scalable pythonic fitting

Jonas Eschle on behalf of zfit
jonas.eschle@cern.ch
HEP Model Fitting in Python

- **Scalable**: large data, complex models
- **Pythonic**: use Python ecosystem/language
- **HEP specific functionality**
Fitting in Python

A lot of projects are around!

- RooFit
- HEP Python fitting projects
- Non-HEP
A lot of projects are around!

- RooFit
- HEP Python fitting projects
- Non-HEP

No feasible Python model fitting library for HEP
A lot of projects are around!

- RooFit
- HEP Python fitting projects
- Non-HEP

No feasible Python model fitting library for HEP

… but a lot to learn and build from!
build *the* stable model fitting ecosystem for HEP

...the time has come
build the stable model fitting ecosystem for HEP
...the time has come

- Functionality limited to model fitting & sampling
build *the* stable model fitting ecosystem for HEP
...the time has come

- Functionality limited to model fitting & sampling
- Use power & knowledge of *existing libraries*
zfit: the project

build *the* stable model fitting ecosystem for HEP

...the time has come

- Functionality limited to model fitting & sampling

- Use power & knowledge of existing libraries

- Build fresh from scratch
build the stable model fitting ecosystem for HEP
...the time has come

- Functionality limited to model fitting & sampling
- Use power & knowledge of existing libraries
- Build fresh from scratch
- Community invocation
build *the* stable model fitting ecosystem for HEP

...the time has come

- Functionality limited to model fitting & sampling
- Use power & knowledge of existing libraries
- Build fresh from scratch
- Community invocation
API & workflow definition

Computational backend

(reference) implementation
API & Workflow
• High level libraries (statistics packages, amplitude fitters,...)
  - "code against an interface, not an implementation"
API & Workflow: why

- High level libraries (statistics packages, amplitude fitters,...)
  - „code against an interface, not an implementation“
- Replace each component
  - Allow other libraries to implement custom parts
  - Provide reference implementation for all parts
API & Workflow: why

• High level libraries (statistics packages, amplitude fitters,...)
  - „code against an interface, not an implementation“
• Replace each component
  - Allow other libraries to implement custom parts
  - Provide reference implementation for all parts

Allows ecosystem to grow "by itself"
Workflow

Five maximally independent parts

Well defined API implemented as interfaces
Workflow

Five maximally independent parts

Well defined API implemented as interfaces

Example: Library as "loss builder"

Your function here
```
obs = zfit.Space("x", limits=(-2, 3))
mu = zfit.Parameter("mu", 1.2, -4, 6)
sigma = zfit.Parameter("sigma", 1.3, 0.1, 10)
gauss = zfit.pdf.Gauss(mu=mu, sigma=sigma, obs=obs)
data = zfit.Data.from_numpy(obs=obs, array=normal_np)
nll = zfit.loss.UnbinnedNLL(model=gauss, data=data)
minimizer = zfit.minimize.Minuit()
result = minimizer.minimize(nll)
param_errors = result.error()
```
Workflow
Computational Backend
Computational Backend
“And now for something completely different.”

Monty Python
Computational Backend

*(very brief) introduction to*

Deep Learning

or Neural Networks
or Machine Learning
or Big Data…
Deep Learning

Neural Network

„One huge, complicated function“
## Deep Learning vs. Model Fitting

<table>
<thead>
<tr>
<th>Similarity</th>
<th>Complicated Models</th>
<th>Large Data</th>
<th>Composed loss</th>
<th>Minimization</th>
<th>Results and uncertainties</th>
</tr>
</thead>
<tbody>
<tr>
<td>HEP</td>
<td>Non-trivial functions</td>
<td>Whole Dataset</td>
<td>simultaneous, constraints</td>
<td>Global min, 2(^{nd}) derivative algorithm</td>
<td>Hesse, profiling</td>
</tr>
<tr>
<td>Deep Learning</td>
<td>Combine many, trivial functions</td>
<td>Many, small Batches</td>
<td><em>Anything!</em> (<em>GANs, RL,...</em>)</td>
<td>Local (!) min, 1(^{th}) derivative, many steps</td>
<td>None</td>
</tr>
<tr>
<td>Conclusion</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Similarity</td>
<td>Complicated Models</td>
<td>Large Data</td>
<td>Composed loss</td>
<td>Minimization</td>
<td>Results and uncertainties</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------</td>
<td>------------</td>
<td>---------------</td>
<td>--------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>HEP</td>
<td>Non-trivial functions</td>
<td>Whole Dataset</td>
<td>simultaneous, constraints</td>
<td>Global min, 2\textsuperscript{nd} derivative algorithm</td>
<td>Hesse, profiling</td>
</tr>
<tr>
<td>Deep Learning</td>
<td>Combine many, trivial functions</td>
<td>Many, small Batches</td>
<td>Anything! (GANs, RL,...)</td>
<td>Local (!) min, 1\textsuperscript{st} derivative, many steps</td>
<td>None</td>
</tr>
<tr>
<td>Conclusion</td>
<td>No real impact</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Deep Learning vs. Model Fitting

<table>
<thead>
<tr>
<th>Similarity</th>
<th>Complicated Models</th>
<th>Large Data</th>
<th>Composed loss</th>
<th>Minimization</th>
<th>Results and uncertainties</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>HEP</strong></td>
<td>Non-trivial functions</td>
<td>Whole Dataset</td>
<td>simultaneous, constraints</td>
<td>Global min, 2\text{nd} derivative algorithm</td>
<td>Hesse, profiling</td>
</tr>
<tr>
<td><strong>Deep Learning</strong></td>
<td>Combine many, trivial functions</td>
<td>Many, small Batches</td>
<td>Anything! (GANs, RL,...)</td>
<td>Local (!) min, 1\text{st} derivative, many steps</td>
<td>None</td>
</tr>
<tr>
<td><strong>Conclusion</strong></td>
<td>No real impact</td>
<td>Optimizations for OOC calculations</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*HEP: High Energy Physics, GANs: Generative Adversarial Networks, RL: Reinforcement Learning*
# Deep Learning vs. Model Fitting

<table>
<thead>
<tr>
<th>Similarity</th>
<th>Complicated Models</th>
<th>Large Data</th>
<th>Composed loss</th>
<th>Minimization</th>
<th>Results and uncertainties</th>
</tr>
</thead>
<tbody>
<tr>
<td>HEP</td>
<td>Non-trivial functions</td>
<td>Whole Dataset</td>
<td>simultaneous, constraints</td>
<td>Global min, 2\textsuperscript{nd} derivative algorithm</td>
<td>Hesse, profiling</td>
</tr>
<tr>
<td>Deep Learning</td>
<td>Combine many, trivial functions</td>
<td>Many, small Batches</td>
<td>\textit{Anything!} (GANs, RL,...)</td>
<td>Local (!) min, 1\textsuperscript{st} derivative, many steps</td>
<td>None</td>
</tr>
<tr>
<td>Conclusion</td>
<td>No real impact</td>
<td>Optimizations for OOC calculations</td>
<td>HEP trivial special case</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Deep Learning vs. Model Fitting

<table>
<thead>
<tr>
<th>Similarity</th>
<th>Complicated Models</th>
<th>Large Data</th>
<th>Composed loss</th>
<th>Minimization</th>
<th>Results and uncertainties</th>
</tr>
</thead>
<tbody>
<tr>
<td>HEP</td>
<td>Non-trivial functions</td>
<td>Whole Dataset</td>
<td>simultaneous, constraints</td>
<td>Global min, $2^{nd}$ derivative algorithm</td>
<td>Hesse, profiling</td>
</tr>
<tr>
<td>Deep Learning</td>
<td>Combine many, trivial functions</td>
<td>Many, small Batches</td>
<td><em>Anything!</em> ($GANs$, $RL,...$)</td>
<td>Local (!) min, $1^{th}$ derivative, many steps</td>
<td>None</td>
</tr>
<tr>
<td>Conclusion</td>
<td>No real impact</td>
<td>Optimizations for OOC calculations</td>
<td>HEP trivial special case</td>
<td>Optimizers „analytic“ derivatives!</td>
<td></td>
</tr>
</tbody>
</table>
# Deep Learning vs. Model Fitting

<table>
<thead>
<tr>
<th>Similarity</th>
<th>Complicated Models</th>
<th>Large Data</th>
<th>Composed loss</th>
<th>Minimization</th>
<th>Results and uncertainties</th>
</tr>
</thead>
<tbody>
<tr>
<td>HEP</td>
<td>Non-trivial functions</td>
<td>Whole Dataset</td>
<td>simultaneous, constraints</td>
<td>Global min, 2\textsuperscript{nd} derivative algorithm</td>
<td>Hesse, profiling</td>
</tr>
<tr>
<td>Deep Learning</td>
<td>Combine many, trivial functions</td>
<td>Many, small Batches</td>
<td>Anything! (GANs, RL,...)</td>
<td>Local (!) min, 1\textsuperscript{th} derivative, many steps</td>
<td>None</td>
</tr>
<tr>
<td>Conclusion</td>
<td>No real impact</td>
<td>Optimizations for OOC calculations</td>
<td>HEP trivial special case</td>
<td>Optimizers „analytic“ derivatives!</td>
<td>No support, but simple</td>
</tr>
</tbody>
</table>
# Deep Learning vs. Model Fitting

<table>
<thead>
<tr>
<th>Similarity</th>
<th>Complicated Models</th>
<th>Large Data</th>
<th>Composed loss</th>
<th>Minimization</th>
<th>Results and uncertainties</th>
</tr>
</thead>
<tbody>
<tr>
<td>HEP</td>
<td>Non-trivial functions</td>
<td>Whole Dataset</td>
<td>simultaneous, constraints</td>
<td>Global min, 2\textsuperscript{nd} derivative algorithm</td>
<td>Hesse, profiling</td>
</tr>
<tr>
<td>Deep Learning</td>
<td>Combine many, trivial functions</td>
<td>Many, small Batches</td>
<td>\textit{Anything! (GANs, RL,…)}</td>
<td>Local (!) min, 1\textsuperscript{st} derivative, many steps</td>
<td>None</td>
</tr>
<tr>
<td>Conclusion</td>
<td>No real impact</td>
<td>Optimizations for OOC calculations</td>
<td>HEP trivial special case</td>
<td>Optimizers „analytic“ derivatives!</td>
<td>No support, but simple</td>
</tr>
</tbody>
</table>
## Deep Learning vs. Model Fitting

<table>
<thead>
<tr>
<th>Similarity</th>
<th>Complicated Models</th>
<th>Large Data</th>
<th>Composed loss</th>
<th>Minimization</th>
<th>Results and uncertainties</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>HEP</strong></td>
<td>Non-trivial functions</td>
<td>Whole Dataset</td>
<td>simultaneous, constraints</td>
<td>Global min, 2\textsuperscript{nd} derivative</td>
<td>Hesse, profiling</td>
</tr>
<tr>
<td><strong>Deep Learning</strong></td>
<td>Combine many, trivial functions</td>
<td>Many, small Batches</td>
<td><em>Anything!</em> (GANs, RL,...)</td>
<td>Local (!) min, 1\textsuperscript{st} derivative, many steps</td>
<td>None</td>
</tr>
<tr>
<td><strong>Conclusion</strong></td>
<td>No real impact</td>
<td>Optimizations for OOM calculations</td>
<td>HEP trivial special case</td>
<td>Optimizers Free „analytic“ derivatives!</td>
<td>No support, but simple</td>
</tr>
</tbody>
</table>

**But… what *is* a Deep Learning library?**
# Deep Learning vs. Model Fitting

<table>
<thead>
<tr>
<th>Similarity</th>
<th>Complicated Models</th>
<th>Large Data</th>
<th>Composed loss</th>
<th>Minimization</th>
<th>Results and uncertainties</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>HEP</strong></td>
<td>Non-trivial functions</td>
<td>Whole Dataset</td>
<td>simultaneous, constraints</td>
<td>Global min, 2(^{nd}) derivative algorithm</td>
<td>Hesse, profiling</td>
</tr>
<tr>
<td><strong>Deep Learning</strong></td>
<td>Combine many, trivial functions</td>
<td>Many, small Batches</td>
<td>Anything! (GANs, RL,...)</td>
<td>Local (!) min, 1(^{st}) derivative, many steps</td>
<td>None</td>
</tr>
<tr>
<td><strong>Conclusion</strong></td>
<td>No real impact</td>
<td>Optimizations for OOM calculations</td>
<td>HEP trivial special case</td>
<td>Optimizers „analytic“ derivatives!</td>
<td>No support, but simple</td>
</tr>
</tbody>
</table>
### Deep Learning vs. Model Fitting

<table>
<thead>
<tr>
<th>Similarity</th>
<th>Complicated Models</th>
<th>Large Data</th>
<th>Composed loss</th>
<th>Minimization</th>
<th>Results and uncertainties</th>
</tr>
</thead>
<tbody>
<tr>
<td>HEP</td>
<td>Non-trivial functions</td>
<td>Whole Dataset</td>
<td>simultaneous, constraints</td>
<td>Global min, 2nd derivative algorithm</td>
<td>Hesse, profiling</td>
</tr>
<tr>
<td>Deep Learning</td>
<td>Combine many, trivial functions</td>
<td>Many, small Batches</td>
<td>Anything! (GANs, RL,...)</td>
<td>Local (!) min, 1th derivative, many steps</td>
<td>None</td>
</tr>
<tr>
<td>Conclusion</td>
<td>No real impact</td>
<td>Optimizations for OOM calculations</td>
<td>HEP trivial special case</td>
<td>Optimizers „analytic“ derivatives!</td>
<td>No support, but simple</td>
</tr>
</tbody>
</table>

**Modern, high performance computing**
TensorFlow

- By Google, highly popular (130k ★, 4th on 🌟)
- Used in multiple physics libraries and analyses
TensorFlow

- By Google, highly popular (130k★, 4th on 🌟)
- Used in multiple physics libraries and analyses
- Consists of "two parts":
  - High level API for building neural networks *(NOT used!)*
  - Low level API with Numpy-style syntax
    tf.sqrt, tf.random.uniform,...
TensorFlow

- By Google, highly popular (130k⭐️, 4th on 🌟)
- Used in multiple physics libraries and analyses
- Consists of "two parts":
  - High level API for building neural networks *(NOT used!)*
  - **Low level API** with Numpy-style syntax
    
    \[
    \text{tf.sqrt, tf.random.uniform,}\ldots
    \]

...but many Deep Learning frameworks are similar
Advantages

- Autograd: automatic gradient calculation
- Native CPU/GPU/distributed support
- Optimizations (graphs,...)
Advantages

- Autograd: automatic gradient calculation
- Native CPU/GPU/distributed support
- Optimizations (graphs, …)

Used (+ maintained!) by industry where performance is money

huge financial interest
Delegating the workload

<table>
<thead>
<tr>
<th></th>
<th>C++ library (RooFit,...)</th>
<th>Numpy based</th>
<th>zfit</th>
</tr>
</thead>
<tbody>
<tr>
<td>HEP specific content/API</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Models</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gradients</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Computational optimizations</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parallelization/GPU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Low level handling</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- SciPy
- TensorFlow Probability
- Numba
- NumPy
- Python
- TensorFlow
- NVIDIA
- Intel

4. Nov 2019
CHEP 2019 Adelaide
Delegating the workload

<table>
<thead>
<tr>
<th></th>
<th>C++ library (RooFit,...)</th>
<th>Numpy based</th>
</tr>
</thead>
<tbody>
<tr>
<td>HEP specific content/API</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Models</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gradients</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Computational optimizations</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parallelization/GPU</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Low level handling</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

"Stepping on the shoulders of a giant"
## Delegating the workload

<table>
<thead>
<tr>
<th>C++ library (RooFit,...)</th>
<th>Numpy based</th>
</tr>
</thead>
<tbody>
<tr>
<td>HEP specific content/API</td>
<td></td>
</tr>
<tr>
<td>Models</td>
<td></td>
</tr>
<tr>
<td>Gradients</td>
<td></td>
</tr>
<tr>
<td>Computational optimizations</td>
<td></td>
</tr>
<tr>
<td>Parallelization/GPU</td>
<td></td>
</tr>
<tr>
<td>Low level handling</td>
<td></td>
</tr>
</tbody>
</table>

"Stepping on the shoulders of a giant"
Performance

Sum of 9 Gaussians, toy fitting time, 6 core CPU: **RooFit vs. zfit**

- **RooFit** has 9 free parameters
- **zfit** has 2 free parameters

Same order of magnitude as RooFit

Time (sec) vs. Number of events

- **x5-8**
Implementation
Complete fit

```python
normal_np = np.random.normal(loc=2., scale=3., size=10000)
obs = zfit.Space("x", limits=(-2, 3))
mu = zfit.Parameter("mu", 1.2, -4, 6)
sigma = zfit.Parameter("sigma", 1.3, 0.1, 10)
gauss = zfit.pdf.Gauss(mu=mu, sigma=sigma, obs=obs)
data = zfit.Data.from_numpy(obs=obs, array=normal_np)
nll = zfit.loss.UnbinnedNLL(model=gauss, data=data)
minimizer = zfit.minimize.Minuit()
result = minimizer.minimize(nll)
param_errors = result.error()
```
normal_np = np.random.normal(loc=2., scale=3., size=10000)

obs = zfit.Space("x", limits=(-2, 3))

mu = zfit.Parameter("mu", 1.2, -4, 6)
sigma = zfit.Parameter("sigma", 1.3, 0.1, 10)
gauss = zfit.pdf.Gauss(mu=mu, sigma=sigma, obs=obs)

data = zfit.Data.from_numpy(obs=obs, array=normal_np)

nll = zfit.loss.UnbinnedNLL(model=gauss, data=data)

minimizer = zfit.minimize.Minuit()
result = minimizer.minimize(nll)

param_errors = result.error()
from zfit import ztf

class CustomPDF(zfit.pdf.ZPDF):
    _PARAMS = ['alpha']

    def unnormalized_pdf(self, x):
        data = x.unstack_x()
        alpha = self.params['alpha']
        return ztf.exp(alpha * data)

    implement custom function
from zfit import ztf

class CustomPDF(zfit.pdf.ZPDF):
    _PARAMS = ['alpha']

    def _unnormalized_pdf(self, x):
        data = x.unstack_x()
        alpha = self.params['alpha']

        return ztf.exp(alpha * data)

custom_pdf = CustomPDF(obs=obs, alpha=0.2)

integral = custom_pdf.integrate(limits=(-1, 2))
sample = custom_pdf.sample(n=1000)
prob = custom_pdf.pdf(sample)
from zfit import ztf

class CustomPDF(zfit.pdf.ZPDF):
    _PARAMS = ['alpha']

def _unnormalized_pdf(self, x):
    data = x.unstack_x()
    alpha = self.params['alpha']

    return ztf.exp(alpha * data)

custom_pdf = CustomPDF(obs=obs, alpha=0.2)

integral = custom_pdf.integrate(limits=(-1, 2))
sample = custom_pdf.sample(n=1000)
prob = custom_pdf.pdf(sample)

Example of Base Classes in general inside zfit

} use functionality of model
class P5pPDF(zfit.pdf.ZPDF):
    __PARAMS = ['FL', 'AT2', 'P5p']
    _N_OBS = 3

def _unnormalized_pdf(self, x):
    FL = self.params['FL']
    AT2 = self.params['AT2']
    P5p = self.params['P5p']
    costheta_k, costheta_l, phi = ztf.unstack_x(x)

    sintheta_k = tf.sqrt(1.0 - costheta_k * costheta_k)
    sintheta_l = tf.sqrt(1.0 - costheta_l * costheta_l)

    sintheta_2k = (1.0 - costheta_k * costheta_k)
    sintheta_2l = (1.0 - costheta_l * costheta_l)

    sin2theta_k = (2.0 * sintheta_k * costheta_k)
    cos2theta_l = (2.0 * costheta_l * costheta_l - 1.0)

    pdf = (3.0 / 4.0) * (1.0 - FL) * sintheta_2k + \\
    FL * costheta_k * costheta_k + \\
    (1.0 / 4.0) * (1.0 - FL) * sintheta_2k * cos2theta_l + \\
    -1.0 * FL * costheta_k * costheta_k * cos2theta_l + \\
    (1.0 / 2.0) * (1.0 - FL) * AT2 * sintheta_2k * sintheta_2l * tf.cos(2.0 * phi) + \\
    tf.sqrt(FL * (1 - FL)) * P5p * sin2theta_k * sintheta_l * tf.cos(phi)

    return pdf
Complete fit: Data

```python
normal_np = np.random.normal(loc=2., scale=3., size=10000)
obs = zfit.Space("x", limits=(-2, 3))
mu = zfit.Parameter("mu", 1.2, -4, 6)
sigma = zfit.Parameter("sigma", 1.3, 0.1, 10)
gauss = zfit.pdf.Gauss(mu=mu, sigma=sigma, obs=obs)
data = zfit.Data.from_numpy(obs=obs, array=normal_np)
nll = zfit.loss.UnbinnedNLL(model=gauss, data=data)
minimizer = zfit.minimize.Minuit()
result = minimizer.minimize(nll)
param_errors = result.error()
```
Complete fit: Loss

```python
normal_np = np.random.normal(loc=2., scale=3., size=10000)

obs = zfit.Space("x", limits=(-2, 3))

mu = zfit.Parameter("mu", 1.2, -4, 6)
sigma = zfit.Parameter("sigma", 1.3, 0.1, 10)
gauss = zfit.pdf.Gauss(mu=mu, sigma=sigma, obs=obs)

data = zfit.Data.from_numpy(obs=obs, array=normal_np)

nll = zfit.loss.UnbinnedNLL(model=gauss, data=data)

minimizer = zfit.minimize.Minuit()
result = minimizer.minimize(nll)

param_errors = result.error()
```
normal_np = np.random.normal(loc=2., scale=3., size=10000)

obs = zfit.Space("x", limits=(-2, 3))

mu = zfit.Parameter("mu", 1.2, -4, 6)
sigma = zfit.Parameter("sigma", 1.3, 0.1, 10)
gauss = zfit.pdf.Gauss(mu=mu, sigma=sigma, obs=obs)

data = zfit.Data.from_numpy(obs=obs, array=normal_np)

nll = zfit.loss.UnbinnedNLL(model=gauss, data=data)

minimizer = zfit.minimize.Minuit()
result = minimizer.minimize(nll)

param_errors = result.error()
normal_np = np.random.normal(loc=2., scale=3., size=10000)

obs = zfit.Space("x", limits=(-2, 3))

mu = zfit.Parameter("mu", 1.2, -4, 6)
sigma = zfit.Parameter("sigma", 1.3, 0.1, 10)
gauss = zfit.pdf.Gauss(mu=mu, sigma=sigma, obs=obs)

data = zfit.Data.from_numpy(obs=obs, array=normal_np)
nll = zfit.loss.UnbinnedNLL(model=gauss, data=data)

minimizer = zfit.minimize.Minuit()
result = minimizer.minimize(nll)

param_errors = result.error()
Complete fit: plots

Gaussian example

Angular Analysis

![Gaussian example plot](image1)

![Angular Analysis graph](image2)
zfit: status

Public beta stage *(pip/conda install zfit)*

- **0.3**: TF 1.x, Unbinned
- **0.6**: TF 2.0, Model, Binned
- **0.9**: Final API

**Focus on**

- stable API & workflow
- core implementations
- interface & base classes

**Not on content**

Summer 2020
Community involvement

Who
Everyone who does likelihood fits

What

- Discussions (API, features, …): zfit-development
  - Usecases
  - Ideas
  - Experience
  - Doubts
- Use it; ask; wish; criticize
Try it out: https://github.com/zfit/zfit-tutorials
Backup Slides
https://zfit.github.io/zfit/

Join the discussion!

zfit@GitHub  Gitter channel  zfit@physik.uzh.ch
Python model fitting in HEP

- **Scalable**: large data, complex models
- **Pythonic**: use Python ecosystem/language
- **Specific HEP functionality**:
  - Normalization: specific range, numerical integration,...
  - Composition of models
  - Multiple dimensions
  - Custom models
  - Non-trivial loss (constraints, simultaneous,...)
RooFit

- Limited customization and extendibility
- Sub-optimal scalability for ever larger datasets and modern computing infrastructure

- Isolated, aging ecosystem, no cutting-edge software

- Not Python native
  - Memory allocation errors
  - Arbitrary C++ limitations
  - No real integration into the Python ecosystem
HEP Python projects

Profbfit, TensorProb,…

- Lack **generality** and extendibility
- “experimental”, but great proof of concept
  - API and Python in general
  - Computational backends (e.g. Cython, TensorFlow)
  - Building an ecosystem (iminuit,…)

General impression in comparison with other HEP packages
Non-HEP

Scipy, Imfit, TensorFlow Probability,…

• Lack of specific HEP features
  - Normalization: specific range, numerical integration,…
  - Composition of models
  - Multiple dimensions
  - Custom models

• Irrelevant functionality supported in API
  - Survival function, …
TFA: approach & differences

- Build «optimized» TensorFlow
  - accept-reject as tf.while_loop, Dataset input, ...
- ...and hide the tedious, unambiguous parts
  - automatic normalization, Tensor cache, ...
- Well defined structures, e.g.
  - String name order (like columns) in PDFs, data, limits, ...
    - pdf("x") * pdf("y") => pdf("x", "y")
      1-dim   1-dim   2-dim
  - Local/recursive dependency resolution of Parameters
Example amplitude

```
RESONANCES = [('rho(770)', 'pi-', 'pi0'), bw_amplitude),
('K(2)*(1430)0', 'K+', 'pi-'), bw_amplitude),
('K(0)*(1430)+', 'K+', 'pi0'), bw_amplitude),
('K*(892)+', 'K+', 'pi0'), bw_amplitude),
('K(0)*(1430)0', 'K+', 'pi-'), bw_amplitude),
('K*(892)0', 'K+', 'pi-'), bw_amplitude))

COEFFS = {...}

D2Kpipi0 = Decay('D0', ['K+', 'pi-', 'pi0'])

for res, children, amp in RESONANCES:
    D2Kpipi0.add_amplitude(res, children, amp, COEFFS[res])

formalism = ThreeBodyDalitzFormalism("Zemach B Frame")

pdf = D2Kpipi0.create_pdf(name="D2Kpipi0", formalism=formalism)
```
zfit project

- zfit: stable core
  - Unbinned fits, binned WIP
  - n-dim models with integral, pdf, sample
- zfit-physics: HEP specific content
  - BreitWigner, DoubleCB,…
  - Faster development, more content
  - Ideal for contributions
    - Auto testing of new pdfs/func
    - Contribution guidelines
Pythonic

- Pure Python («pip install zfit»)
- Integrated into python ecosystem
  - Load ROOT files (uproot, no ROOT dependence!)
  - Use Minuit for minimization (iminuit)
  - Data preprocessing with Pandas DataFrame
  - Plotting with matplotlib
  - High level statistics (lauztat, more WIP)
- Extendable classes
  - e.g. custom PDF
Scalable

- TensorFlow hidden backend, uses graphs
  - numpy-like syntax
  - parallelization on CPU/GPU, analytic gradient,...
- Writing functions simple for users and developers
  - No Cython, MPI, CUDA,... for state-of-the-art performance
  - No low-level maintenance required!
- Used in multiple physics libraries and analyses
Scalable: TensorFlow

- Deep Learning framework by Google
- Modern, declarative graph approach
- Built for highly parallelized, fast communicating CPU, GPU, TPU,… clusters
- Built to use «Big Data»
Can we express model fitting as static graphs?
Graph elements

… do not have to be constant!
Graph elements

... do not have to be constant!

Parameters
Can change their value
Graph elements

… do not have to be constant!

**Parameters**
Can change their value

**Random numbers**
Generate newly on every graph execution: MC integration,…
Graph elements

... do not have to be constant!

Parameters
  Can change their value

Random numbers
  Generate newly on every graph execution: MC integration,…

Control flow (if, while)
  Steer the execution: Accept-reject sampling (while), etc.
Graph elements

... do not have to be constant!

**Parameters**
- Can change their value

**Random numbers**
- Generate newly on every graph execution: MC integration, ...

**Control flow (if, while)**
- Steer the execution: Accept-reject sampling (while), etc.

Static, not constant
Can we express model fitting as static graphs?

Yes!
HPC perspective

1) Definition of computation, shape etc. (add static knowledge)
2) Compilation of the graph
3) Execution of computation (re-use optimized graph)

Inside TF, hidden to end-user

HPC: the more is know *before* the execution, the better

TensorFlow takes care of *how* to use this knowledge
Model, loss building

sum of two pdfs

```python
sum_pdf = zfit.pdf.SumPDF([gauss, exponential], fracs=frac)
```

shared parameters

```python
mu_shared = zfit.Parameter("mu_shared", 1., -4, 6)
```

```python
gauss1 = zfit.pdf.Gauss(mu=mu_shared, sigma=sigma1, obs=obs)
gauss2 = zfit.pdf.Gauss(mu=mu_shared, sigma=sigma2, obs=obs)
```

simultaneous loss

```python
nll1 = zfit.loss.UnbinnedNLL(model=gauss1, data=data1)
nll2 = zfit.loss.UnbinnedNLL(model=gauss2, data=data2)
nll_simultaneous2 = nll1 + nll2
```
Model, loss building

Simple combinations
func_n = zfit.func.ZFunc(...)  # pseudo code
func = func_1 + func_2 * func_3

Composite Parameter
pdf = zfit.pdf.Gauss(mu=tensor1, sigma=4)

Custom Loss
loss = zfit.loss.SimpleLoss(lambda: tensor_loss)

=> use all of zfit functionality like minimizers
Model building

```python
obs = zfit.Space("x", limits=(-10, 10))
mu = zfit.Parameter("mu", 1, -4, 6)
sigma = zfit.Parameter("sigma", 1, 0.1, 10)
lambda = zfit.Parameter("lambda", -1, -5, 0)
frac = zfit.Parameter("fraction", 0.5, 0, 1)
goost = zfit.pdf.Gauss(mu=mu, sigma=sigma, obs=obs)
exponential = zfit.pdf.Exponential(lambda, obs=obs)
```
Simultaneous fit

```python
mu_shared = zfit.Parameter("mu_shared", 1., -4, 6)
sigma1 = zfit.Parameter("sigma_one", 1., 0.1, 10)
sigma2 = zfit.Parameter("sigma_two", 1., 0.1, 10)

gauss1 = zfit.pdf.Gauss(mu=mu_shared, sigma=sigma1, obs=obs)
gauss2 = zfit.pdf.Gauss(mu=mu_shared, sigma=sigma2, obs=obs)

nll_simultaneous = zfit.loss.UnbinnedNLL(model=[gauss1, gauss2], data=[data1, data2])
nll1 = zfit.loss.UnbinnedNLL(model=gauss1, data=data1)
nll2 = zfit.loss.UnbinnedNLL(model=gauss2, data=data2)
nll_simultaneous2 = nll1 + nll2
```
Scalable: TensorFlow

• Machine learning in a nutshell:
  - Build a model (a lot of matrix multiplications with simple non-linear functions in between) with 100k+ free parameters
  - Create a loss function (see how good/bad the predictions are)
  - Minimize it
Pythonic: statistics tool «lauztat»

- Author: Matthieu Marinangeli
- WIP, pre-beta
- Python statistics tool for limits, significance etc. (~ RooStats)
- lauztat on Github with example notebooks using zfit
Pythonic: «phasespace»

- Author: Albert Puig
- Python tool for n-body phasespace generation (~ TGenPhaseSpace)
Fitting: complete structure

Model \rightarrow Loss \rightarrow Minimize

PDF \rightarrow Func \rightarrow Parameters

Data \rightarrow Result & Errors

Space
Sources

- Neural network: [https://cdn-media-1.freecodecamp.org/images/Q8MdDayhWcsEz9A2fItm7lFEovErXC4a8L7S](https://cdn-media-1.freecodecamp.org/images/Q8MdDayhWcsEz9A2fItm7lFEovErXC4a8L7S)