Matthew Poncini, Daniella Arteaga Mendoza
J.B. Speed School of Engineering
University of Louisville

19 September 2025

—

Project 1: Gradient Descent

Introduction

The purpose of this project is to explore four different optimization methodologies using gradient descent. A **gradient vector** represents the slope of a function at a specific point. In this project, we will be applying gradient descent techniques to 3-Dimensional functions and each gradient can be described as:

$$\nabla f(x, y) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right)$$

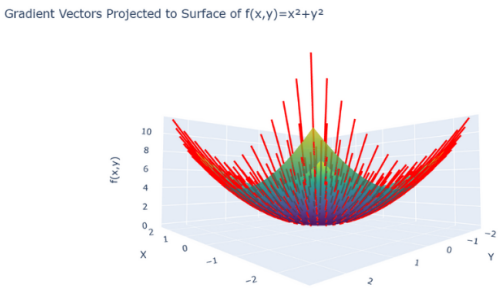
Gradient vectors point in the direction of the steepest increase/decrease of the function. In 3-dimensional space, gradient vectors can be seen pointing in a direction perpendicular to a contour.

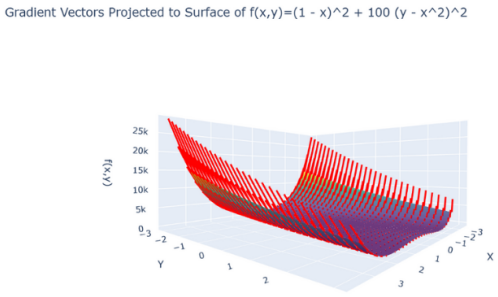
An optimization algorithm using gradient descent will start at a point on the function, and select a new point in the direction of the gradient vector at that specified. While the direction is determined by the gradient vector, the distance of the new point will be determined by the **learning rate**. This process will repeat until one of the following conditions has been met: 1) the gradient vector is close to zero, 2) the change of the function value (**loss**) is close to zero, or 3) the max number of iterations has been reached. If the first or second conditions are met, the algorithm has likely found a local extrema. In the case of condition 3, the algorithm was to slow in it's attempt in finding the local extrema either because the learning rate was too small, the starting point was to far from the extrema, there is no extrema, or the learning rate was too big and passed the extrema causing **oscillation**.

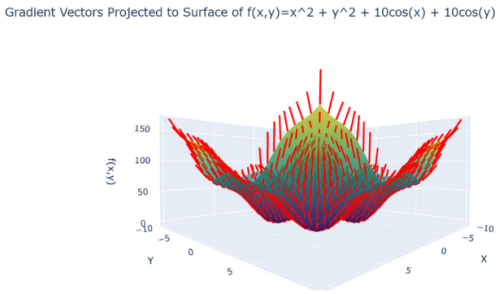
The methodology section will go into depth about the four algorithms used for optimization:

- Vanilla Gradient Descent
- Newton's Method
- AdaGrad
- Adam

Each algorithm will be applied to the following graphs at various points and learning rates.







Methodology: Vanilla Gradient Descent

The vanilla gradient descent algorithm begins at an initial point and then takes iterative steps in the direction of a gradient vector, recording each landing point along it's path. The algorithm continues carving it's path until convergence or the maximum number of points are added to it's path. The code used to implement this algorithm comes from the Gradient Descent Project Scaffold Colab Notebook provided to the class, with a few alterations made for the purpose of this project. This algorithm was implemented in Python. Listed below are the parameters and their purpose in the algorithm:

Parameter	Type	Description
`step_size`	float	Learning rate; a multiplier of the gradient vector, determining the magnitude of each iterative step
`grad_fn`	Callable	The gradient function of $f(x, y)$
`x0`	tuple[float, float]	(x_0, y_0); the starting point of the path
`max_iters`	int	Maximum number of iterations allowed (default 1,000,000).
`tol`	float	Stopping tolerance; if $\|x_{\text{next}} - x\|$ is smaller than this threshold, the algorithm stops and adds the final point to the path
`label`	str	A string label to tag results in global dictionaries (`iterations`, `output`, `starting_point`, `learning_rate`).

Gradient Descent Pseudocode

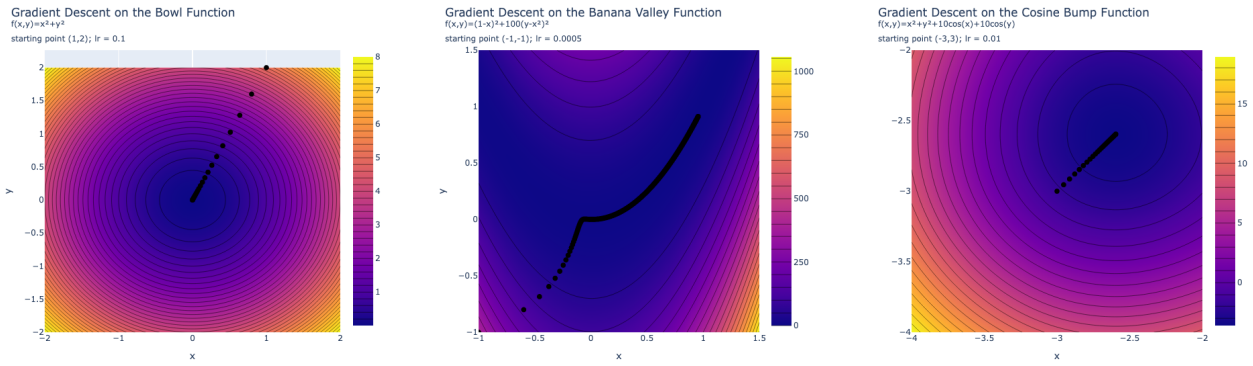
Input: step_size α, gradient function $grad\_fn$, initial point x_0, max iterations N, tolerance ε, label
Output: Path of iterates
$x = x_0$ path $= [x]$
for $k = 1$ **to** N **do**
  $x_{\text{next}} = x - \alpha \cdot grad\_fn(x)$, append x_{next} to path
  if $\|x_{next} - x\| < \varepsilon$ **then**
    print "Converged at x_{next} in k iterations", **break**
  end
  $x = x_{\text{next}}$
end
return path

The algorithm initializes a list, `path`, with the initial point, then begins iterating up to `max_iters`. At each step, the gradient is computed using `grad_fn`, and the next point is updated by:

$$x_{\text{next}} = x - \alpha \cdot grad\_fn(x)$$

This point is appended to `path`. If the distance $\|x_{\text{next}} - x\|$ is smaller than `tol`, the algorithm converges early, prints the result, and updates global dictionaries. Otherwise, x is set to x_{next} and the process continues.

Vanilla Gradient Descent Results



The vanilla gradient descent algorithm was successful in converging for the convex bowl, unable to converge in the banana valley, and was inconsistent in cosines bump. In the convex bowl, the learning rate was able to speed up convergence. This was not the case in cosines bump, where larger learning rates led to oscillation. The algorithm had trouble in the banana valley function. The algorithm is too dependent on favorable starting points and learning rates in banana valley.

Methodology: Newton's Method

Newton's method is a second-order optimization algorithm that uses both first and second derivatives to find the optimal step size and direction. Unlike vanilla gradient descent which uses a fixed learning rate, Newton's method computes the step size using the inverse of the Hessian matrix (second derivatives). This approach can lead to faster convergence, especially near the optimum, but requires computing and inverting the Hessian matrix at each iteration.

Parameter	Type	Description
`grad_fn`	Callable	The gradient function of $f(x, y)$
`hess_fn`	Callable	The Hessian function returning the matrix of second partial derivatives
`x0`	tuple[float, float]	(x_0, y_0); the starting point of the path
`max_iters`	int	Maximum number of iterations allowed (default 1,000,000).
`tol`	float	Stopping tolerance; if $\|x_{\text{next}} - x\|$ is smaller than this threshold, the algorithm stops
`label`	str	A string label to tag results in global dictionaries

Newton's Method Pseudocode

Input: gradient function $grad\_fn$, Hessian function $hess\_fn$, initial point x_0, max iterations N, tolerance ε, label
Output: Path of iterates
$x = x_0$ path $= [x]$
for $k = 1$ **to** N **do**
  $g = grad\_fn(x)$ $H = hess\_fn(x)$ $x_{\text{next}} = x - H^{-1} \cdot g$
  append x_{next} to path
  if $\|x_{next} - x\| < \varepsilon$ **then**
    print "Converged at x_{next} in k iterations" **break**
  end
  $x = x_{\text{next}}$
end
return path

Newton's method computes both the gradient g and Hessian matrix H at each iteration. The next point is updated by:

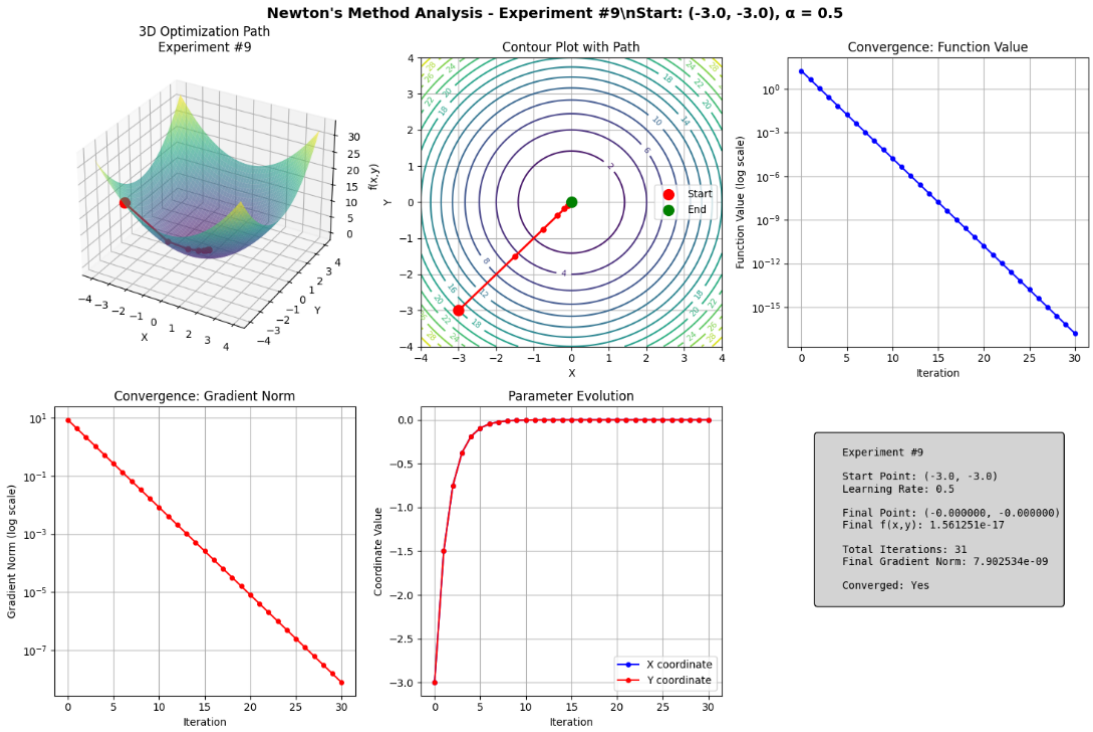
$$x_{\text{next}} = x - H^{-1} \cdot g$$

where H^{-1} is the inverse of the Hessian matrix. This eliminates the need for a learning rate parameter since the optimal step size is computed from the second-order information. The algorithm converges when the step size becomes smaller than the tolerance threshold.

Newton's Method Results

We would expect Newton's method to excel in the convex bowl since the Hessian is constant. Newton's method was effective when the learning rate was 0.5 and 0.1. When the learning rate was smaller at 0.01, Newton's Method began to take significantly more iterations.

Function	Start Point	Learning Rate	Iterations	Output (x, y)
Convex Bowl	(1,2)	0.01	1982	(0, 0)
Convex Bowl	(1,2)	0.10	190	(0, 0)
Convex Bowl	(1,2)	0.50	29	(0, 0)



Methodology: AdaGrad

AdaGrad (Adaptive Gradient Algorithm) is an adaptive learning rate method that adjusts the learning rate for each parameter individually based on the historical gradients. Parameters that receive large gradients will have their learning rates reduced more rapidly, while parameters with small gradients will maintain larger learning rates. This helps with sparse data and prevents oscillation in steep dimensions.

Parameter	Type	Description
`step_size`	float	Initial learning rate; will be adapted during optimization
`grad_fn`	Callable	The gradient function of $f(x, y)$
`x0`	tuple[float, float]	(x_0, y_0); the starting point of the path
`epsilon`	float	Small constant added to denominator for numerical stability (default 10^{-8})
`max_iters`	int	Maximum number of iterations allowed (default 1,000,000).
`tol`	float	Stopping tolerance; if $\|x_{\text{next}} - x\|$ is smaller than this threshold, the algorithm stops
`label`	str	A string label to tag results in global dictionaries

The update for parameter θ at iteration t is:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} g_t$$

where: η is the parameter step_size, g_t is `grad_fn`, G_t is the accumulated squared gradients, and ϵ = small constant.

AdaGrad Pseudocode

Input: step_size α, gradient function $grad\_fn$, initial point x_0, epsilon ϵ, max iterations N, tolerance ε, label
Output: Path of iterates
$x = x_0$ $G = 0$ `// accumulated squared gradients`
path $= [x]$
for $k = 1$ **to** N **do**
  $g = grad\_fn(x)$ $G = G + g \odot g$ `// element-wise square and accumulate`
  $x_{\text{next}} = x - \frac{\alpha}{\sqrt{G+\epsilon}} \odot g$ append x_{next} to path
  if $\|x_{next} - x\| < \varepsilon$ **then**
    print "Converged at x_{next} in k iterations" **break**
  end
  $x = x_{\text{next}}$
end
return path

AdaGrad maintains an accumulator G of squared gradients. At each iteration, the gradient is computed and its square is added to G. The update rule becomes:

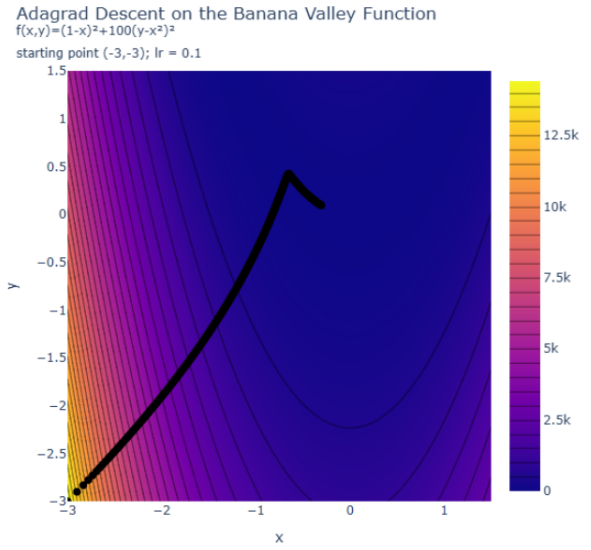
$$x_{\text{next}} = x - \frac{\alpha}{\sqrt{G+\epsilon}} \odot g \qquad \text{[H]}$$

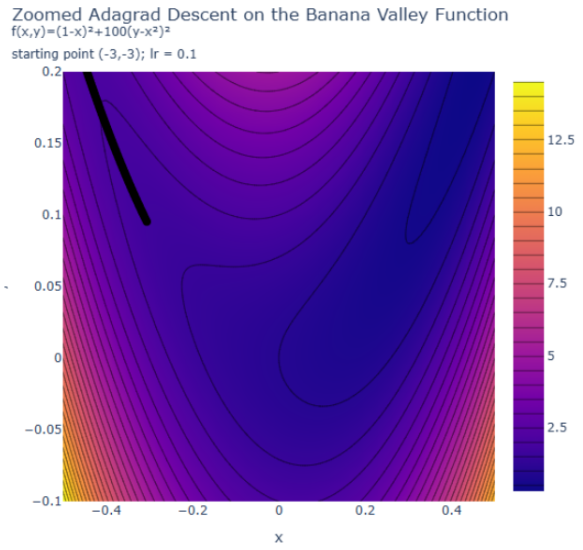
where $\odot$ denotes element-wise operations. The adaptive learning rate $\frac{\alpha}{\sqrt{G+\epsilon}}$ decreases over time, with dimensions having larger accumulated gradients receiving smaller updates.

AdaGrad Results

Performance slows dramatically when the learning rate is set too small for Adagrad. This is because the algorithm continually decreases the effective learning rate as squared gradients accumulate. Progress eventually plateaus, leaving the optimizer unable to make meaningful updates. At larger values, such as a learning rate of 0.5, AdaGrad achieves the best balance, converging quickly and accurately to the minimum without excessive iteration.

In the Banana Valley, AdaGrad did not converge within the 100,000 step limit at the starting point of (-3, -3) and a learing rate of 0.1. It reached (0.3056,0.0953) at the 10,000th iteration. This outcome highlights a key weaknesses of AdaGrad. As the accumulated squared gradients grow, the effective learning rate becomes so small that the optimizer can no longer make meaningful progress. This exposed in the Banana Valley where the minima appears to be approaching, but is still a distance away.





Methodology: Adam

Adam (Adaptive Moment Estimation) combines the advantages of AdaGrad and momentum methods. It maintains exponentially decaying averages of both past gradients (first moment) and past squared gradients (second moment). Adam is particularly effective for problems with sparse gradients and non-stationary objectives, making it one of the most popular optimization algorithms in deep learning.

Parameter	Type	Description
`step_size`	float	Learning rate (default 0.001)
`grad_fn`	Callable	The gradient function of $f(x, y)$
`x0`	tuple[float, float]	(x_0, y_0); the starting point of the path
`beta1`	float	Exponential decay rate for first moment estimates (default 0.9)
`beta2`	float	Exponential decay rate for second moment estimates (default 0.999)
`epsilon`	float	Small constant for numerical stability (default 10^{-8})
`max_iters`	int	Maximum number of iterations allowed (default 1,000,000).
`tol`	float	Stopping tolerance; if $\|x_{\text{next}} - x\|$ is smaller than this threshold, the algorithm stops
`label`	str	A string label to tag results in global dictionaries

Adam Pseudocode

Input: step_size α, gradient function $grad\_fn$, initial point x_0, β_1, β_2, epsilon ϵ, max iterations N, tolerance ε, label
Output: Path of iterates
$x = x_0$ $m = 0, v = 0$ `// first and second moments`
path $= [x]$
for $k = 1$ **to** N **do**
  $g = grad\_fn(x)$ $m = \beta_1 \cdot m + (1 - \beta_1) \cdot g$ $v = \beta_2 \cdot v + (1 - \beta_2) \cdot g \odot g$ $\hat{m} = \frac{m}{1-\beta_1^k}$ `// bias correction`
  $\hat{v} = \frac{v}{1-\beta_2^k}$ `// bias correction`
  $x_{\text{next}} = x - \frac{\alpha}{\sqrt{\hat{v}}+\epsilon} \odot \hat{m}$ append x_{next} to path
  if $\|x_{next} - x\| < \varepsilon$ **then**
    print "Converged at x_{next} in k iterations" **break**
  end
  $x = x_{\text{next}}$
end
return path

Adam maintains exponential moving averages of gradients (m) and squared gradients (v). The bias-corrected estimates are:

$$\hat{m} = \frac{m}{1 - \beta_1^k}, \quad \hat{v} = \frac{v}{1 - \beta_2^k}$$

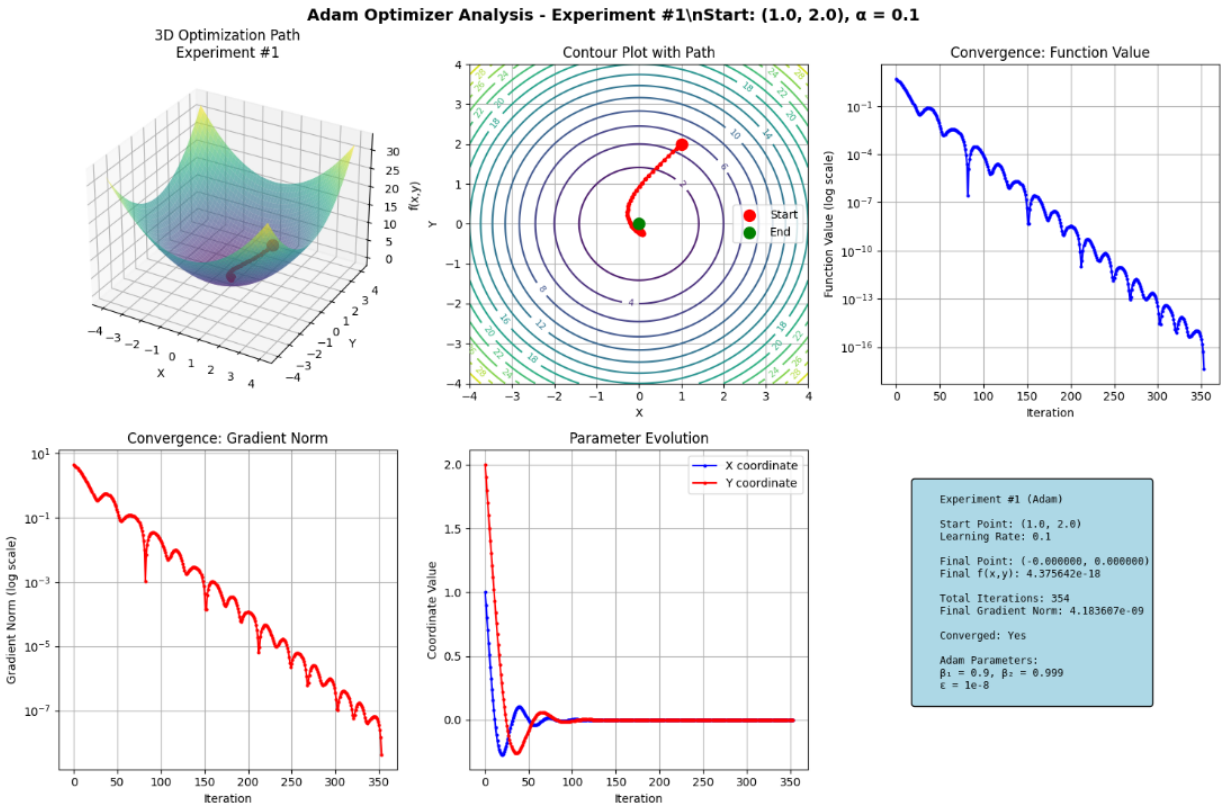
The update rule combines momentum with adaptive learning rates:

$$x_{\text{next}} = x - \frac{\alpha}{\sqrt{\hat{v}} + \epsilon} \odot \hat{m}$$

This provides the benefits of both momentum (through $\hat{m}$) and adaptive learning rates (through $\hat{v}$).

ADAM Results

Adam performed consistently well across all three functions. Unlike AdaGrad, which often plateaued due to an overly shrinking learning rate, Adam maintained stable progress. This comes from contributions of adaptive learning rates with momentum. For the challenging Banana Valley, Adam successfully navigated the curved valley to in thousands of iterations. It waslower than Newton's Method, but far more robust than Gradient Descent or AdaGrad.



1 Experimental Analysis

1.1 Increasing Problem Difficulty

As the functions became more complex (from **Convex Bowl** → **Banana Valley** → **Cosine Bumps**), optimization performance diverged significantly between the algorithms:

- **Convex Bowl (simple quadratic surface):** All methods converged reliably and quickly. Vanilla gradient descent, Adam, AdaGrad, and Newton all reached near-optimal solutions, with Newton requiring the fewest iterations.
- **Banana Valley (Rosenbrock-like function):** The narrow curved valley challenged the descent of the vanilla gradient, especially with higher learning rates that caused oscillations. Adaptive methods (Adam, AdaGrad) stabilized convergence, though they required more iterations. Newton's method performed well if the Hessian was stable but occasionally showed sensitivity to initialization.
- **Cosine Bumps (non-convex, many local minima):** This was the most challenging case. Vanilla GD often got trapped in local minima. Adaptive methods helped escape shallow basins but could still stall. Newton sometimes converged to spurious minima if started poorly. Here, the choice and initialization of the method had the greatest impact.

1.2 Function Shape/Surface and Method Suitability

- **Smooth convex surfaces (bowl)** favor simple methods: Vanilla GD with appropriate learning rates is sufficient, while Newton accelerates convergence with fewer steps.
- **Ill-conditioned valleys (banana)** highlight the advantage of second-order or adaptive methods: learning rate scheduling or curvature information is crucial.

- **Highly non-convex landscapes (cosine bumps)** emphasize exploration: adaptive methods are somewhat better, but no algorithm guarantees global optimum.

1.3 Impact of Initial Solution

- In convex problems, initialization had a minimal effect beyond minor iteration differences.
- In the banana valley, starting farther from the minimum required smaller learning rates to avoid divergence.
- In cosine bumps, initialization often determined which local minimum the optimizer converged to. Different runs could produce significantly different outputs.

1.4 Impact of Method and Parameters

- **Learning rate** was critical: too large caused divergence (especially in vanilla GD), too small led to slow convergence.
- **Adaptive optimizers** tolerated wider ranges of learning rates and reduced manual tuning but sometimes plateaued before reaching exact optima.
- **Newton's method** converged fastest when Hessians were well-conditioned but incurred higher per-iteration cost and sensitivity to poor starts.

1.5 Runtime vs. Iterations

- **Newton:** Very few iterations but higher runtime per step.
- **Vanilla GD:** Many iterations, but each iteration was computationally cheap. Sensitive to learning rate.
- **AdaGrad / Adam:** Iteration counts between vanilla GD and Newton. More robust to hyperparameter choices, but runtime per step slightly higher due to internal bookkeeping.

2 Conclusion

1. **Problem difficulty dictates algorithm choice:** simple convex problems can be solved with basic gradient descent, while complex non-convex problems require adaptive or second-order methods.
2. **Initialization matters more in non-convex landscapes:** good starting points reduce the risk of poor minima, especially for cosine bumps.
3. **Adaptive methods (Adam, AdaGrad)** outperform vanilla GD in challenging settings without requiring Hessian computations.
4. **Newton's method** is extremely powerful on well-conditioned problems but impractical if the curvature is difficult to compute or the surface is ill-conditioned.
5. **Trade-offs between runtime and iterations:** no single method dominates all scenarios, the choice depends on the structure of the problem and the computational budget.

Hyperparameter sensitivity is one of the key bottlenecks in optimization, even though adaptive methods help these do not eliminate the issue. Furthermore, function geometry should inform algorithm choice: convex vs. non-convex surfaces behave very differently. The initialization strategy is as important as the optimization algorithm in highly nonconvex problems. Some possible experiments to improve results include: trying different learning rate schedules to mitigate oscillations. Add stochastic sampling or noise (like in SGD) to escape local minima in cosine bumps. Investigate hybrid methods that combine adaptive learning with second-order updates for robustness.

method	Function	iterations	output x	output y	starting_point	learning_rate	run_times
AdaGrad	Convex Bowl	1057	0	0.00011	(1, 2)	0.1	2.1975
AdaGrad	Convex Bowl	1057	-0.00011	0	(-2, 1)	0.1	0.3545
AdaGrad	Convex Bowl	2282	-0.00017	0.00017	(-3, 3)	0.1	0.7334
AdaGrad	Convex Bowl	48603	0	0.01014	(1, 2)	0.01	13.0922
AdaGrad	Convex Bowl	48603	-0.01014	0	(-2, 1)	0.01	11.657
AdaGrad	Convex Bowl	100000	-0.0224	0.0224	(-3, 3)	0.01	24.3632
AdaGrad	Convex Bowl	76	0	0	(1, 2)	0.5	0.017
AdaGrad	Convex Bowl	76	0	0	(-2, 1)	0.5	0.019
AdaGrad	Convex Bowl	154	-0.00001	0.00001	(-3, 3)	0.5	0.036
AdaGrad	Banana Valley	41887	1.00385	1.00773	(1, 2)	0.1	13.4512
AdaGrad	Banana Valley	100000	0.84736	0.71761	(-2, 1)	0.1	33.0499
AdaGrad	Banana Valley	100000	-1.12874	1.2787	(-3, 3)	0.1	32.9677
AdaGrad	Banana Valley	100000	1.26942	1.61218	(1, 2)	0.01	32.5213
AdaGrad	Banana Valley	10401	-1.31758	1.73861	(-2, 1)	0.01	3.3787
AdaGrad	Banana Valley	18555	-2.01912	4.06823	(-3, 3)	0.01	5.9256
AdaGrad	Banana Valley	10867	1.0008	1.00159	(1, 2)	0.5	3.5786
AdaGrad	Banana Valley	29629	0.99776	0.99551	(-2, 1)	0.5	9.5008
AdaGrad	Banana Valley	76106	0.99387	0.98776	(-3, 3)	0.5	25.0653
AdaGrad	Cosines Bump	443	2.59569	2.59574	(1, 2)	0.1	0.1948
AdaGrad	Cosines Bump	443	-2.59574	2.59569	(-2, 1)	0.1	0.2054
AdaGrad	Cosines Bump	74	-2.59574	2.59574	(-3, 3)	0.1	0.0333
AdaGrad	Cosines Bump	22506	2.59151	2.59564	(1, 2)	0.01	8.2948
AdaGrad	Cosines Bump	22506	-2.59564	2.59151	(-2, 1)	0.01	8.257
AdaGrad	Cosines Bump	3119	-2.59601	2.59601	(-3, 3)	0.01	1.0328
AdaGrad	Cosines Bump	26	2.59574	2.59574	(1, 2)	0.5	0.009
AdaGrad	Cosines Bump	26	-2.59574	2.59574	(-2, 1)	0.5	0.009
AdaGrad	Cosines Bump	7	-2.59574	2.59574	(-3, 3)	0.5	0.002
Adam	Convex Bowl	353	0	0	(1, 2)	0.1	0.0083
Adam	Convex Bowl	353	0	0	(-2, 1)	0.1	0.0064
Adam	Convex Bowl	282	0	0	(-3, 3)	0.1	0.005
Adam	Convex Bowl	1085	0	0	(1, 2)	0.01	0.0186
Adam	Convex Bowl	1085	0	0	(-2, 1)	0.01	0.018
Adam	Convex Bowl	1667	0	0	(-3, 3)	0.01	0.0665
Adam	Convex Bowl	361	0	0	(1, 2)	0.5	0.0142
Adam	Convex Bowl	361	0	0	(-2, 1)	0.5	0.01
Adam	Convex Bowl	327	0	0	(-3, 3)	0.5	0.006
Adam	Banana Valley	2994	1	1	(1, 2)	0.1	0.0831
Adam	Banana Valley	5176	1	1	(-2, 1)	0.1	0.1279
Adam	Banana Valley	7121	1	1	(-3, 3)	0.1	0.198
Adam	Banana Valley	8110	1	1	(1, 2)	0.01	0.1784
Adam	Banana Valley	11562	1	1	(-2, 1)	0.01	0.3191
Adam	Banana Valley	13855	1	1	(-3, 3)	0.01	0.3455
Adam	Banana Valley	1211	1	1	(1, 2)	0.5	0.023

method	Function	iterations	output x	output y	starting_point	learning_rate	run_times
Adam	Banana Valley	2331	1	1	(-2, 1)	0.5	0.0499
Adam	Banana Valley	3713	1	1	(-3, 3)	0.5	0.1022
Adam	Cosines Bump	380	2.59574	2.59574	(1, 2)	0.1	0.0122
Adam	Cosines Bump	380	-2.59574	2.59574	(-2, 1)	0.1	0.015
Adam	Cosines Bump	327	-2.59574	2.59574	(-3, 3)	0.1	0.0085
Adam	Cosines Bump	520	2.59574	2.59574	(1, 2)	0.01	0.0173
Adam	Cosines Bump	520	-2.59574	2.59574	(-2, 1)	0.01	0.0114
Adam	Cosines Bump	336	-2.59574	2.59574	(-3, 3)	0.01	0.0123
Adam	Cosines Bump	371	2.59574	2.59574	(1, 2)	0.5	0.0113
Adam	Cosines Bump	371	-2.59574	2.59574	(-2, 1)	0.5	0.0087
Adam	Cosines Bump	323	-2.59574	2.59574	(-3, 3)	0.5	0.012
Newton's Method	Convex Bowl	190	0	0	(1, 2)	0.1	0.0341
Newton's Method	Convex Bowl	190	0	0	(-2, 1)	0.1	0.005
Newton's Method	Convex Bowl	196	0	0	(-3, 3)	0.1	0.004
Newton's Method	Convex Bowl	1982	0	0	(1, 2)	0.01	0.037
Newton's Method	Convex Bowl	1982	0	0	(-2, 1)	0.01	0.0231
Newton's Method	Convex Bowl	2046	0	0	(-3, 3)	0.01	0.0236
Newton's Method	Convex Bowl	29	0	0	(1, 2)	0.5	0
Newton's Method	Convex Bowl	29	0	0	(-2, 1)	0.5	0.001
Newton's Method	Convex Bowl	30	0	0	(-3, 3)	0.5	0
Newton's Method	Banana Valley	233	1	1	(1, 2)	0.1	0.0054
Newton's Method	Banana Valley	310	1	1	(-2, 1)	0.1	0.007
Newton's Method	Banana Valley	334	1	1	(-3, 3)	0.1	0.007
Newton's Method	Banana Valley	2441	1	1	(1, 2)	0.01	0.0681
Newton's Method	Banana Valley	2708	1	1	(-2, 1)	0.01	0.0347
Newton's Method	Banana Valley	2856	1	1	(-3, 3)	0.01	0.0372
Newton's Method	Banana Valley	36	1	1	(1, 2)	0.5	0.001
Newton's Method	Banana Valley	64	1	1	(-2, 1)	0.5	0.001
Newton's Method	Banana Valley	72	1	1	(-3, 3)	0.5	0
Newton's Method	Banana Valley	29	0	2.59574	(1, 2)	0.5	0.001
Newton's Method	Cosines Bump	195	0	2.59574	(1, 2)	0.1	0.006
Newton's Method	Cosines Bump	195	-2.59574	0	(-2, 1)	0.1	0.0063
Newton's Method	Cosines Bump	193	-2.59574	2.59574	(-3, 3)	0.1	0.006
Newton's Method	Cosines Bump	2042	0	2.59574	(1, 2)	0.01	0.0613
Newton's Method	Cosines Bump	2042	-2.59574	0	(-2, 1)	0.01	0.061
Newton's Method	Cosines Bump	2019	-2.59574	2.59574	(-3, 3)	0.01	0.0536
Newton's Method	Cosines Bump	29	-2.59574	0	(-2, 1)	0.5	0.001
Newton's Method	Cosines Bump	30	-2.59574	2.59574	(-3, 3)	0.5	0.001
Gradient Descent	Convex Bowl	59	0	0	(1, 2)	0.1	0.002
Gradient Descent	Convex Bowl	59	0	0	(-2, 1)	0.1	0.002
Gradient Descent	Convex Bowl	62	0	0	(-3, 3)	0.1	0.001
Gradient Descent	Convex Bowl	531	0.00002	0.00004	(1, 2)	0.01	0.0053
Gradient Descent	Convex Bowl	531	-0.00004	0.00002	(-2, 1)	0.01	0.005

method	Function	iterations	output x	output y	starting_point	learning_rate	run_times
Gradient Descent	Convex Bowl	562	-0.00003	0.00003	(-3, 3)	0.01	0.005
Gradient Descent	Convex Bowl	1	0	0	(1, 2)	0.5	0
Gradient Descent	Convex Bowl	1	0	0	(-2, 1)	0.5	0
Gradient Descent	Convex Bowl	1	0	0	(-3, 3)	0.5	0
Gradient Descent	Banana Valley	9999	1.07862	1.16371	(1, 2)	0.0005	0.1083
Gradient Descent	Banana Valley	9999	0.93731	0.87829	(-2, 1)	0.0005	0.0825
Gradient Descent	Banana Valley	9999	1.24806	1.55852	(-3, 3)	0.0005	0.0733
Gradient Descent	Banana Valley	9999	1.31699	1.73552	(1, 2)	5.00E-05	0.0738
Gradient Descent	Banana Valley	9999	-0.68729	0.48039	(-2, 1)	5.00E-05	0.0695
Gradient Descent	Banana Valley	9999	-1.58394	2.51628	(-3, 3)	5.00E-05	0.0742
Gradient Descent	Banana Valley	9999	1.35431	1.8353	(1, 2)	5.00E-06	0.0715
Gradient Descent	Banana Valley	9999	-1.09471	1.20631	(-2, 1)	5.00E-06	0.0709
Gradient Descent	Banana Valley	9999	-1.78179	3.18201	(-3, 3)	5.00E-06	0.0675
Gradient Descent	Cosines Bump	7	2.59574	2.59574	(1, 2)	0.1	0.001
Gradient Descent	Cosines Bump	7	-2.59574	2.59574	(-2, 1)	0.1	0
Gradient Descent	Cosines Bump	5	-2.59574	2.59574	(-3, 3)	0.1	0
Gradient Descent	Cosines Bump	117	2.59573	2.59574	(1, 2)	0.01	0.003
Gradient Descent	Cosines Bump	117	-2.59574	2.59573	(-2, 1)	0.01	0.001
Gradient Descent	Cosines Bump	98	-2.59575	2.59575	(-3, 3)	0.01	0.001
Gradient Descent	Cosines Bump	9999	-4.9063	-4.9063	(1, 2)	0.5	0.1035
Gradient Descent	Cosines Bump	9999	4.9063	-4.9063	(-2, 1)	0.5	0.0988
Gradient Descent	Cosines Bump	9999	4.9063	-4.9063	(-3, 3)	0.5	0.1275

Submitted by Matthew Poncini, Daniella Arteaga Mendoza on 19 September 2025.

method	Function	iterations	output x	output y	starting_point	learning_rate	run_times
AdaGrad	Convex Bowl	1057	0	0.00011	(1, 2)	0.1	2.1975
AdaGrad	Convex Bowl	1057	-0.00011	0	(-2, 1)	0.1	0.3545
AdaGrad	Convex Bowl	2282	-0.00017	0.00017	(-3, 3)	0.1	0.7334
AdaGrad	Convex Bowl	48603	0	0.01014	(1, 2)	0.01	13.0922
AdaGrad	Convex Bowl	48603	-0.01014	0	(-2, 1)	0.01	11.657
AdaGrad	Convex Bowl	100000	-0.0224	0.0224	(-3, 3)	0.01	24.3632
AdaGrad	Convex Bowl	76	0	0	(1, 2)	0.5	0.017
AdaGrad	Convex Bowl	76	0	0	(-2, 1)	0.5	0.019
AdaGrad	Convex Bowl	154	-0.00001	0.00001	(-3, 3)	0.5	0.036
AdaGrad	Banana Valley	41887	1.00385	1.00773	(1, 2)	0.1	13.4512
AdaGrad	Banana Valley	100000	0.84736	0.71761	(-2, 1)	0.1	33.0499
AdaGrad	Banana Valley	100000	-1.12874	1.2787	(-3, 3)	0.1	32.9677
AdaGrad	Banana Valley	100000	1.26942	1.61218	(1, 2)	0.01	32.5213
AdaGrad	Banana Valley	10401	-1.31758	1.73861	(-2, 1)	0.01	3.3787
AdaGrad	Banana Valley	18555	-2.01912	4.06823	(-3, 3)	0.01	5.9256
AdaGrad	Banana Valley	10867	1.0008	1.00159	(1, 2)	0.5	3.5786
AdaGrad	Banana Valley	29629	0.99776	0.99551	(-2, 1)	0.5	9.5008
AdaGrad	Banana Valley	76106	0.99387	0.98776	(-3, 3)	0.5	25.0653
AdaGrad	Cosines Bump	443	2.59569	2.59574	(1, 2)	0.1	0.1948
AdaGrad	Cosines Bump	443	-2.59574	2.59569	(-2, 1)	0.1	0.2054
AdaGrad	Cosines Bump	74	-2.59574	2.59574	(-3, 3)	0.1	0.0333
AdaGrad	Cosines Bump	22506	2.59151	2.59564	(1, 2)	0.01	8.2948
AdaGrad	Cosines Bump	22506	-2.59564	2.59151	(-2, 1)	0.01	8.257
AdaGrad	Cosines Bump	3119	-2.59601	2.59601	(-3, 3)	0.01	1.0328
AdaGrad	Cosines Bump	26	2.59574	2.59574	(1, 2)	0.5	0.009
AdaGrad	Cosines Bump	26	-2.59574	2.59574	(-2, 1)	0.5	0.009
AdaGrad	Cosines Bump	7	-2.59574	2.59574	(-3, 3)	0.5	0.002
Adam	Convex Bowl	353	0	0	(1, 2)	0.1	0.0083
Adam	Convex Bowl	353	0	0	(-2, 1)	0.1	0.0064
Adam	Convex Bowl	282	0	0	(-3, 3)	0.1	0.005
Adam	Convex Bowl	1085	0	0	(1, 2)	0.01	0.0186
Adam	Convex Bowl	1085	0	0	(-2, 1)	0.01	0.018
Adam	Convex Bowl	1667	0	0	(-3, 3)	0.01	0.0665
Adam	Convex Bowl	361	0	0	(1, 2)	0.5	0.0142
Adam	Convex Bowl	361	0	0	(-2, 1)	0.5	0.01
Adam	Convex Bowl	327	0	0	(-3, 3)	0.5	0.006
Adam	Banana Valley	2994	1	1	(1, 2)	0.1	0.0831
Adam	Banana Valley	5176	1	1	(-2, 1)	0.1	0.1279
Adam	Banana Valley	7121	1	1	(-3, 3)	0.1	0.198
Adam	Banana Valley	8110	1	1	(1, 2)	0.01	0.1784
Adam	Banana Valley	11562	1	1	(-2, 1)	0.01	0.3191
Adam	Banana Valley	13855	1	1	(-3, 3)	0.01	0.3455
Adam	Banana Valley	1211	1	1	(1, 2)	0.5	0.023

Figure 1: Clean output — page 1

method	Function	iterations	output x	output y	starting_point	learning_rate	run_times
Adam	Banana Valley	2331	1	1	(-2, 1)	0.5	0.0499
Adam	Banana Valley	3713	1	1	(-3, 3)	0.5	0.1022
Adam	Cosines Bump	380	2.59574	2.59574	(1, 2)	0.1	0.0122
Adam	Cosines Bump	380	-2.59574	2.59574	(-2, 1)	0.1	0.015
Adam	Cosines Bump	327	-2.59574	2.59574	(-3, 3)	0.1	0.0085
Adam	Cosines Bump	520	2.59574	2.59574	(1, 2)	0.01	0.0173
Adam	Cosines Bump	520	-2.59574	2.59574	(-2, 1)	0.01	0.0114
Adam	Cosines Bump	336	-2.59574	2.59574	(-3, 3)	0.01	0.0123
Adam	Cosines Bump	371	2.59574	2.59574	(1, 2)	0.5	0.0113
Adam	Cosines Bump	371	-2.59574	2.59574	(-2, 1)	0.5	0.0087
Adam	Cosines Bump	323	-2.59574	2.59574	(-3, 3)	0.5	0.012
Newton's Method	Convex Bowl	190	0	0	(1, 2)	0.1	0.0341
Newton's Method	Convex Bowl	190	0	0	(-2, 1)	0.1	0.005
Newton's Method	Convex Bowl	196	0	0	(-3, 3)	0.1	0.004
Newton's Method	Convex Bowl	1982	0	0	(1, 2)	0.01	0.037
Newton's Method	Convex Bowl	1982	0	0	(-2, 1)	0.01	0.0231
Newton's Method	Convex Bowl	2046	0	0	(-3, 3)	0.01	0.0236
Newton's Method	Convex Bowl	29	0	0	(1, 2)	0.5	0
Newton's Method	Convex Bowl	29	0	0	(-2, 1)	0.5	0.001
Newton's Method	Convex Bowl	30	0	0	(-3, 3)	0.5	0
Newton's Method	Banana Valley	233	1	1	(1, 2)	0.1	0.0054
Newton's Method	Banana Valley	310	1	1	(-2, 1)	0.1	0.007
Newton's Method	Banana Valley	334	1	1	(-3, 3)	0.1	0.007
Newton's Method	Banana Valley	2441	1	1	(1, 2)	0.01	0.0681
Newton's Method	Banana Valley	2708	1	1	(-2, 1)	0.01	0.0347
Newton's Method	Banana Valley	2856	1	1	(-3, 3)	0.01	0.0372
Newton's Method	Banana Valley	36	1	1	(1, 2)	0.5	0.001
Newton's Method	Banana Valley	64	1	1	(-2, 1)	0.5	0.001
Newton's Method	Banana Valley	72	1	1	(-3, 3)	0.5	0
Newton's Method	Banana Valley	29	0	2.59574	(1, 2)	0.5	0.001
Newton's Method	Cosines Bump	195	0	2.59574	(1, 2)	0.1	0.006
Newton's Method	Cosines Bump	195	-2.59574	0	(-2, 1)	0.1	0.0063
Newton's Method	Cosines Bump	193	-2.59574	2.59574	(-3, 3)	0.1	0.006
Newton's Method	Cosines Bump	2042	0	2.59574	(1, 2)	0.01	0.0613
Newton's Method	Cosines Bump	2042	-2.59574	0	(-2, 1)	0.01	0.061
Newton's Method	Cosines Bump	2019	-2.59574	2.59574	(-3, 3)	0.01	0.0536
Newton's Method	Cosines Bump	29	-2.59574	0	(-2, 1)	0.5	0.001
Newton's Method	Cosines Bump	30	-2.59574	2.59574	(-3, 3)	0.5	0.001
Gradient Descent	Convex Bowl	59	0	0	(1, 2)	0.1	0.002
Gradient Descent	Convex Bowl	59	0	0	(-2, 1)	0.1	0.002
Gradient Descent	Convex Bowl	62	0	0	(-3, 3)	0.1	0.001
Gradient Descent	Convex Bowl	531	0.00002	0.00004	(1, 2)	0.01	0.0053
Gradient Descent	Convex Bowl	531	-0.00004	0.00002	(-2, 1)	0.01	0.005

Figure 2: Clean output — page 2

method	Function	iterations	output x	output y	starting_point	learning_rate	run_times
Gradient Descent	Convex Bowl	562	-0.00003	0.00003	(-3, 3)	0.01	0.005
Gradient Descent	Convex Bowl	1	0	0	(1, 2)	0.5	0
Gradient Descent	Convex Bowl	1	0	0	(-2, 1)	0.5	0
Gradient Descent	Convex Bowl	1	0	0	(-3, 3)	0.5	0
Gradient Descent	Banana Valley	9999	1.07862	1.16371	(1, 2)	0.0005	0.1083
Gradient Descent	Banana Valley	9999	0.93731	0.87829	(-2, 1)	0.0005	0.0825
Gradient Descent	Banana Valley	9999	1.24806	1.55852	(-3, 3)	0.0005	0.0733
Gradient Descent	Banana Valley	9999	1.31699	1.73552	(1, 2)	5.00E-05	0.0738
Gradient Descent	Banana Valley	9999	-0.68729	0.48039	(-2, 1)	5.00E-05	0.0695
Gradient Descent	Banana Valley	9999	-1.58394	2.51628	(-3, 3)	5.00E-05	0.0742
Gradient Descent	Banana Valley	9999	1.35431	1.8353	(1, 2)	5.00E-06	0.0715
Gradient Descent	Banana Valley	9999	-1.09471	1.20631	(-2, 1)	5.00E-06	0.0709
Gradient Descent	Banana Valley	9999	-1.78179	3.18201	(-3, 3)	5.00E-06	0.0675
Gradient Descent	Cosines Bump	7	2.59574	2.59574	(1, 2)	0.1	0.001
Gradient Descent	Cosines Bump	7	-2.59574	2.59574	(-2, 1)	0.1	0
Gradient Descent	Cosines Bump	5	-2.59574	2.59574	(-3, 3)	0.1	0
Gradient Descent	Cosines Bump	117	2.59573	2.59574	(1, 2)	0.01	0.003
Gradient Descent	Cosines Bump	117	-2.59574	2.59573	(-2, 1)	0.01	0.001
Gradient Descent	Cosines Bump	98	-2.59575	2.59575	(-3, 3)	0.01	0.001
Gradient Descent	Cosines Bump	9999	-4.9063	-4.9063	(1, 2)	0.5	0.1035
Gradient Descent	Cosines Bump	9999	4.9063	-4.9063	(-2, 1)	0.5	0.0988
Gradient Descent	Cosines Bump	9999	4.9063	-4.9063	(-3, 3)	0.5	0.1275

Figure 3: Clean output — page 3