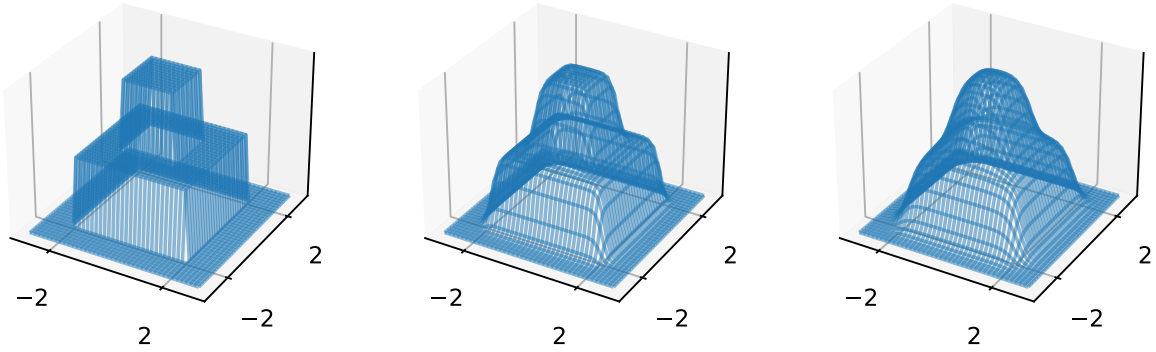
# Smoothing Discontinuous Concatenated Functions

Sebastian Christodoulou

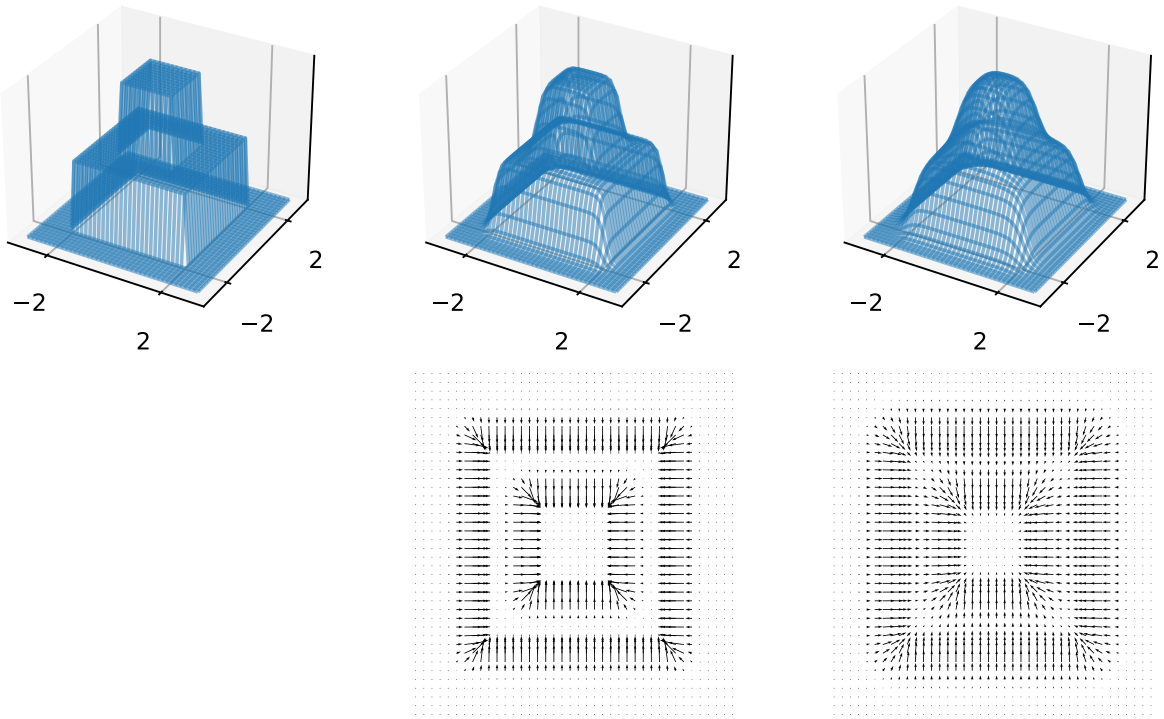
STCE, RWTH Aachen

September 8, 2022

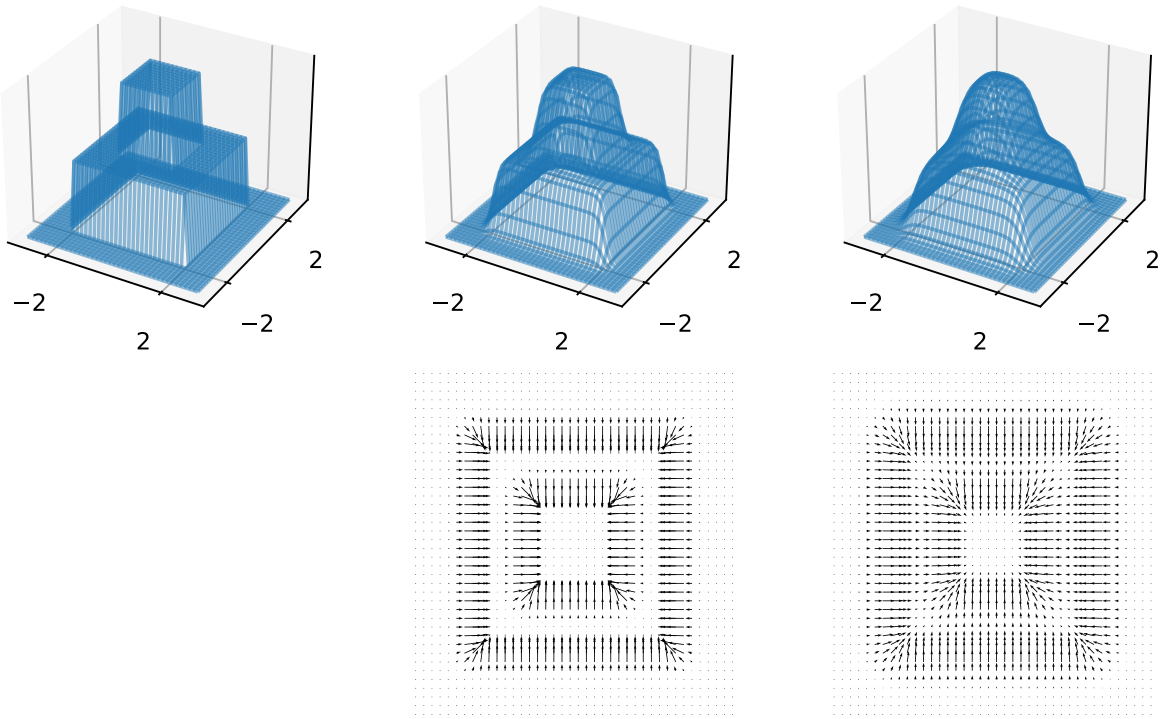
# Idea



# Idea

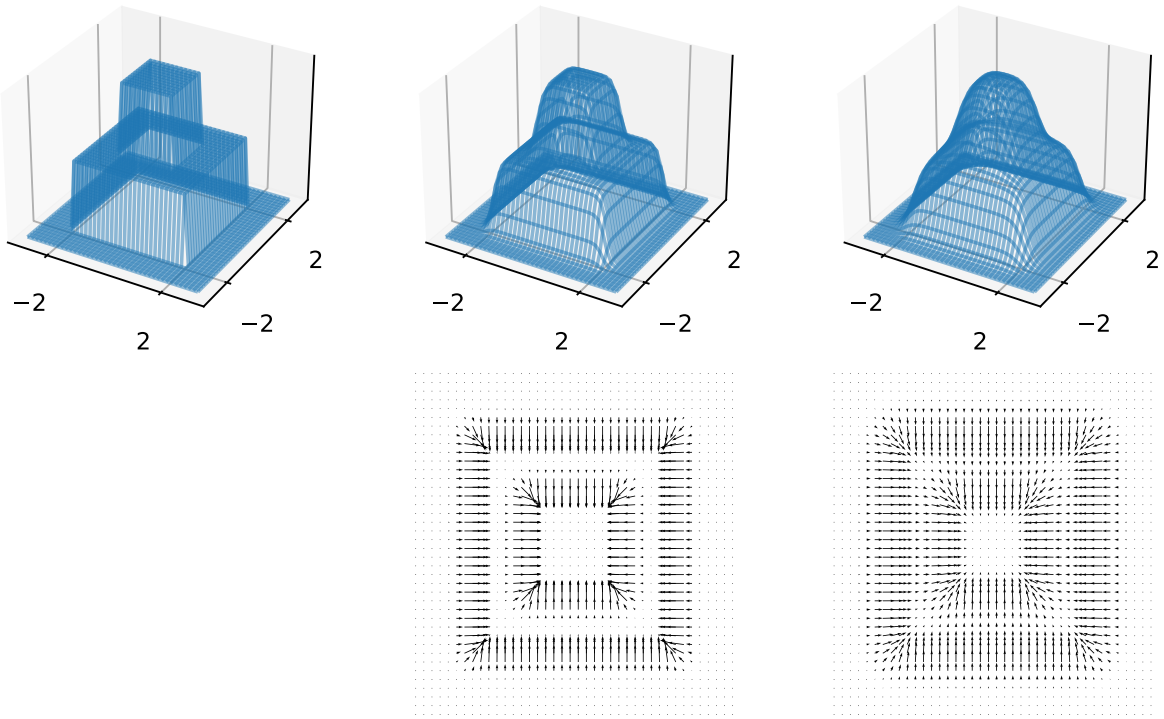


# Idea



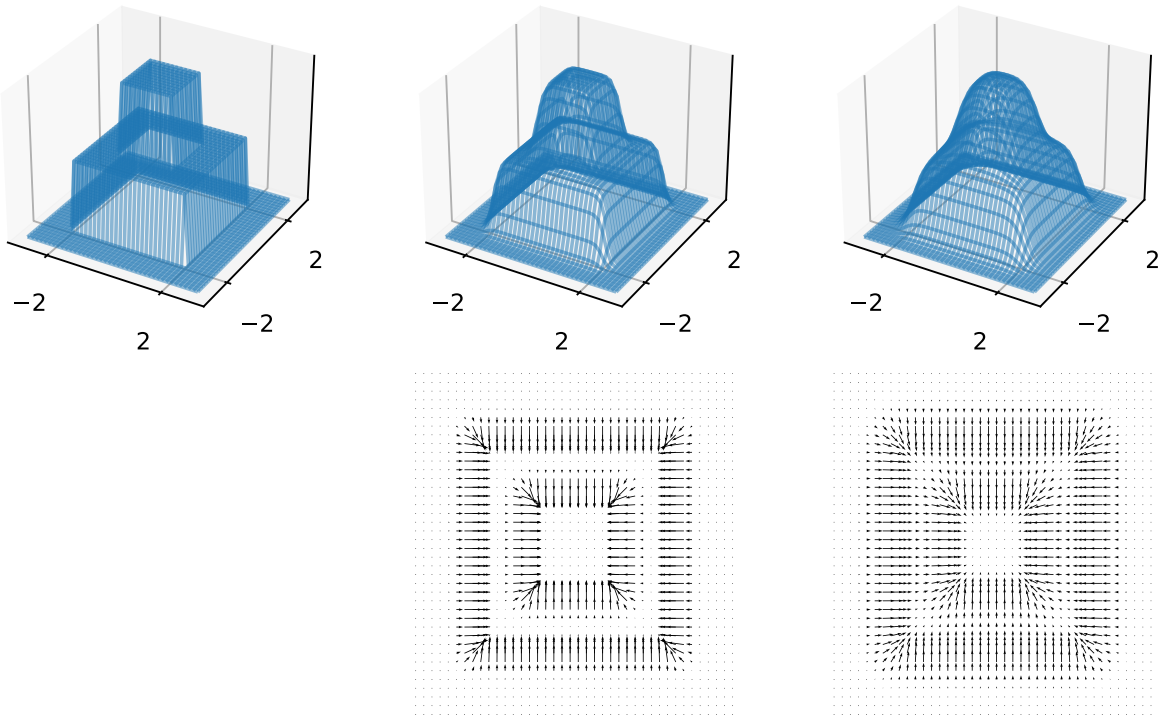

Smoothing

# Idea




Smoothing, independent of input-dimensionality

# Idea



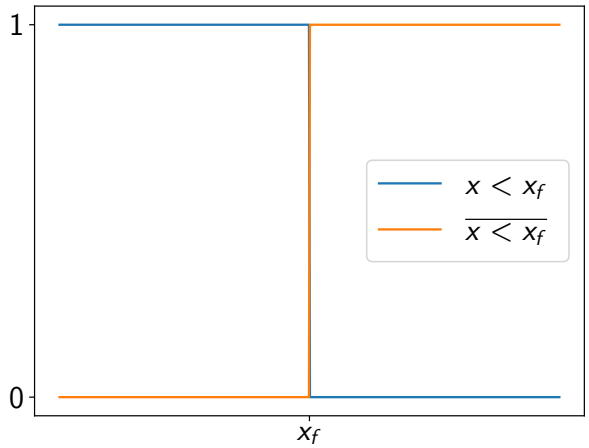

Smoothing, independent of input-dimensionality
with 'few' function-evaluations

# Idea

We will smooth $f \circ g$ where

$$f = \begin{cases} f_1, & \text{if } x < x_f \\ f_2, & \text{else} \end{cases} \quad \textit{and} \quad g = \begin{cases} g_1, & \text{if } x < x_g \\ g_2, & \text{else} \end{cases}$$

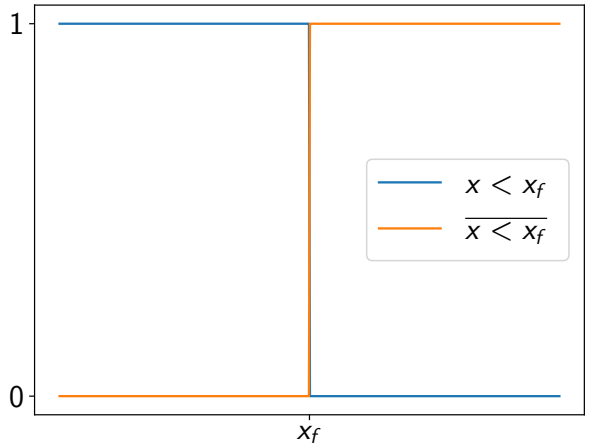
# Smoothing Concatenations



# Smoothing Concatenations

A sigmoid gives the *probability* of a condition being fulfilled

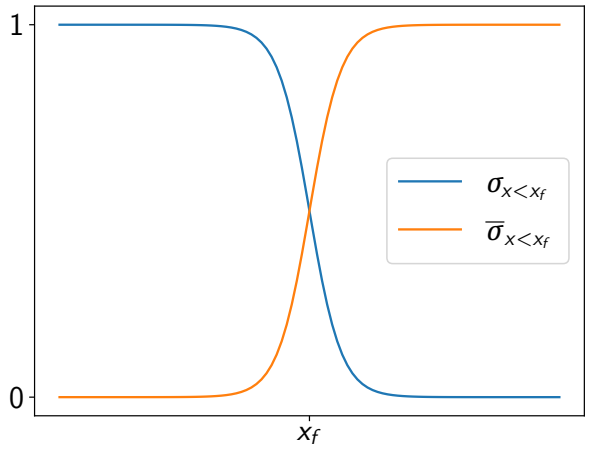
$$\sigma_{x<x_f}(x) = \frac{1}{1 + e^{x - x_f}} \qquad \overline{\sigma}_{x<x_f} = 1 - \sigma_{x<x_f}$$



# Smoothing Concatenations

A sigmoid gives the *probability* of a condition being fulfilled

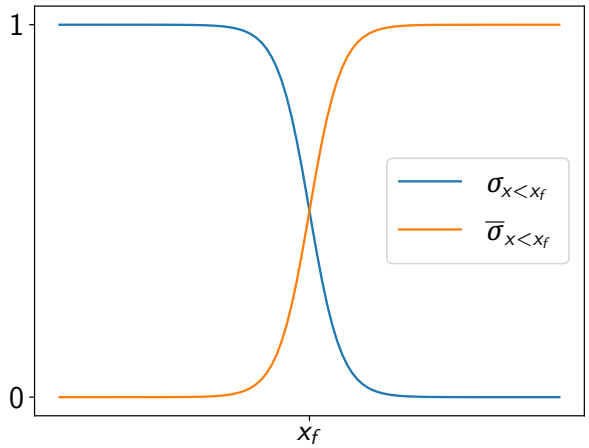
$$\sigma_{x<x_f}(x) = \frac{1}{1 + e^{x - x_f}} \qquad \overline{\sigma}_{x<x_f} = 1 - \sigma_{x<x_f}$$



# Smoothing Concatenations

A sigmoid gives the *probability* of a condition being fulfilled

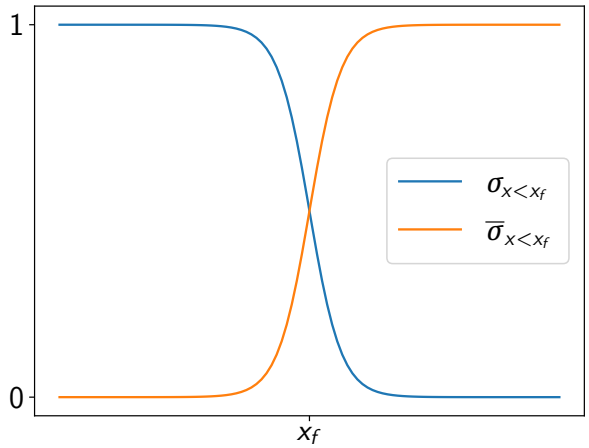
$$\sigma_{x<x_f}(x) = \frac{1}{1 + e^{x - x_f}} \qquad \overline{\sigma}_{x<x_f} = 1 - \sigma_{x<x_f} =: \sigma_{\overline{x<x_f}}$$



# Smoothing Concatenations

A sigmoid gives the *probability* of a condition being fulfilled

$$\sigma_{x<x_f}(x) = \frac{1}{1+e^{x-x_f}} \qquad \overline{\sigma}_{x<x_f} = 1 - \sigma_{x<x_f} =: \sigma_{\overline{x<x_f}}$$




We smooth by interpolation with sigmoids

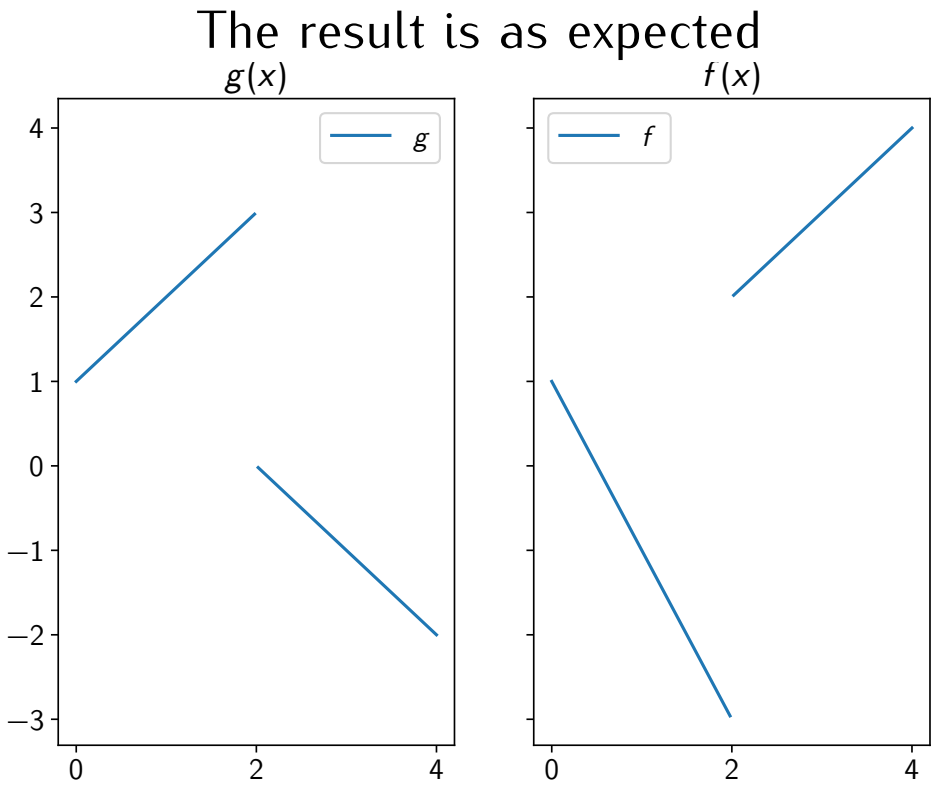
$$f_{sm} = \sigma_{x<x_f} f_1 + \overline{\sigma}_{x<x_f} f_2$$

$$g_{sm} = \sigma_{x<x_g} g_1 + \overline{\sigma}_{x<x_g} g_2$$

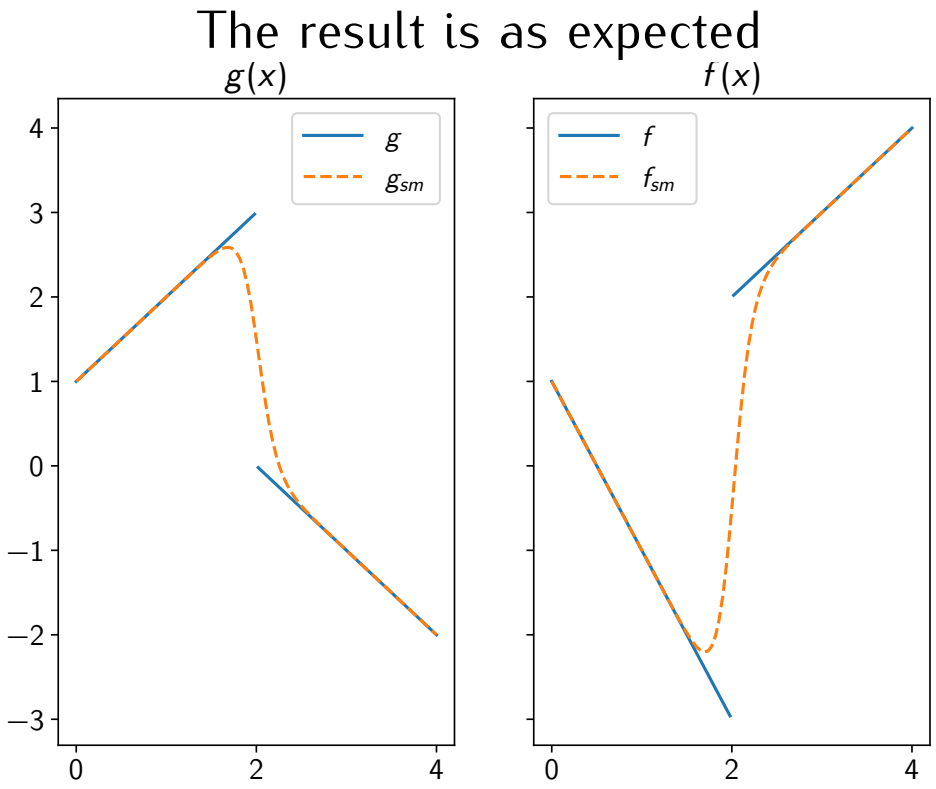
# Smoothing Concatenations

The result is as expected

# Smoothing Concatenations

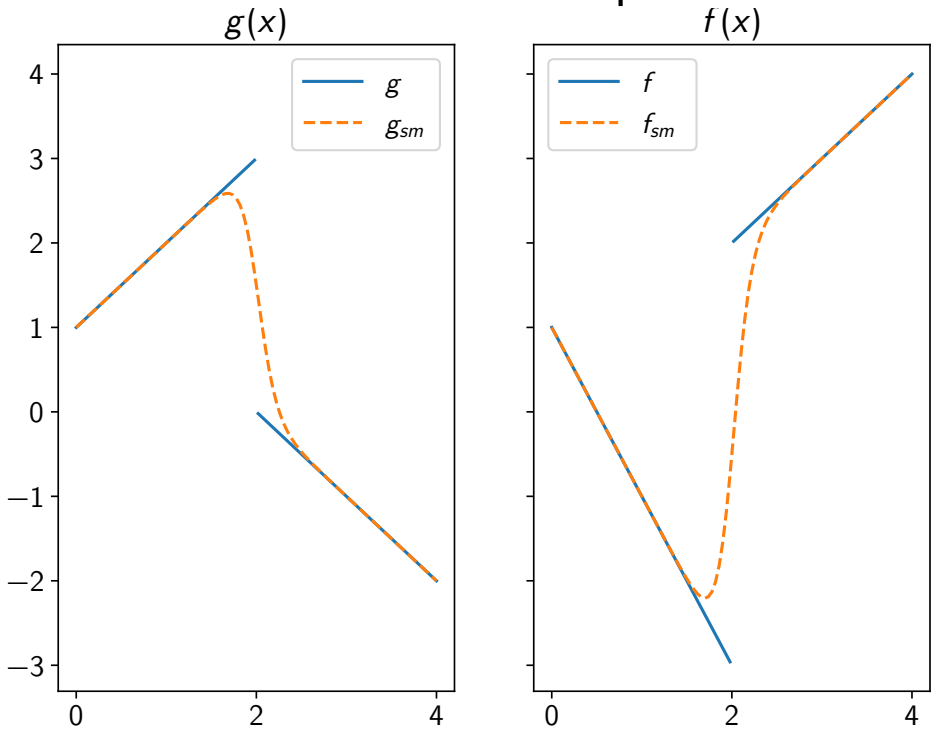


# Smoothing Concatenations



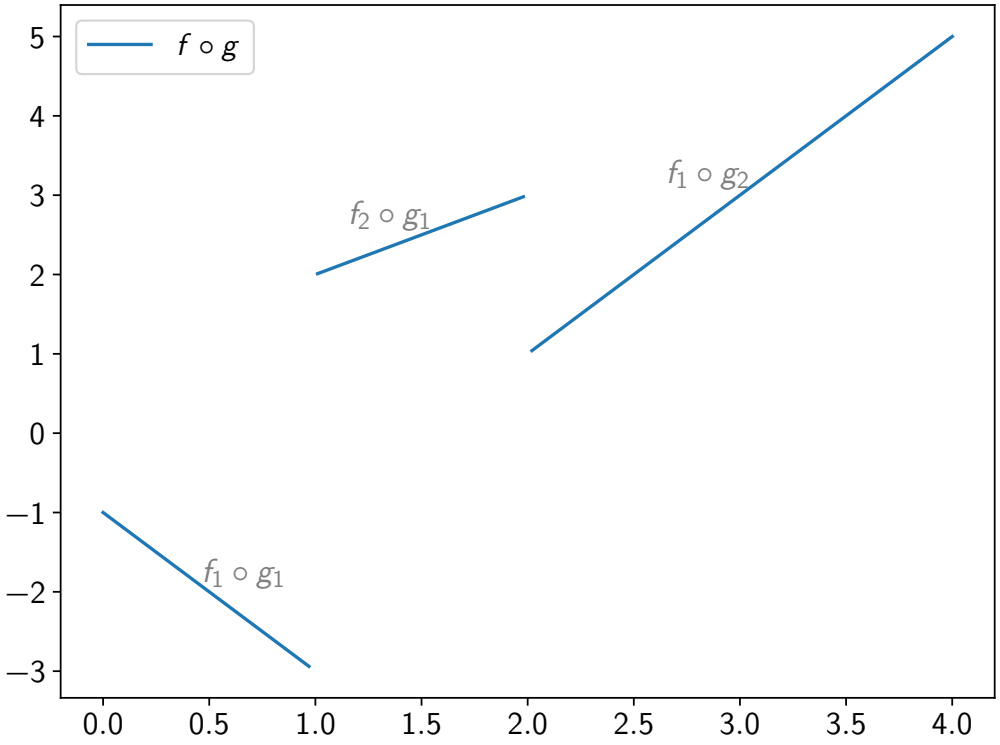
# Smoothing Concatenations

The result is as expected



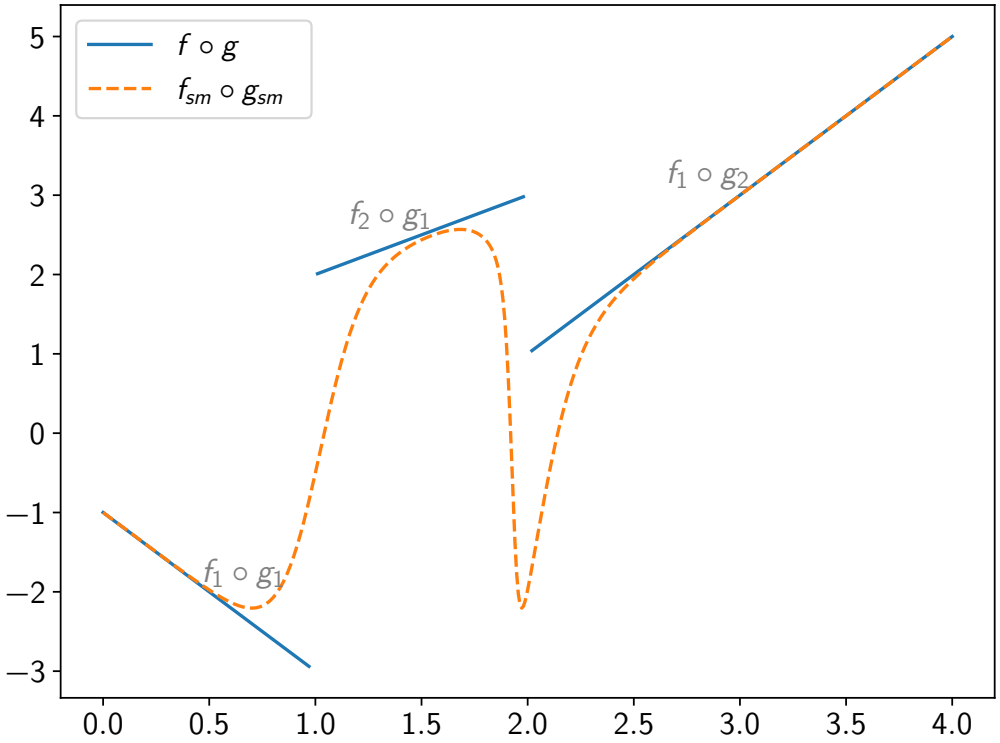

But not for concatenation

# Smoothing Concatenations



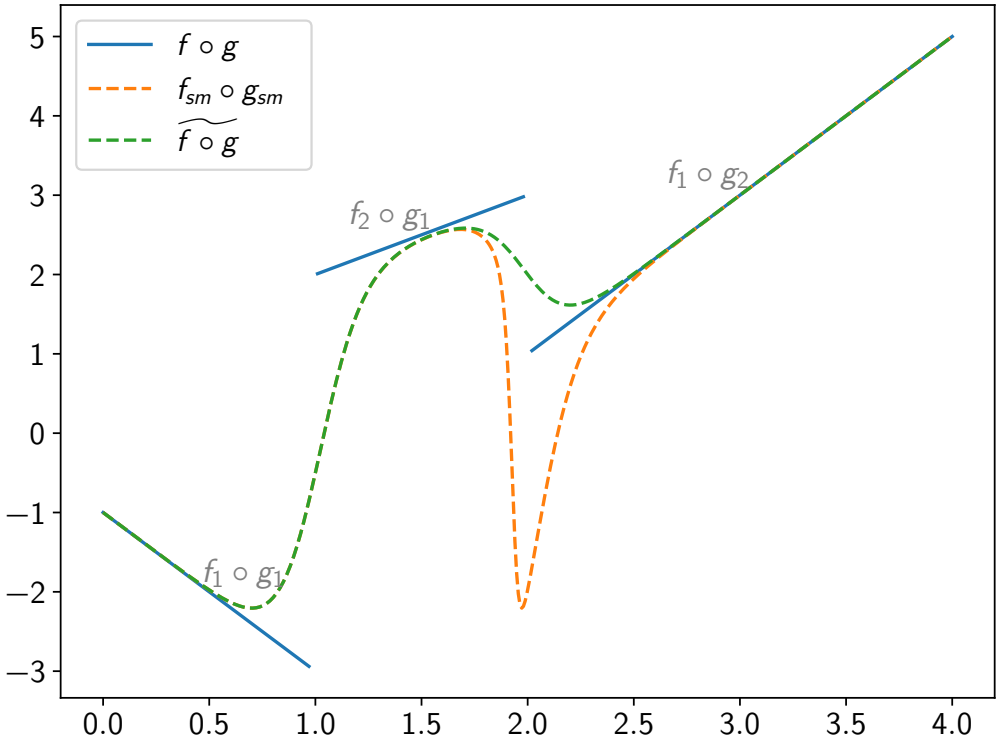
# Smoothing Concatenations

$f_{sm} \circ g_{sm}$ contains aftefacts.



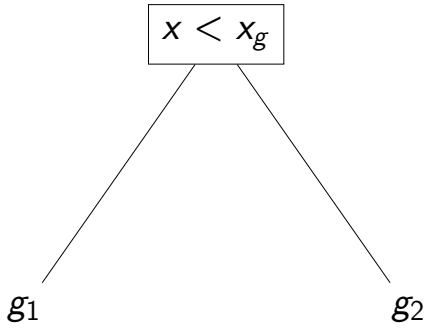
# Smoothing Concatenations

$f_{sm} \circ g_{sm}$ contains aftefacts. Better: Smooth after concatenation



# Smoothing Concatenations

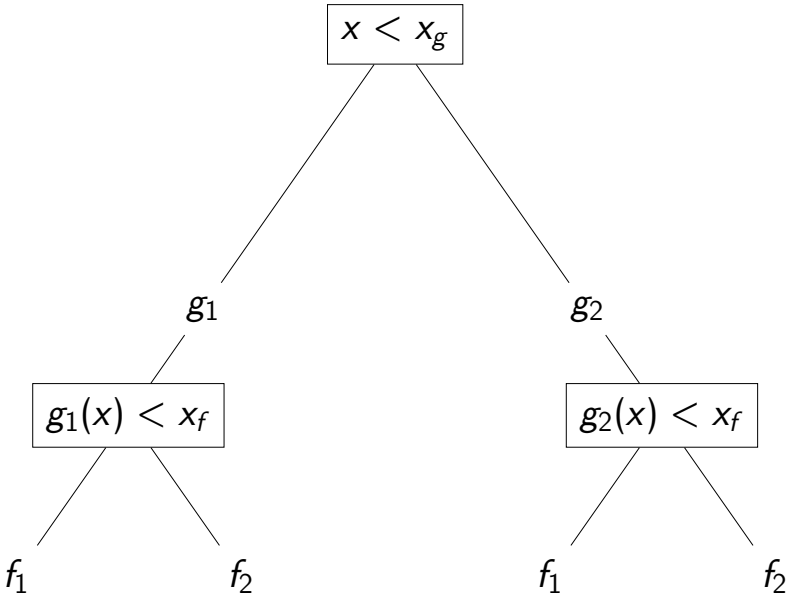
## Concatenation Tree $f \circ g$




$\widetilde{(f \circ g)} =$

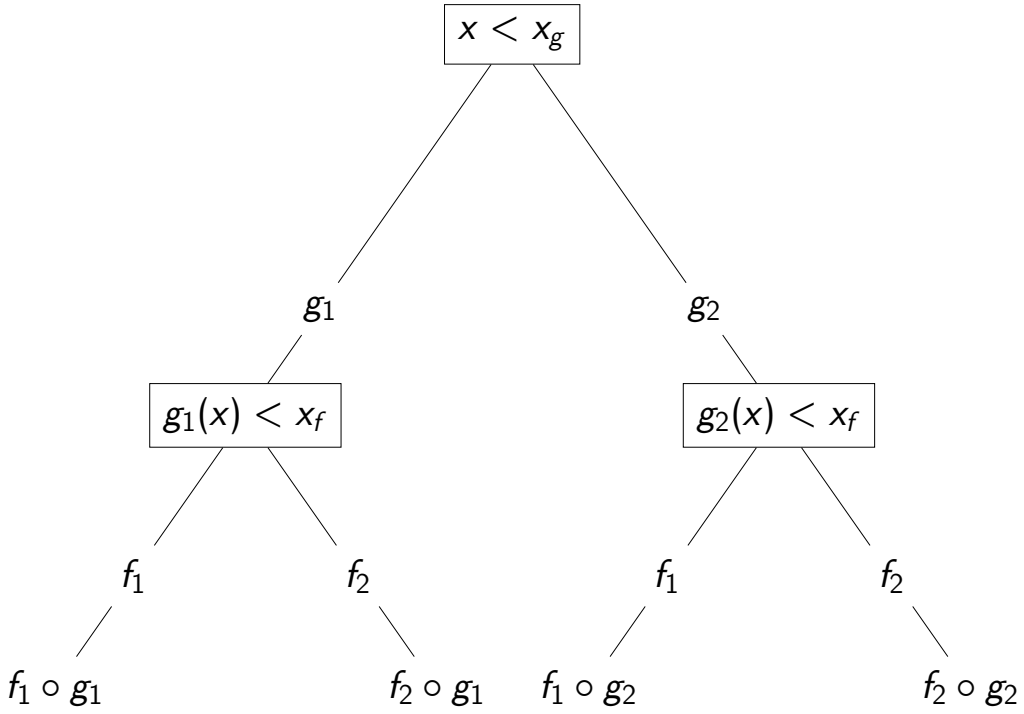
# Smoothing Concatenations

## Concatenation Tree $f \circ g$



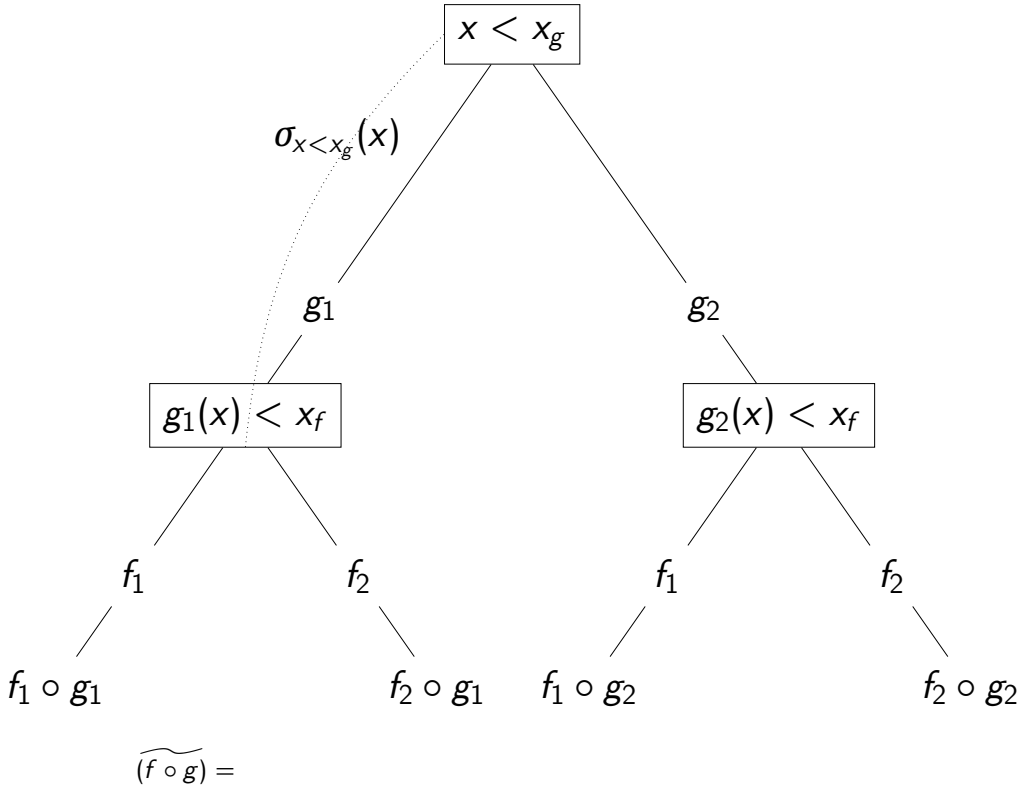

$\widetilde{(f \circ g)} =$

## Concatenation Tree $f \circ g$



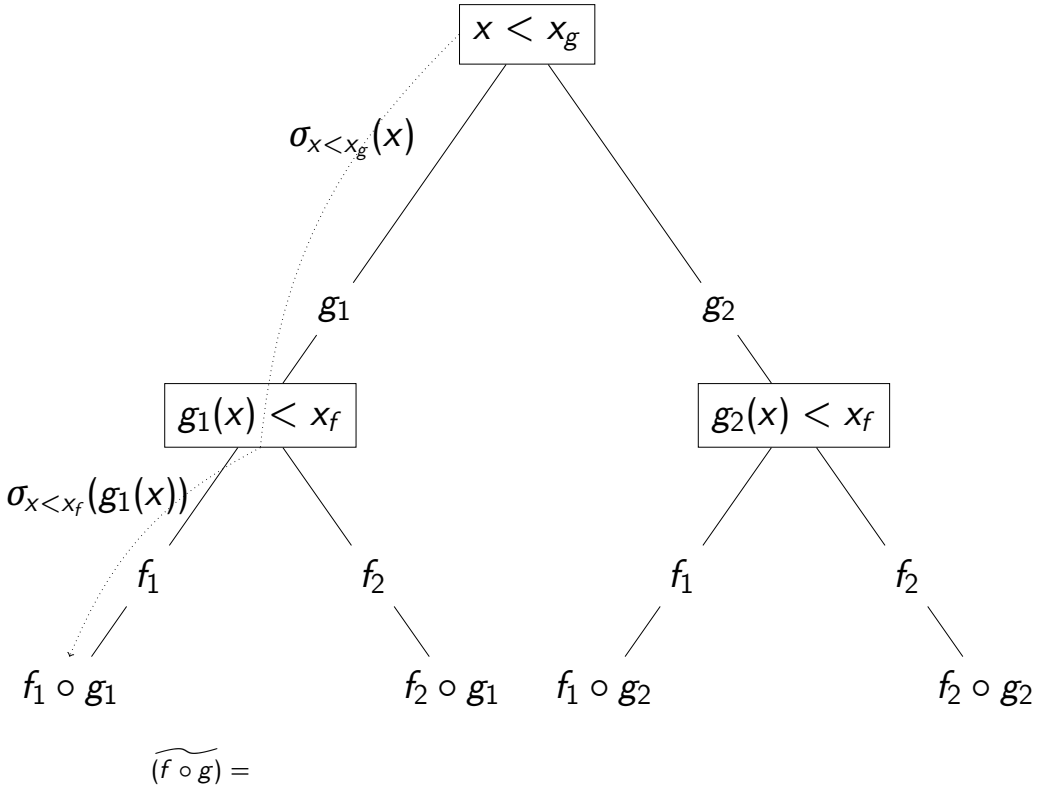

$\widetilde{(f \circ g)} =$

## Concatenation Tree $f \circ g$

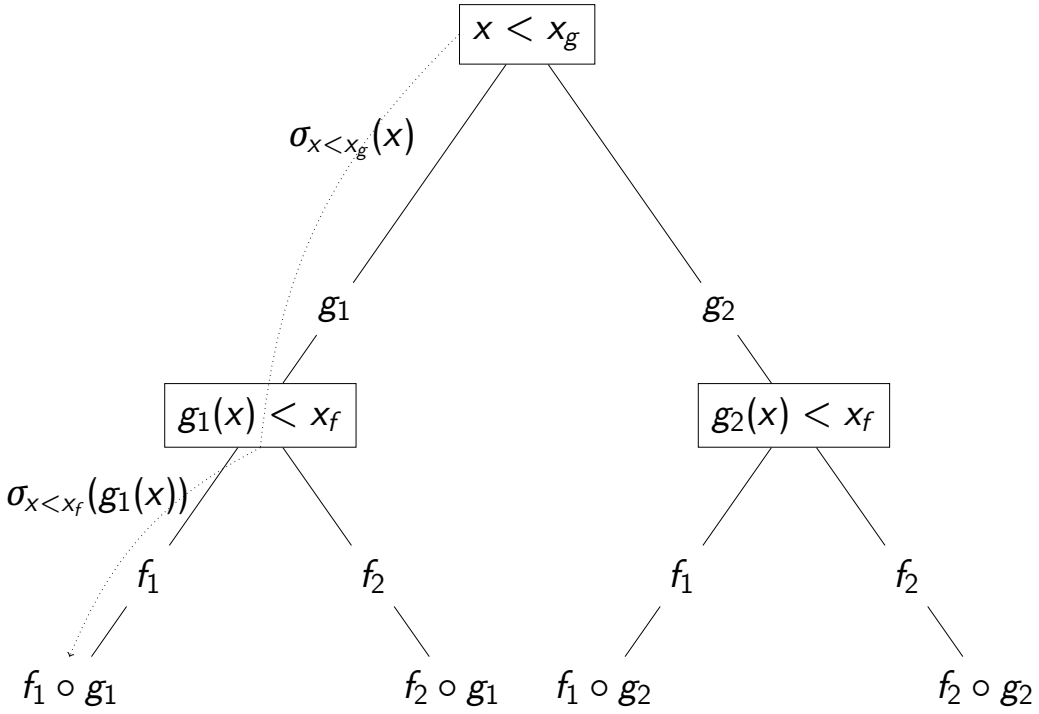


# Smoothing Concatenations

## Concatenation Tree $f \circ g$



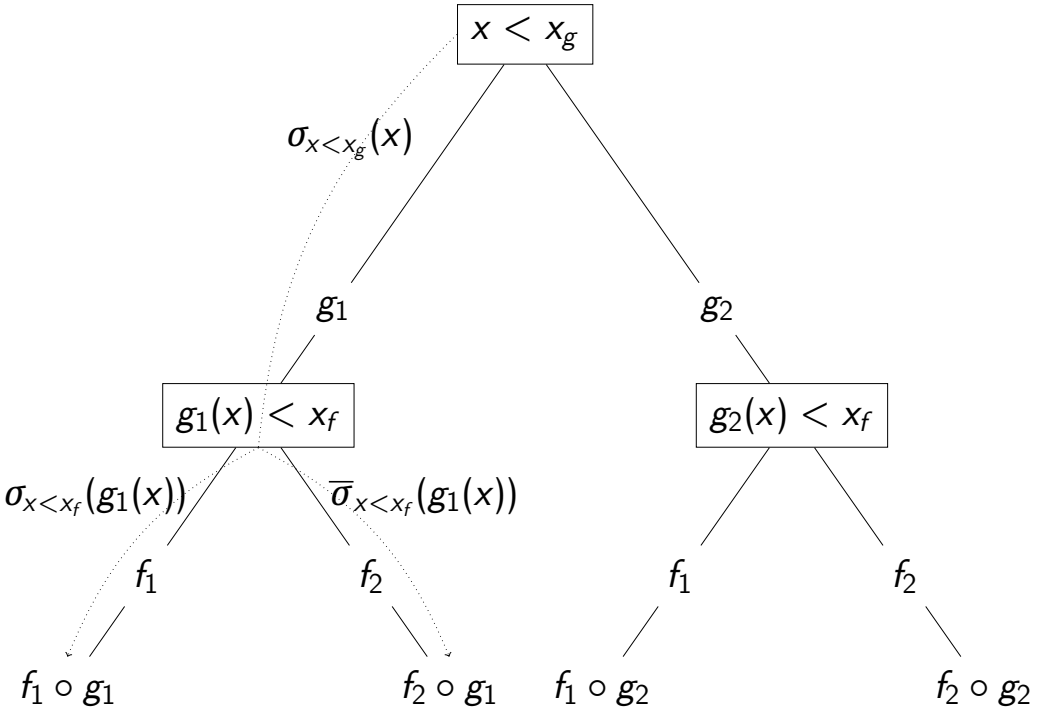
## Concatenation Tree $f \circ g$




$$\widetilde{(f \circ g)} =$$
$$\sigma_{x<x_g}(x)\ \sigma_{x<x_f}(g_1(x))\ (f_1 \circ g_1)(x) +$$

# Smoothing Concatenations

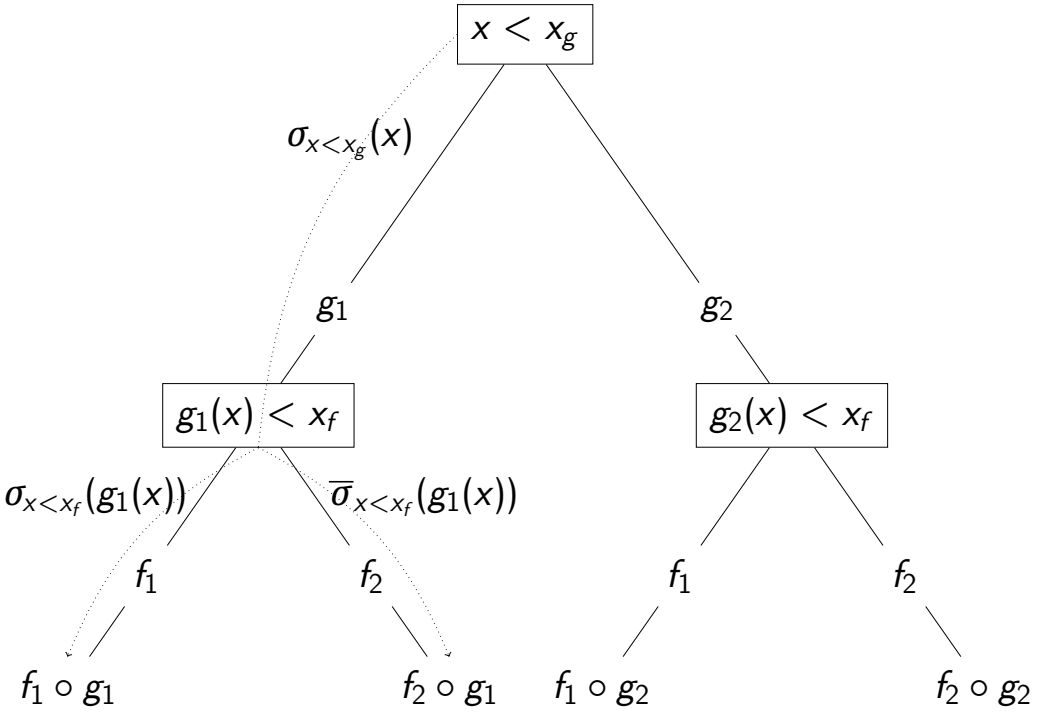
## Concatenation Tree $f \circ g$




$$\widetilde{(f \circ g)} =$$
$$\sigma_{x<x_g}(x)\ \sigma_{x<x_f}(g_1(x))\ (f_1 \circ g_1)(x) +$$

# Smoothing Concatenations

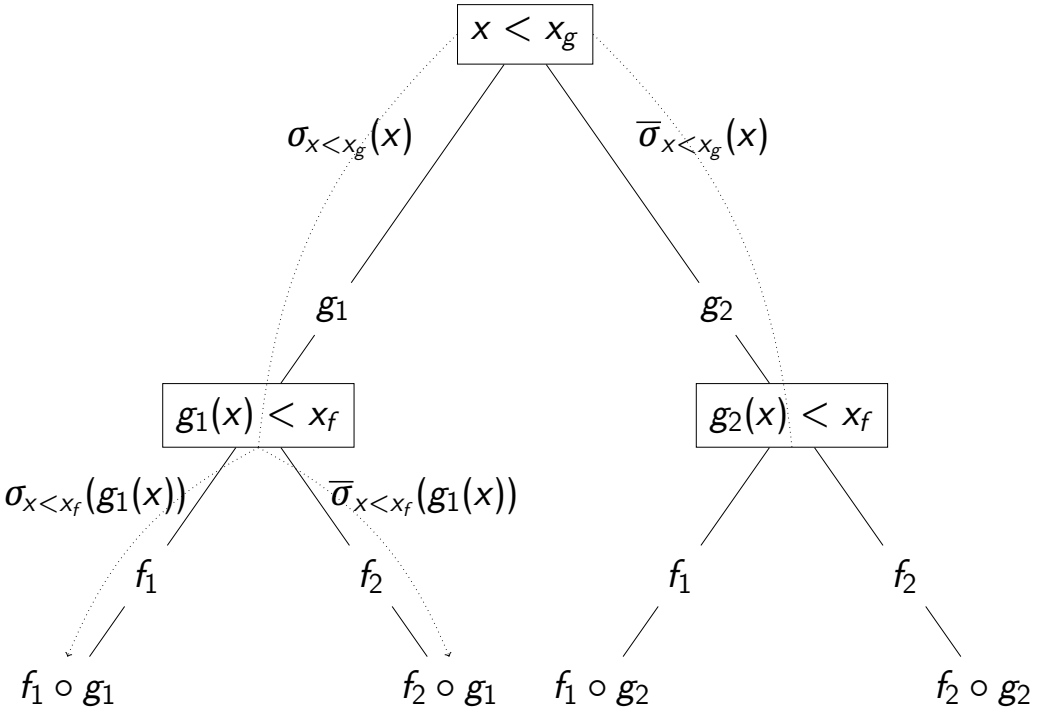
## Concatenation Tree $f \circ g$




$$\widetilde{(f \circ g)} =$$
$$\sigma_{x<x_g}(x)\ \sigma_{x<x_f}(g_1(x))\ (f_1 \circ g_1)(x) + \sigma_{x<x_g}(x)\ \overline{\sigma}_{x<x_f}(g_1(x))\ (f_2 \circ g_1)(x) +$$

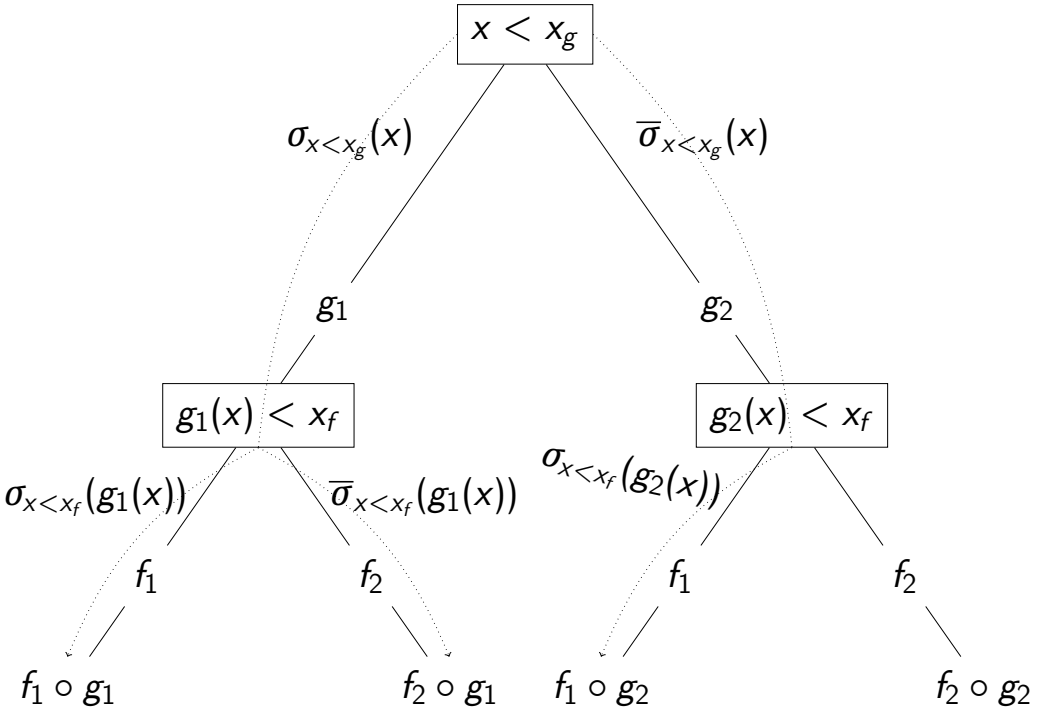
# Smoothing Concatenations

## Concatenation Tree $f \circ g$



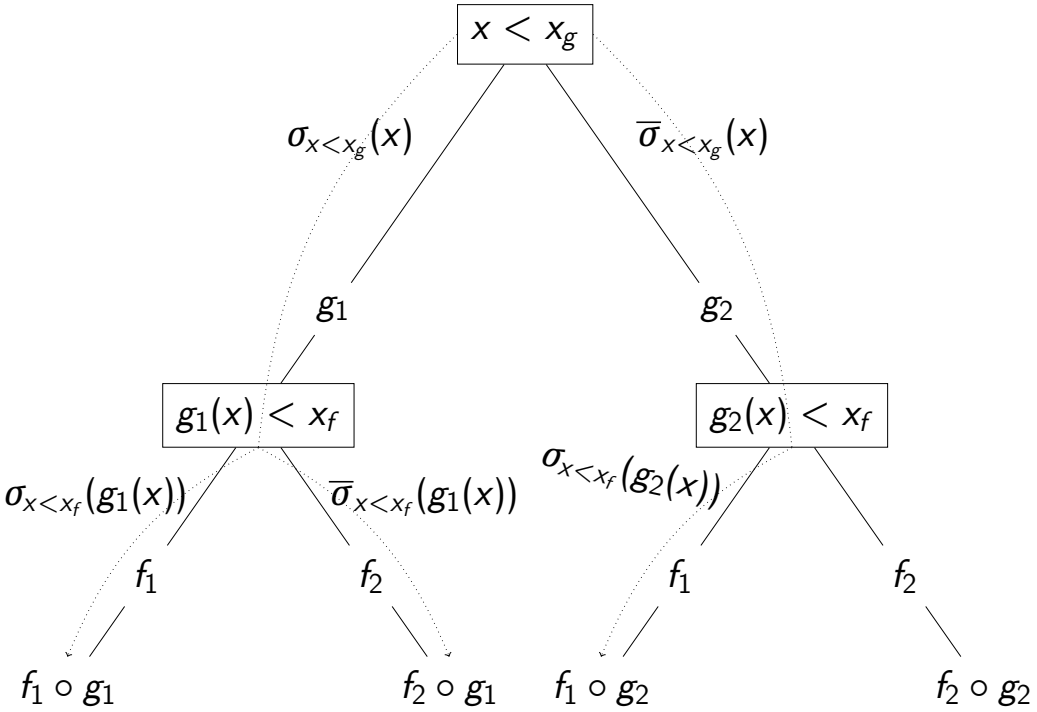

$$\widetilde{(f \circ g)} =$$
$$\sigma_{x<x_g}(x)\ \sigma_{x<x_f}(g_1(x))\ (f_1 \circ g_1)(x) + \sigma_{x<x_g}(x)\ \overline{\sigma}_{x<x_f}(g_1(x))\ (f_2 \circ g_1)(x) +$$

## Concatenation Tree $f \circ g$



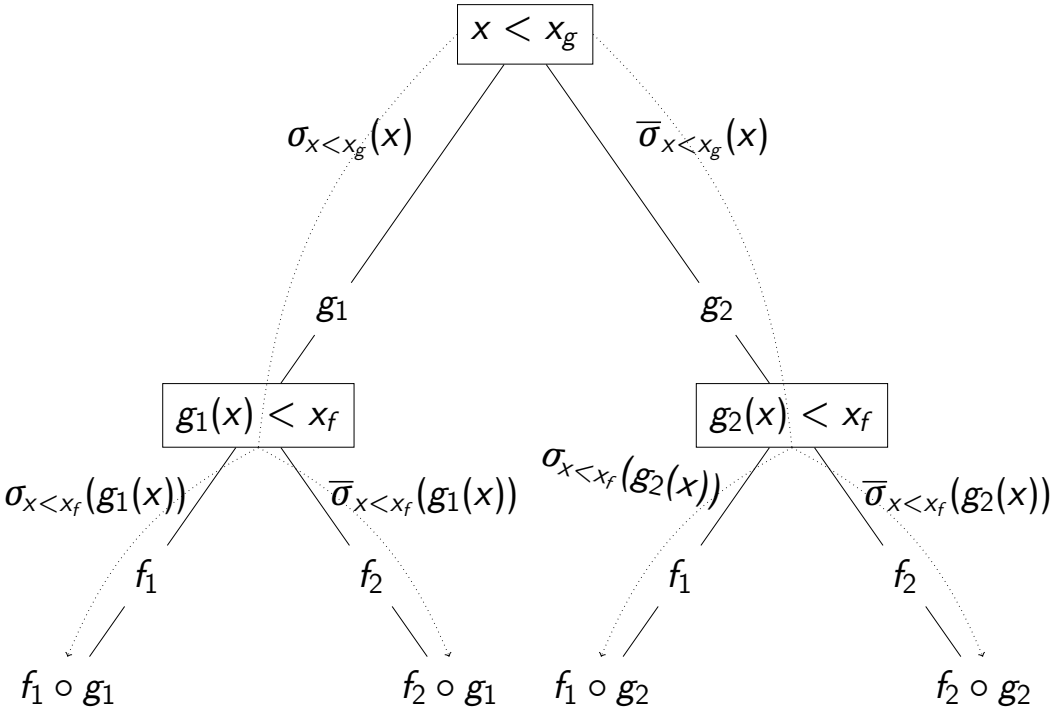

$$\widetilde{(f \circ g)} =$$
$$\sigma_{x<x_g}(x)\ \sigma_{x<x_f}(g_1(x))\ (f_1 \circ g_1)(x) + \sigma_{x<x_g}(x)\ \overline{\sigma}_{x<x_f}(g_1(x))\ (f_2 \circ g_1)(x) +$$

## Concatenation Tree $f \circ g$



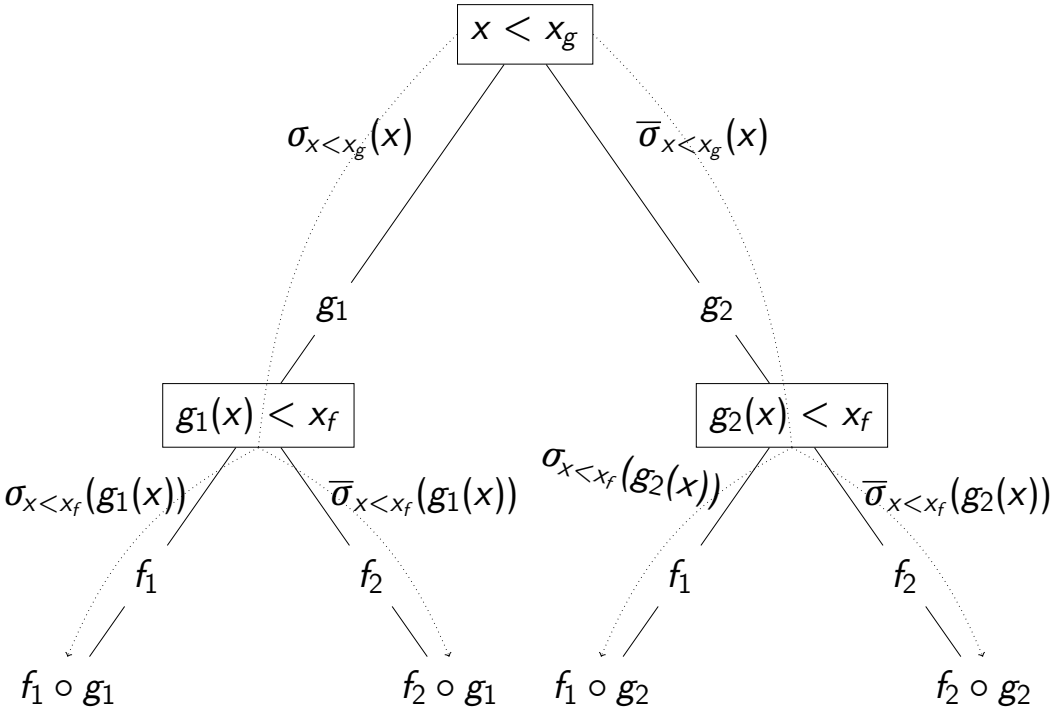

$$\widetilde{(f \circ g)} =$$
$$\sigma_{x<x_g}(x)\ \sigma_{x<x_f}(g_1(x))\ (f_1 \circ g_1)(x) + \sigma_{x<x_g}(x)\ \overline{\sigma}_{x<x_f}(g_1(x))\ (f_2 \circ g_1)(x) +$$
$$\sigma_{x<x_g}(x)\ \overline{\sigma}_{x<x_f}(g_2(x))\ (f_1 \circ g_2)(x) +$$

## Concatenation Tree $f \circ g$



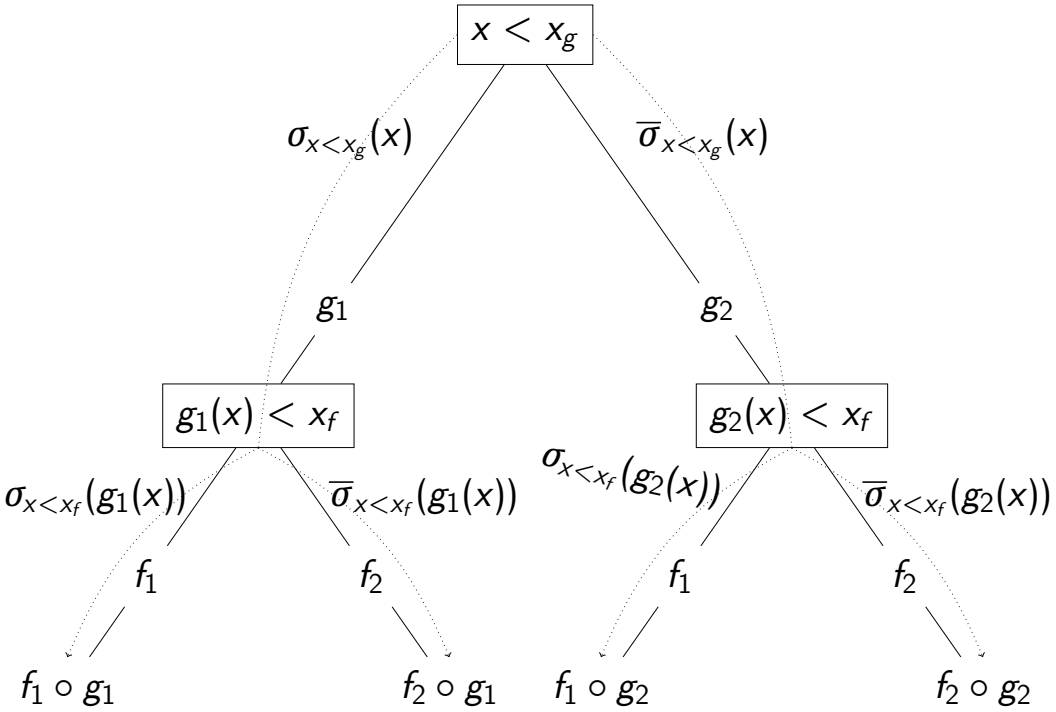

$$\widetilde{(f \circ g)} =$$
$$\sigma_{x<x_g}(x)\ \sigma_{x<x_f}(g_1(x))\ (f_1 \circ g_1)(x) + \sigma_{x<x_g}(x)\ \overline{\sigma}_{x<x_f}(g_1(x))\ (f_2 \circ g_1)(x) +$$
$$\sigma_{x<x_g}(x)\ \overline{\sigma}_{x<x_f}(g_2(x))\ (f_1 \circ g_2)(x) +$$

## Concatenation Tree $f \circ g$



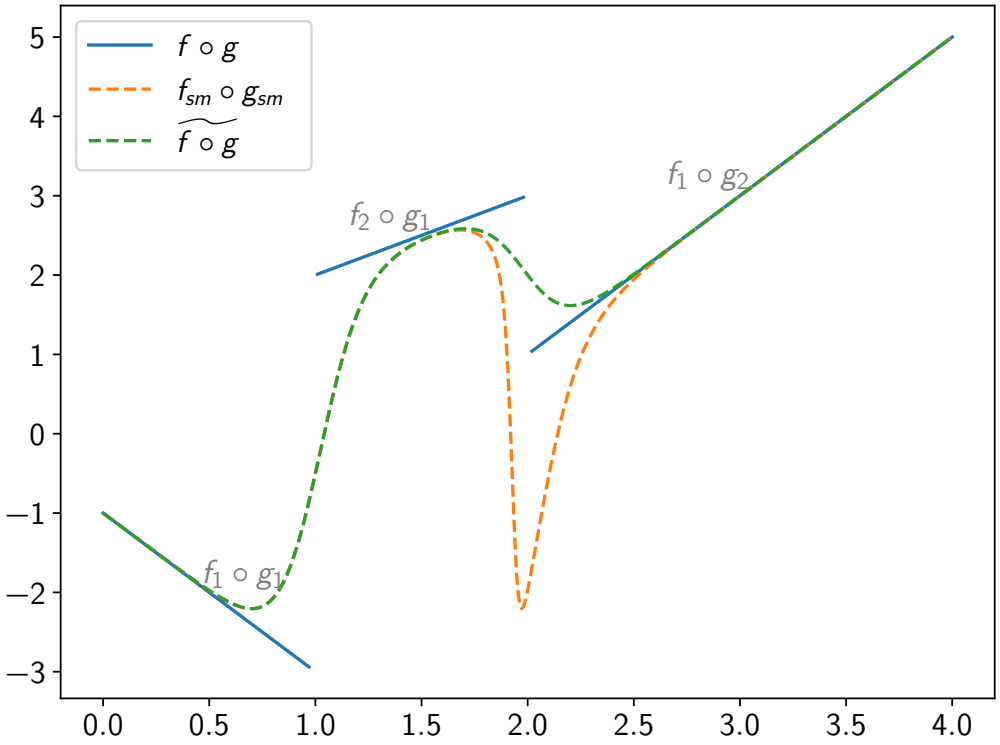

$$\widetilde{(f \circ g)} =$$
$$\sigma_{x<x_g}(x)\ \sigma_{x<x_f}(g_1(x))\ (f_1 \circ g_1)(x) + \sigma_{x<x_g}(x)\ \overline{\sigma}_{x<x_f}(g_1(x))\ (f_2 \circ g_1)(x) +$$
$$\sigma_{x<x_g}(x)\ \overline{\sigma}_{x<x_f}(g_2(x))\ (f_1 \circ g_2)(x) + \overline{\sigma}_{x<x_g}(x)\ \overline{\sigma}_{x<x_f}(g_2(x))\ (f_2 \circ g_2)(x)$$

## Concatenation Tree $f \circ g$




$$\widetilde{(f \circ g)} = \sum_{(i,j) \in \{1,2\}} p_{f_i \circ g_j}(x)\ (f_i \circ g_j)(x)$$

# Smoothing Concatenations



## Identifying contributing Branches

## Identifying contributing Branches

- Generally: Combinatorial Explosion $\widetilde{(f \circ g)} = \sum_{(i,j)\in\{1,2\}} (f_i \circ g_j)(x)\, p_{f_i \circ g_j}(x)$ evaluates all possible combination of branches

## Identifying contributing Branches

- Generally: Combinatorial Explosion $\widetilde{(f \circ g)} = \sum_{(i,j)\in\{1,2\}} (f_i \circ g_j)(x)\, p_{f_i \circ g_j}(x)$ evaluates all possible combination of branches
- Big portion of leaves have $p(x) \approx 0$! Only evaluate the relevant branches, $f_i \circ g_j$ whose contribution $p_{f_i \circ g_j} > 0$ numerically

## Identifying contributing Branches

- Generally: Combinatorial Explosion $\widetilde{(f \circ g)} = \sum_{(i,j)\in\{1,2\}} (f_i \circ g_j)(x)\, p_{f_i \circ g_j}(x)$ evaluates all possible combination of branches
- Big portion of leaves have $p(x) \approx 0$! Only evaluate the relevant branches, $f_i \circ g_j$ whose contribution $p_{f_i \circ g_j} > 0$ numerically

When does the contribution of a path to a leaf become 0?

## Identifying contributing Branches

- Generally: Combinatorial Explosion $\widetilde{(f \circ g)} = \sum_{(i,j)\in\{1,2\}} (f_i \circ g_j)(x)\, p_{f_i \circ g_j}(x)$ evaluates all possible combination of branches
- Big portion of leaves have $p(x) \approx 0$! Only evaluate the relevant branches, $f_i \circ g_j$ whose contribution $p_{f_i \circ g_j} > 0$ numerically

When does the contribution of a path to a leaf become 0?

$$p_{f_1 \circ g_1}(x) =$$

## Identifying contributing Branches

- Generally: Combinatorial Explosion $\widetilde{(f \circ g)} = \sum_{(i,j)\in\{1,2\}} (f_i \circ g_j)(x)\, p_{f_i \circ g_j}(x)$ evaluates all possible combination of branches
- Big portion of leaves have $p(x) \approx 0$! Only evaluate the relevant branches, $f_i \circ g_j$ whose contribution $p_{f_i \circ g_j} > 0$ numerically

When does the contribution of a path to a leaf become 0?

$$p_{f_1 \circ g_1}(x) = \sigma_{x<x_g}(x)\, \sigma_{x<x_f}(g_1(x)))$$

## Identifying contributing Branches

- Generally: Combinatorial Explosion $\widetilde{(f \circ g)} = \sum_{(i,j)\in\{1,2\}} (f_i \circ g_j)(x)\, p_{f_i \circ g_j}(x)$ evaluates all possible combination of branches
- Big portion of leaves have $p(x) \approx 0$! Only evaluate the relevant branches, $f_i \circ g_j$ whose contribution $p_{f_i \circ g_j} > 0$ numerically

When does the contribution of a path to a leaf become 0?

$$\begin{aligned} p_{f_1 \circ g_1}(x) &= \sigma_{x<x_g}(x)\, \sigma_{x<x_f}(g_1(x))) \\ &= 0 \end{aligned}$$

## Identifying contributing Branches

- Generally: Combinatorial Explosion $\widetilde{(f \circ g)} = \sum_{(i,j)\in\{1,2\}} (f_i \circ g_j)(x)\, p_{f_i \circ g_j}(x)$ evaluates all possible combination of branches
- Big portion of leaves have $p(x) \approx 0$! Only evaluate the relevant branches, $f_i \circ g_j$ whose contribution $p_{f_i \circ g_j} > 0$ numerically

When does the contribution of a path to a leaf become 0?

$$p_{f_1 \circ g_1}(x) = \sigma_{x<x_g}(x)\, \sigma_{x<x_f}(g_1(x)))$$
$$= 0 \text{ only if}$$

## Identifying contributing Branches

- Generally: Combinatorial Explosion $\widetilde{(f \circ g)} = \sum_{(i,j)\in\{1,2\}} (f_i \circ g_j)(x)\, p_{f_i \circ g_j}(x)$ evaluates all possible combination of branches
- Big portion of leaves have $p(x) \approx 0$! Only evaluate the relevant branches, $f_i \circ g_j$ whose contribution $p_{f_i \circ g_j} > 0$ numerically

When does the contribution of a path to a leaf become 0?

$$\begin{aligned} p_{f_1 \circ g_1}(x) &= \sigma_{x<x_g}(x)\, \sigma_{x<x_f}(g_1(x))) \\ &= 0 \text{ only if} \end{aligned}$$

$$\sigma_{x<x_g}(x) = 0 \text{ or } \sigma_{x<x_f}(g_1(x))) = 0$$

## Identifying contributing Branches

- Generally: Combinatorial Explosion $\widetilde{(f \circ g)} = \sum_{(i,j)\in\{1,2\}} (f_i \circ g_j)(x)\, p_{f_i \circ g_j}(x)$ evaluates all possible combination of branches
- Big portion of leaves have $p(x) \approx 0$! Only evaluate the relevant branches, $f_i \circ g_j$ whose contribution $p_{f_i \circ g_j} > 0$ numerically

When does the contribution of a path to a leaf become 0?

$$p_{f_1 \circ g_1}(x) = \sigma_{x<x_g}(x)\, \sigma_{x<x_f}(g_1(x)))$$
$$=0 \text{ only if}$$
$$\sigma_{x<x_g}(x) = 0 \text{ or } \sigma_{x<x_f}(g_1(x))) = 0$$

A path containing a factor smaller than $\epsilon$ (machine precision), need not be computed

## Identifying contributing Branches

- Generally: Combinatorial Explosion $\widetilde{(f \circ g)} = \sum_{(i,j)\in\{1,2\}} (f_i \circ g_j)(x)\, p_{f_i \circ g_j}(x)$ evaluates all possible combination of branches
- Big portion of leaves have $p(x) \approx 0$! Only evaluate the relevant branches, $f_i \circ g_j$ whose contribution $p_{f_i \circ g_j} > 0$ numerically

When does the contribution of a path to a leaf become 0?

$$p_{f_1 \circ g_1}(x) = \sigma_{x<x_g}(x)\, \sigma_{x<x_f}(g_1(x)))$$
$$=0 \text{ only if}$$
$$\sigma_{x<x_g}(x) < \epsilon \text{ or } \sigma_{x<x_f}(g_1(x))) < \epsilon$$

A path containing a factor smaller than $\epsilon$ (machine precision), need not be computed

# Pruning

*A path containing a local probability smaller than $\epsilon$ (machine precision), need not be computed*

# Pruning

*A path containing a local probability smaller than $\epsilon$ (machine precision), need not be computed*

**Algorithm idea:**

# Pruning

*A path containing a local probability smaller than $\epsilon$ (machine precision), need not be computed*

**Algorithm idea:**

- Executing the tree normally, calculate the leaf's probability on the way

# Pruning

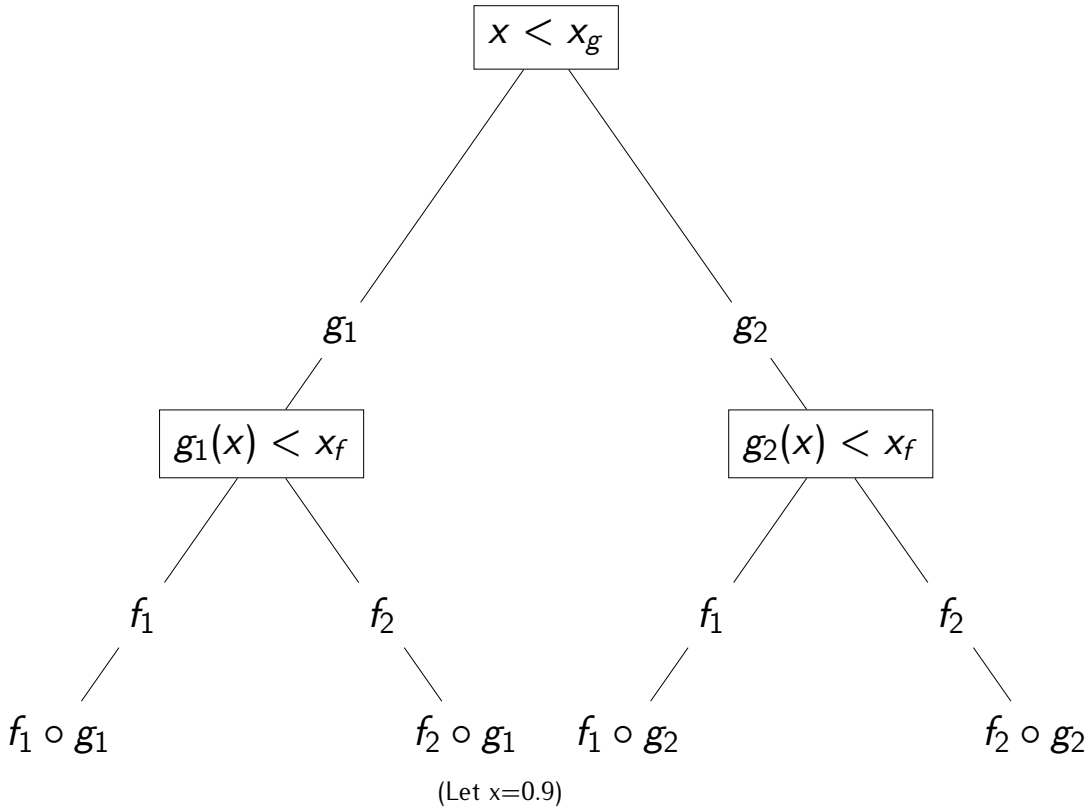
*A path containing a local probability smaller than $\epsilon$ (machine precision), need not be computed*

**Algorithm idea:**

- Executing the tree normally, calculate the leaf's probability on the way
- Additionally go down the opposite case's path, if the local contribution is not 1 numerically. (*"incomplete contribution"*)

# Conditions and Transition Zone

$f \circ g$

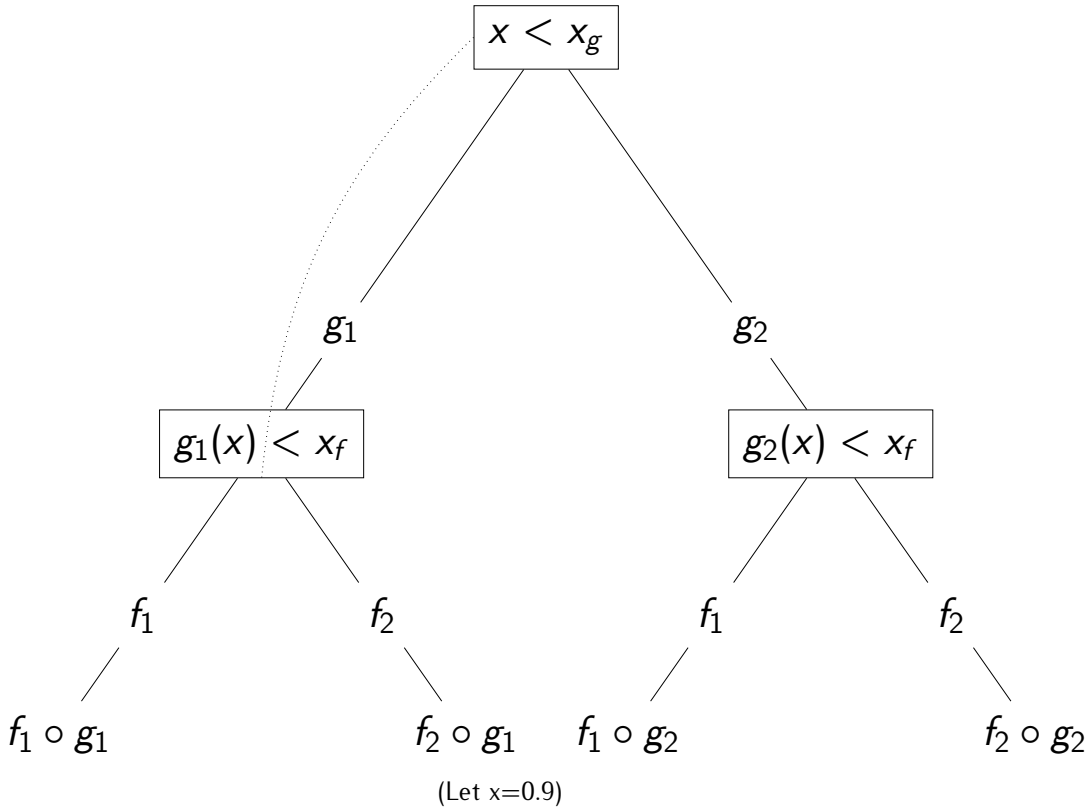



(Let x=0.9)

$\widetilde{(f \circ g)}(x) =$

# Conditions and Transition Zone

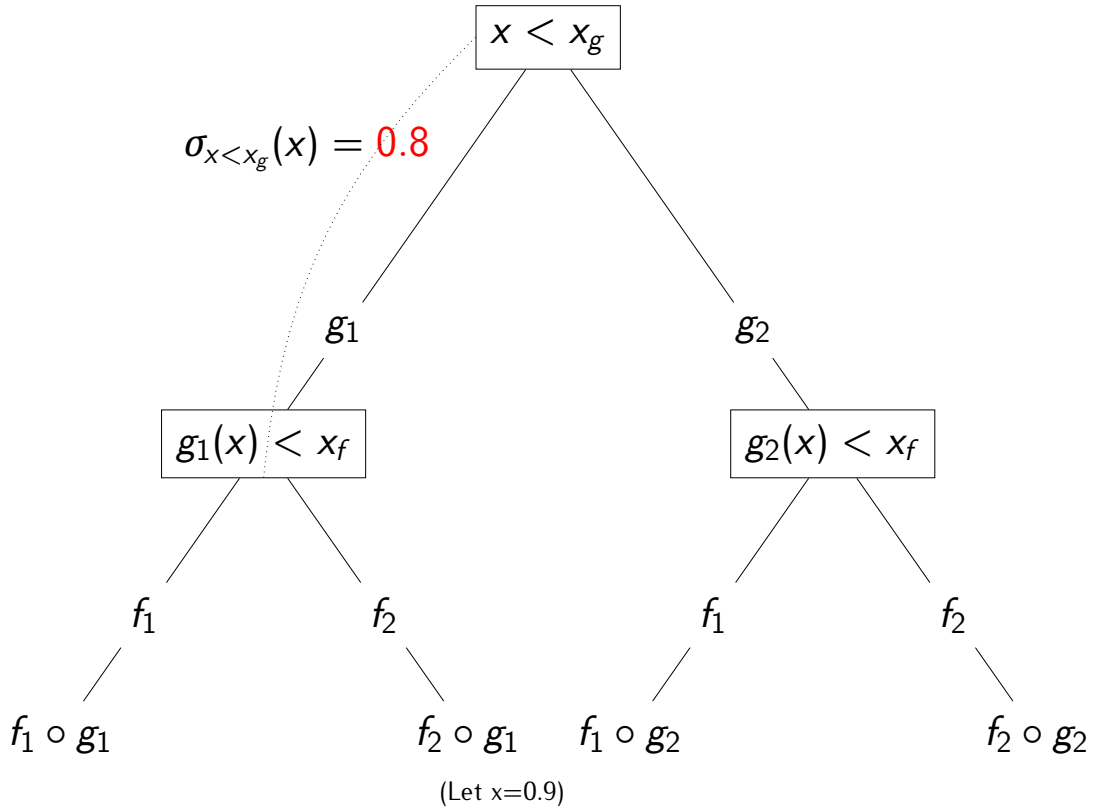
$f \circ g$




$\widetilde{(f \circ g)}(x) =$

# Conditions and Transition Zone

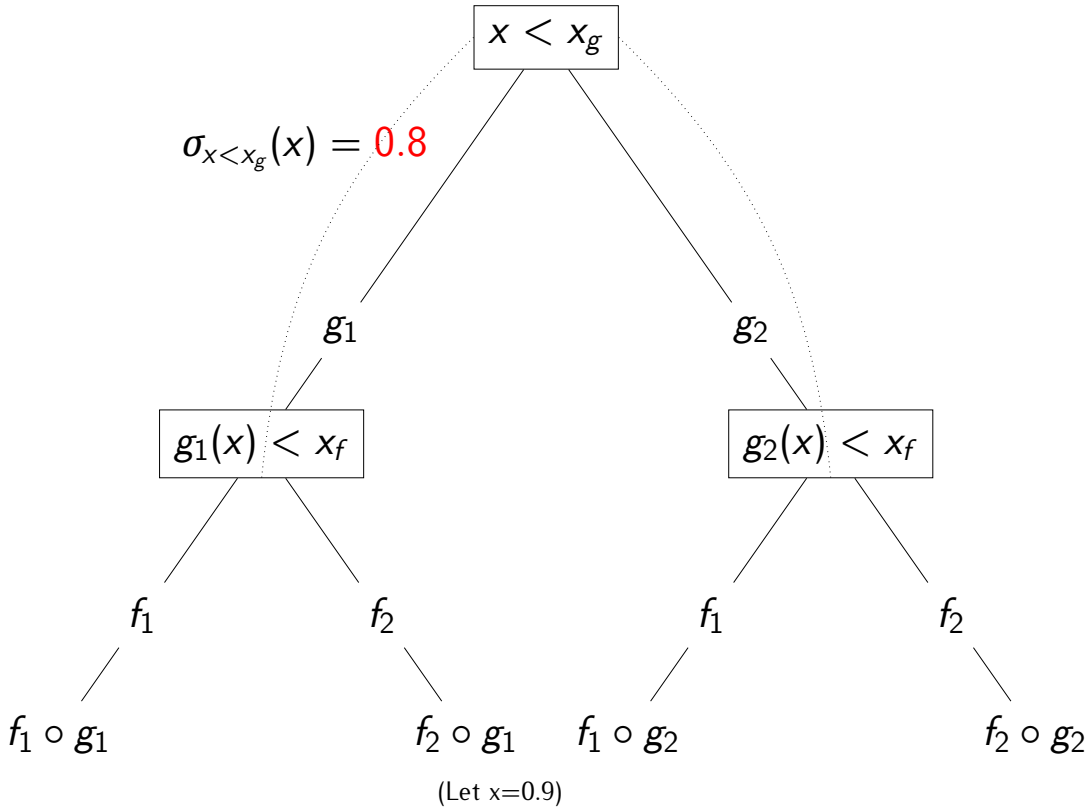
$f \circ g$




$\widetilde{(f \circ g)}(x) =$

# Conditions and Transition Zone

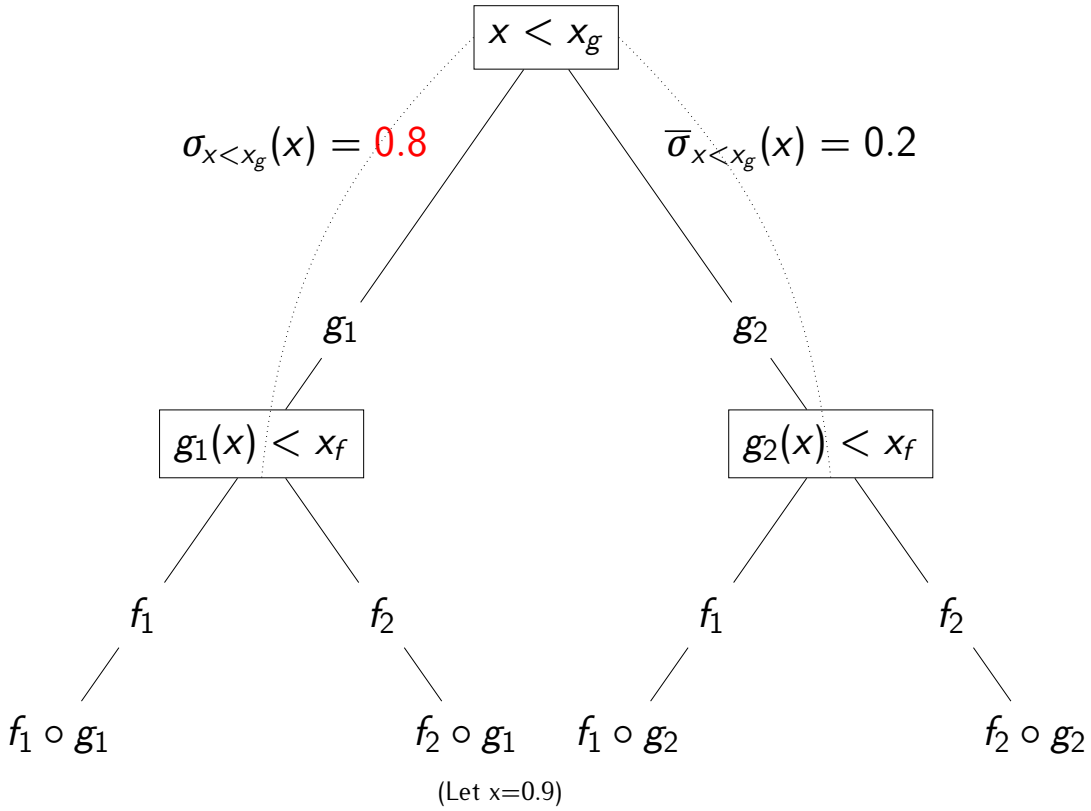
$f \circ g$




$\widetilde{(f \circ g)}(x) =$

# Conditions and Transition Zone

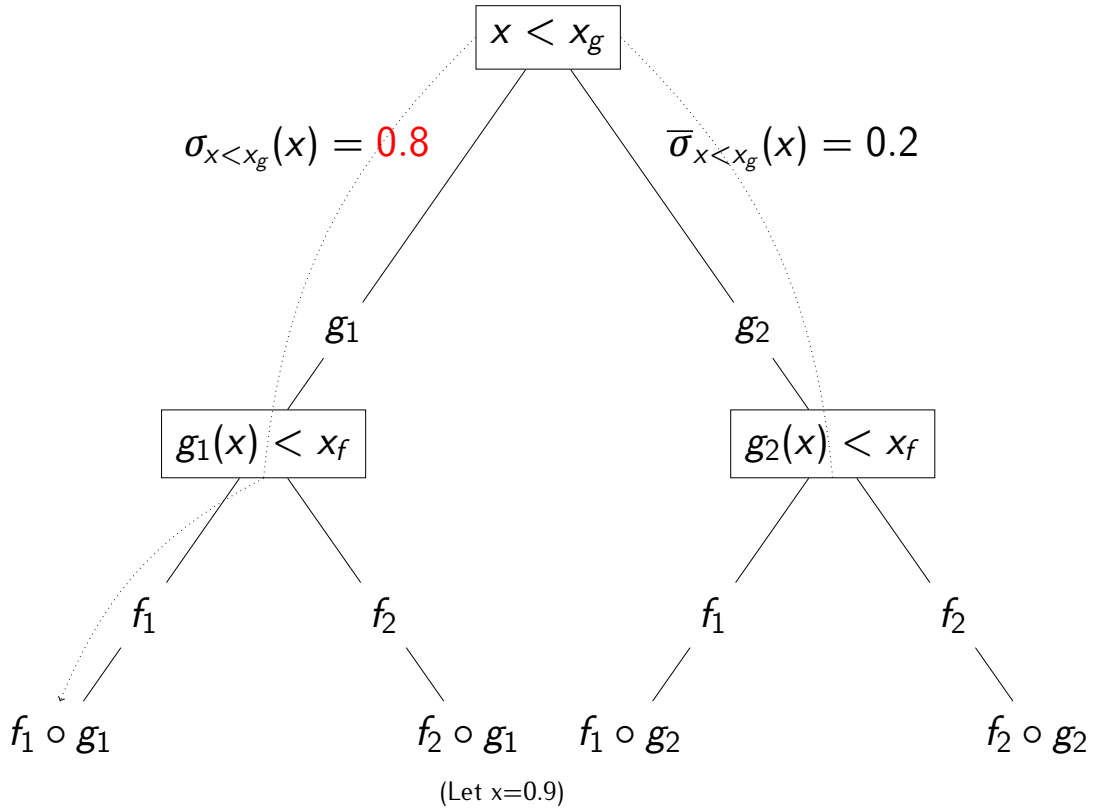
$f \circ g$




$\widetilde{(f \circ g)}(x) =$

# Conditions and Transition Zone

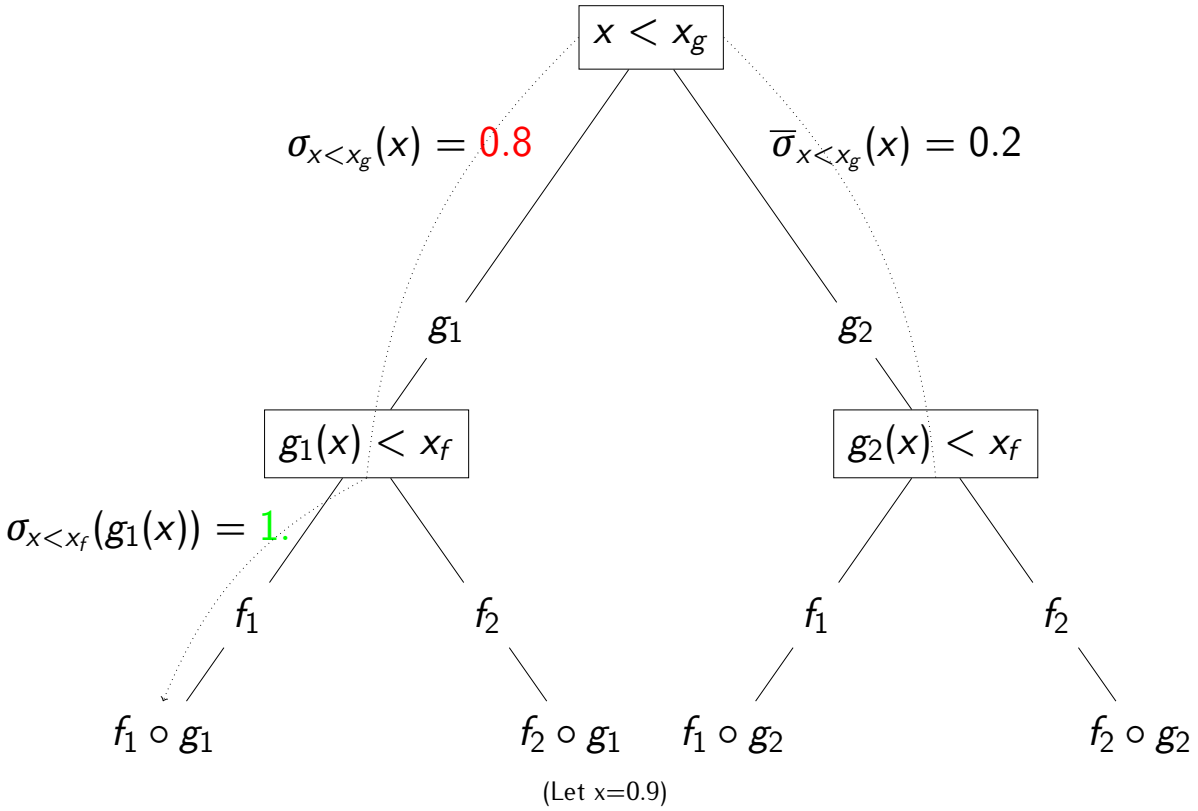
$f \circ g$




$\widetilde{(f \circ g)}(x) =$

# Conditions and Transition Zone

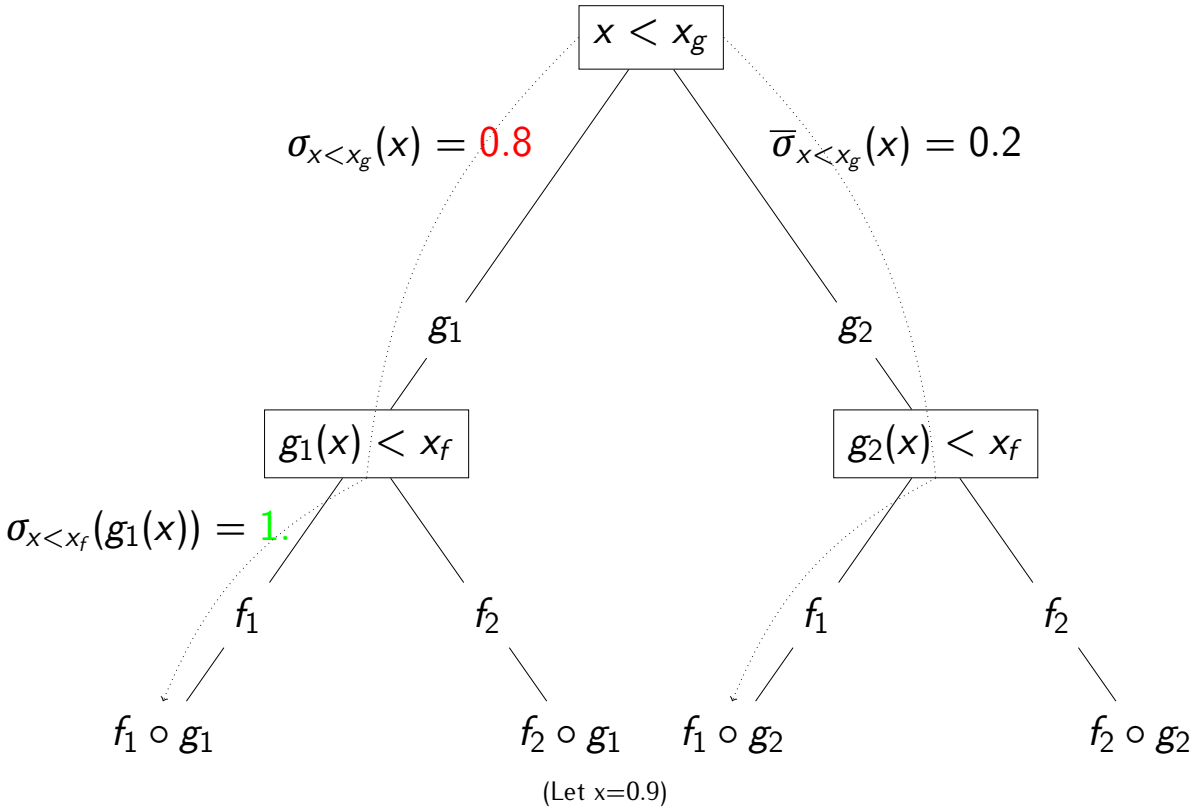
$f \circ g$




$\widetilde{(f \circ g)}(x) =$

# Conditions and Transition Zone

$f \circ g$

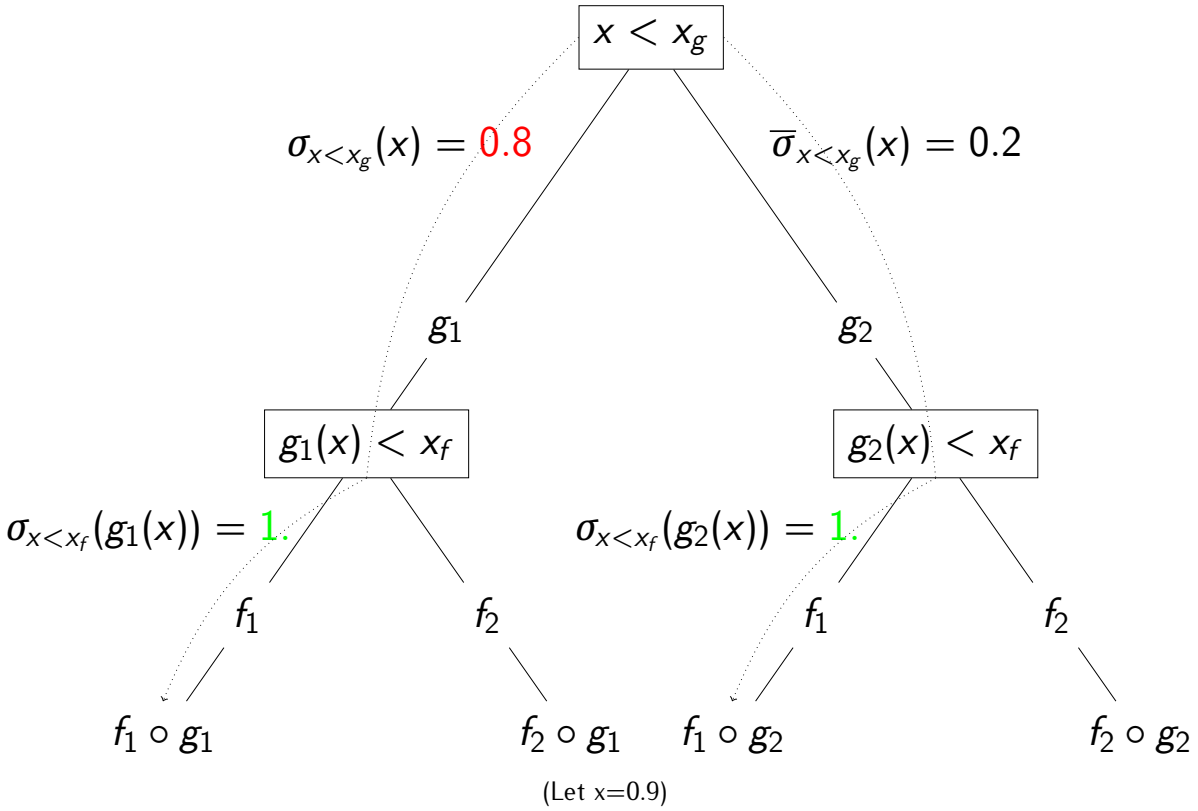



(Let x=0.9)

$\widetilde{(f \circ g)}(x) =$

# Conditions and Transition Zone

$f \circ g$

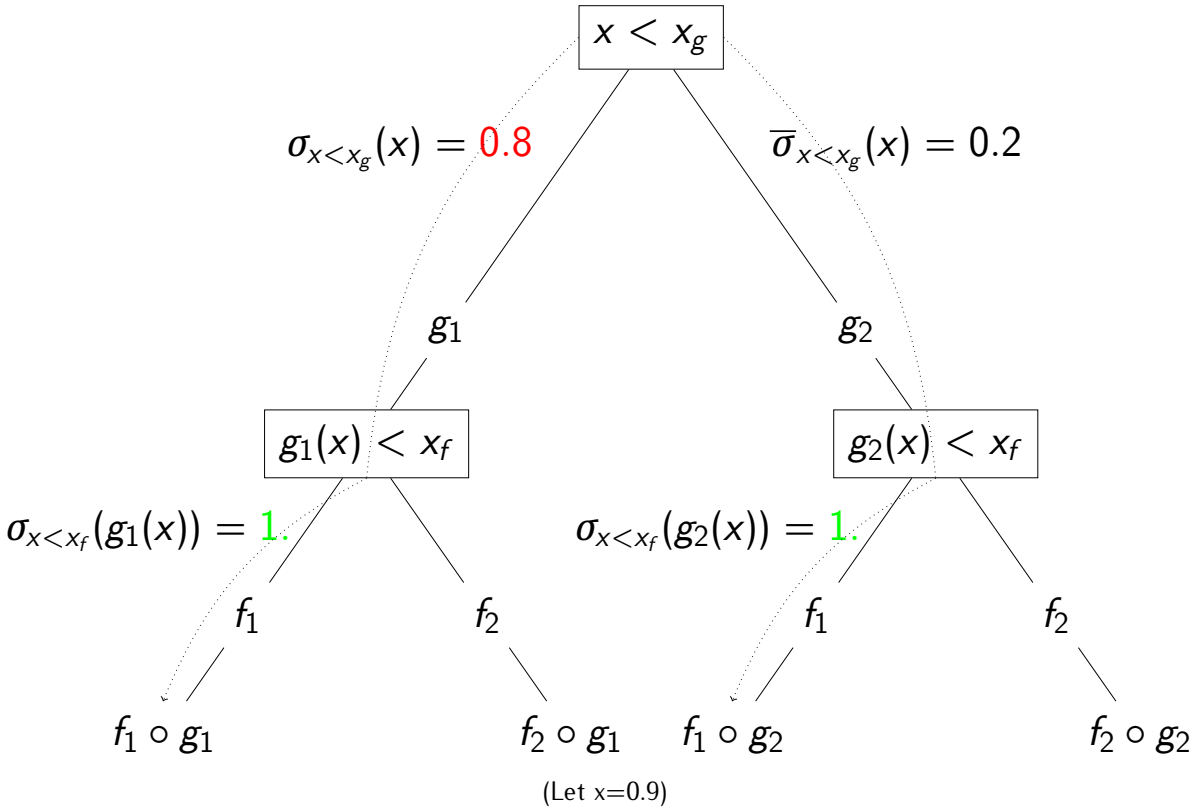



(Let x=0.9)

$\widetilde{(f \circ g)}(x) =$

# Conditions and Transition Zone

$f \circ g$




(Let x=0.9)

$$\widetilde{(f \circ g)}(x) = 0.8 \cdot 1 \cdot f_1 \circ g_1 + 0.2 \cdot 1 \cdot f_1 \circ g_2$$

# Conditions and Transition Zone

Worst-Case: $2^n$ evaluations. When does it Occur?

# Conditions and Transition Zone

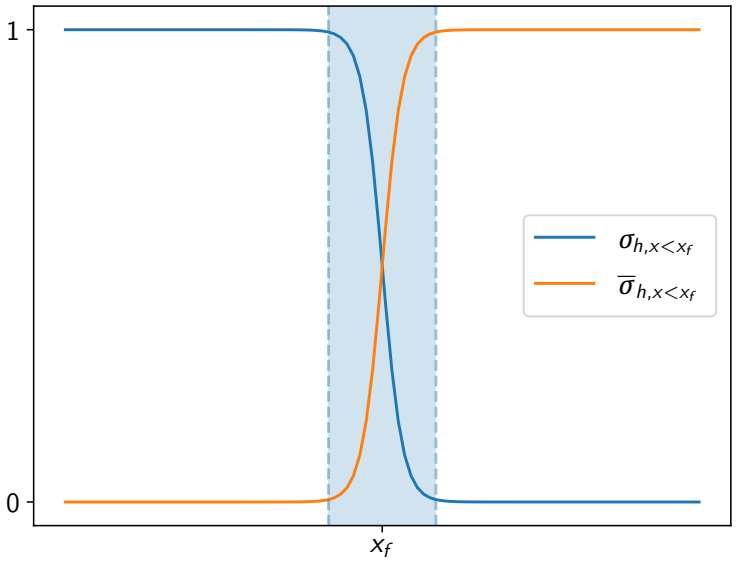
Worst-Case: $2^n$ evaluations. When does it Occur?
... it depends how *smooth* we want to be

# Conditions and Transition Zone

Worst-Case: $2^n$ evaluations. When does it Occur?
... it depends how *smooth* we want to be
i.e. how far the interpolation reaches

# Conditions and Transition Zone

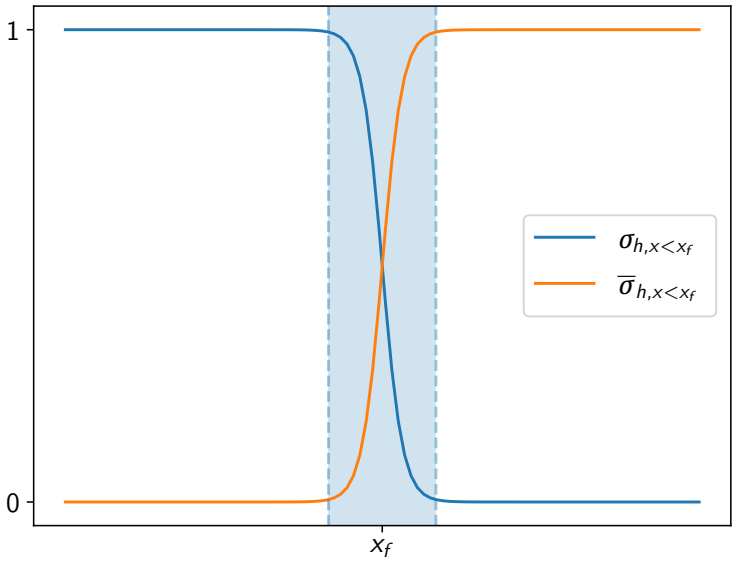
Transition-zone is where both branches are evaluated.




$$\sigma_{h,x<x_f} = \sigma((x - x_f) \cdot h)$$
$$h = 6$$

# Conditions and Transition Zone

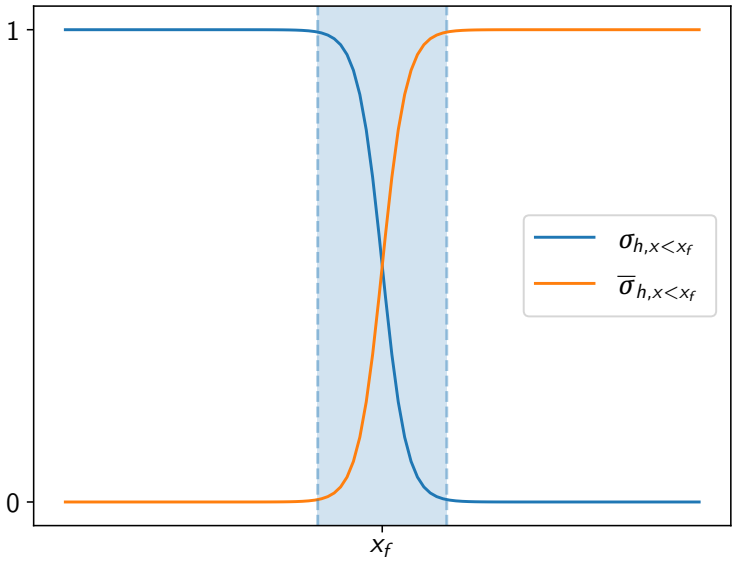
Transition-zone is where both branches are evaluated.
It changes with the sigmoid's sharpness




$$\sigma_{h,x<x_f} = \sigma((x - x_f) \cdot h)$$
$$h = 6$$

# Conditions and Transition Zone

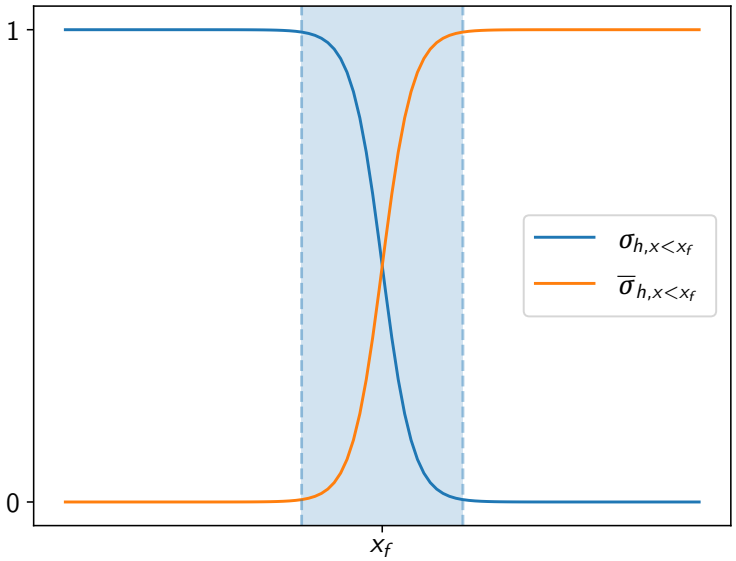
Transition-zone is where both branches are evaluated.
It changes with the sigmoid's sharpness




$$\sigma_{h,x<x_f} = \sigma((x - x_f) \cdot h)$$
$$h = 5$$

# Conditions and Transition Zone

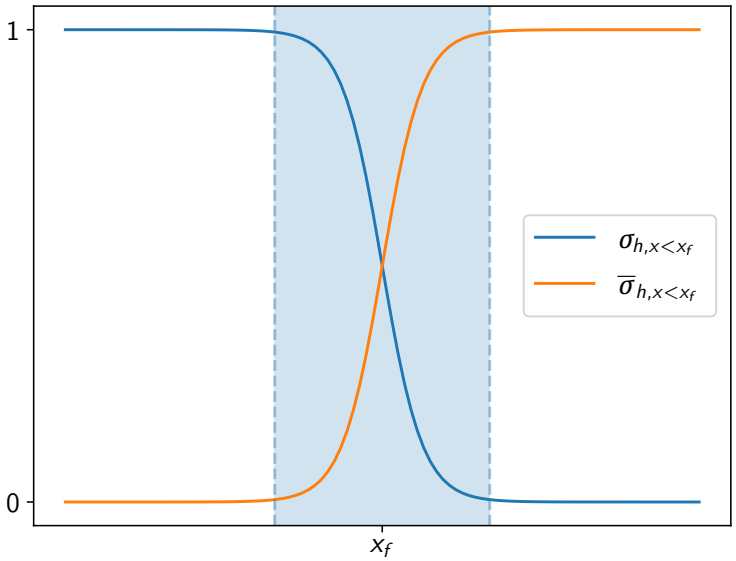
Transition-zone is where both branches are evaluated.
It changes with the sigmoid's sharpness




$$\sigma_{h,x<x_f} = \sigma((x - x_f) \cdot h)$$
$$h = 4$$

# Conditions and Transition Zone

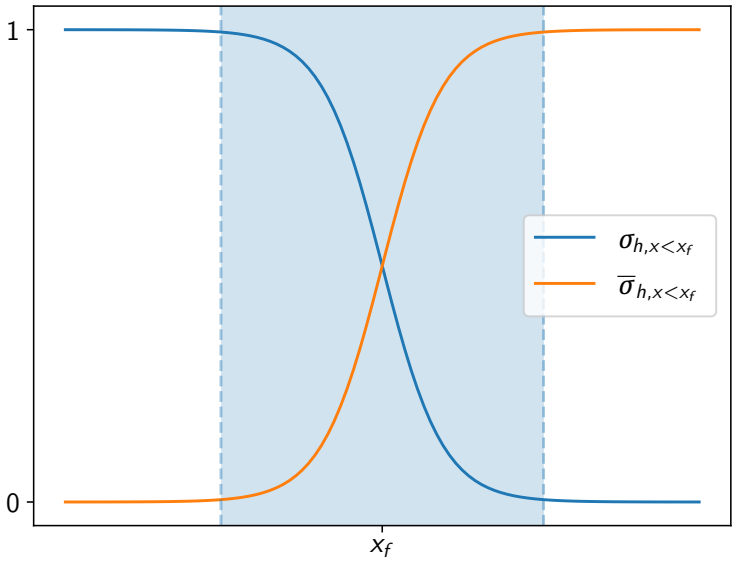
Transition-zone is where both branches are evaluated.
It changes with the sigmoid's sharpness




$$\sigma_{h,x<x_f} = \sigma((x - x_f) \cdot h)$$
$$h = 3$$

# Conditions and Transition Zone

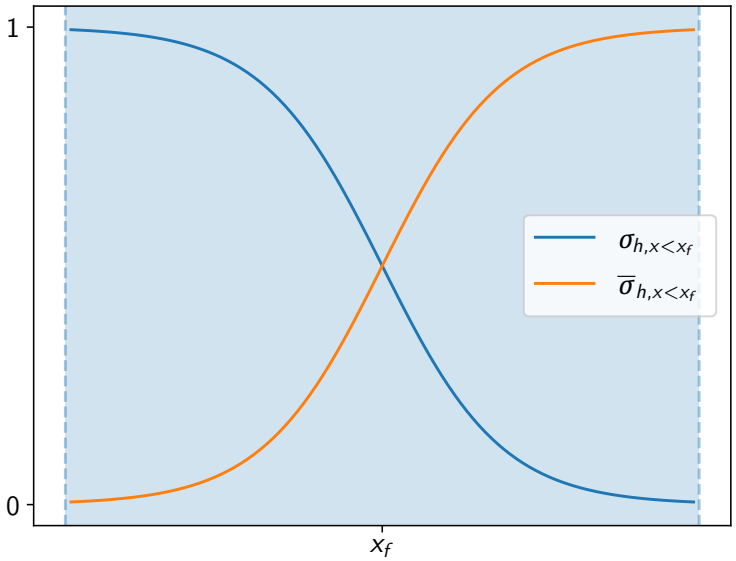
Transition-zone is where both branches are evaluated.
It changes with the sigmoid's sharpness




$$\sigma_{h,x<x_f} = \sigma((x - x_f) \cdot h)$$
$$h = 2$$

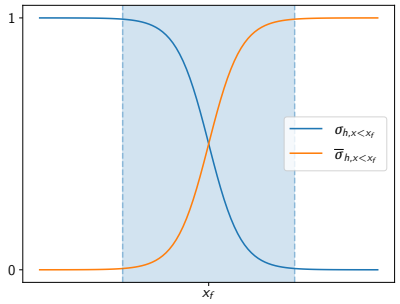
# Conditions and Transition Zone

Transition-zone is where both branches are evaluated.
It changes with the sigmoid's sharpness



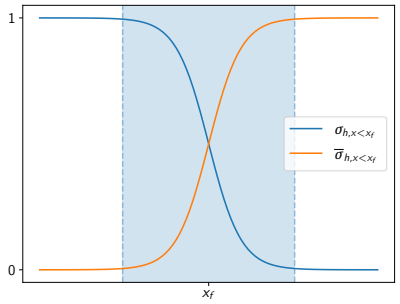

$$\sigma_{h,x<x_f} = \sigma((x - x_f) \cdot h)$$
$$h = 1$$

# Conditions and Transition Zone




In a path $\mathcal{P}$ through the nesting, we come across conditions $(k_i)_{i\in\{i_1,\dots,i_n\}}$ of nodes $\{i_1,\dots,i_n\}$.

# Conditions and Transition Zone




In a path $\mathcal{P}$ through the nesting, we come across conditions $(k_i)_{i\in\{i_1,\dots,i_n\}}$ of nodes $\{i_1,\dots,i_n\}$.

The range where $p_{k_i} < \epsilon$ is where both branches are evaluated (*transition zone*)

# Conditions and Transition Zone

Previously defined operators

# Conditions and Transition Zone

Previously defined operators

# Conditions and Transition Zone

Previously defined operators

- Logical inversion: $\overline{\sigma}_{x<x_f} = 1 - \sigma_{x<x_f}$ $\sigma_{\overline{x<x_f}} = \sigma_{x \geq x_f}$

# Conditions and Transition Zone

Previously defined operators

- Logical inversion: $\overline{\sigma}_{x<x_f} = 1 - \sigma_{x<x_f}$ $\sigma_{\overline{x<x_f}} = \sigma_{x \geq x_f}$
- Same tendencies: $\sigma_{x<x_f} = \sigma_{x \leq x_f}$.

# Conditions and Transition Zone

Previously defined operators allow conditions $k$ with $<, \leq, >, \geq$

- Logical inversion: $\overline{\sigma}_{x<x_f} = 1 - \sigma_{x<x_f}$ $\sigma_{\overline{x<x_f}} = \sigma_{x \geq x_f}$
- Same tendencies: $\sigma_{x<x_f} = \sigma_{x \leq x_f}$.

# Conditions and Transition Zone

Previously defined operators allow conditions $k$ with $<, \leq, >, \geq$

- Logical inversion: $\overline{\sigma}_{x<x_f} = 1 - \sigma_{x<x_f}$ $\sigma_{\overline{x<x_f}} = \sigma_{x \geq x_f}$
- Same tendencies: $\sigma_{x<x_f} = \sigma_{x \leq x_f}$.

These can be combined into clauses with Fuzzy Logic

# Conditions and Transition Zone

Previously defined operators allow conditions $k$ with $<, \leq, >, \geq$

- Logical inversion: $\overline{\sigma}_{x<x_f} = 1 - \sigma_{x<x_f}$ $\sigma_{\overline{x<x_f}} = \sigma_{x \geq x_f}$
- Same tendencies: $\sigma_{x<x_f} = \sigma_{x \leq x_f}$.

These can be combined into clauses with Fuzzy Logic

- $\sigma_{k_1 \wedge k_2} = \sigma_{k_1} \cdot \sigma_{k_2}$

# Conditions and Transition Zone

Previously defined operators allow conditions $k$ with $<, \leq, >, \geq$

- Logical inversion: $\overline{\sigma}_{x<x_f} = 1 - \sigma_{x<x_f}$ $\sigma_{\overline{x<x_f}} = \sigma_{x \geq x_f}$
- Same tendencies: $\sigma_{x<x_f} = \sigma_{x \leq x_f}$.

These can be combined into clauses with Fuzzy Logic

- $\sigma_{k_1 \wedge k_2} = \sigma_{k_1} \cdot \sigma_{k_2}$
- $\sigma_{k_1 \vee k_2} = \sigma_{k_1} + \sigma_{k_2} - \sigma_{k_1} \cdot \sigma_{k_2}$

# Conditions and Transition Zone

Previously defined operators allow conditions $k$ with $<, \leq, >, \geq$

- Logical inversion: $\overline{\sigma}_{x<x_f} = 1 - \sigma_{x<x_f}$ $\sigma_{\overline{x<x_f}} = \sigma_{x \geq x_f}$
- Same tendencies: $\sigma_{x<x_f} = \sigma_{x \leq x_f}$.

These can be combined into clauses with Fuzzy Logic

- $\sigma_{k_1 \wedge k_2} = \sigma_{k_1} \cdot \sigma_{k_2}$
- $\sigma_{k_1 \vee k_2} = \sigma_{k_1} + \sigma_{k_2} - \sigma_{k_1} \cdot \sigma_{k_2}$

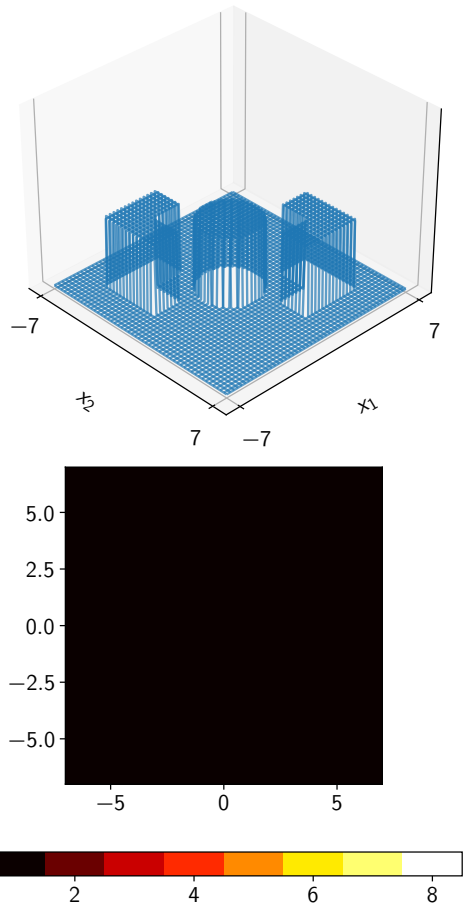
With $\overline{\ }, \vee, \wedge$, the logic is complete

# Conditions and Transition Zone

Every 'shape' represents a condition.

$2^3 = 8$ possible paths.

```
if(k_1)
  y +=0.5;
else
  y -=0.5;

if(k_2)
  y +=0.5;
else
  y -=0.5;

if(k_3)
  y +=0.5;
else
  y -=0.5;
```



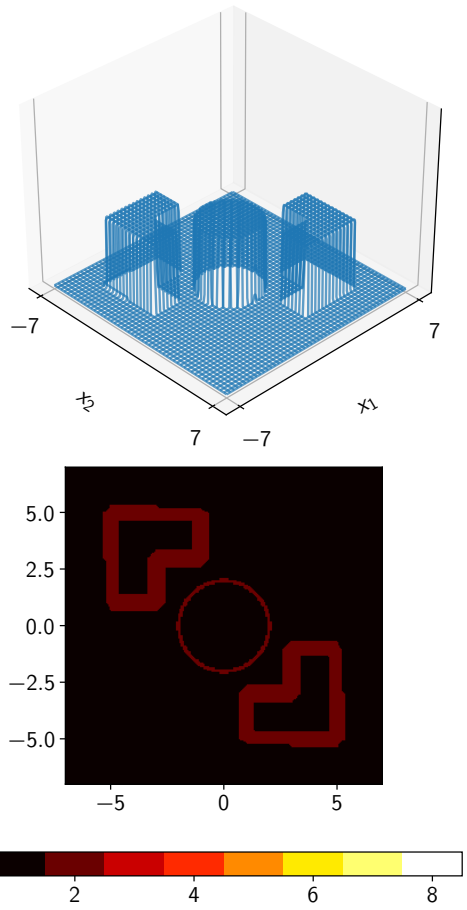

smoothing-sharpness $\infty$

# Conditions and Transition Zone

Every 'shape' represents a condition.

$2^3 = 8$ possible paths.

```
if(k_1)
  y +=0.5;
else
  y -=0.5;

if(k_2)
  y +=0.5;
else
  y -=0.5;

if(k_3)
  y +=0.5;
else
  y -=0.5;
```



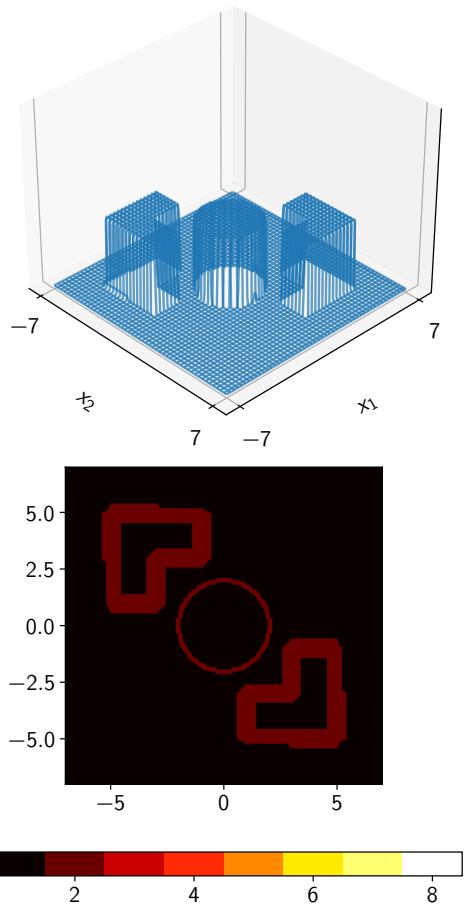

smoothing-sharpness 100

# Conditions and Transition Zone

Every 'shape' represents a condition.

$2^3 = 8$ possible paths.

```
if(k_1)
  y +=0.5;
else
  y -=0.5;

if(k_2)
  y +=0.5;
else
  y -=0.5;

if(k_3)
  y +=0.5;
else
  y -=0.5;
```



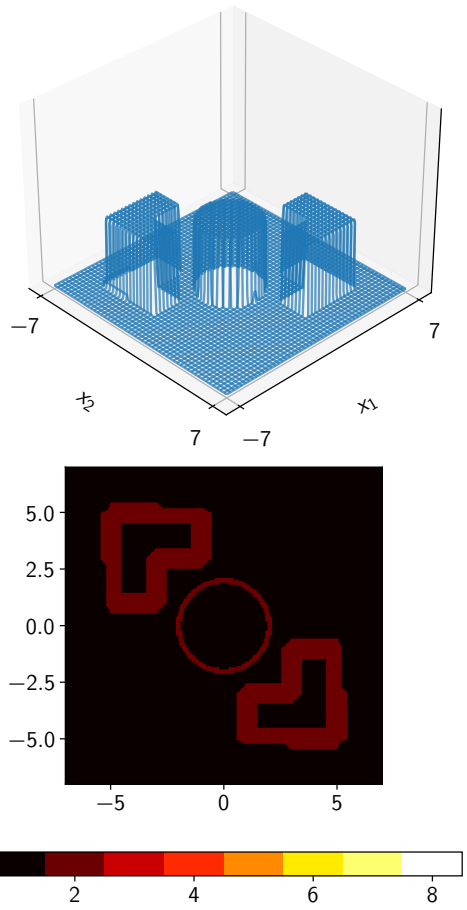

smoothing-sharpness 90

# Conditions and Transition Zone

Every 'shape' represents a condition.

$2^3 = 8$ possible paths.

```
if(k_1)
  y +=0.5;
else
  y -=0.5;

if(k_2)
  y +=0.5;
else
  y -=0.5;

if(k_3)
  y +=0.5;
else
  y -=0.5;
```



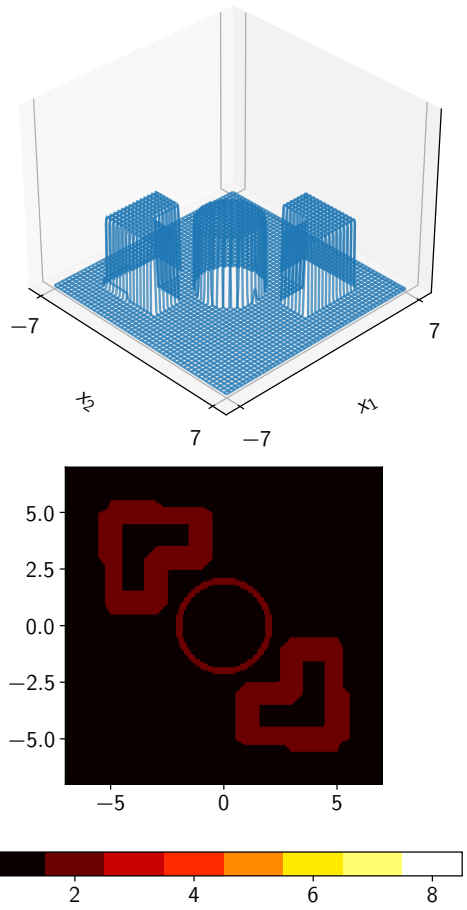

smoothing-sharpness 80

# Conditions and Transition Zone

Every 'shape' represents a condition.

$2^3 = 8$ possible paths.

```
if(k_1)
  y +=0.5;
else
  y -=0.5;

if(k_2)
  y +=0.5;
else
  y -=0.5;

if(k_3)
  y +=0.5;
else
  y -=0.5;
```



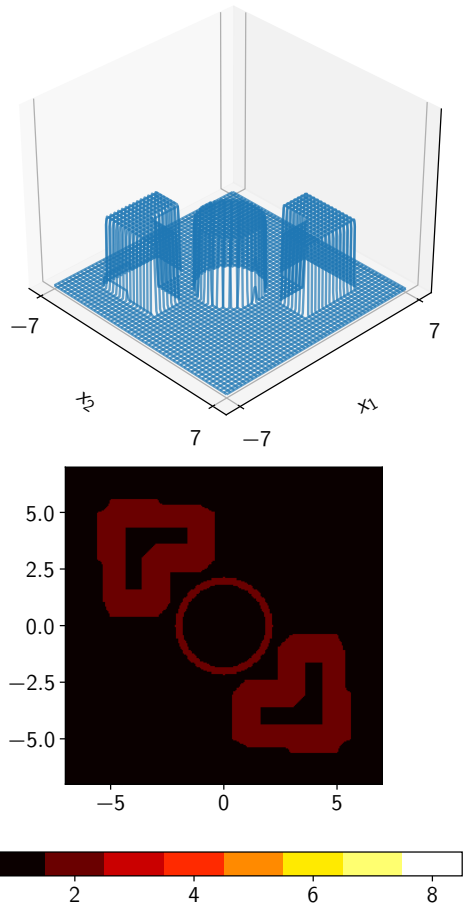

smoothing-sharpness 70

# Conditions and Transition Zone

Every 'shape' represents a condition.

$2^3 = 8$ possible paths.

```
if(k_1)
  y +=0.5;
else
  y -=0.5;

if(k_2)
  y +=0.5;
else
  y -=0.5;

if(k_3)
  y +=0.5;
else
  y -=0.5;
```



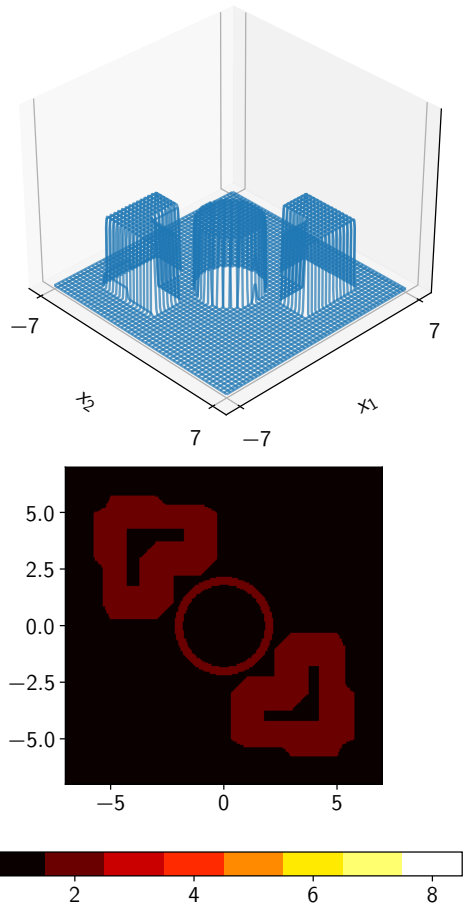

smoothing-sharpness 60

# Conditions and Transition Zone

Every 'shape' represents a condition.

$2^3 = 8$ possible paths.

```
if(k_1)
  y +=0.5;
else
  y -=0.5;

if(k_2)
  y +=0.5;
else
  y -=0.5;

if(k_3)
  y +=0.5;
else
  y -=0.5;
```



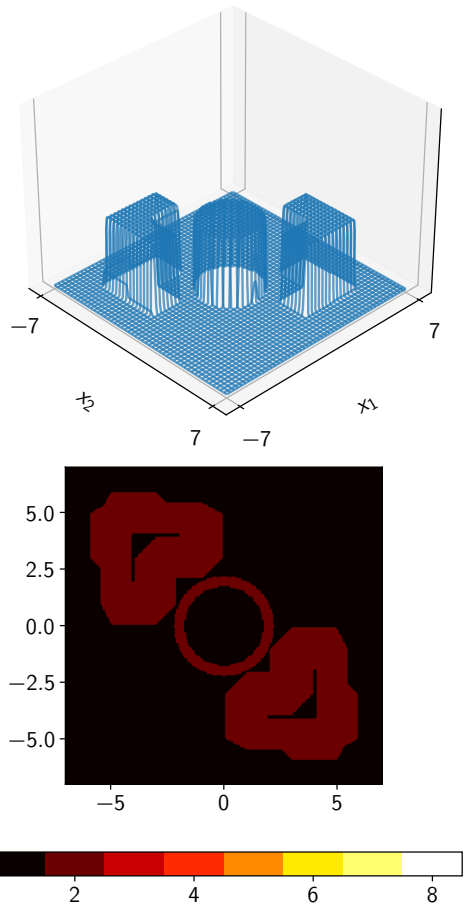

smoothing-sharpness 50

# Conditions and Transition Zone

Every 'shape' represents a condition.

$2^3 = 8$ possible paths.

```
if(k_1)
  y +=0.5;
else
  y -=0.5;

if(k_2)
  y +=0.5;
else
  y -=0.5;

if(k_3)
  y +=0.5;
else
  y -=0.5;
```



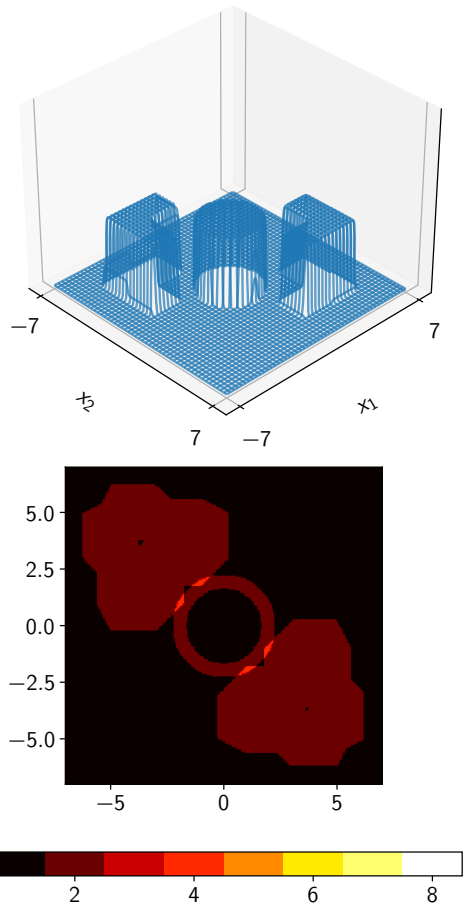

smoothing-sharpness 40

# Conditions and Transition Zone

Every 'shape' represents a condition.

$2^3 = 8$ possible paths.

```
if(k_1)
  y +=0.5;
else
  y -=0.5;

if(k_2)
  y +=0.5;
else
  y -=0.5;

if(k_3)
  y +=0.5;
else
  y -=0.5;
```



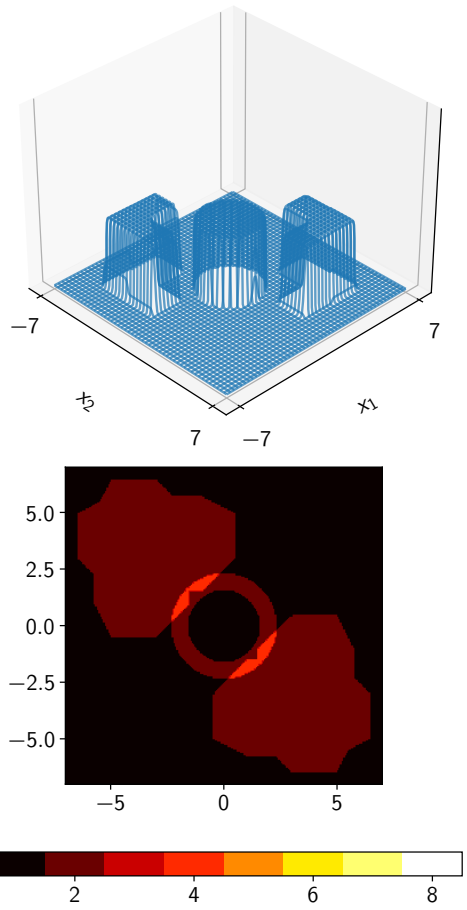

smoothing-sharpness 30

# Conditions and Transition Zone

Every 'shape' represents a condition.

$2^3 = 8$ possible paths.

```
if(k_1)
  y +=0.5;
else
  y -=0.5;

if(k_2)
  y +=0.5;
else
  y -=0.5;

if(k_3)
  y +=0.5;
else
  y -=0.5;
```



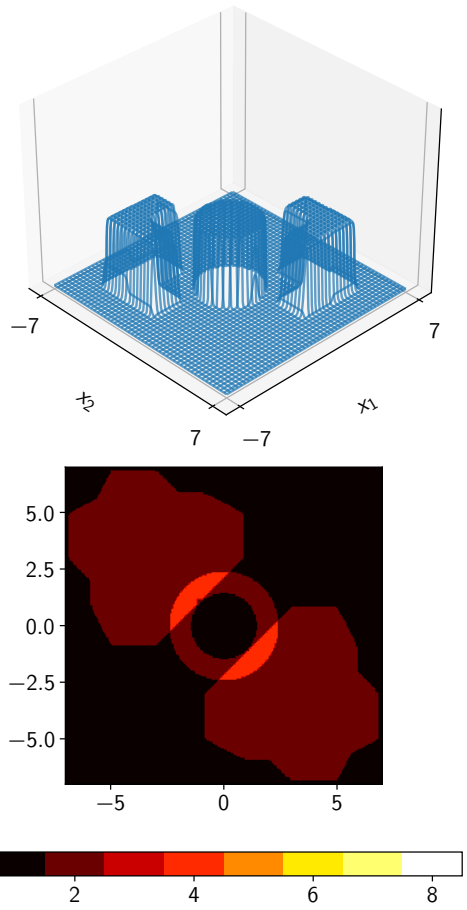

smoothing-sharpness 25

# Conditions and Transition Zone

Every 'shape' represents a condition.

$2^3 = 8$ possible paths.

```
if(k_1)
  y +=0.5;
else
  y -=0.5;

if(k_2)
  y +=0.5;
else
  y -=0.5;

if(k_3)
  y +=0.5;
else
  y -=0.5;
```



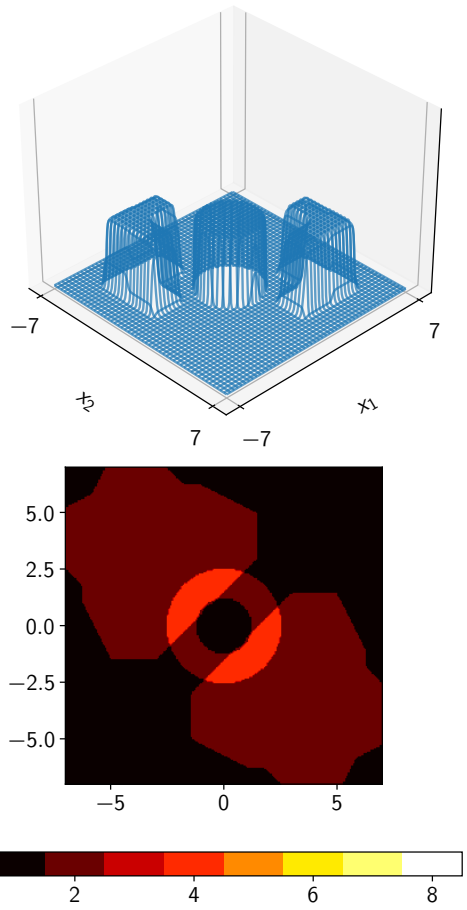

smoothing-sharpness 20

# Conditions and Transition Zone

Every 'shape' represents a condition.

$2^3 = 8$ possible paths.

```
if(k_1)
  y +=0.5;
else
  y -=0.5;

if(k_2)
  y +=0.5;
else
  y -=0.5;

if(k_3)
  y +=0.5;
else
  y -=0.5;
```



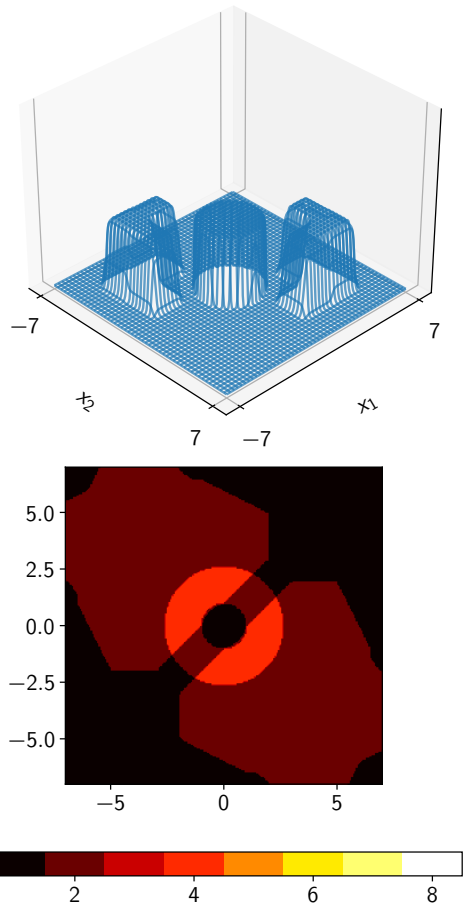

smoothing-sharpness 15

# Conditions and Transition Zone

Every 'shape' represents a condition.

$2^3 = 8$ possible paths.

```
if(k_1)
  y +=0.5;
else
  y -=0.5;

if(k_2)
  y +=0.5;
else
  y -=0.5;

if(k_3)
  y +=0.5;
else
  y -=0.5;
```



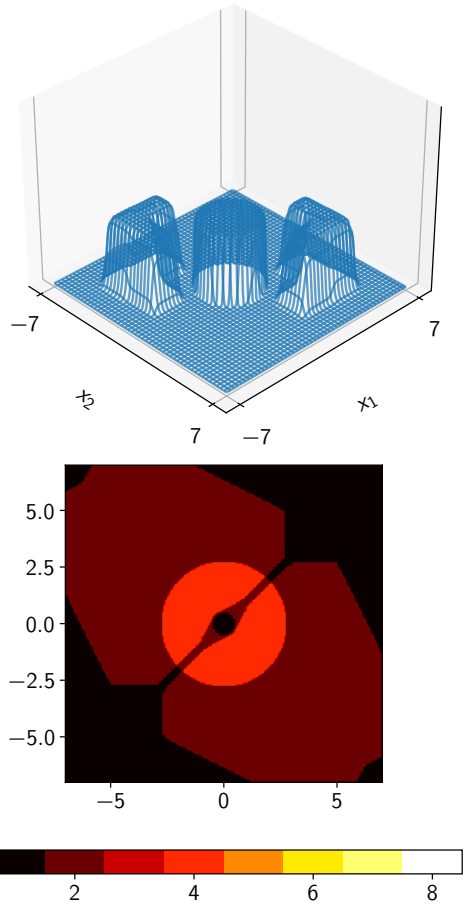

smoothing-sharpness 12.5

# Conditions and Transition Zone

Every 'shape' represents a condition.

$2^3 = 8$ possible paths.

```
if(k_1)
  y +=0.5;
else
  y -=0.5;

if(k_2)
  y +=0.5;
else
  y -=0.5;

if(k_3)
  y +=0.5;
else
  y -=0.5;
```



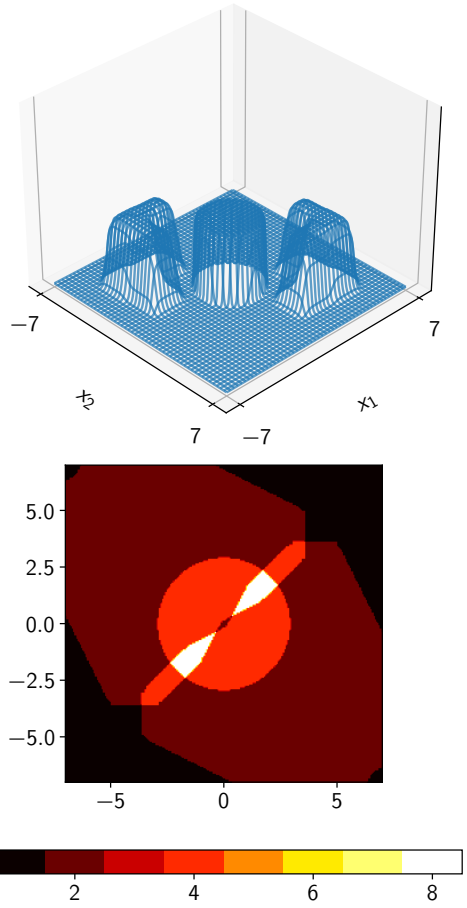

smoothing-sharpness 10

# Conditions and Transition Zone

Every 'shape' represents a condition.

$2^3 = 8$ possible paths.

```
if(k_1)
  y +=0.5;
else
  y -=0.5;

if(k_2)
  y +=0.5;
else
  y -=0.5;

if(k_3)
  y +=0.5;
else
  y -=0.5;
```



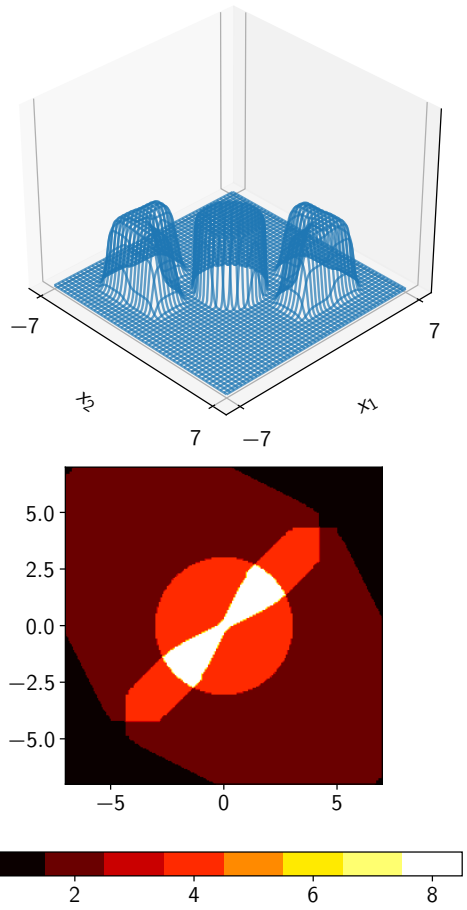

smoothing-sharpness 8

# Conditions and Transition Zone

Every 'shape' represents a condition.

$2^3 = 8$ possible paths.

```
if(k_1)
  y +=0.5;
else
  y -=0.5;

if(k_2)
  y +=0.5;
else
  y -=0.5;

if(k_3)
  y +=0.5;
else
  y -=0.5;
```



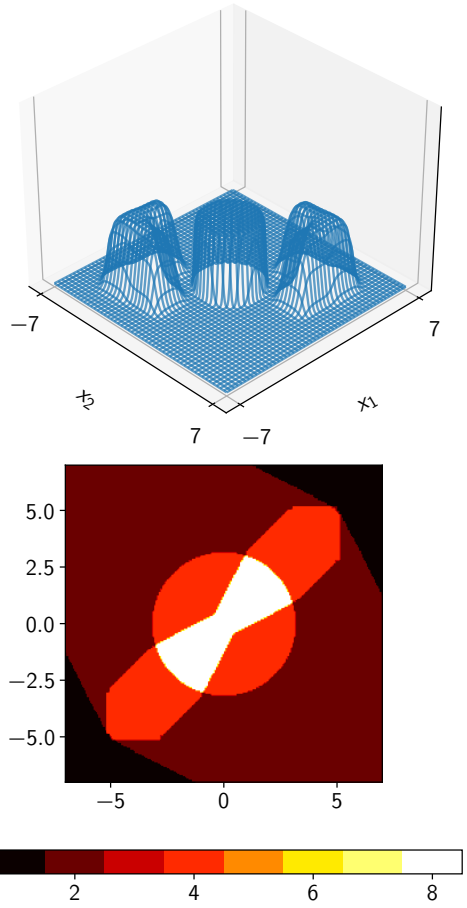

smoothing-sharpness 7

# Conditions and Transition Zone

Every 'shape' represents a condition.

$2^3 = 8$ possible paths.

```
if(k_1)
  y +=0.5;
else
  y -=0.5;

if(k_2)
  y +=0.5;
else
  y -=0.5;

if(k_3)
  y +=0.5;
else
  y -=0.5;
```



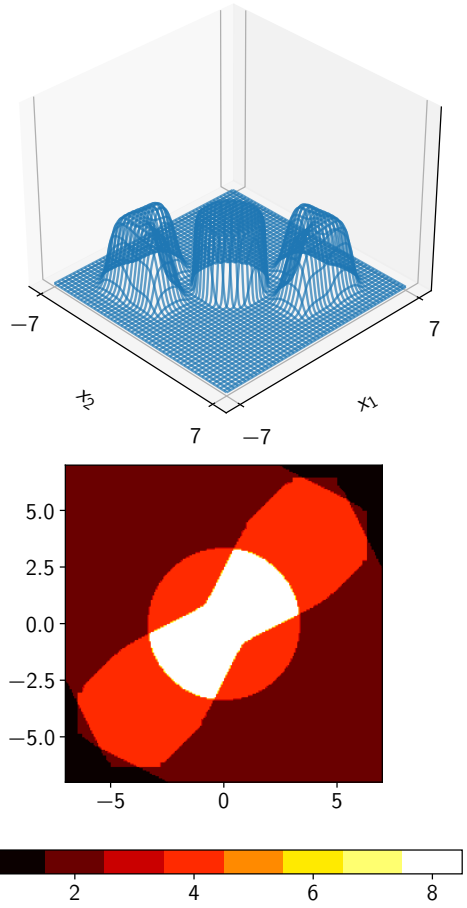

smoothing-sharpness 6

# Conditions and Transition Zone

Every 'shape' represents a condition.

$2^3 = 8$ possible paths.

```
if(k_1)
  y +=0.5;
else
  y -=0.5;

if(k_2)
  y +=0.5;
else
  y -=0.5;

if(k_3)
  y +=0.5;
else
  y -=0.5;
```



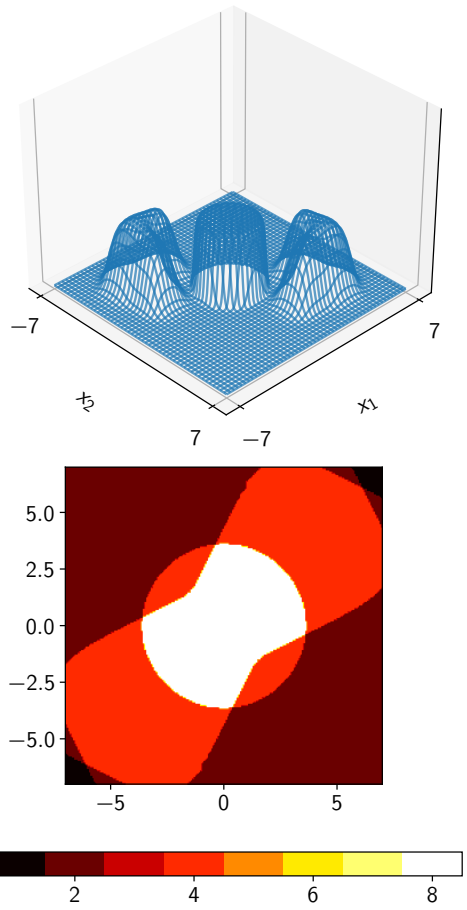

smoothing-sharpness 5

# Conditions and Transition Zone

Every 'shape' represents a condition.

$2^3 = 8$ possible paths.

```
if(k_1)
  y +=0.5;
else
  y -=0.5;

if(k_2)
  y +=0.5;
else
  y -=0.5;

if(k_3)
  y +=0.5;
else
  y -=0.5;
```



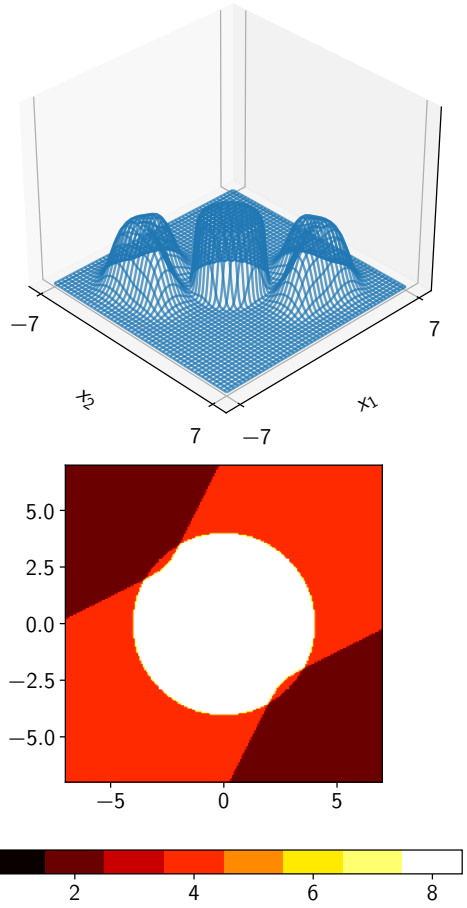

smoothing-sharpness 4

# Conditions and Transition Zone

Every 'shape' represents a condition.

$2^3 = 8$ possible paths.

```
if(k_1)
  y +=0.5;
else
  y -=0.5;

if(k_2)
  y +=0.5;
else
  y -=0.5;

if(k_3)
  y +=0.5;
else
  y -=0.5;
```



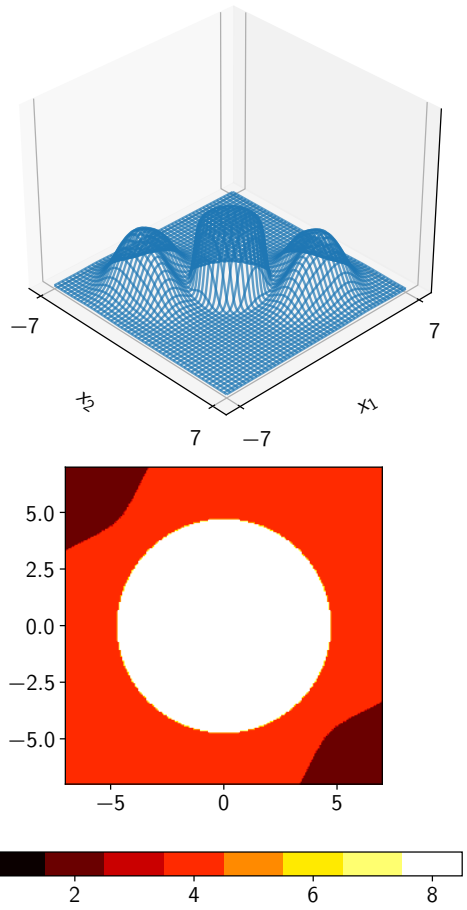

smoothing-sharpness 3

# Conditions and Transition Zone

Every 'shape' represents a condition.

$2^3 = 8$ possible paths.

```
if(k_1)
  y +=0.5;
else
  y -=0.5;

if(k_2)
  y +=0.5;
else
  y -=0.5;

if(k_3)
  y +=0.5;
else
  y -=0.5;
```



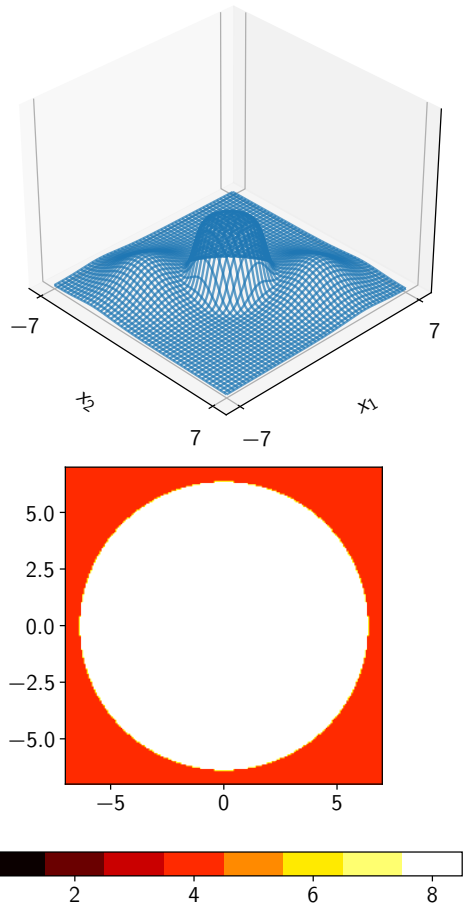

smoothing-sharpness 2

# Conditions and Transition Zone

Every 'shape' represents a condition.

$2^3 = 8$ possible paths.

```
if(k_1)
  y +=0.5;
else
  y -=0.5;

if(k_2)
  y +=0.5;
else
  y -=0.5;

if(k_3)
  y +=0.5;
else
  y -=0.5;
```



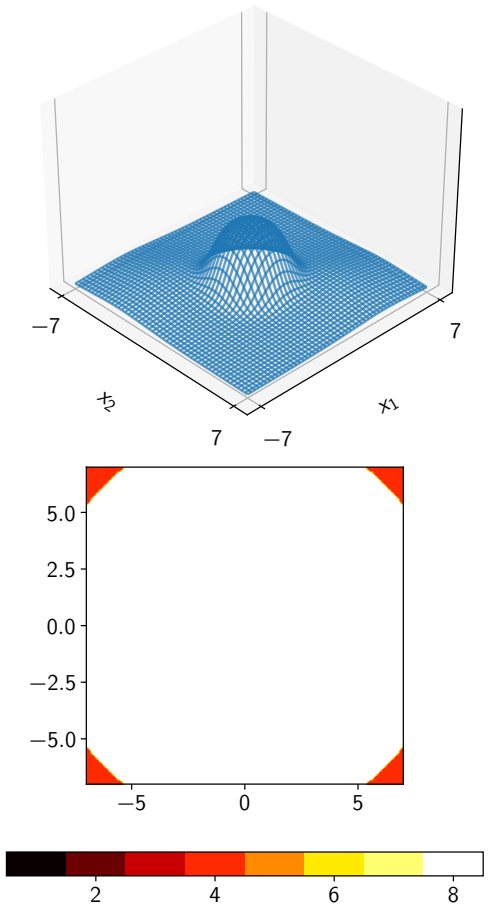

smoothing-sharpness 1

# Conditions and Transition Zone

Every 'shape' represents a condition.

$2^3 = 8$ possible paths.

```
if(k_1)
  y +=0.5;
else
  y -=0.5;

if(k_2)
  y +=0.5;
else
  y -=0.5;

if(k_3)
  y +=0.5;
else
  y -=0.5;
```

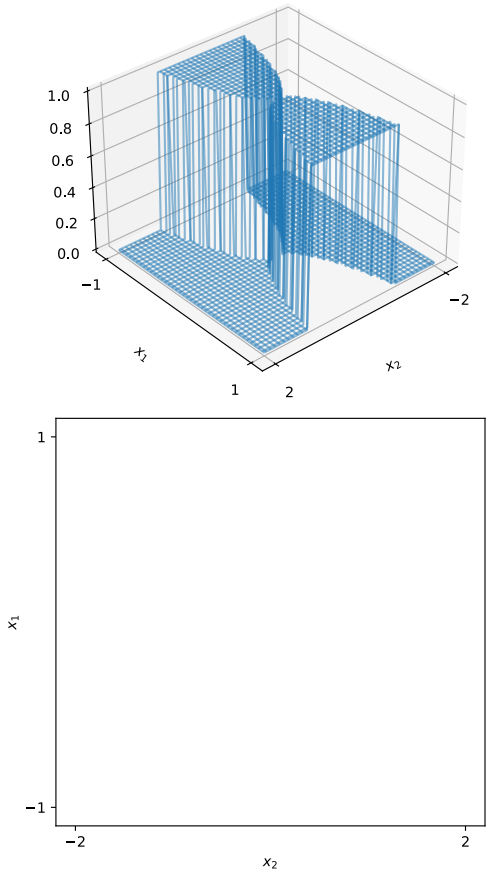



smoothing-sharpness 0.5

# Application to Code

$k =$
$x_1 > {x_2}^2 \wedge x_1 < -{x_2}^2$

```
if(k):
  y = 1;
else:
  y = 0;
```

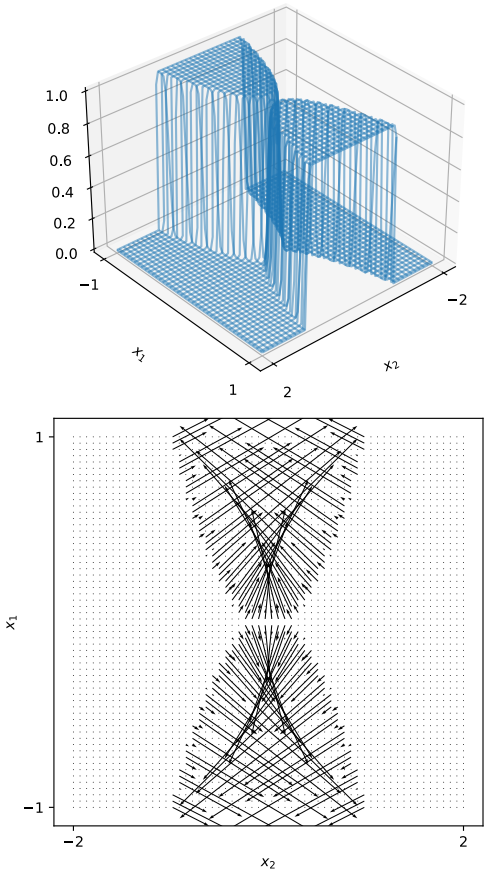



smoothing-
sharpness: $\infty$

# Application to Code

$k =$
$x_1 > {x_2}^2 \wedge x_1 < -{x_2}^2$

```
if(k):
  y = 1;
else:
  y = 0;
```

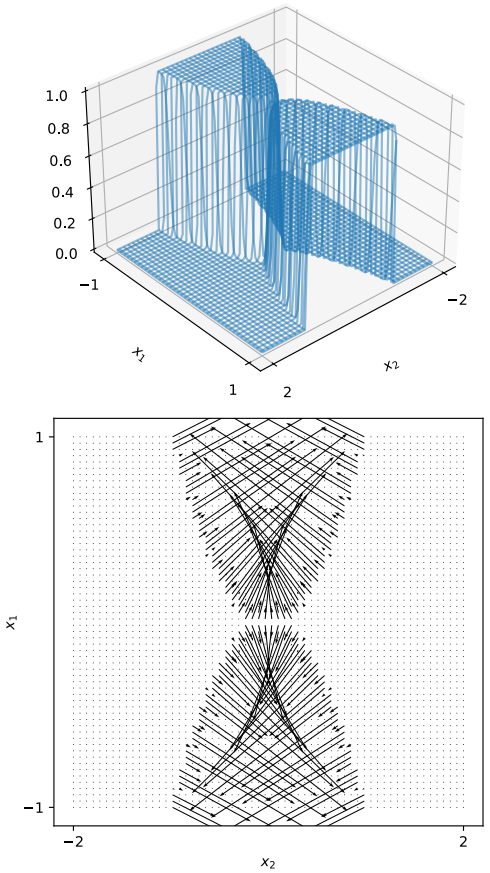



smoothing-sharpness: 100

# Application to Code

$k =$
$x_1 > {x_2}^2 \wedge x_1 < -{x_2}^2$

```
if(k):
  y = 1;
else:
  y = 0;
```

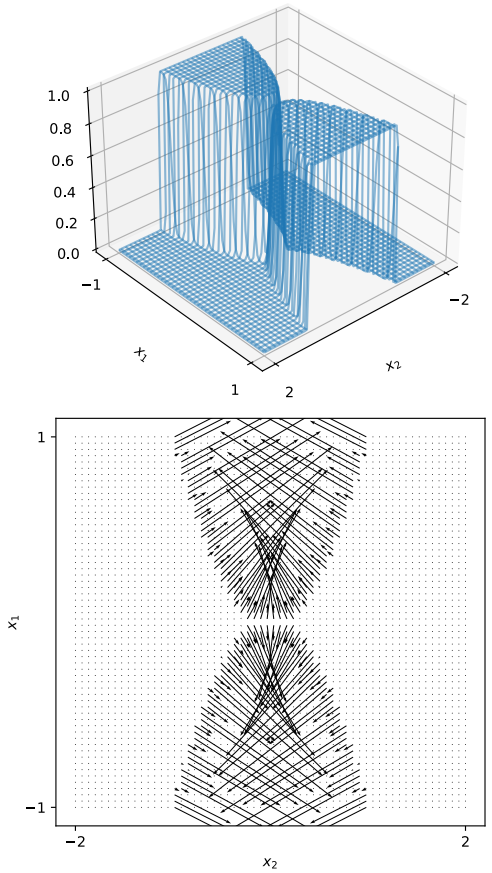



smoothing-sharpness: 90

# Application to Code

k =
$x_1 > {x_2}^2 \wedge x_1 < -{x_2}^2$

```
if(k):
  y = 1;
else:
  y = 0;
```

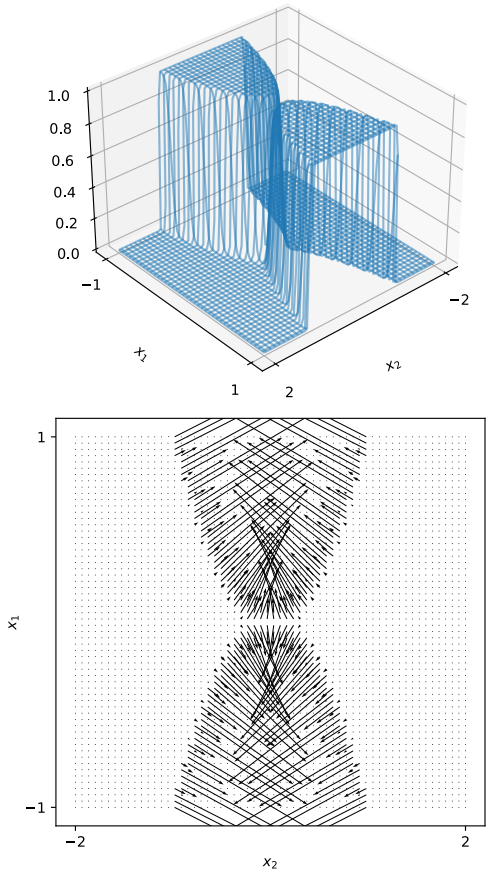



smoothing-
sharpness: 75

# Application to Code

k =
$x_1 > {x_2}^2 \wedge x_1 < -{x_2}^2$

```
if(k):
  y = 1;
else:
  y = 0;
```

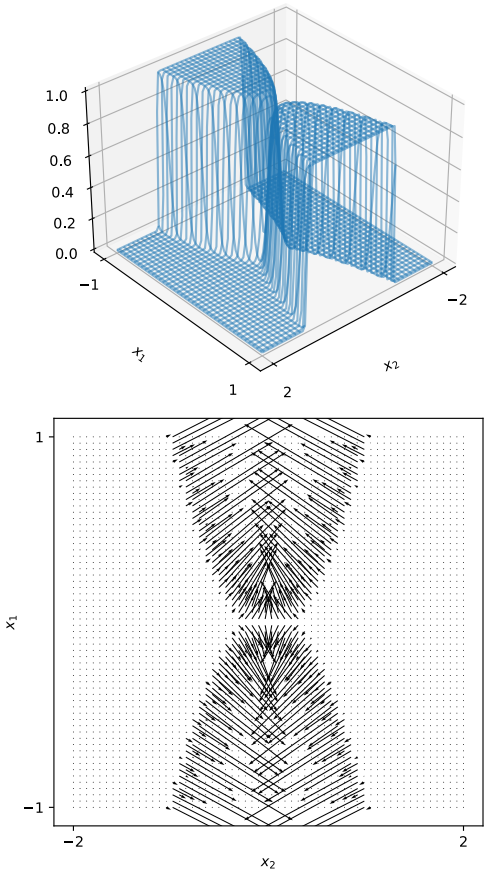



smoothing-
sharpness: 60

# Application to Code

k =
$x_1 > {x_2}^2 \wedge x_1 < -{x_2}^2$

```
if(k):
  y = 1;
else:
  y = 0;
```

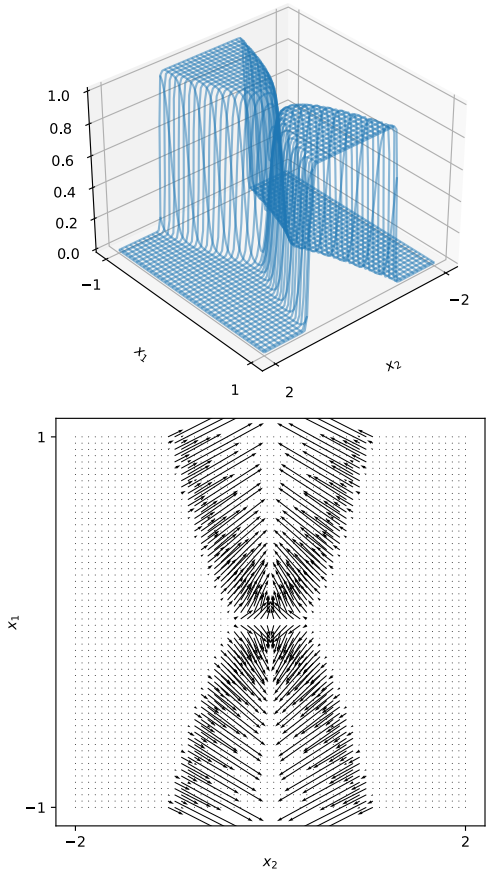



smoothing-
sharpness: 45

# Application to Code

k =
$x_1 > {x_2}^2 \wedge x_1 < -{x_2}^2$

```
if(k):
  y = 1;
else:
  y = 0;
```

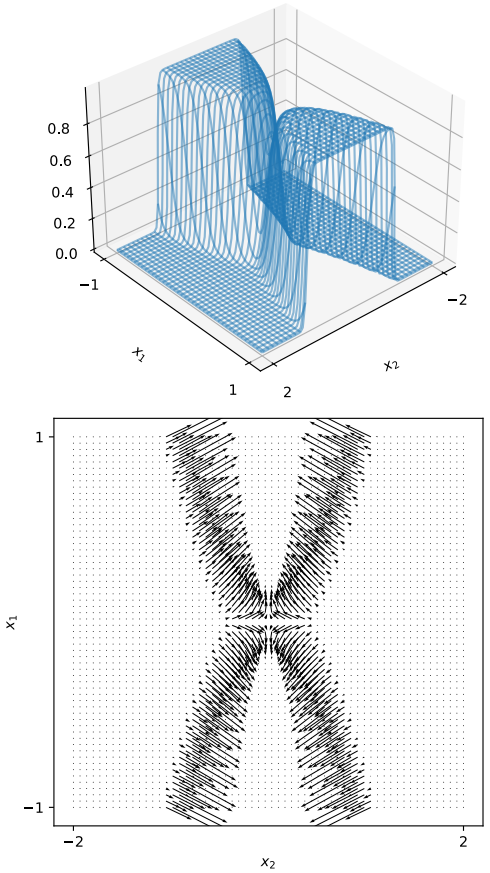



smoothing-sharpness: 30

# Application to Code

k =
$x_1 > {x_2}^2 \wedge x_1 < -{x_2}^2$

```
if(k):
  y = 1;
else:
  y = 0;
```

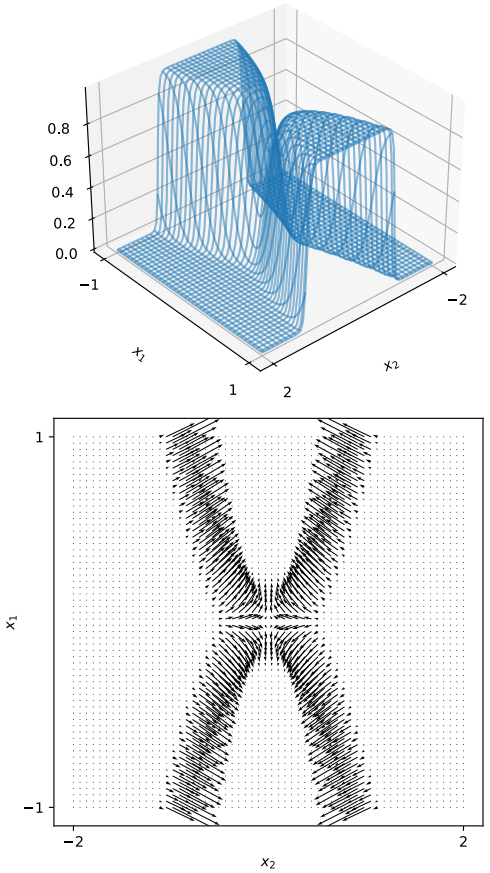



smoothing-
sharpness: 20

# Application to Code

k =
$x_1 > {x_2}^2 \wedge x_1 < -{x_2}^2$

```
if(k):
  y = 1;
else:
  y = 0;
```

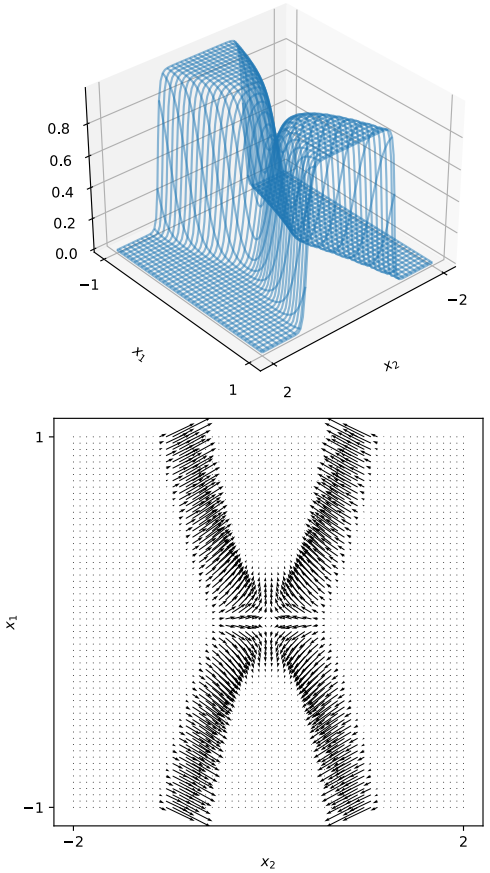



smoothing-
sharpness: 15

# Application to Code

k =
$x_1 > {x_2}^2 \wedge x_1 < -{x_2}^2$

```
if(k):
  y = 1;
else:
  y = 0;
```

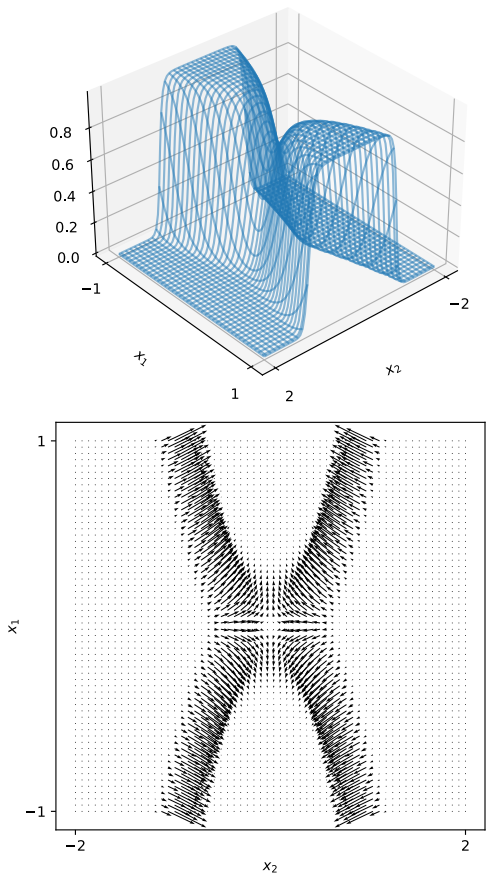



smoothing-
sharpness: 12.5

# Application to Code

$k =$
$x_1 > {x_2}^2 \wedge x_1 < -{x_2}^2$

```
if(k):
  y = 1;
else:
  y = 0;
```

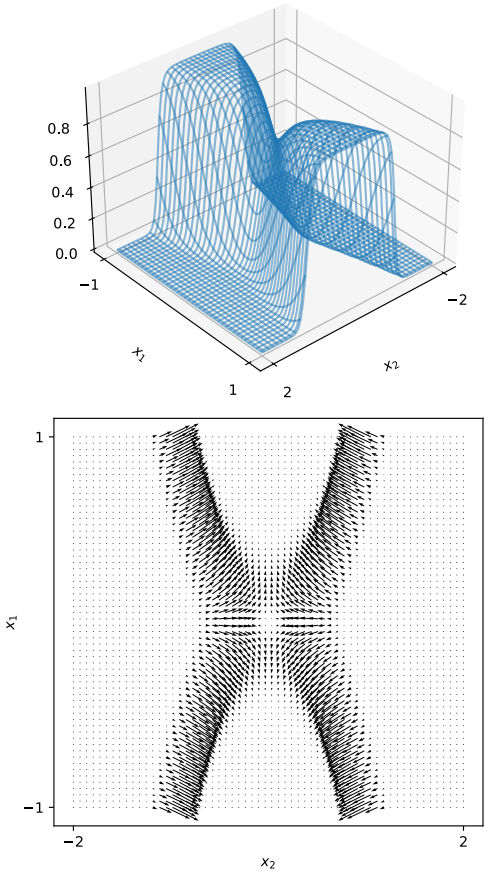



smoothing-sharpness: 10

# Application to Code

k =
$x_1 > {x_2}^2 \wedge x_1 < -{x_2}^2$

```
if(k):
  y = 1;
else:
  y = 0;
```

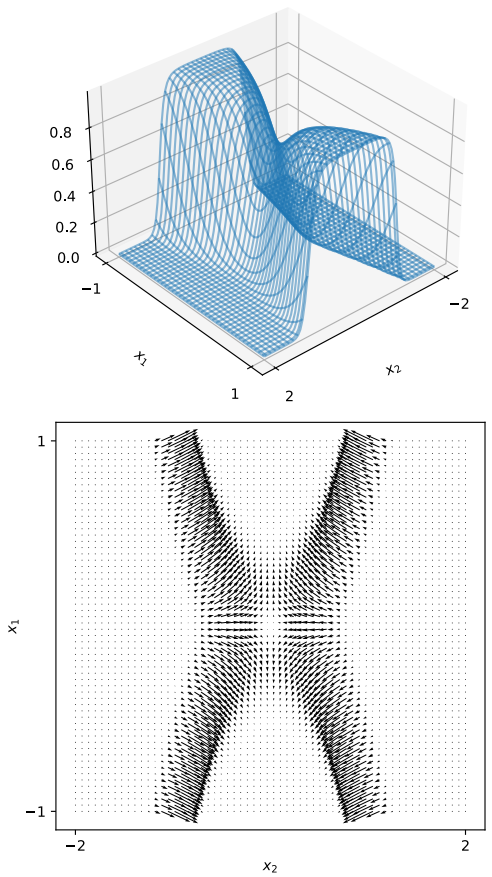



smoothing-
sharpness: 8

# Application to Code

$k =$
$x_1 > {x_2}^2 \wedge x_1 < -{x_2}^2$

```
if(k):
  y = 1;
else:
  y = 0;
```

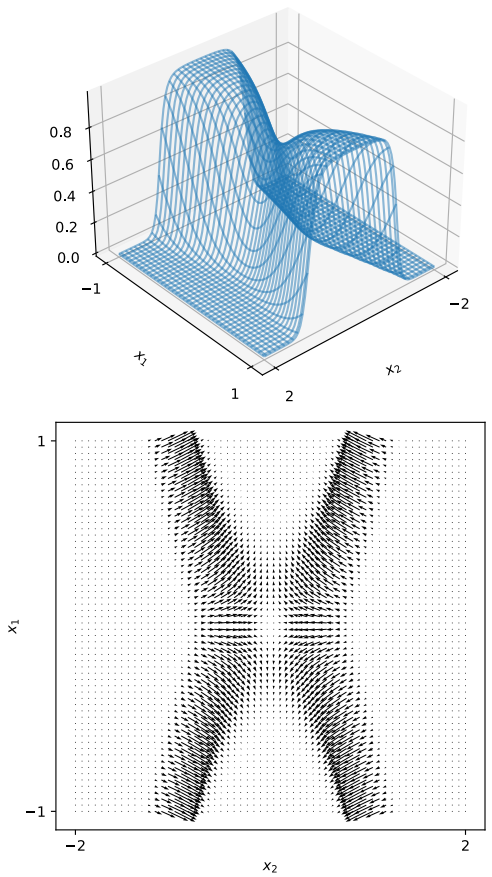



smoothing-
sharpness: 7

# Application to Code

k =
$x_1 > {x_2}^2 \wedge x_1 < -{x_2}^2$

```
if(k):
  y = 1;
else:
  y = 0;
```

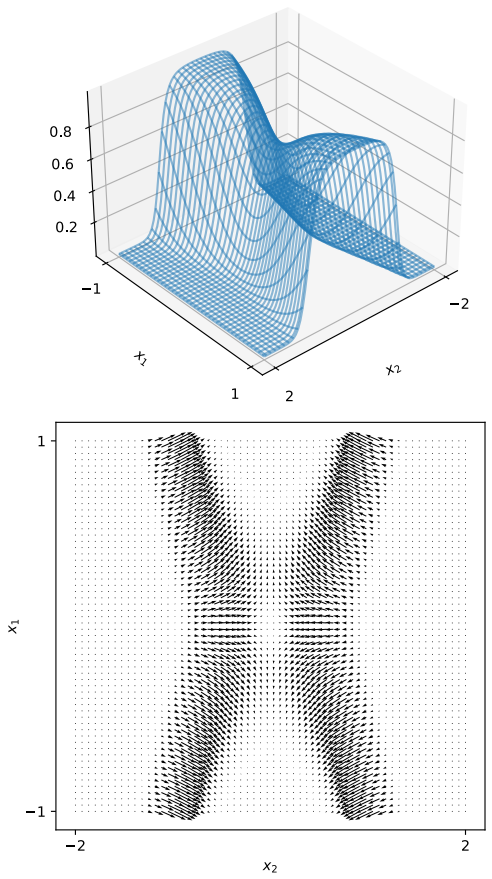



smoothing-
sharpness: 6

# Application to Code

k =
$x_1 > {x_2}^2 \wedge x_1 < -{x_2}^2$

```
if(k):
  y = 1;
else:
  y = 0;
```




smoothing-
sharpness: 5

# Automatic Differentiation (AD) and Smoothing

```
template <typename T> f(const T& x1, const T& x2){
  // ...
}
double x = 2;
double y = f(x);
```

# Automatic Differentiation (AD) and Smoothing

```
template <typename T> f(const T& x1, const T& x2){
  // ...
}
double x = 2;
double y = f(x);
```

AD implemented by overloading

```
#include <ad_tool>

ad_type<double> x = 2;
auto [y, y_dx] = f(x);
```

# Automatic Differentiation (AD) and Smoothing

```
template <typename T> f(const T& x1, const T& x2){
  // ...
}
double x = 2;
double y = f(x);
```

AD implemented by overloading

```
#include <ad_tool>

ad_type<double> x = 2;
auto [y, y_dx] = f(x);
```

Smoothing by overloading

```
#include "smoothing.h"

sm_type<double> x = 2;
double y_sm = f(x);
```

# Automatic Differentiation (AD) and Smoothing

Derivative of smooth function

```
#include <ad_tool>
#include "smoothing.h"

sm_type<ad_type<double>> x = 2;
auto [y, y_dx_smooth] = f(x);
```

# Automatic Differentiation (AD) and Smoothing

Derivative of smooth function

```
#include <ad_tool>
#include "smoothing.h"

sm_type<ad_type<double>> x = 2;
auto [y, y_dx_smooth] = f(x);
```

Smoothed derivative

```
#include <ad_tool>
#include "smoothing.h"

ad_type<sm_type<double>> x = 2;
auto [y_smooth, y_smooth_dx] = f(x);
```

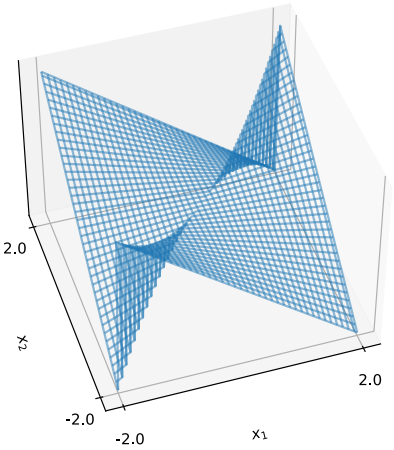
# Automatic Differentiation (AD) and Smoothing

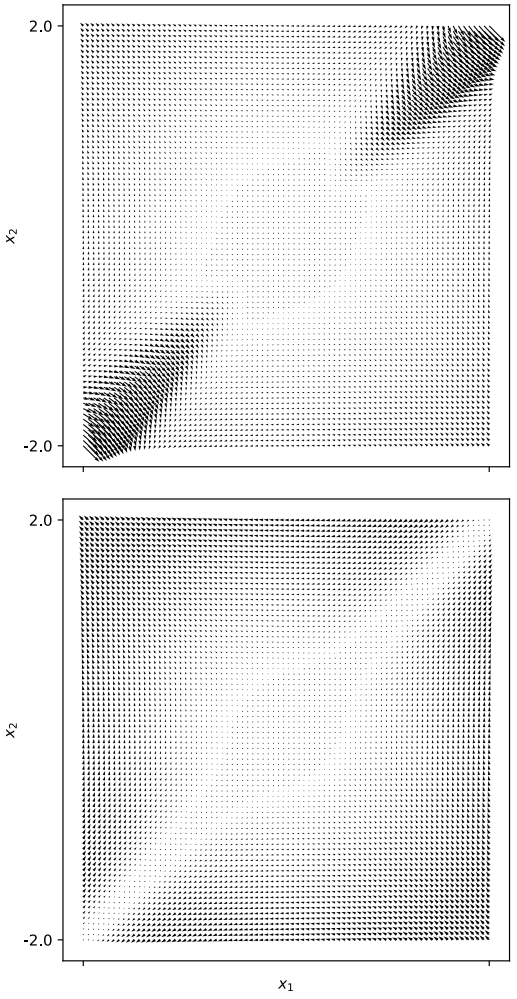
Derivative of smooth function

```
#include <ad_tool>
#include "smoothing.h"

sm_type<ad_type<double>> x = 2;
auto [y, y_dx_smooth] = f(x);
```

Smoothed derivative

```
#include <ad_tool>
#include "smoothing.h"

ad_type<sm_type<double>> x = 2;
auto [y_smooth, y_smooth_dx] = f(x);
```



# Summary

# Summary

- Smoothing of Concatenated Functions, complexity dependent on branching, not dimensionality

# Summary

- Smoothing of Concatenated Functions, complexity dependent on branching, not dimensionality
- Heuristic reduces complexity: Only contributing paths

# Summary

- Smoothing of Concatenated Functions, complexity dependent on branching, not dimensionality
- Heuristic reduces complexity: Only contributing paths
- Implementation: Operator Overloading + Tape (And Automatic Differentiation)

# Summary

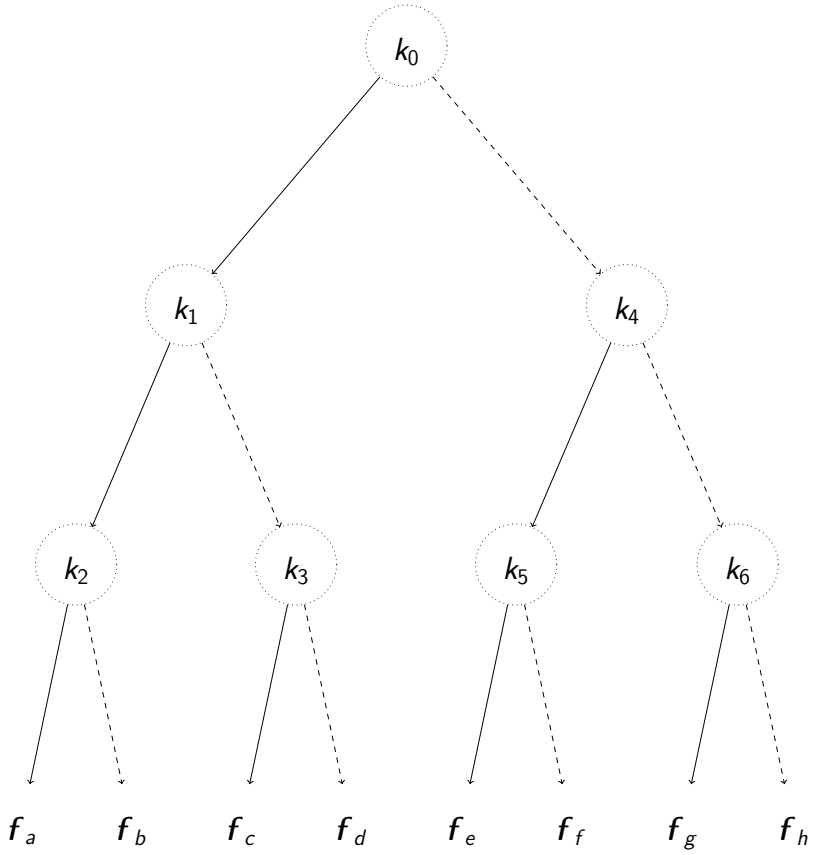
- Smoothing of Concatenated Functions, complexity dependent on branching, not dimensionality
- Heuristic reduces complexity: Only contributing paths
- Implementation: Operator Overloading + Tape (And Automatic Differentiation)
- Smooth Function and meaningful gradient at discontinuities

# Summary

Thanks for your Attention.
Questions

## Application to Generic Code

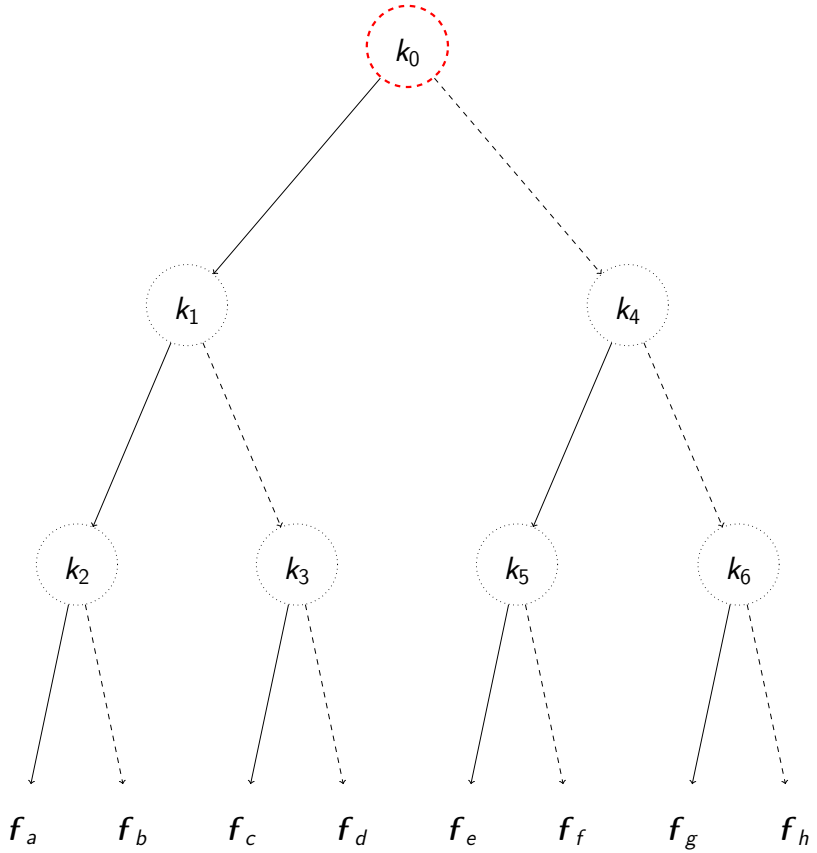
We can convert any code to a tree



# Appendix

$c =$

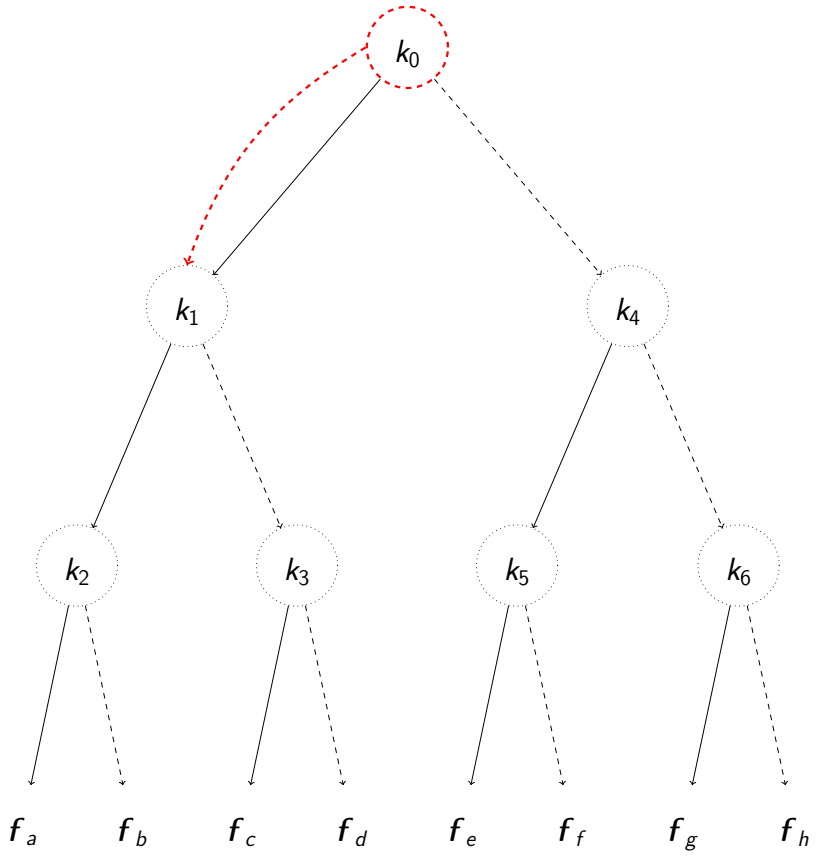
$y =$



# Appendix

$$c = \sigma_{k_0}$$

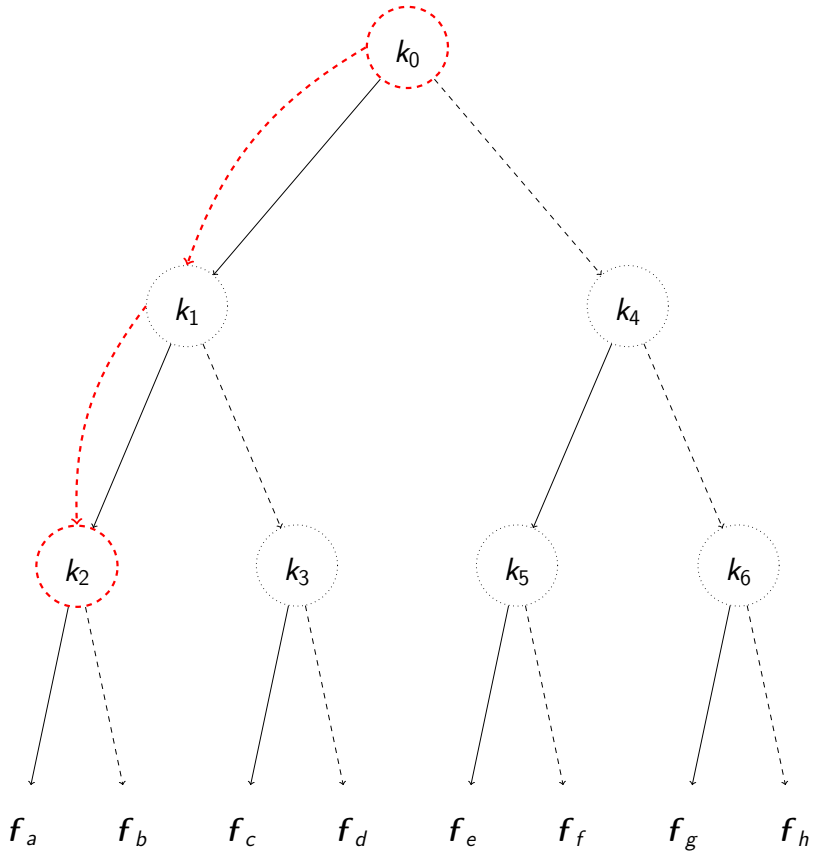
$$y =$$



# Appendix

$$c = \sigma_{k_0} \cdot 1$$

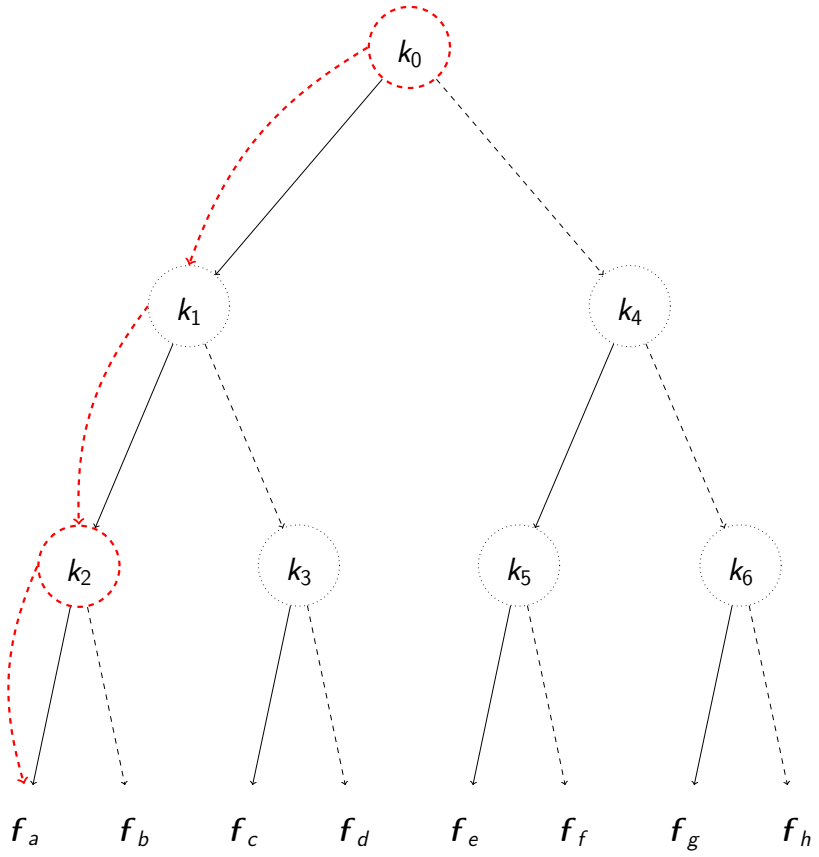
$$y =$$



# Appendix

$$c = \sigma_{k_0} \cdot 1 \cdot \sigma_{k_2}$$

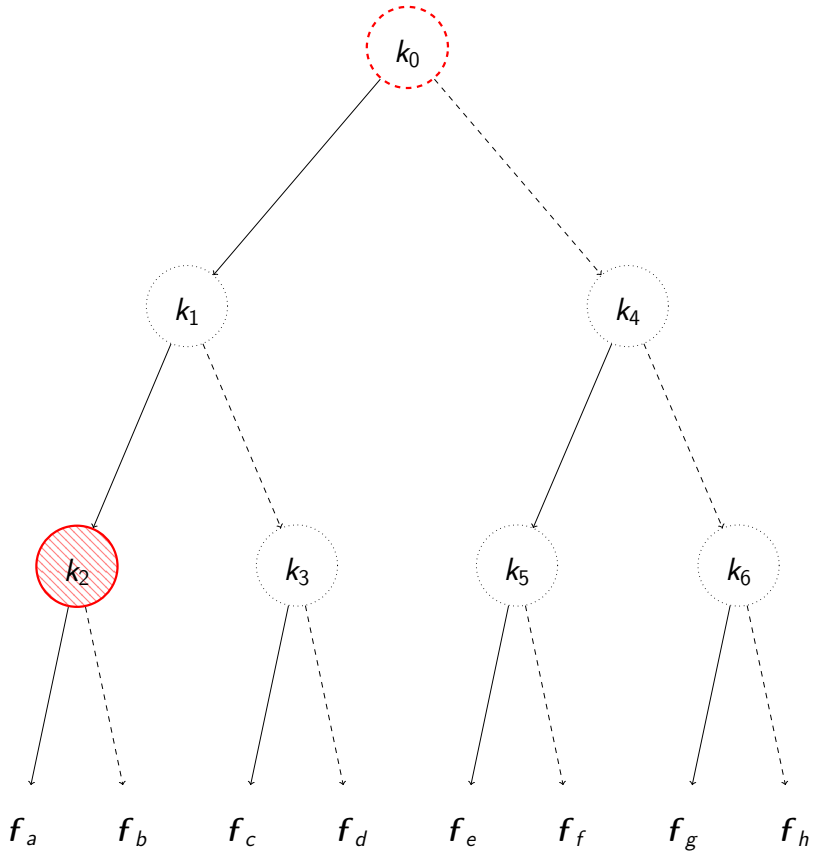
$$y =$$



# Appendix

$$c =$$

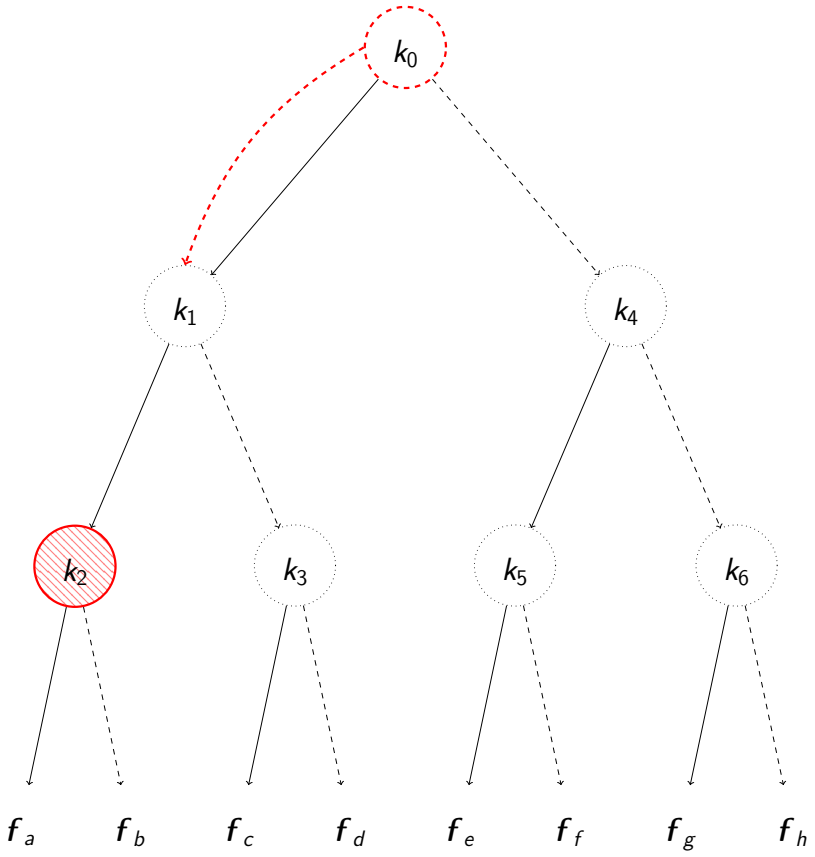
$$y = c_1 \cdot f_a$$



# Appendix

$$c = \sigma_{k_0}$$

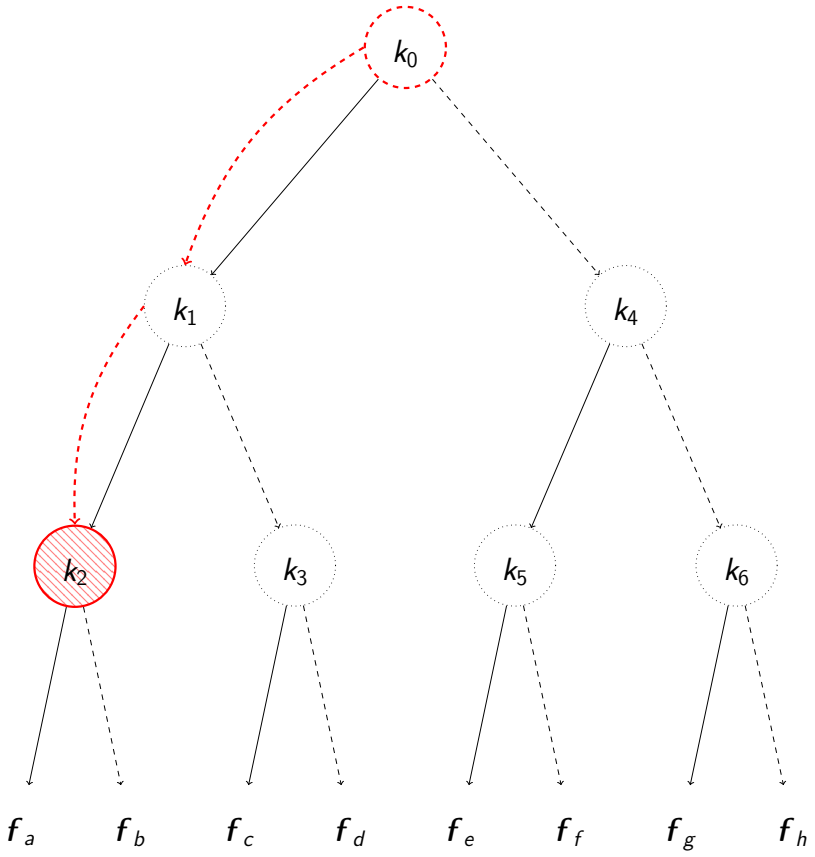
$$y = c_1 \cdot f_a$$



# Appendix

$$c = \sigma_{k_0} \cdot 1$$

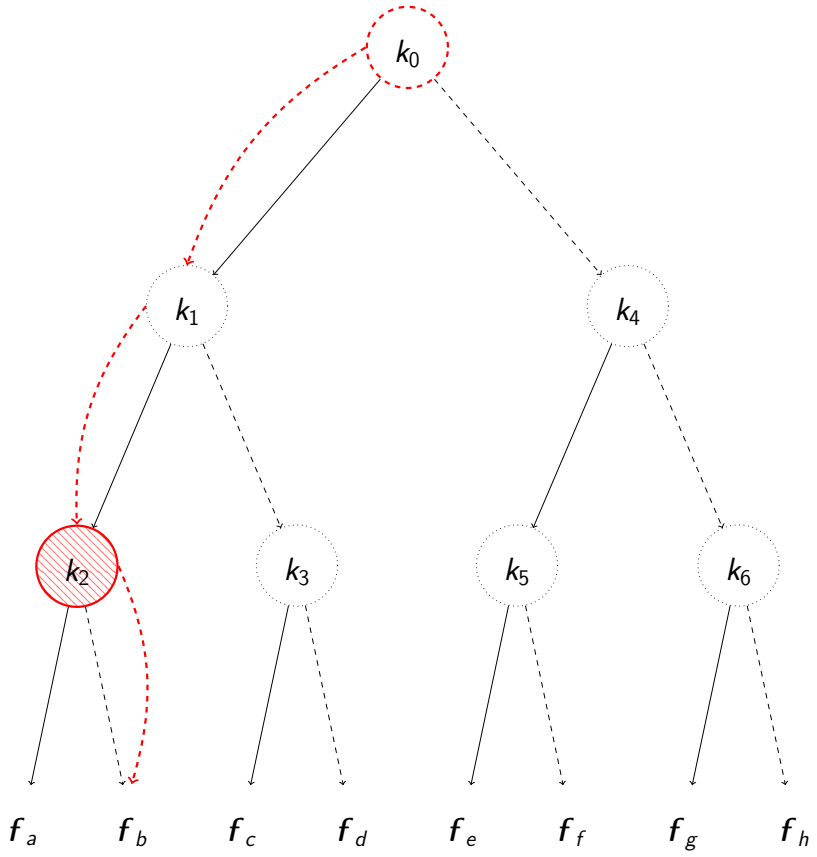
$$y = c_1 \cdot f_a$$



# Appendix

$$c = \sigma_{k_0} \cdot 1 \cdot \overline{\sigma}_{k_2}$$

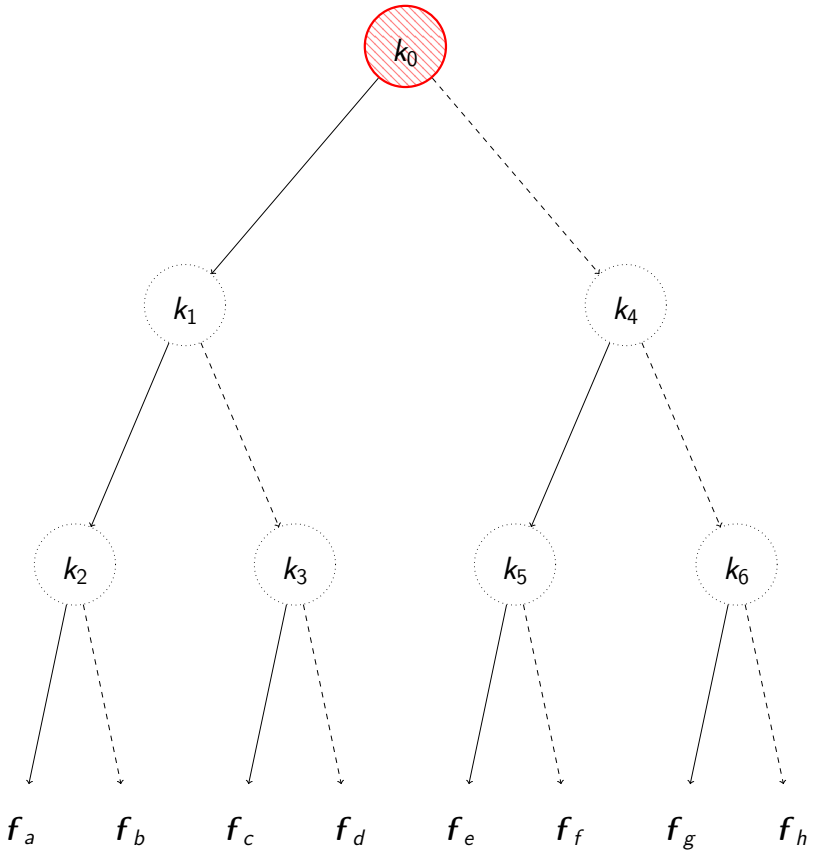
$$y = c_1 \cdot f_a$$



# Appendix

$$c =$$

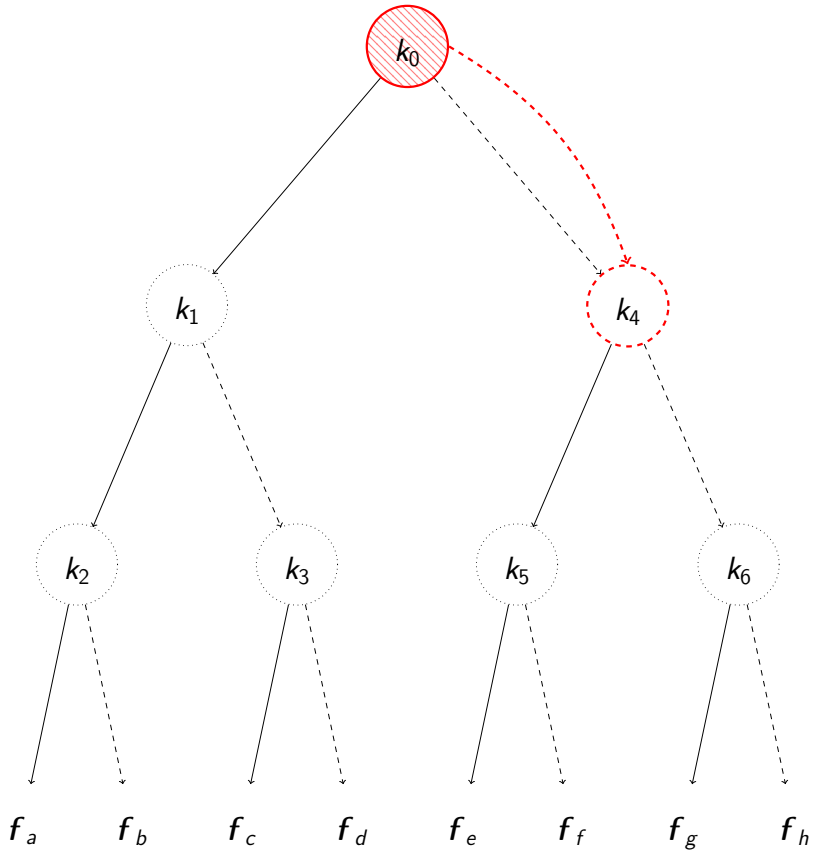
$$y = c_1 \cdot f_a + c_2 \cdot f_b$$



# Appendix

$$c = \overline{\sigma}_{k_0}$$

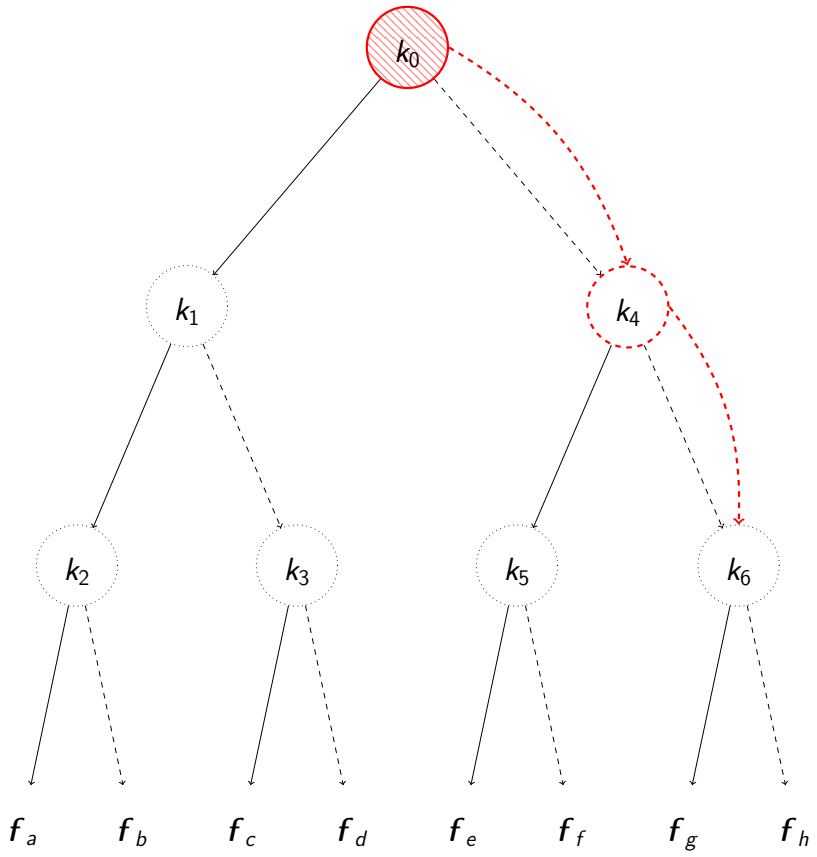
$$y = c_1 \cdot f_a + c_2 \cdot f_b$$



# Appendix

$$c = \overline{\sigma}_{k_0} \cdot \overline{\sigma}_{k_4}$$

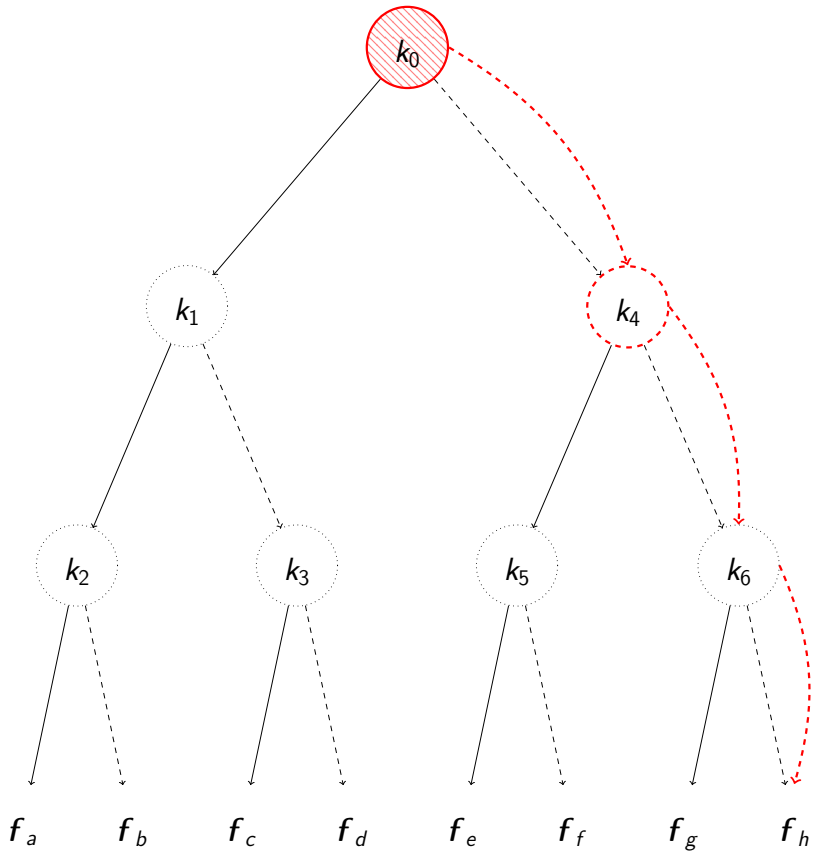
$$y = c_1 \cdot f_a + c_2 \cdot f_b$$



# Appendix

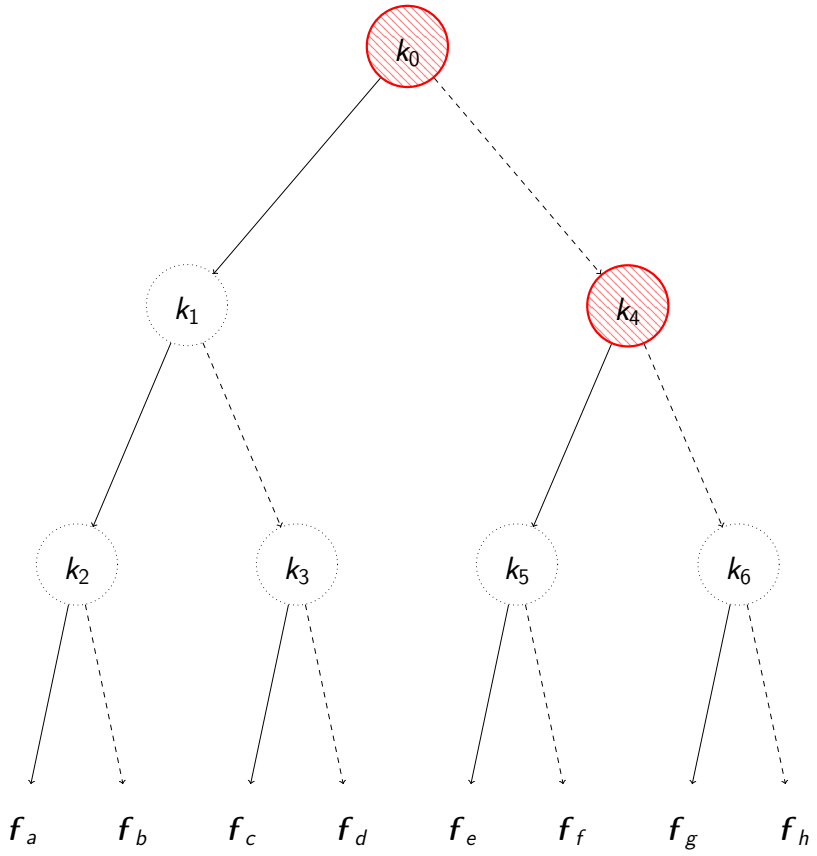
$$c = \overline{\sigma}_{k_0} \cdot \overline{\sigma}_{k_4} \cdot 1$$

$$y = c_1 \cdot f_a + c_2 \cdot f_b$$



# Appendix

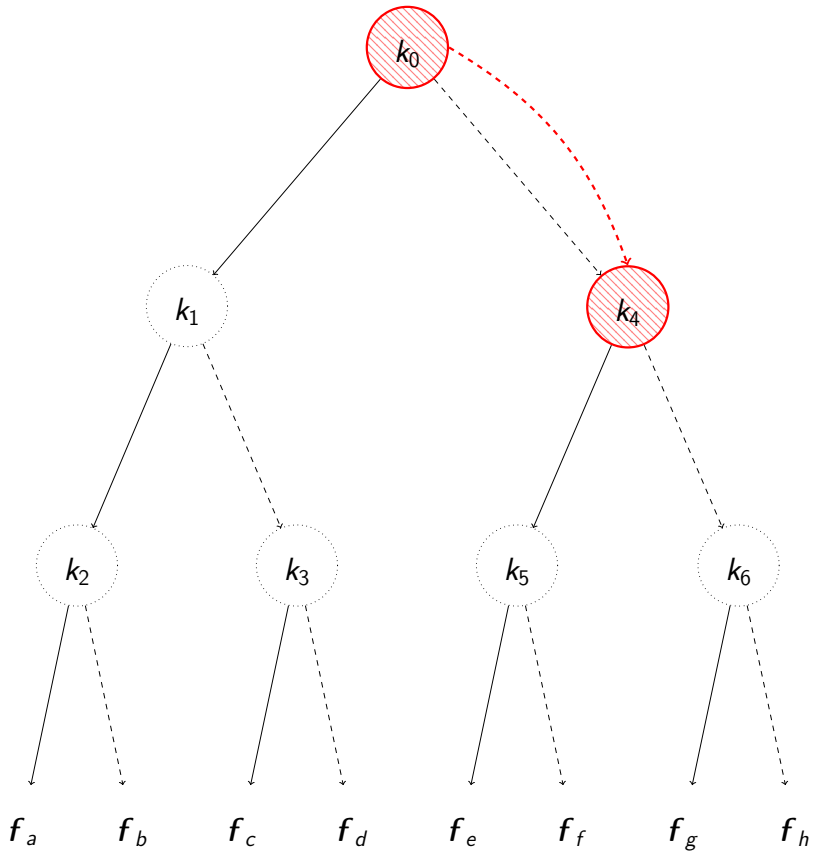
$$c = \begin{cases} y = c_1 \cdot f_a + c_2 \cdot f_b \\ \qquad + c_3 \cdot f_h \end{cases}$$



# Appendix

$$c = \overline{\sigma}_{k_0}$$

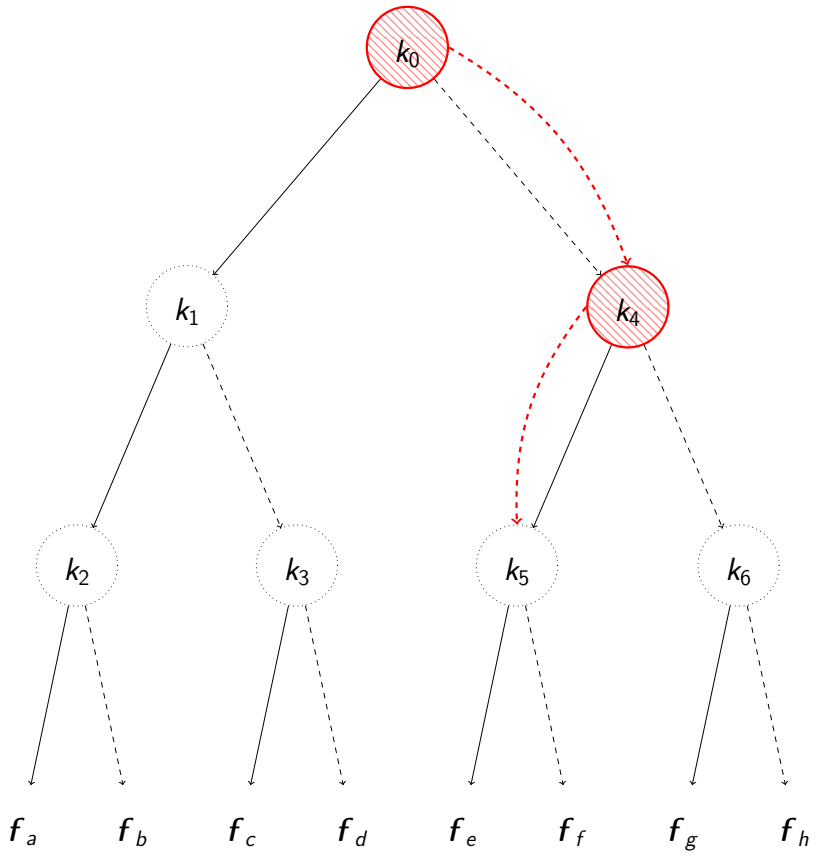
$$y = c_1 \cdot f_a + c_2 \cdot f_b + c_3 \cdot f_h$$



# Appendix

$$c = \overline{\sigma}_{k_0} \cdot \sigma_{k_4}$$

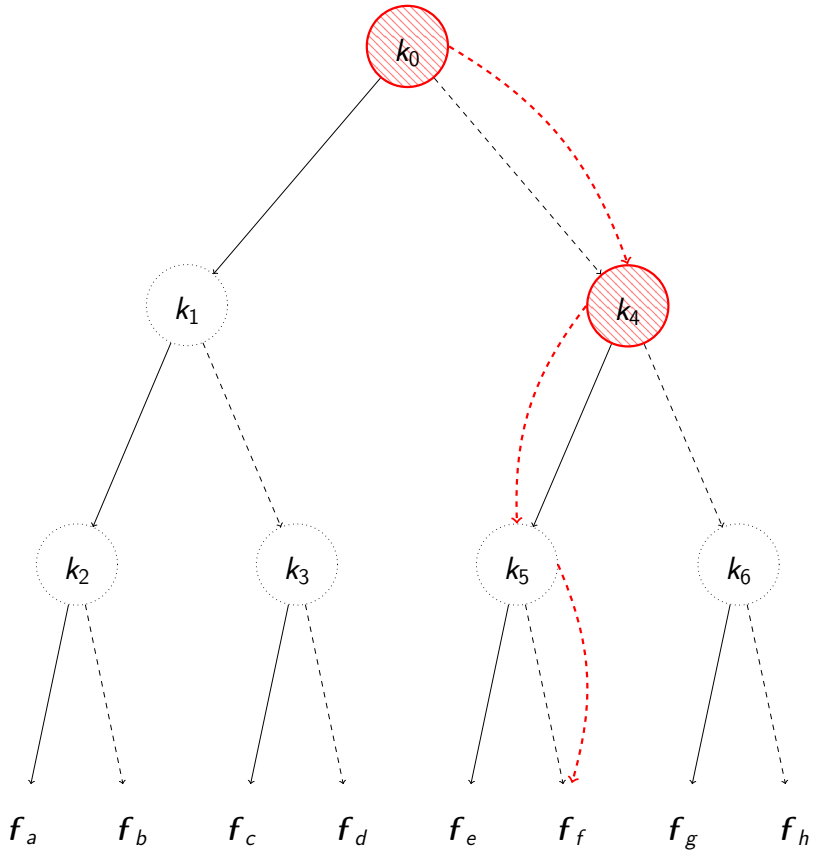
$$y = c_1 \cdot f_a + c_2 \cdot f_b + c_3 \cdot f_h$$



# Appendix

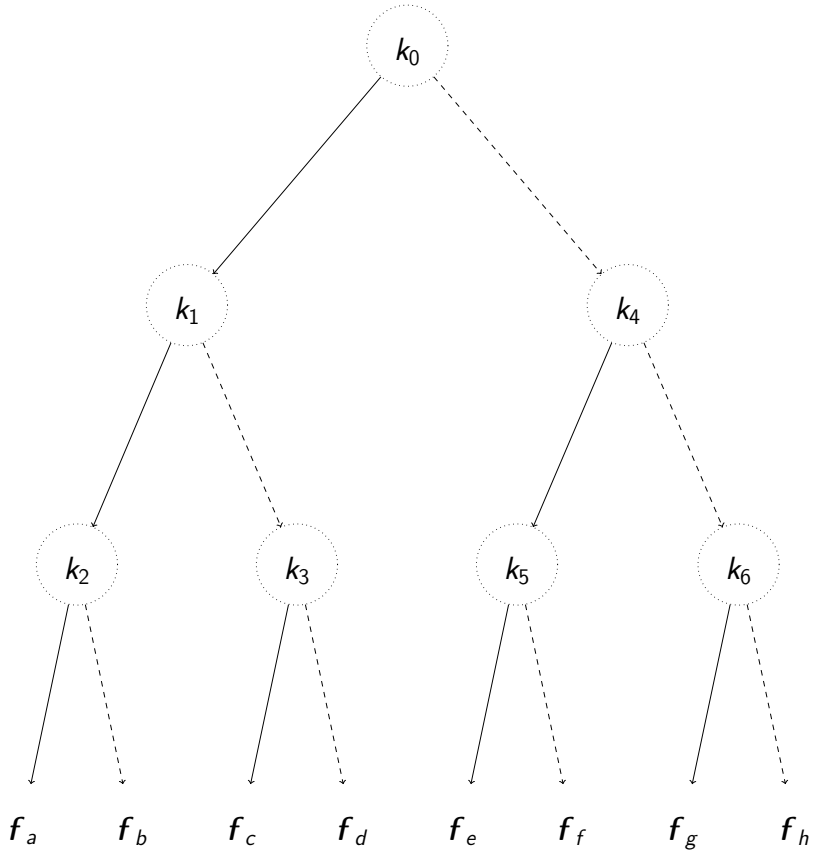
$$c = \overline{\sigma}_{k_0} \cdot \sigma_{k_4} \cdot 1$$

$$y = c_1 \cdot f_a + c_2 \cdot f_b + c_3 \cdot f_h$$

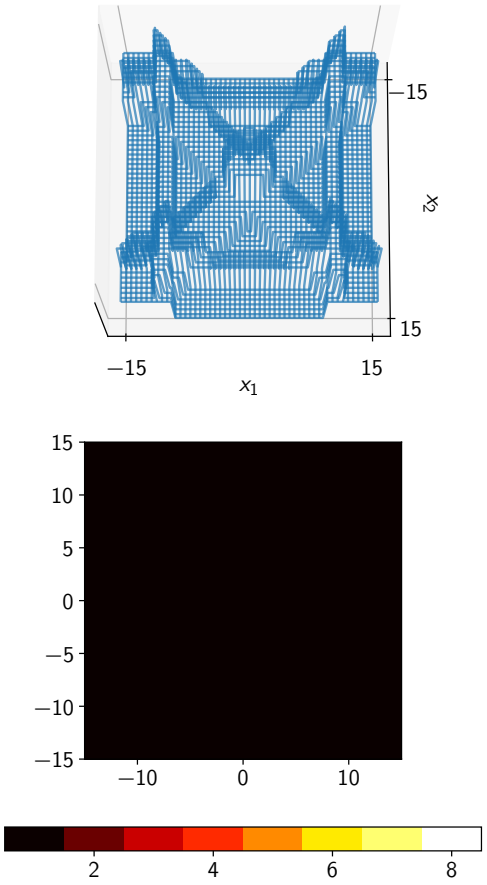


# Appendix

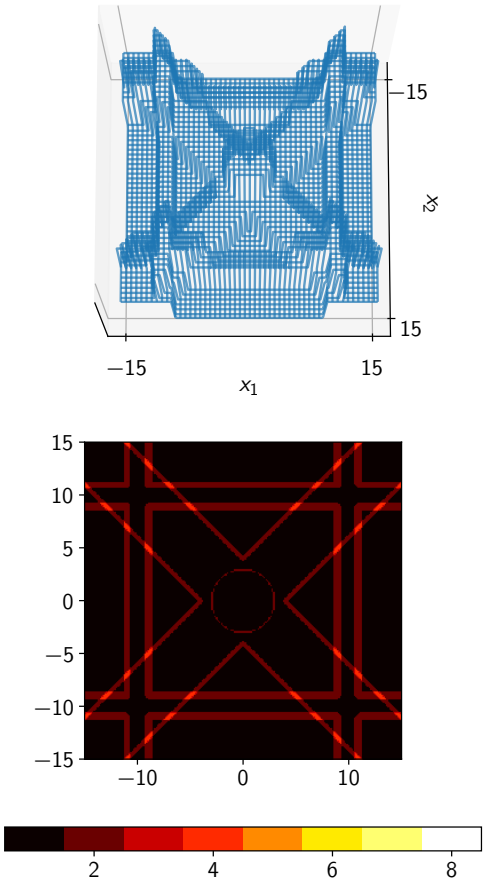
$$c = \begin{cases} y = c_1 \cdot f_a + c_2 \cdot f_b \\ \qquad + c_3 \cdot f_h + c_4 \cdot f_f \end{cases}$$



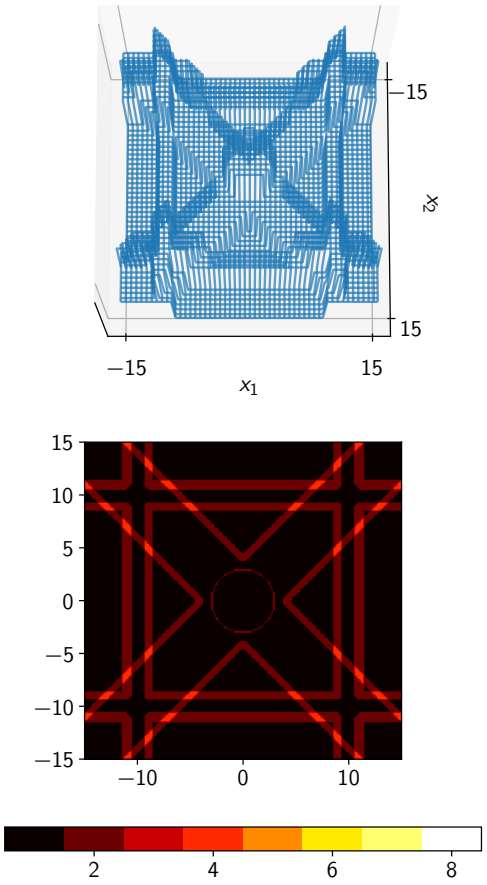
# Appendix



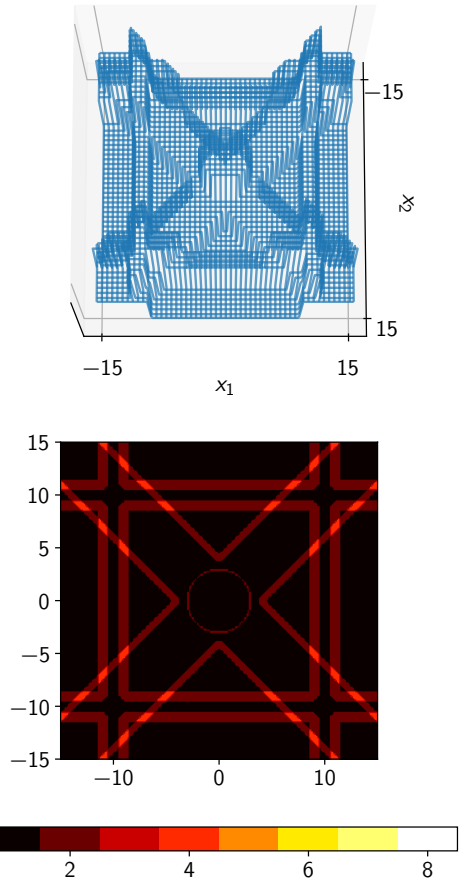
# Appendix



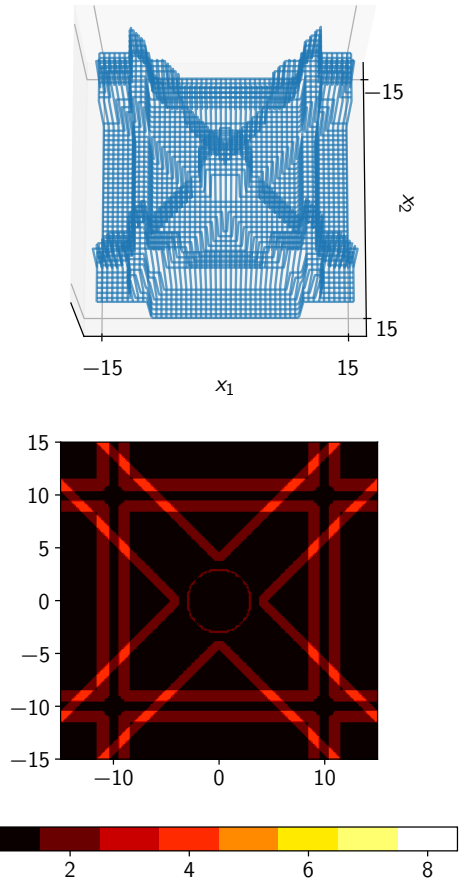
# Appendix



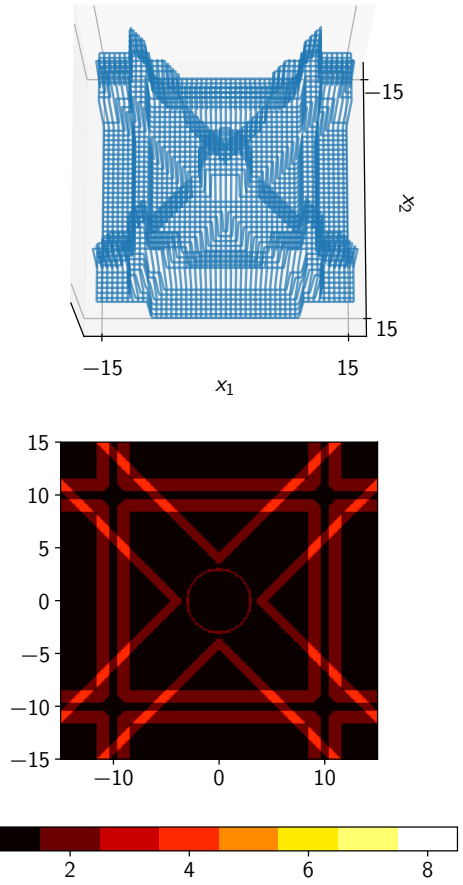
# Appendix



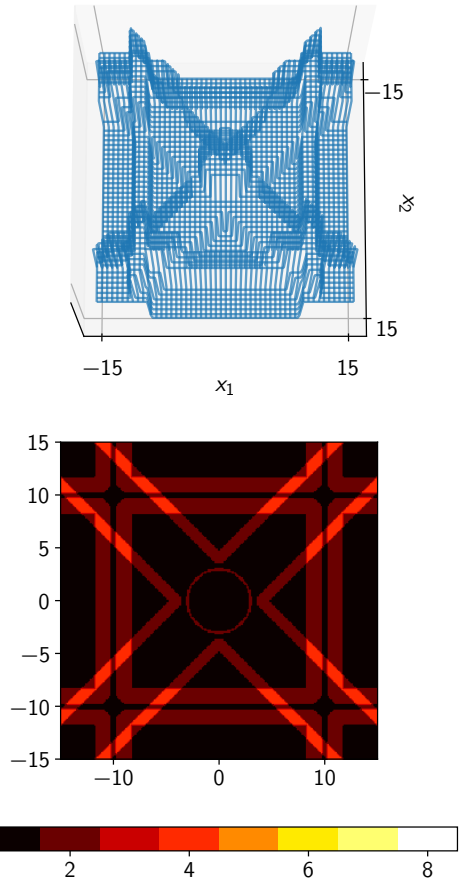
# Appendix



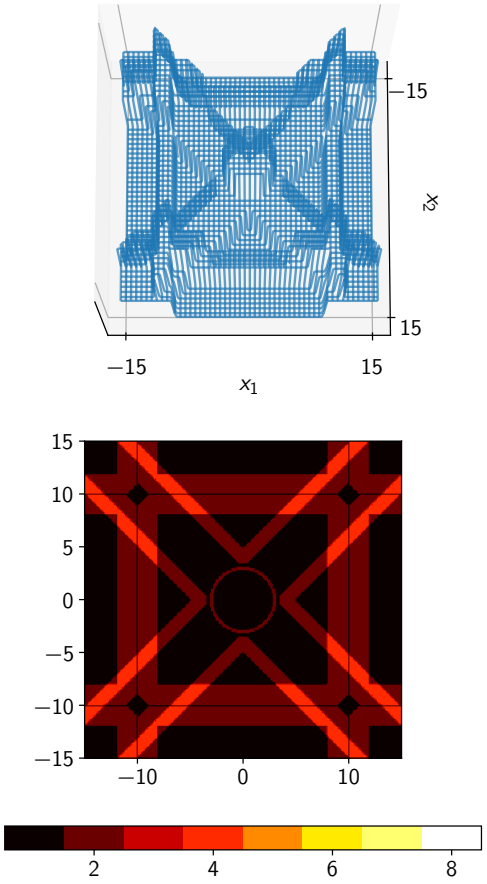
# Appendix



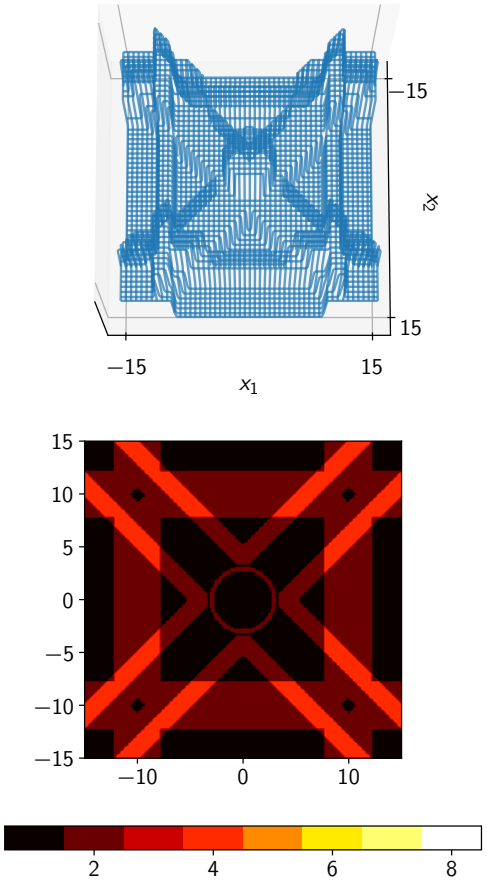
# Appendix



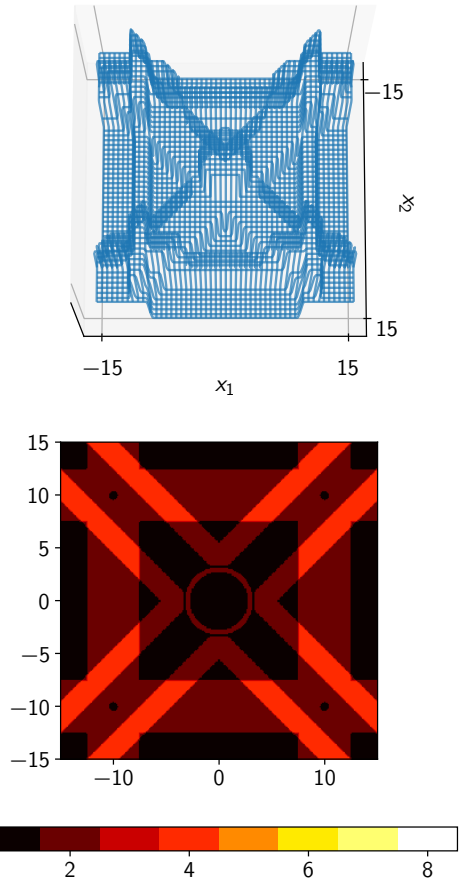
# Appendix



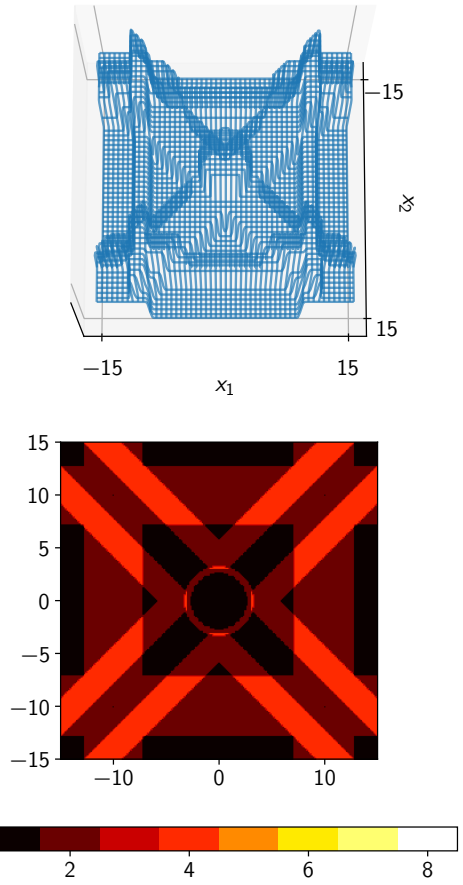
# Appendix



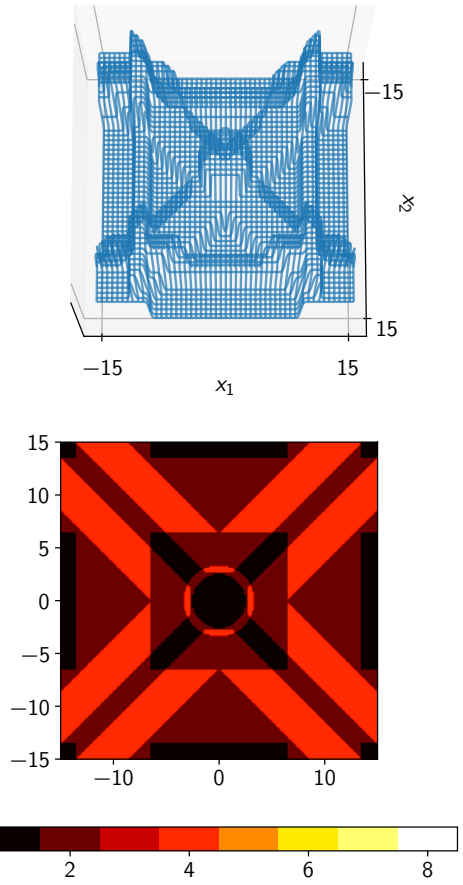
# Appendix



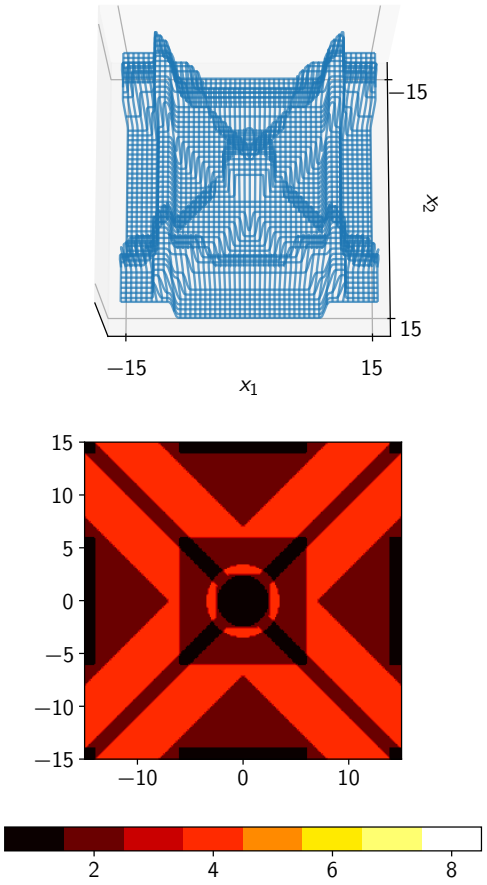
# Appendix



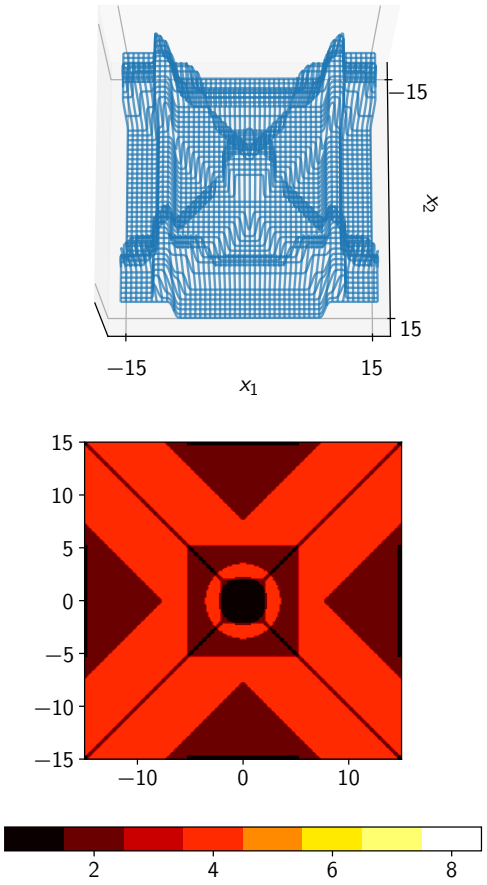
# Appendix



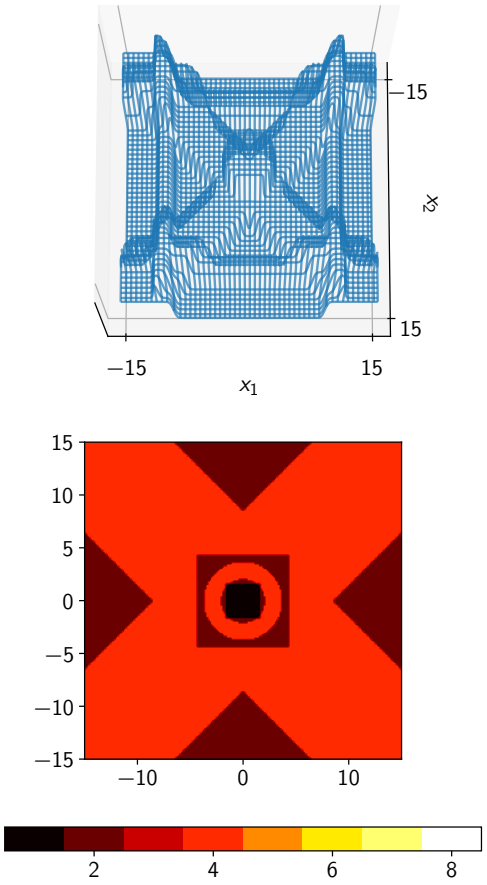
# Appendix



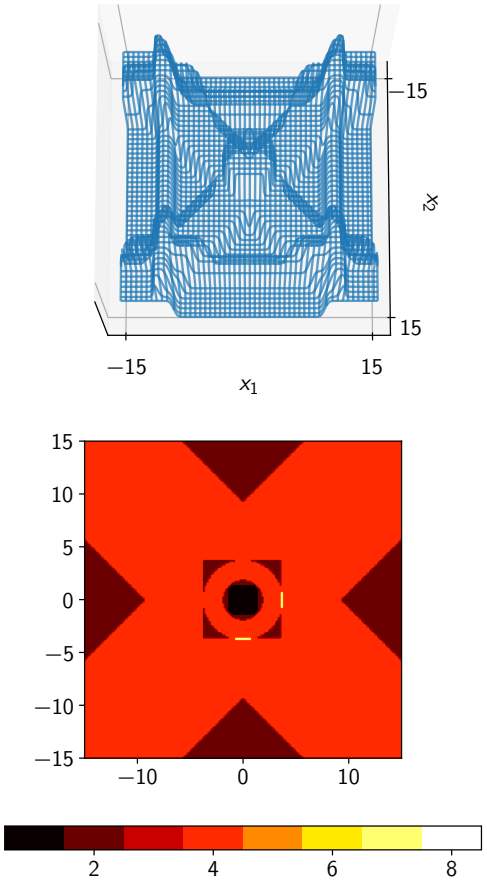
# Appendix



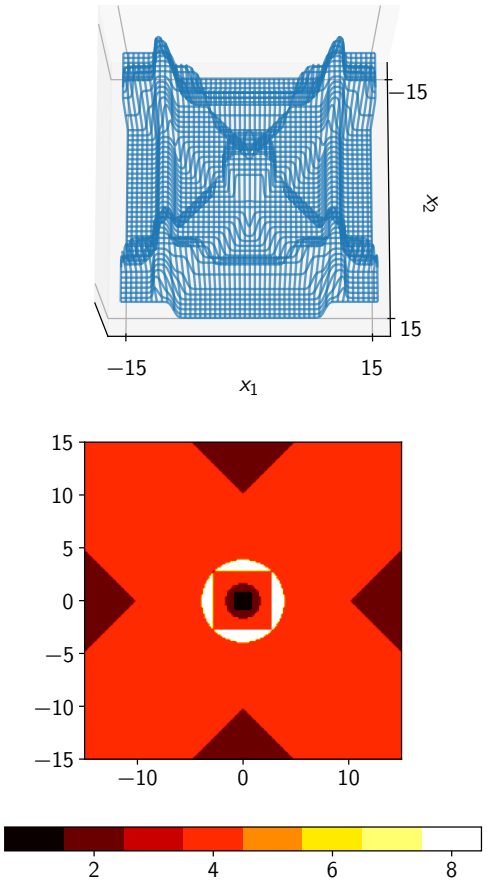
# Appendix



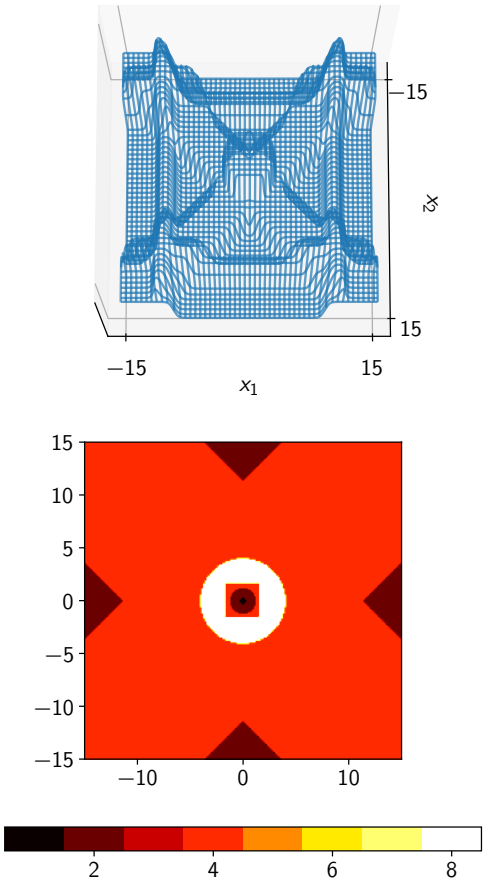
# Appendix



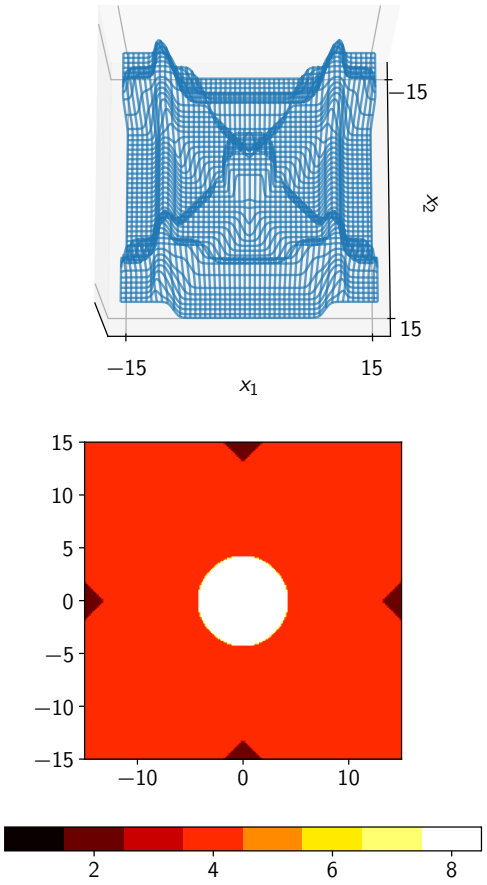
# Appendix



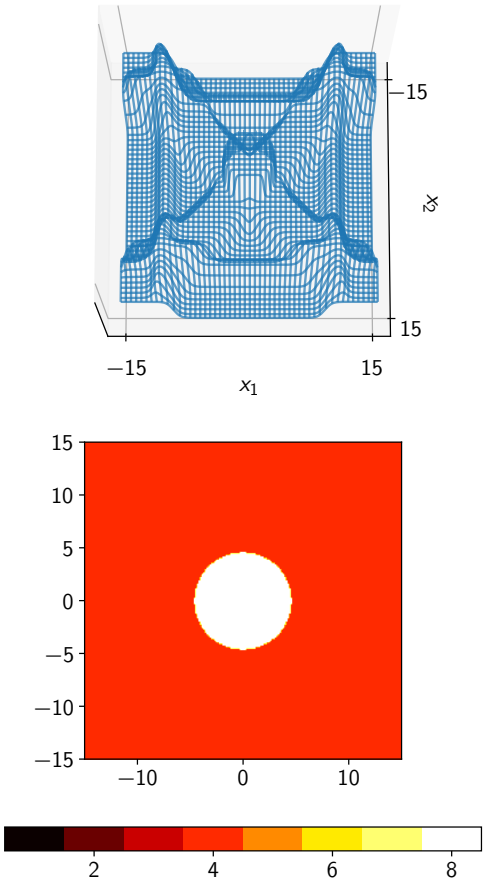
# Appendix



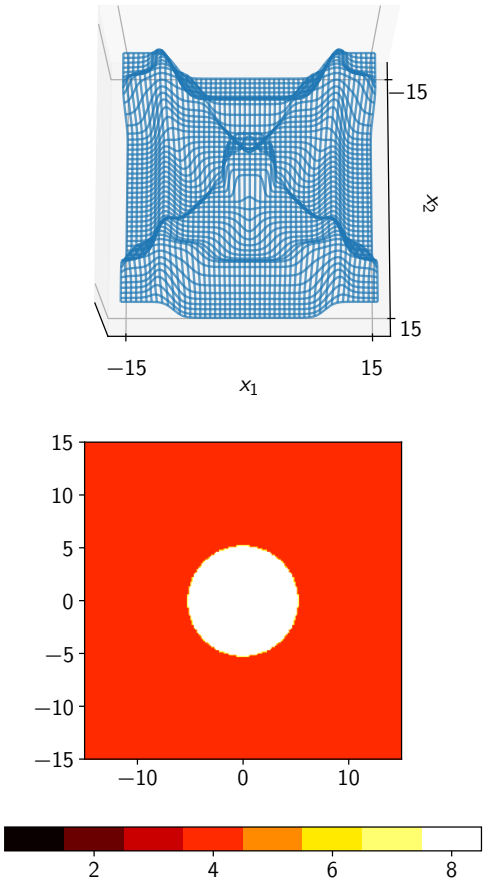
# Appendix



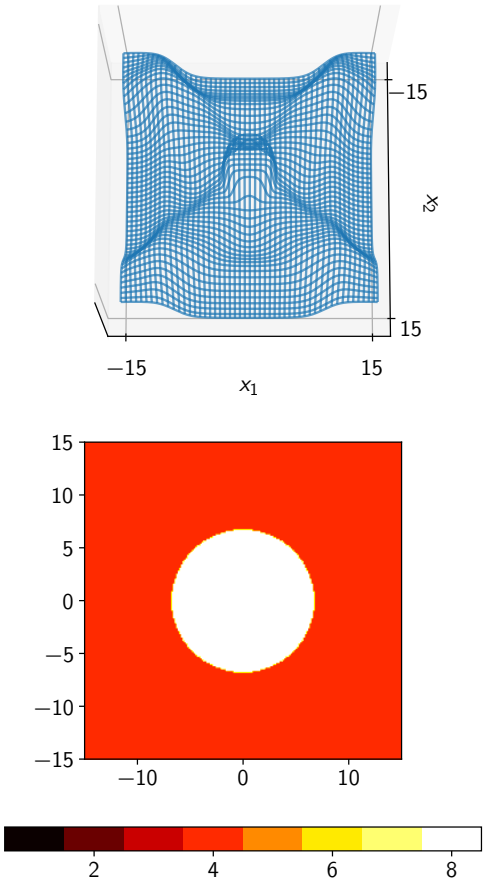
# Appendix



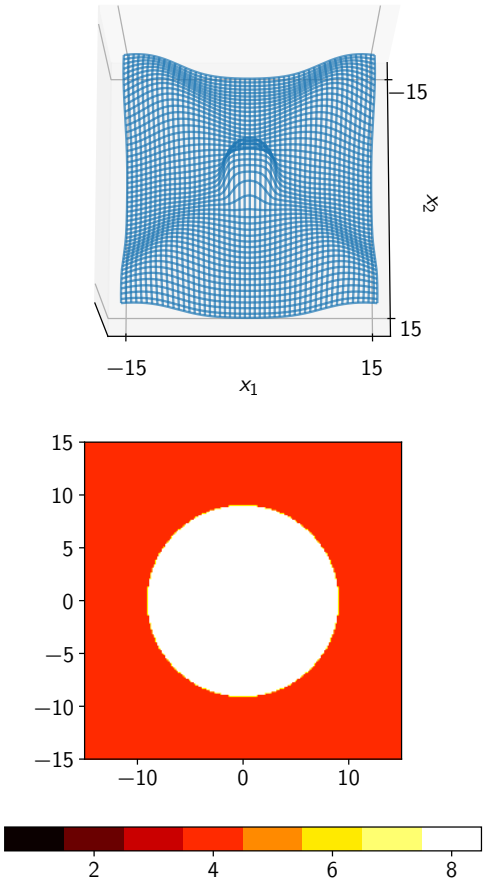
# Appendix



# Appendix



# Appendix



# Appendix

General Problem with Smoothing

$$f = \begin{cases} f_1(x) = x, & \text{if } x < 2 \\ f_2(x) = -x, & \text{else} \end{cases} \quad \textit{and} \quad g(x) = -x^2 + 2$$

In $f \circ g$

# Appendix

General Problem with Smoothing

$$f = \begin{cases} f_1(x) = x, & \text{if } x < 2 \\ f_2(x) = -x, & \text{else} \end{cases} \quad \textit{and} \quad g(x) = -x^2 + 2$$

In $f \circ g$

$g(x)$ *touches* $x_f = 2$ but does not go further. How should we smooth

# Appendix

General Problem with Smoothing

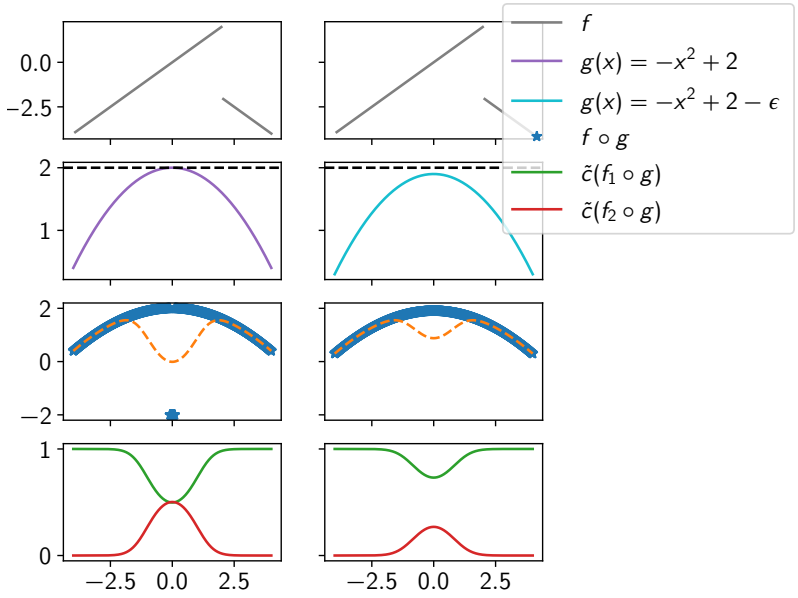
$$f = \begin{cases} f_1(x) = x, & \text{if } x < 2 \\ f_2(x) = -x, & \text{else} \end{cases} \quad \textit{and} \quad g(x) = -x^2 + 2$$

In $f \circ g$

$g(x)$ *touches* $x_f = 2$ but does not go further. How should we smooth

Smoothing will be similar similar if $g(x) = -x^2 + 2 - \epsilon$ and $g(x)$ in $f \circ g$ never reaches $x_f = 2$.

# Appendix


0.0
−2.5
2
1
2
0
−2
1
0
−2.5
0.0
2.5
−2.5
0.0
2.5
$f$
$g(x) = -x^2 + 2$
$g(x) = -x^2 + 2 - \epsilon$
$f \circ g$
$\tilde{c}(f_1 \circ g)$
$\tilde{c}(f_2 \circ g)$