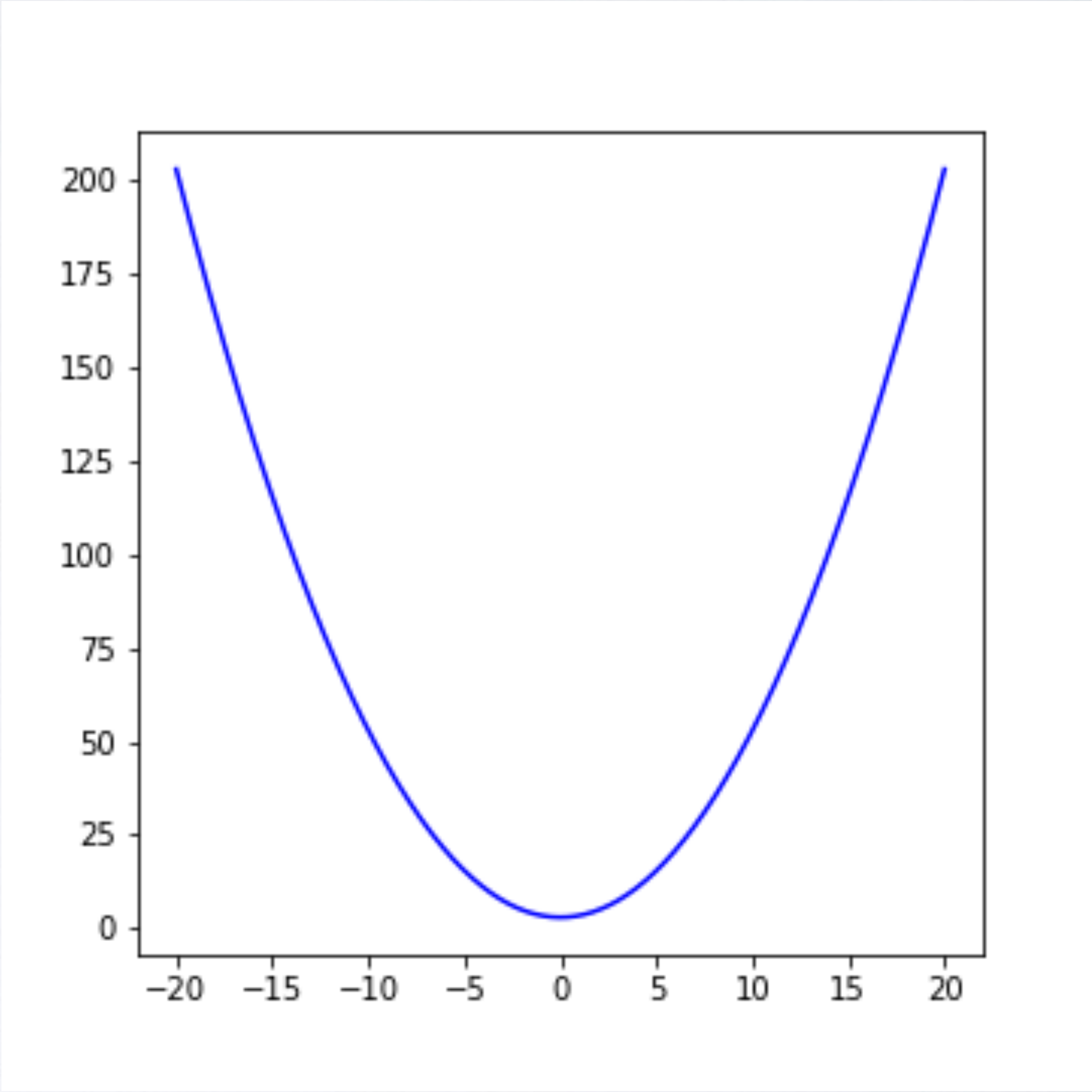
GRADIENT DESCENT

Inzamam Rahaman

OPTIMISATION

- Given some function (or functions), find the point(s) where the max or min occurs
  - Function to minimise: loss function
  - Function to maximise: objective function
  - Can convert minimisation to maximisation
- Important on its own and as a component of other computational tasks
  - Machine Learning
  - Maximum-Likelihood Estimation
  - Fitting HBMs
  - etc…



TECHNIQUES

- Many different scenarios
  - Discrete vs continuous
  - Constrained vs unconstrained
  - Convex vs non-convex
- Different techniques for different scenarios

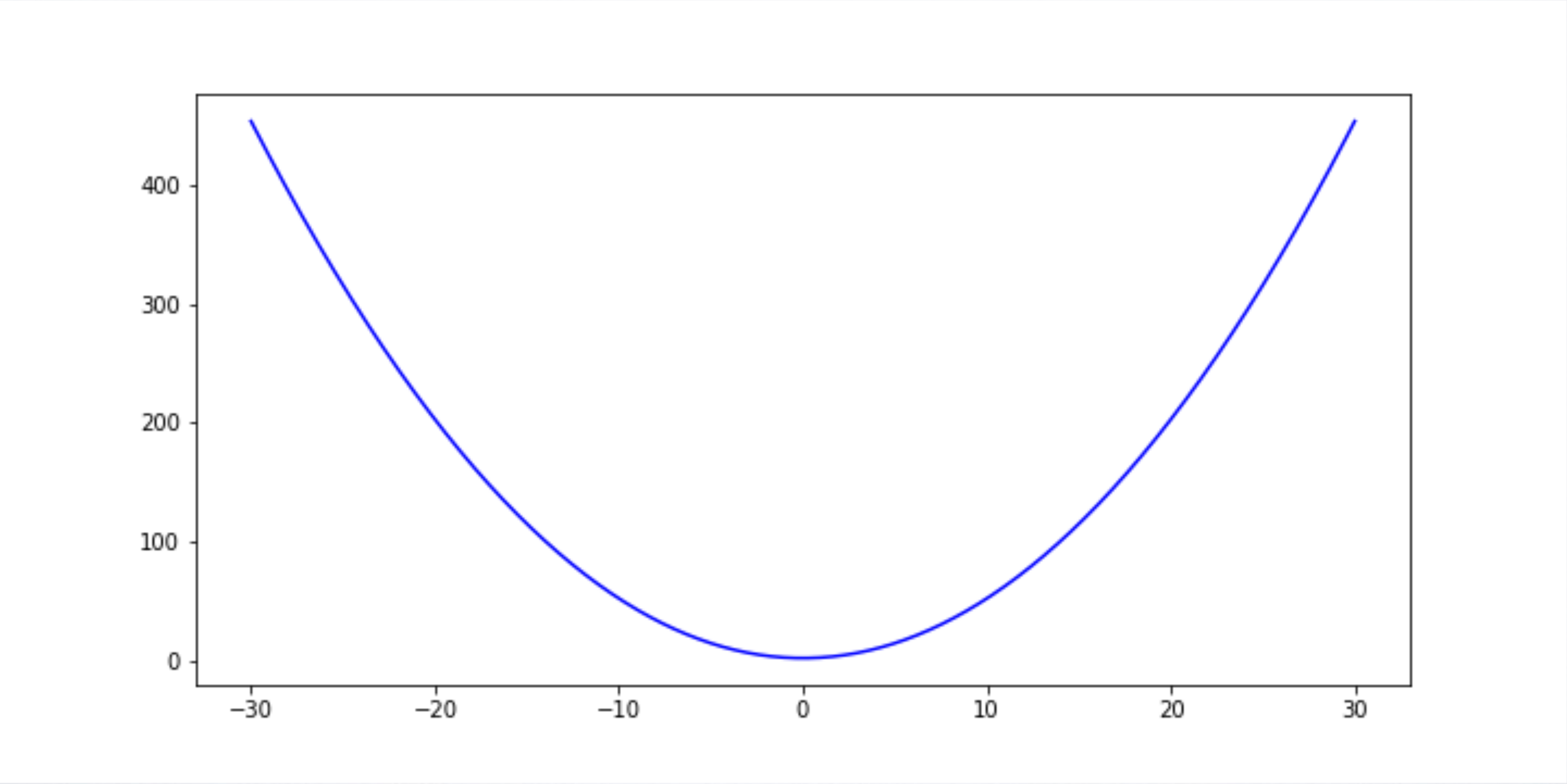
GRADIENT DESCENT

- Optimisation technique for continuous, unconstrained, convex* optimisation
- 1st order method - requires computation of first derivative of **loss** function

* - can be used in non-convex cases and work "well" -but we lose formal guarantees about quality of solution

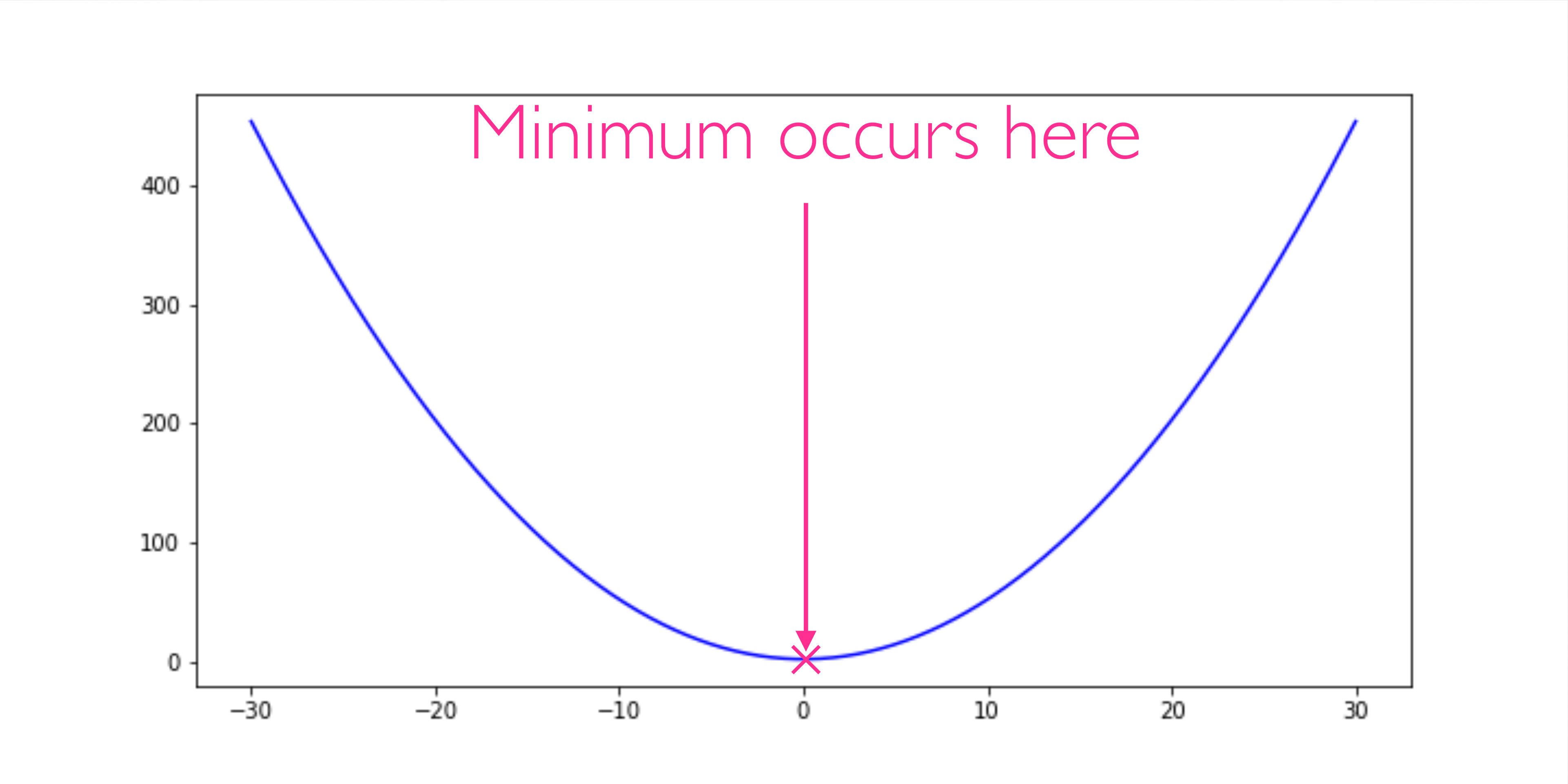
CORE INTUITION OF GRADIENT DESCENT

- Suppose that we have the function $f(x) = \frac{1}{2}x^2 + 3$



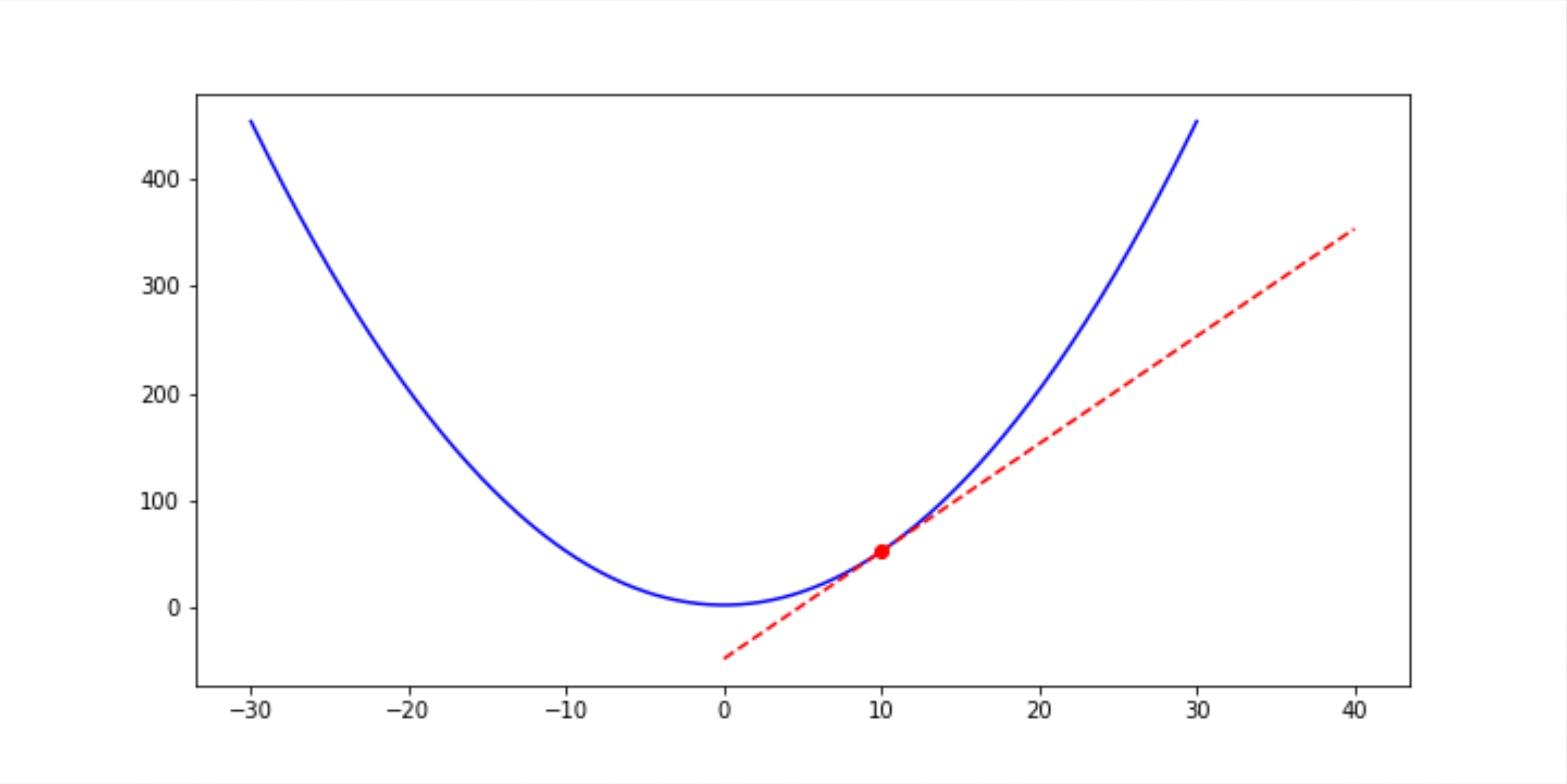
CORE INTUITION OF GRADIENT DESCENT

- Suppose that we have the function $f(x) = \dfrac{1}{2}x^2 + 3$



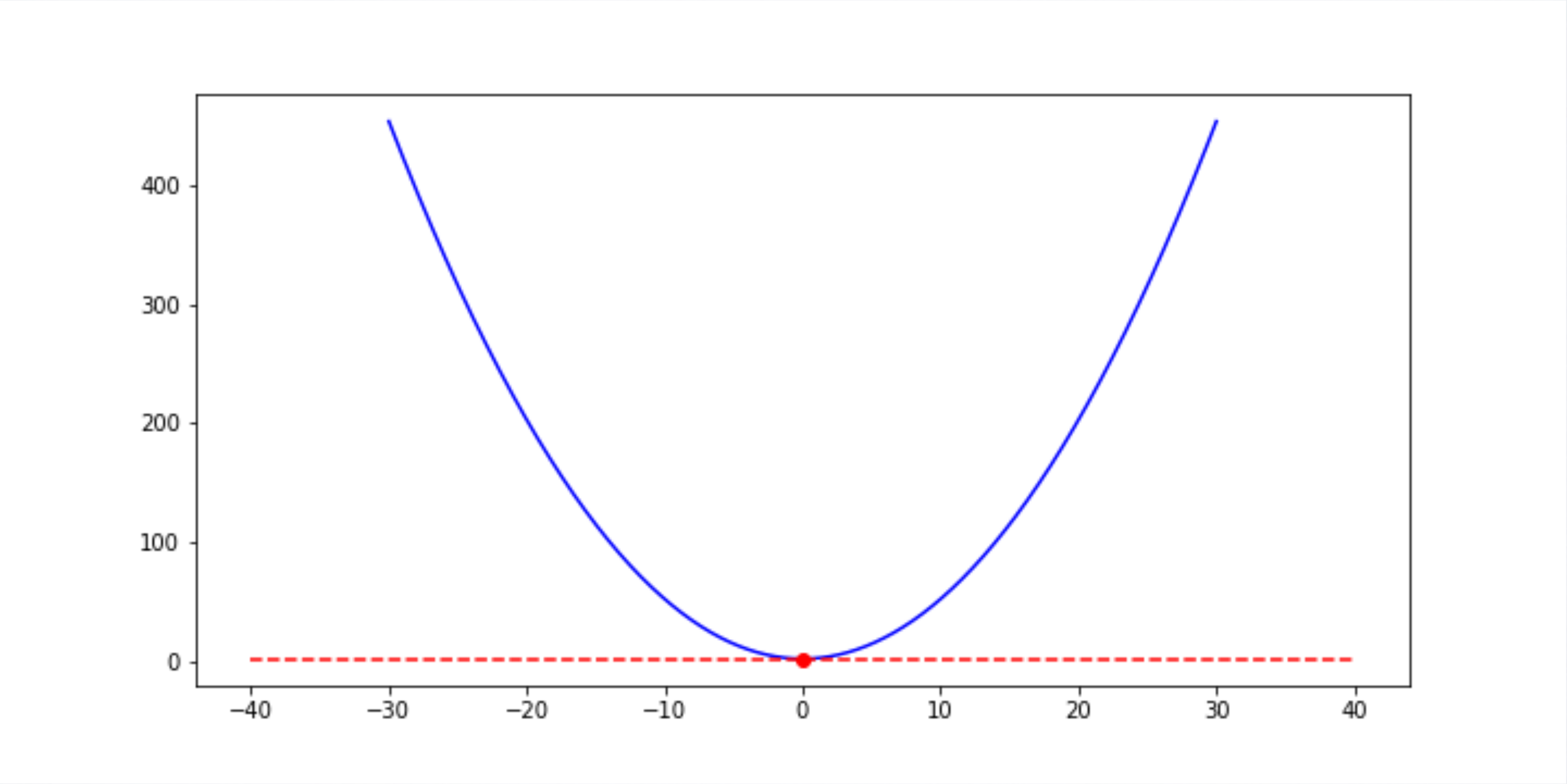
CORE INTUITION OF GRADIENT DESCENT

- From calculus, the derivative gives the gradient of the tangent to the curve s.t. $\frac{df}{dx} = x$ describes the gradient of the tangent line at any point



CORE INTUITION OF GRADIENT DESCENT

- Optimal occurs at "saddle" point, i.e. where $\frac{df}{dx} = 0$

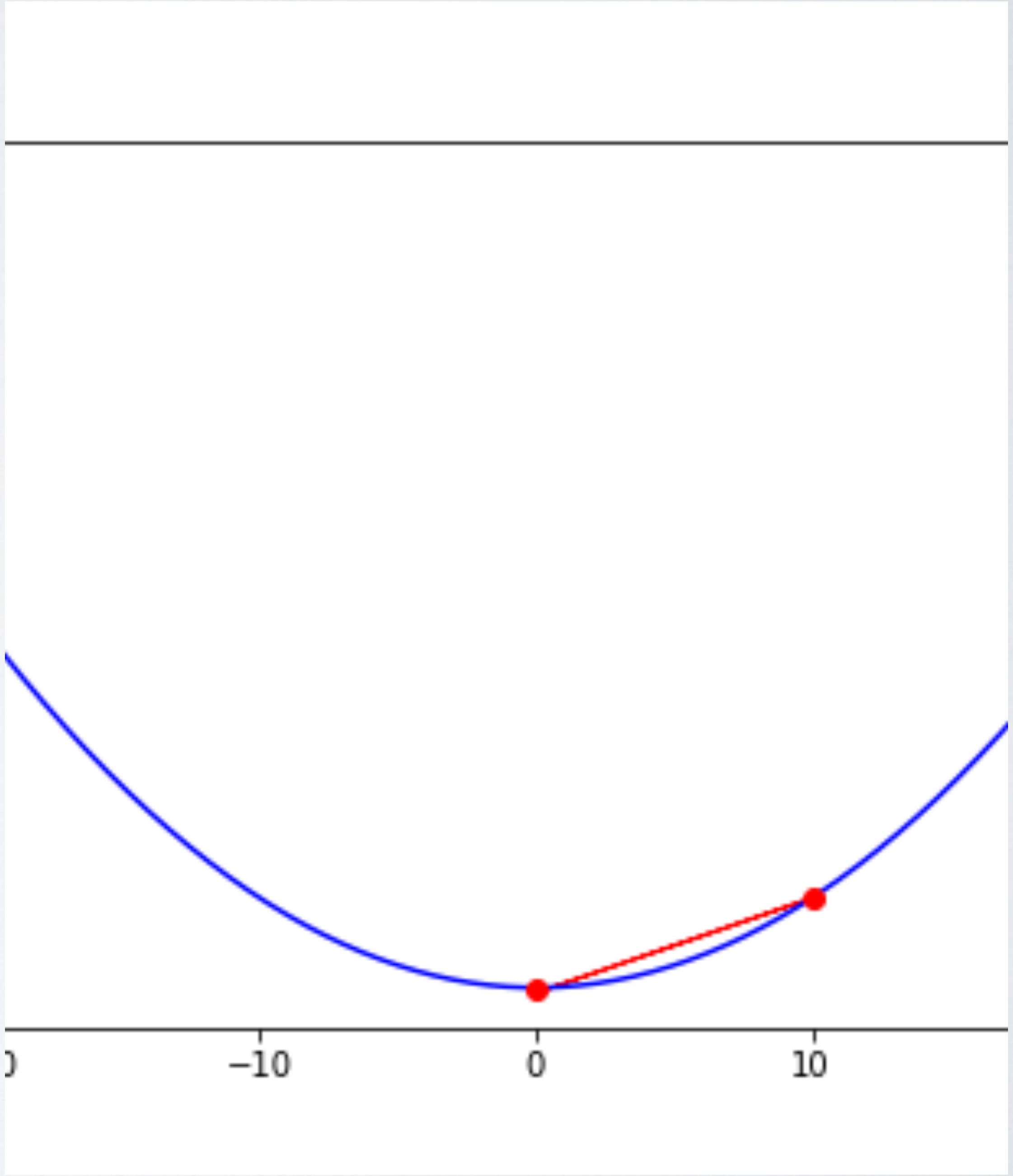


GRADIENT DESCENT

- Suppose that we can't find an analytical closed-form solution to $\frac{df}{dx} = 0$, can we still use the gradient to find the minimum?
  - Yes!
- Gradient gives direction of steepest **ascent**
  - Move in opposite direction to descend towards minimum

GRADIENT DESCENT

- Let's choose a random point, say $x_0 = 10$.
- Let's keep moving in the direction opposite to the gradient
  - At $x_0 = 10$, $\dfrac{df}{dx} = 10$
  - Move in direction opposite to $\dfrac{df}{dx}$
  - So $x_1 = x_0 - \dfrac{df}{dx} = 0$
  - At this point, gradient is 0
  - Continue until convergence

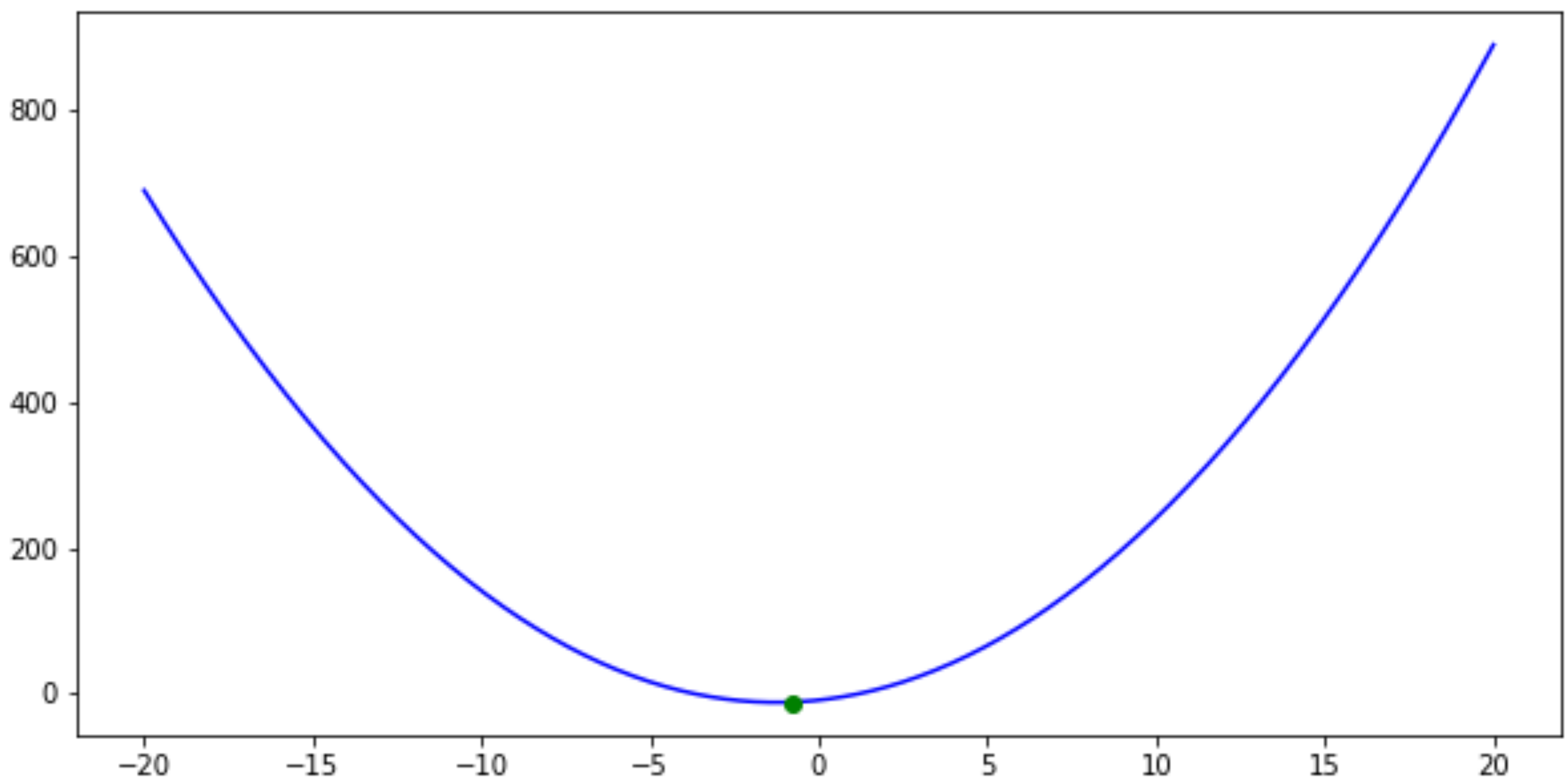


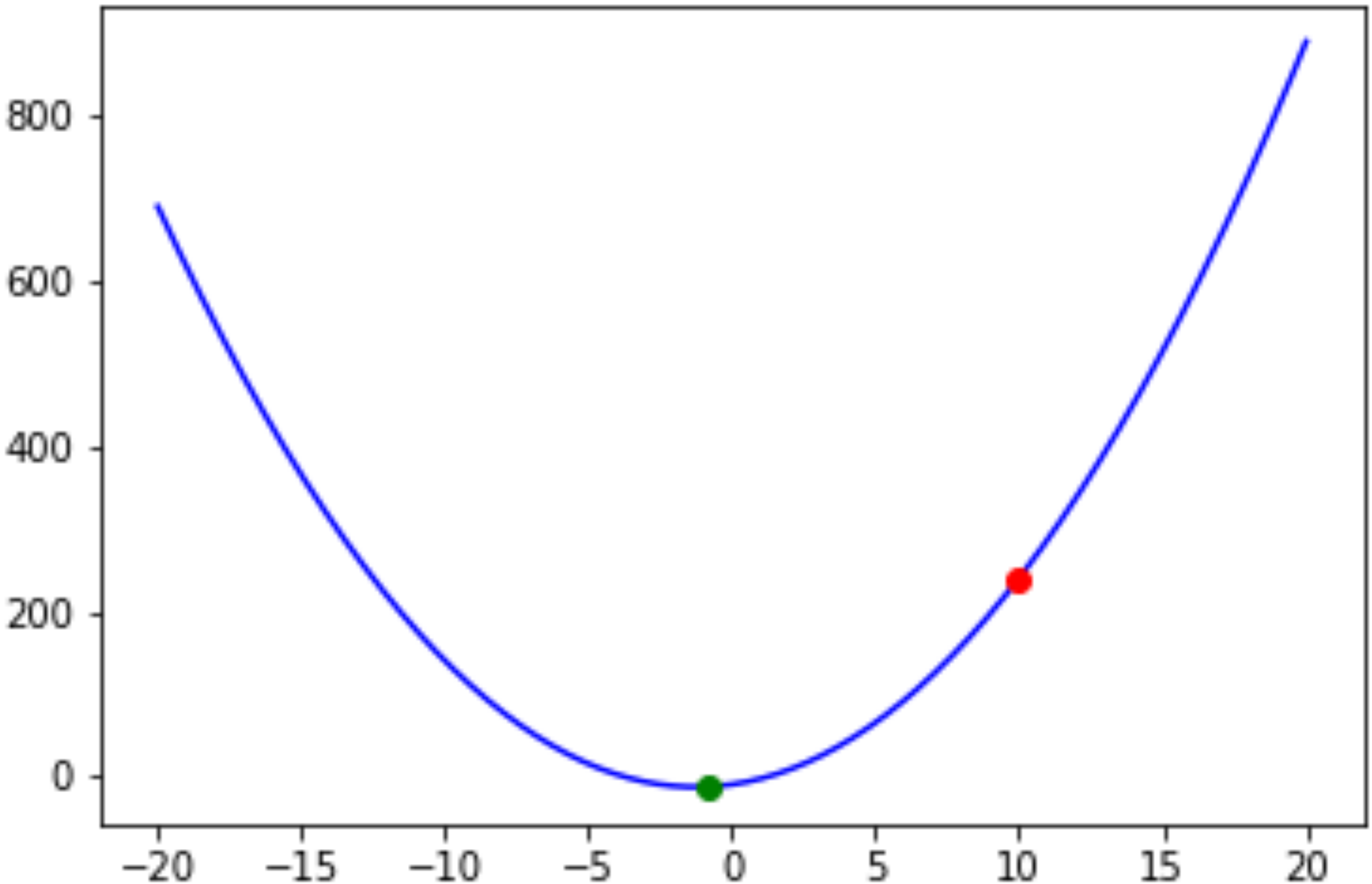
GRADIENT DESCENT

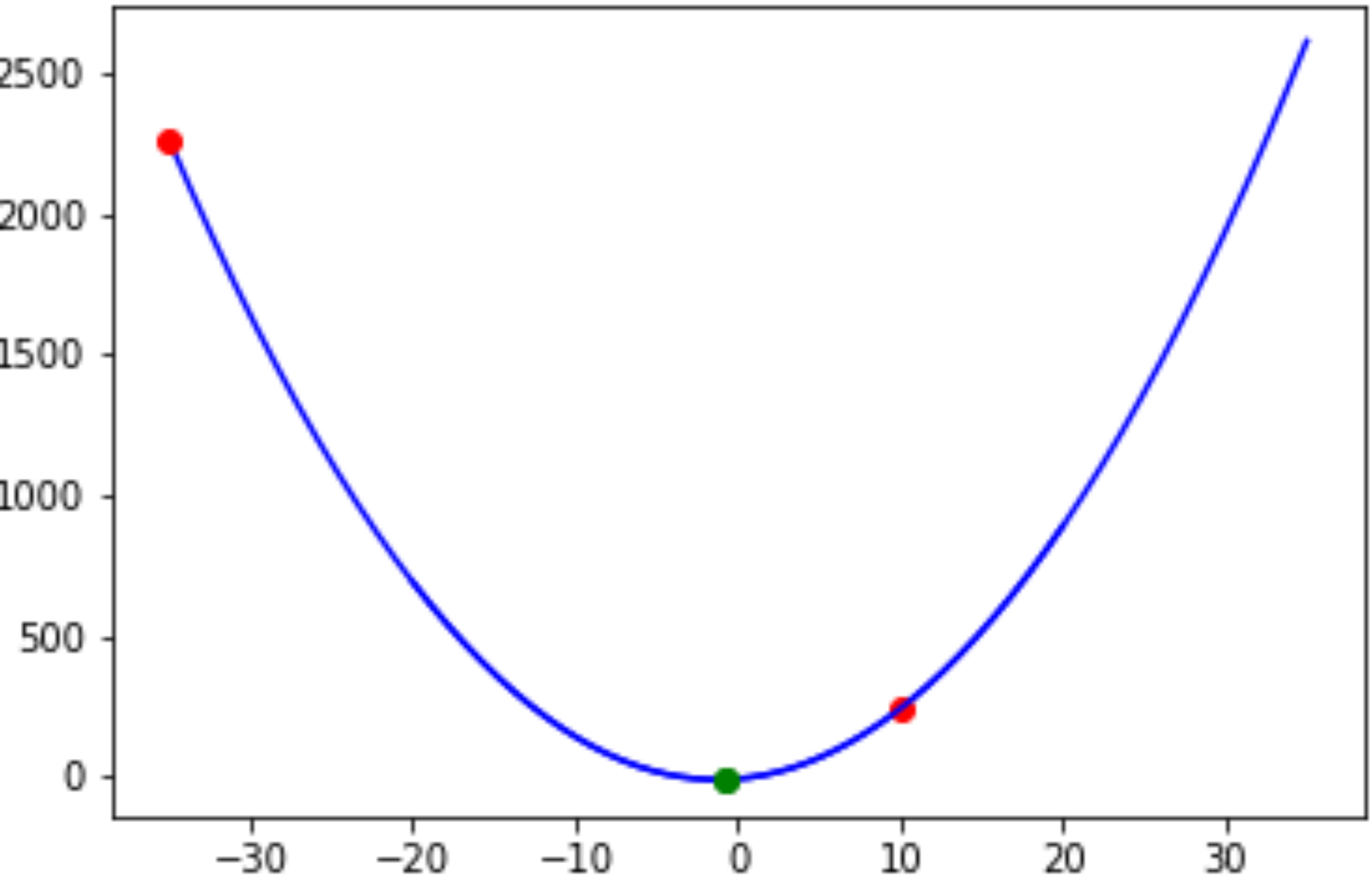
```
function gd(f, f', lo=100, hi=100):
  xt = uniform_random(lo, hi)
  gradt = f'(xt)
  while not converged:
    xt = xt - f'(xt)
  return xt
```

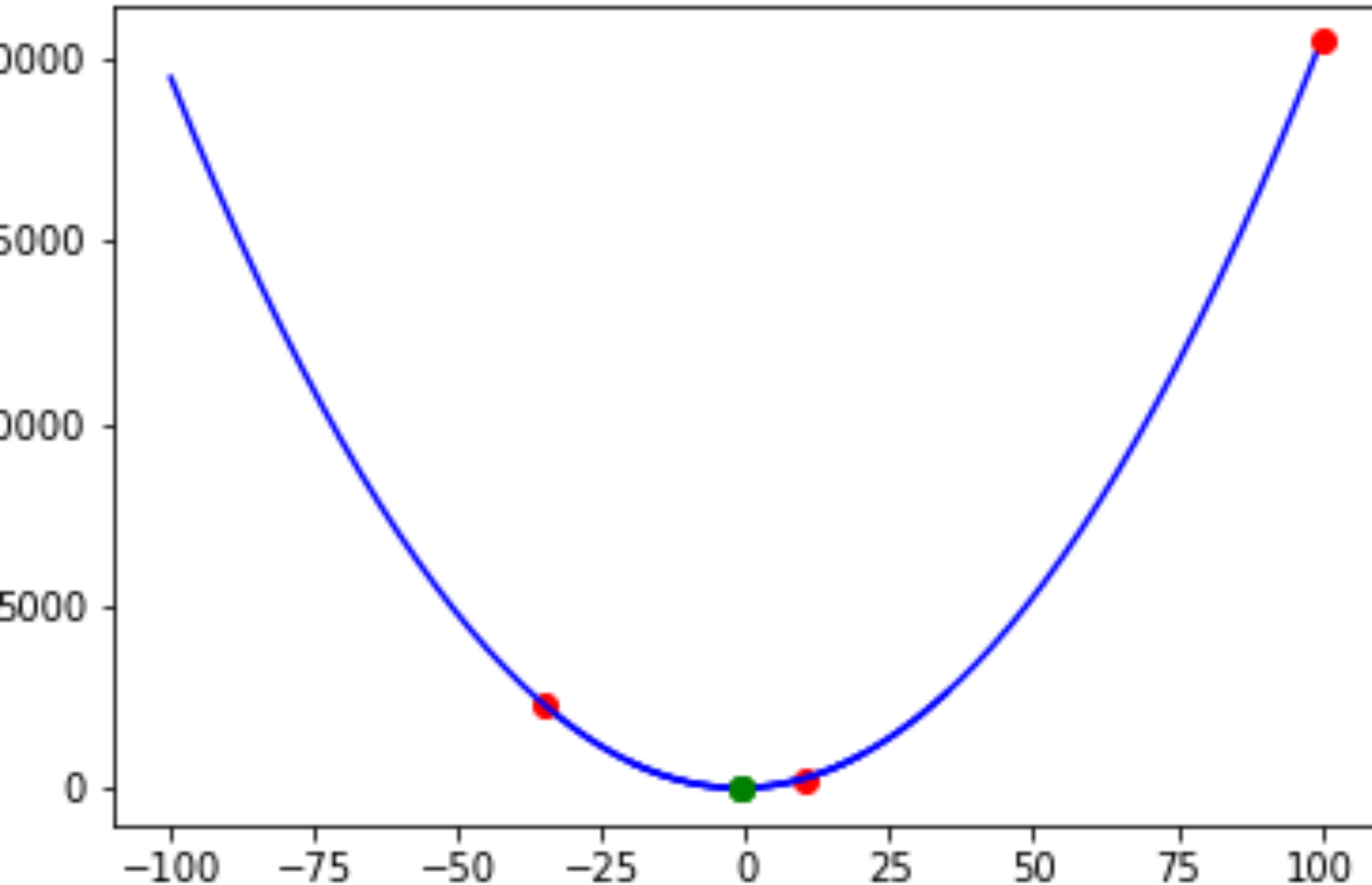
GRADIENT DESCENT

- Everything seems fine
  - But shape of the function can cause problems!
  - If we take too large steps, we can overshoot minimum!


800
600
400
200
0
−20
−15
−10
−5
0
5
10
15
20


800
600
400
200
0
−20
−15
−10
−5
0
5
10
15
20


2500
2000
1500
1000
500
0
−30
−20
−10
0
10
20
30



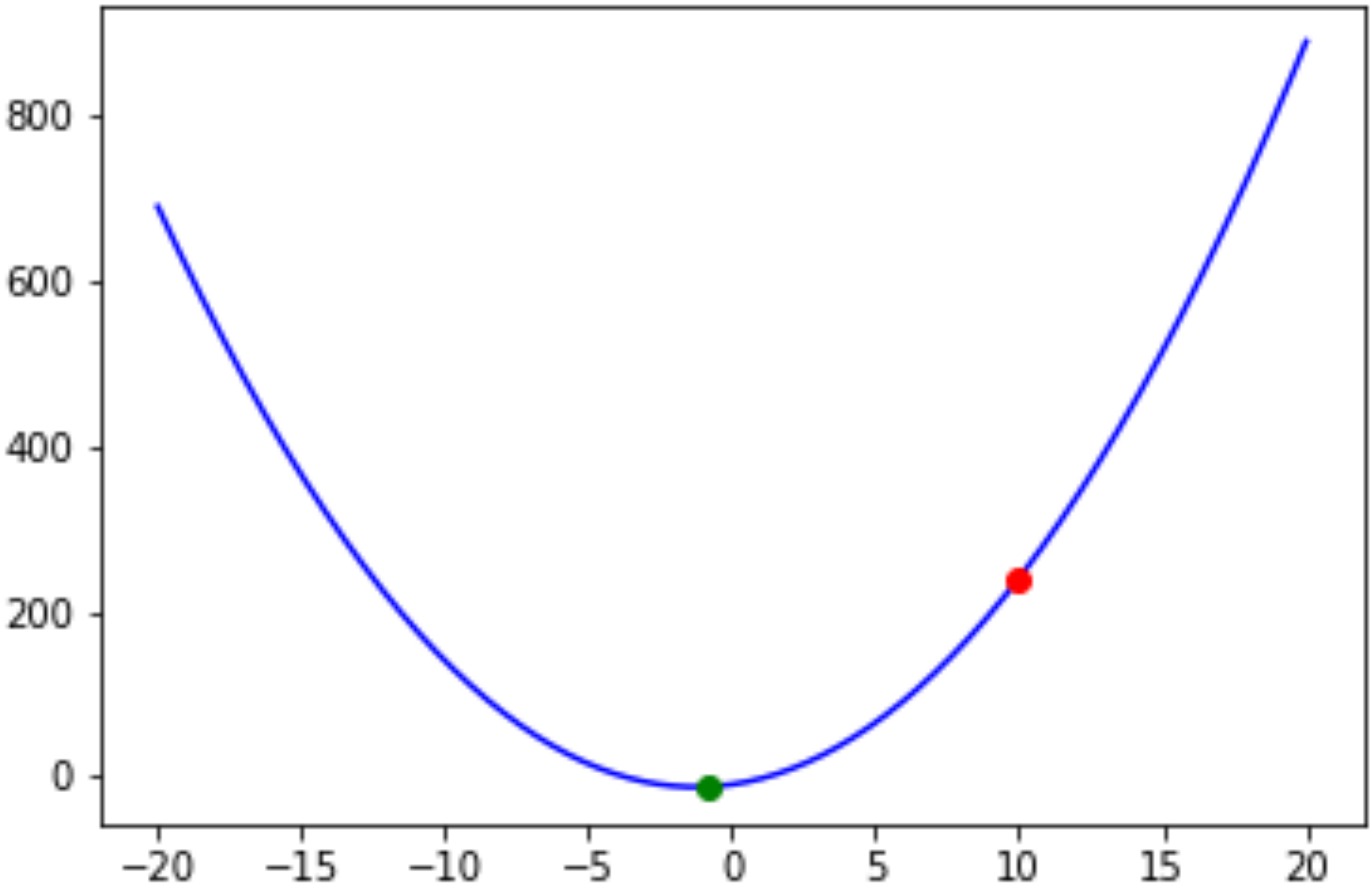
0000
5000
0000
5000
0
−100
−75
−50
−25
0
25
50
75
100

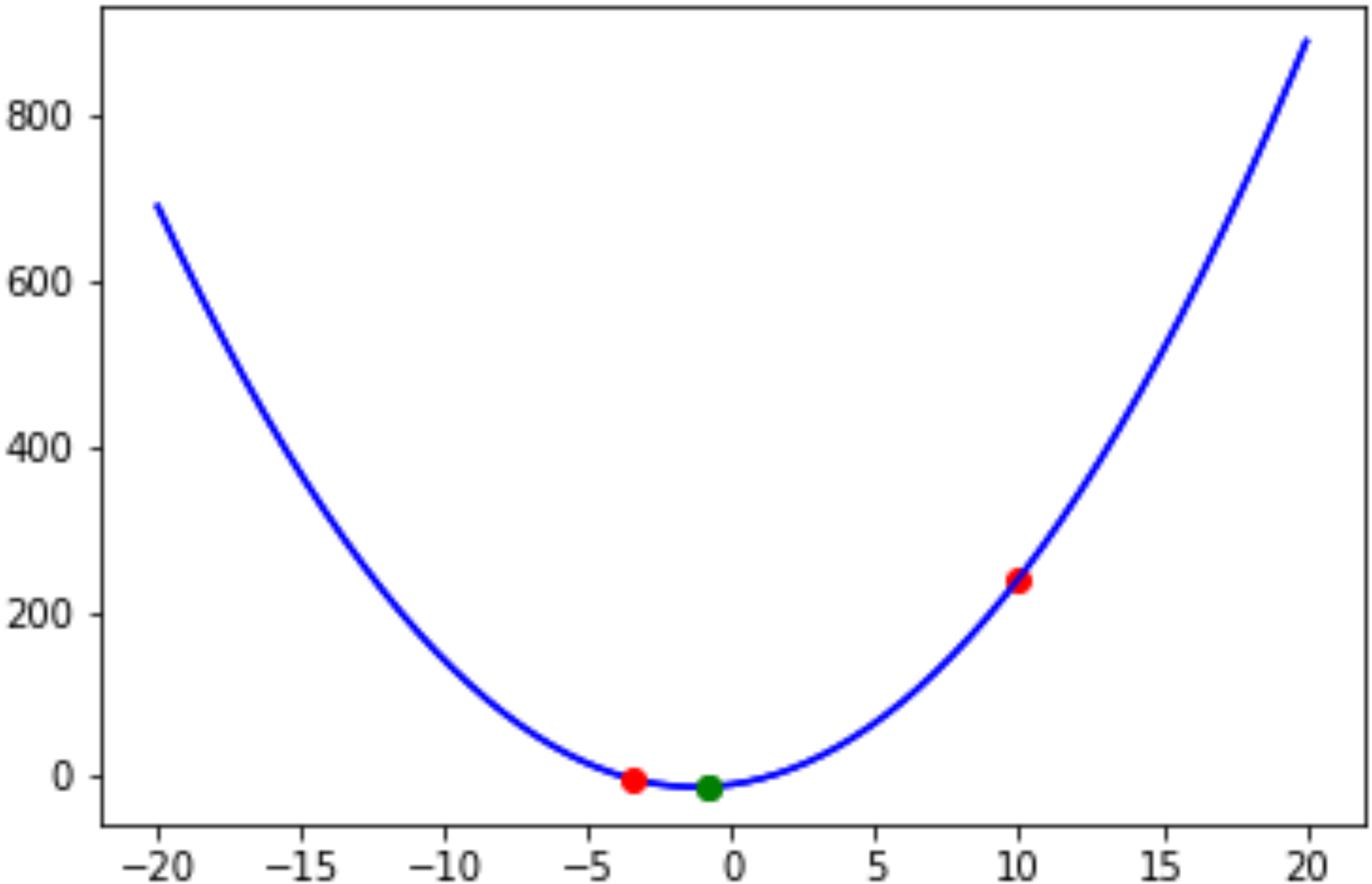
STEP SIZE

- In principle, since this function is convex, we should eventually converge
- But would take longer than if we moved in smaller steps
- Instead of jumping the direction opposite to the gradient, we "crawl"
- Introduce step size (or learning rate) into algorithm
  - denoted as α in most ML lit
  - denoted as η in most older optimisation lit

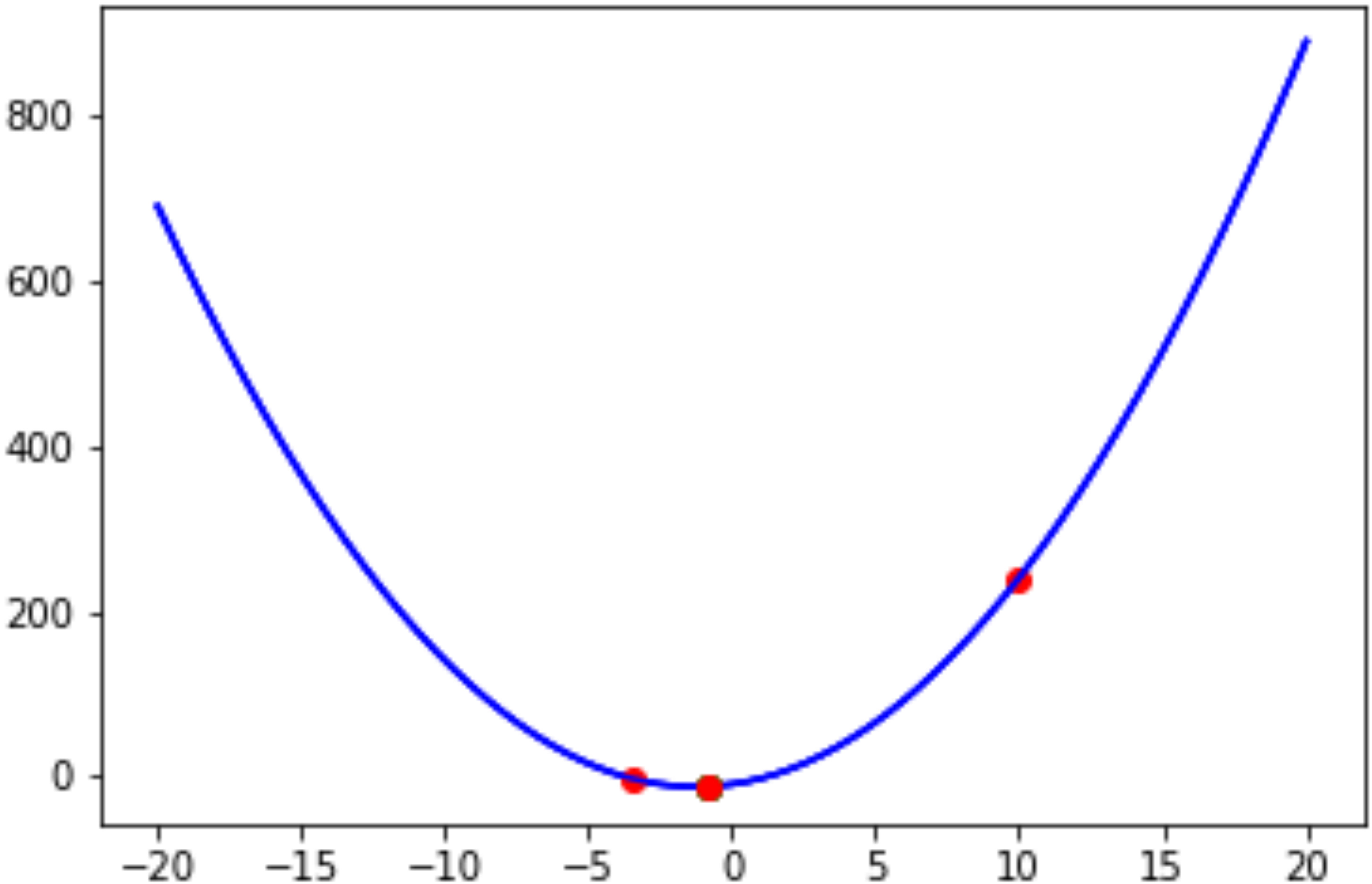
GRADIENT DESCENT

```
function gd(f, f′, α, lo=100, hi=100):
  xt = uniform_random(lo, hi)
  gradt = f′(xt)
  while not converged:
    xt = xt - αf′(xt)
  return xt
```


800
600
400
200
0
−20
−15
−10
−5
0
5
10
15
20



800
600
400
200
0
−20
−15
−10
−5
0
5
10
15
20



800
600
400
200
0
−20
−15
−10
−5
0
5
10
15
20