High-Assurance Scientific Computing: Boundary Value Analysis

Zhoulai Fu

IT University of Copenhagen

October 11, 2021

- Common sense: test cases exploring boundary conditions have a higher payoff than test cases that do not.
- Example: Inputs triggering “==“ for “if (x>=y) …”, or anything close
- The problem of finding such boundary inputs is know as boundary value analysis.

Literature on this topic

Effective Floating-Point Analysis via Weak-Distance Minimization. In 40th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), 2019.

Achieving High Coverage for Floating-Point Code via Unconstrained Programming. In 38th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), 2017.

XSat: A Fast Floating-Point Satisfiability Solver. In 28th International Conference on Computer Aided Verification (CAV), 2016.

G. J. Myers. **The Art of Software Testing**. pages I–XV, 1–234, 2004.

Example (with Demo)

```
void Prog(double x) {
  if (x < 1){
    x = x + 1;
    assert(x < 2);
  }}
```

- **The code may look correct.**
- **However, if we set the input to 0.999 999 999 999 999 9, the branch “if (x < 1)” will be taken, but the subsequent “assert (x + 1 < 2)” will fail (x + 1 = 2 in this case).**
- **This hidden bug can be found through generating boundary inputs.**

“So what” !?

The bigger picture is:

We will illustrate a perspective of how CS and Math can be fundamentally related

Today's lecture

- How to do automated testing via math
- Importantly, what you will see underlines a general method for program analysis, which we will elaborate next weeks.

Tentative schedule

- Lecture until 9h10; Exercises from 9h40-12h

Defining boundary inputs

- $A(P)$: *arithemtic conditions* of program P, namely $b ::= x \bowtie y$, where $\bowtie \in \{\leq, <, ==, >, \geq\}$.
- $\bar{b}$: *boundary form* of b, namely, $\bar{b}$ denotes $x == y$.
- $BV(P)$: *boundary values* of program P, namely the set of program inputs that trigger one or more than one of

$$\{\bar{b} \mid b \in A(P)\}$$

Manually generate boundary inputs

Boundary condition x==1 @ L.2

Boundary condition y==4 @ L.6

```
1  double foo(double x){
2    if (x<=1.0)
3      x++;
4
5    y = x*x;
6    if (y<=4.0)
7      x--;
8    return x;
9  }
```

x=-2 or 2 @L.5

-3, 1, 2 are boundary inputs

How random testing would work

Finding a needle in a haystack

How symbolic execution would work

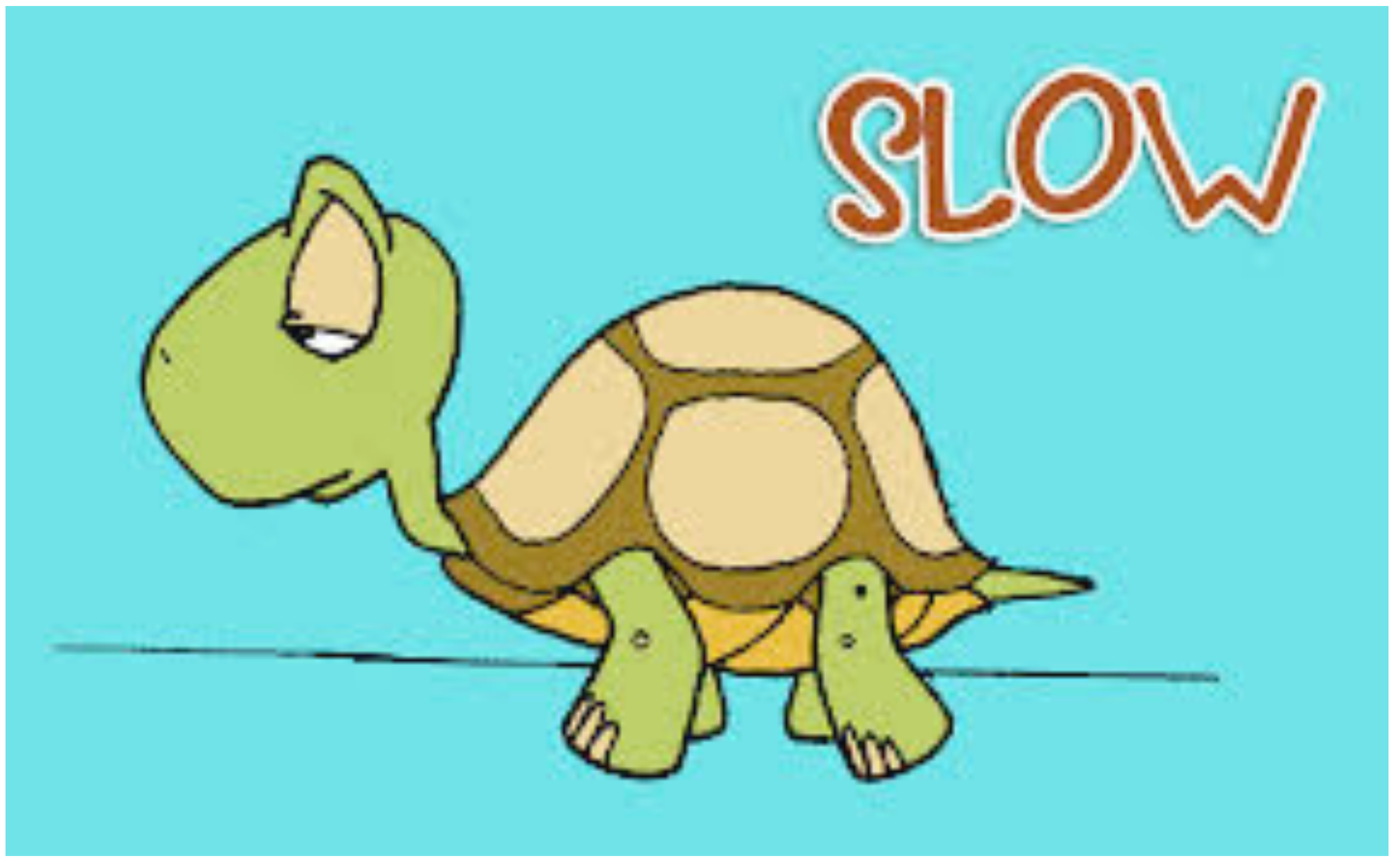
Idea:

every path reachability corresponds to a constraint on the inputs

```
double foo(double x){
  if (x<=1.0)
    x++;

  y = x*x;
  if (y<=4.0)
    x--;
  return x;
}
```

x == 1 OR (x > 1 AND x * x = 4) OR (x<=1.0 and (x + 1) ** 2 == 4)

Issues with symbolic execution




```
double foo(double x){
  if (x<=1.0)
    x++;

  y = x*x;
  if (y<=4.0)
    x--;
  return x;
}
```

- Path explosion
- Floating-point constraint solving

Boundary Value Analysis, mathematically (today's topic)

- **Boundary Value Analysis —> a fundamental math Problem**
- **Solve the math problem**
- **Interpret the results in the original BVA problem**

Step 1

```
double foo(double x){
  if (x<=1.0)
    x++;

  y = x*x;
  if (y<=4.0)
    x--;
  return x;
}
```

BV(foo)={-3, 1, 2}

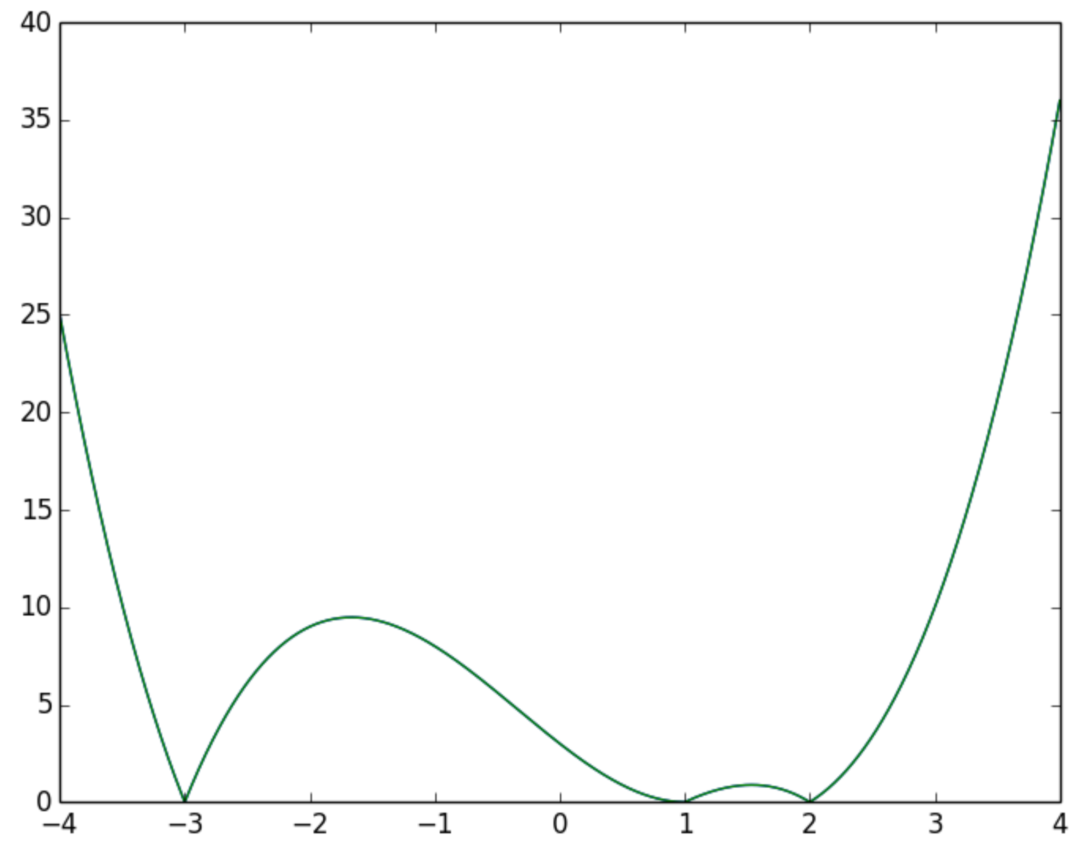
Transform

```
double _foo(double x, long double r){
  r = 1;
  r = r * abs(x - 1.0);
  if (x< = 1.0)
    x ++;

  y = x * x;
  r = r * abs(y - 4.0);
  if (y <= 4.0)
    x --;
  return x;
}

double t_foo(double x){
  _foo(x, &r);
  return r;
}
```

Important Property:

$$x \in BV(\texttt{foo}) \Leftrightarrow \texttt{t\_foo}(\texttt{x}) = 0$$

Properties of t_foo



Prop1.

t_foo(x) is continuous

(under some conditions)

Prop2.

t_foo(x)>=0 for all x

Prop3.

$$x \in BV(\texttt{foo}) \Leftrightarrow \texttt{t\_foo(x)} = 0$$

Key Insight

Reducing the problem of finding BV(foo) to:

- **finding zeros of t_foo (Prop. 3)**
- **finding minima of t_foo (Prop. 2)**
- **finding minima of t_foo can be easy (Prop. 1)**

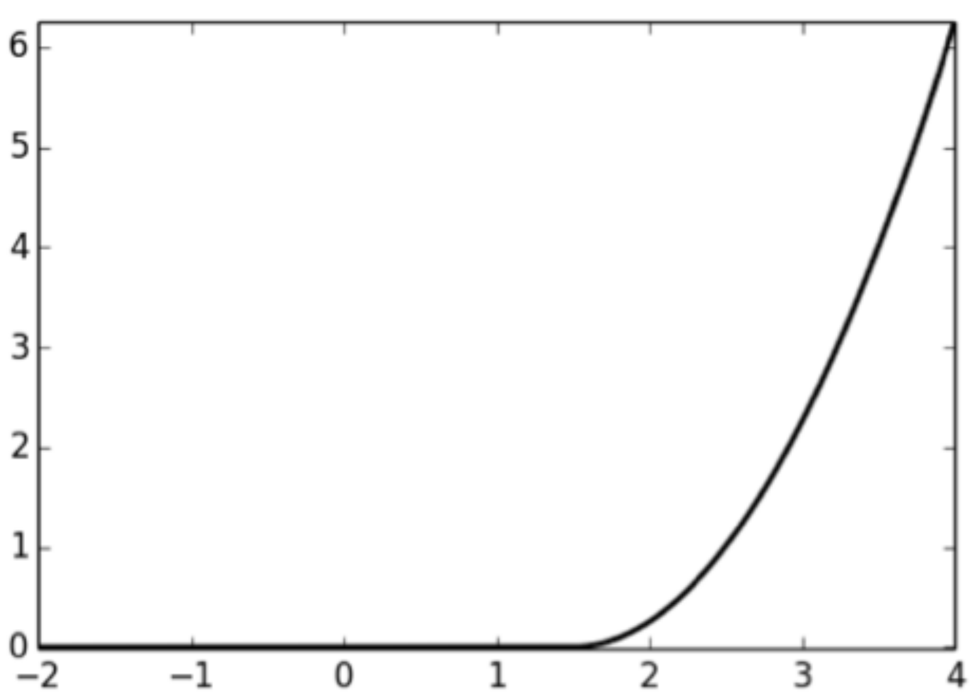
Minimize t_foo

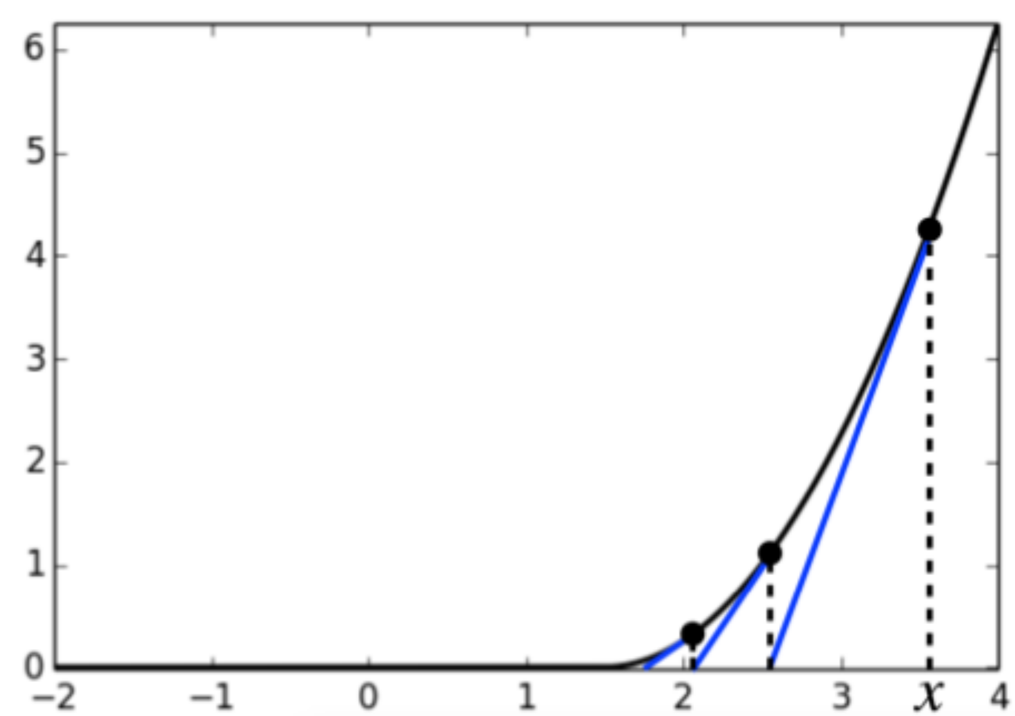
Mathematical Optimization

Local optimization

$$f_1(x) = \begin{cases} 0 & \text{if } x \leq 1.5 \\ (x-1.5)^2 & \text{otherwise} \end{cases}$$

Example for a common local optimization method: Use tangents of the curve to quickly converge to the minimum point.

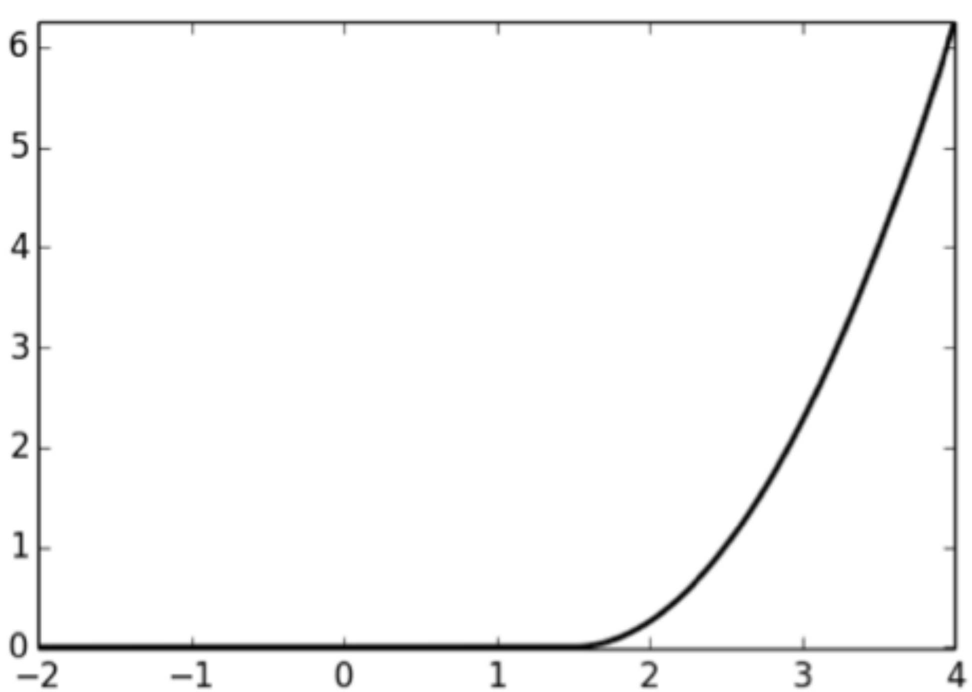


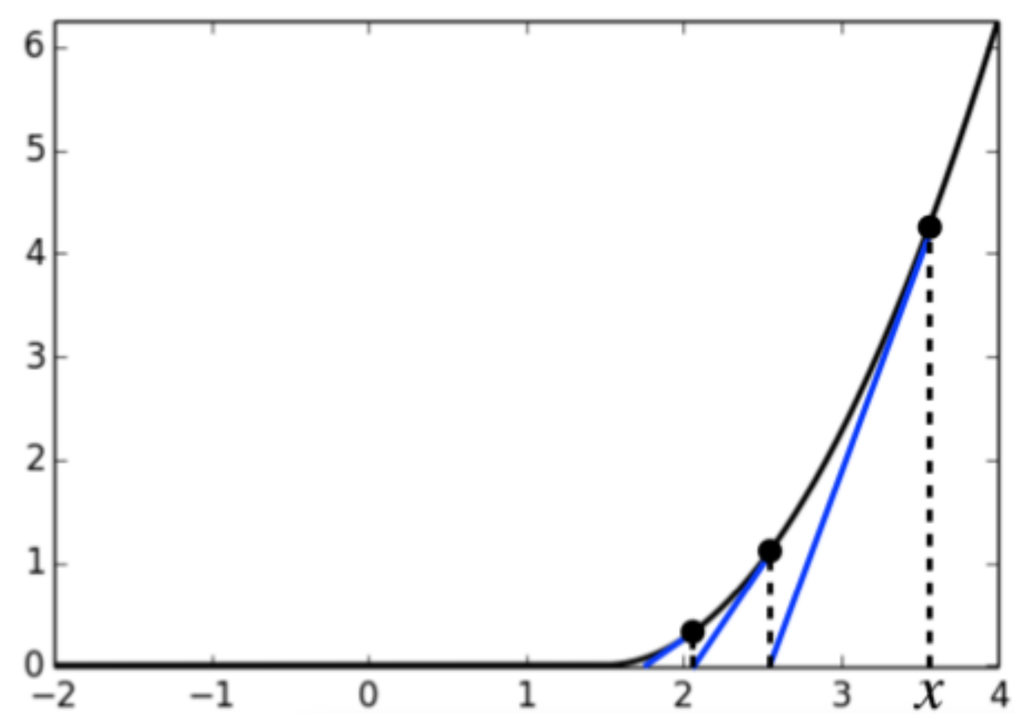


Demo on local optimization

$$f_1(x) = \begin{cases} 0 & \text{if } x \leq 1.5 \\ (x-1.5)^2 & \text{otherwise} \end{cases}$$

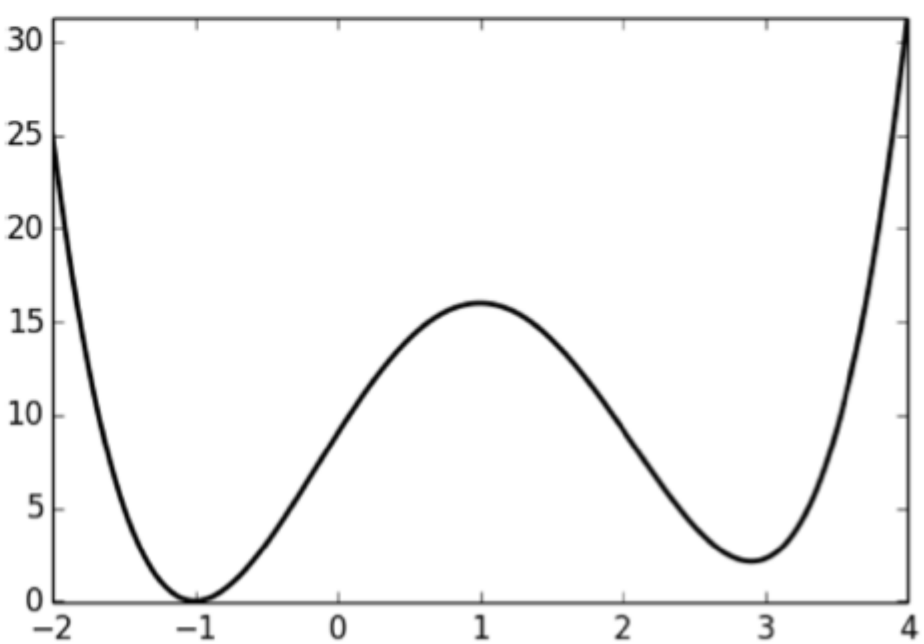
Example for a common local optimization method: Use tangents of the curve to quickly converge to the minimum point.

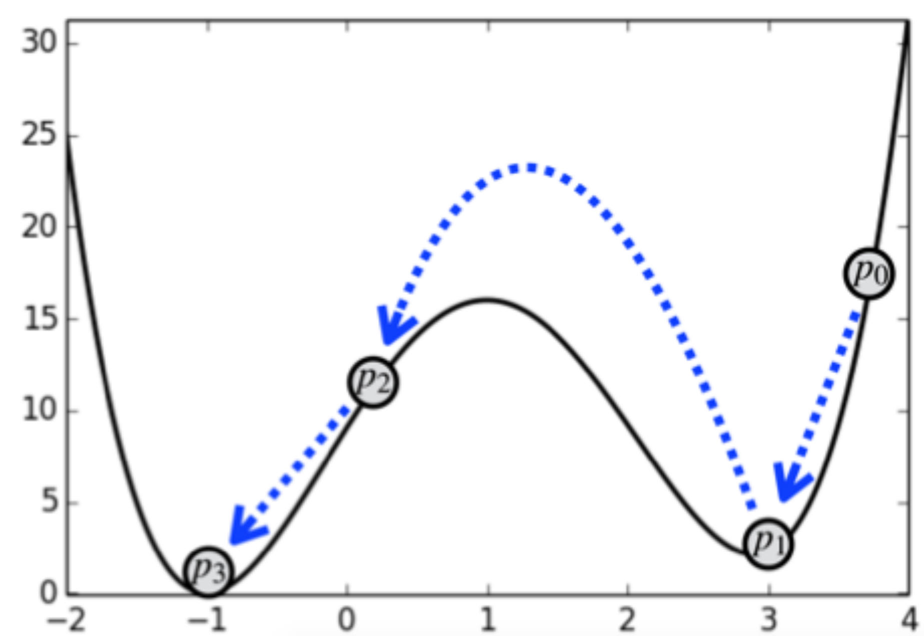




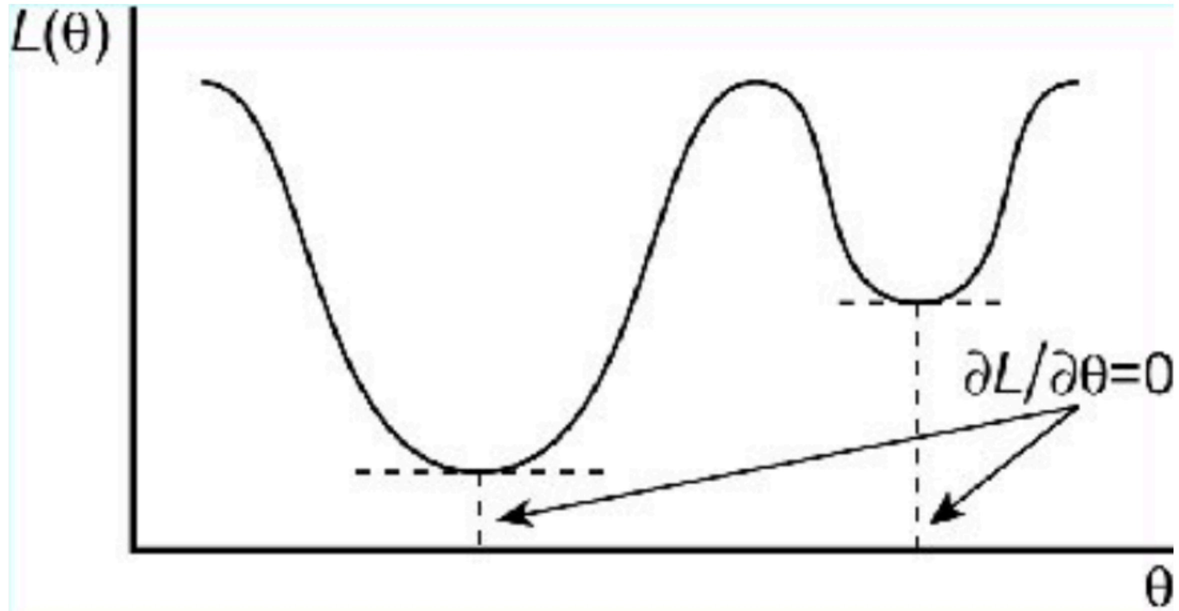
Global optimization

Example for a common global optimization method (MCMC): Jump following the Monte Carlo style, and then do local optimization. Global methods include: genetic algorithms, evolutionary strategies, simulated annealing, etc.





A key of the presented approach is to use optimization methods as blackboxes

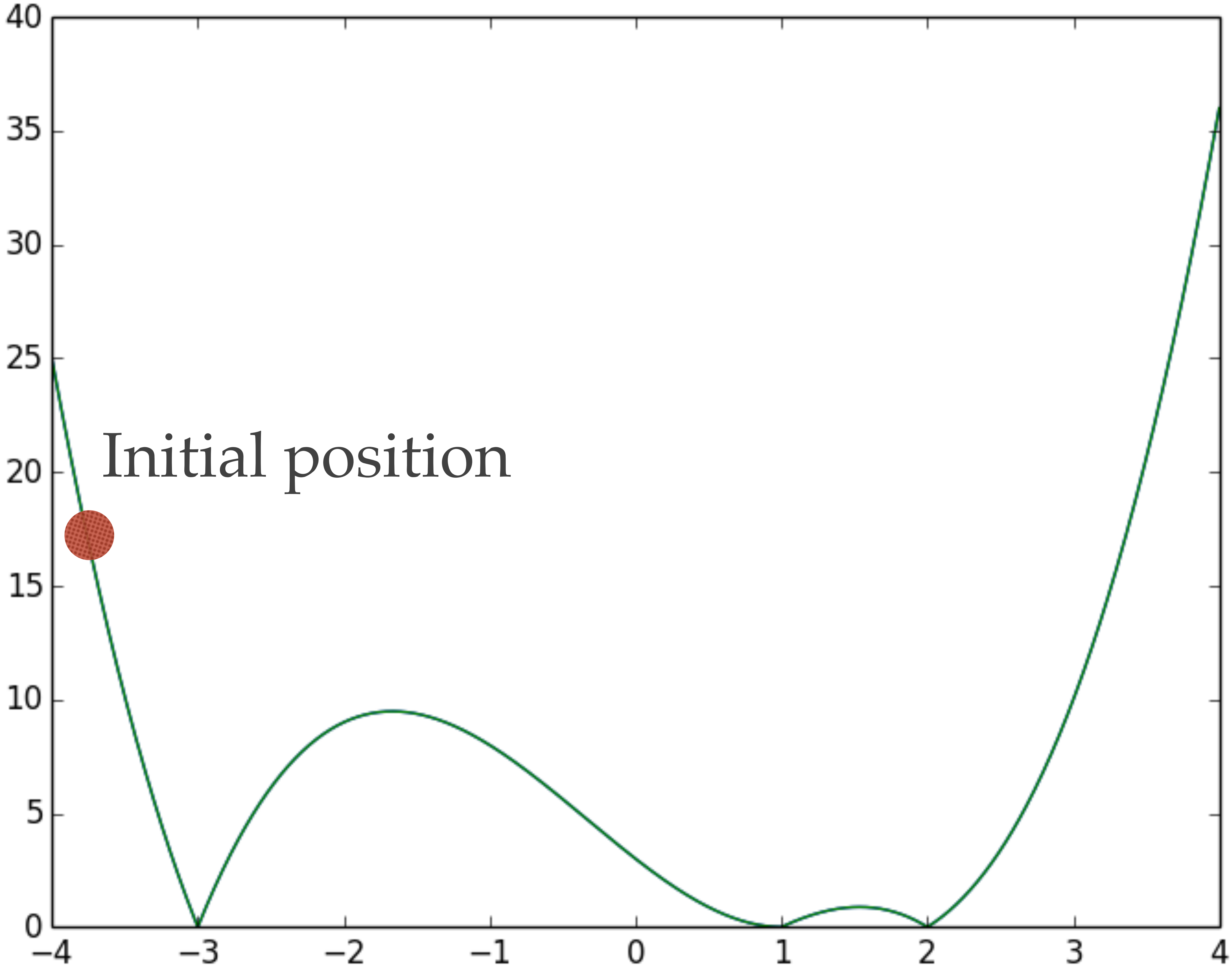


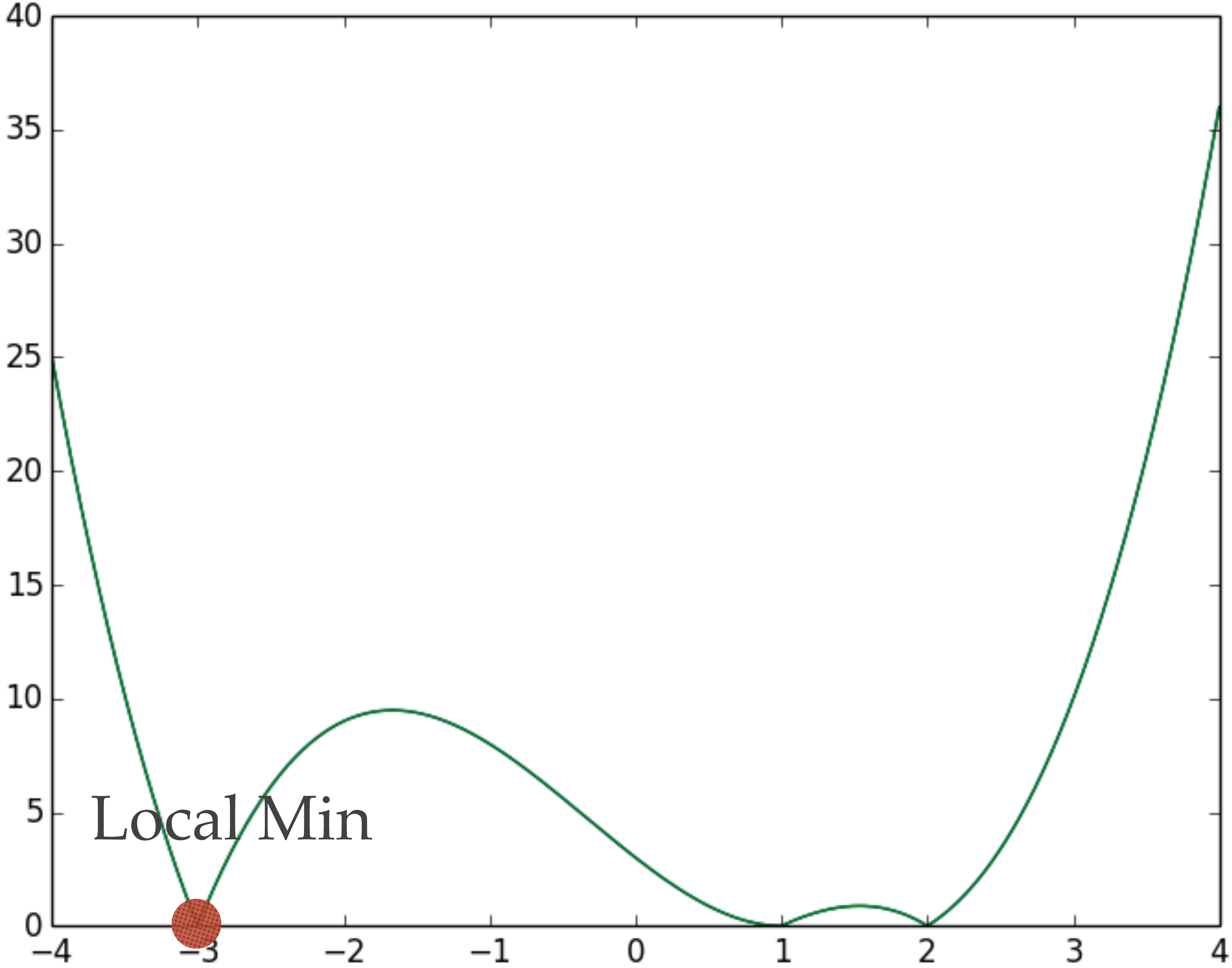
- **Typically, the solution to local minimization can be found at** $\partial L / \partial \theta = 0$

- **Generally, global optimization problem is harder than local optimization.**

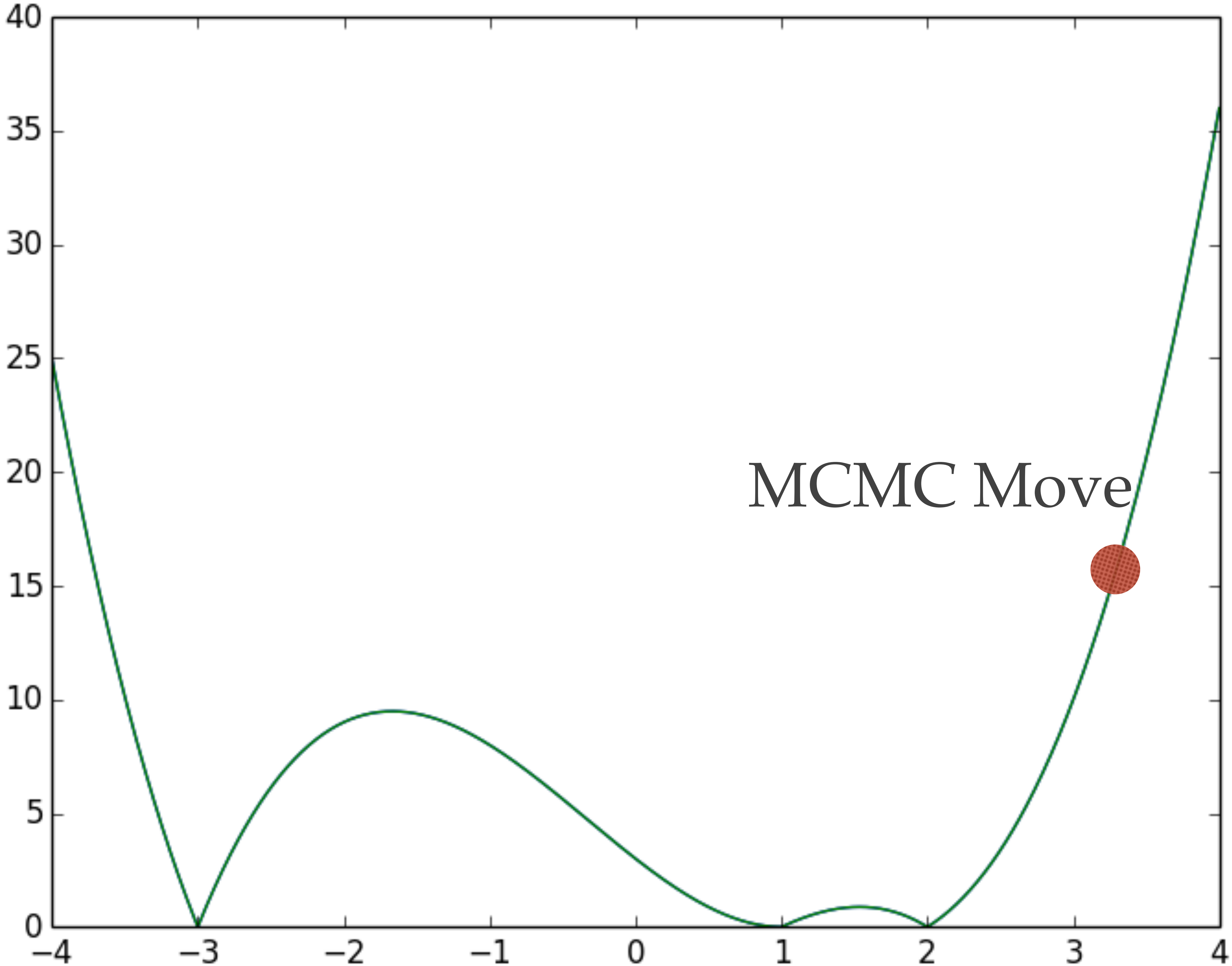
- **Both local and global optimizations may fail, providing sub-optimal results.**

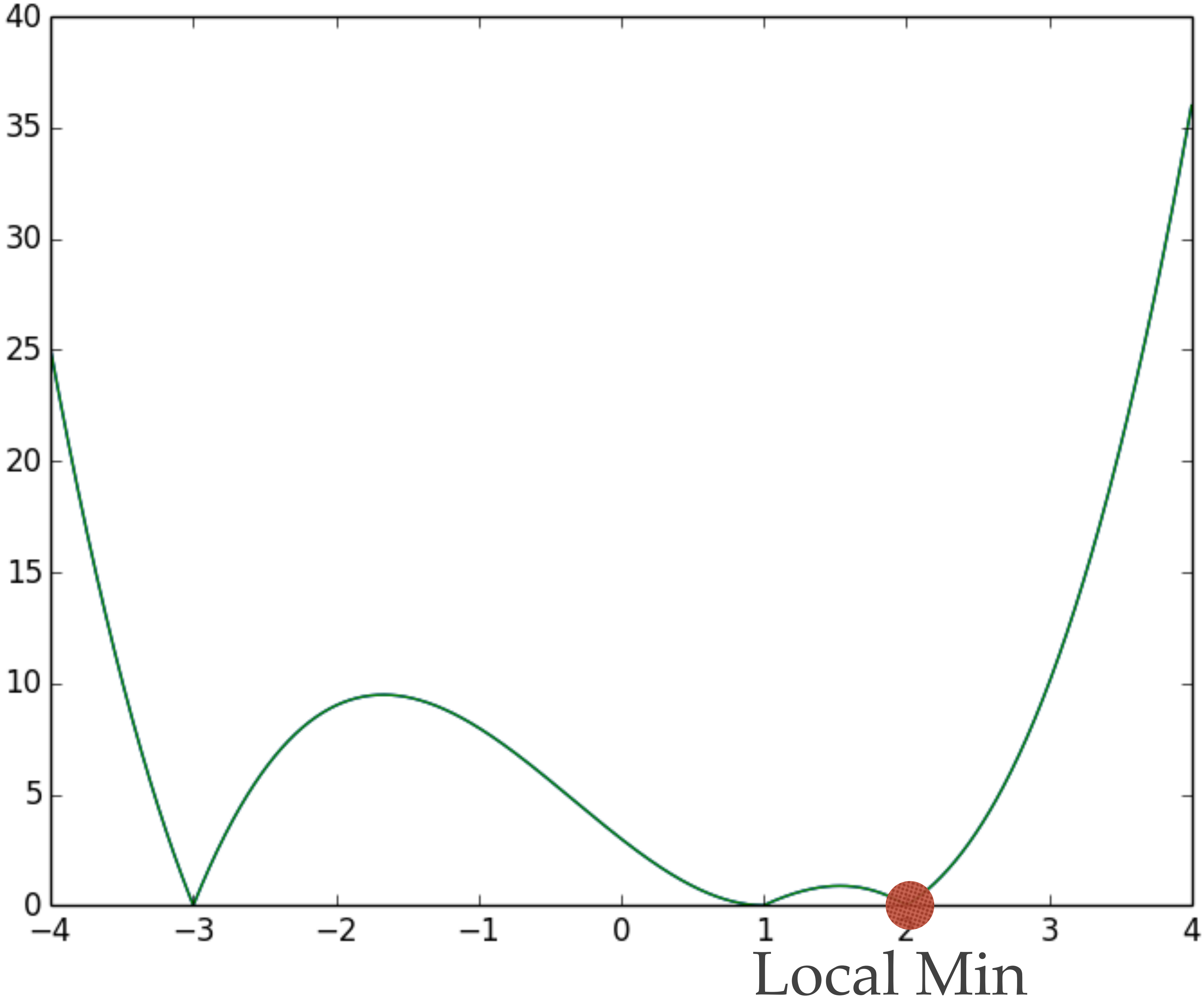
- **But they can produce precise results for certain problems, e.g. linear, convex problems.**


Initial position
40
35
30
25
20
15
10
5
0
−4
−3
−2
−1
0
1
2
3
4



40
35
30
25
20
15
10
5
0
−4
−3
−2
−1
0
1
2
3
4
Local Min


MCMC Move
40
35
30
25
20
15
10
5
0
−4
−3
−2
−1
0
1
2
3
4



40
35
30
25
20
15
10
5
0
−4
−3
−2
−1
0
1
2
3
4
Local Min

E.g., Python's implementation

Basin-hopping

A popular MCMC implementation

[Global Optimization by Basin-Hopping and the Lowest Energy](#)
pubs.acs.org/doi/abs/10.1021/jp970984n
American Chemical Society
by DJ Wales - 1997 - Cited by 1329 - Related articles
Jul 10, 1997 - We describe a global optimization technique using "*basin-hopping*" in which the potential energy surface is transformed into a collection of ...

We will use optimization methods as blackboxes

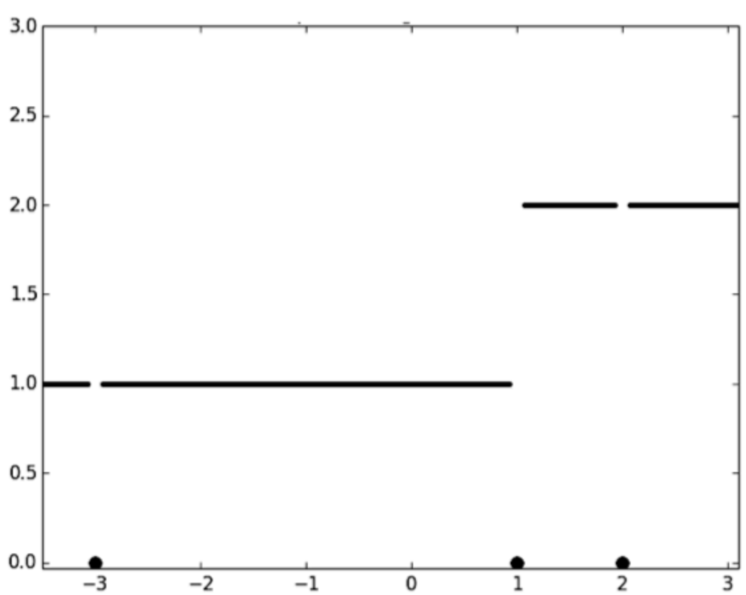
Advantage for using mathematical optimization

- The complexity of the approach relies on the executing programs, rather than analyzing

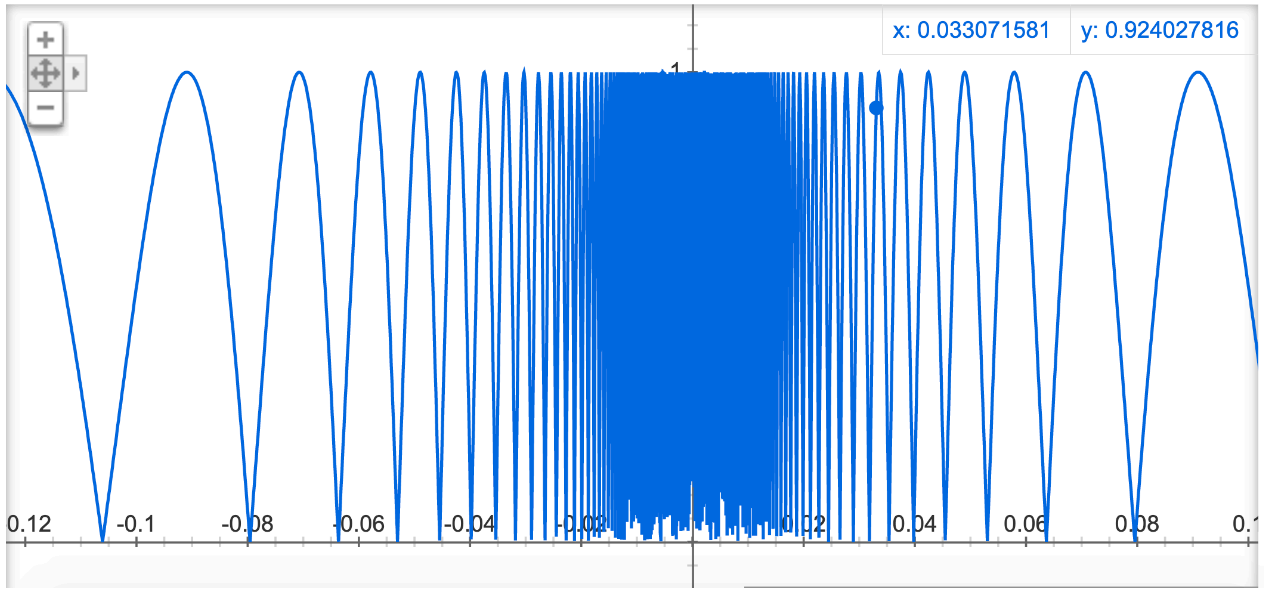
- Yet, unlike testing, the approach also provides a theoretical guarantee: As long as the MO tool can precisely find the minimum, the boundary value analysis problem is solved.

Challenge for Using MO in this Problem

- To find minimum can be challenging for badly-formed functions
- Infinite # of local minimum points



Graph for abs(sin(1/x))




Demo

Demo 1: Revisited

```
def FOO(x):
    if x <= 1.0:
        x= x + 1

    y = x*x
    if y <= 4.0:
        x = x - 1
    return x
```

Expected Results

BV(FOO)={-3,1,2}

Users/zhfu/projects/20_teaching_asa/code/demo2_lec1.py

Demo 2: Finding a bug

```
def FOO(x):
    if x < 1.0:
        y= x + 1
        if y>=2: raise Exception ("UNEXPECTED! Input %.17f" %x)
```

Users/zhfu/projects/20_teaching_asa/code/demo2_lec2.py

Conclusion

What matters is not the part. Bugs, bug the generality of the approach
Peek into what it looks in CS research