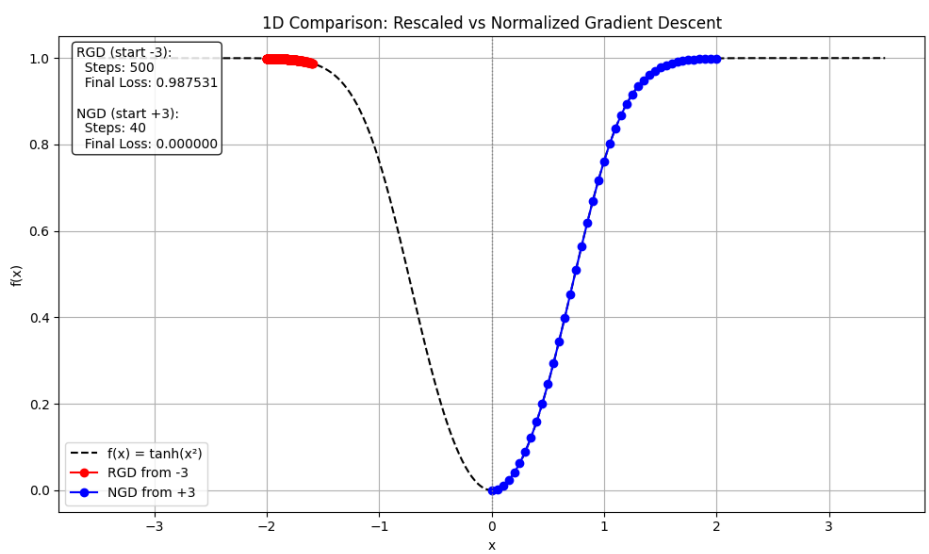


Figure 37: Tanh flat surface

2.6.3 Constrained Optimization Methods

2.6.3.1 Augmented Lagrangian

Minimize the quadratic objective:

$$f(x, y) = (x-2)^2 + (y-1)^2$$

subject to the constraints:

$$\text{Equality: } g(x,y) = x + y - 1 = 0, \qquad \text{Inequality: } h(x,y) = x - 0.5 \geq 0$$

Augmented Lagrangian Form. The augmented Lagrangian is defined as:

$$\mathscr{L}_\rho(x,y,\lambda,\mu) = f(x,y) + \lambda g(x,y) + \frac{\rho}{2} g(x,y)^2 + \mu \cdot \max(0, -h(x,y)) + \frac{\rho}{2} \cdot \max(0, -h(x,y))^2$$

One Iteration at Initial Point $(x_0, y_0) = (0,0)$**.** Let $\lambda = 0$, $\mu = 0$, and $\rho = 10$. Then:

$$f(0,0) = (0-2)^2 + (0-1)^2 = 4 + 1 = 5$$

$$g(0,0) = 0 + 0 - 1 = -1, \qquad h(0,0) = 0 - 0.5 = -0.5$$

Since $h(0,0) < 0$, the inequality is violated.

Compute the augmented Lagrangian,

$$\mathscr{L}_\rho(0,0,0,0) = 5 + 0 + \frac{10}{2}(-1)^2 + 0 + \frac{10}{2}(-0.5)^2 = 5 + 5 + 1.25 = 11.25$$

Gradient Descent Update. Compute the gradient numerically or symbolically (not shown), then update,

$$x_1 = x_0 - \eta \nabla_x \mathscr{L}_\rho(x_0, y_0, \lambda, \mu)$$

$$y_1 = y_0 - \eta \nabla_y \mathscr{L}_\rho(x_0, y_0, \lambda, \mu)$$

Multiplier Updates.

$$\lambda_1 = \lambda_0 + \rho \cdot g(x_1, y_1), \qquad \mu_1 = \max(0, \mu_0 + \rho \cdot \max(0, -h(x_1, y_1)))$$

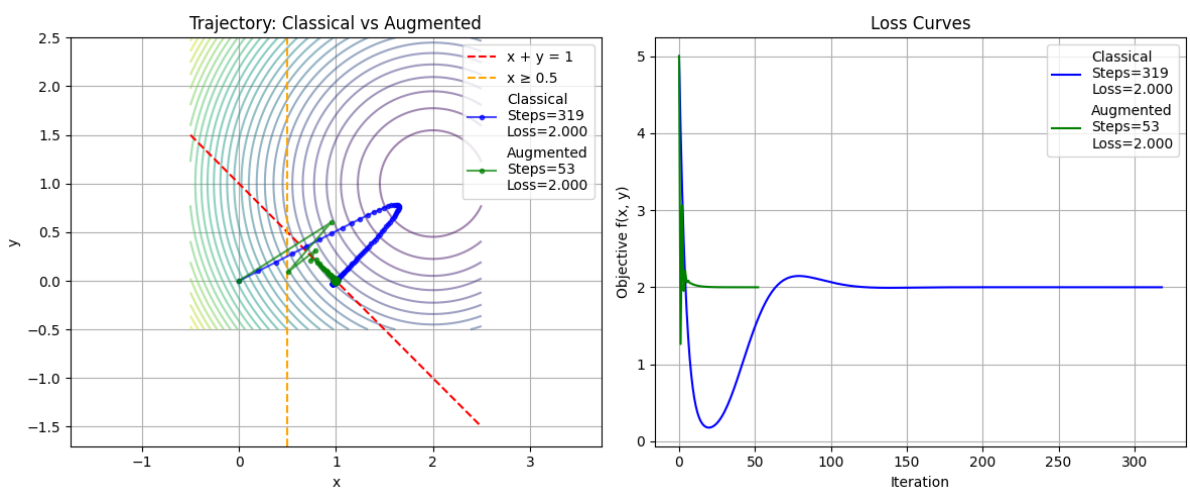


Figure 38: Augmented Lagrangian optimization

Algorithm 22 Augmented Lagrangian Optimization

```
Input: Initial point x_0, multipliers λ_0, μ_0, penalty ρ > 0, step size η
Initialize x ← x_0, λ ← λ_0, μ ← μ_0
for k = 0 to MaxIterations do
    Compute constraint values:
        g ← g(x),    h ← h(x)
    Compute gradient of objective: ∇f(x)
    Compute constraint gradients: ∇g(x), ∇h(x)
    Compute augmented gradient:
        ∇L_ρ ← ∇f(x) + λ∇g(x) + ρg · ∇g(x)
    if h > 0 then
        ∇L_ρ ← ∇L_ρ + μ∇h(x) + ρh · ∇h(x)
    end if
    Gradient step: x ← x − η · ∇L_ρ
    Update multipliers:
        λ ← λ + ρ · g
        μ ← max(0, μ + ρ · h)
    if convergence criteria met then
        break
    end if
end for
Return: Final point x, multipliers λ, μ
```

2.6.3.2 Method of Multipliers

The Method of Multipliers (MoM) solves constrained optimization problems by minimizing an augmented Lagrangian function. For equality constraints $c(\mathbf{x}) = 0$, each iteration consists of the following updates:

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \alpha \cdot \left(\nabla f(\mathbf{x}^k) + \left(\lambda^k + \rho^k \cdot c(\mathbf{x}^k)\right) \cdot \nabla c(\mathbf{x}^k)\right)$$

$$\lambda^{k+1} = \lambda^k + \rho^k \cdot c(\mathbf{x}^{k+1})$$

$$\rho^{k+1} = \begin{cases} \beta \cdot \rho^k & \text{if} \quad |c(\mathbf{x}^{k+1})| > \tau \\ \rho^k & \text{otherwise} \end{cases}$$

where α is the step size, ρ^k is the penalty parameter, λ^k is the Lagrange multiplier, and $\beta > 1$ controls penalty scaling.

This method combines gradient descent with multiplier updates, balancing objective minimization and constraint enforcement adaptively. Compared to penalty methods, MoM avoids instability caused by large penalties, improves numerical conditioning, and accelerates convergence by guiding iterates toward feasibility using multipliers rather than relying solely on penalties.

Example 1

Consider minimizing $f(x, y) = x^2 + y^2$ subject to $x + y = 1$ using Method of Multipliers. The gradient step is

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \alpha \cdot \left(\nabla f(\mathbf{x}^k) + \left(\lambda^k + \rho^k \cdot c(\mathbf{x}^k)\right) \cdot [1, 1]\right)$$

and multiplier update is as below where $c(x, y) = x + y - 1$.

$$\lambda^{k+1} = \lambda^k + \rho^k \cdot c(\mathbf{x}^{k+1})$$

Penalty ρ is updated as below where $\beta = 2$ and $\tau = 0.1$.

$$\rho^{k+1} = \begin{cases} \beta \cdot \rho^k & \text{if } |c(\mathbf{x}^{k+1})| > \tau \\ \rho^k & \text{otherwise} \end{cases}$$

Given $x^0 = 0,\ y^0 = 0,\ \lambda^0 = 0,\ \rho^0 = 10,\ \alpha = 0.1$, compute one iteration step as follows.

Constraint violation is

$$c(\mathbf{x}^0) = 0 + 0 - 1 = -1$$

Gradient of f is

$$[2x^0, 2y^0] = [0, 0]$$

Multiplier term is

$$\lambda^0 + \rho^0 \cdot c(\mathbf{x}^0) = 0 + 10 \times (-1) = -10$$

Gradient step is $[0,0] + (-10) \times [1,1] = [-10, -10]$.
Update variables,

$$x^1 = 0 - 0.1 \times (-10) = 1$$

$$y^1 = 0 - 0.1 \times (-10) = 1$$

Compute $c(\mathbf{x}^1) = 1 + 1 - 1 = 1$.
Update multiplier,

$$\lambda^1 = 0 + 10 \times 1 = 10$$

Since $|c(\mathbf{x}^1)| = 1 > 0.1$, update ρ,

$$\rho^1 = 2 \times 10 = 20$$

The table below shows values for 5 iterations using this method.

Table 9: Stabilized Method of Multipliers Iteration Data

Iter	x_k	y_k	$f(x_k, y_k)$	λ_k	ρ_k
0	0.09901	0.09901	0.01961	-0.50000	10.0
10	0.50578	0.50578	0.51162	-2.06426	113.9
20	0.55017	0.55017	0.60538	-120.81717	1946.2
30	0.51688	0.51688	0.53432	-254.72118	22168.4
40	0.50073	0.50073	0.50147	-336.46376	568151.3

This process is repeated until constraint violation and variables stabilize.

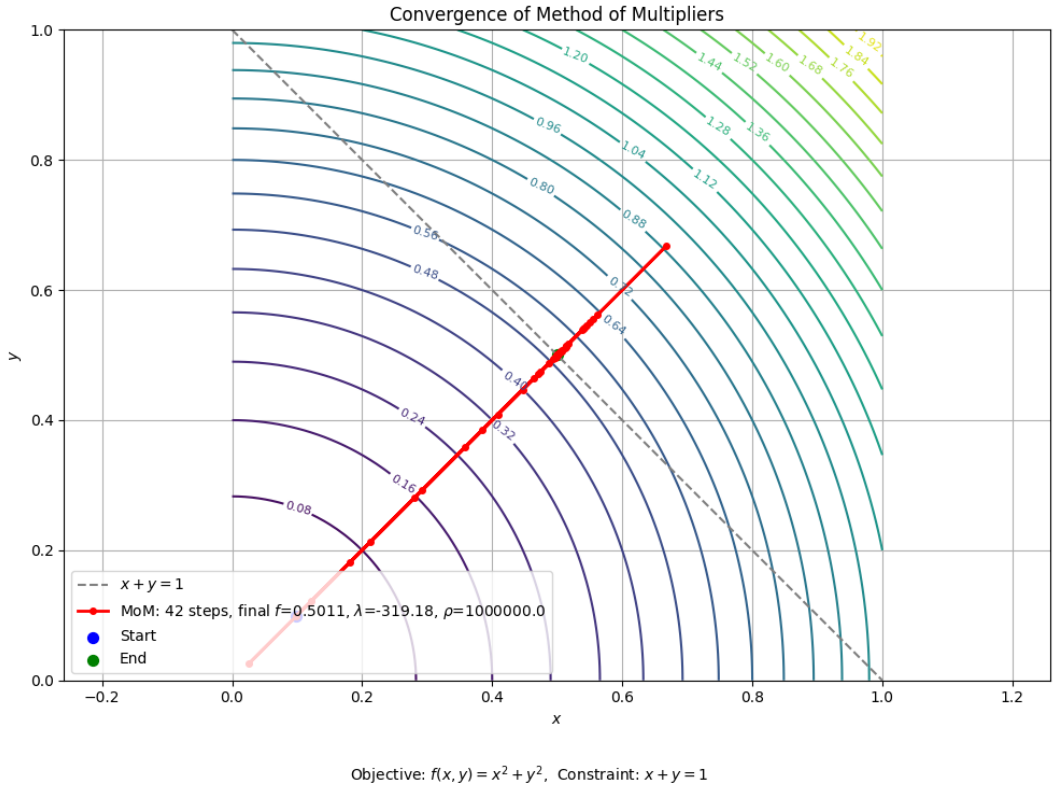


Figure 39: Method of Multipliers

Example 2

Consider the optimization problem as below.

$$\min_{x,y} \quad 0.5x^2 + 2y^2 + \sin(3x)\cos(2y) + 0.2(x-y)^2$$

subject to the linear constraint:

$$x + y = 1$$

This is a non-convex objective function due to the sinusoidal interaction term. We compare two constrained optimization methods with multiple starting points. Convergence is declared when the standard deviation of the last 10 iterates is below a small tolerance.

- **Projected Gradient Descent (PGD)** After each gradient descent step, the iterates are projected back onto the feasible set defined by the constraint $x + y = 1$.
- **Method of Multipliers (MoM)** An augmented Lagrangian is optimized using gradient descent. The Lagrange multiplier is updated at each iteration to enforce the constraint progressively.

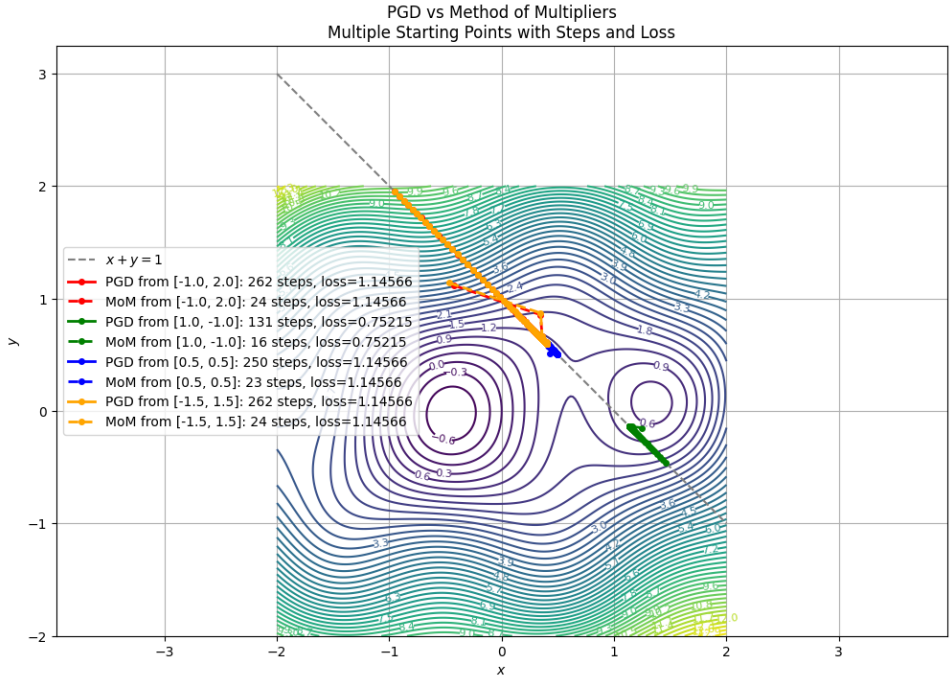


Figure 40: PGD vs Method of Multipliers

Algorithm 23 Stabilized Method of Multipliers

Initialize x, y, λ, ρ
Set parameters: α_0, η, clip_λ, $\text{clip}_{\text{mult}}$, $\rho_{\max}$, tol
for $k = 1$ to max_iters **do**
  Compute constraint violation: $c = x + y - 1$
  Compute adaptive learning rate: $\alpha = \frac{\alpha_0}{1+\rho/1000}$
  Compute gradients,
    $\nabla f_x = 2x$
    $\nabla f_y = 2y$
    $\text{multiplier} = \lambda + \rho \cdot c$
  Clip multiplier term,
    $\text{multiplier} = \text{clip}(\text{multiplier}, -\text{clip}_{\text{mult}}, \text{clip}_{\text{mult}})$
  Compute full gradients,
    $g_x = \nabla f_x + \text{multiplier}$
    $g_y = \nabla f_y + \text{multiplier}$
  Update variables,
    $x \leftarrow x - \alpha \cdot g_x$
    $y \leftarrow y - \alpha \cdot g_y$
  Update multiplier with damping,
    $\lambda \leftarrow \lambda + \rho \cdot c \cdot \eta$
  Clip multiplier,
    $\lambda \leftarrow \text{clip}(\lambda, -\text{clip}_\lambda, \text{clip}_\lambda)$
  if constraint violation has not decreased **then**
    $\rho \leftarrow \min(\rho \cdot \text{growth_factor}, \rho_{\max})$
  end if
  if standard deviation of last iterates $<$ tol **then**
    stop
  end if
end for

The dual ascent method approaches constrained optimization problems by solving the dual problem directly. In this approach, the primal variables (decision variables) are optimized in an inner loop,

and the dual variables (Lagrange multipliers) are updated iteratively using constraint violations. This represents a form of gradient ascent in the dual space, seeking to maximize the dual function.

2.6.3.3 [todo] Penalty method

2.6.3.4 [todo] Barrier method

2.7 [todo] Gradient Fields

3 Distributing optimization

Three countries each operate their own networks of ATMs. Due to data privacy regulations and operational policies, they cannot share raw ATM transaction data with each other. However, all three countries need to forecast daily cash withdrawals at each ATM to prevent cash shortages or unnecessary oversupply. Each country exhibits distinct cash withdrawal patterns as below.

- **Country A:** Cash withdrawals peak on weekends.
- **Country B:** Withdrawals spike at the end of each month (salary days).
- **Country C:** Withdrawals are stable but decrease on public holidays.

Distributed Learning Approach

Since raw ATM data cannot be shared, each country trains its own forecasting model locally. A distributed optimization method is used to collaboratively train a global cash demand forecasting model

- Each country trains a local model using its own data.
- Instead of sharing data, countries share model updates.
- A central coordinator aggregates these updates to improve the global model.

This distributed approach ensures privacy while enabling collaborative learning, allowing each country to benefit from shared knowledge without compromising data security.In order to achieve the said logic, we need a method or (combination of methods) which lead to high convergence guarantee and helps with problem decomposability.

3.1 Dual Ascent Method

The dual ascent method is an optimization approach designed to solve constrained convex problems by iteratively updating dual variables associated with the constraints. Consider a general constrained optimization problem

$$\begin{aligned} minimize \quad & f(x) \\ \text{subject to} \quad & Ax = b \end{aligned}$$

The corresponding Lagrangian is as below where $\lambda \geq 0$ is the Lagrange multiplier.

The Fenchel dual equation below that shows relationship between f and its conjugate (f^*)

$$f^*(y) = \max_{x \in R^n} \left[y^T \cdot x - f(x) \right]$$

$$
\begin{aligned}
\min_{x,\lambda} L(x,\lambda) &= \min_{x,\lambda} f(x) + \lambda^T(Ax-b) \\
&= \min_{x,\lambda} -\lambda^T b + f(x) + \lambda^T Ax \\
&= -\lambda^T b + \min_x \left[f(x) + (\lambda^T A)x \right] \\
&= -\lambda^T b + \min_x \left[f(x) + (-A^T\lambda)^T x \right] \\
&= -\lambda^T b - \max_x \left[(-A^T\lambda)^T x - f(x) \right] \\
&= -\lambda^T b - f^*(-A^T\lambda)
\end{aligned} \tag{3.1}
$$

The dual ascent method alternates between minimizing the Lagrangian over the primal variable x and updating the dual variable λ using a gradient ascent step. Specifically, the updates are as follows.
Update Steps.

$$
x^{k+1} = \arg\min_x L(x, \lambda^k) \tag{Primal Update}
$$

$$
\lambda^{k+1} = \lambda^k + \alpha_k \cdot (Ax^{k+1} - b) \tag{Dual Ascent Update}
$$

Here, α_k is a positive step size (learning rate). The dual variable λ penalizes constraint violation and is updated in the ascent direction since the dual problem involves maximizing over λ. The primal variable x is updated via descent as part of minimizing the Lagrangian.

Interpretation

- The **primal update** improves the objective while considering current constraint penalties.
- The **dual update** adjusts λ based on how much x violates the constraint $Ax = b$, effectively steering future updates towards feasibility.

This alternating process allows the algorithm to gradually enforce the constraint while improving the objective function. Dual ascent is particularly effective for resource allocation problems where a global constraint (such as total capacity or budget) must be distributed optimally among multiple agents or users.

Problem Description.
The dual ascent method can be applied to a bandwidth allocation problem.

A total bandwidth budget B_{total} must be distributed among N users over T discrete time steps. Each user i has a demand profile $d_i(t)$. The goal is to compute per-step allocations $x_i(t)$ satisfying the following

$$
\sum_{i=1}^{N} x_i(t) \le B_{\text{total}}
$$

while optimizing the utility:

$$
\min_{x_i(t)\ge 0} \sum_{i=1}^{N} \left(-d_i(t) \cdot \log(x_i(t)) + \text{Penalty}(x_i(t)) + \lambda_k \cdot x_i(t) \right)
$$

The penalty term ensures smooth convergence and models saturation effects:

$$
\text{Penalty}'(x_i) = \frac{\gamma\beta e^{\beta x_i}}{1 + e^{\beta x_i}}
$$

Update Steps.

At each time step t, the following steps are taken.

- Solve the primal allocation problem for all users in parallel:

$$x_i(t) \leftarrow \arg\min_{x_i \geq 0} \left(-d_i(t)\log(x_i) + \text{Penalty}(x_i) + \lambda_k \cdot x_i\right)$$

- Compute the residual:

$$r(t) = \sum_{i=1}^{N} x_i(t) - B_{\text{total}}$$

- Update the dual variable:

$$\lambda_{k+1} = \max(0.01, \lambda_k + \alpha \cdot r(t))$$

This problem is decomposable as each user solves their bandwidth allocation problem given a global constraint of total bandwidth. The compute of each user can be distributed by parallel processing methods like OpenMPI, OpenMP, CUDA. In this example, we take the primal solver and made that a CUDA kernel where the number of users reflect the number of blocks. After the primal compute for an iteration is done given a value of λ_k , the primal values are aggregated on the CPU and the dual variable is updated to λ_{k+1}.

The algorithms and the results are below for CPU only and CUDA based method are below.

Algorithm 24 CPU Dual Ascent Bandwidth Allocation

```
Initialize λ_k ← 1.0
for t = 1 to T do
    for i = 1 to N do
        x_i(t) ← SolvePrimal(d_i(t), λ_k)
    end for
    Compute r(t) = Σ_{i=1}^{N} x_i(t) − B_total
    λ_k ← max(0.01, λ_k + α · r(t))
end for
```

Algorithm 25 CUDA Dual Ascent Bandwidth Allocation

```
Initialize λ_k ← 1.0
for t = 1 to T do
    Launch GPU Kernel:
    Each <block,thread> computes x_i(t) ← SolvePrimal(d_i(t), λ_k) in parallel.
    Synchronize GPU threads.
    Aggregate x_i(t) values to host.
    Compute r(t) = Σ_{i=1}^{N} x_i(t) − B_total on CPU.
    λ_k ← max(0.01, λ_k + α · r(t))
end for
```

Solve Primal Subroutine

This algorithm uses projected gradient descent to optimize the local variable x_i. In each iteration, the gradient combines demand-based reciprocal, sigmoid regularization, and dual penalty terms. The update step applies a learning rate α, followed by projection onto box constraints to ensure $x_i \in [0.01, B_{\text{total}}]$.

Algorithm 26 SolvePrimal($d_i(t)$, λ_k)

Initialize $x_i \leftarrow 1.0$
for iter = 1 to max_iters **do**
  Compute gradient:
  $\nabla f(x_i) = -\frac{d_i(t)}{x_i} + \frac{\gamma \beta e^{\beta x_i}}{1 + e^{\beta x_i}} + \lambda_k$
  Update: $x_i \leftarrow x_i - \alpha \cdot \nabla f(x_i)$
  Project:
  $x_i \leftarrow \min(\max(0.01, x_i), B_{\text{total}})$
end for

return x_i

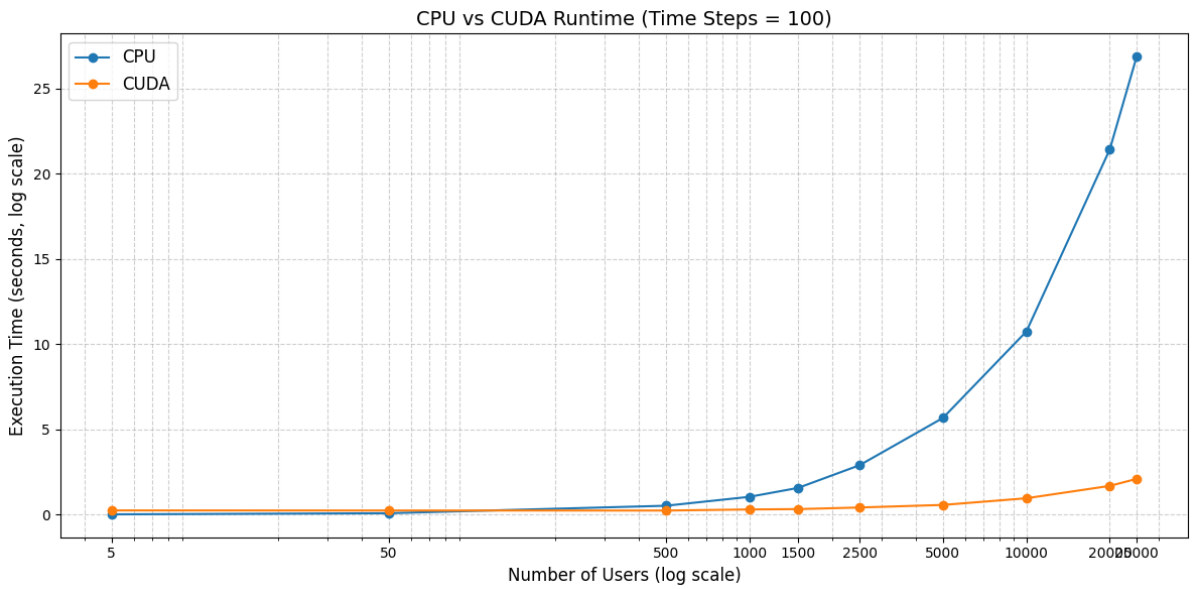


Figure 41: Comparison between CPU and CUDA kernel execution for 500 steps of primal update

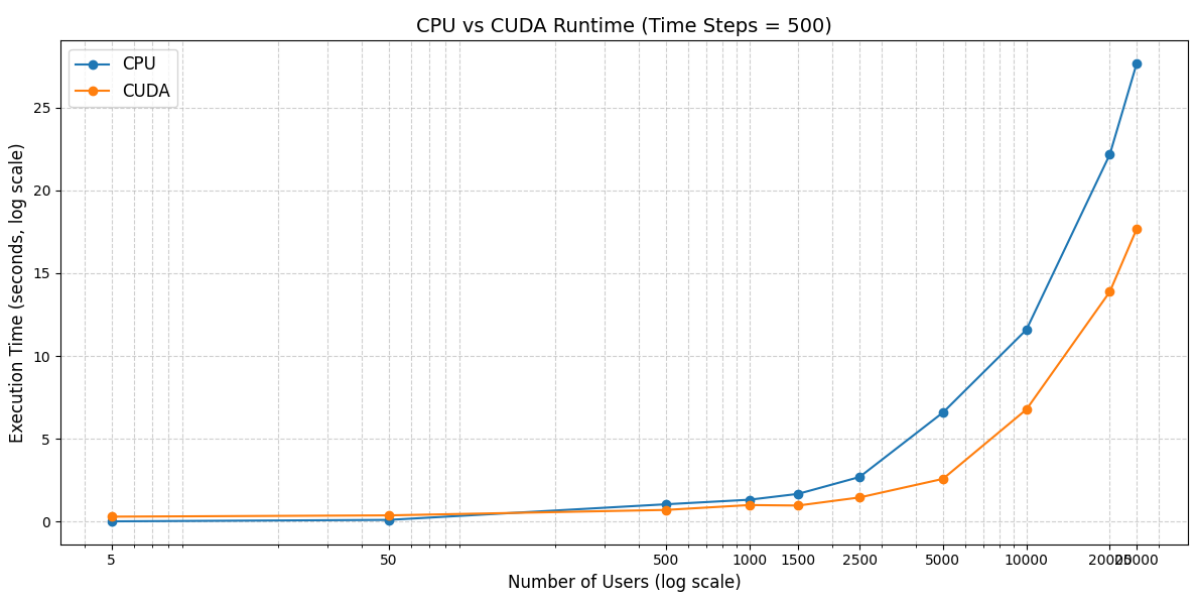


Figure 42: Comparison between CPU and CUDA kernel execution for 100 steps of primal update

3.2 ADDM

In distributed optimization, the global objective may differ from local objectives, or it may be beneficial to split the objective into two parts to simplify solving. ADMM handles this by introducing two variables and reformulating the problem as below.

$$\begin{aligned} \min_{x,z} \quad & f(x) + g(z) \\ \text{subject to} \quad & x = z \end{aligned} \tag{1}$$

The various variables above are,

- x is optimized for part of the problem involving $f(x)$.
- z is optimized for another part involving $g(z)$.
- The constraint $x = z$ ensures both solutions agree in the end.

This splitting allows each part of the problem to be solved separately, with ADMM iteratively enforcing consensus between x and z. ADMM is built on top of Dual Ascent and Method of Multipliers by splitting the problem like Dual Ascent but adding the augmented Lagrangian penalty to stabilize and enforce agreement, allowing decentralized solving with strong constraint handling.

Dual Ascent

$$\begin{aligned} x^{k+1} &= \arg\min_x \; f(x) + \lambda^k (Ax - b) \\ \lambda^{k+1} &= \lambda^k + \alpha (Ax^{k+1} - b) \end{aligned} \tag{Eq 3.2.1}$$

Method of Multipliers

$$\begin{aligned} x^{k+1} &= \arg\min_x \; f(x) + \lambda^k (Ax - b) + \frac{\rho}{2} \|Ax - b\|^2 \\ \lambda^{k+1} &= \lambda^k + \rho (Ax^{k+1} - b) \end{aligned} \tag{Eq 3.2.2}$$

Using eq 3.2.1 and 3.2.2 with the fact that $f(x)$ and $g(x)$ are optimized iteratively, the following iterations happen in ADMM execution.

$$\begin{aligned} x^{k+1} &= \arg\min_x \; f(x) + \frac{\rho}{2} \|Ax + Bz^k - c + u^k\|^2 \\ z^{k+1} &= \arg\min_z \; g(z) + \frac{\rho}{2} \|Ax^{k+1} + Bz - c + u^k\|^2 \\ u^{k+1} &= u^k + Ax^{k+1} + Bz^{k+1} - c \end{aligned} \tag{ADMM}$$

The variables – x, z and u can be defined as below.

- x_i: Local variable at node i.
- z: Global variable representing the shared solution.
- u_i: Dual variable (or scaled Lagrange multiplier) at node i. It records the disagreement between the local variable x_i and the global variable z, and adjusts updates to enforce consensus over iterations.

Among the various types of formulations for distributed optimization problem solving, the most used types are the ones formulated as below.

Consensus Problems

Each node solves the same problem on its own data but must agree on a shared solution. Local variables x_i are forced to equal a global variable z (the shared model). Example: Sensors fitting local curves must agree on a global temperature profile where typically $g(z) = 0$ (pure consensus).

$$\min_{x_i, z} \sum_{i=1}^{N} f_i(x_i) + g(z) \quad \text{s.t.} \quad x_i = z$$

Updates

$$x_i^{k+1} = \arg\min_{x_i} \; f_i(x_i) + \frac{\rho}{2}\|x_i - z^k + u_i^k\|^2$$

$$z^{k+1} = \frac{1}{N}\sum_{i=1}^{N}(x_i^{k+1} + u_i^k)$$

$$u_i^{k+1} = u_i^k + x_i^{k+1} - z^{k+1}$$

Sharing Problems

Nodes solve different local problems but share a global variable z, which represents the aggregated total (e.g., total demand or resource usage). Each node optimizes its own $f_i(x_i)$, while a proper function $g(z)$ penalizes or regulates the global total.

Example: Substations forecast local demands (x_i), but total generation (z) incurs a global cost $g(z)$ (such as energy cost or penalty for over-usage).

$$\min_{x_i,z} \sum_{i=1}^{N} f_i(x_i) + g(z) \quad \text{s.t.} \quad \sum_{i=1}^{N} x_i = z$$

Updates

$$x_i^{k+1} = \arg\min_{x_i} \; f_i(x_i) + \frac{\rho}{2}\left\|x_i - (z^k - u^k)\right\|^2$$

$$z^{k+1} = \arg\min_{z} \; g(z) + \frac{\rho}{2}\left\|z - \left(\sum_{i=1}^{N} x_i^{k+1} + u^k\right)\right\|^2$$

$$u^{k+1} = u^k + \sum_{i=1}^{N} x_i^{k+1} - z^{k+1}$$

Here, $g(z)$ is explicitly minimized, and z acts as a true decision variable rather than merely enforcing a constraint.

3.2.1 Consensus example

In a distributed temperature estimation problem, multiple sensors are placed across a city to measure ambient temperature. Sensors at different locations experience varying influences due to differences in population density, with sensors in the city center contributing more significantly due to higher density. Each sensor i estimates a single temperature value x_i using its own noisy local data and attempts to agree on a global city-wide temperature z. To reflect differing importance, the global temperature is computed as a density-weighted consensus.

$$\min_{x_i,z} \sum_{i=1}^{N} f_i(x_i) \quad \text{subject to} \quad x_i = z$$

$$f_i(x_i) = \sum_{t=1}^{T} (y_i(t) - x_i)^2$$

Each sensor updates its local temperature estimate by solving an optimization problem that minimizes its local fitting error together with a penalty term that forces consensus:

$$x_i^{k+1} = \arg\min_{x_i} \; f_i(x_i) + \frac{\rho}{2}\left(x_i - z^k + u_i^k\right)^2$$

The global temperature z is computed as a weighted average of the local estimates and dual variables, where weights correspond to population densities:

$$z^{k+1} = \frac{1}{W}\sum_{i=1}^{N} w_i \cdot \left(x_i^{k+1} + u_i^k\right)$$

In this expression, w_i represents the local density at sensor i, and W is the sum of all weights, given by $W = \sum_{i=1}^{N} w_i$.
Each dual variable u_i is then updated to track the disagreement between the local estimate and the global estimate:

$$u_i^{k+1} = u_i^k + x_i^{k+1} - z^{k+1}$$

Synthetic data description

In this simulation, synthetic temperature data is generated using a strategically placed sensor network as described below. (refer to fig 43.)

- Total Sensors: 33
  - 1 center sensor located at coordinate $(0.5, 0.5)$.
  - 32 peripheral sensors arranged in 3 concentric rings:
    - Ring 1 (radius = 0.15): 8 sensors.
    - Ring 2 (radius = 0.30): 12 sensors.
    - Ring 3 (radius = 0.45): 12 sensors.
- Temperature Profile:
  - The center sensor records a higher baseline temperature (approximately 30°C), simulating the urban heat island effect.
  - Peripheral sensors record progressively lower temperatures (approximately 24°C at the edges).
  - The temperature at each sensor is computed as:

    $$T_i = T_{\text{center}} - (T_{\text{center}} - T_{\text{edge}}) \times \left(\frac{d_i}{d_{\max}}\right) + \epsilon_i$$

    where:
    - $T_{\text{center}} = 30°\text{C}$
    - $T_{\text{edge}} = 24°\text{C}$
    - d_i is the distance of sensor i from the center.
    - $d_{\max}$ is the maximum possible radius (0.45).
    - $\epsilon_i \sim \mathcal{N}(0, 0.5^2)$ is Gaussian noise.
- Sensor Weights:
  - Center sensor weight: 10.0
  - Peripheral sensor weights: 1.0
  - All weights normalized to sum to 1.

Optimization objective
Each sensor i aims to minimize its local cost based on its temperature reading t_i and weight w_i in equation below.

$$f_i(x_i) = w_i \cdot (x_i - t_i)^2$$

The consensus optimization problem is formulated as below.

$$\min_{\{x_i\}, z} \quad \sum_{i=1}^{N} f_i(x_i)$$

$$\text{subject to} \quad x_i = z, \quad \forall i$$

where:

- x_i: Local estimate of global temperature at sensor i.
- z: Global (consensus) temperature estimate.

The augmented Lagrangian for sensor i is:

$$\mathscr{L}_i(x_i, z, u_i) = w_i \cdot (x_i - t_i)^2 + \frac{\rho}{2}\|x_i - z + u_i\|^2$$

where:

- u_i: Dual variable (scaled Lagrange multiplier) for sensor i.
- ρ: Penalty parameter.

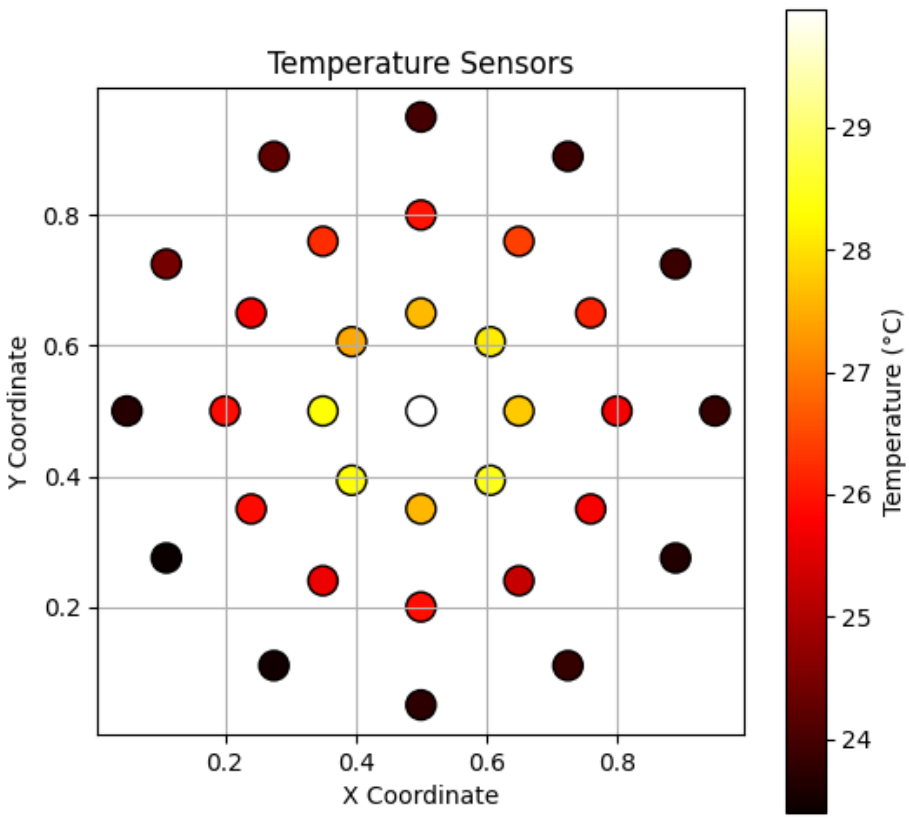


Figure 43: Sensor position and reading

Algorithm 27 CUDA-Accelerated ADMM for Consensus Temperature Estimation

Input: Sensor temperatures t_i, weights w_i, penalty ρ, learning rate η
Initialize local estimates $x_i^0 = t_i$, dual variables $u_i^0 = 0$, global estimate z^0
for iteration $k = 1$ to K **do**
  // Local Update (Parallelized on GPU)
  for all sensor i (CUDA thread) **do**
    for inner iteration $j = 1$ to LOCAL_ITERS **do**
      $x_i \leftarrow x_i - \eta\,(2w_i(x_i - t_i) + \rho(x_i - z + u_i))$
    end for
  end for
  // Global Variable Update (CPU)
  $z \leftarrow \frac{1}{N}\sum_{i=1}^{N}(x_i + u_i)$
  // Dual Variable Update (Parallelized on GPU)
  for all sensor i (CUDA thread) **do**
    $u_i \leftarrow u_i + (x_i - z)$
  end for
end for
Output: Global consensus temperature z

During training, the global consensus temperature steadily converges over 300 ADMM iterations, as shown in the training log plot. The final estimated temperature distribution is visualized as a masked heatmap overlaying the sensor network, highlighting the spatial variation from the warmer center to cooler edges.

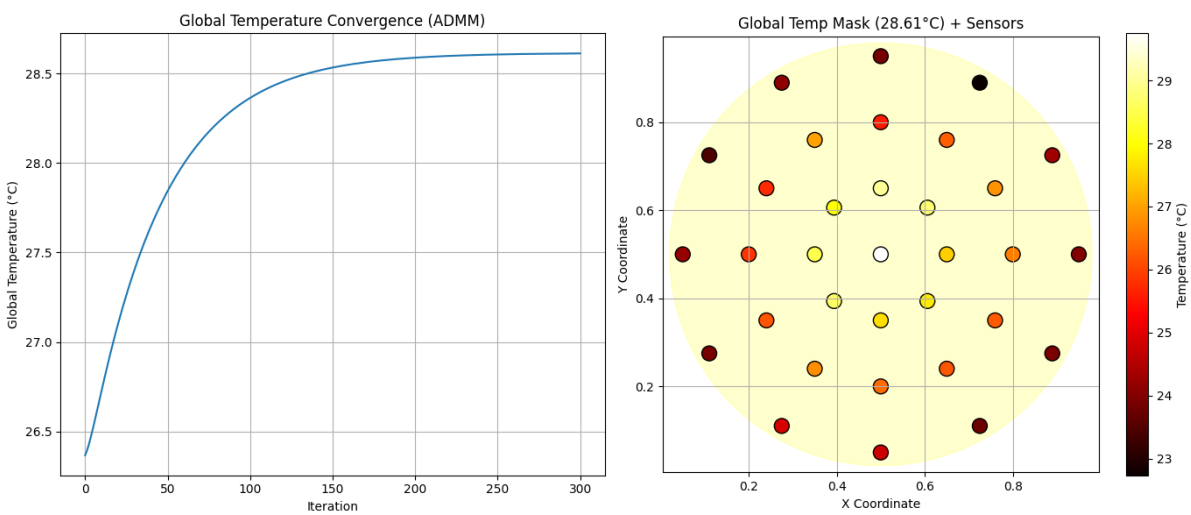


Figure 44: ADMM consensus - Training history and Result

3.2.2 Sharing global objective example

Consider a power grid with N independent power stations. Each station $i \in \{1, 2, \ldots, N\}$ decides its local power output x_i, subject to the following constraints:

$$x_i^{\min} \le x_i \le x_i^{\max}, \quad \forall i$$

and the global demand constraint:

$$\sum_{i=1}^{N} x_i = D$$

The total generation effort is modeled as a smooth quadratic function where $\gamma_i \ge 0$ are station-specific effort coefficients.

$$f(x) = \sum_{i=1}^{N} \gamma_i x_i^2$$

The operating cost is modeled as below.

$$g(z) = \sum_{i=1}^{N} \left(\alpha_i z_i^2 + \beta_i z_i\right)$$

The optimization problem is formulated as below.

$$\begin{aligned} \min_{x,z} \quad & f(x) + g(z) \\ \text{subject to} \quad & x_i = z_i, \quad \forall i \\ & x_i^{\min} \le x_i \le x_i^{\max}, \quad \forall i \\ & \sum_{i=1}^{N} x_i = D \end{aligned}$$

This splitting allows power generation and operating costs to be handled separately while maintaining consistency via $x_i = z_i$.

ADMM Update Steps

At iteration k, the updates proceed as follows:

1. **Local Power Generation Update (x-update):**

   Solve the following quadratic program:

   $$x^{(k+1)} = \arg\min_{x} \sum_{i=1}^{N} \gamma_i x_i^2 + \frac{\rho}{2} \sum_{i=1}^{N} \left(x_i - z_i^{(k)} + u_i^{(k)}\right)^2$$

   subject to:

   $$x_i^{\min} \le x_i \le x_i^{\max}, \quad \forall i$$

   and

   $$\sum_{i=1}^{N} x_i = D$$

   At each ADMM iteration, for each generator i, solve the following quadratic optimization problem:

   $$\min_{x_i} \ \gamma_i x_i^2 + \frac{\rho}{2} \left(x_i - z_i^{(k)} + u_i^{(k)}\right)^2$$

   Expand the squared term:

   $$\left(x_i - z_i^{(k)} + u_i^{(k)}\right)^2 = x_i^2 - 2x_i \left(z_i^{(k)} - u_i^{(k)}\right) + \left(z_i^{(k)} - u_i^{(k)}\right)^2$$

   Ignoring constants (independent of x_i), the objective simplifies to:

   $$\gamma_i x_i^2 + \frac{\rho}{2} x_i^2 - \rho x_i \left(z_i^{(k)} - u_i^{(k)}\right)$$

   Group the quadratic and linear terms:

   $$\left(\gamma_i + \frac{\rho}{2}\right) x_i^2 - \rho \left(z_i^{(k)} - u_i^{(k)}\right) x_i$$

   Take the derivative with respect to x_i and set to zero:

   $$2\left(\gamma_i + \frac{\rho}{2}\right) x_i - \rho \left(z_i^{(k)} - u_i^{(k)}\right) = 0$$

Solve for x_i:

$$x_i^{\text{unproj}} = \frac{\rho\left(z_i^{(k)} - u_i^{(k)}\right)}{2\gamma_i + \rho}$$

$$\boxed{x_i^{\text{unproj}} = \frac{\rho\left(z_i^{(k)} - u_i^{(k)}\right)}{2\gamma_i + \rho}}$$

After computing x_i^{unproj}, the following steps are applied:

- A global correction δ is computed to enforce the total demand constraint:

$$\sum_{i=1}^{N}\left(x_i^{\text{unproj}} + \delta\right) = D$$

- Each x_i is then projected onto its local box constraints:

$$x_i = \min\left(\max\left(x_i^{\text{unproj}} + \delta, x_i^{\min}\right), x_i^{\max}\right)$$

2. **Operating Cost Minimization (z-update):**

Each station solves locally:

$$z_i^{(k+1)} = \arg\min_{z_i}\ \alpha_i z_i^2 + \beta_i z_i + \frac{\rho}{2}\left(x_i^{(k+1)} - z_i + u_i^{(k)}\right)^2$$

At each iteration k, station i solves the following scalar optimization problem:

$$\min_{z_i} \quad \alpha_i z_i^2 + \beta_i z_i + \frac{\rho}{2}\left(x_i^{(k+1)} - z_i + u_i^{(k)}\right)^2$$

This is a quadratic minimization problem. Expanding the squared term:

$$\left(x_i^{(k+1)} - z_i + u_i^{(k)}\right)^2 = (A - z_i)^2$$

where:

$$A = x_i^{(k+1)} + u_i^{(k)}$$

Therefore:

$$(A - z_i)^2 = A^2 - 2Az_i + z_i^2$$

Substituting back into the objective:

$$\alpha_i z_i^2 + \beta_i z_i + \frac{\rho}{2}A^2 - \rho A z_i + \frac{\rho}{2}z_i^2$$

Grouping like terms:

$$\left(\alpha_i + \frac{\rho}{2}\right)z_i^2 + (\beta_i - \rho A)z_i + \frac{\rho}{2}A^2$$

Taking the derivative with respect to z_i:

$$\frac{d}{dz_i} = 2\left(\alpha_i + \frac{\rho}{2}\right)z_i + (\beta_i - \rho A)$$

Setting the derivative equal to zero:

$$2\left(\alpha_i + \frac{\rho}{2}\right) z_i + \beta_i - \rho A = 0$$

Solving for z_i:

$$z_i = \frac{\rho A - \beta_i}{2\alpha_i + \rho}$$

Finally, substituting back $A = x_i^{(k+1)} + u_i^{(k)}$, the closed-form solution is:

$$\boxed{z_i^{(k+1)} = \frac{\rho\left(x_i^{(k+1)} + u_i^{(k)}\right) - \beta_i}{2\alpha_i + \rho}}$$

The closed-form solution:

$$z_i^{(k+1)} = \frac{\rho\left(x_i^{(k+1)} + u_i^{(k)}\right) - \beta_i}{2\alpha_i + \rho}$$

3. **Dual Variable Update:**

$$u_i^{(k+1)} = u_i^{(k)} + x_i^{(k+1)} - z_i^{(k+1)}$$

Iterate until convergence (e.g., until primal and dual residuals are below tolerance).

Synthetic Data Generation, CUDA Acceleration, and Algorithm Overview

Data is generated to simulate a network of N independent power generation stations. Each station has unique characteristics defined by the following parameters. The coefficient γ_i represents the local quadratic generation cost and is sampled uniformly between 0.1 and 1.0. The coefficients α_i and β_i define the operating cost function, with α_i sampled uniformly between 0.01 and 0.1, and β_i sampled between 10 and 100. Each station also has minimum and maximum output limits, where $x_i^{\min}$ is sampled between 10 and 30, and $x_i^{\max}$ is sampled between 70 and 120. These generated parameters are stored as rows in a CSV file with columns: `gamma, alpha, beta, x_min, x_max`.

A parallel solution to the optimization problem is implemented using CUDA, where each power station independently solves its local subproblem using a dedicated CUDA thread. In each ADMM iteration, the algorithm proceeds as follows. First, the x-update kernel is launched where each thread computes the unconstrained power output for its station using the below.

$$x_i^{\text{unproj}} = \frac{\rho(z_i - u_i)}{2\gamma_i + \rho}$$

The resulting unconstrained outputs are aggregated on the CPU to compute a global correction term δ which enforces that the sum of all power outputs equals the total demand. Once δ is computed, a second CUDA kernel called the **projection kernel** is launched. In this step, each thread adjusts its local power output using:

$$x_i = \min\left(\max\left(x_i^{\text{unproj}} + \delta, x_i^{\min}\right), x_i^{\max}\right)$$

After the power outputs are projected, the z**-update kernel** is launched. Each CUDA thread independently solves the local operating cost minimization using:

$$z_i = \frac{\rho(x_i + u_i) - \beta_i}{2\alpha_i + \rho}$$

Finally, the dual variables u_i are updated sequentially on the CPU using:

$$u_i \leftarrow u_i + x_i - z_i$$

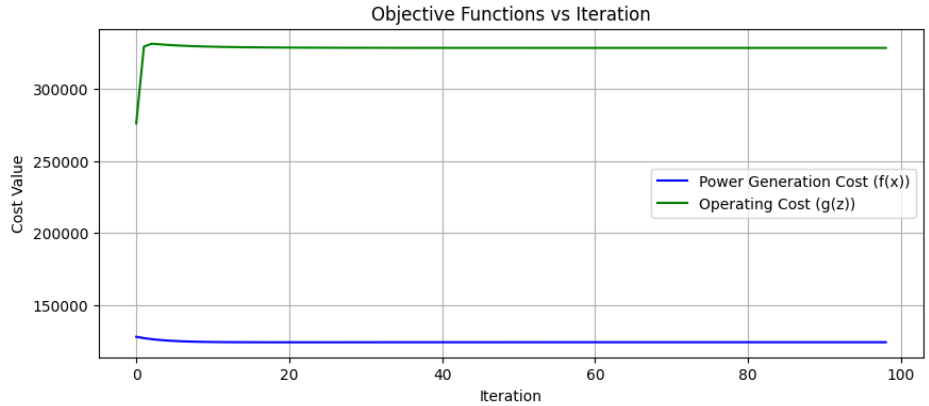


Figure 46: Convergence

The overall algorithm is summarized as follows:

CUDA-Accelerated ADMM Algorithm (Pseudocode)

```
Load CSV parameters into GPU memory.
Initialize dual variables u_i = 0 and shared variables z_i to mid-range values.
for each ADMM iteration until convergence do
    Launch CUDA kernel to compute x_i^unproj for all i.
    On CPU, compute demand correction δ and launch projection kernel.
    Launch CUDA kernel to update z_i for all i.
    On CPU, update u_i for all i.
    Evaluate residual and check stopping criterion.
end for
Report final power generation outputs and convergence metrics.
```

CUDA threads allow all local updates (x_i and z_i) to be performed in parallel across all power stations, while the CPU handles global coordination steps such as enforcing the demand constraint and updating the dual variables.

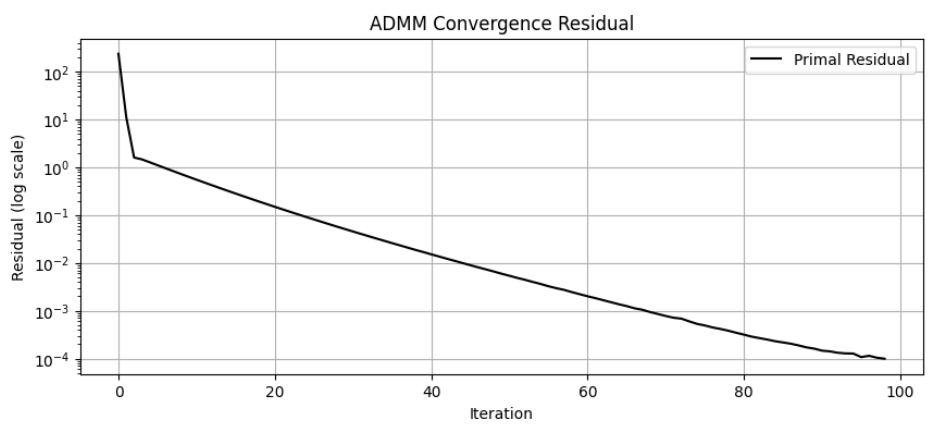


Figure 45: Training log

3.2.3 ATM withdrawl example

Each country $i \in \{1, 2, 3\}$ has local transaction data with the following properties.

- $X_i \in \mathbb{R}^{n_i \times d}$: Feature matrix of ATM-related data (e.g., day of week, holiday indicator, past withdrawals).

- $y_i \in \mathbb{R}^{n_i}$: Daily ATM cash withdrawal amounts.

Each country trains a local model $w_i \in \mathbb{R}^d$. To build a shared global model $z \in \mathbb{R}^d$, the following distributed optimization problem is solved:

$$\begin{aligned} \min_{w_1,w_2,w_3,z} \quad & \sum_{i=1}^{3} \frac{1}{2} \|X_i w_i - y_i\|_2^2 \\ \text{subject to} \quad & w_i - z = 0, \quad \forall i \in \{1,2,3\} \end{aligned}$$

Here:

- The objective minimizes the sum of local squared regression losses.
- The consensus constraints $w_i = z$ ensure all local models agree on the global model.

Data

Each country exhibits distinct cash withdrawal behaviors, reflecting regional transaction habits:

- **Country A (Weekend Peaks):**
  Withdrawal amounts increase significantly on weekends (Saturday and Sunday). This pattern reflects customers withdrawing larger amounts before the weekend.
- **Country B (Month-End Salary Spikes):**
  Withdrawals sharply increase during the last few days of each month (typically from day 25 onwards), corresponding to salary disbursement periods when customers withdraw large sums.
- **Country C (Holiday Drops):**
  Withdrawal volumes drop noticeably on public holidays, as ATM usage decreases during non-working days.

Data Features

For each ATM and each day, the following features are available:

- Day of the week (0 = Monday, 6 = Sunday).
- Day of the month (1 to 31).
- Holiday indicator (1 = holiday, 0 = normal day).
- Previous day's withdrawal amount.
- Target variable: Current day's withdrawal amount (to be predicted).

These patterns as shown in Fig 43. result in non-identical and non-independent data distributions across countries, motivating the need for a distributed learning approach to collaboratively predict cash demands without sharing sensitive transaction data.

The algorithm (No. 27) solves a distributed regression problem using ADMM with CUDA acceleration. Each country independently updates its local model weights using gradient descent, run in parallel on GPUs for efficiency. These updates incorporate both the prediction error and a penalty term that enforces alignment with a shared global model. After local updates, all countries synchronize by averaging their adjusted weights to update the global model, promoting consensus. This cycle of local optimization and global averaging repeats until the global model stabilizes, ensuring all countries' models converge together. In fig 44, the convergence is plotted per country.
If we plot the data back based on weight, only a resemblance of the pattern is retrieved.

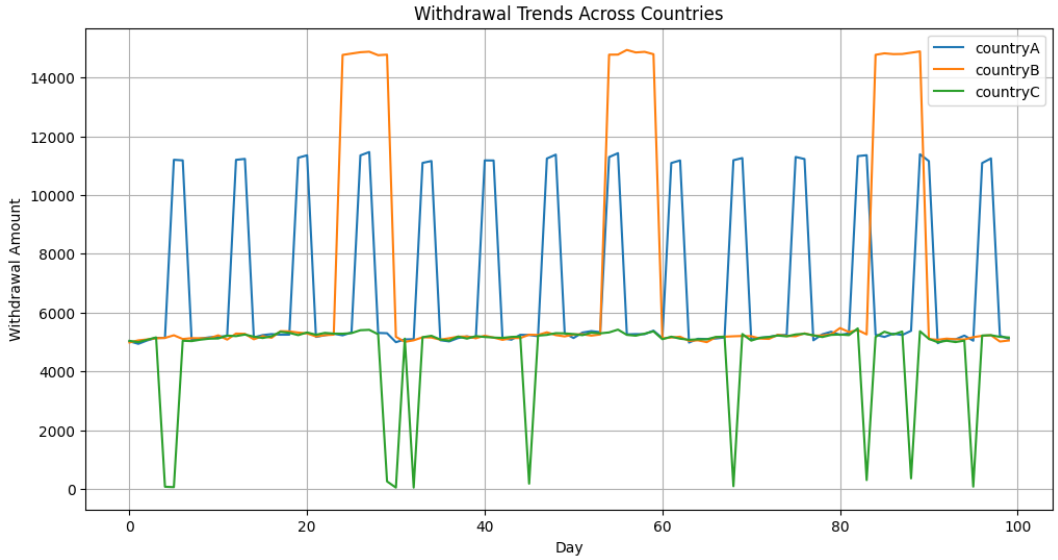


Figure 47: Data input pattern

Algorithm 28 ADMM-Based Distributed Training (CUDA Accelerated)

Input: Training data $\{X_c, y_c\}$ for $c = 1, ..., C$, penalty parameter ρ, learning rate lr, max iterations T, local steps K, tolerance ϵ
Initialize: $w_c \leftarrow 0$, $\lambda_c \leftarrow 0$, $z \leftarrow 0$ for all c
Normalize features and labels
for $t = 1$ to T **do**
  for all countries $c = 1$ to C **in parallel on GPU do**
    for $k = 1$ to K **do**
      Compute gradient:

$$\nabla f_c(w_c) = \frac{1}{n} X_c^\top (X_c w_c - y_c) + \rho(w_c - z + \lambda_c)$$

      Update weights:

$$w_c \leftarrow w_c - lr \times \nabla f_c(w_c)$$

    end for
  end for
  Update shared variable:

$$z \leftarrow \frac{1}{C} \sum_{c=1}^{C} (w_c + \lambda_c)$$

  for $c = 1$ to C **do**
    Dual update:

$$\lambda_c \leftarrow \lambda_c + w_c - z$$

  end for
  Compute MSE for each country and record
  Check convergence:
  if $\|z^{(t)} - z^{(t-1)}\| < \epsilon$ **then**
    break
  end if
end for
Output: Final weights $\{w_c\}$ and MSE history

In distributed optimization, a consensus rule ($w_i = z$) ensures that multiple agents (in this case, different countries) collaboratively agree on a shared solution by averaging or synchronizing certain param-

eters—in your setup, the model weights. Instead of allowing each country to independently learn its weights, the consensus step computes a global variable (often called z) as the average of local weights plus adjustments via dual variables (λ). This shared z value acts as the target towards which each country's local weights are gradually pulled, enforcing agreement across all participants. The expectation from applying such a rule is that while each country learns from its own data, the final model parameters converge to a common consensus value that balances individual learning with collective agreement, preventing divergence and promoting fairness across all agents.

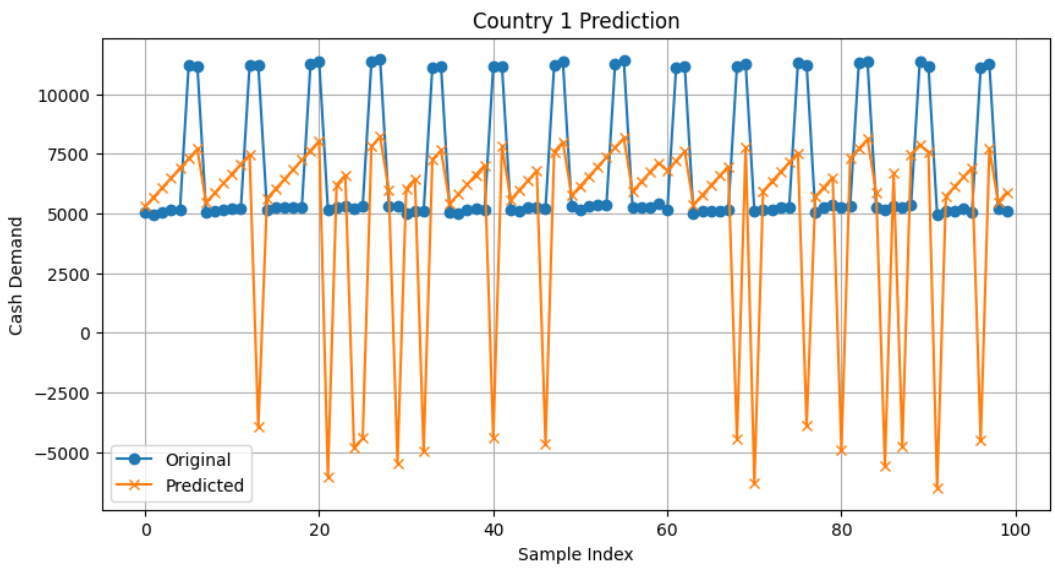


Figure 48: Comparison pred vs orig - country 1

3.2.4 Linear Regression Consensus

Consider a distributed linear regression problem where the goal is to estimate a weight vector $w \in \mathbb{R}^d$ that models a linear relationship between input data $X \in \mathbb{R}^{n \times d}$ and observations $y \in \mathbb{R}^n$, subject to a non-negativity constraint on w. The optimization problem is formulated as

$$\begin{aligned} \min_{w,z} \quad & f(w) + g(z) \\ \text{subject to} \quad & w - z = 0 \end{aligned}$$

where

$$f(w) = \frac{1}{2}\|Xw - y\|_2^2$$

$$g(z) = \begin{cases} 0 & \text{if } z \geq 0 \\ \infty & \text{otherwise} \end{cases}$$

The augmented Lagrangian in unscaled form is written as

$$\mathscr{L}_\rho(w, z, \lambda) = \frac{1}{2}\|Xw - y\|_2^2 + g(z) + \lambda^T(w - z) + \frac{\rho}{2}\|w - z\|_2^2$$

The ADMM update steps at iteration k are as follows.

1. **w-update**

   Solve the quadratic minimization

$$w^{(k+1)} = \arg\min_w \frac{1}{2}\|Xw - y\|_2^2 + \lambda^{(k)T}(w - z^{(k)}) + \frac{\rho}{2}\|w - z^{(k)}\|_2^2$$

Expanding the terms:

$$\frac{1}{2}w^T X^T X w - y^T X w + \lambda^{(k)T} w - \lambda^{(k)T} z^{(k)} + \frac{\rho}{2} w^T w - \rho z^{(k)T} w + \text{const}$$

Collecting quadratic and linear terms:

$$\frac{1}{2}w^T (X^T X + \rho I) w + \left(X^T y + \rho z^{(k)} - \lambda^{(k)}\right)^T w + \text{const}$$

Taking the gradient with respect to w and setting to zero:

$$(X^T X + \rho I) w^{(k+1)} = X^T y + \rho z^{(k)} - \lambda^{(k)}$$

Hence, the closed-form solution is:

$$w^{(k+1)} = (X^T X + \rho I)^{-1} \left(X^T y + \rho z^{(k)} - \lambda^{(k)}\right)$$

2. **z-update**

$$z^{(k+1)} = \max\left(0,\ w^{(k+1)} + \frac{1}{\rho}\lambda^{(k)}\right)$$

3. **Dual variable update**

$$\lambda^{(k+1)} = \lambda^{(k)} + \rho \left(w^{(k+1)} - z^{(k+1)}\right)$$

The algorithm alternates between solving a regularized least-squares system, projecting onto the non-negative orthant, and updating the Lagrange multipliers.

In this system, synthetic datasets are generated independently for each node to simulate a distributed environment. Each dataset consists of two input features and corresponding output values, following a linear model with added noise. These datasets are saved as CSV files, representing data partitions across nodes. During training, each CUDA thread uses data from one CSV file corresponding to a node and processes only its own data. This design effectively emulates a distributed data setup, where different nodes hold distinct local datasets. CUDA threads then perform local optimization steps concurrently, enabling parallelized training across nodes.

Each CUDA thread is responsible for computing gradients based on its local data, updating local model weights using gradient descent combined with ADMM penalty adjustments. After local updates, the CPU aggregates weight updates from all threads to compute a global consensus model, which is broadcast back to GPU threads. This CPU-mediated step ensures consensus across nodes. Dual variables are also updated on the CPU to enforce agreement between local models and the global consensus. By alternating between parallel local updates on the GPU and centralized synchronization on the CPU, the system achieves efficient distributed optimization through the ADMM framework.

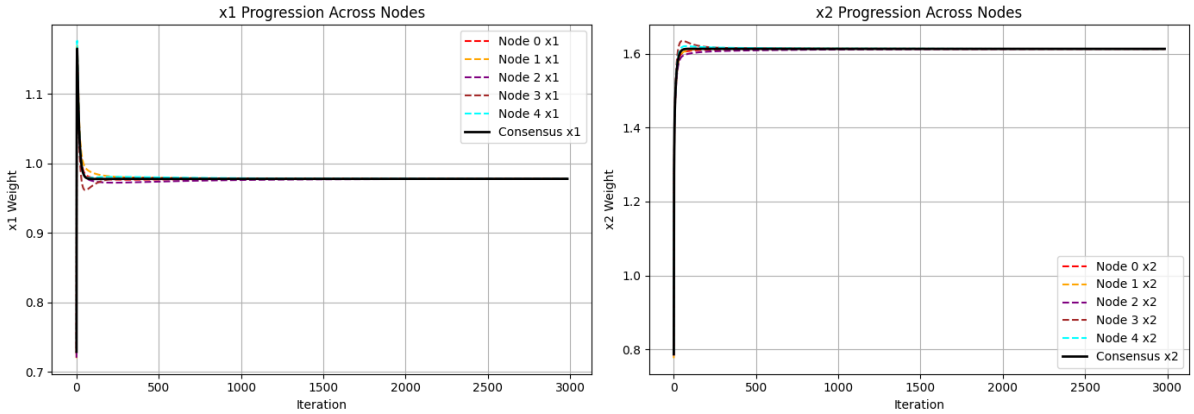


Figure 49: Distributed optimization progress across nodes(theads).

Algorithm 29 CUDA-CPU Hybrid ADMM Consensus Training

```
Generate synthetic CSV datasets for each node using a linear model with noise
Initialize local weights w_i, consensus weights z, and dual variables u_i
Load each node's CSV data into GPU memory
for each iteration until convergence do
    Local Updates (CUDA)
    for each CUDA thread (one per node) do
        Read node-specific data
        Compute local gradients using node's data
        Update local weights using gradient descent and ADMM penalties
    end for
    Global Consensus Update (CPU)
    Aggregate local weights and dual variables to compute consensus weights z
    Update dual variables for each node based on deviation from consensus
    Broadcast consensus weights and updated dual variables back to GPU
end for
Output final consensus model weights
```

3.2.5 Logistic regression

In this distributed logistic regression setup, N organizations each hold private data (A_i, b_i), where $A_i \in \mathbb{R}^{m_i \times n}$ and $b_i \in \{0,1\}^{m_i}$. To address differing currencies, each node applies a known, fixed scaling vector $w_i \in \mathbb{R}^n$ (based on exchange rates), transforming local features:

$$\tilde{A}_i = A_i \cdot \mathrm{diag}(w_i)$$

Each node maintains a local logistic regression model $x_i \in \mathbb{R}^n$ and collaborates to learn a global model $z \in \mathbb{R}^n$, regularized towards a reference model W.

Optimization Problem.
Each node solves the equations below.

$$f_i(x_i) = \frac{1}{m_i} \sum_{j=1}^{m_i} \log\left(1 + e^{-b_{ij} \cdot \tilde{A}_{ij}^\top x_i}\right)$$

$$g(z) = \frac{1}{2} \|z - W\|_2^2$$

The distributed consensus problem is as below.

$$\min_{\{x_i\},z} \quad \sum_{i=1}^{N} f_i(x_i) + \frac{1}{2}\|z - W\|_2^2$$

$$\text{s.t.} \quad x_i = z, \quad \forall i = 1, \ldots, N$$

Augmented Lagrangian.
Using dual variables λ_i, the augmented Lagrangian is:

$$\mathscr{L}_\rho(\{x_i\}, z, \{\lambda_i\}) = \sum_{i=1}^{N} \left[f_i(x_i) + \lambda_i^\top (x_i - z) + \frac{\rho}{2}\|x_i - z\|_2^2 \right] + \frac{1}{2}\|z - W\|_2^2$$

Derivation of ADMM Updates

1. Local Update: x_i^{k+1} Each node solves the following optimization problem:

$$x_i^{k+1} = \arg\min_{x_i} \left[f_i(x_i) + \lambda_i^{k\top}(x_i - z^k) + \frac{\rho}{2}\|x_i - z^k\|_2^2 \right]$$

Taking gradient with respect to x_i and setting to zero:

$$\nabla f_i(x_i) + \lambda_i^k + \rho(x_i - z^k) = 0$$

Solving for x_i:

$$x_i^{k+1} = z^k - \frac{1}{\rho}\left(\nabla f_i(x_i) + \lambda_i^k\right)$$

In case $f_i(x_i)$ is quadratic, say:

$$f_i(x_i) = \frac{1}{2} x_i^\top Q_i x_i + c_i^\top x_i$$

then:

$$\nabla f_i(x_i) = Q_i x_i + c_i$$

and solving:

$$(Q_i + \rho I) x_i^{k+1} = \rho z^k - \lambda_i^k - c_i$$

$$x_i^{k+1} = (Q_i + \rho I)^{-1}\left(\rho z^k - \lambda_i^k - c_i\right)$$

2. Global Update: z^{k+1} Solve:

$$z^{k+1} = \arg\min_z \left[\frac{1}{2}\|z - W\|_2^2 + \frac{\rho}{2}\sum_{i=1}^{N} \|x_i^{k+1} + \frac{1}{\rho}\lambda_i^k - z\|_2^2 \right]$$

Taking gradient with respect to z and setting to zero:

$$(z - W) + \rho \sum_{i=1}^{N} \left(z - x_i^{k+1} - \frac{1}{\rho}\lambda_i^k \right) = 0$$

Simplify:

$$z \cdot (1 + N\rho) = W + \rho \sum_{i=1}^{N} \left(x_i^{k+1} + \frac{1}{\rho}\lambda_i^k \right)$$

Solve:

$$z^{k+1} = \frac{W + \rho \sum_{i=1}^{N} \left(x_i^{k+1} + \frac{1}{\rho}\lambda_i^k \right)}{1 + N\rho}$$

3. Dual Update: λ_i^{k+1} Simple additive update:

$$\lambda_i^{k+1} = \lambda_i^k + \rho\left(x_i^{k+1} - z^{k+1}\right)$$

The penalty parameter ρ influences convergence behavior, balancing constraint satisfaction and optimization progress. The algorithm alternates between local minimization, global averaging, and dual variable adjustment until convergence.

Synthetic data used for test

Each of the 5 nodes generates 1,000 samples with 5 or 10 features simulating financial transaction data. Features are scaled using node-specific exchange rates (randomly between 0.5 and 2.0) to mimic different currencies or units. A shared global weight vector defines the decision boundary. Labels are generated using a sigmoid function applied to scaled features, sharpened by multiplying logits by 10, then sampled as binary outcomes. This setup creates 5,000 distributed samples reflecting local scaling but shared global patterns.

Cuda usage

CUDA helps in this problem by parallelizing the computation of gradients during local logistic regression updates. Instead of processing each transaction sequentially, CUDA assigns each sample (transaction) to a separate thread. These threads compute their contributions to the gradient simultaneously, using GPU cores for massive parallelism. This significantly accelerates the per-node optimization step, especially when processing thousands of transactions per node. Atomic operations ensure thread-safe accumulation of gradients across threads. Thus, CUDA reduces local computation time in each ADMM iteration, making distributed training more efficient.

Algorithm 30 CUDA-Accelerated Local Gradient Update (Node i)

```
Input: Local features A_i ∈ ℝ^{m×n}, labels b_i ∈ {0,1}^m, model weights x_i ∈ ℝ^n, consensus variable z ∈ ℝ^n,
       dual variable λ_i ∈ ℝ^n
Initialize: Allocate device memory and zero gradient accumulator
for each CUDA thread (sample index j) do
    Compute prediction: y_j = σ(A_j x_i)
    Compute residual: r_j = y_j − b_j
    Compute partial gradient: ∇_j = A_j^T · r_j
    Atomically add ∇_j/m to shared gradient accumulator
end for
Update weights: x_i ← x_i − η(gradient + λ_i + ρ(x_i − z))
```

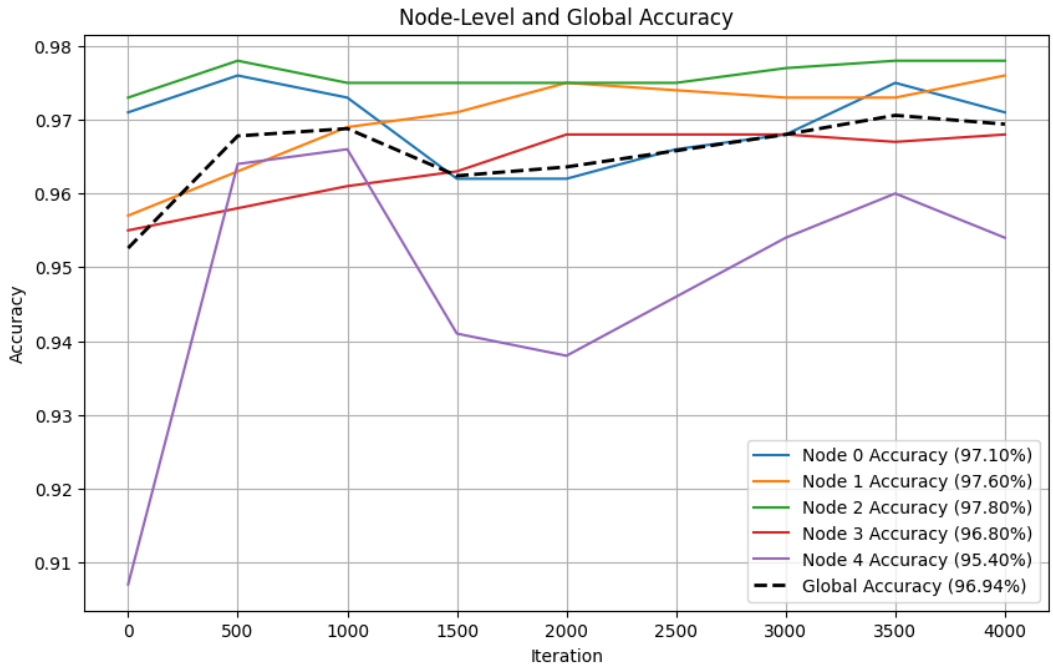


Figure 50: Distributed Training across nodes

Note: As the features, scale the method above will suffer loss of accuracy and will need better acceleration methods.

3.3 Representation as graph

The network topology of a distributed system can be envisioned as a graph where the nodes are connected to other nodes using a specified structure.

Table 10: Consensus Algorithms by Graph Type and Weights

Graph Type	**Weights**	**Method Name**	**Update Formula**
Undirected	No Weights	Average Consensus	$x_i^{k+1} = x_i^k + \epsilon \sum_{j \in \mathcal{N}_i} (x_j^k - x_i^k)$
Undirected	No Weights	Sign-Based Consensus	$x_i^{k+1} = x_i^k - \epsilon \sum_{j \in \mathcal{N}_i} \text{sign}(x_i^k - x_j^k)$
Undirected	With Weights	Weighted Average Consensus	$x_i^{k+1} = x_i^k + \epsilon \sum_{j \in \mathcal{N}_i} w_{ij} (x_j^k - x_i^k)$
Undirected	With Weights	Metropolis-Hastings Consensus	$x_i^{k+1} = \sum_{j \in \mathcal{N}_i \cup \{i\}} w_{ij} x_j^k$
Directed	No Weights	Directed Consensus	$x_i^{k+1} = x_i^k + \epsilon \sum_{j \in \mathcal{N}_i^{\text{in}}} (x_j^k - x_i^k)$
Directed	No Weights	Ratio Consensus	$x_i^{k+1} = \frac{\sum_{j \in \mathcal{N}_i^{\text{in}}} x_j^k}{\lvert \mathcal{N}_i^{\text{in}} \rvert}$
Directed	With Weights	Push-Sum Consensus	$x_i^k = \frac{y_i^k}{z_i^k}$, update y_i, z_i via row-stochastic matrix
Directed	With Weights	Fast Consensus (Accelerated)	Uses Nesterov-type updates with row-stochastic

4 Appendix

Gradient Flow	$\dot{x}(t) = -\nabla f(x(t))$: continuous-time dynamics that decrease f over time.
Gradient Descent	$x_{k+1} = x_k - \eta \nabla f(x_k)$: discrete step to minimize a function using gradient.
Gradient Ascent	$x_{k+1} = x_k + \eta \nabla f(x_k)$: used to maximize functions.
Hessian Matrix	$\nabla^2 f(x)$: matrix of second-order partial derivatives, describes curvature.
Second-Order Approximation	$f(x+\delta) \approx f(x) + \nabla f^\top \delta + \frac{1}{2}\delta^\top \nabla^2 f \delta$: local quadratic model.
Differential (Scalar)	$\mathrm{d}f = \nabla f(x)^\top \mathrm{d}x$: linear approximation of change in f.
Integral (1D)	$\int_a^b f(x)\,dx$: area under the curve from a to b.
Integral (Multivariate)	$\int_\Omega f(x)\,dx$: integrates f over domain $\Omega \subseteq \mathbb{R}^n$.
Chain Rule (Scalar)	$\frac{d}{dx} f(g(x)) = f'(g(x)) \cdot g'(x)$
Chain Rule (Vector)	$\nabla f(g(x)) = J_g(x)^\top \nabla f$: uses Jacobian of $g(x)$.

Method	Nature of Convergence	Best Suited For	Update Rule / Notes	Comments
Vanilla GD	Infinite time (asymptotic)	• Smooth • Well-conditioned convex	$x_{k+1} = x_k - \eta \nabla f(x_k)$ Works on $f : \mathbb{R}^n \to \mathbb{R}$	Simple and general; slow near flat regions.
Finite-Time GD	Finite time (init-dependent)	• Strongly convex • Gradient-dominated	$x_{k+1} = x_k - \theta \nabla f(x_k) \cdot \lVert \nabla f(x_k) \rVert^{-\frac{q-2}{q-1}}$ Assumes $\nabla f \neq 0$	Can diverge if gradient vanishes; sensitive to scaling.
Fixed-Time GD (FxTS)	Fixed time (any init)	• Gradient-dominated • Quasi-convex	$x_{k+1} = x_k - \theta \nabla f(x_k) \left[\lVert \nabla f \rVert^{-\frac{p-2}{p-1}} + f(x)^{-\frac{q-1}{p-1}} \right]$	Independent of init; fast; may overshoot without tuning.
Newton FxTS	Fixed time with curvature	• Twice differentiable • Ill-conditioned	$x_{k+1} = x_k - \theta H^{-1} \left[\frac{\nabla f}{\lVert \nabla f \rVert^{\frac{p-2}{p-1}}} + \frac{\nabla f}{\lVert \nabla f \rVert^{\frac{q-2}{q-1}}} \right]$	Very fast and robust; needs Hessian inverse.
Prescribed-Time GD	Finite time via decay	• Smooth unconstrained	$x_{k+1} = x_k - \frac{\eta \nabla f(x_k)}{t_k^r} \quad (r = 2)$	Can stall if decay too steep. Controlled scheduling.
Projected FxTS	Fixed time + constraints	• Convex feasible sets	$x_{k+1} = \Pi_{\mathscr{C}} \left[x_k - \theta \nabla f(x_k) \left(\lVert \nabla f \rVert^{-\alpha} + f^{-\beta} \right) \right]$	Maintains feasibility; useful in constrained optimization.
Proximal FxTS	Fixed time + non-smooth	• Composite $f = g + h$ • Sparsity, L1-regularized ML	$x_{k+1} = \mathrm{prox}_{\eta h} \left(x_k - \eta \nabla g(x_k) \cdot (\ldots) \right)$	Efficient for Lasso, elastic net, TV-regularized problems.

Table 11: Comparison of Gradient Descent Variants with Fixed-Time and Finite-Time Properties (thin cell borders)