Adam (and beyond)

Team: Boson AI

Project summary

- Implemented Gradient-descent (GD), Adam, and AdaMax using Numpy; Extended GD and Adam's codes to Tensors and finally a class based method that can be used to train NN models
- Compared Adam and GD on various test-functions – Paraboloid, Rosenbrock, Rastrigin, Ackley, Spherical – to gain better understanding optimization process. The tests were done on level-curves and optimizers' update vectors were plotted
- Implemented a very, very, simple CNN in Numpy to visualize the update vectors of GD and Adam on loss level-curves of MSE loss
- Compared Gradient-descent, (implemented) Adam and Pytorch's Adam using a simple MLP trained on MNIST data
- Explored the concept of natural gradient (NG) and how optimizers like GD are not apt for loss manifolds where Riemannian metric tensor $\neq I$
- Implemented a test-code of NG in Numpy for Rosenbrock and extended it in Pytorch for KL divergence
- Explored Hessian based methods – Newton and L-BFGS – and implemented their codes using Numpy and Torch; Compared Newton, GD and Adam on a test function
- Explored and implemented a simple code of Latin-hypercube sampling

Introduction

Unconstrained optimization problem

$$w = \arg\min f(w)$$
$$w \in \square^n$$

where f could be a scalar valued or a vector valued function
More often that not, $f(w)$, is a loss function of a model that can be expressed as:

$$y = g(x; w)$$

Strictly speaking, the solution exists only if $f(w)$ is convex

Optimization methods

- First-order: gradient-based, e.g. , gradient-descent
- Second-order: Hessian-based, e.g., Newton-Gauss
- Non-Euclidean: Natural-gradient, e.g. natural-gradient descent
- Heuristic: derivative free algorithms, e.g, Nelder-Mead

Gradient-based methods: Gradient descent et. al

$$f(w + \Delta w) \approx f(w) + \Delta w^T \nabla f(w)$$

$\Delta w^T \nabla f(w)$ can be viewed as directional derivative

Simple gradient based update

$$w(t+1) = w(t) - \eta \nabla f(w(t))$$

The simple gradient descent updates only depend upon current gradient. This leads to the algorithm getting stuck in:

- Saddle points
- Ravines
- Local minimas (although all gradient based optimizers do to an extent)

An elegant solution – updates based on momentum/velocity

$$w(t+1) = w(t) - \eta v$$
$$v = \sum_t \alpha^t (1-\alpha) g_t$$

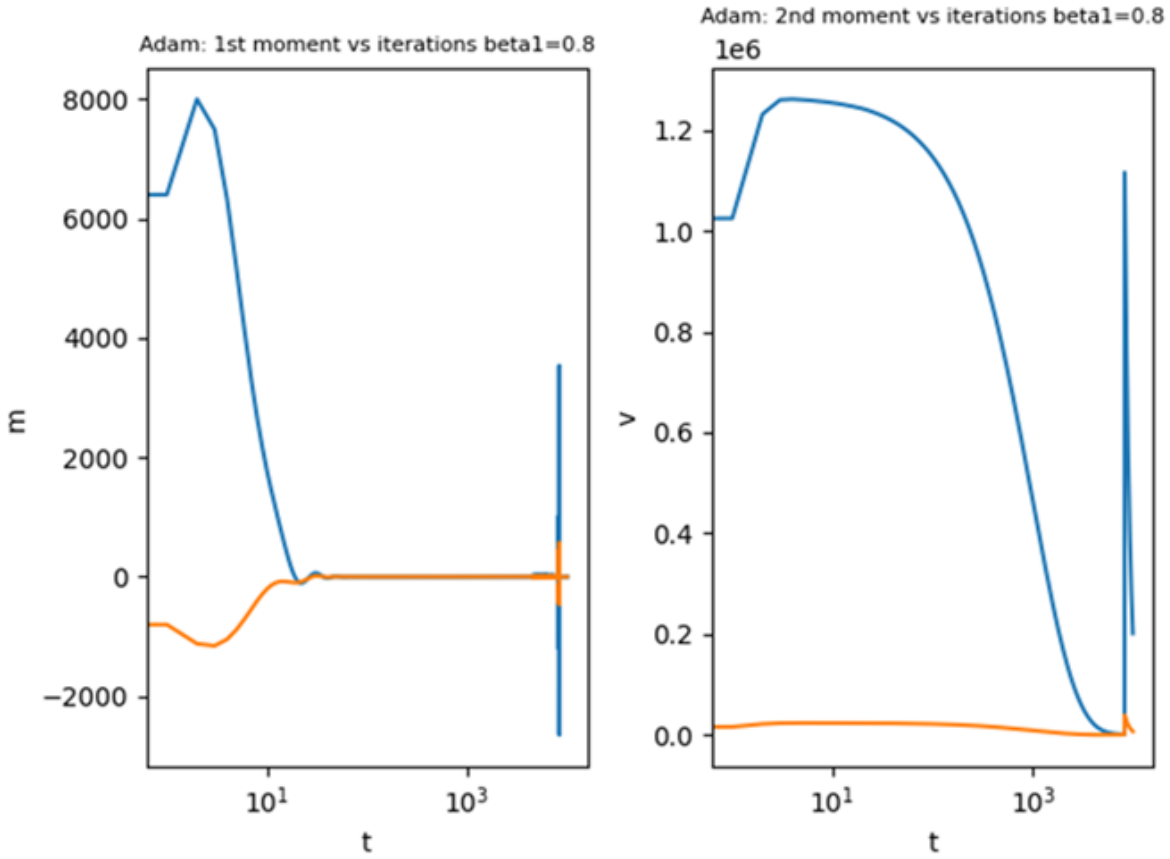
Adam

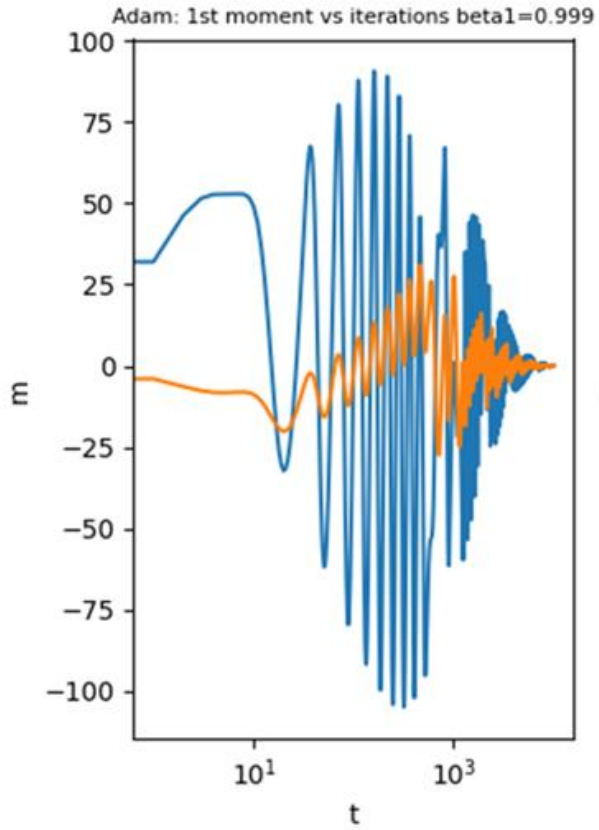
$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
$\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
$\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)

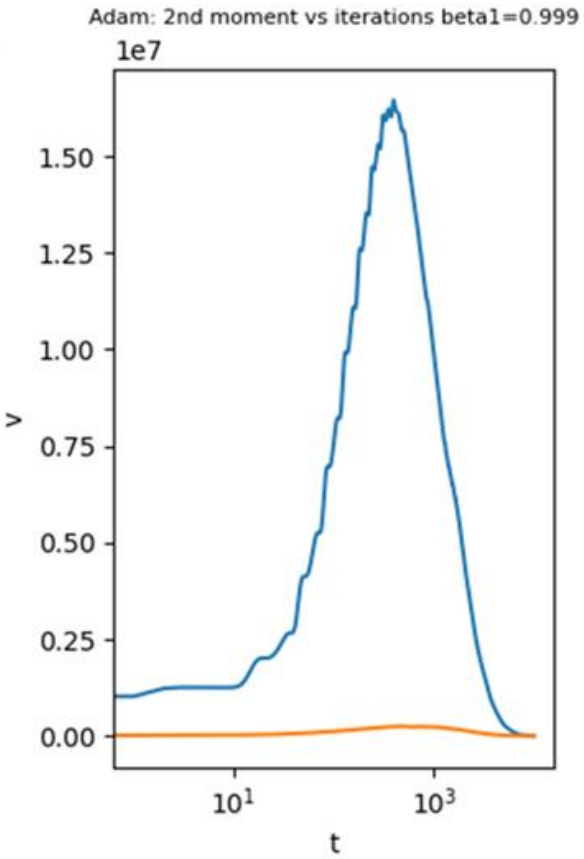
AdaMax

$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
$u_t \leftarrow \max(\beta_2 \cdot u_{t-1}, |g_t|)$ (Update the exponentially weighted infinity norm)
$\theta_t \leftarrow \theta_{t-1} - (\alpha / (1 - \beta_1^t)) \cdot m_t / u_t$ (Update parameters)

Variation of moments with iteration (optimization on 2D Rosenbrock function). The two lines are for two variables (x, y). Beta2 is fixed=0.999 in all these simulations







Optimization tests (1)

- **Tests were done on standard function used for testing optimization algorithm: Rosenbrock, Rastrigin, Ackley**
- **The domain of these functions were restricted to R^2**
- **The goal was to study how an update vector (for a given optimizer) transverses the level curves of a given function**
- **In the plots in next few slides, the lines represent level curves of a function, black arrows represent its gradient vectors, and blue and red curves represent update vectors from an optimizer**

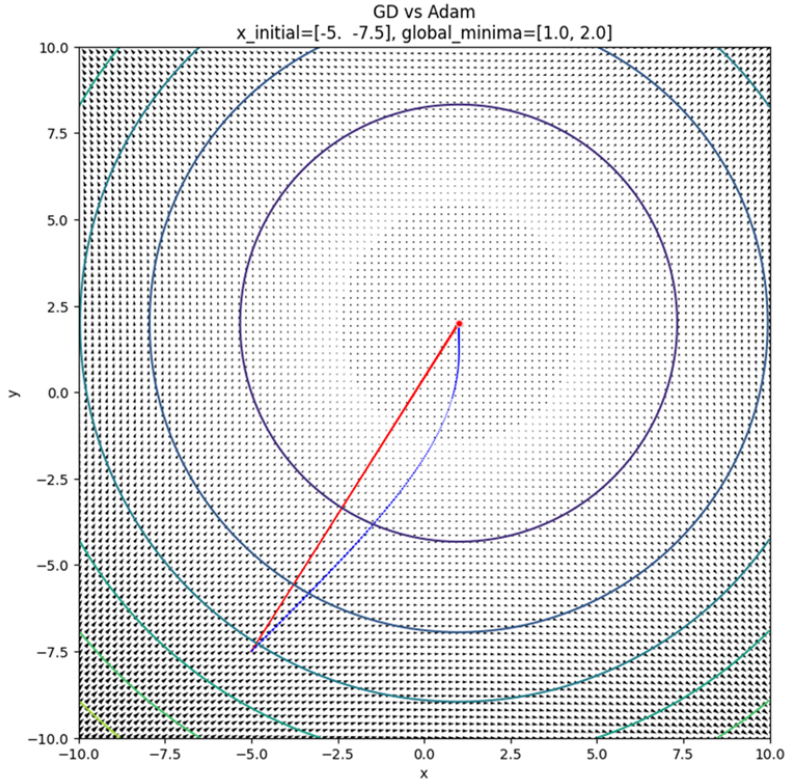
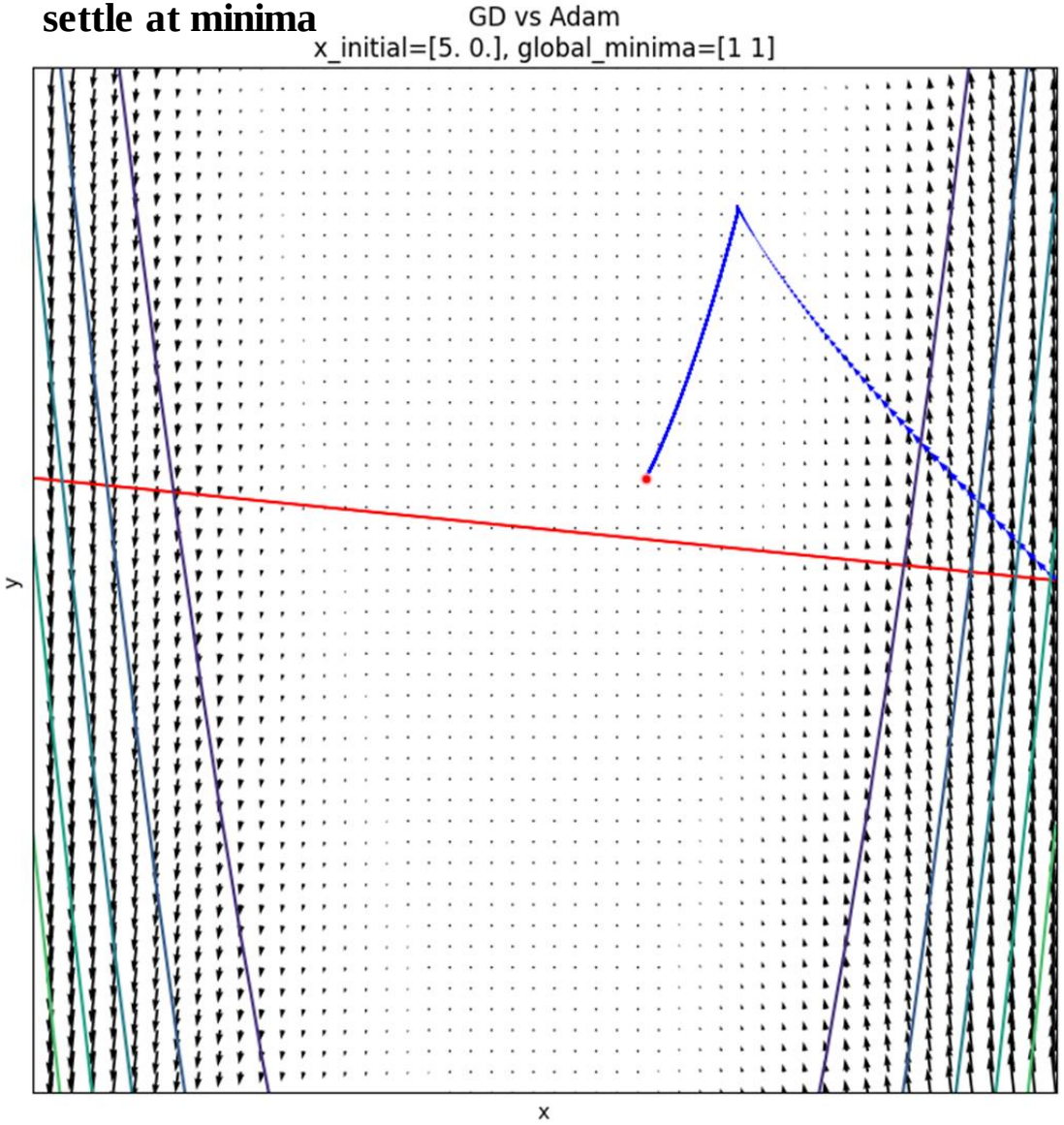


Figure shows optimization on 2D Paraboloid function. Red vectors are from GD and blue are from Adam

Optimization tests (2)

Rosenbrock function: initial point deep in the ravine. Adam finds its way whereas GD is unable to settle at minima

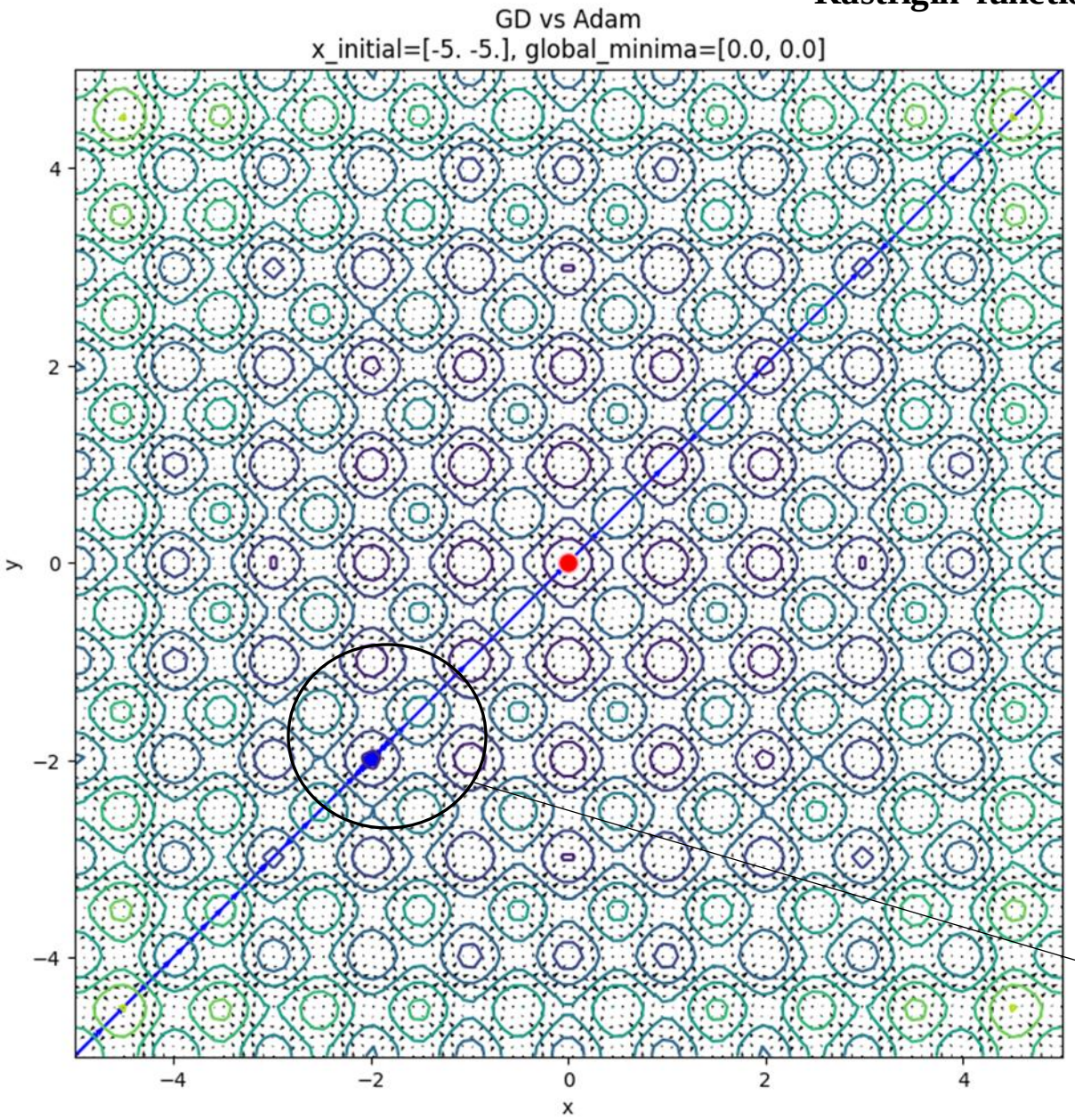


Gradient descent	
Nitr	10000
Lr	1e-3
x_initial	[5, 0]
x_final	[-1.91897874e+13, 4.09146131e+02]

Adam	
Nitr	10000
Lr	1e-1
betas	[0.9, 0.999]
eps	1e-7
x_initial	[5, 0]
x_final	[1.0001373 1.00027483]

Optimization tests (2)

Rastrigin function



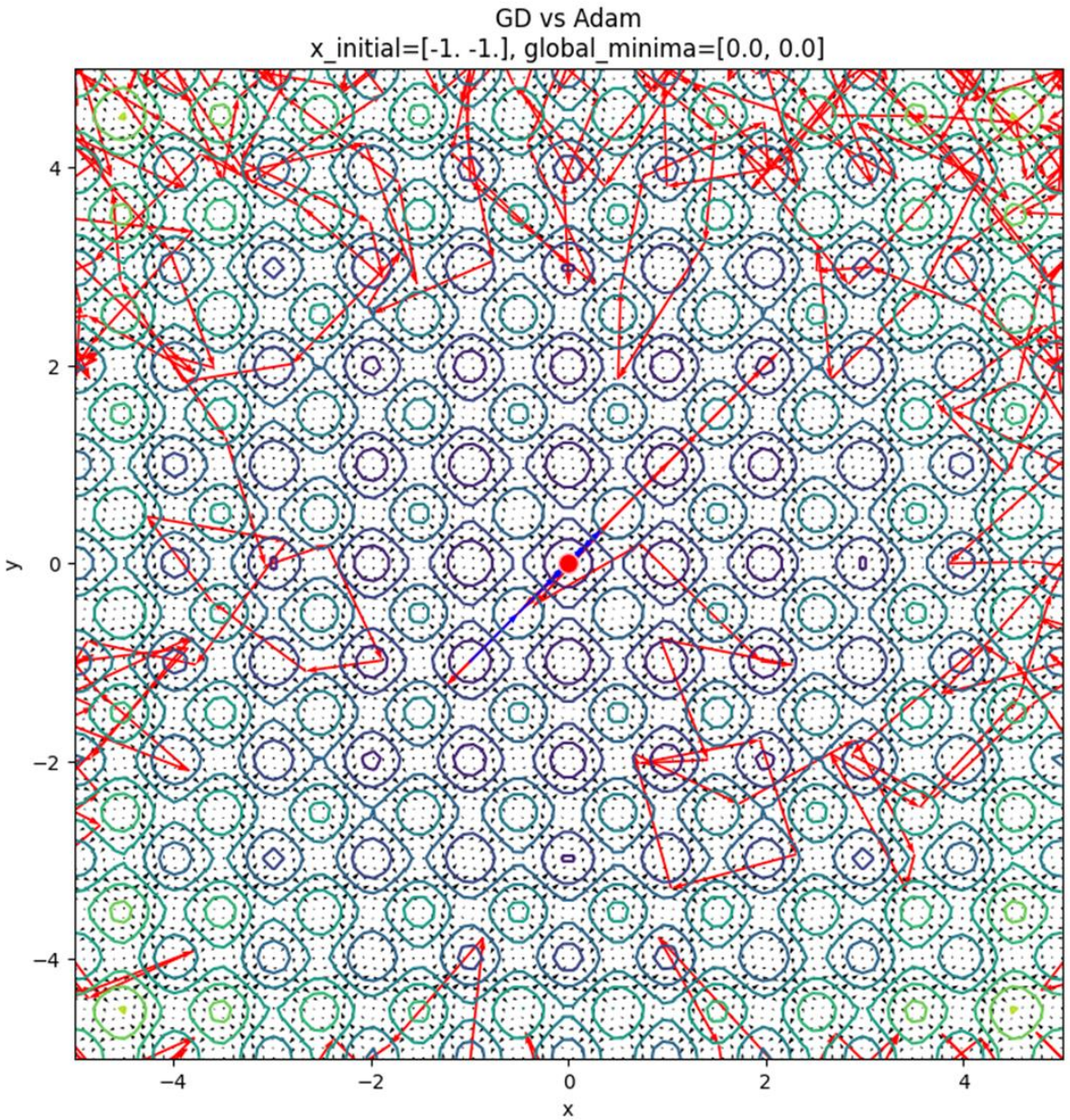
Gradient descent	
Nitr	10000
Lr	1e-3
x_initial	[-5, -5]
x_final	[-4.97469139, -4.97469139]

Adam	
Nitr	10000
Lr	1
betas	[0.9, 0.999]
eps	1e-7
x_initial	[-5, -5]
x_final	[-1.99923524, -1.99755086]

Got pushed out of a local minima

Optimization tests (3)

Rastrigin function



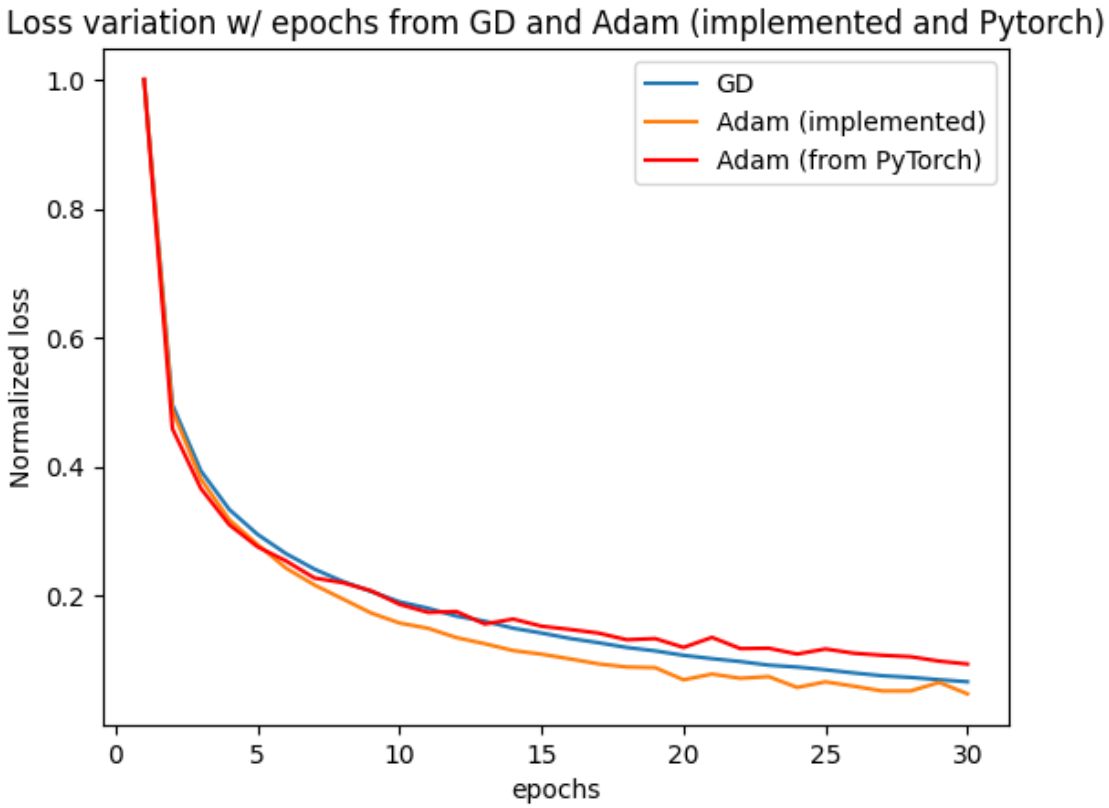
Gradient descent	
Nitr	10000
Lr	2e-2
x_initial	[-1, -1]
x_final	8.62994166, -8.96991123]

Adam	
Nitr	10000
Lr	0.5
betas	[0.9, 0.999]
eps	1e-7
x_initial	[-1, -1]
x_final	[0.01386211, 0.02081304]

GD w/ large learning rate – jumps from minima to minima

Adam test: test on NNs

MLP trained on MNIST data

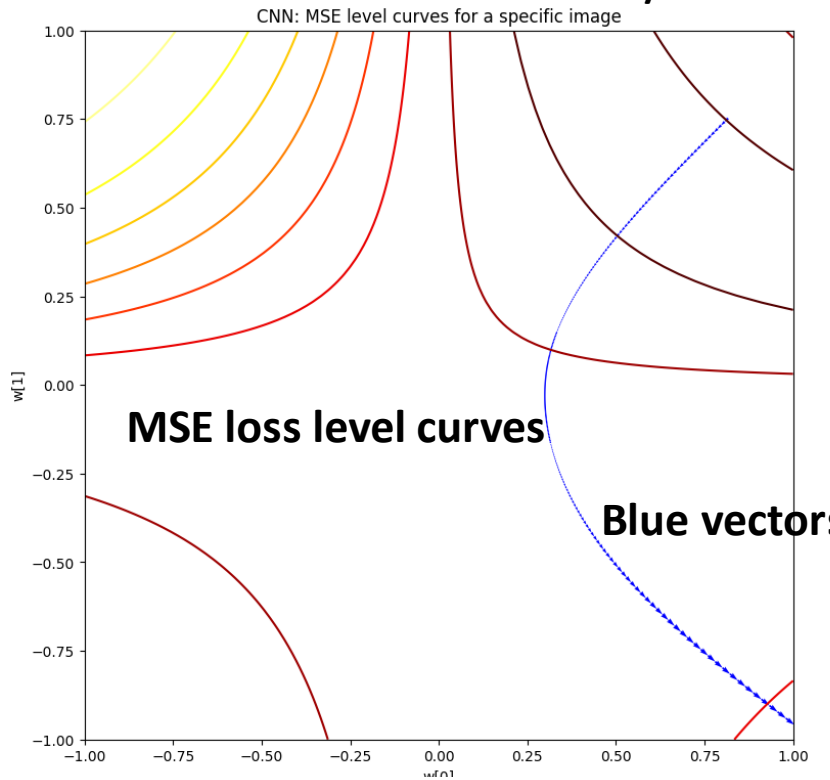


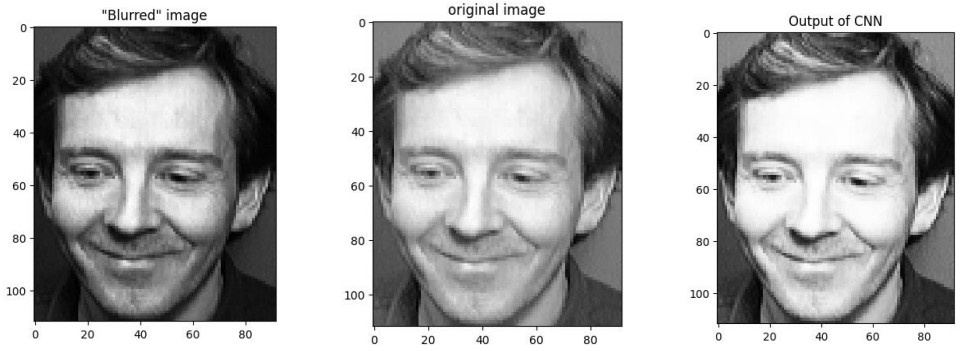
- **MLP with two hidden layers and cross-entropy loss**
- **Training done with default parameters of Adam and lr = 1e-3**

A very, very simple CNN implemented in Numpy

1-hidden layer, kernel_size=1,activation=leaky-RelU

Goal: to visualize the loss-curves/surfaces





Natural gradients (1)

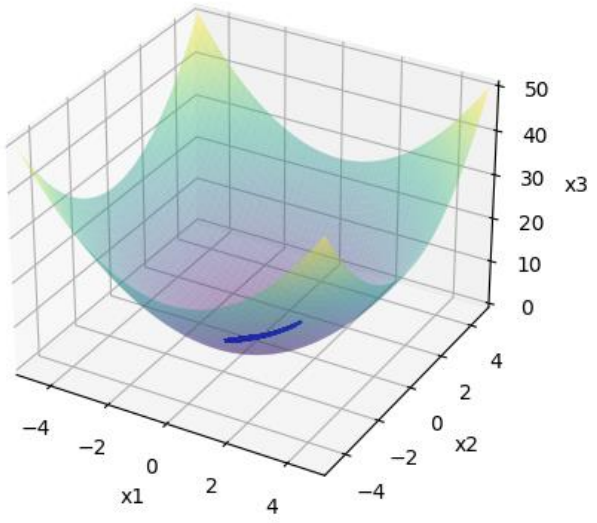
$$w(k+1) = w(k) - \eta G^{-1} g$$

$$G = \begin{bmatrix} g_{11} & g_{12} \\ g_{21} & g_{22} \end{bmatrix}$$

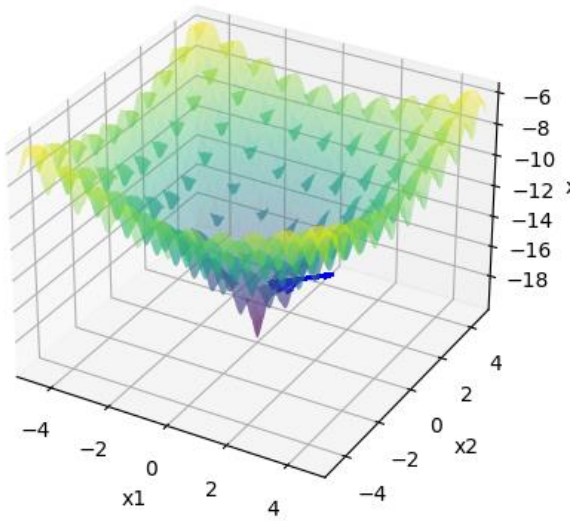
G is Riemannian metric tensor

Classical Gradient-descent on two functions: spherical has constant curvature (G=I) and Ackley. For first case the optimization vector can follow the curvature

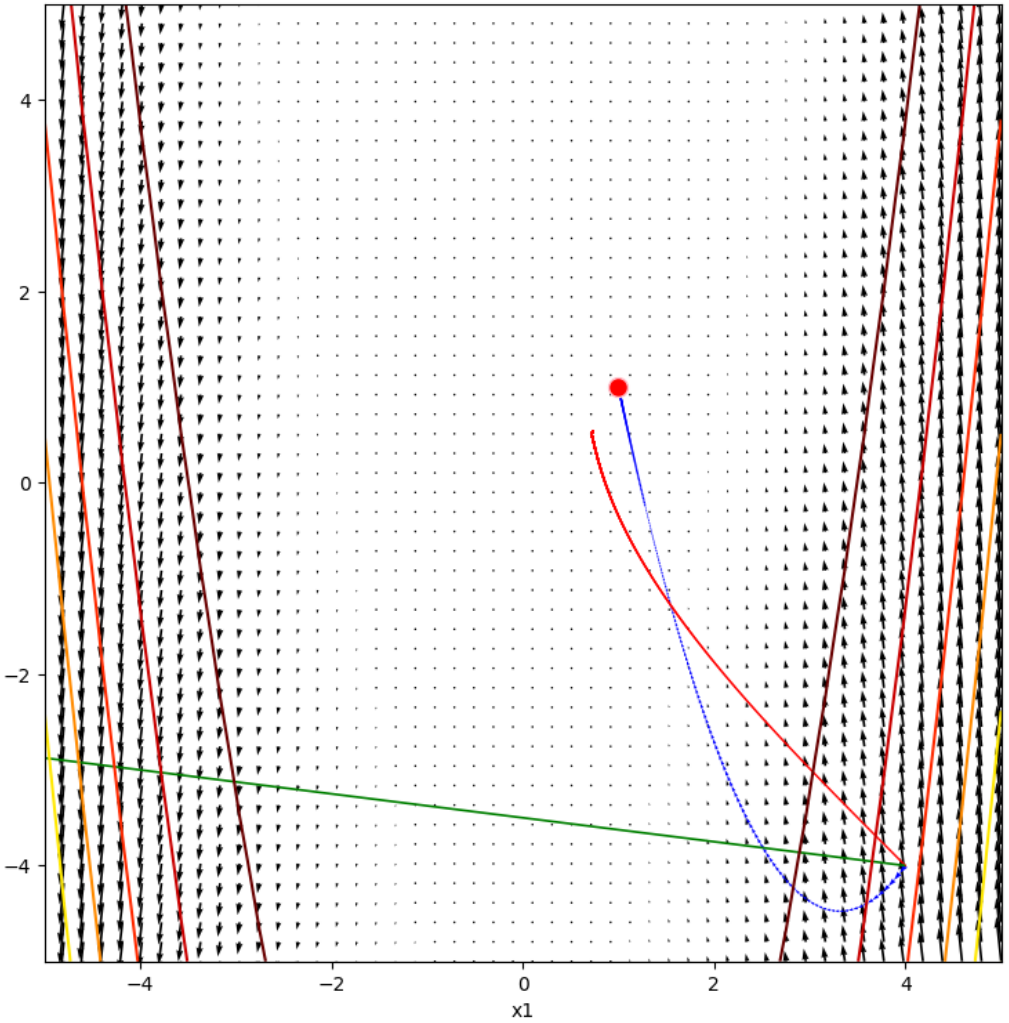
Spherical function: the G is an identity matrix for this function...
...and thus function gradient and natural gradient are same



Ackley function: G is no longer an identity matrix



Natural-gradient vs Classical Gradient vs Adam (Same lr and nitr)
Blue-NGD; red-Adam; green-GD



Natural GD vs ADAM vs GD

Natural gradients (2)

Chentsov's theorem: Fisher information metric is equivalent to Riemannian metric on statistical manifolds

$$F_{ij} = \frac{\partial^2 D_{KL}}{\partial\theta_i \partial\theta_j}$$

where D_{KL} represents KL-divergence

- Ideally, this method should work for DL applications since most of the losses are a special case of KL divergence
- However, calculating Hessian of KL divergence is not easy – matrix is singular or other issues
- Approximation methods exist but memory consumption is still an issue

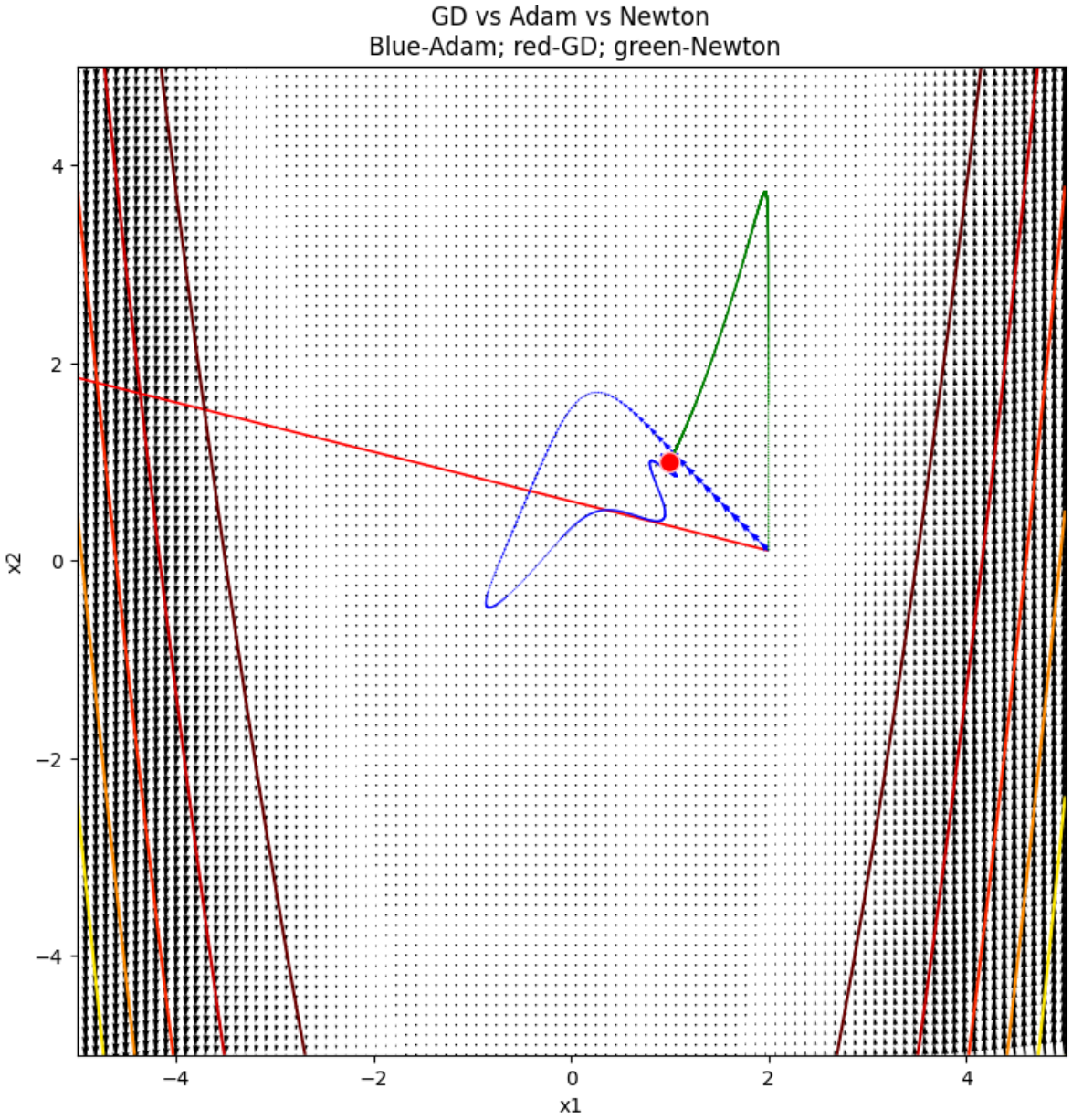
```
def opt_NGDT(f, xi, nitr=1000, eta=1e-3):
  it_ = 0
  xu = xi
  while it_ < nitr:
    y = f(xu)
    grad = torch.autograd.grad(y, xu, retain_graph=True, create_graph=True)
    input = torch.empty(grad[0].size(0), grad[0].size(0))
    Hess = torch.zeros_like(input)
    for i in range(grad[0].size(0)):
      for j in range(grad[0].size(0)):
        h1 = torch.autograd.grad(grad[0][i], xu, retain_graph=True, create_graph=True)[0]
        Hess[i, j] = torch.autograd.grad(h1[j], xu, retain_graph=True)[0][j]
        Hess[i, j] += 1e-9
    sv =  torch.matmul(Hess, grad[0])
    with torch.no_grad():
      xu -= eta*sv
    it_ += 1
  return xu
```

Hessain based methods

$$f(w+\Delta w) \approx f(w) + \Delta w^T \nabla f(w) + \frac{1}{2}\Delta w^T H(w) \Delta w$$

$$w(t+1) = w(t) - \eta H^{-1} g$$

- **Second order methods tend to converge faster**
- **They are more robust close to saddle points**
- **However, they suffer from higher memory consumption**



L-BFGS (Limited-memory Broyden-Fletcher-Goldfarb-Shanno)

Initialization: Start with an initial solution vector and set the initial approximation of the Hessian matrix as an identity matrix or a diagonal matrix.

Evaluation: Compute the gradient of the objective function at the current solution point.

Direction computation: Multiply the gradient by the current approximation of the Hessian matrix to obtain the search direction. This step makes L-BFGS a quasi-Newton method, as it approximates the Hessian using gradient information.

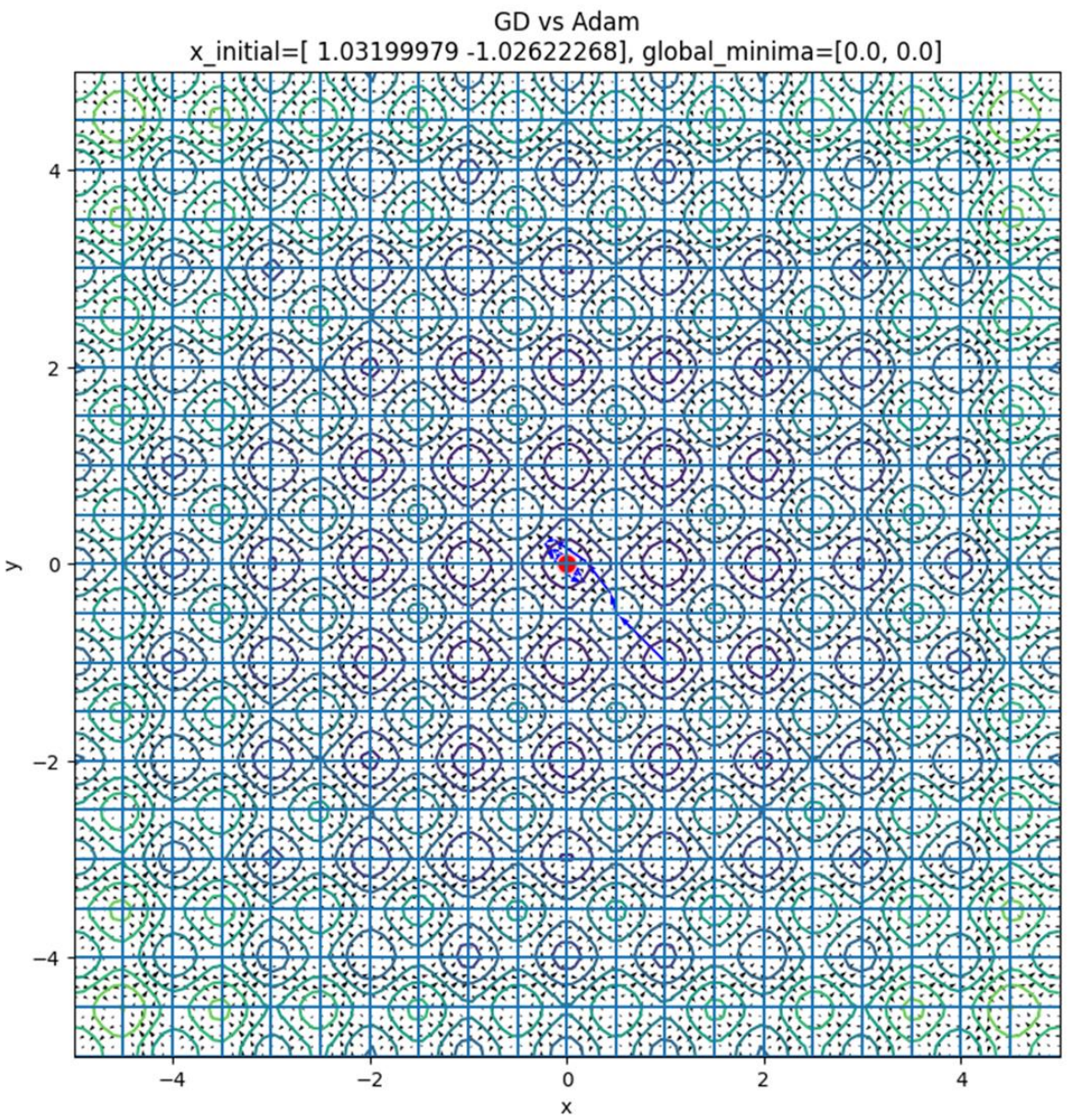
Line search: Perform a line search to find an optimal step size in the search direction. This step ensures that the objective function decreases sufficiently.

Update: Use the information from the line search to update the solution vector.

Curvature update: Update the approximation of the Hessian matrix using information from the previous iterations. This step is what distinguishes L-BFGS from other optimization algorithms.

Convergence check: Check if the convergence criteria are met. If not, go back to step 2 and repeat the process until convergence is achieved

Latin Hypercube sampling



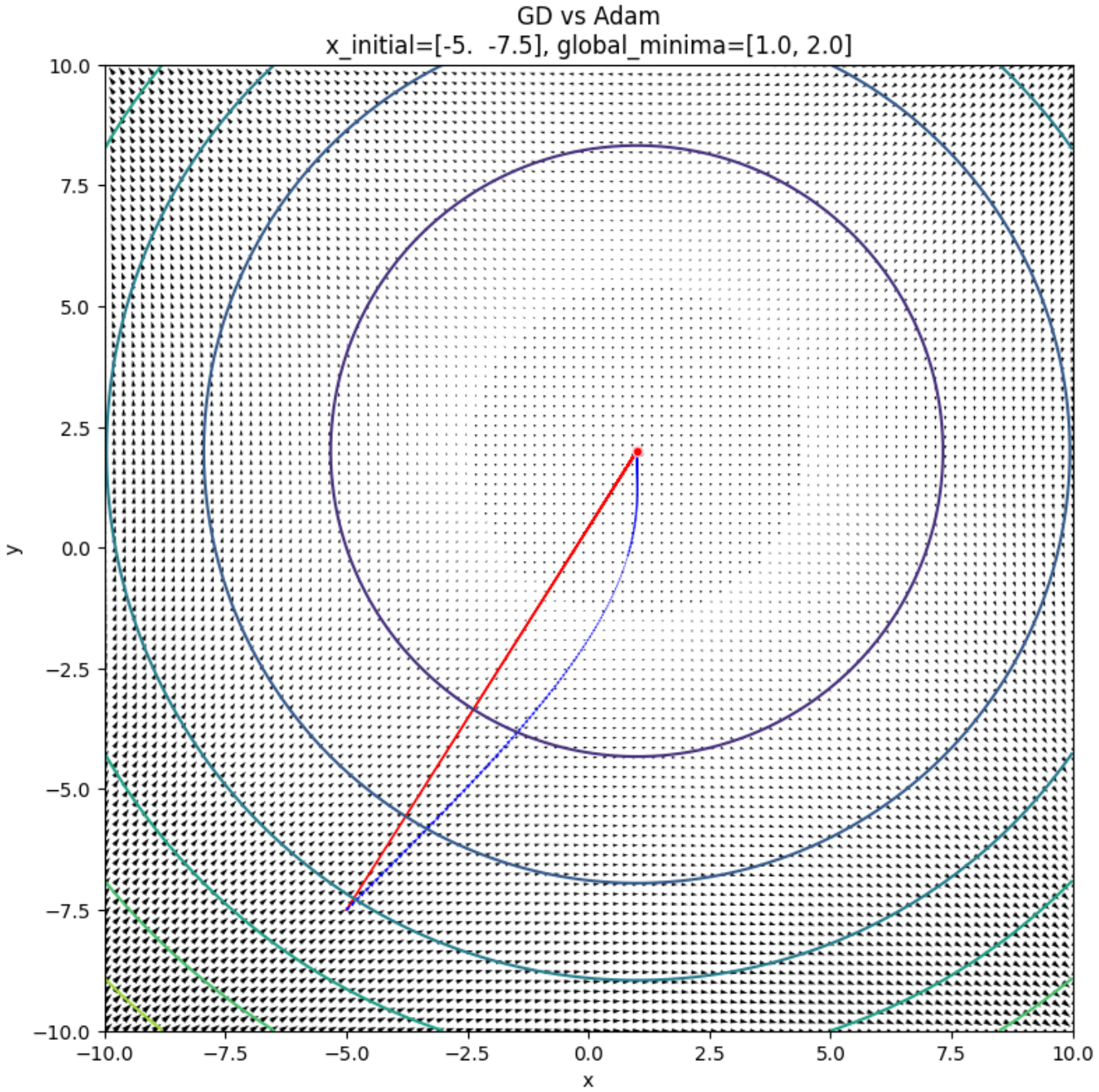
Adam	
Nitr	10000
Lr	0.5
betas	[0.9, 0.999]
eps	1e-7
x_initial	[-1, -1]
x_final	[-0.01154259, -0.00684768]

$x \in \square^d$

$x_i \in \square \sim U(lb_i, ub_i) \forall i \in \{1, 2, .., d\}$

- **Assume parameter space is limited by certain bounds**
- **Divide the regions in N "hypercube"**
- **Sample each point independently**
- **Points can be used to approximate initial guess ot run a multi-start optimization**

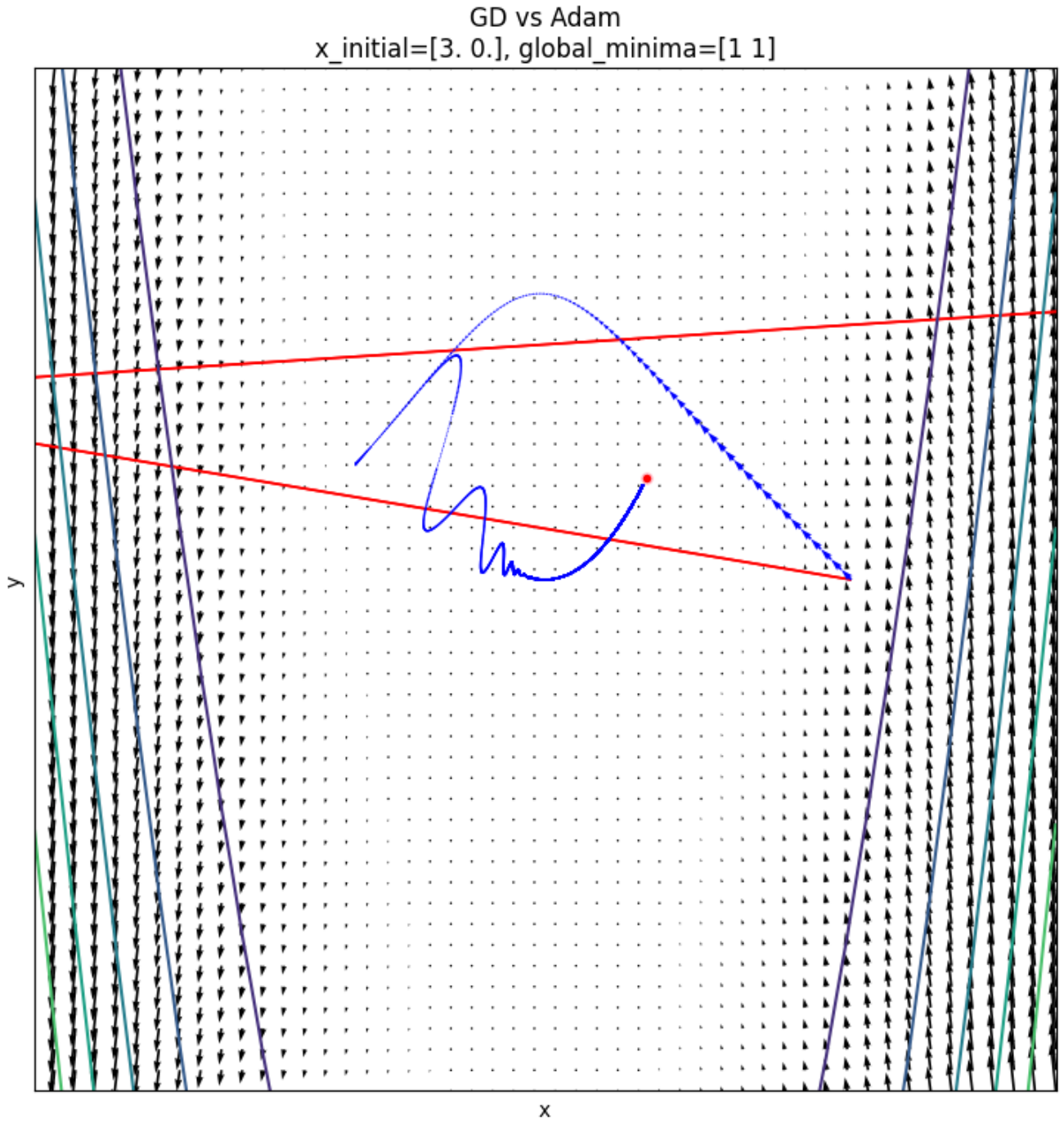
Appendix



Gradient descent	
Nitr	10000
Lr	1e-3
x_initial	[-5, -7.5]
x_final	[0.99999999, 1.99999998]

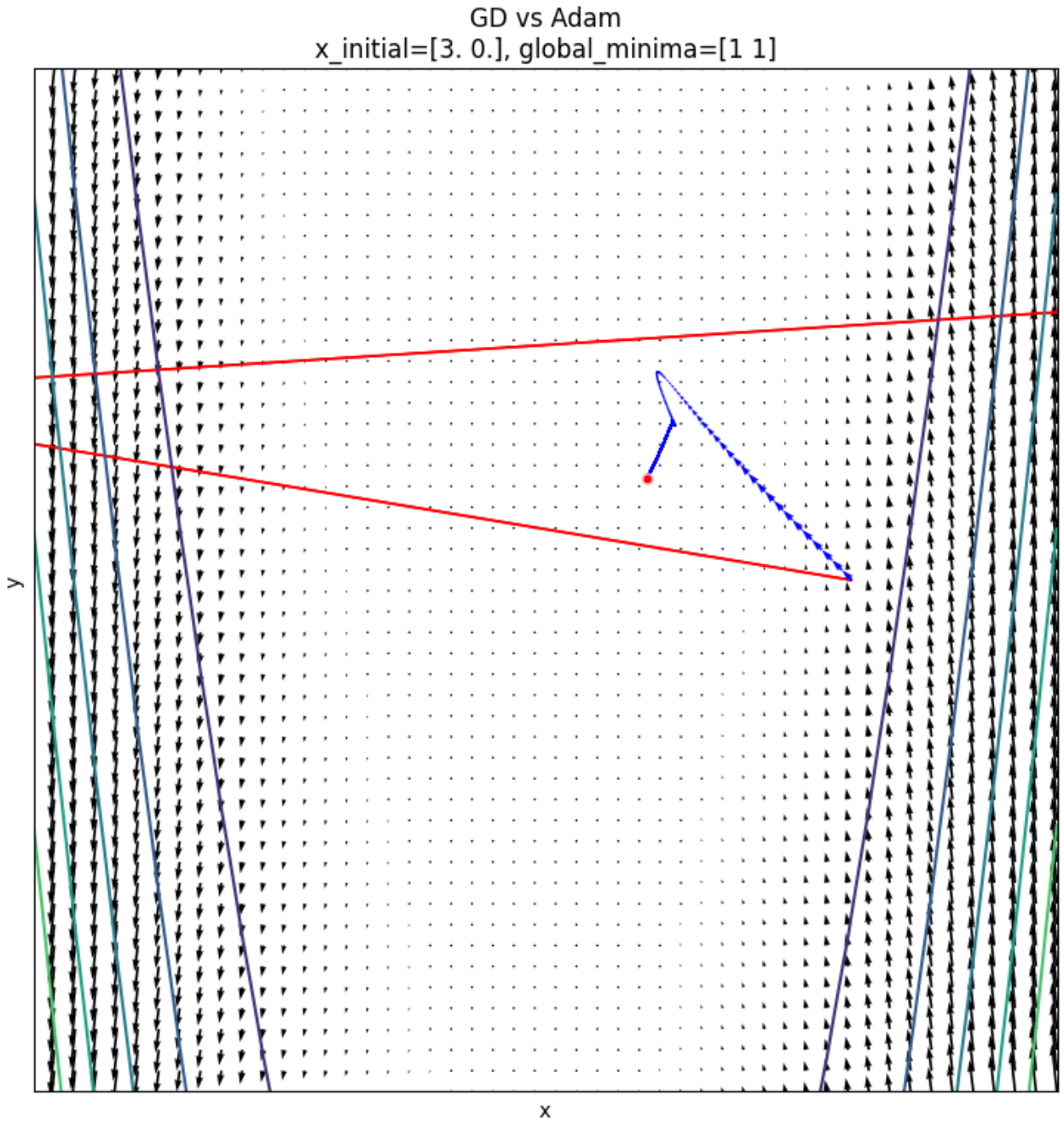
Adam	
Nitr	10000
Lr	1e-1
betas	[0.99, 0.9999]
eps	1e-7
x_initial	[-5, -7.5]
x_final	[1., 2.]

Rosenbrock



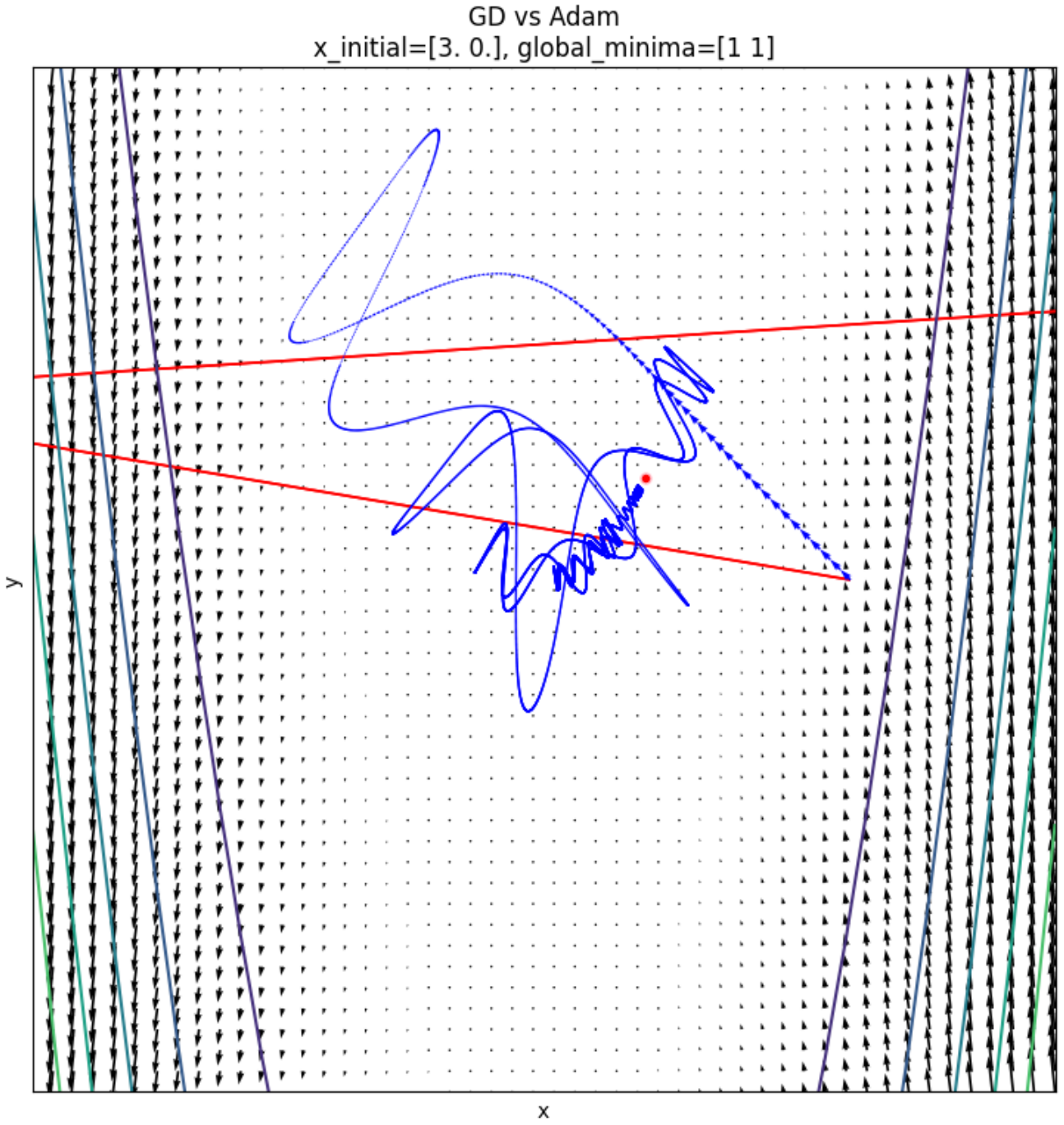
Gradient descent	
Nitr	10000
Lr	1e-3
x_initial	[3, 0]
x_final	[4.23722310e+18, 6.25647912e+03]

Adam	
Nitr	10000
Lr	1e-1
betas	[0.99, 0.9999]
eps	1e-7
x_initial	[3, 0]
x_final	[0.96279886 , 0.92688621]



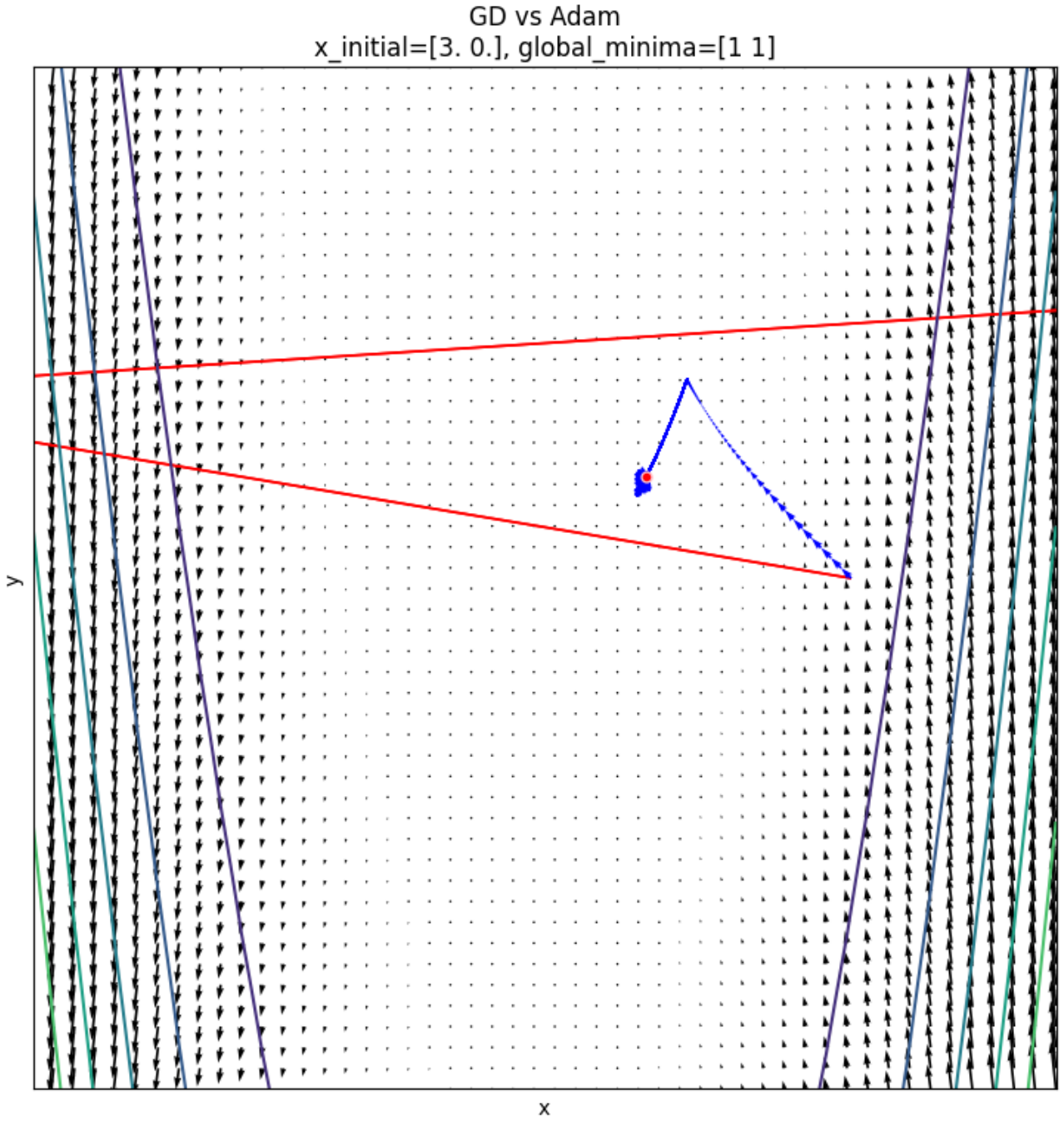
Gradient descent	
Nitr	10000
Lr	1e-3
x_initial	[3, 0]
x_final	[4.23722310e+18, 6.25647912e+03]

Adam	
Nitr	10000
Lr	1e-1
betas	[0.9, 0.999]
eps	1e-7
x_initial	[3, 0]
x_final	[1 , 1]



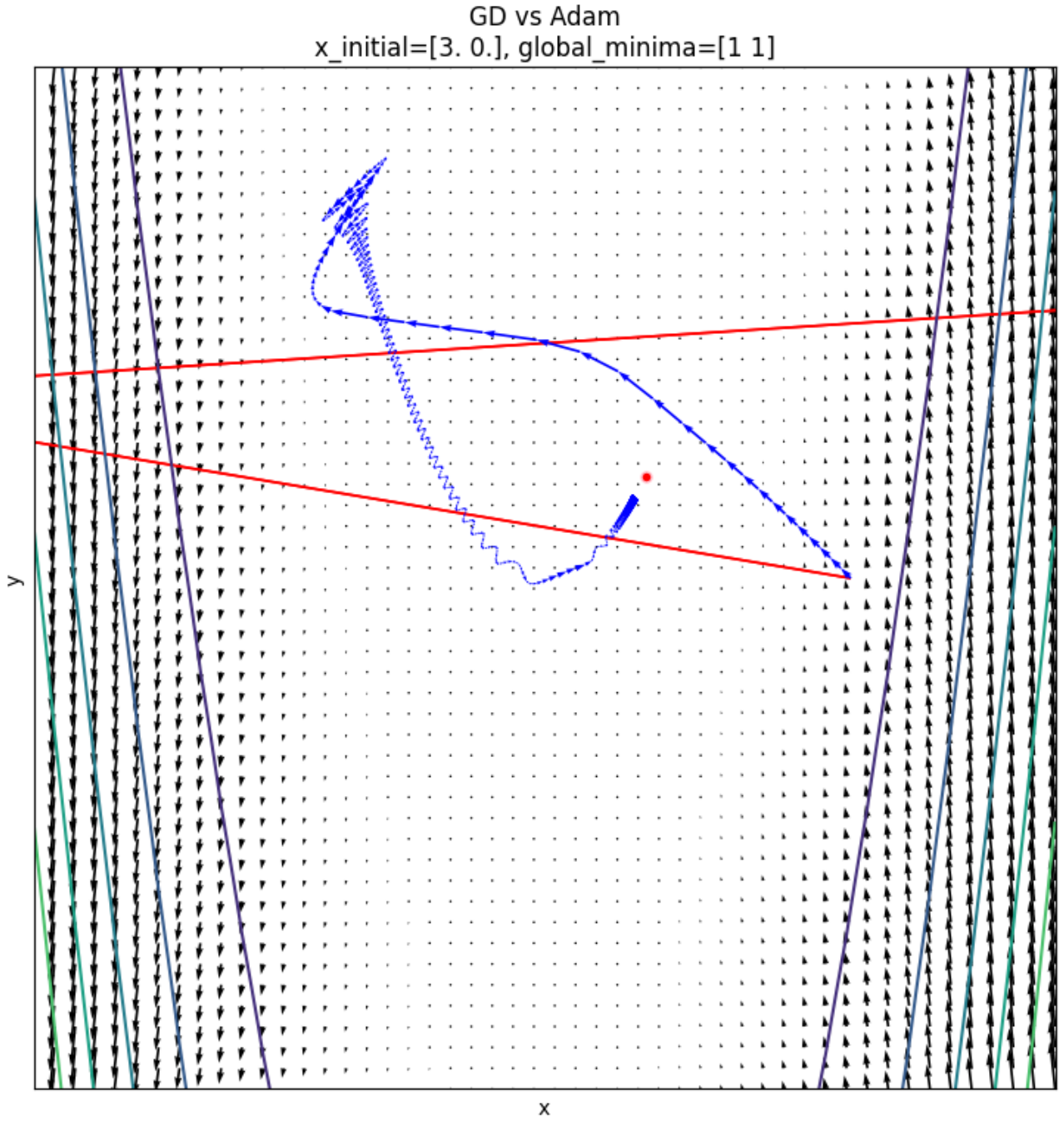
Gradient descent	
Nitr	10000
Lr	1e-3
x_initial	[3, 0]
x_final	[4.23722310e+18, 6.25647912e+03]

Adam	
Nitr	10000
Lr	1e-1
betas	[0.999, 0.999]
eps	1e-7
x_initial	[3, 0]
x_final	[0.89075591, 0.82099768]



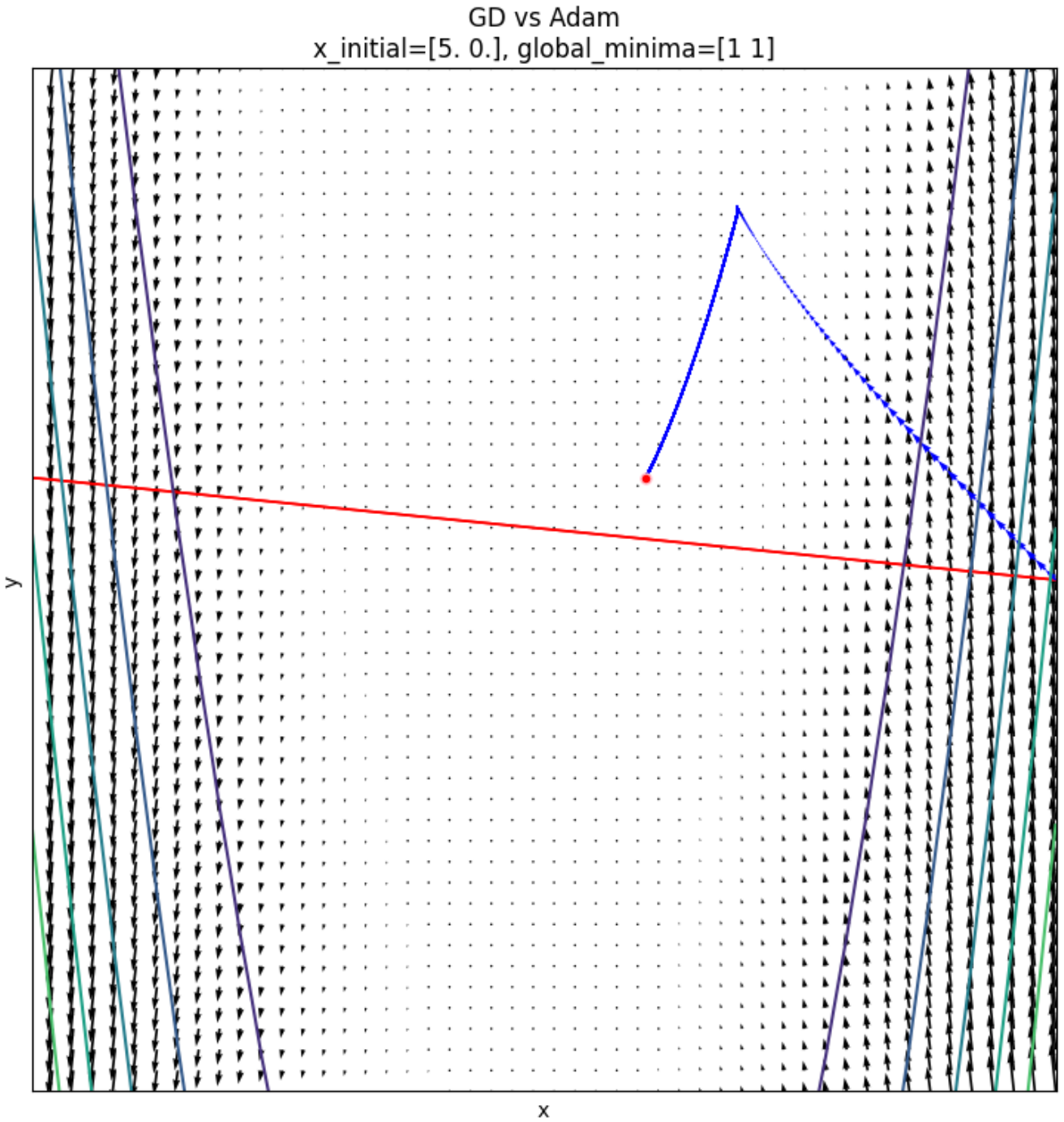
Gradient descent	
Nitr	10000
Lr	1e-3
x_initial	[3, 0]
x_final	[4.23722310e+18, 6.25647912e+03]

Adam	
Nitr	10000
Lr	1e-1
betas	[0.7, 0.999]
eps	1e-7
x_initial	[3, 0]
x_final	[0.98250643, 0.96526179]



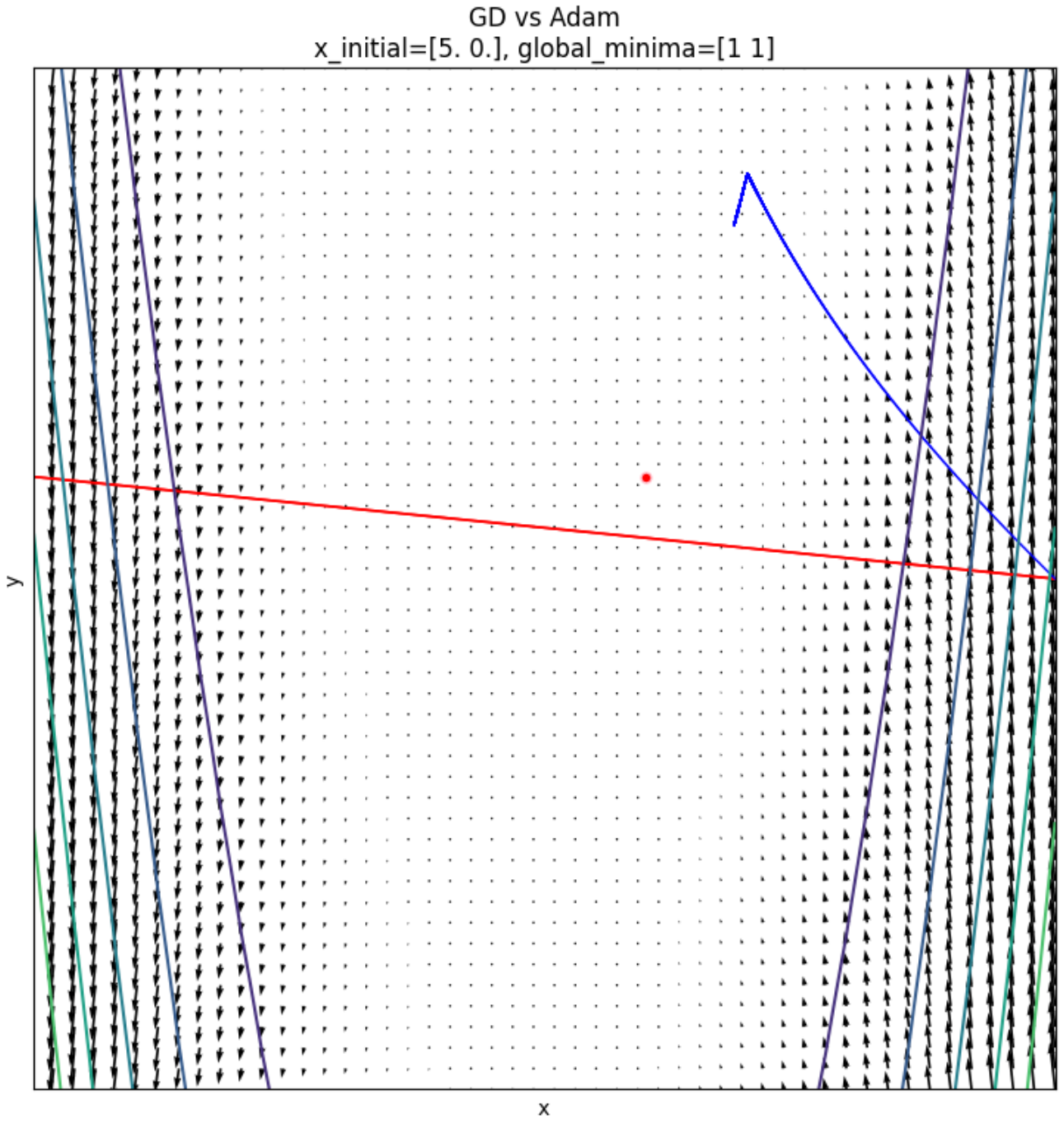
Gradient descent	
Nitr	10000
Lr	1e-3
x_initial	[3, 0]
x_final	[4.23722310e+18, 6.25647912e+03]

Adam	
Nitr	10000
Lr	1e-1
betas	[0.9, 0.5]
eps	1e-7
x_initial	[3, 0]
x_final	[0.86305156, 0.80067135]



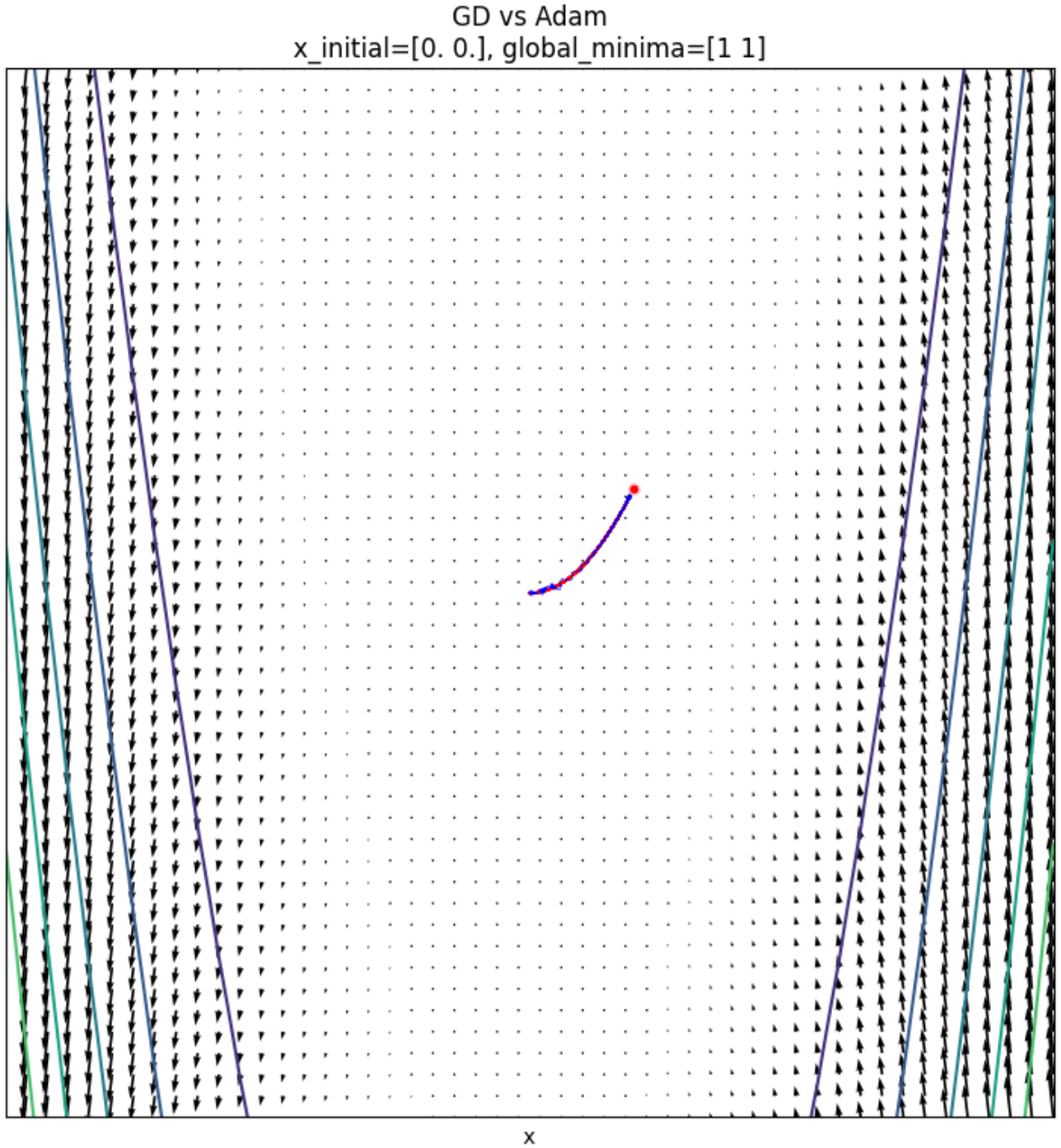
Gradient descent	
Nitr	10000
Lr	1e-3
x_initial	[5, 0]
x_final	[-1.91897874e+13, 4.09146131e+02]

Adam	
Nitr	10000
Lr	1e-1
betas	[0.9, 0.999]
eps	1e-7
x_initial	[5, 0]
x_final	[1.0001373 1.00027483]



Gradient descent	
Nitr	10000
Lr	1e-3
x_initial	[3, 0]
x_final	[-1.91897874e+13, 4.09146131e+02]

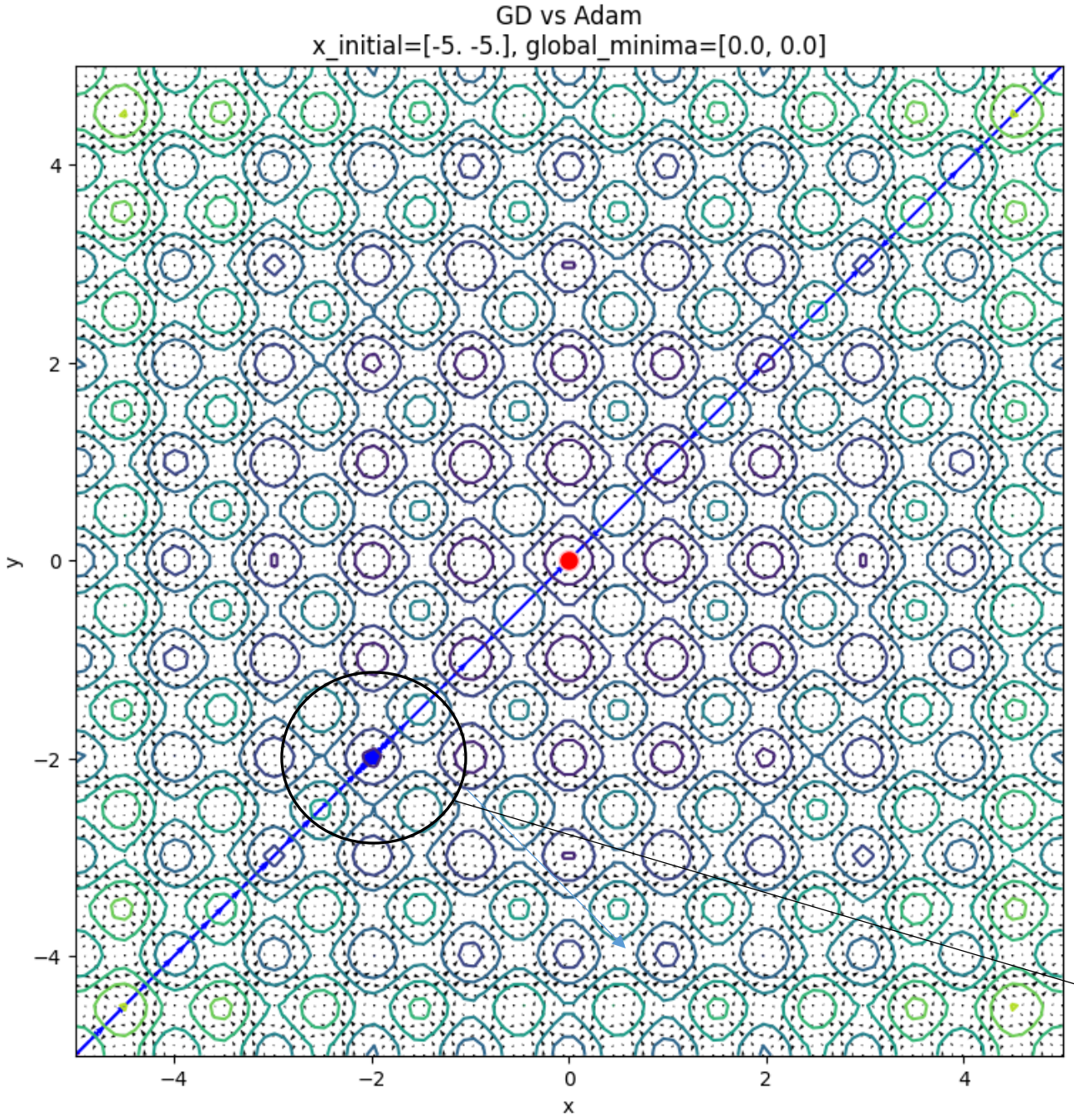
Adam	
Nitr	10000
Lr	1e-2
betas	[0.9, 0.999]
eps	1e-7
x_initial	[5, 0]
x_final	[1.86150594, 3.46665981]



Gradient descent	
Nitr	10000
Lr	1e-3
x_initial	[3, 0]
x_final	[0.99440095, 0.98881076]

Adam	
Nitr	10000
Lr	1e-1
betas	[0.9, 0.999]
eps	1e-7
x_initial	[5, 0]
x_final	[0.99993208, 0.99986374]

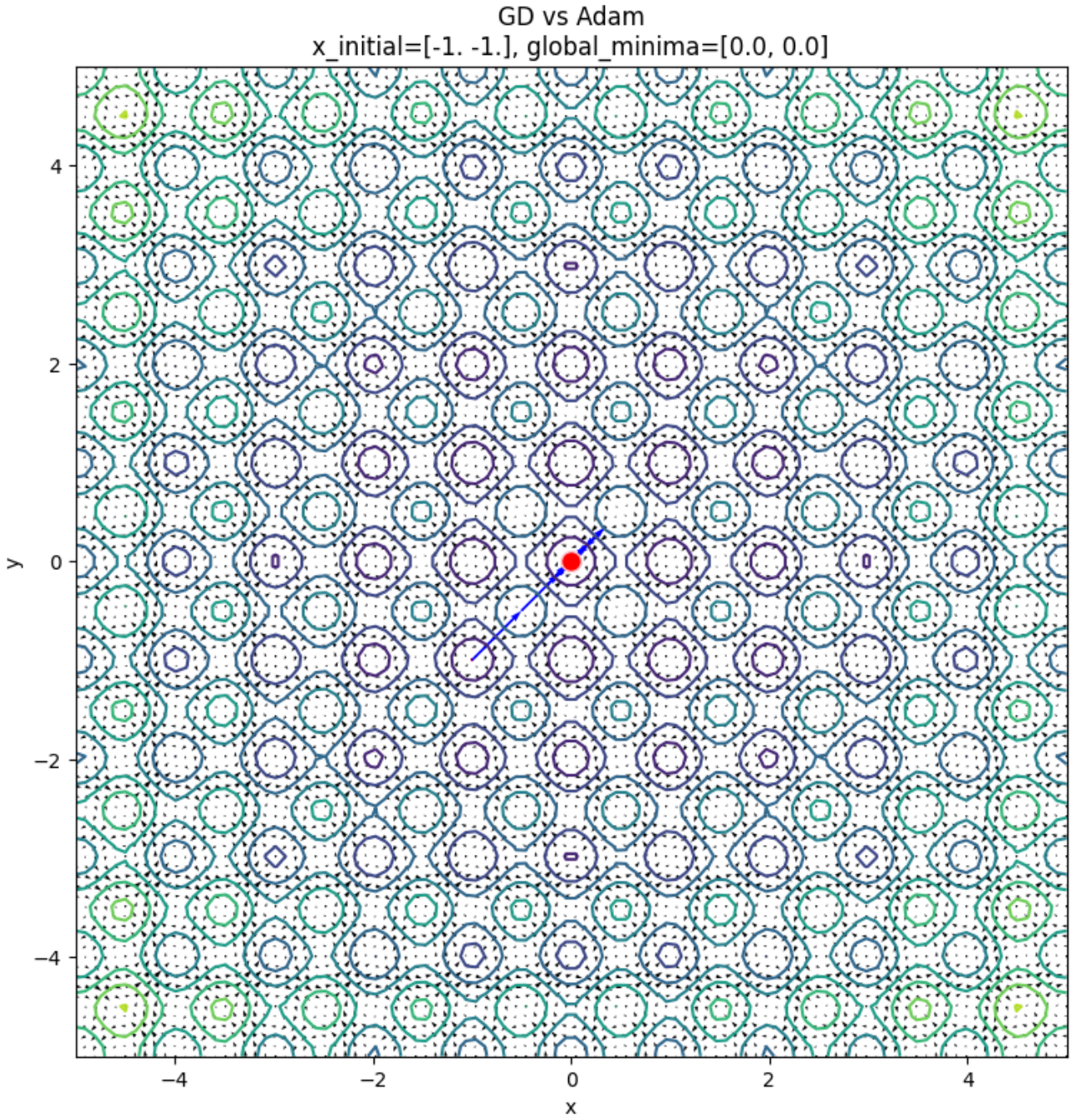
Rastrigin



Gradient descent	
Nitr	10000
Lr	1e-3
x_initial	[-5, -5]
x_final	[-4.97469139, -4.97469139]

Adam	
Nitr	10000
Lr	1
betas	[0.9, 0.999]
eps	1e-7
x_initial	[-5, -5]
x_final	[-1.99923524, -1.99755086]

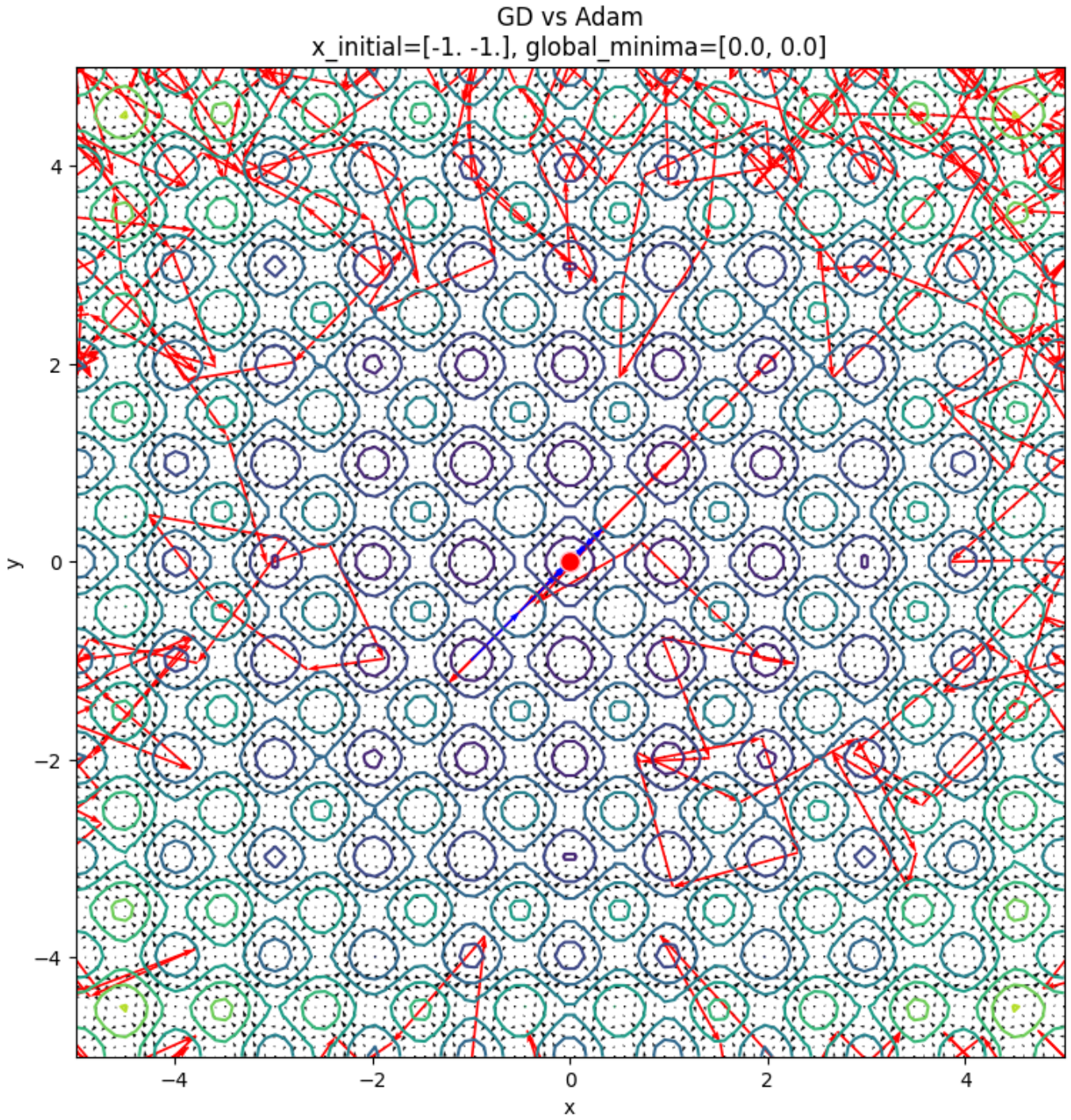
Got pushed out of a local minima



Gradient descent	
Nitr	10000
Lr	1e-3
x_initial	[-1, -1]
x_final	[-0.99495864, -0.99495864]

Adam	
Nitr	10000
Lr	0.5
betas	[0.9, 0.999]
eps	1e-7
x_initial	[-1, -1]
x_final	[0.01386211, 0.02081304]

A good initial point with appropriate Lr; GD, however, doesn't move

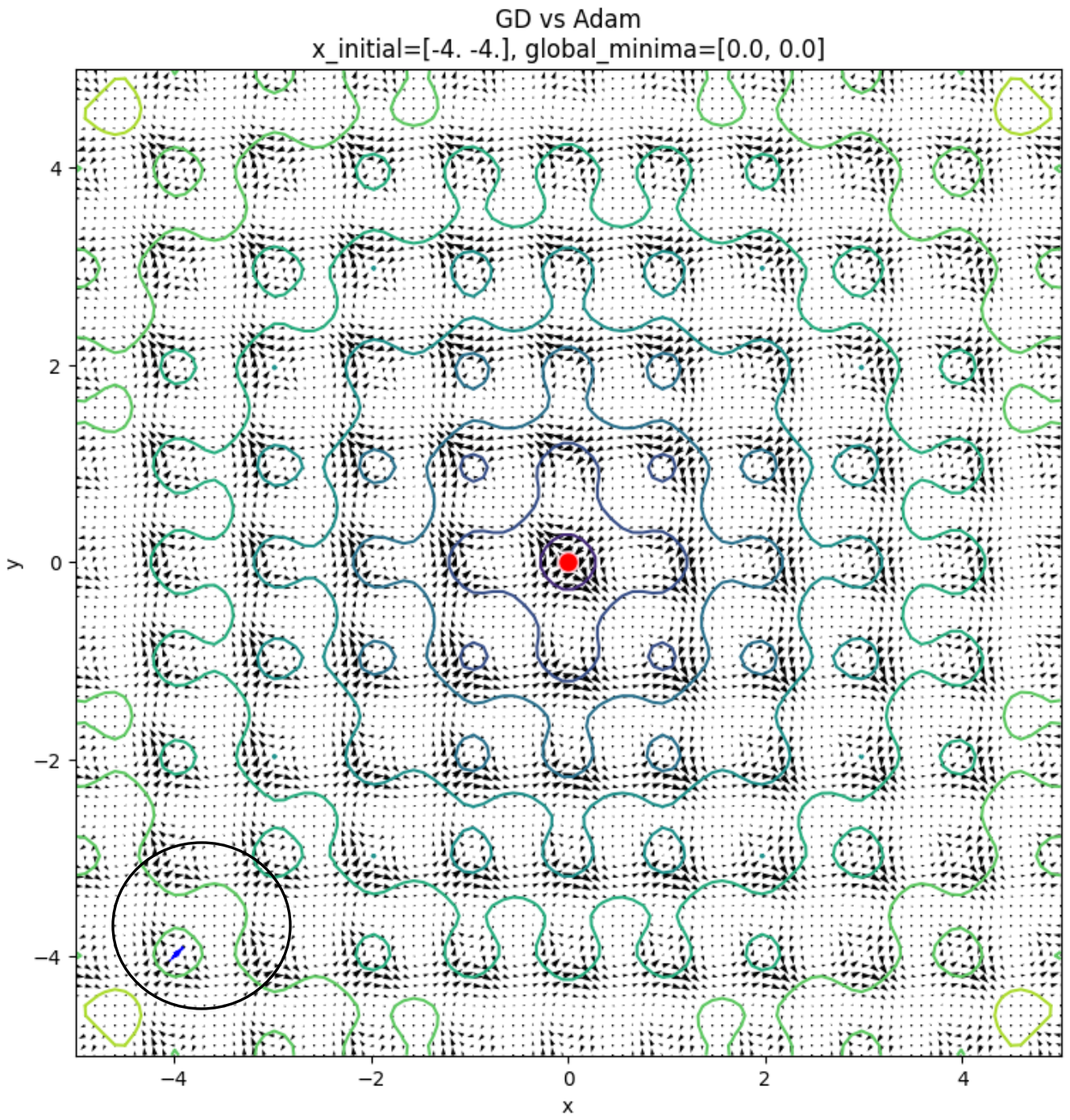


Gradient descent	
Nitr	10000
Lr	2e-2
x_initial	[-1, -1]
x_final	8.62994166, -8.96991123]

Adam	
Nitr	10000
Lr	0.5
betas	[0.9, 0.999]
eps	1e-7
x_initial	[-1, -1]
x_final	[0.01386211, 0.02081304]

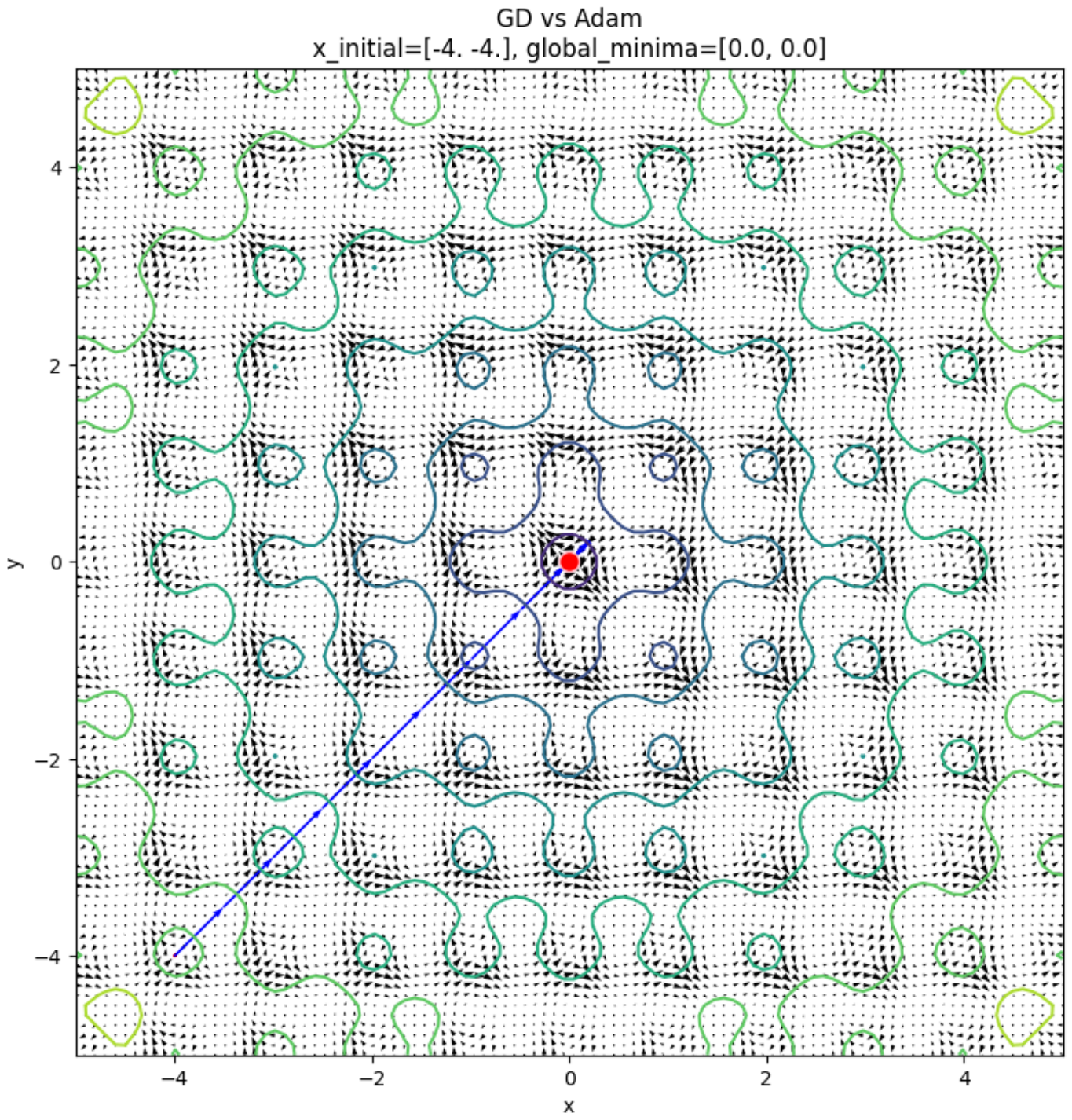
GD w/ large learning rate – jumps from minima to minima

Ackley



Gradient descent	
Nitr	10000
Lr	1e-3
x_initial	[-5, -5]
x_final	[-3.9830677, -3.9830677]

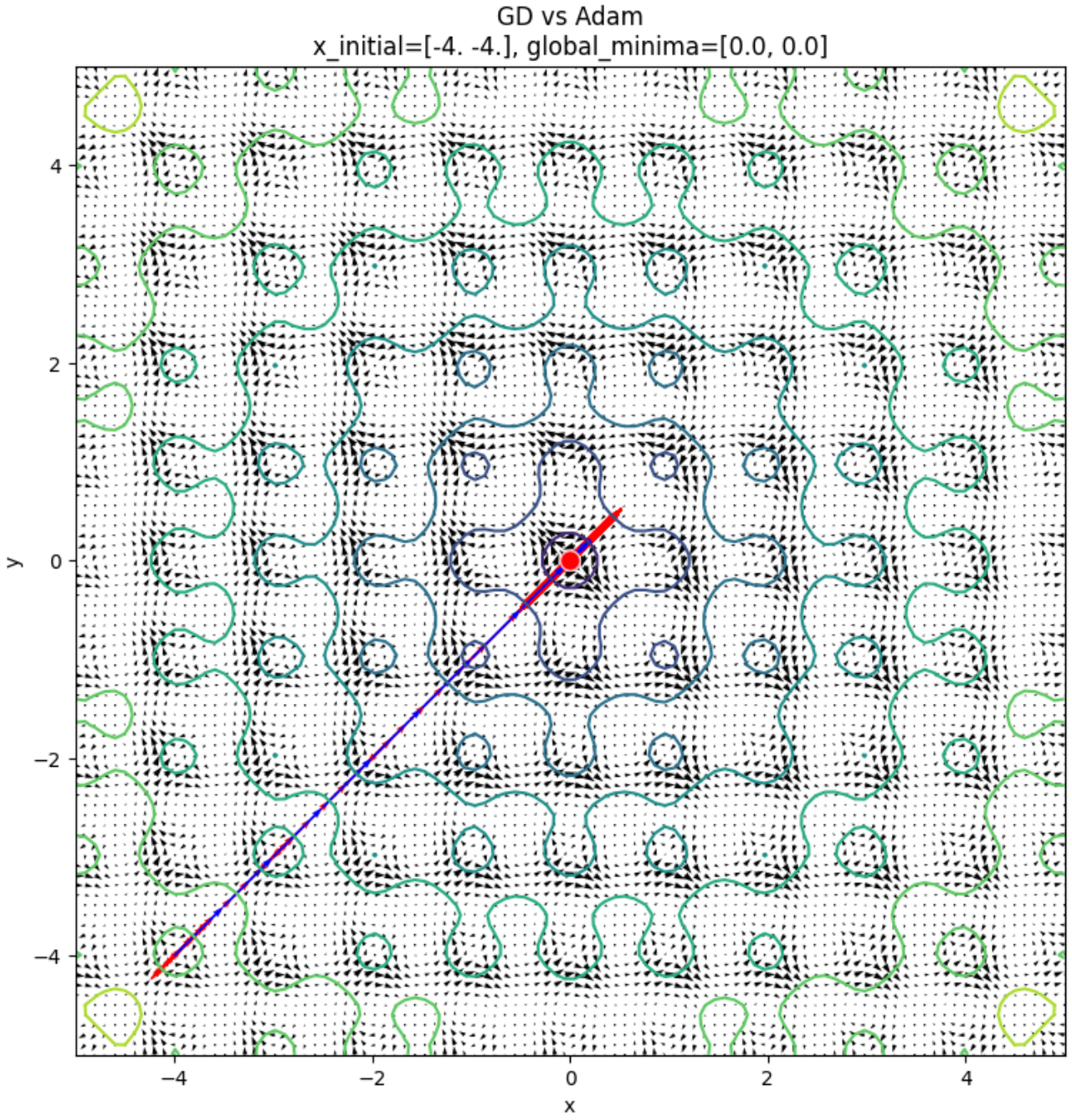
Adam	
Nitr	10000
Lr	1e-1
betas	[0.9, 0.999]
eps	1e-7
x_initial	[-5, -5]
x_final	[-3.98493574, -3.98493574]



Gradient descent	
Nitr	10000
Lr	1e-3
x_initial	[-5, -5]
x_final	[-3.9830677, -3.9830677]

Adam	
Nitr	10000
Lr	0.5
betas	[0.9, 0.999]
eps	1e-7
x_initial	[-5, -5]
x_final	[0.02544463, 0.02544463]

Very close to global minima…but it's almost not feasible for it to settle at global minima



Gradient descent	
Nitr	10000
Lr	0.1
x_initial	[-5, -5]
x_final	[0.34391327, 0.34391327]

Adam	
Nitr	10000
Lr	0.5
betas	[0.9, 0.999]
eps	1e-7
x_initial	[-5, -5]
x_final	[0.02544463, 0.02544463]

GD also – as expected – improves with large Lr