zfit
scalable pythonic fitting

Jonas Eschle on behalf of zfit
jonas.eschle@cern.ch

FNS SNF
Swiss National Science Foundation

University of Zurich
Statistical inference in HEP

Analysis: from collecting data to extracting statistical statement

Reconstruction, stripping, ...
Statistical inference in HEP

Analysis: from collecting data to extracting statistical statement

Reconstruction, stripping, ...

Selection, corrections, ...

![Graph showing $q^2_{	ext{hadron}}$ vs. $q^2$ with different colored data points for $\pi^+\pi^-$, $\pi^+\pi^-$, $J/\psi\rightarrow\mu^+\mu^-$, and $\psi(2S)\rightarrow\mu^+\mu^-\pi^+$]
Statistical inference in HEP

Analysis: from collecting data to extracting statistical statement
Involves often likelihood fitting and hypothesis testing

Reconstruction, stripping, ...
Selection, corrections, ...
Model and fit
Hypothesis testing
Statistical inference in HEP

Analysis: from collecting data to extracting statistical statement
Involves often likelihood fitting and hypothesis testing
How to use Python for analysis?

Reconstruction, stripping, ...
Selection, corrections, ...
Model and fit
Hypothesis testing
Statistical inference in HEP

Analysis: from collecting data to extracting statistical statement
Involves often likelihood fitting and hypothesis testing

Reconstruction, stripping, ...
Selection, corrections, ...
Model and fit
Hypothesis testing
HEP Model Fitting in Python

HEP advanced features, simply extendable

Scalable large data, complex models

Pythonic integrate into ecosystem, stable API
HEP Model Fitting in Python

HEP
advanced features, simply extendable

Scalable
large data, complex models

Pythonic
integrate into ecosystem, stable API

2 Sep 2021 SPS & ÖPG meeting 2021 - zfit by Jonas Eschle
normal_np = np.random.normal(2., 3., size=10_000)

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

mu = zfit.Parameter("mu", 1.2, -4, 6)
sigma = zfit.Parameter("sigma", 1.3, 0.5, 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.hesse()  # param_errors_asymmetric, new_result = result.errors()
Complete fit: Model

```python
normal_np = np.random.normal(2., 3., size=10_000)

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

mu = zfit.Parameter("mu", 1.2, -4, 6)
sigma = zfit.Parameter("sigma", 1.3, 0.5, 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.hesse()
param_errors_asymmetric, new_result = result.errors()
```
normal_np = np.random.normal(2., 3., size=10_000)

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

mu = zfit.Parameter("mu", 1.2, -4, 6)
sigma = zfit.Parameter("sigma", 1.3, 0.5, 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.hesse()
param_errors_asymmetric, new_result = result.errors()}
normal_np = np.random.normal(2., 3., size=10_000)

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

mu = zfit.Parameter("mu", 1.2, -4, 6)
sigma = zfit.Parameter("sigma", 1.3, 0.5, 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.hesse()
param_errors_asymmetric, new_result = result.errors()
normal_np = np.random.normal(2., 3., size=10_000)

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

mu = zfit.Parameter("mu", 1.2, -4, 6)
sigma = zfit.Parameter("sigma", 1.3, 0.5, 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.hesse()
param_errors_asymmetric, new_result = result.errors()
normal_np = np.random.normal(2., 3., size=10_000)

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

mu = zfit.Parameter("mu", 1.2, -4, 6)
sigma = zfit.Parameter("sigma", 1.3, 0.5, 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.hesse()
param_errors_asymmetric, new_result = result.errors()
Basic API example

```python
mu = zfit.Parameter("mu", 1.2, -4, 6)
sigma = zfit.Parameter("sigma", 1.3, 0.5, 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.hesse()
param_errors_asymmetric, new_result = result.errors()
```
Workflow

Five maximally independent parts

"Fits look always the same"
HEP Model Fitting in Python

- HEP
  - advanced features, simply extendable

- Scalable
  - large data, complex models

- Pythonic
  - integrate into ecosystem, stable API

Scikit

2 Sep 2021
SPS & ÖPG meeting 2021 - zfit by Jonas Eschle
There is no free lunch

- Initial overhead, flat increase
- TensorFlow (JAX, ...) backend
- JIT compiled, CPU or GPU time

- Single, simple fit "slow"
  - 0.01 or 1 sec not relevant
  - 1 or 10 hours relevant
Sum of 9 Gaussians, toy fitting time, 6 core CPU: **RooFit vs. zfit**

**Same order of magnitude as RooFit**

- RooFit
- zfit CPU
- zfit CPU nograd
- zfit GPU

![Graph](image-url)
Scalable: Usability

*Things should not be easy or hard, but consistent*

- Code lines
  - 5 or 10: irrelevant
  - 50 or 300: matters
Scalable: Usability

Things should not be easy or hard, but consistent

- Code lines
  - 5 or 10: irrelevant
  - 50 or 300: matters
- Cover all use cases out of the box is impossible
  - Convenient base classes, allow full control
  - Modular structure; provide all elements (e.g. shapes)
HEP Model Fitting in Python

- Scalable: large data, complex models
- Advanced features, simply extendable
- Pythonic: integrate into ecosystem, stable API
```python
normal_np = np.random.normal(2., 3., size=10_000)

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

mu = zfit.Parameter("mu", 1.2, -4, 6)
sigma = zfit.Parameter("sigma", 1.3, 0.5, 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.hesse()
param_errors_asymmetric, new_result = result.errors()```
Complete fit: Model

```python
normal_np = np.random.normal(2., 3., size=10_000)

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

mu = zfit.Parameter("mu", 1.2, -4, 6)
sigma = zfit.Parameter("sigma", 1.3, 0.5, 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.hesse()
param_errors_asymmetric, new_result = result.errors()
```
Example: Mass fit

- Sum, Product, \textit{(Convolution)}
- Gauss, (double) Crystalball,...
- Exponential, Polynomials,...

```python
lambda = zfit.Parameter("lambda", -0.06, -1, -0.01)
frac = zfit.Parameter("fraction", 0.3, 0, 1)

gauss = zfit.pdf.Gauss(mu=mu, sigma=sigma, obs=obs)
exponential = zfit.pdf.Exponential(lambda, obs=obs)
model = zfit.pdf.SumPDF([gauss, exponential], fracs=frac)
```
Example: Mass fit

- Sum, Product, *Convolution*
- Gauss, (double) Crystalball,...
- Exponential, Polynomials,...

```
lambda = zfit.Parameter("lambda", -0.06, -1, -0.01)
frac = zfit.Parameter("frac", 0.5, 0, 1)

gauss = zfit.pdf.Gauss(mu=mu, sigma=sigma, obs=obs)
exponential = zfit.pdf.Exponential(loge=loge, obs=obs)
model = zfit.pdf.SumPDF([gauss, exponential], fracs=frac)
```

Good for out-of-the-box but... does not cover even closely all HEP PDFs
```python
from zfit import z
from zfit.z import numpy as znp

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

    def _unnormalized_pdf(self, x):
        data = z.unstack_x(x)
        alpha = self.params['alpha']
        return znp.exp(alpha * data)
```

implement custom function
from zfit import z
from zfit.z import numpy as znp

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

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

} use functionality of model
$B^0 \rightarrow K^{*0} l^+ l^-$ angular: P5'

P5': optimised observable
Fit of P5', from [1, 2]


2 Sep 2021

SPS & ÖPG meeting 2021 - zfit by Jonas Eschle
$B^0 \to K^{*0} l^+ l^-$ angular: fitted P5'

Projections of three angles

Plot with mplhep, matplotlib
normal_np = np.random.normal(2., 3., size=10_000)

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

mu = zfit.Parameter("mu", 1.2, -4, 6)
sigma = zfit.Parameter("sigma", 1.3, 0.5, 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.hesse()
param_errors_asymmetric, new_result = result.errors()
Complete fit: Data

• From different sources
  – Numpy, Pandas, ROOT, ...

• Sampled from a model (toy studies)

```python
data = model.create_sampler(n_sample, limits=obs)
```
$B^0 \rightarrow K^{*0} l^+ l^-$ angular: toy study

Result of toy study

P5' value

P5' error
normal_np = np.random.normal(2., 3., size=10_000)

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

mu = zfit.Parameter("mu", 1.2, -4, 6)
sigma = zfit.Parameter("sigma", 1.3, 0.5, 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.hesse()
param_errors_asymmetric, new_result = result.errors()
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

Constraints (also arbitrary) are fully supported
normal_np = np.random.normal(2., 3., size=10_000)

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

mu = zfit.Parameter("mu", 1.2, -4, 6)
sigma = zfit.Parameter("sigma", 1.3, 0.5, 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.hesse()
param_errors_asymmetric, new_result = result.errors()
Minimize

• Many different minimizers available, simply switch

```python
minimizer = zfit.minimize.IpyoptV1()
minimizer = zfit.minimize.Minuit()
minimizer = zfit.minimize.ScipyTrustConstrV1()
minimizer = zfit.minimize.NLoptLBFGSV1()
```

• Can use zfit loss, but also **pure Python function**

```python
result = minimizer.minimize(func, params)
```
normal_np = np.random.normal(2., 3., size=10_000)

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

mu = zfit.Parameter("mu", 1.2, -4, 6)
sigma = zfit.Parameter("sigma", 1.3, 0.5, 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.hesse()
param_errors_asymmetric, new_result = result.errors()
Back to HEP ecosystem: hepstats

- High level statistics library for hypothesis tests
- Takes model, data, loss from zfit
- sWeights, CI, limits, ... asymptotic or toys

```python
calculator = AsymptoticCalculator(loss, minimizer)
poinull = POIarray(Nsig, np.linspace(0.0, 25, 20))
poialt = POI(Nsig, 0)
ul = UpperLimit(calculator, poinull, poialt)
ul.upperlimit(alpha=0.05, CLs=True)
```

---

2 Sep 2021  
SPS & ÖPG meeting 2021 - zfit by Jonas Eschle
Close future: binned fits

- Integrate with boost_histogram, hist and mplhep
- Dedicated method "counts" (returns histogram)
- Features (expected):
  - Morphing of templates
  - Systematics of any kind
  - Interpolate (-> unbinned)
  - Irregular binning

Expected in ~ 1 months:
Discussion welcome! Ideas? Use-cases?
Conclusions

**HEP**
advanced features, simply extendable

**Scalable**
large data, complex models

**Pythonic**
integrate into ecosystem, stable API
scalable  pythonic  fitting

Try it out: https://github.com/zfit/zfit-tutorials
Conclusions

HEP advanced features, simply extendable

Scalable large data, complex models

Pythonic integrate into ecosystem, stable API
Sources

Backup Slides
https://zfit.github.io/zfit/

zfit@GitHub

Gitter channel

zfit@physik.uzh.ch

Join the discussion!
• Backend & TF
• Amplitude
• K*ll toys
• K*mumu Wilson coeffs
• Other fitting packages
• Zfit (associated) packages
• Zfit project
• Zfit elements examples
Backend & TensorFlow
Backend: tracing and autograd

Tracing
execute Python once, remember (algebraic) computation

Autograd
"analytic" gradient of function

Recent rise of big data industry created libraries that support this

Includes GPU support, optimizations, caching,...
build *the* stable model fitting ecosystem for HEP

- Integrate into HEP ecosystem
  - functionality limited; stable API
- Technical requirements
  - performance; maintainability
- HEP requirements
  - advanced features; simply extendable code
Deep Learning
lessons for model fitting
Deep Learning

Neural Network

„One huge, complicated function“

[Model]

[Data]

„Big Data“

[Loss]

[Minimizer]

[Results]
Main backend: TensorFlow

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

- By Google, highly popular (150k⭐, 4th on 🌐)
Main backend: 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,…
- Two modes:
  - "numpy"-like (full Python flexibility)
  - "compiled" (very performant)
## 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>SciPy</td>
</tr>
<tr>
<td>Models</td>
<td>TF Probability</td>
</tr>
<tr>
<td>Gradients</td>
<td>Numba, NumPy</td>
</tr>
<tr>
<td>Computational optimizations</td>
<td>TensorFlow</td>
</tr>
<tr>
<td>Parallelization/GPU</td>
<td>Python</td>
</tr>
<tr>
<td>Low level handling</td>
<td></td>
</tr>
</tbody>
</table>
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>SciPy</td>
</tr>
<tr>
<td>Models</td>
<td>TensorFlow</td>
</tr>
<tr>
<td>Gradients</td>
<td>Numba</td>
</tr>
<tr>
<td>Computational optimizations</td>
<td>Numpy</td>
</tr>
<tr>
<td>Parallelization/GPU</td>
<td>Python</td>
</tr>
<tr>
<td>Low level handling</td>
<td></td>
</tr>
</tbody>
</table>

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

HEP specific content/API
Models

Gradients
Computational optimizations
Parallelization/GPU
Low level handling

C++ library (RooFit,...)  Numpy based

Used & maintained (!) by industry

API & Workflow

TF Probability

SPS & ÖPG meeting 2021 - zfit by Jonas Eschle
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
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
## 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></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>HEP</td>
<td>Non-trivial functions</td>
<td>Whole Dataset</td>
<td>simultaneous, constraints</td>
<td>Global min, 2(^{nd}) derivative</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(^{th}) 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</td>
<td>Free „analytic“ derivatives!</td>
</tr>
</tbody>
</table>

But... what *is* a Deep Learning library?
<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, 1st 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>
<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^{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^{th} 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>
Scalability: Performance

Fitting time (lower is better): **RooFit vs. zfit**

**9 free parameters**

**2 free parameters**
Amplitude
RESONANCES = [('\text{rho}(770)', ['\pi-', '\pi\theta']), ('K(2)*(1430)\theta', ['K+', '\pi-']), ('K(0)*(1430)+', ['K+', '\pi\theta']), ('K*(892)+', ['K+', '\pi\theta']), ('K(0)*(1430)\theta', ['K+', '\pi-']), ('K*(892)\theta', ['K+', '\pi-'])]

COEFFS = {...}

D2Kpipi\theta = \text{Decay('D\theta'}, ['K+', '\pi-', '\pi\theta'])

for res, children, amp in RESONANCES:
    D2Kpipi\theta.add_amplitude(res, children, amp, COEFFS[res])

formalism = ThreeBodyDalitzFormalism("Zemach B Frame")

pdf = D2Kpipi\theta.create_pdf(name="D2Kpipi\theta", formalism=formalism)
Angular toys
Sensitivity study
- draw toys (sample) from PDF
- Fit to sample

```python
for i in range(ntoys):
    # set initial sampling values
    for param in params:
        param.set_value(...)
    sampler.resample()

    # set random initial values
    for param in params:
        param.set_value(...)
    result = minimizer.minimize(nll)

    if result.converged:
        ...
```
B^0 \rightarrow K^{*0} l^+ l^- \text{ angular: toys}

Sensitivity study
- draw toys (sample) from PDF
- Fit to sample

```python
for i in range(n_toys):
    # set initial sampling values
    for param in params:
        param.set_value(...)  
    sampler.resample()

    # set random initial values
    for param in params:
        param.set_value(...)

    result = minimizer.minimize(nll)

    if result.converged:
        ...
```
$B^0 \rightarrow K^{*0} l^+ l^-$ angular: toy study

Result of toy study

P5' value

P5' error
Extending with a mass shape

```python
# Create mass pdf
mu = zfit.Parameter("mu", 5279, 5200, 5400)
sigma = zfit.Parameter("sigma", 30, 0, 300)
a0 = zfit.Parameter("a0", 1.0, 0, 10)
a1 = zfit.Parameter("a1", 1.0, 0, 10)
n0 = zfit.Parameter("n0", 5, 0, 10)
n1 = zfit.Parameter("n1", 5, 0, 10)

mass = zfit.Space("mass", limits=(4900, 5600))

massPDF = zfit.pdf.DoubleCB(obs=mass, mu=mu, sigma=sigma,
                             alphal=a0, nl=n0, alphar=a1, nr=n1)

pdf = massPDF * angularPDF
```
$B^0 \rightarrow K^{*0} \mu^+ \mu^-$ full amplitude

- Measuring the full differential decay ratio [1, 2]
  - Angular, $q^2$ distribution
  - Branching ratio information

\[ A^L,R_\lambda = N_\lambda \left\{ \left( C_9 \pm C'_9 \right) + \left( C_{10} \pm C'_{10} \right) \right\} F_\lambda(q^2) + \frac{2m_b M_B}{q^2} \left[ C_7 \pm C'_7 \right] F^T_\lambda(q^2) - 16\pi^2 \frac{M_B}{m_b} H_\lambda(q^2) \]
Fitting libraries and comparison
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,...)
• 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

Probfit, 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
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
Zfit related packages
phasespace

- Package for phasespace generation of particles
- Covers functionality of TGenPhaseSpace (and more)
- Pure Python (& TensorFlow), integrates seemless with zfit

```python
pion = GenParticle('pi+', PION_MASS)
kaon = GenParticle('K+', KAON_MASS)
kstar = GenParticle('K*', KSTARZ_MASS).set_children(pion, kaon)
gamma = GenParticle('gamma', 0)
bz = GenParticle('B0', B0_MASS).set_children(kstar, gamma)

weights, particles = bz.generate(n_events=1000)
```
Zfit: project description
• 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
build stable model fitting ecosystem for HEP

- Integrate into HEP ecosystem
  functionality limited; stable API
- Technical requirements
  performance; maintainability
- Analysis requirements
  advanced features; simply extendable code
Fitting in Python

A lot of projects are around

TensorFlow Analysis
TensorProb
probfit

RooFit
mlfit
scipy
carl

TensorFlow Probability

HEP Python
Non-HEP
A lot of projects are around

TensorProb
TensorFlow Analysis
  probfit

RooFit

TensorFlow Probability
  carl
  scipy
  mlfit

HEP Python Non-HEP
A lot of projects are around

TensorFlow Analysis

TensorProb
probfit

RooFit
mlfit
carl
scipy
TensorFlow Probability

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

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

- RooFit
- HEP-Python
- Non-HEP

No optimised model fitting library for HEP that is well integrated into Python
Fitting in Python

A lot of projects are around

- RooFit
- HEP-Python
- Non-HEP

No optimised model fitting library for HEP that is well integrated into Python

... 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

- Integrate into HEP ecosystem
  functionality limited; stable API

- Technical requirements
  performance; maintainability

- Analysis requirements
  advanced features; simply extendable code
Ecosystem

vector
phasespace
zfit-physics

Scikit HEP

Extend

hepstats
zfit-amplitude

Build on top

zfit

stable core package

TensorFlow
uproot
Python
iminuit

Scikit HEP

For user contributions

2 Sep 2021
SPS & ÖPG meeting 2021 - zfit by Jonas Eschle
**Ecosystem**

- **vector**
- **phasespace**
- **zfit-physics**

---

**zfit**

*stable core package*

**model fitting & sampling**
Establish a stable API

- High level libraries (statistics, plotting,...)
  - „code against an interface, not an implementation“
- Replace each component
  - Allow other libraries to implement custom parts

Many discussions with community to avoid splitting/duplication
**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»
Zfit library examples
def func(x):
    x = np.array(x)  # make sure it's an array
    return np.sum((x - 0.1) ** 2 + x[1] ** 4)

func.errordef = 0.5

params = [1, -3, 2, 1.4, 11]

result = minimizer.minimize(func, params)
Model, loss building

sum of two pdfs

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

shared parameters

```
mu_shared = zfit.Parameter("mu_shared", 1., -4, 6)
gauss1 = zfit.pdf.Gauss(mu=mu_shared, sigma=sigma1, obs=obs)
gauss2 = zfit.pdf.Gauss(mu=mu_shared, sigma=sigma2, obs=obs)
```

simultaneous loss

```
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

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

Composite Parameter

```python
pdf = zfit.pdf.Gauss(mu=tensor1, sigma=4)
```

Custom Loss

```python
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)

# parameters

gauss = zfit.pdf.Gauss(mu=mu, sigma=sigma, obs=obs)
exponential = zfit.pdf.Exponential(lambda_, obs=obs)
```

---

2 Sep 2021
SPS & ÖPG meeting 2021 - zfit by Jonas Eschle
120
Simultaneous fit

\[
\text{mu\_shared} = \text{zfit.Parameter("mu\_shared", 1., -4, 6)} \\
\text{sigma1} = \text{zfit.Parameter("sigma\_one", 1., 0.1, 10)} \\
\text{sigma2} = \text{zfit.Parameter("sigma\_two", 1., 0.1, 10)} \\
\text{gauss1} = \text{zfit.pdf.Gauss(mu=mu\_shared, sigma=sigma1, obs=obs)} \\
\text{gauss2} = \text{zfit.pdf.Gauss(mu=mu\_shared, sigma=sigma2, obs=obs)} \\
\]

\[
\text{nll\_simultaneous} = \text{zfit.loss.UnbinnedNLL(model=[gauss1, gauss2], data=[data1, data2])} \\
\text{nll1} = \text{zfit.loss.UnbinnedNLL(model=gauss1, data=data1)} \\
\text{nll2} = \text{zfit.loss.UnbinnedNLL(model=gauss2, data=data2)} \\
\text{nll\_simultaneous2} = \text{nll1 + nll2}
\]