

Lecture 35

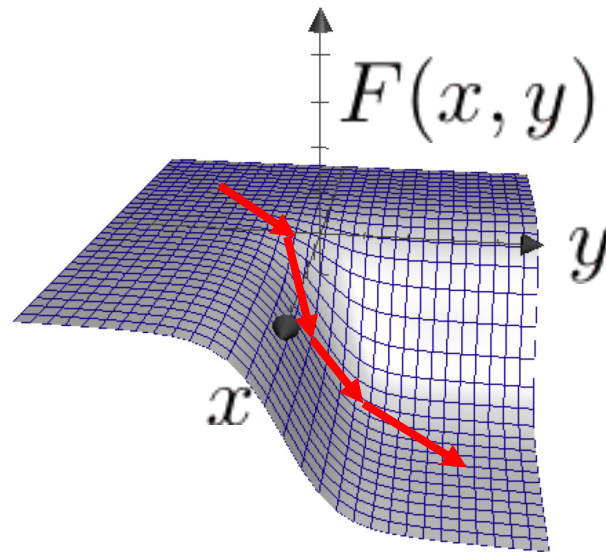
Synthesis with Optimization

Sankha Narayan Guria
with slides from
Armando Solar-Lezama
in turn with slides from
Swarat Chaudhuri

Numerical Optimization

Goal: Find a (hopefully global) minima of a function

- Sample the function on a small set of values
- Based on the result sample in a new set of locations



- Gradient descent
- Newton iteration
- ...

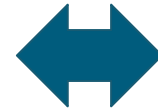
Numerical Optimization for Synthesis

Many synthesis problems are best framed as optimization problems

- Especially true for problems involving hybrid systems or probabilities

The Parameter Synthesis Problem

```
tOff := ??;  tOn := ??;  h := ??  
forever{  
  temp := readTemp();  
  if (isOn() && temp > tOff)  
    switchHeaterOff();  
  elseif ( !isOn() && temp < tOn)  
    switchHeaterOn(h);  
}
```



Warming:

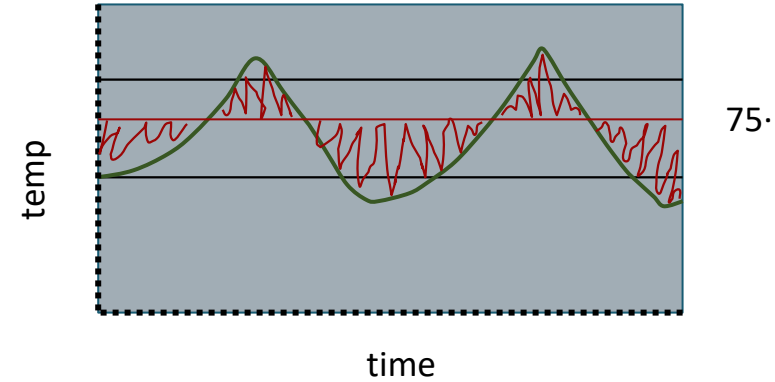
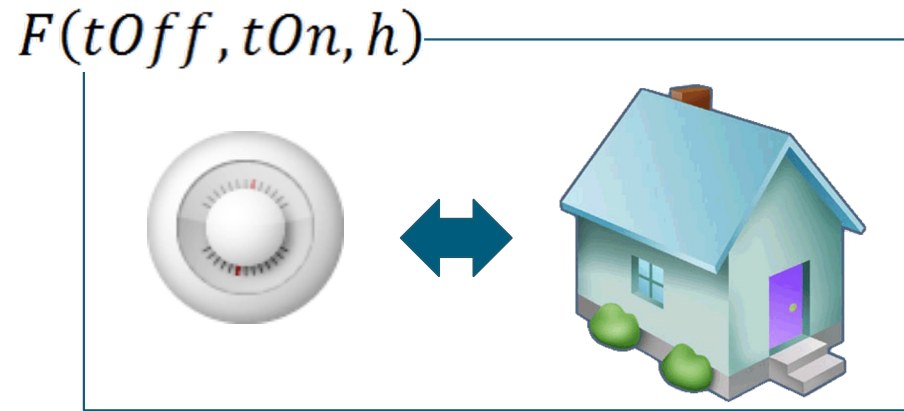
$$\frac{d}{dt}temp = -k \cdot temp + h$$

Cooling:

$$\frac{d}{dt}temp = -k \cdot temp$$



The Parameter Synthesis Problem

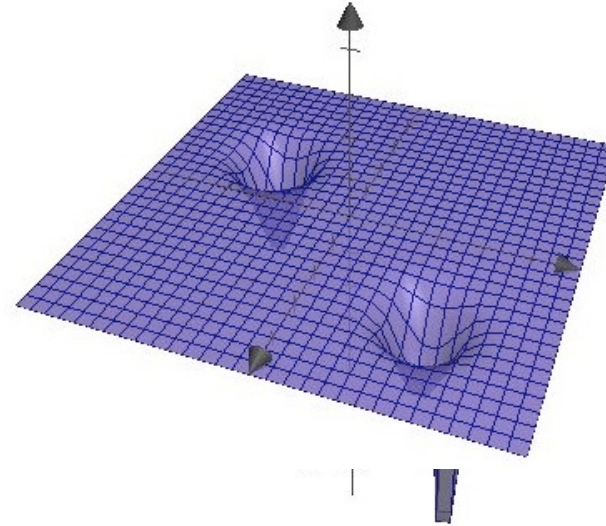
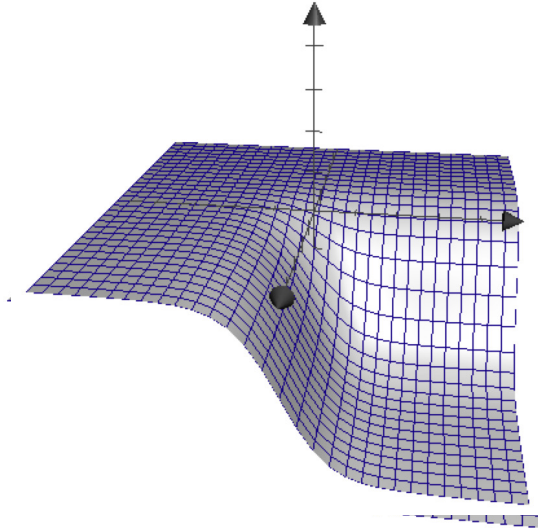


$$F(tOff, tOn, h) = [t_0, t_1, \dots, t_k]$$

$$Err(tOff, tOn, h) := \sum (t_i - 75)^2$$

Can we find values of $(tOff, tOn, h)$ to minimize Err?

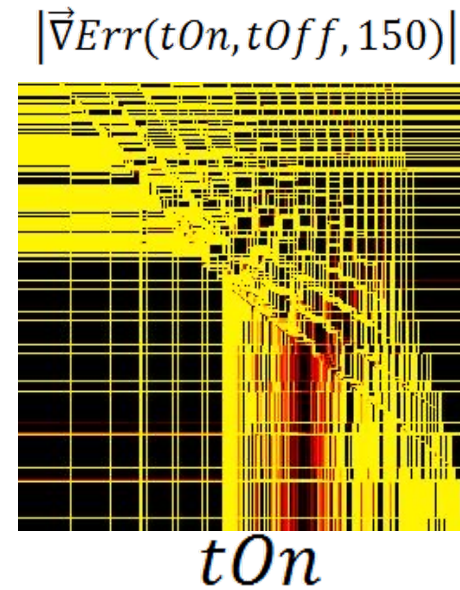
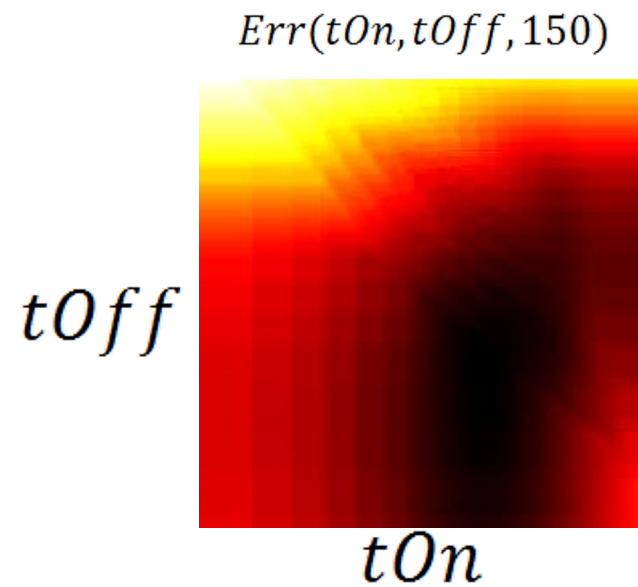
The challenge of optimizing programs



Plateaus and discontinuities thwart search

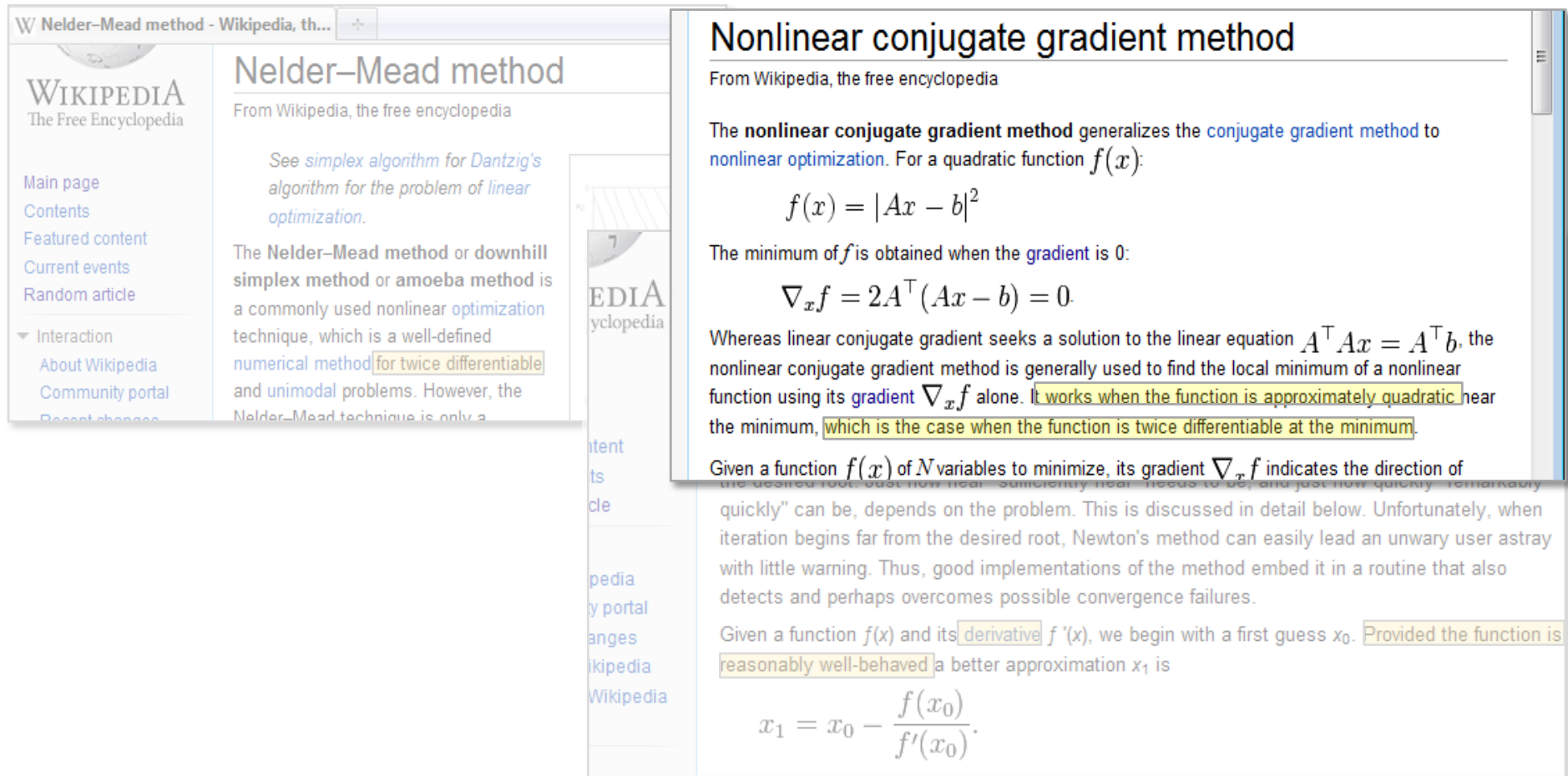
Branches introduce discontinuities

For the thermostat example



Minimization by numerical Search

Discontinuities affect even the best methods



The image shows two overlapping Wikipedia browser windows. The background window is titled 'Nelder-Mead method - Wikipedia, the free encyclopedia'. It features the Wikipedia logo and a sidebar with links like 'Main page', 'Contents', and 'Random article'. The main text area is titled 'Nelder-Mead method' and includes a reference to the 'simplex algorithm' and a description of the method as a 'downhill simplex method' or 'amoeba method' used for nonlinear optimization.

The foreground window is titled 'Nonlinear conjugate gradient method'. It also has the Wikipedia logo and a similar sidebar. The main text area is titled 'Nonlinear conjugate gradient method' and describes how it generalizes the conjugate gradient method to nonlinear optimization. It includes the formula for a quadratic function $f(x) = |Ax - b|^2$ and the condition for the minimum when the gradient is zero: $\nabla_x f = 2A^T(Ax - b) = 0$. It also discusses the method's application to finding local minima of nonlinear functions and provides the formula for the next iteration: $x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$.

Nelder-Mead method
From Wikipedia, the free encyclopedia

See *simplex algorithm* for Dantzig's algorithm for the problem of linear optimization.

The Nelder-Mead method or downhill simplex method or amoeba method is a commonly used nonlinear optimization technique, which is a well-defined numerical method for twice differentiable and unimodal problems. However, the Nelder-Mead technique is only a

Nonlinear conjugate gradient method
From Wikipedia, the free encyclopedia

The nonlinear conjugate gradient method generalizes the conjugate gradient method to nonlinear optimization. For a quadratic function $f(x)$:

$$f(x) = |Ax - b|^2$$

The minimum of f is obtained when the gradient is 0:

$$\nabla_x f = 2A^T(Ax - b) = 0.$$

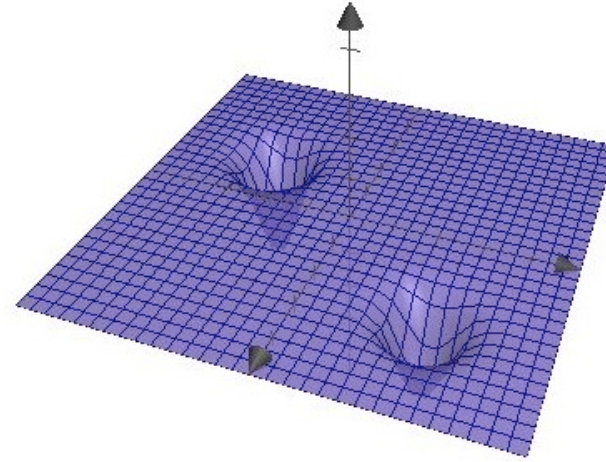
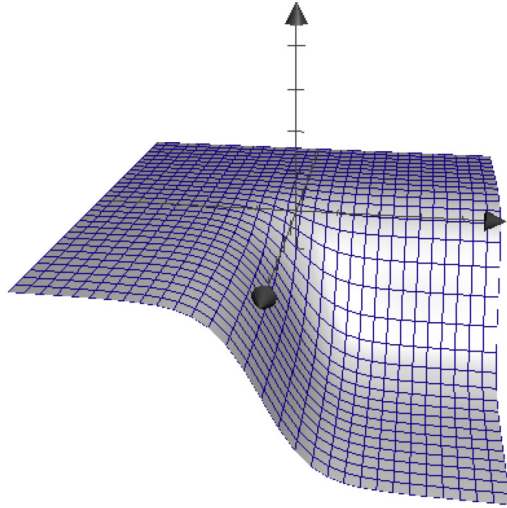
Whereas linear conjugate gradient seeks a solution to the linear equation $A^T Ax = A^T b$, the nonlinear conjugate gradient method is generally used to find the local minimum of a nonlinear function using its gradient $\nabla_x f$ alone. It works when the function is approximately quadratic near the minimum, which is the case when the function is twice differentiable at the minimum.

Given a function $f(x)$ of N variables to minimize, its gradient $\nabla_x f$ indicates the direction of the desired root. Just how near, sufficiently near, needs to be, and just how quickly, remarkably quickly" can be, depends on the problem. This is discussed in detail below. Unfortunately, when iteration begins far from the desired root, Newton's method can easily lead an unwary user astray with little warning. Thus, good implementations of the method embed it in a routine that also detects and perhaps overcomes possible convergence failures.

Given a function $f(x)$ and its derivative $f'(x)$, we begin with a first guess x_0 . Provided the function is reasonably well-behaved a better approximation x_1 is

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}.$$

Key Idea: Smooth the Function



Approximate the problem by a smooth function

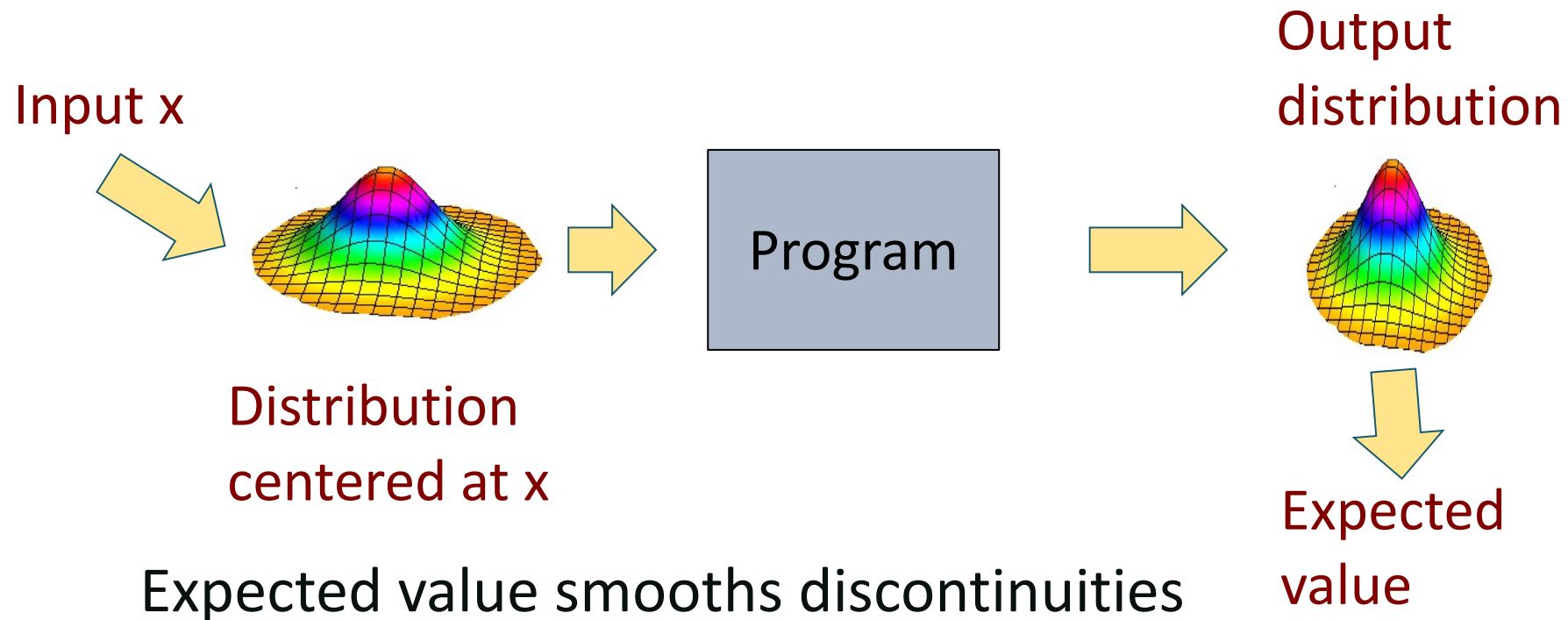
Example: Gaussian Smoothing

Execute on a distribution instead of a value



Example: Gaussian Smoothing

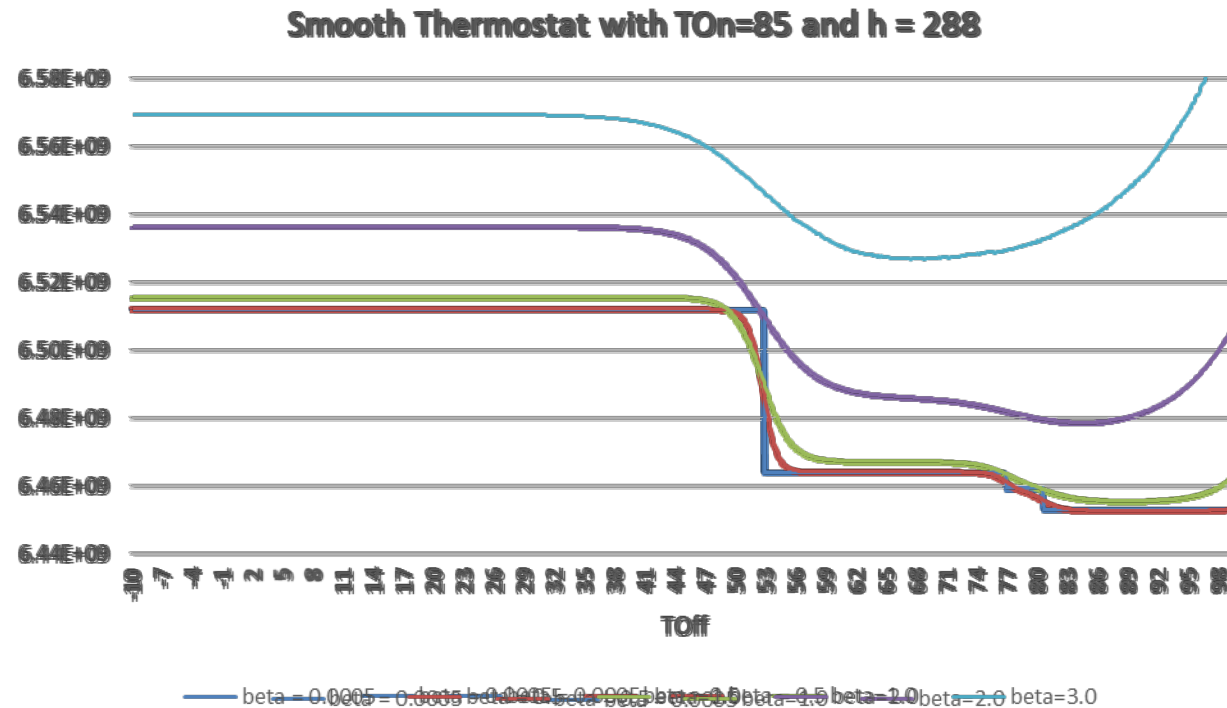
Execute on a distribution instead of a value



Parameterized Smoothing

Parameterized by a value β

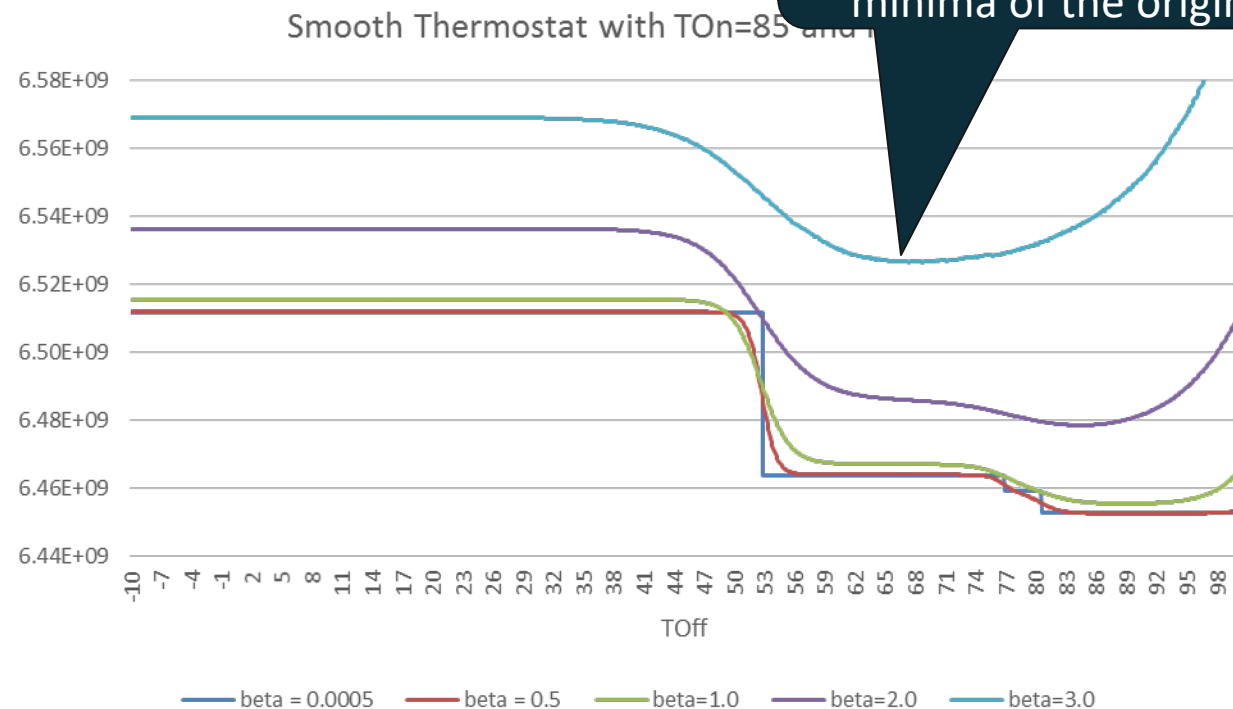
- Corresponds to std dev when smoothing with Gaussian noise



Parameterized Smoothing

$F_{smooth}(p, \beta_i)$ parameterized by a value β

- Corresponds to std dev when smoothing with Gaussian noise



Smoothed Numerical Search

$p_0 = \text{random starting point}$

$\beta_0 = \text{initial beta}$

While $\beta_i > \epsilon$

$p_{i+1} = \text{optimize } F_{\text{smooth}}(p, \beta_i) \text{ starting at } p_i$

$\beta_{i+1} = \beta_i * q$

return p_i


$$0 \leq q < 1$$

Smoothed Numerical Search

Very general procedure

Steps:

- Frame your problem as an optimization problem
- Define a parameterized smoothing procedure
- Run the algorithm

Smoothing doesn't have to be perfect

- But it should approximate the desired function in the limit as $\beta \rightarrow 0$

Euler:
SNS for programs

SNS in Euler

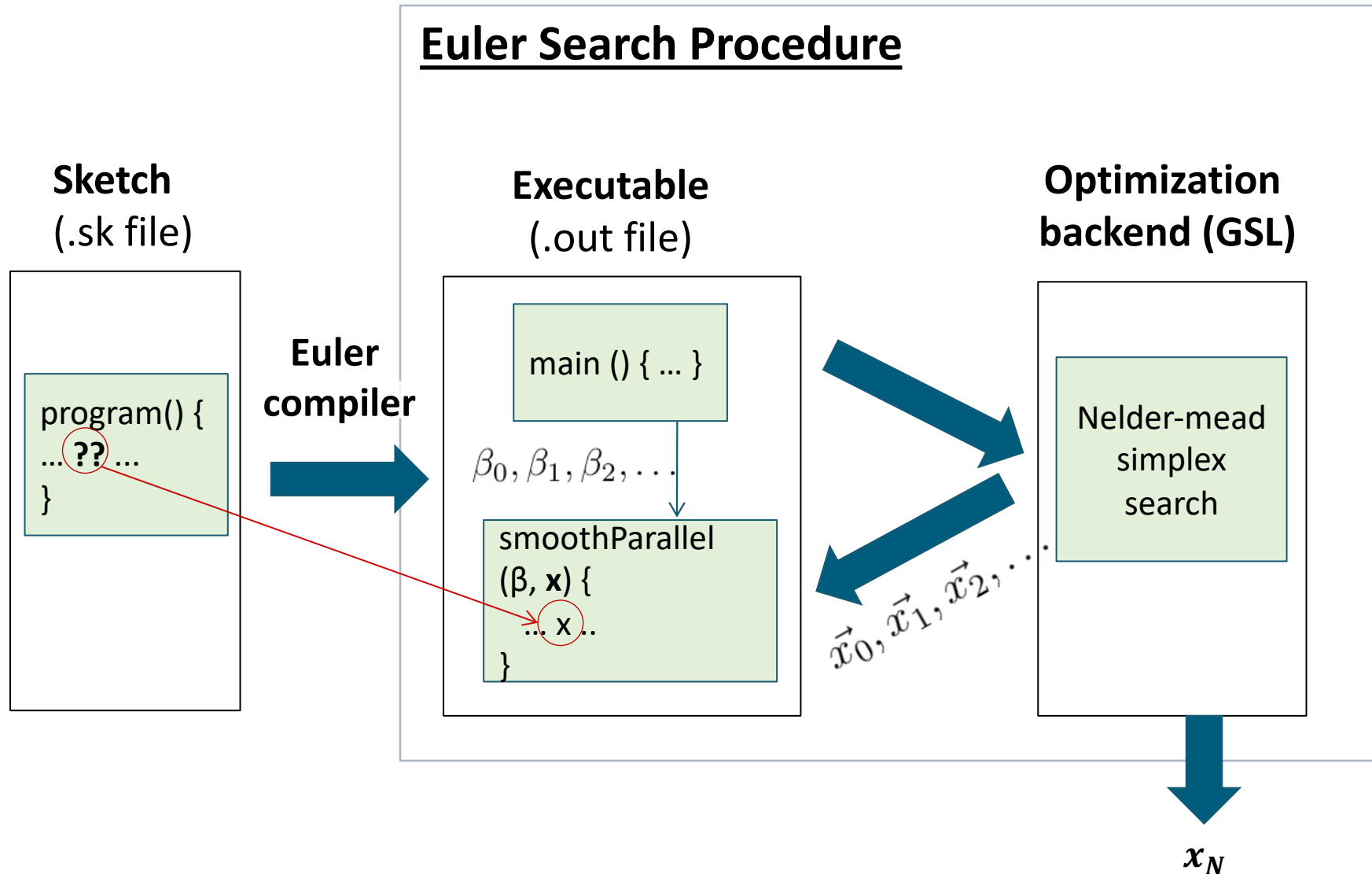
Use Symbolic Gaussian Smoothing

- β is simply the standard deviation

Quantifier alternation is handled through fixed test harnesses

- not ideal,
- but works well in practice

Smoothed Numerical Search in Euler

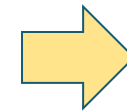
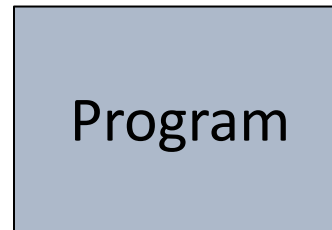
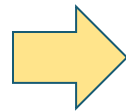
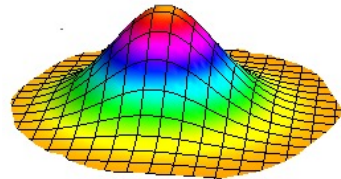
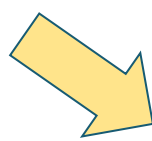


Smooth Interpretation

How do we execute a program on a distribution of inputs?

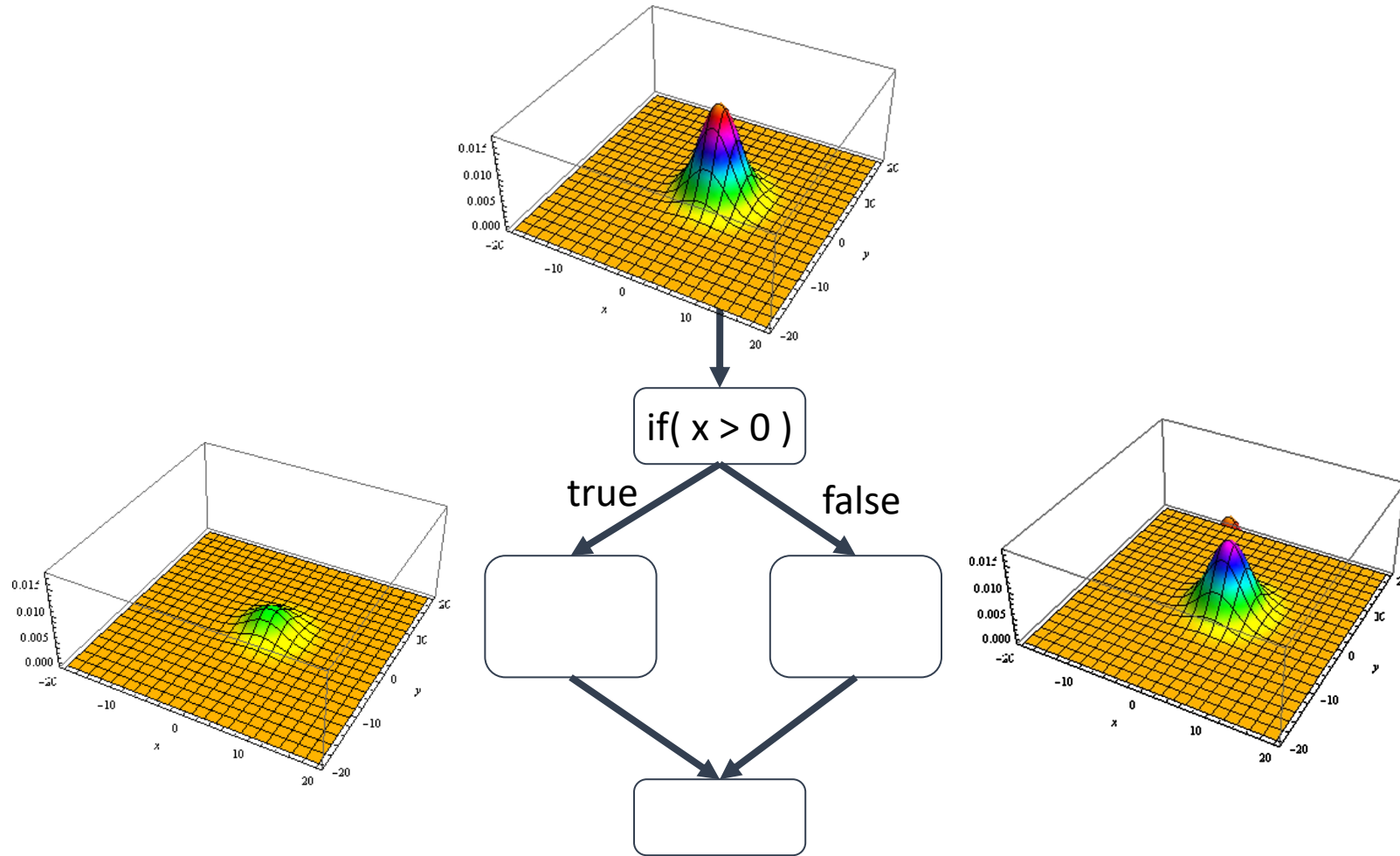
- Not by sampling
 - we would miss essential features of the search landscape.
- Symbolic execution:
 - propagate symbolic representations of distributions.

Input x

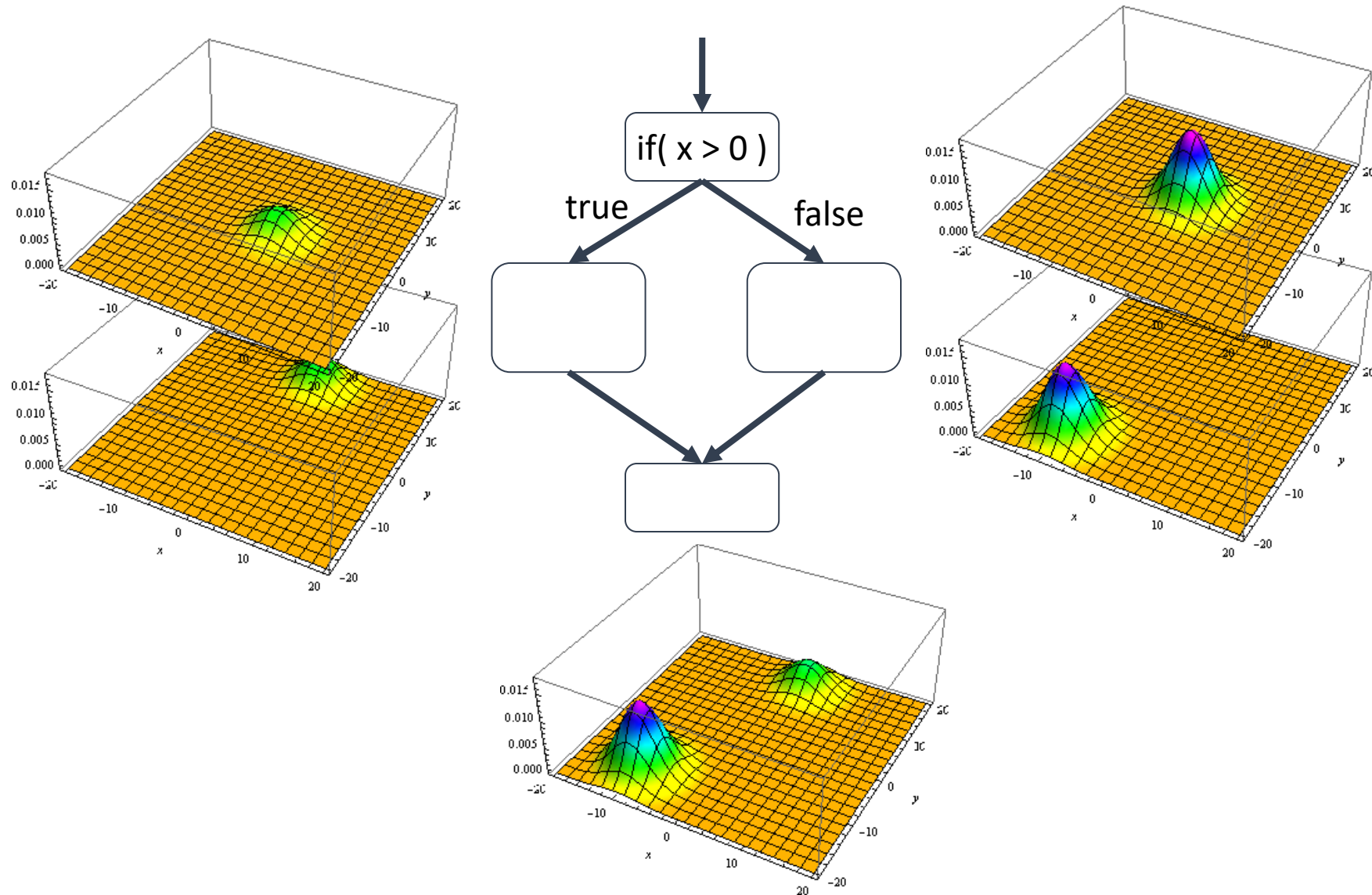


Approximation of
expected
output

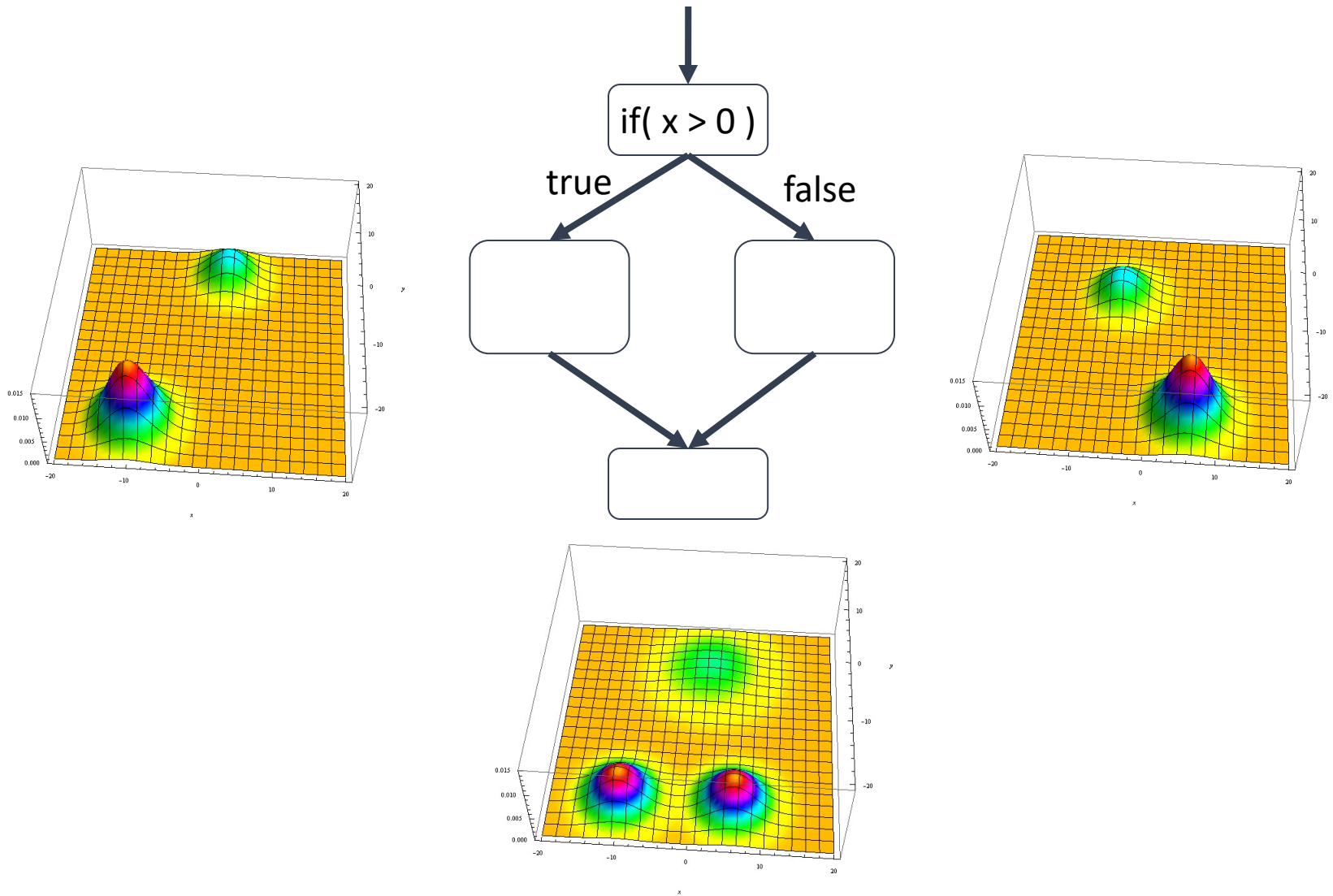
Smooth Interpretation: Branch



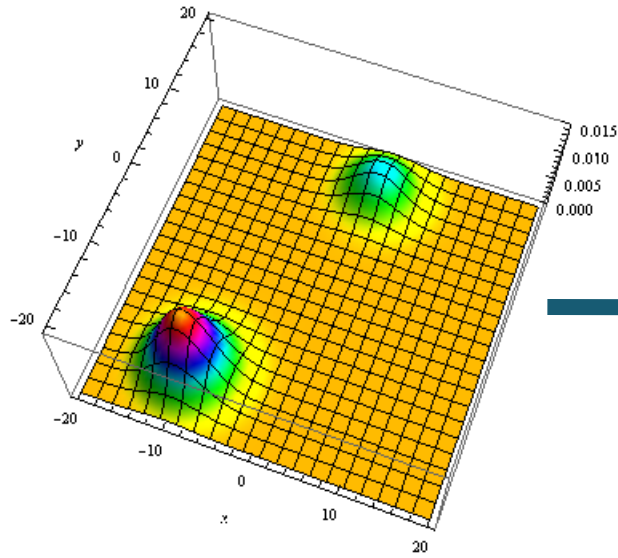
Smooth Interpretation: Join



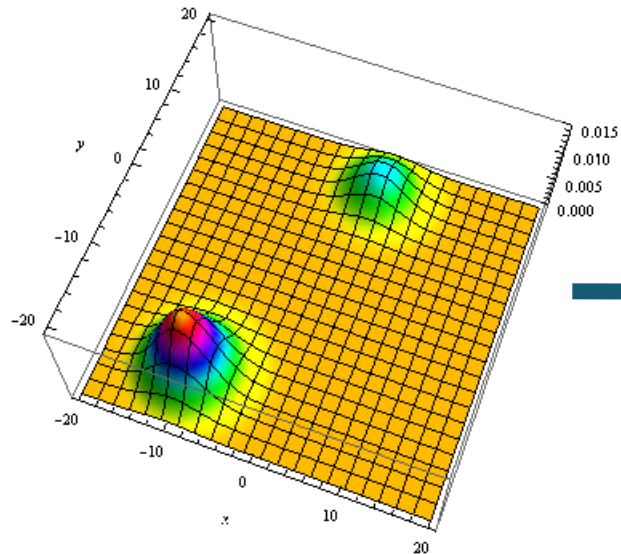
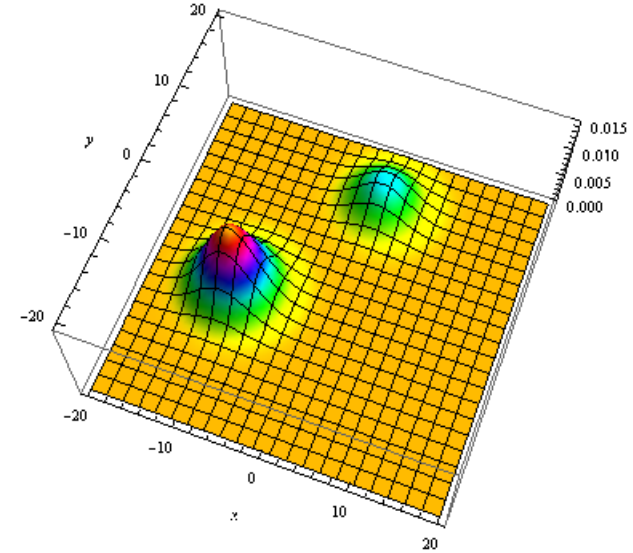
Smooth Interpretation: Join



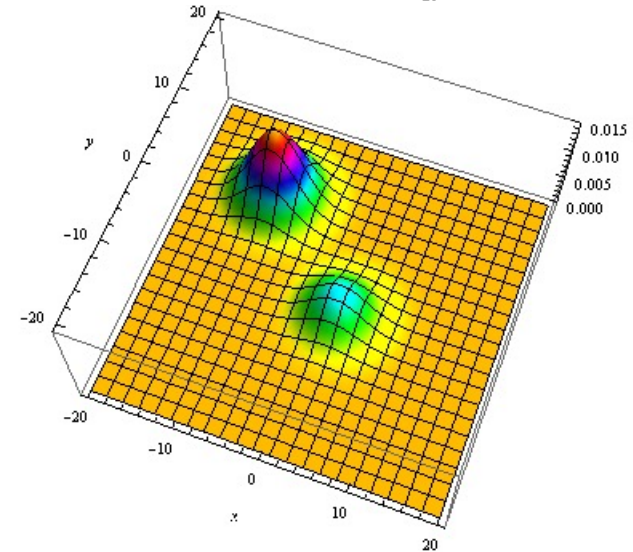
Smooth Interpretation: Assignment



$$x = x - y;$$



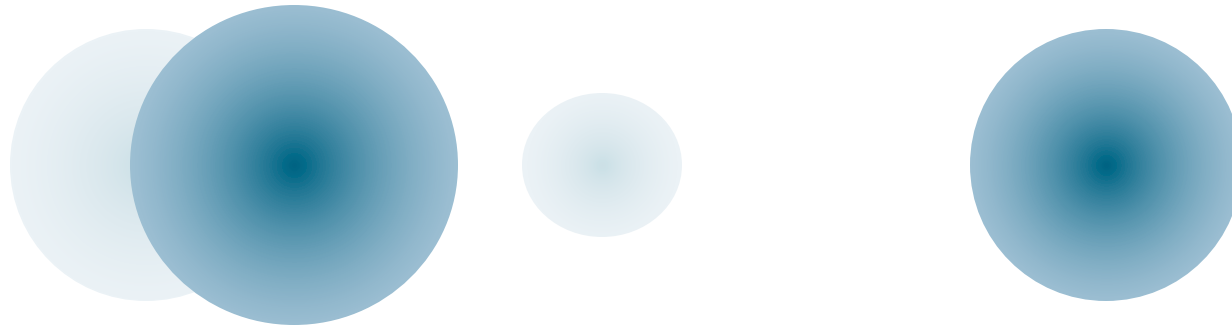
$$x = -y;$$



Approximations

Trick Question:

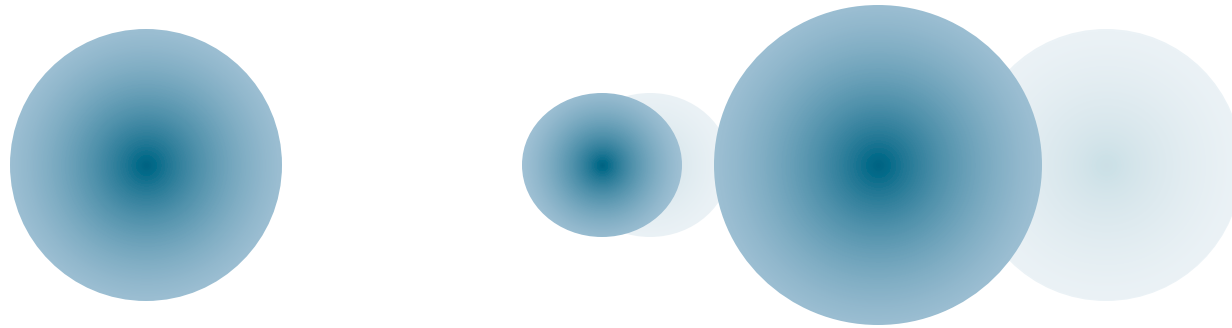
- Which of the previous approximations is problematic?



Approximations

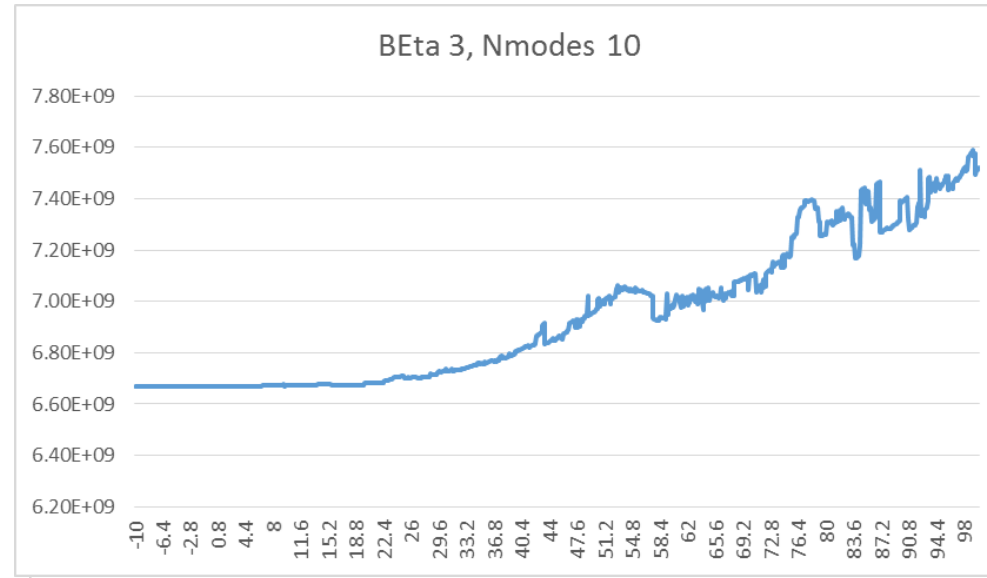
Trick Question:

- Which of the previous approximations is problematic?



Merging can introduce discontinuities!

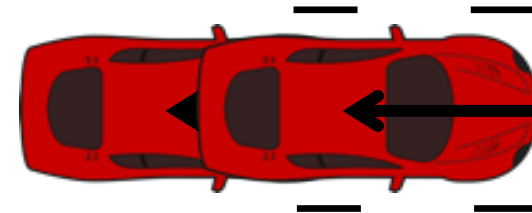
Excessive merging can cause problems



Examples

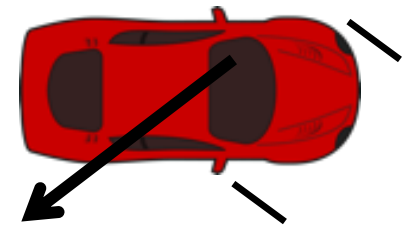
Parameter synthesis

```
t1 := ??;   t2 := ??; A1 := ??;  
A2 := ??;  
repeat (every ΔT) {  
  if (stage = BACK && time > t1)  
    stage = INTURN;  
  if (stage = INTURN) {  
    if (time > t2) then stage = OUTTURN;  
    angle = angle - A1; }  
  if (stage = OUTTURN)  
    angle = angle + A2;  
  moveCar();  
}
```



Parameter synthesis

```
t1 := ??;   t2 := ??; A1 := ??;  
A2 := ??;  
repeat (every ΔT) {  
  if (stage = BACK && time > t1)  
    stage = INTURN;  
  if (stage = INTURN) {  
    if (time > t2) then stage = OUTTURN;  
    angle = angle - A1; }  
  if (stage = OUTTURN)  
    angle = angle + A2;  
  moveCar();  
}
```



Parameter synthesis

```
t1 := ??;   t2 := ??;  A1 := ??;  
A2 := ??;  
repeat (every ΔT) {  
    if (stage = BACK && time > t1)  
        stage = INTURN;  
    if (stage = INTURN) {  
        if (time > t2) then stage = OUTTURN;  
        angle = angle - A1; }  
    if (stage = OUTTURN)  
        angle = angle + A2;  
    moveCar();  
}
```

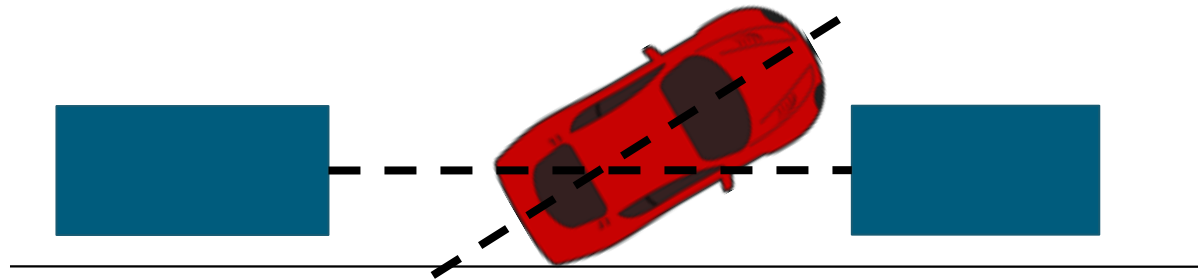


Application : Parameter synthesis

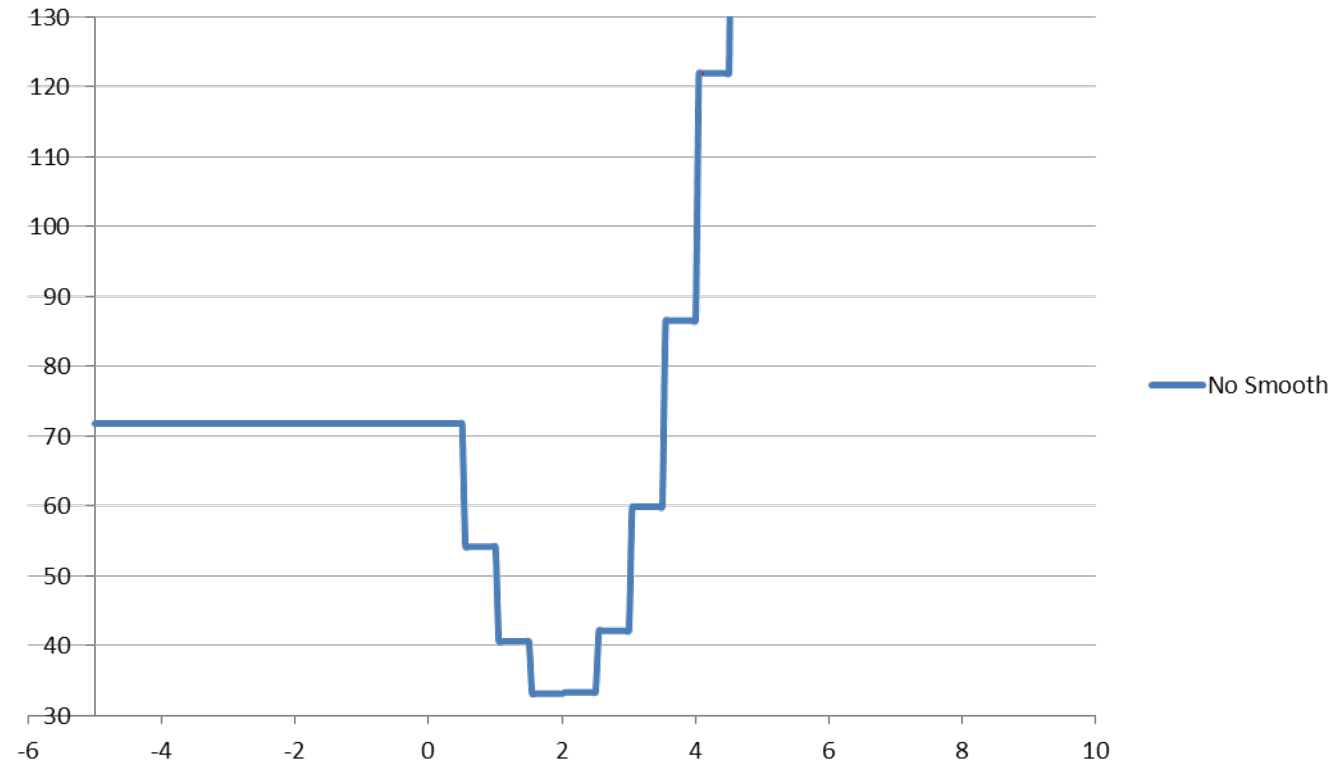
```
t1 := ??;   t2 := ??; A1 := ??;  
A2 := ??;  
repeat (every ΔT) {  
  if (stage = BACK && time > t1)  
    stage = INTURN;  
  if (stage = INTURN) {  
    if (time > t2) then stage = OUTTURN;  
    angle = angle - A1; }  
  if (stage = OUTTURN)  
    angle = angle + A2;  
  moveCar();  
}
```

**Error = distance between
ideal and actual final position**

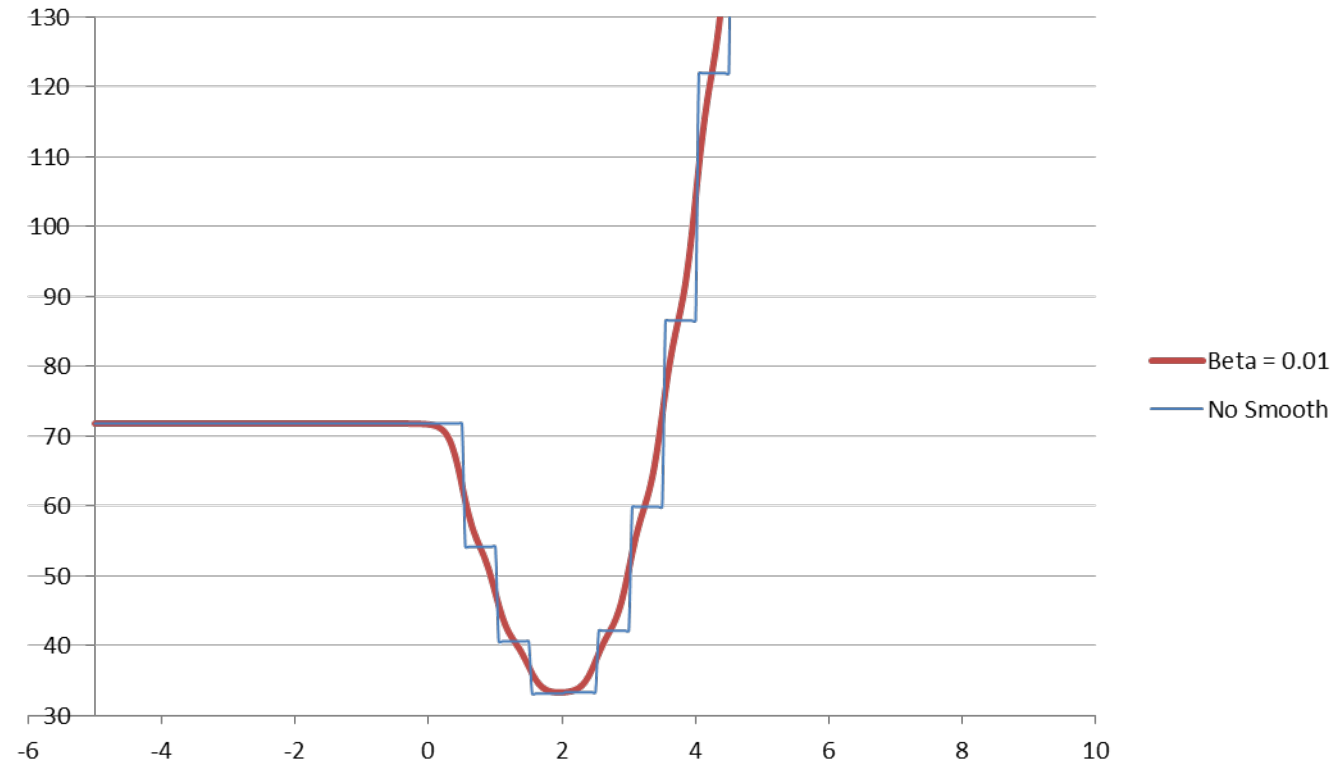
Goal: minimize Error.



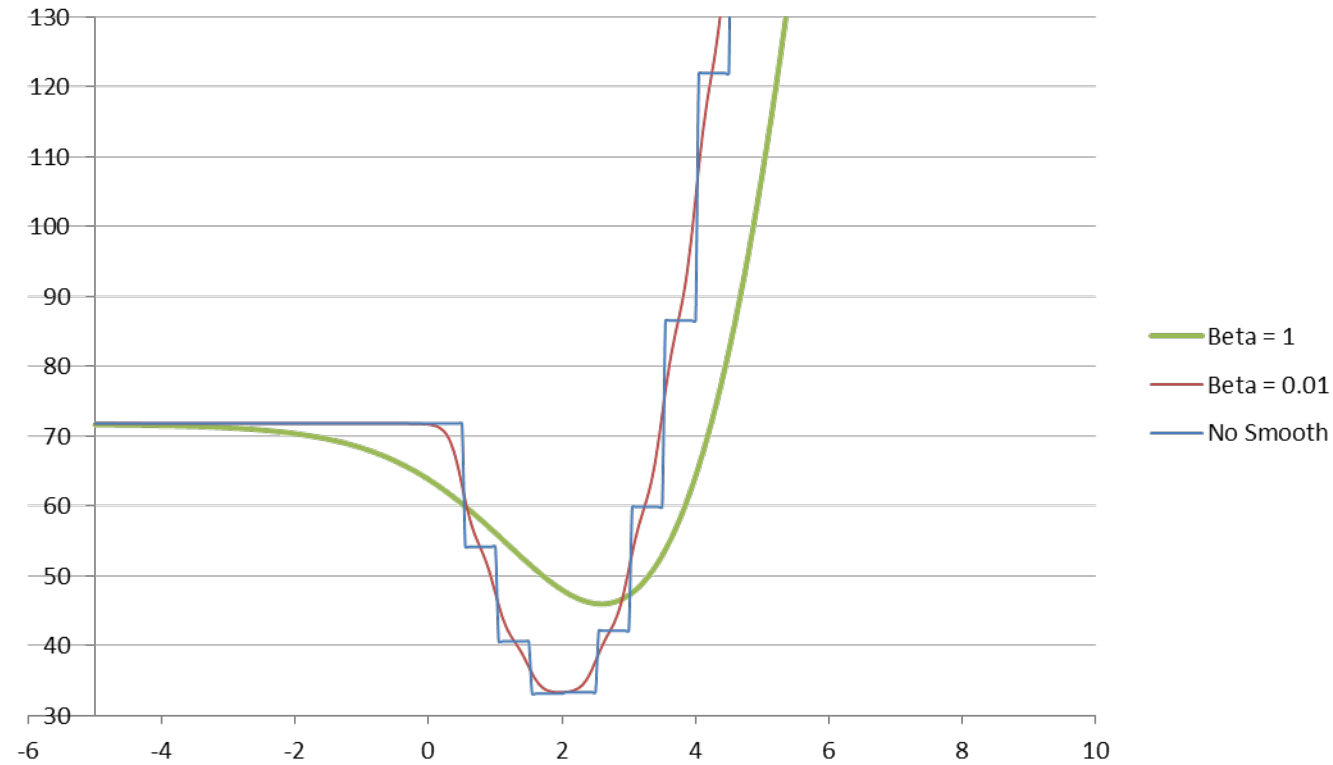
Slice of Error function WRT T2



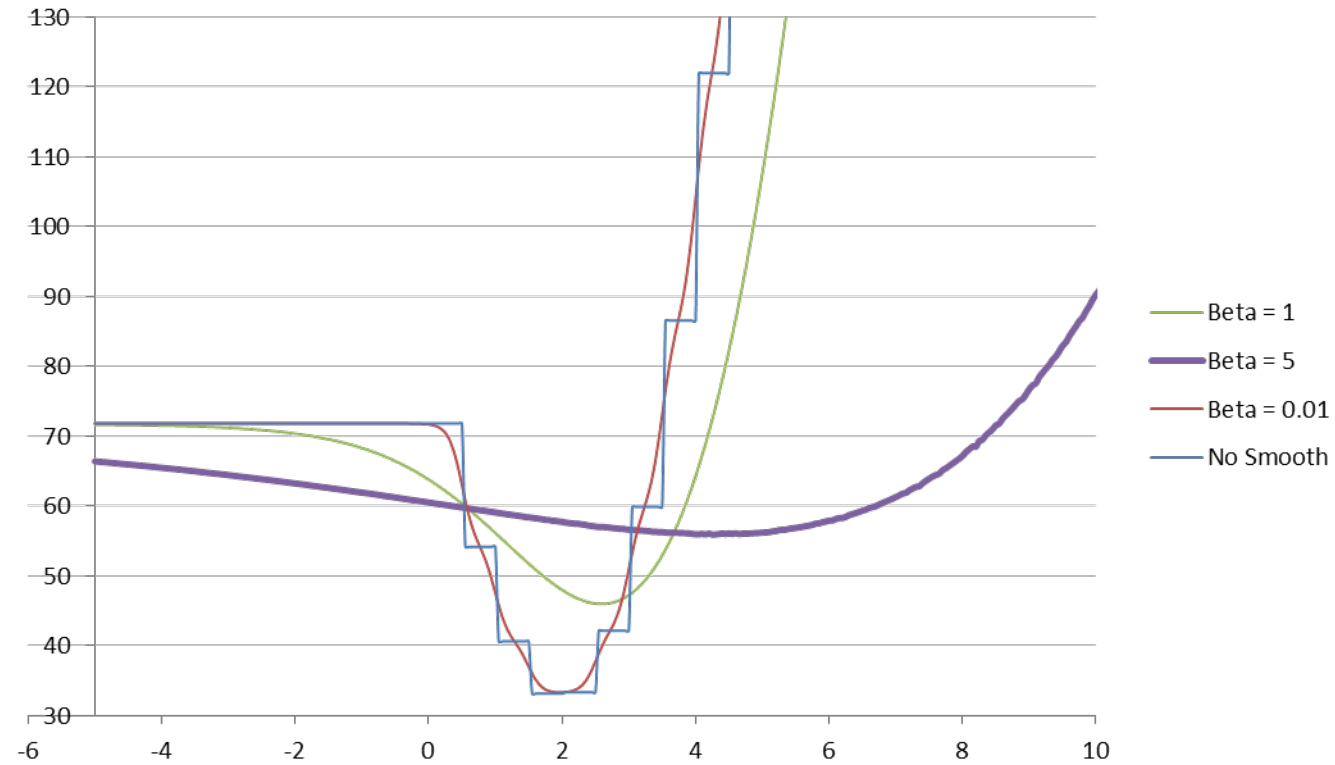
Slice of Error function WRT T_2



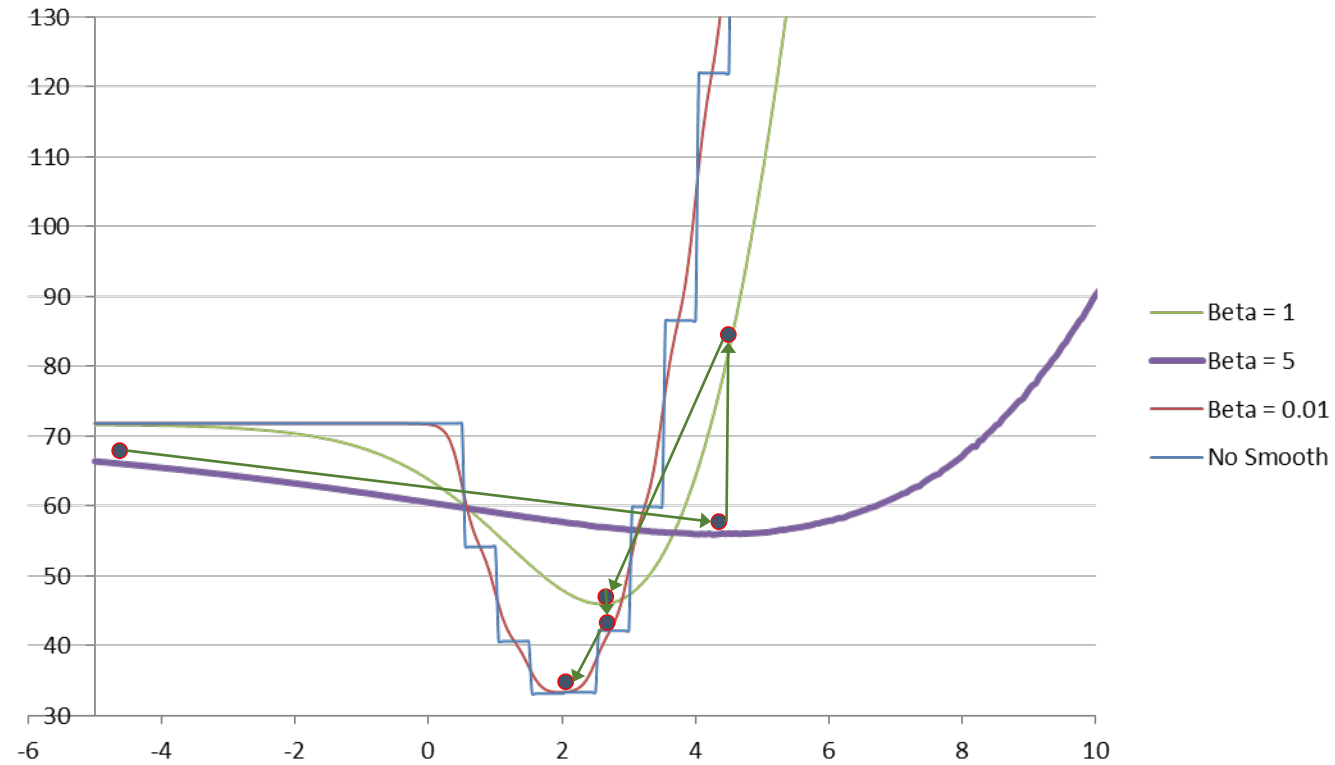
Slice of Error function WRT T_2



Slice of Error function WRT T_2



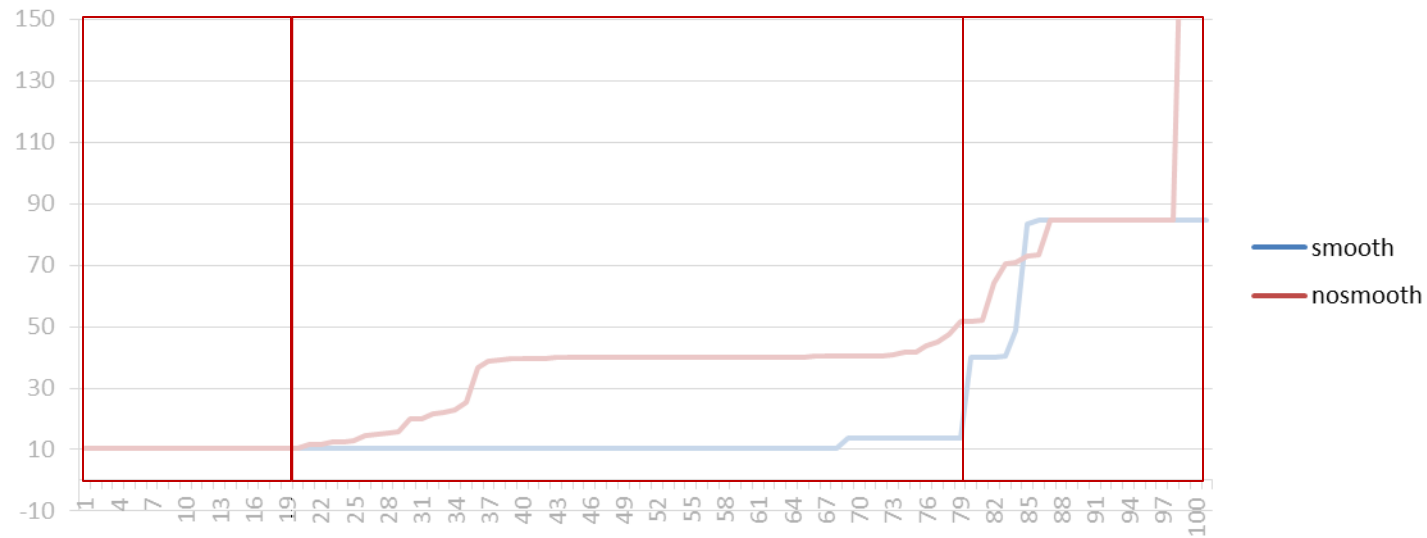
Numerical Search with Smoothing



Errors from different starting points

Both are quite good

Both are quite bad



Result of smoothing is much better