ETH zürich

# Seminar Arbeit: *Implementation of “Density Estimation with Distribution Element Trees”*

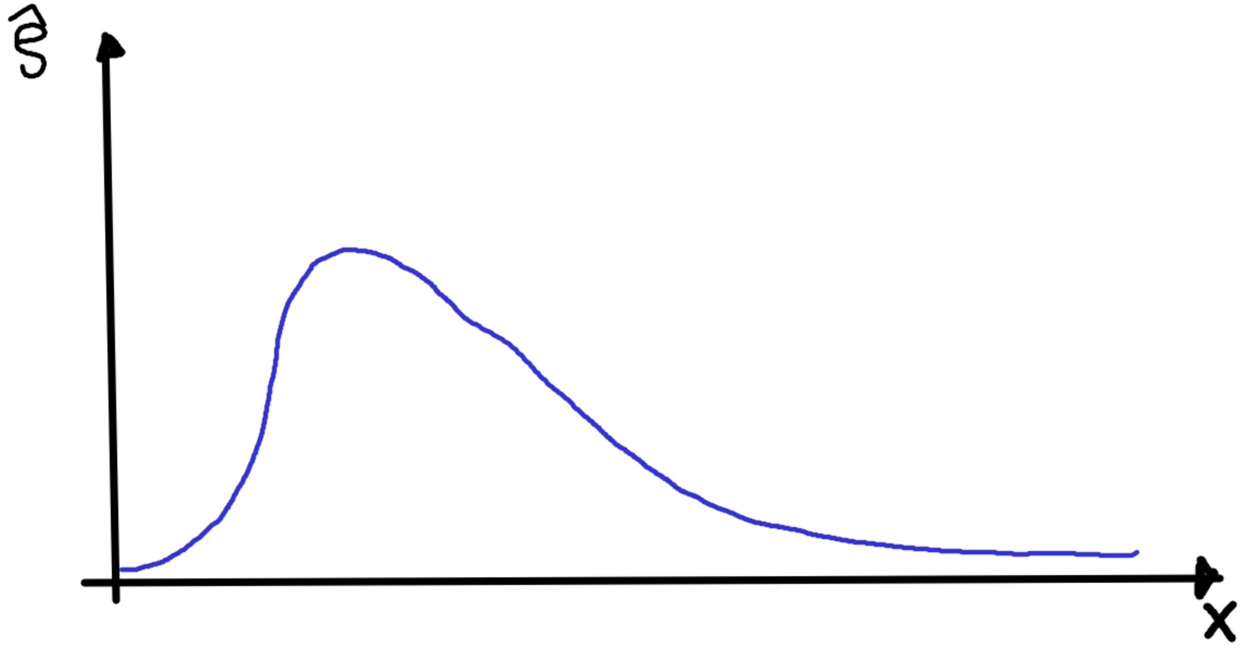
By Philip Paul Müller

## General Idea of the Method

- Approximate a (complicated) pdf by (many) simple ones.
  - We assume pairwise independent dimensions.
- If the simple model can not explain the data, split the domain into smaller ones.
  - Repeat the process until the simple model is “good enough”.

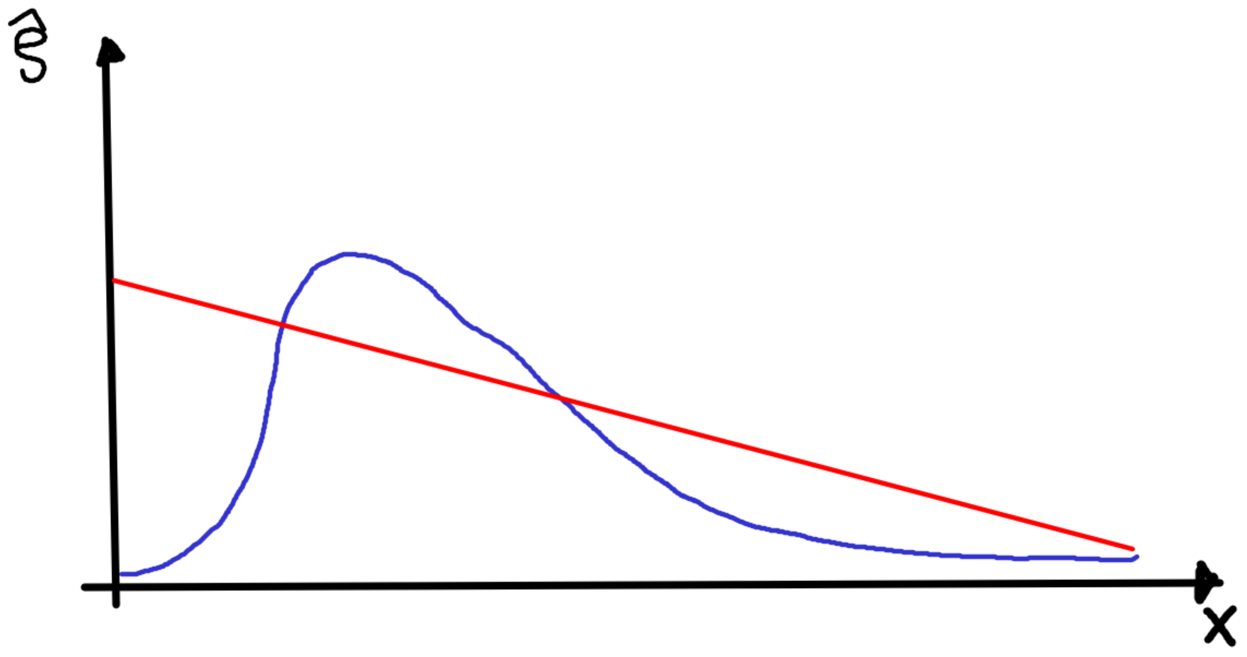
## General Idea of the Method

- Approximate a (complicated) pdf by (many) simple ones.
  - We assume pairwise independent dimensions.
- If the simple model can not explain the data, split the domain into smaller ones.
  - Repeat the process until the simple model is “good enough”.



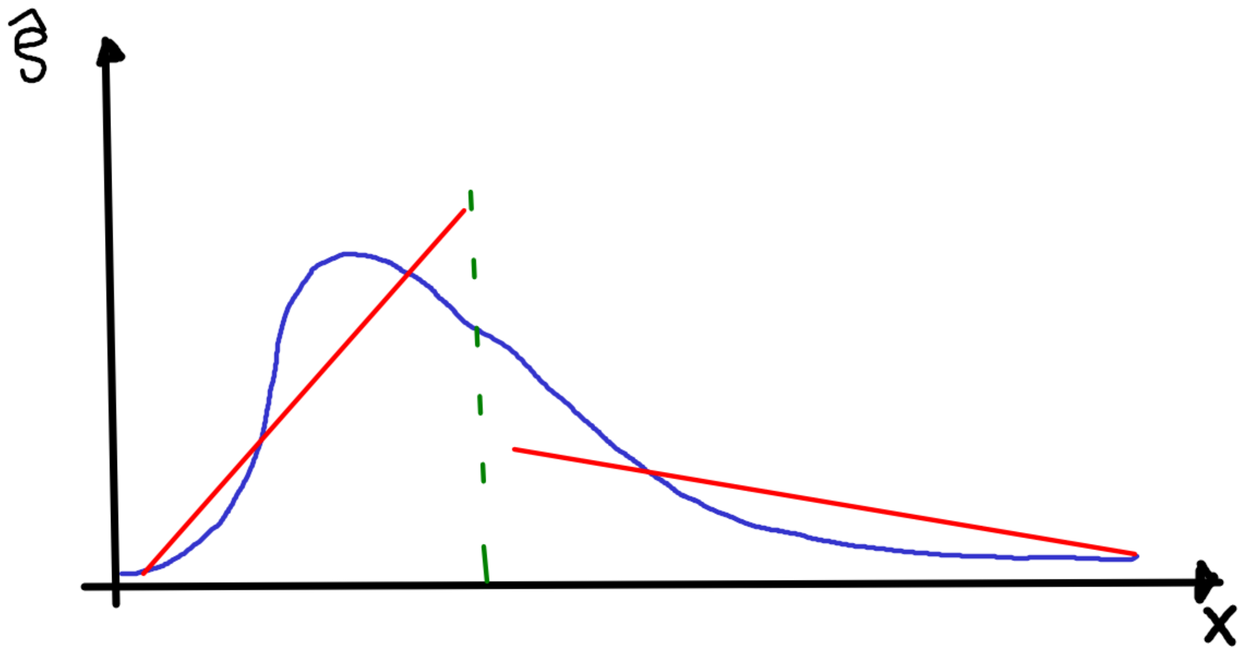
## General Idea of the Method

- Approximate a (complicated) pdf by (many) simple ones.
  - We assume pairwise independent dimensions.
- If the simple model can not explain the data, split the domain into smaller ones.
  - Repeat the process until the simple model is “good enough”.



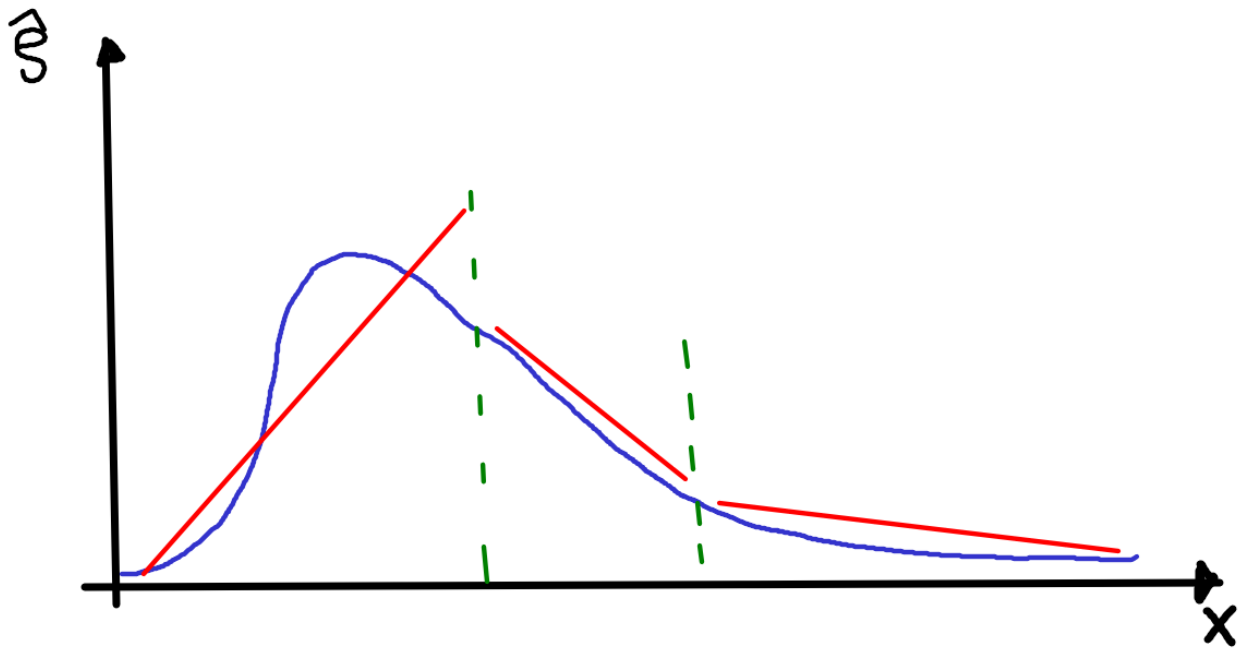
## General Idea of the Method

- Approximate a (complicated) pdf by (many) simple ones.
  - We assume pairwise independent dimensions.
- If the simple model can not explain the data, split the domain into smaller ones.
  - Repeat the process until the simple model is “good enough”.



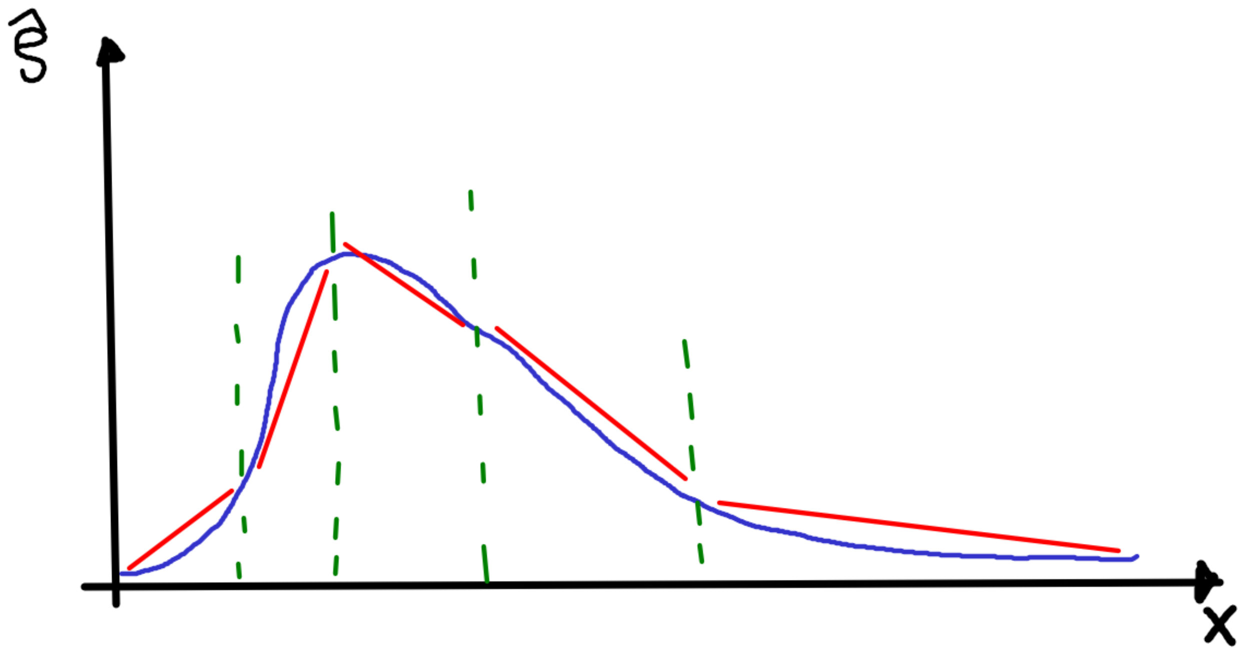
## General Idea of the Method

- Approximate a (complicated) pdf by (many) simple ones.
  - We assume pairwise independent dimensions.
- If the simple model can not explain the data, split the domain into smaller ones.
  - Repeat the process until the simple model is “good enough”.



## General Idea of the Method

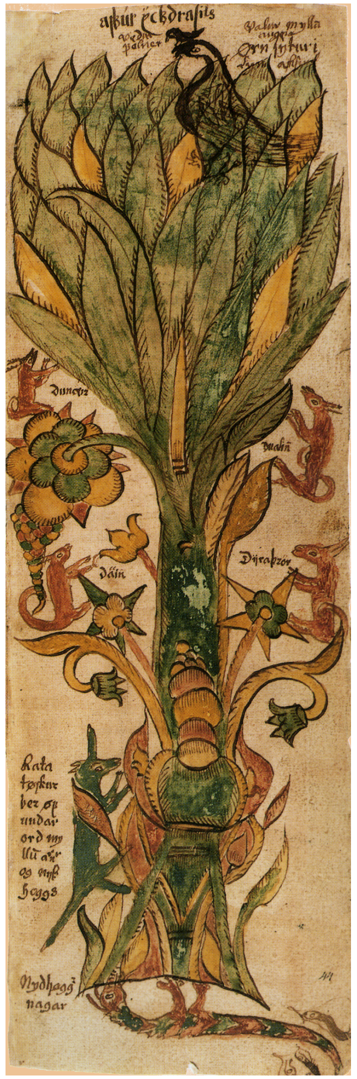
- Approximate a (complicated) pdf by (many) simple ones.
  - We assume pairwise independent dimensions.
- If the simple model can not explain the data, split the domain into smaller ones.
  - Repeat the process until the simple model is “good enough”.



## General Idea of the Method

- Approximate a (complicated) pdf by (many) simple ones.
  - We assume pairwise independent dimensions.
- If the simple model can not explain the data, split the domain into smaller ones.
  - Repeat the process until the simple model is “good enough”.
- The resulting tree can also be used to draw new samples.
  - It is also possible to apply conditions to them.

## Yggdrasil



https://commons.wikimedia.org/wiki/File:AM_738_4to_Yggdrasill.png
https://upload.wikimedia.org/wikipedia/commons/8/8d/The_Ash_Yggdrasil_by_Friedrich_Wilhelm_Heine.jpg

## Yggdrasil

- Written in C++14:
  - Self monitoring design.
  - *Enforcing* of pre/post conditions.
  - No platform dependent code.
- CMake is used for building the code.

## Yggdrasil

- Written as a (static) library.
    - Easy integration into other projects.
- Object oriented design.
    - Written in an extensible fashion.

## Yggdrasil

- Well documented
  - Extensively commented code (in addition to Doxygen)
  - A ~30 pages manual
  - Jupyter notebook tutorials

```
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
C++                            129           8231           6531          23938
C/C++ Header                    73           5540          18133          13084
Markdown                         4            216              0            781
Python                          10            225             70            531
CMake                           16            165            137            413
Bourne Shell                     6             62             54             80
-------------------------------------------------------------------------------
SUM:                           238          14439          24925          38827
-------------------------------------------------------------------------------
```

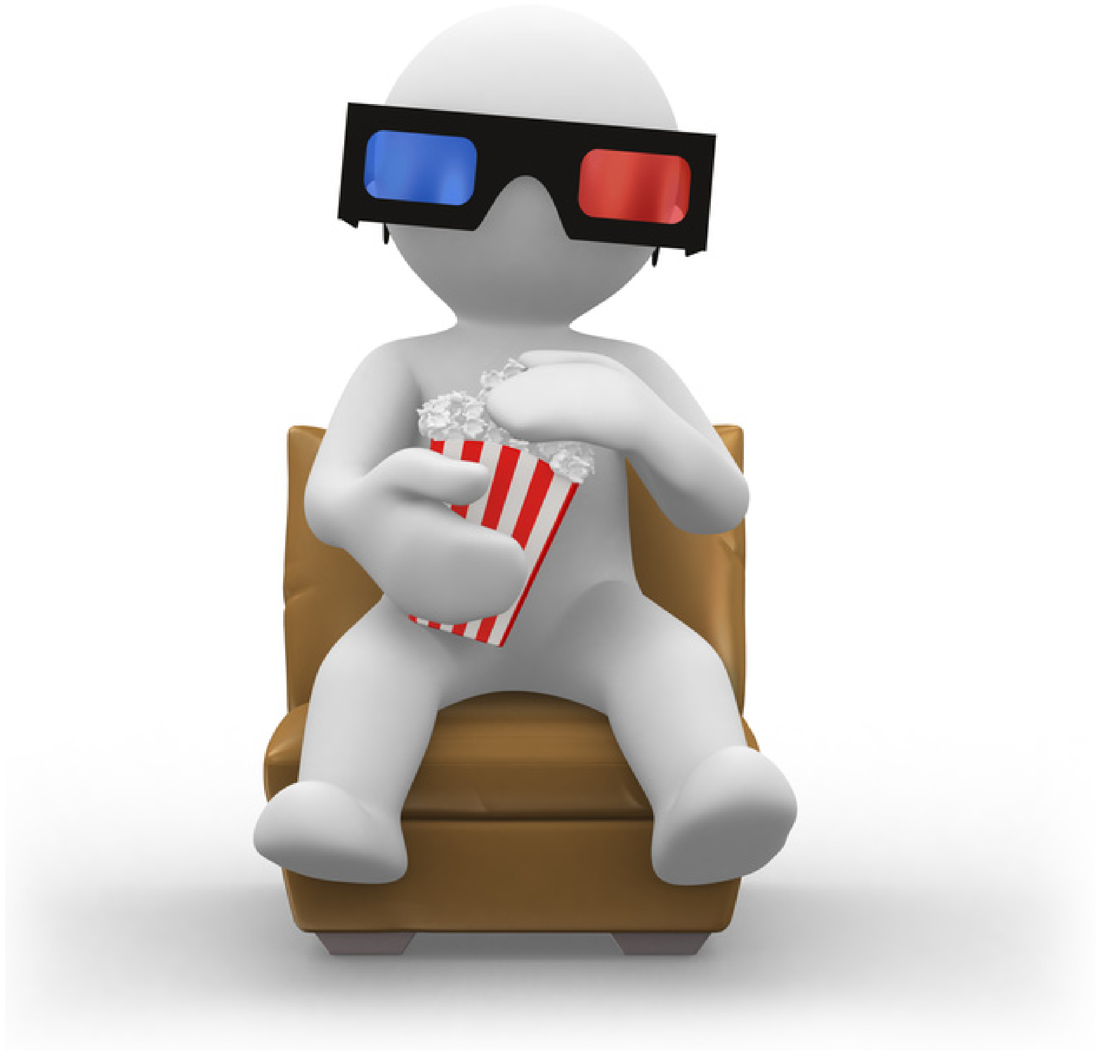
## pyYggdrasil

- A wrapper that allows using Yggdrasil from within Python.

## pyYggdrasil

- pyYggdrasil makes use of pybind11.
  - pybind11 was written by Jakob Wenzel, et al.
- Direct integration of the Python “Docstring” system, *i.e.* `help()` works.
- Direct interaction between NumPy and Eigen (no copy needed).
- Transparent conversions of C++ exceptions to Python exceptions.
- But, no references so copies are created (often).

## pyYggdrasil Demo



https://mlhf.de/vorfuehrungen/
https://en.wikipedia.org/wiki/File:Flyingcircus_2.jpg

## Any Questions?

## References

- Daniel W. Meyer; “Density estimation with distribution element trees”
  - `https://doi.org/10.1007/s11222-017-9751-9`
- Daniel W. Meyer; “(Un)Conditional Sample Generation Based on Distribution Element Trees”
  - `https://doi.org/10.1080/10618600.2018.1482768`
- Wenzel Jakob and Jason Rhinelander and Dean Moldovan; “pybind11 -- Seamless operability between C++11 and Python”
  - `https://github.com/pybind/pybind11`

## Extra Slides

ETH zürich

## Parametric Model

- We assume independent dimensions
  - Linear Model

$$p\left[x_i \,\middle|\, \vec{\theta}_i^{(k)}\right] = \frac{\left(\frac{x_i - x_{i,l}^{(k)}}{x_{i,u}^{(k)} - x_{i,l}^{(k)}} - \frac{1}{2}\right)\theta_{i,1}^{(k)} + 1}{x_{i,u}^{(k)} - x_{i,l}^{(k)}}$$

  - Constant Model $$p\left[x_i \,\middle|\, \vec{\theta}_i^{(k)}\right] = \frac{1}{x_{i,u}^{(k)} - x_{i,l}^{(k)}}$$

ETH zürich

## What is “Good Enough”?

- For that a Chi2 test is performed in each dimension.
- The sub domain (only one dimension) is split into B bins.
  - The edges of the bins are determined by the quantile of the model PDF.

$$\chi^2 = \sum_{i=1}^{B} \frac{(O_i - E_i)^2}{E_i}$$

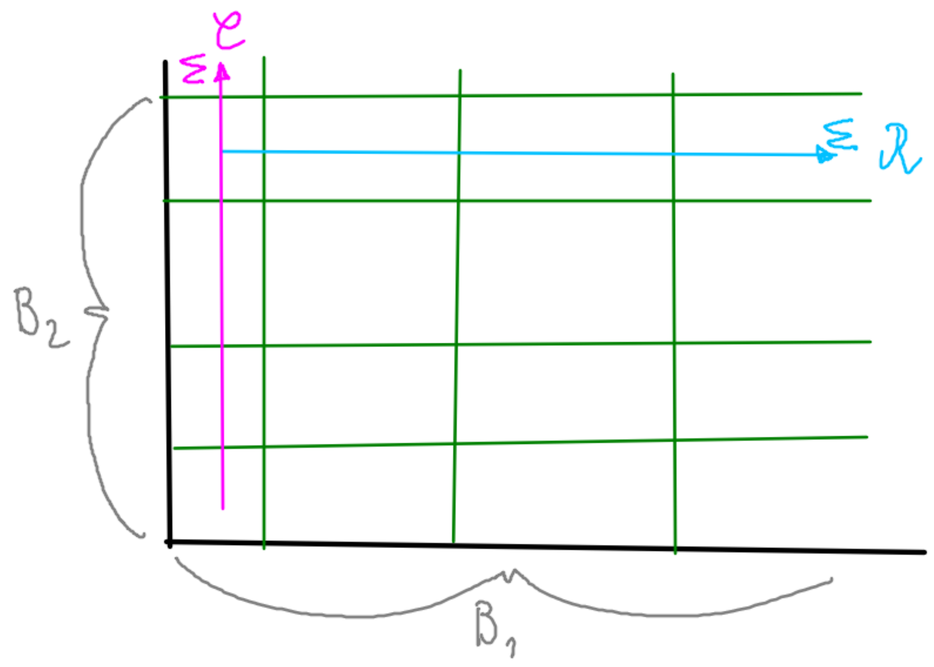
ETH zürich

## Wait didn't you forgot something?

- We have assumed that the dimensions are independent from each other.
  - We must verify this!
- We approximate mutual independence by pairwise independence.
- We test this again by a Chi2 test.

$$\chi^2 = \sum_{i=1}^{B_1} \sum_{j=1}^{B_2} \frac{(O_{i,j} - E_{i,j})^2}{E_{i,j}}$$

$$E_{i,j} = \frac{\mathcal{R}_i \cdot \mathcal{C}_j}{\mathcal{T}}$$



## Conditional Sampling

- Yggdrasil also allows the generation (conditioned) sampling that are given by the tree.

$$p(x_1,\, x_2,\, \ldots, x_q \mid x_{q+1},\, \ldots,\, x_d)$$

- Probability Mass

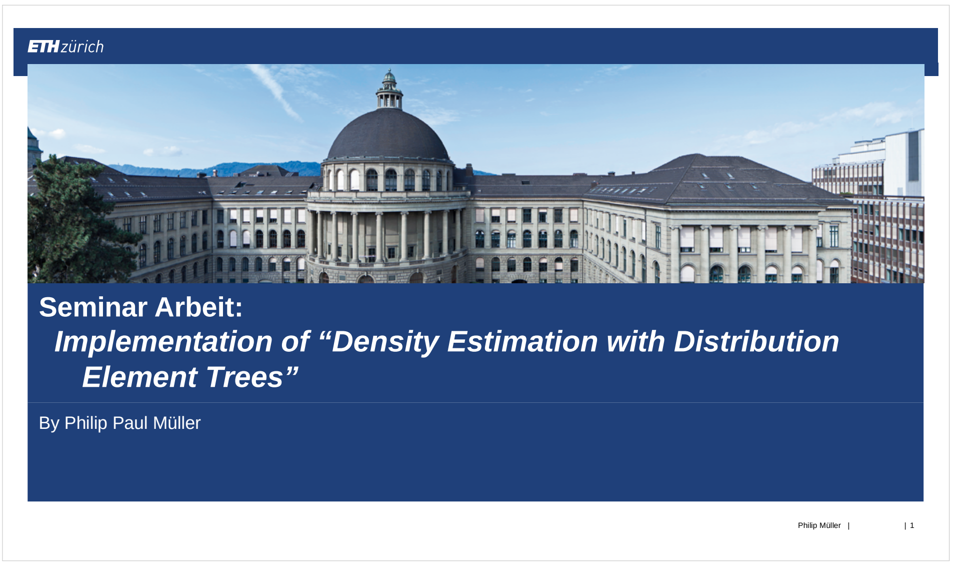
$$\widehat{\mathcal{M}}_k = \frac{n_k}{n_t} \cdot \prod_{i=q+1}^{d} p\left[x_i \,\middle|\, \vec{\theta}_i^{(k)}\right] \qquad \mathcal{M}_k = \frac{1}{p(\vec{x}^c)} \cdot \widehat{\mathcal{M}}_k$$

- Scaling Factor

$$p(\vec{x}^c) = \sum_{k=1}^{m} \widehat{\mathcal{M}}_k$$

## Difficulties

- The project was a very big.
- Numerical problems (very small values).
  - I would also suggest to implement an adaptive rescaling scheme.
- The program is designed for a lot more functionalities that where not implemented due to time constraints.
  - Task based Parallelism (openMP is probably to restrictive).
  - Incremental tree building.
  - Integration into R.



ETH zürich

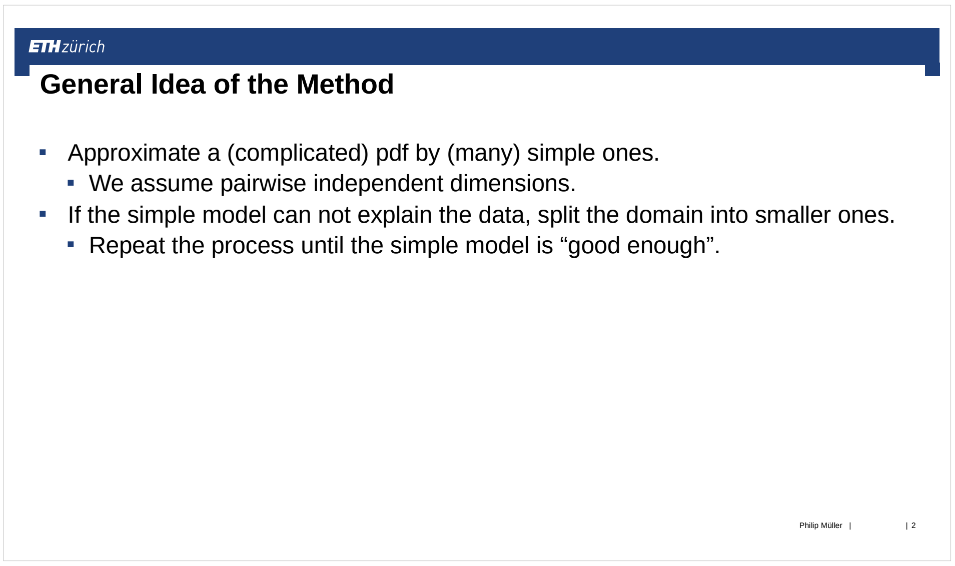
## Seminar Arbeit: *Implementation of “Density Estimation with Distribution Element Trees”*

By Philip Paul Müller

Hello I welcome you to my presentation about my Seminararbeit.

I have implemented a method that is presented in the paper “NAME”.
It was written by Daniel Meyer one of your group member.

This project was mainly about the implementation of the method, so I will shortly explain the method, since not everybody may be familiar with it.
However the main part will consist of a demo that shows how the resulted program can be used.
Although only a small part of it will be demonstrated.



ETH zürich

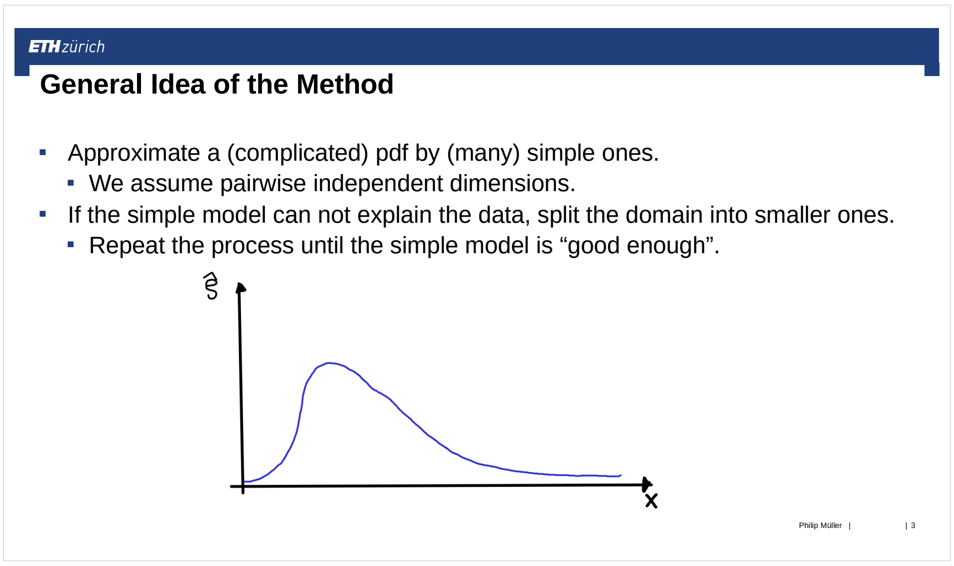
## General Idea of the Method

- Approximate a (complicated) pdf by (many) simple ones.
  - We assume pairwise independent dimensions.
- If the simple model can not explain the data, split the domain into smaller ones.
  - Repeat the process until the simple model is “good enough”.

Philip Müller | | 2

The method is quite simple.

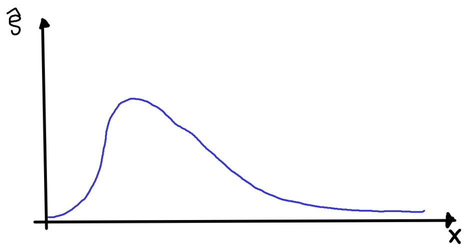
We approximate a complicated pdf by many simple ones.
We assume that we have pairwise independent dimensions, this is an approximation to mutual independence, but weaker.



ETH zürich

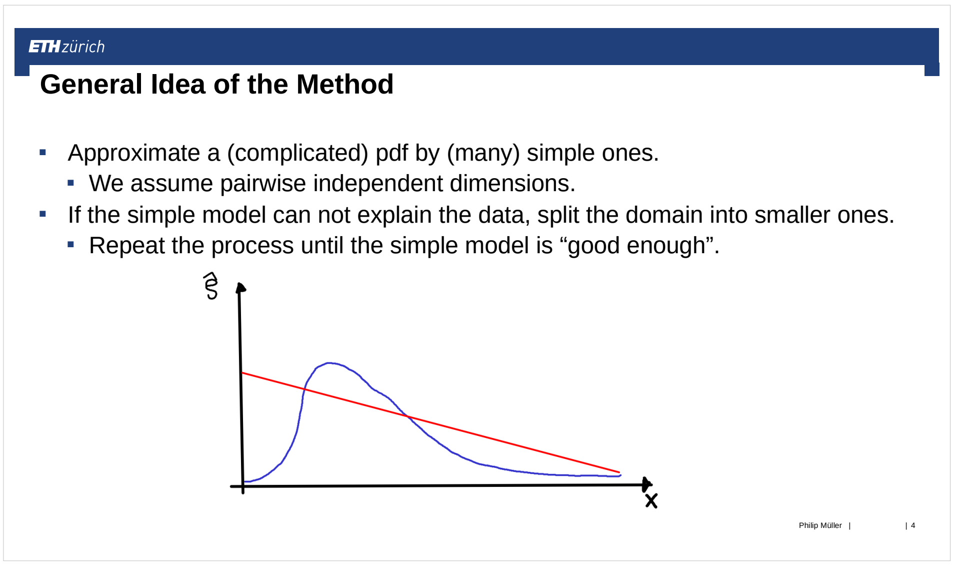
## General Idea of the Method

- Approximate a (complicated) pdf by (many) simple ones.
  - We assume pairwise independent dimensions.
- If the simple model can not explain the data, split the domain into smaller ones.
  - Repeat the process until the simple model is “good enough”.



Now we will have an example.

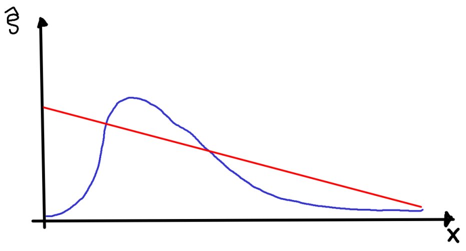
Here we have data.



ETH zürich

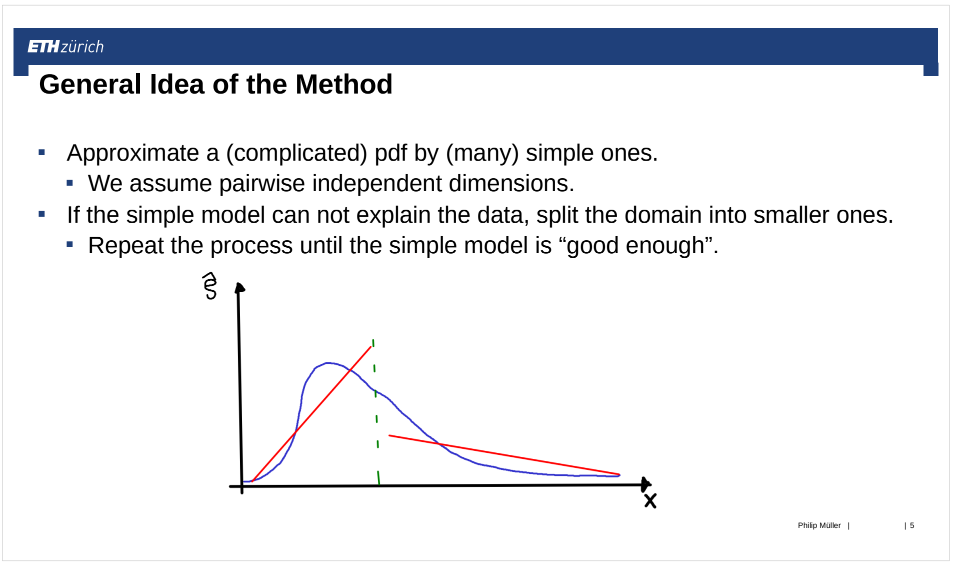
## General Idea of the Method

- Approximate a (complicated) pdf by (many) simple ones.
  - We assume pairwise independent dimensions.
- If the simple model can not explain the data, split the domain into smaller ones.
  - Repeat the process until the simple model is “good enough”.



We first fit our simple parametric model to the whole data.

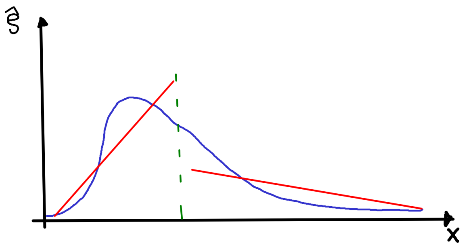
It is obvious, that the simple model, is not able to capture the structure of the distribution, when it is applied to the whole data.
So the gof test will fail and we will split the domain into two new ones.



ETHzürich

## General Idea of the Method

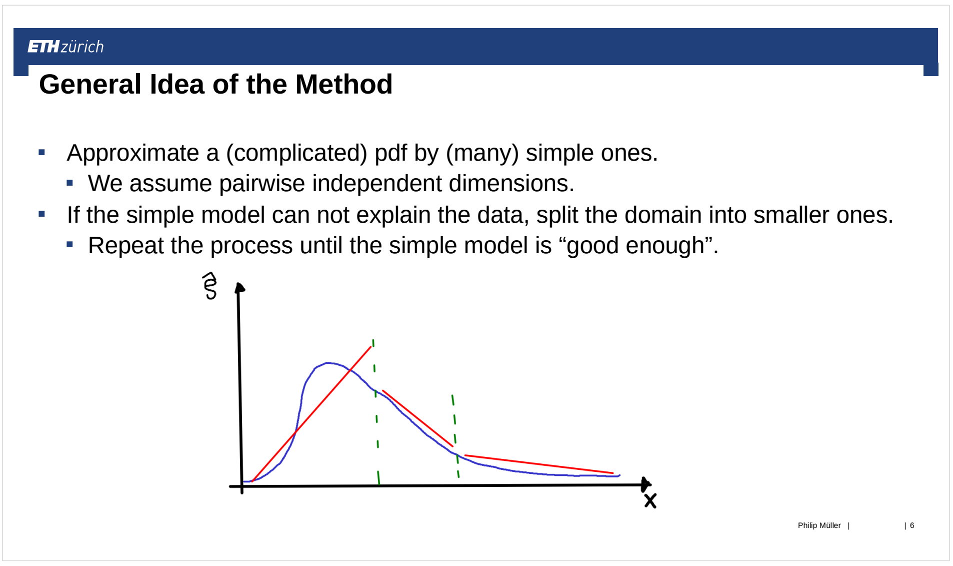
- Approximate a (complicated) pdf by (many) simple ones.
  - We assume pairwise independent dimensions.
- If the simple model can not explain the data, split the domain into smaller ones.
  - Repeat the process until the simple model is “good enough”.



We have split the domain in two new domains.
On each of the two new domains we have fitted a model again.

We see that the agreement is better than it was before, but still not very good.

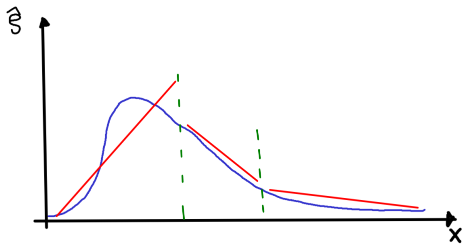
So the gof tests will again fail.



ETH zürich

## General Idea of the Method

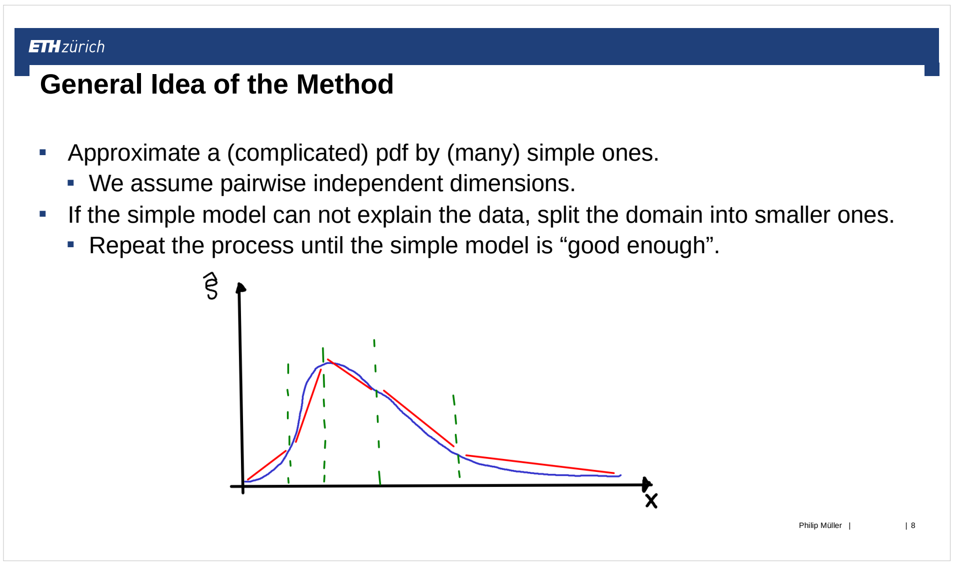
- Approximate a (complicated) pdf by (many) simple ones.
  - We assume pairwise independent dimensions.
- If the simple model can not explain the data, split the domain into smaller ones.
  - Repeat the process until the simple model is “good enough”.



We will first descend into the right sub domain.
We split it, we have three domains in total, we again fit the simeple simple model to explain the data.

We see that the agreement is much better, so the gof test will accept that both of them. This means that we are done.

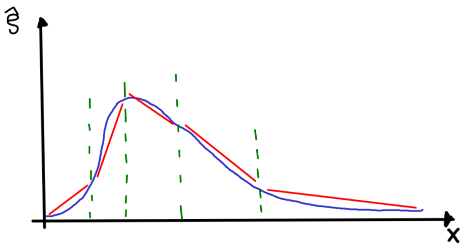
If this would be an example with more than one dimensions, then we would now also perform a pairwise independence test between any two dimensions.
If one of them fails we would split those two dimensions and repeat the process of the four resulting sub domains.



ETH zürich

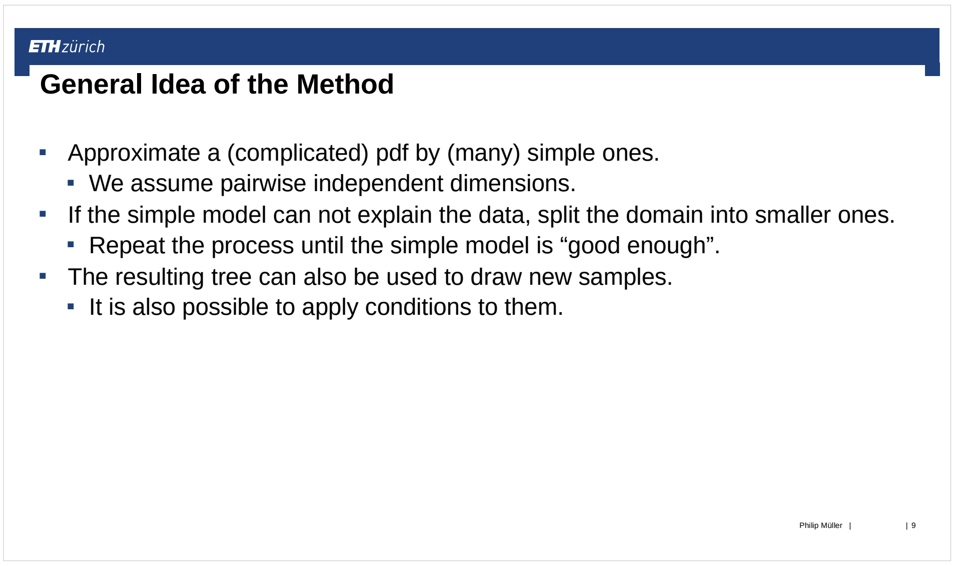
## General Idea of the Method

- Approximate a (complicated) pdf by (many) simple ones.
  - We assume pairwise independent dimensions.
- If the simple model can not explain the data, split the domain into smaller ones.
  - Repeat the process until the simple model is “good enough”.



We new perform the fitting process also on the left sub domain, that resulted from the very first split.
As we see it was split again but was then accepted and fits the curve well.

Now the process is done and we have generated an distributed element tree.



ETH zürich

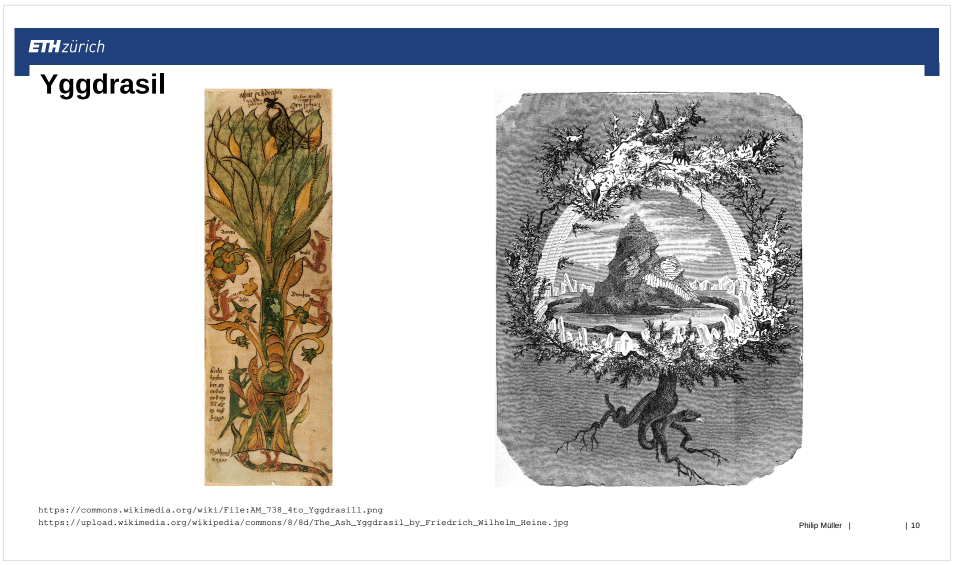
## General Idea of the Method

- Approximate a (complicated) pdf by (many) simple ones.
  - We assume pairwise independent dimensions.
- If the simple model can not explain the data, split the domain into smaller ones.
  - Repeat the process until the simple model is “good enough”.
- The resulting tree can also be used to draw new samples.
  - It is also possible to apply conditions to them.

Philip Müller | | 9

There is a second paper, also authored by Daniel Meyer.
The paper present a way to turn the tree into a generator that allows the generation of samples, which distribution is given by the tree.

The nice thing about this generator is, that it is very easy to apply a condition to the samples, that should be generated.



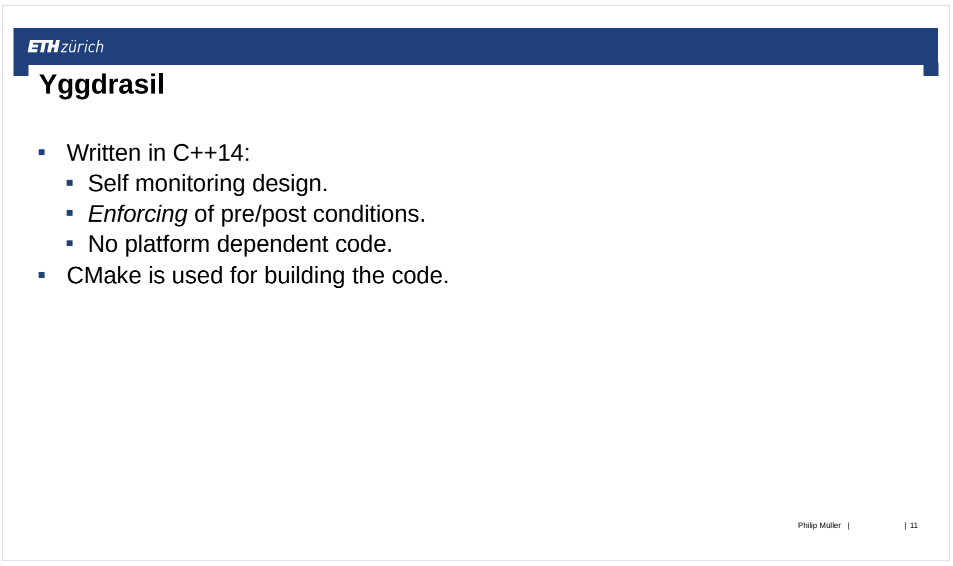
ETH zürich

## Yggdrasil

https://commons.wikimedia.org/wiki/File:AM_738_4to_Yggdrasill.png
https://upload.wikimedia.org/wikipedia/commons/8/8d/The_Ash_Yggdrasil_by_Friedrich_Wilhelm_Heine.jpg

The method is already implemented in Matlab and R.
My job was to implement it in C++ and to provide a way to use it from Python.

I named the library Yggdrasil. Yggdrasil is a tree from the Nordic mythology. It was the first tree in the entire universe and thus the biggest. Its branches extends to every location of the universe.
So I thought that the name was appropriate for such a project.


ETH zürich

## Yggdrasil

- Written in C++14:
  - Self monitoring design.
  - *Enforcing* of pre/post conditions.
  - No platform dependent code.
- CMake is used for building the code.

Philip Müller | | 11

Yggdrasil is written in C++14, actually it supports C++11, but you should not use it, since C++14 is considered the bug fix for C++11.

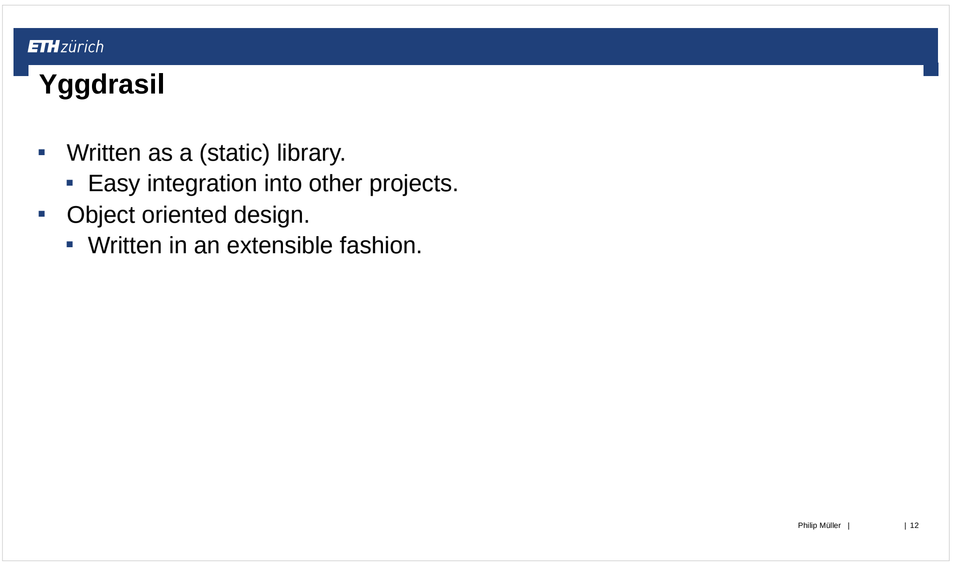
I want to say at that point, that I am not a feature fan boy.
I do not use a language/library feature, just for the sake of using them.
You will not see vardiac templates and mind blowing meta programming magic inside the code nor extensive use of lambda functions.

But where the usage of a feature was appropriate I used it.

The code is designed to be self monitoring. This means that pre- and postconditions are checked and invariants too.
Important conditions are enforced by using exceptions, other checks that are primary needed for debugging are done with asserts.

I have not used platform dependent functionality. Instead (platfrom independent) libraries were used.

The build process is done by CMake, this allows building on different systems.


ETH zürich

## Yggdrasil

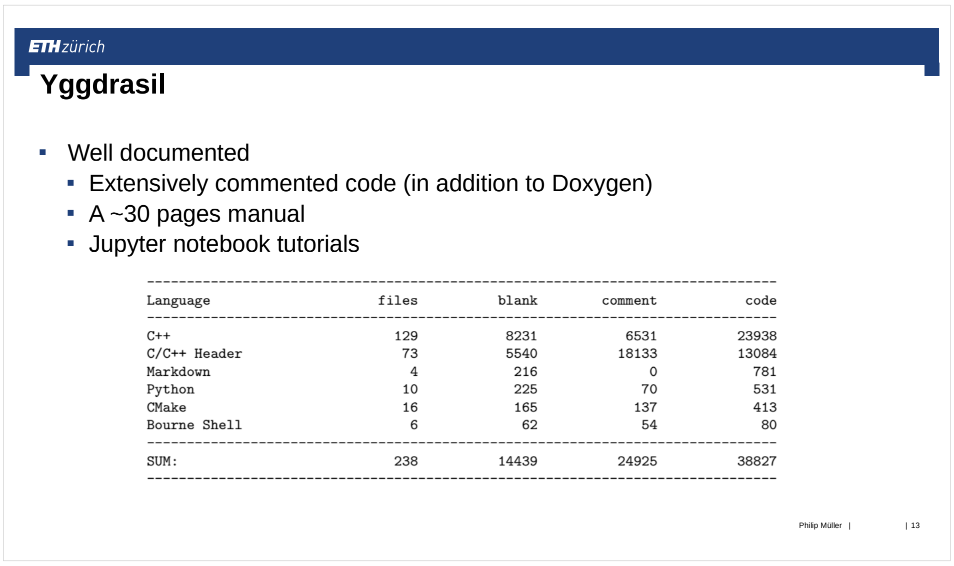
- Written as a (static) library.
  - Easy integration into other projects.
- Object oriented design.
  - Written in an extensible fashion.

Philip Müller | | 12

I created a library that implements the tree-estimator, the generator and all the auxiliary code as a normal library.
All other aspects of the project like the unit tests, the validation programs or even the Python interface, is then simply including and linking against Yggdrasil.
The library design of Yggdrasil allows its usage also in other programs.

Yggdrasil is written in an object oriented way.
First of all, Yggdrasil provides a class for every concept. You do not need to create an array and fill it in an obscure nonsensical order. Also you can not mix up arguments, since they have different types, so the compiler will complain about it.
For example Yggdrasil has its own class for representing intervals. This means that the entire functionality that deals with intervals is at one central location and is thus consistently handled/changed. It also prevents the root of all evil, copy-past.

Where appropriate Yggdrasil uses dynamic dispatch, to achieve runtime polymorphism. For example the parametric models and the tests are implemented that way. This makes extending the functionality and scope of Yggdrasil very easy, since the core of the process must not be modified. And by providing a general builder concept, the integration of the new behavior is possible without much problem.



ETH zürich

## Yggdrasil

- Well documented
  - Extensively commented code (in addition to Doxygen)
  - A ~30 pages manual
  - Jupyter notebook tutorials

```
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
C++                            129           8231           6531          23938
C/C++ Header                    73           5540          18133          13084
Markdown                         4            216              0            781
Python                          10            225             70            531
CMake                           16            165            137            413
Bourne Shell                     6             62             54             80
-------------------------------------------------------------------------------
SUM:                           238          14439          24925          38827
-------------------------------------------------------------------------------
```

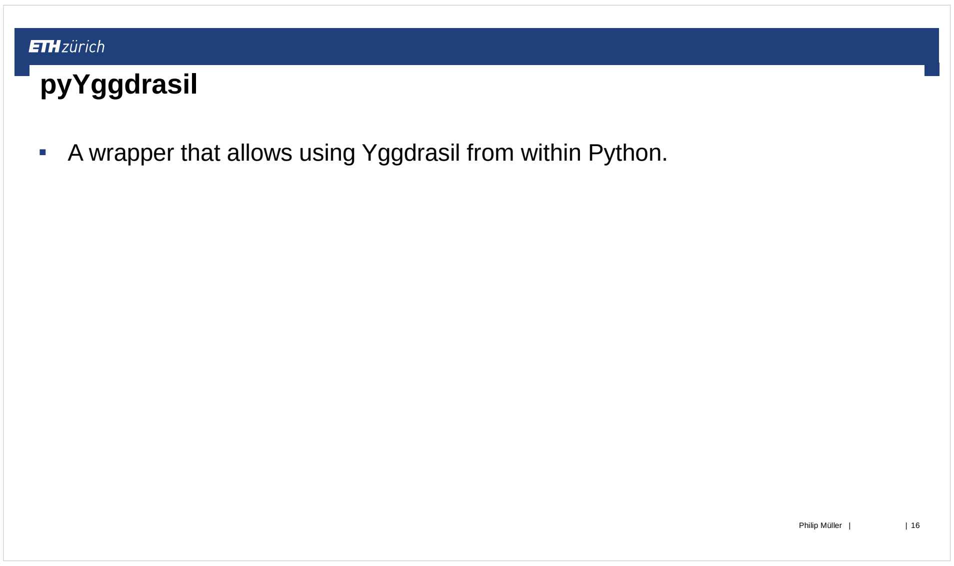
Philip Müller | | 13

It is well documented.

First of all the code base itself is heavily commented. Every class and function is commented with Doxygen, all arguments are described.
I am no Doxygen expert, so do not expect something like the Eigen documentation, but it will give you a complete overview over the functionality that is implemented.

The manual is not a simple listing of all the functions that are provided. This is the realm of Dxygen or Python Docstrings.
It is a document which gives you an idea of the high level structure of the code, explains the meaning of the important classes and their intentions.

Then also a long tutorial of Jupyter Notebooks is provided. It shows how the library can be used and gives also some hints, why one should do it a certain way.


ETH zürich

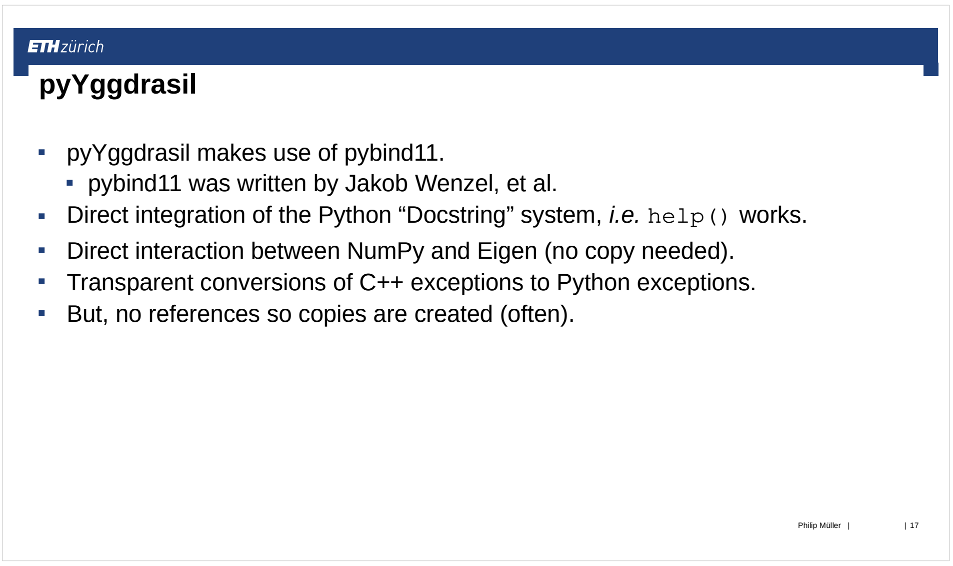
## pyYggdrasil

- A wrapper that allows using Yggdrasil from within Python.

Philip Müller | | 16

PyYggdrasil, brings Yggdrasil to Python.

As I have said before, pyYggdrasil is implemented by using Yggdrasil's interface.
It is a wrapper around the C++ interface, that allows the interacting of Yggdrasil with Python.


ETH zürich

## pyYggdrasil

- pyYggdrasil makes use of pybind11.
  - pybind11 was written by Jakob Wenzel, et al.
- Direct integration of the Python “Docstring” system, *i.e.* `help()` works.
- Direct interaction between NumPy and Eigen (no copy needed).
- Transparent conversions of C++ exceptions to Python exceptions.
- But, no references so copies are created (often).

Philip Müller | | 17

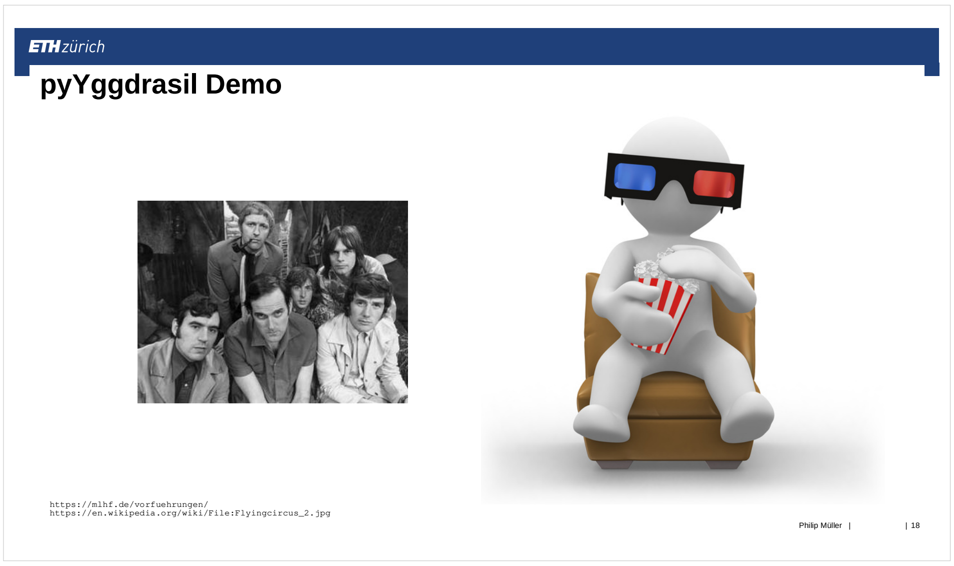
PyYggdrasil in its current form would hardly be possible without pybind11.
Pybind11 is a library that was developed by Wenzel Jakob and others, who is a professor at EPFL for computer graphic.
It uses very fancy and terrifying meta programming magic, to achieve the interaction, which is as the project, rightfully, claims seamless.

This is more advertising for pybind11 than anything else, all features I list are implemented by pybind11, I just wanted to list then to show, what pyYggdrasil can do.

It allows to use the Docstring system of Python.
pyYggdrasil provides a description for every class, module and function, that can easy be accessed by writing help().

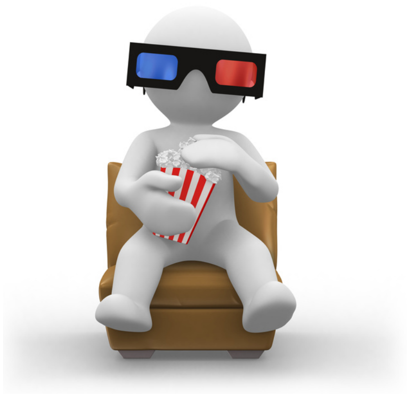
It also allows that NumPy objects can be used as Eigen objects, and this functionality is exploited.

A downside of pyYggdrasil is, that sometimes copies are created.
The main sources of copies is that Python does not have references as C++ has them. So one would need to write a proxy class for each class that models such a reference, this was not done, since it would also require larger modification to achieve consistency.
Some care must be taken when writing Python code to minimize costly copy operations.
It could be worth to create a dedicated C++ function for a task and integrate it into pyYggdrasil. There is a reason why on the C++ layer you have three different containers for storing the samples.



ETH zürich

## pyYggdrasil Demo



https://mlhf.de/vorfuehrungen/
https://en.wikipedia.org/wiki/File:Flyingcircus_2.jpg

Now we comes to a life demo.
I must say that the code does not run on my laptop, since it is relatively old, instead the demonstration is done on my Desktop at home. There a i7 7700K is running, so a normal consumer PC.
openMP, which is supported in a very basic form, is disabled.

The demo is also not supper clean, this is because it has to fit into a very tight time frame.

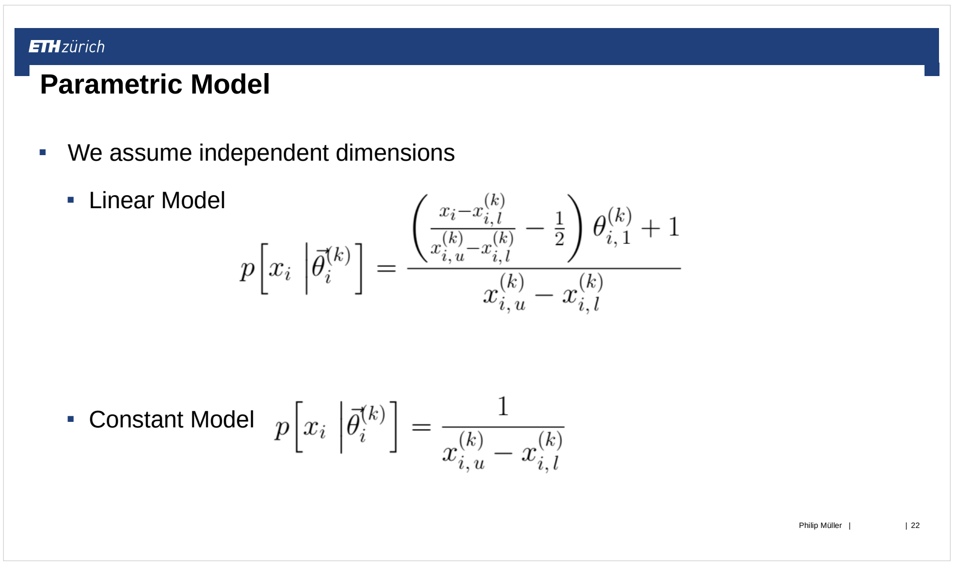
ETH zürich

## Any Questions?

## References

- Daniel W. Meyer; “Density estimation with distribution element trees”
  - `https://doi.org/10.1007/s11222-017-9751-9`
- Daniel W. Meyer; “(Un)Conditional Sample Generation Based on Distribution Element Trees”
  - `https://doi.org/10.1080/10618600.2018.1482768`
- Wenzel Jakob and Jason Rhinelander and Dean Moldovan; “pybind11 -- Seamless operability between C++11 and Python”
  - `https://github.com/pybind/pybind11`

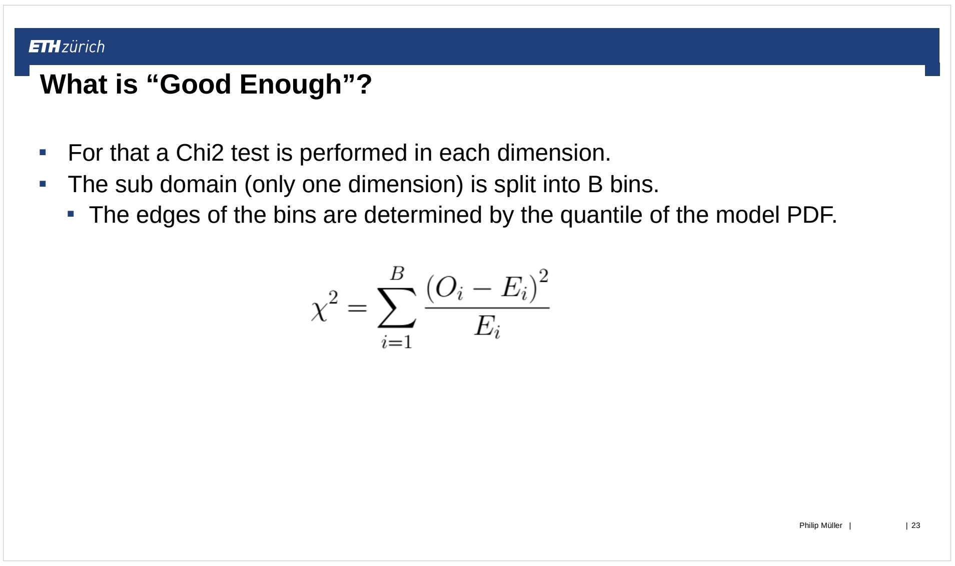
## Extra Slides


ETH zürich

## Parametric Model

- We assume independent dimensions
  - Linear Model
$$p\left[x_i \,\middle|\, \vec{\theta}_i^{(k)}\right] = \frac{\left(\frac{x_i - x_{i,l}^{(k)}}{x_{i,u}^{(k)} - x_{i,l}^{(k)}} - \frac{1}{2}\right)\theta_{i,1}^{(k)} + 1}{x_{i,u}^{(k)} - x_{i,l}^{(k)}}$$
  - Constant Model
$$p\left[x_i \,\middle|\, \vec{\theta}_i^{(k)}\right] = \frac{1}{x_{i,u}^{(k)} - x_{i,l}^{(k)}}$$

An even simpler model is the constant model.


ETH zürich

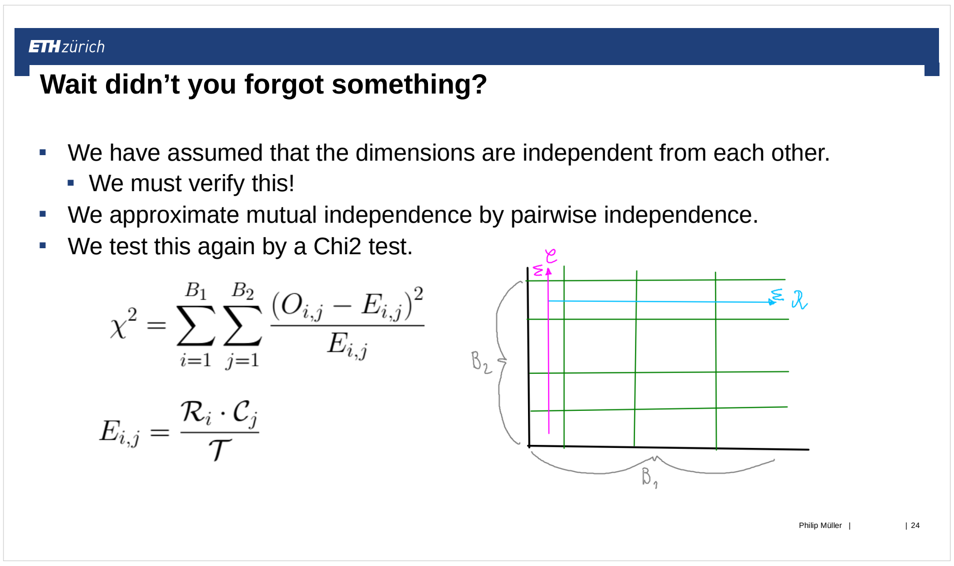
## What is “Good Enough”?

- For that a Chi2 test is performed in each dimension.
- The sub domain (only one dimension) is split into B bins.
  - The edges of the bins are determined by the quantile of the model PDF.

$$\chi^2 = \sum_{i=1}^{B} \frac{(O_i - E_i)^2}{E_i}$$

The edges are determined by the quantile of the pdf function that is given by the parametric model.

If a test was rejected we will perform a split and repeat the procedure at the two newly created leafs.



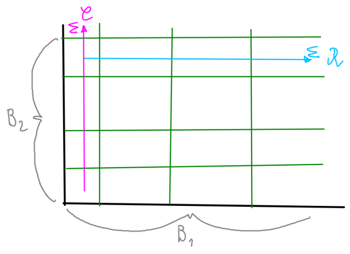
ETH zürich

## Wait didn't you forgot something?

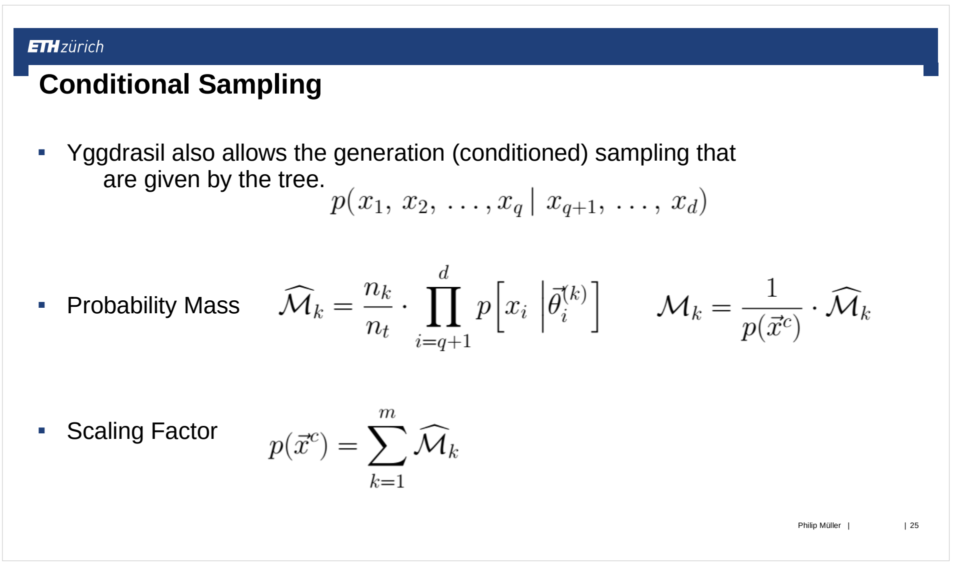
- We have assumed that the dimensions are independent from each other.
  - We must verify this!
- We approximate mutual independence by pairwise independence.
- We test this again by a Chi2 test.

$$\chi^2 = \sum_{i=1}^{B_1} \sum_{j=1}^{B_2} \frac{(O_{i,j} - E_{i,j})^2}{E_{i,j}}$$

$$E_{i,j} = \frac{\mathcal{R}_i \cdot \mathcal{C}_j}{\mathcal{T}}$$



Again we use a Chi2 Test for this.


ETH zürich

## Conditional Sampling

- Yggdrasil also allows the generation (conditioned) sampling that are given by the tree. $p(x_1,\, x_2,\, \ldots, x_q \mid x_{q+1},\, \ldots,\, x_d)$
- Probability Mass $\widehat{\mathcal{M}}_k = \frac{n_k}{n_t} \cdot \prod_{i=q+1}^{d} p\left[x_i \,\middle|\vec{\theta}_i^{(k)}\right]$ $\mathcal{M}_k = \frac{1}{p(\vec{x}^c)} \cdot \widehat{\mathcal{M}}_k$
- Scaling Factor $p(\vec{x}^c) = \sum_{k=1}^{m} \widehat{\mathcal{M}}_k$

Philip Müller | | 25

p(x^c) is the scaling factor of the mass.
It is actually defined by a complicated integral.
After some work it comes out that it is the normalization constant of the masses, so it can be estimated by just the sum of all the masses.

The process of generation of samples is quite easy.

In a preprocessing step the following is done:
> In a first step all domains are checked if they fulfill the constraints.
> Then their mass is calculated

When a sample is generated we do the following:
> A sub domain is selected. The probability of selecting one is proportional to its mass.
> Inside the subdomain a sample is generated by means of the inversion method.

ETH zürich

## Difficulties

- The project was a very big.
- Numerical problems (very small values).
  - I would also suggest to implement an adaptive rescaling scheme.
- The program is designed for a lot more functionalities that where not implemented due to time constraints.
  - Task based Parallelism (openMP is probably to restrictive).
  - Incremental tree building.
  - Integration into R.