scikit-HEP affiliated

scalable pythonic fitting

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

FNSNF Swiss National Science Foundation

University of Zurich[UZH]

A brief history

- A few years ago: analyses transition from C++ to Python
  - Scikit-HEP was created
  - Change of philosophy: non-monolithic packages
- Fitting packages still in in C++
  - Many scattered, specialized packages
  - Speed crucial aspect (and non-trivial in python)

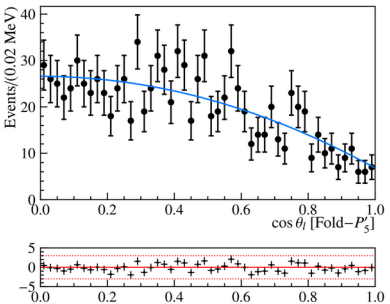
Fitting in Python

A lot of projects are around

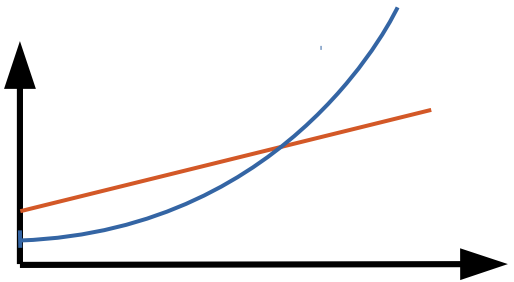
- ~~RooFit~~
- ~~HEP Python~~
- ~~Non-HEP~~

No real model fitting ecosystem/library for HEP that is well integrated into Python

HEP Model Fitting in Python



HEP
advanced features,
simply extendable

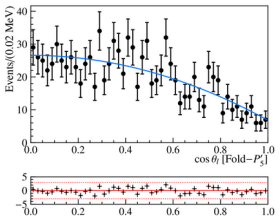


Scalable
large data, complex models

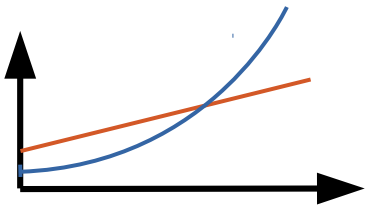
Scikit HEP

Pythonic
NumPy
python™
integrate into ecosystem, stable API

HEP Model Fitting in Python



HEP
advanced features,
simply extendable

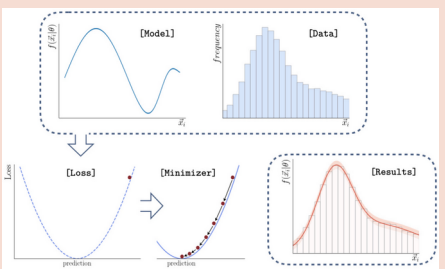


Scalable
large data, complex models

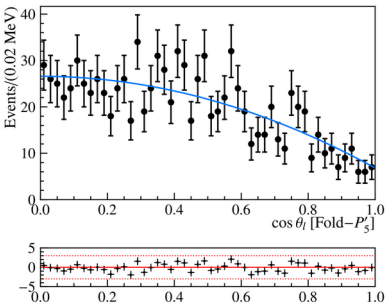
Pythonic
integrate into ecosystem, stable API

Computing backend

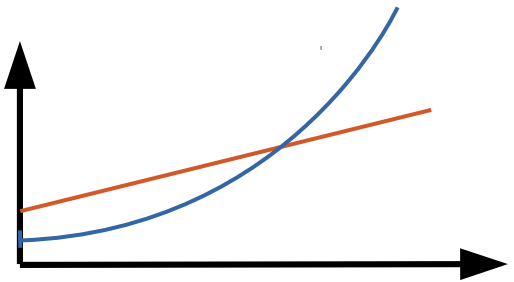
API & Workflow



HEP Model Fitting in Python



HEP
advanced features,
simply extendable

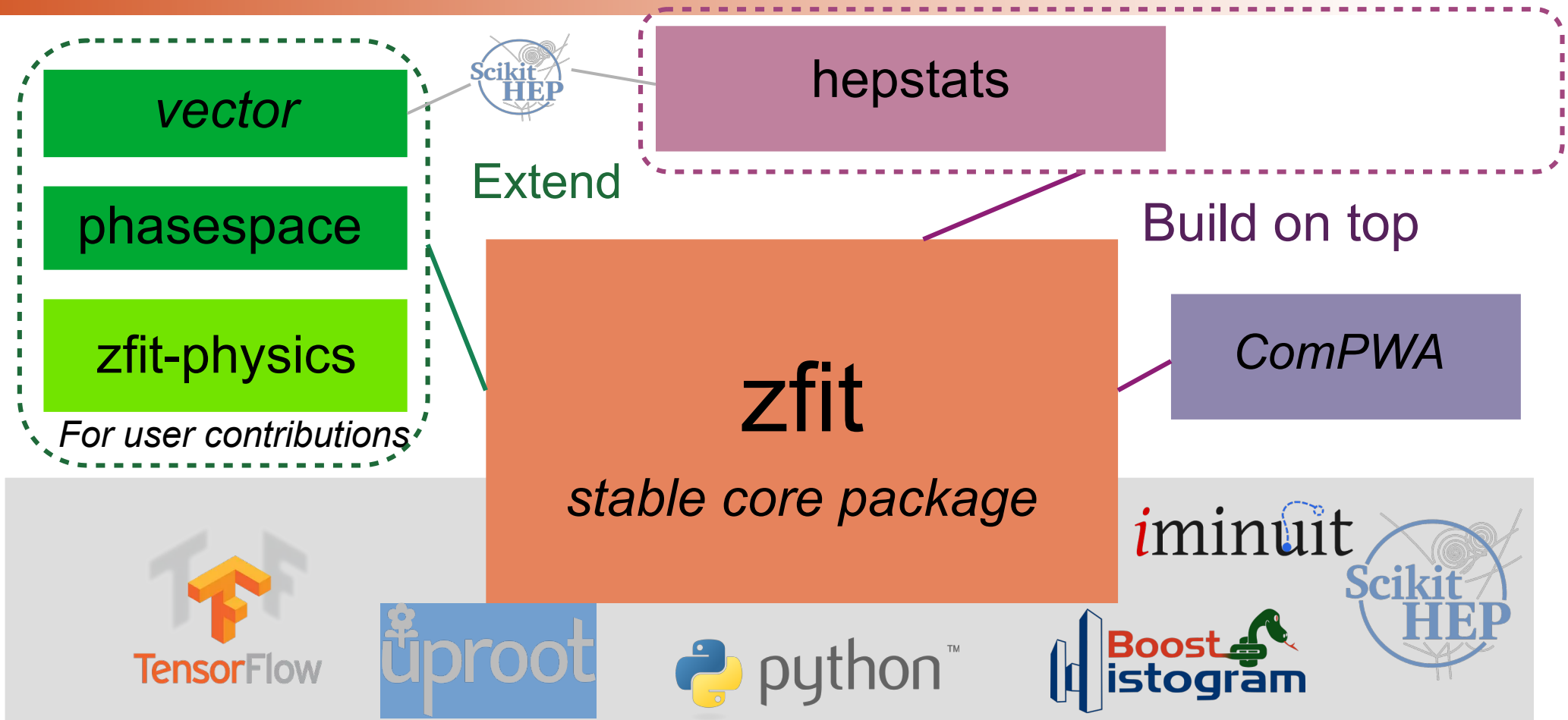


Scalable
large data, complex models

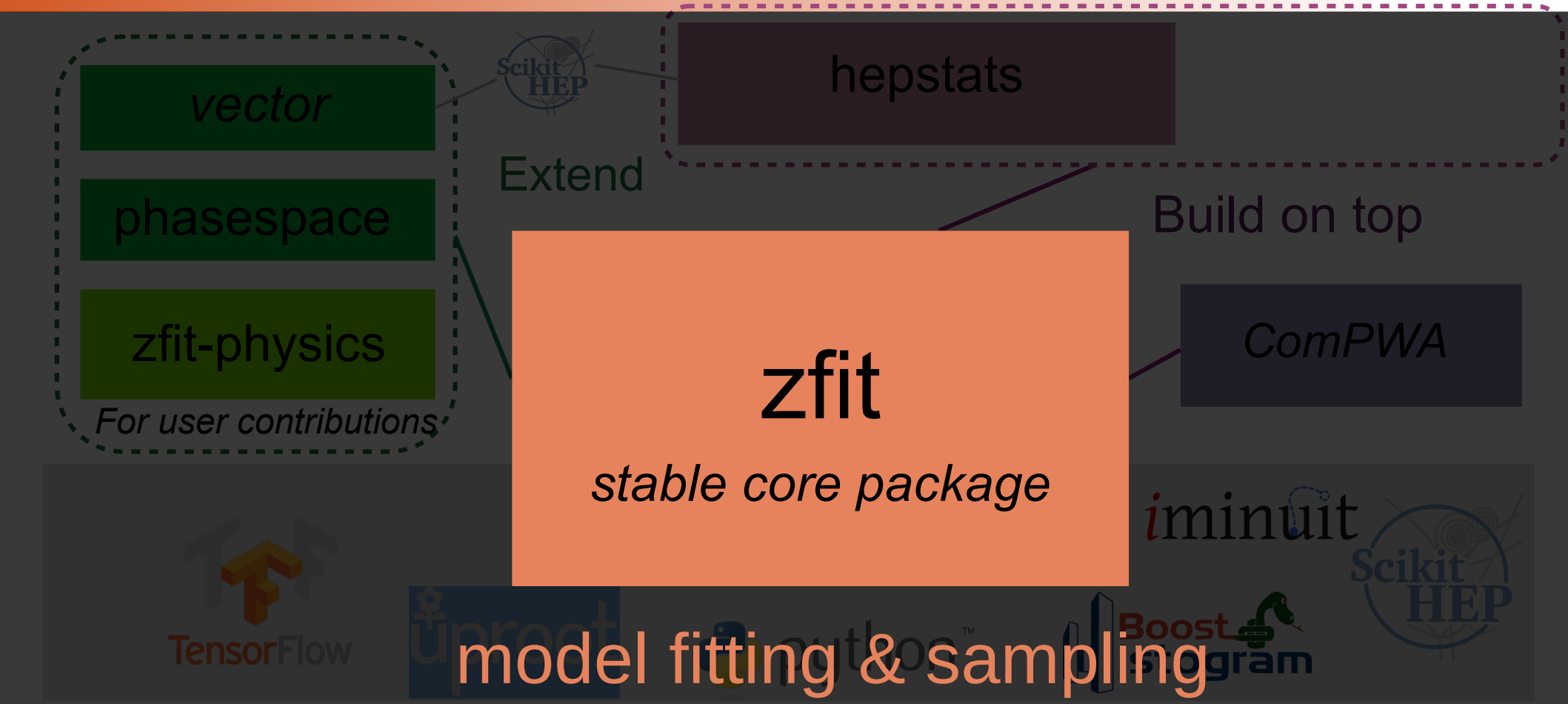
Scikit HEP

Pythonic
NumPy
python™
integrate into ecosystem, stable API

Ecosystem


vector
phasespace
zfit-physics
For user contributions
Scikit HEP
hepstats
Extend
Build on top
zfit
stable core package
ComPWA
TensorFlow
uproot
python
iminuit
Boost histogram
Scikit HEP

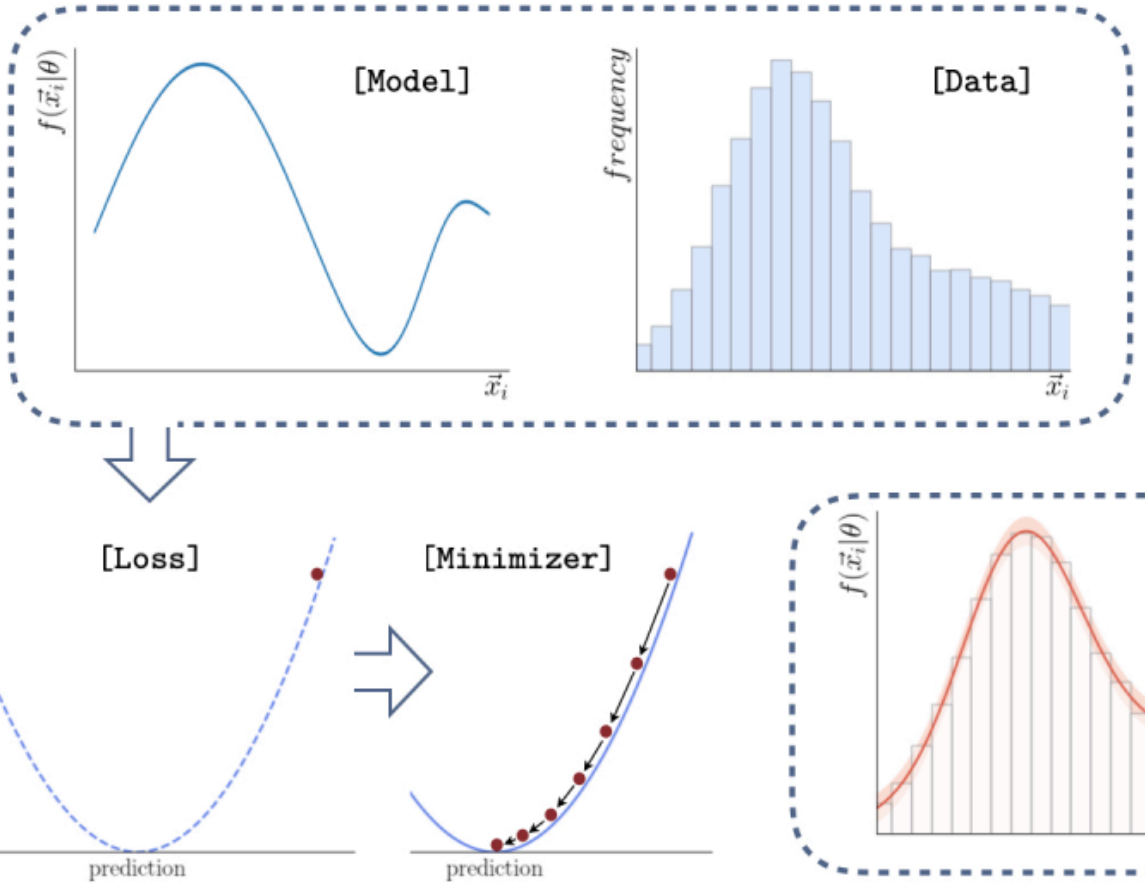
Ecosystem


vector
phasespace
zfit-physics
For user contributions
Scikit HEP
hepstats
Extend
Build on top
ComPWA
zfit
stable core package
TensorFlow
uproot
pyhf
Boost.Histogram
iminuit
Scikit HEP
model fitting & sampling

API & Workflow

Five maximally independent parts

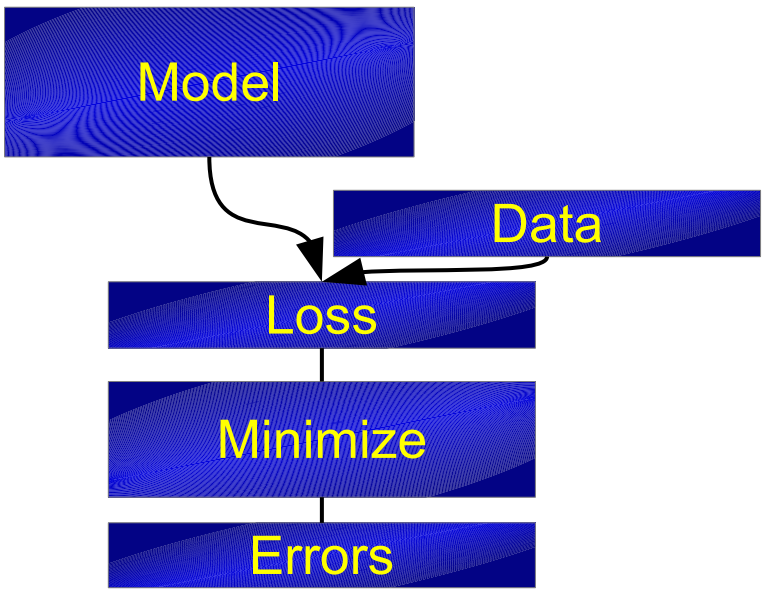
"Fits look always the same"



Complete fit

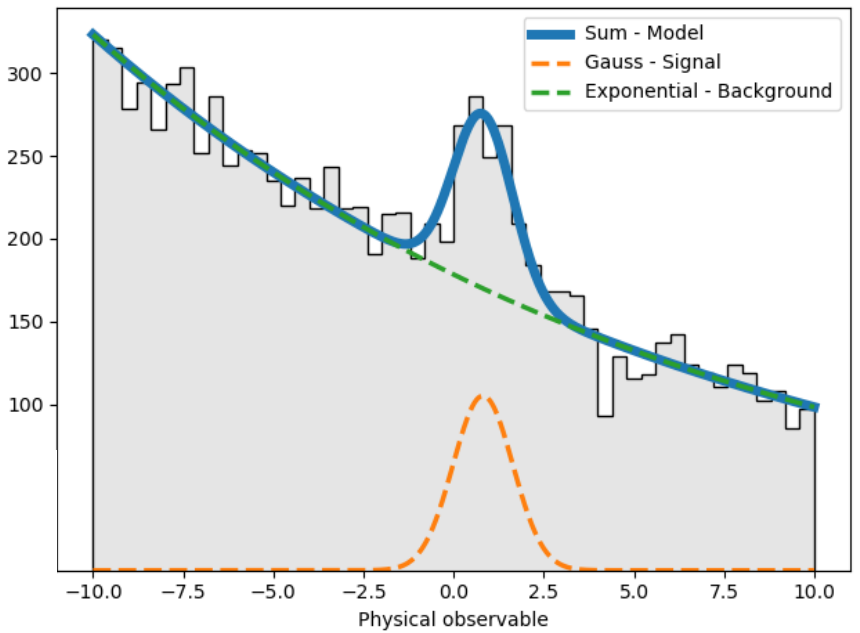
Disclaimer:
unbinned fits way more developed, binned very new in pre-release

```
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, *(Convolution)*
- Gauss, (double) Crystalball,...
- Exponential, Polynomials,…
- Histograms, SplineInterpolation,...




```
lambd = 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(lambd, obs=obs)
model = zfit.pdf.SumPDF([gauss, exponential], fracs=frac)
```

Example: Mass fit

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


```
lambd = 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(lambd, 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

Custom PDF

```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

Custom PDF

```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)
```

```python
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

Custom PDF

```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)
```

Example of zfit Base Classes

Can also override:

- integrate → _integrate
- pdf → _pdf
- sample → _sample

Or register integral

```python
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)
```

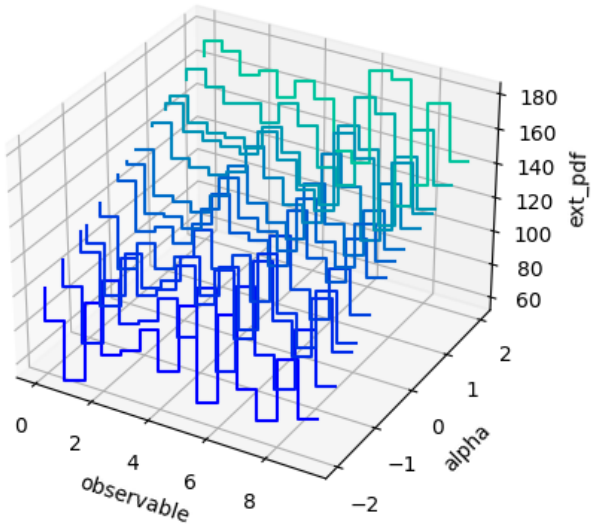
} use functionality of model

Arbitrary analytic shapes

```python
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_l, costheta_k, 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
```

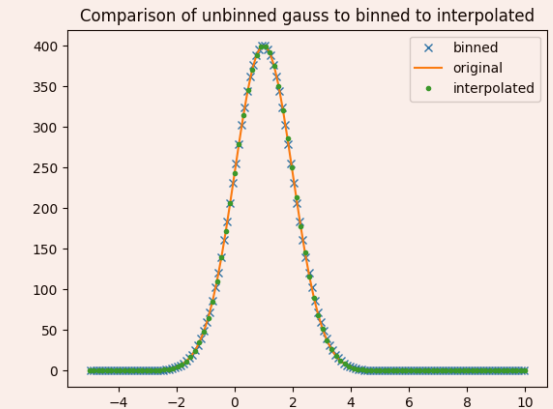
For example, create amplitude with ComPWA and fit with zfit

More histograms

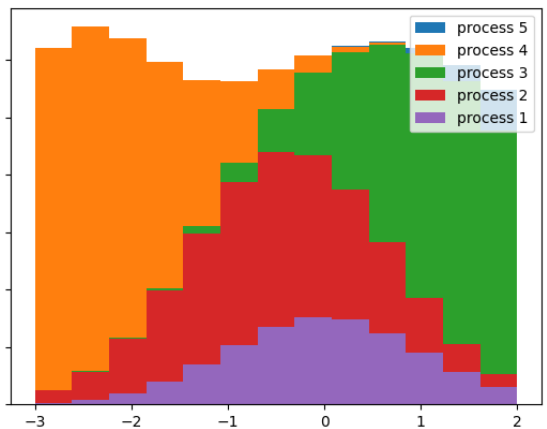


Shape modifier

```
pdf_syst = zfit.pdf.BinwiseScaleModifier(pdf, modifiers=True)
```



Unbinned → binned → interpolated

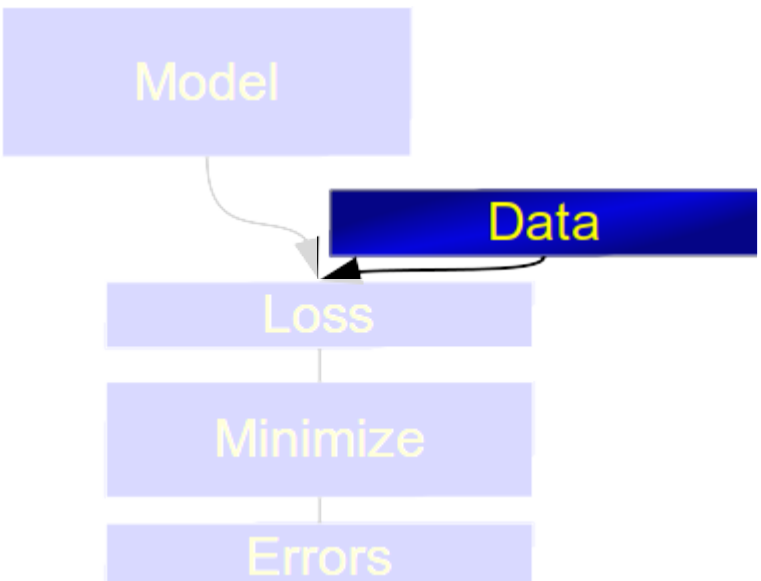



```
pdfs = [zfit.pdf.HistogramPDF(h) for h in histos]
alpha = zfit.Parameter('alpha', 0, -5, 5)
morph = SplineMorphingPDF(alpha=alpha, hists=pdfs)
```

```
pdfs = [zfit.pdf.HistogramPDF(h) for h in histos]
sumpdf = zfit.pdf.BinnedSumPDF(pdfs)
```

Complete fit: Data

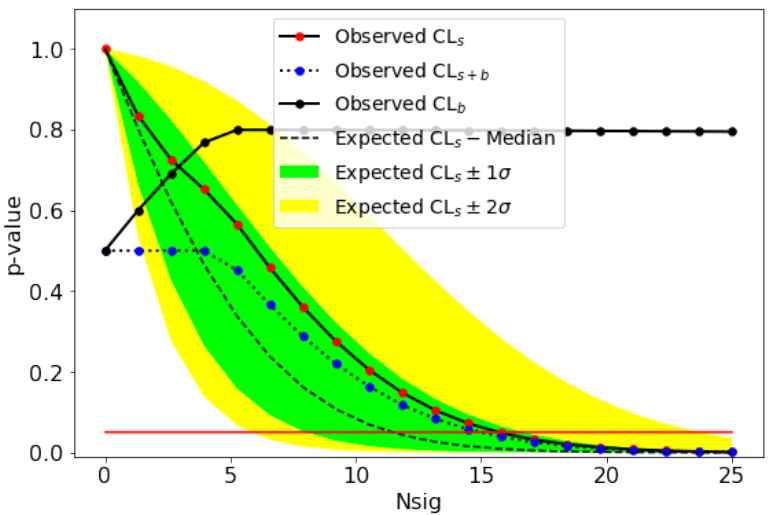
```
normal_np = np.random.normal(2., 3., size=10_000)

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

- Inference library for hypothesis tests
- Takes model, data, loss from zfit
- sWeights, CI, limits, …
  asymptotic or toys
- New: can also handle
  multi-dimensional PDFs

```
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)
```



zfit: status

Public testing stage *(pip install zfit)*

A lot of experience and proven API, but also design flaws (global parameters, ...)

Continue to incorporate feedback and adaptability to other libraries

Binned fits: still rough edges (!)

Conclusion

build stable model fitting ecosystem for HEP

- Integrate into HEP ecosystem

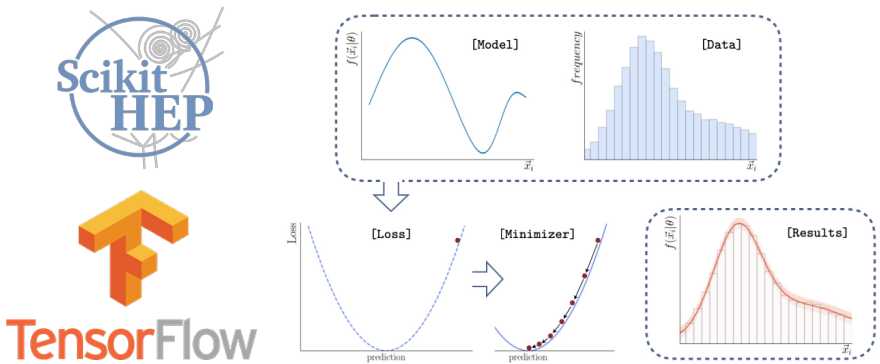
  functionality limited; stable API

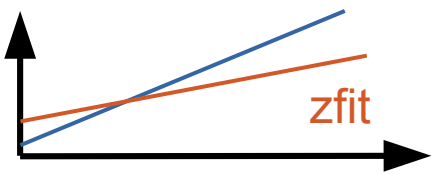
- Technical requirements

  performance; maintainability

- HEP requirements

  advanced features; simply extendable code





Sources

- LHCb collision: https://physicsworld.com/wp-content/uploads/2018/08/LHCb-collision.png

Backup Slides

https://zfit.github.io/zfit/

zfit@GitHub  Gitter channel

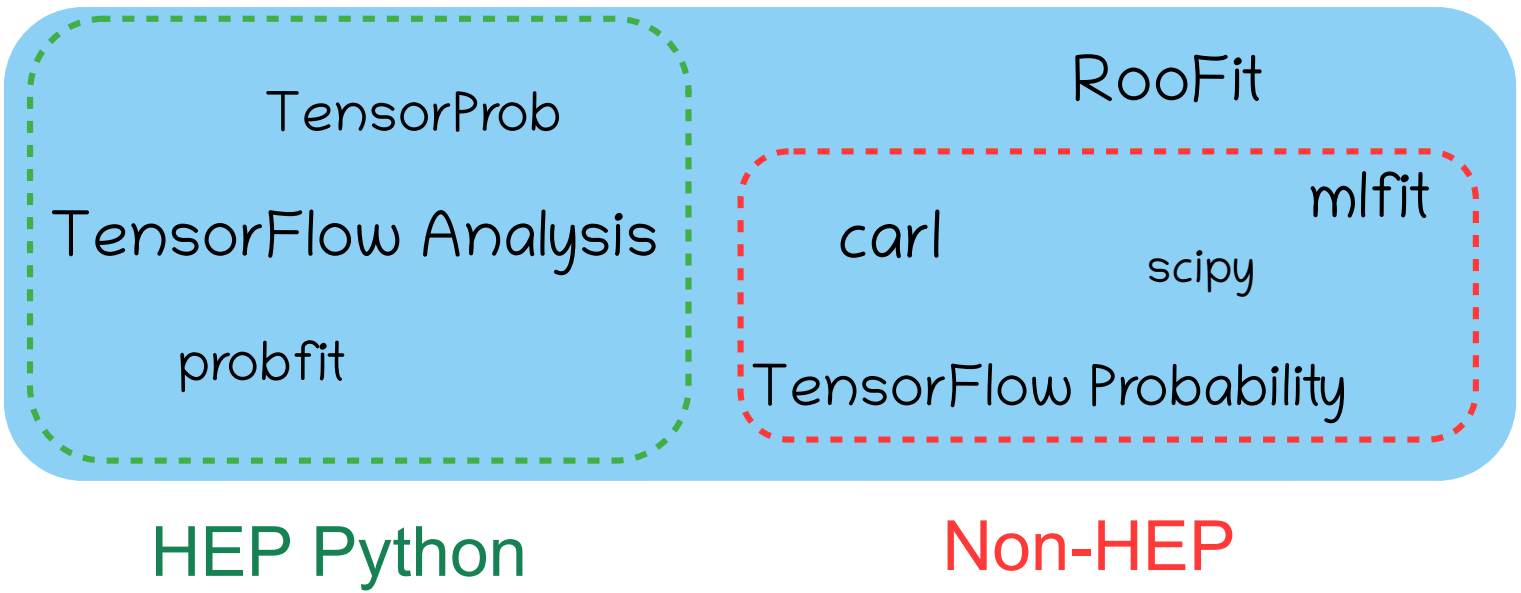
zfit@physik.uzh.ch

Join the discussion!

- Backend & TF
- Amplitude
- K*ll toys
- K*mumu Wilson coeffs
- Other fitting packages
- Zfit (associated) packages
- Zfit project
- Zfit elements examples

Fitting in Python

A lot of projects are around



Backend & TensorFlow

Backend: a comparison

- TensorFlow: supports the most features to this day
- PyTorch: missing advanced math (complex support, …)
- Numpy/SciPy: Too slow, no gradient, no GPU
- JAX: very promising, but *no globals (cache,…)*,
  only static known shapes (adaptive algorithms, accept-reject...), only JAX/Numpy arrays compatible
- SymPy: limited to mathematical expressions (no control-flow,…)
  but can convert to any other backend (used by TensorWaves)

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,...

zfit: the project

build *the* stable model fitting ecosystem for HEP

- Integrate into HEP ecosystem

  functionality limited; stable API
- Technical requirements

  performance; maintainability
- HEP requirements

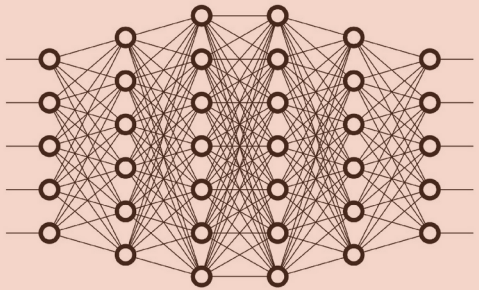
  advanced features; simply extendable code

Deep Learning

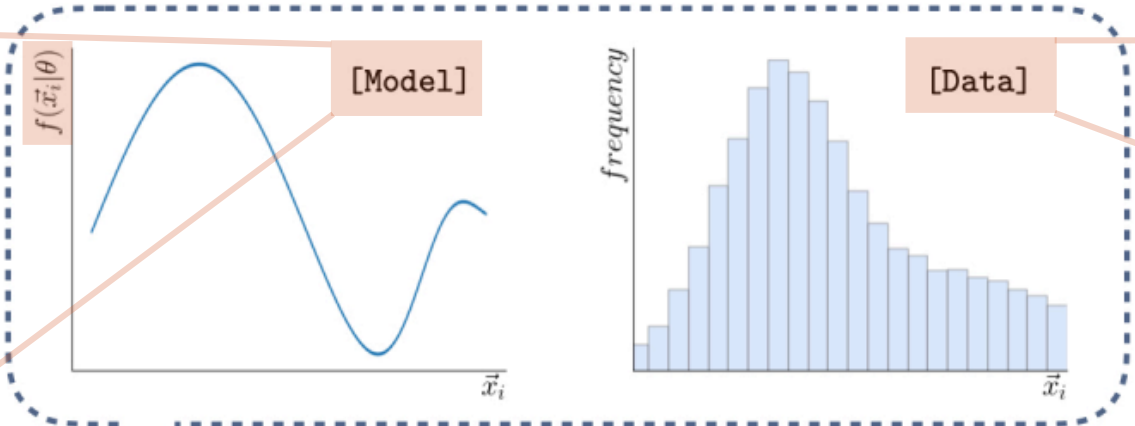
lessons for model fitting

Deep Learning

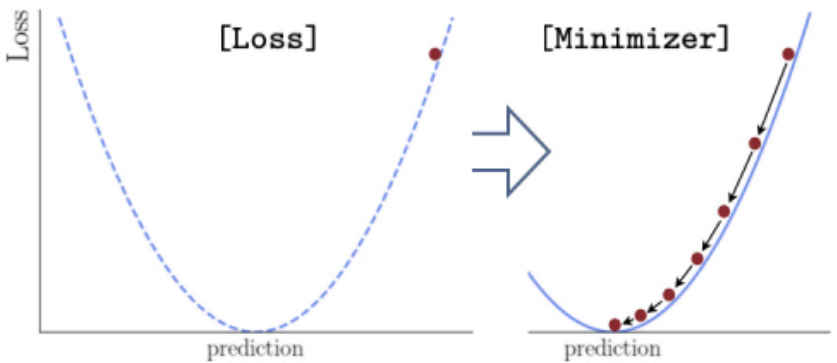
Neural Network



„One huge, complicated function“



„Big Data“



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

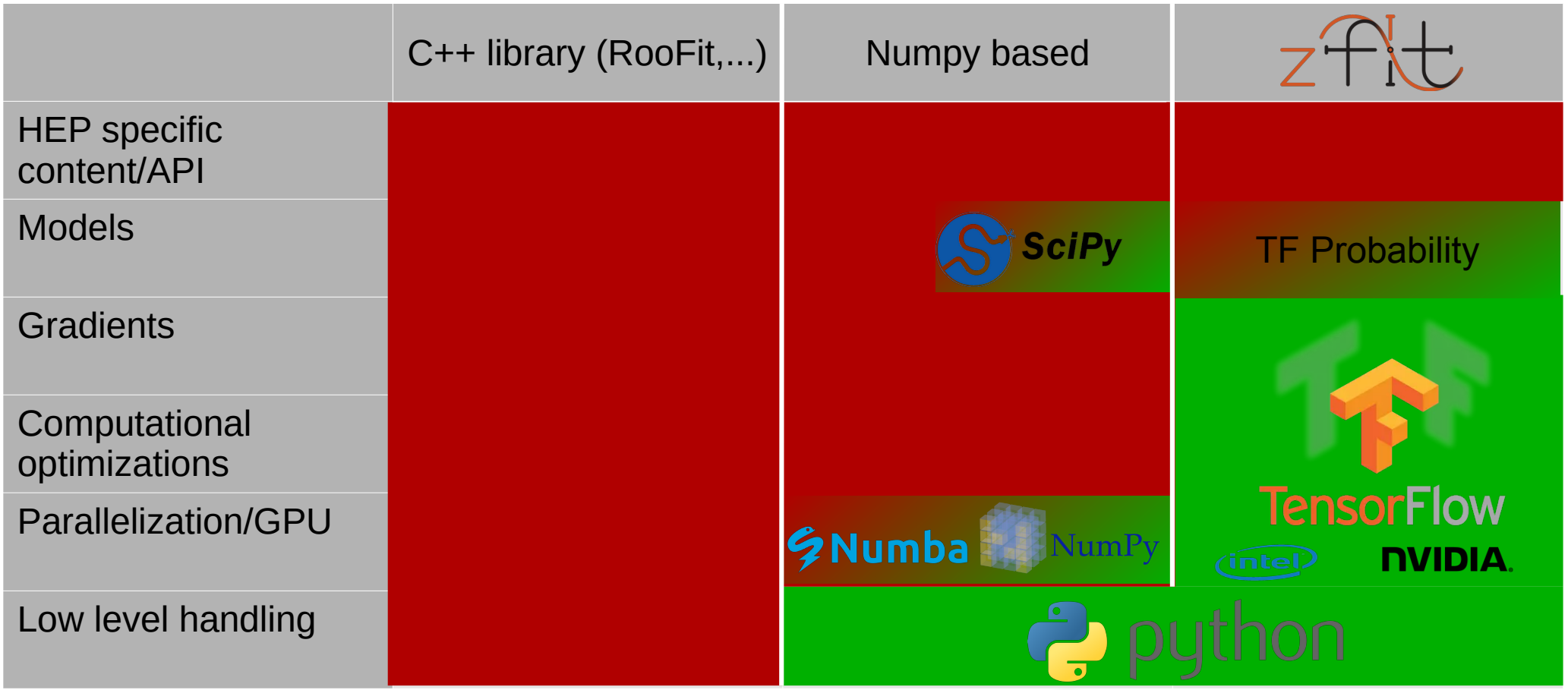
Main backend: TensorFlow

- By Google, highly popular (150k★, 4th on)
- 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)

} GPU/Multi CPU support

Delegating the workload



Delegating the workload

Delegating the workload

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

Similarity	Complicated Models	Large Data	Composed loss	Minimization	Results and uncertainties
HEP	Non-trivial functions	Whole Dataset	simultaneous, constraints	Global min, 2nd derivative algorithm	Hesse, profiling
Deep Learning	Combine many, trivial functions	Many, small Batches	*Anything! (GANs, RL,...)*	Local (!) min, 1th derivative, many steps	None
Conclusion					

Deep Learning vs. Model Fitting

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

But...

what ***is*** a Deep Learning library?

Deep Learning vs. Model Fitting

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

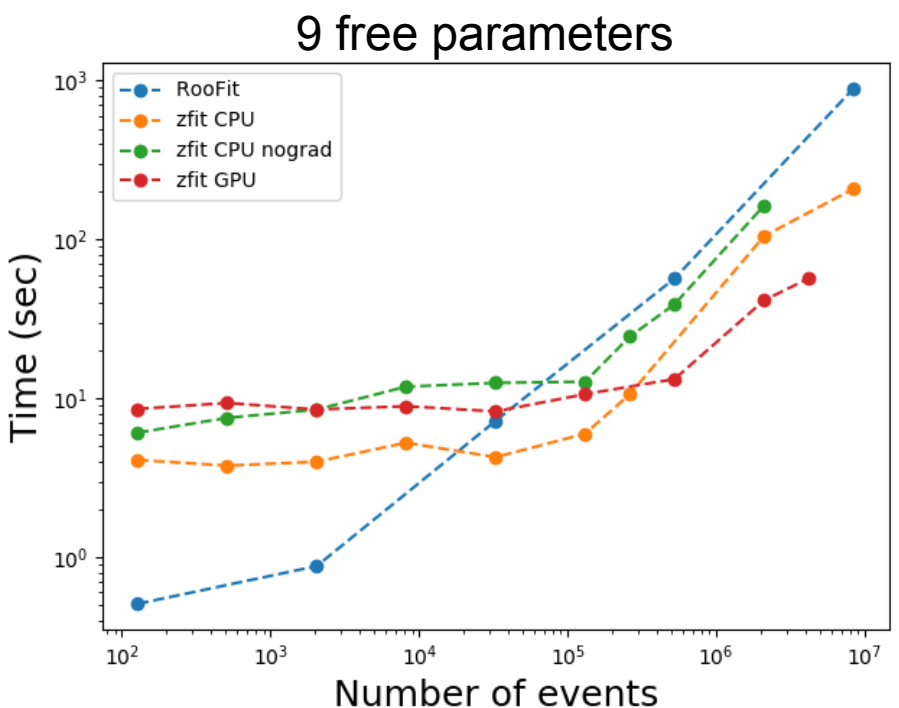
Deep Learning vs. Model Fitting

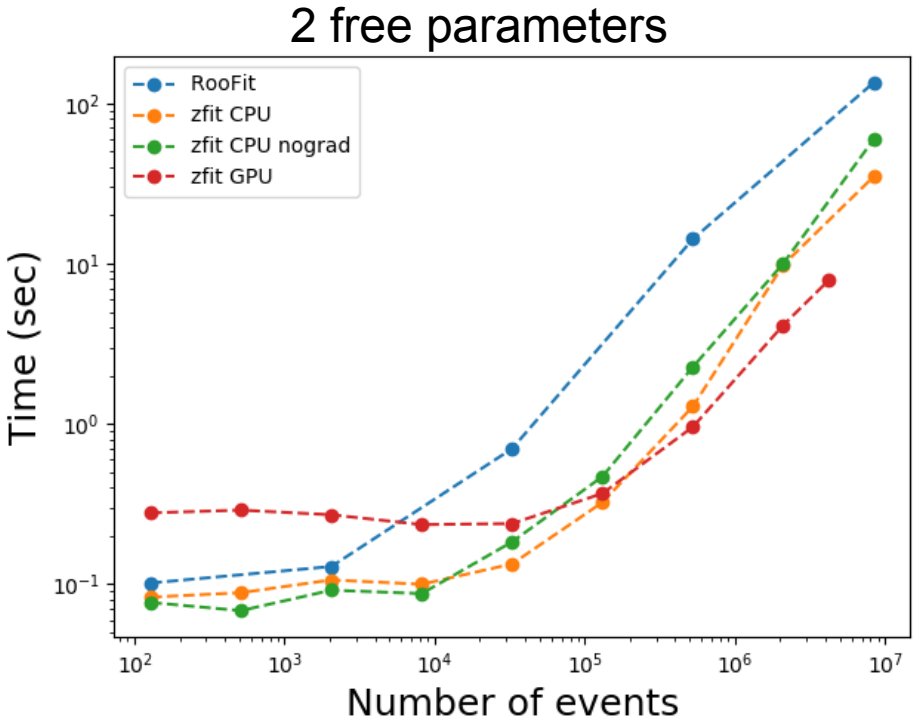
Similarity	Complicated Models	Large Data	Composed loss	Minimization	Results and uncertainties
HEP	Non-trivial functions	Whole Dataset	simultaneous, constraints	Global min, 2nd derivative algorithm	Hesse, Profiling
Deep Learning	Combine many, trivial functions	Many, small Batches	*Anything! (GANs, RL,...)*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

Modern, high performance computing

Scalability: Performance

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

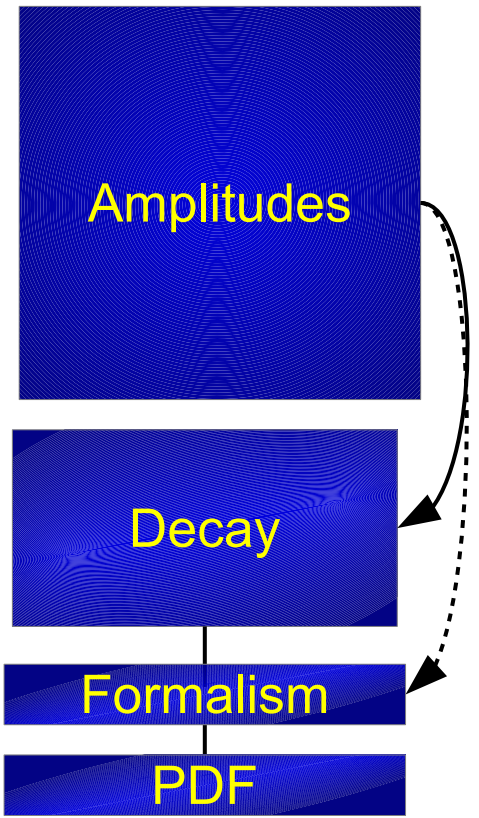




Amplitude

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)
```



Angular toys

$\mathrm{B}^0 \to K^{*0} l^+ l^-$ angular: toy study

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:
        ...
```

$\mathrm{B}^0 \to K^{*0} l^+ l^-$ angular: toys

Sensitivity study

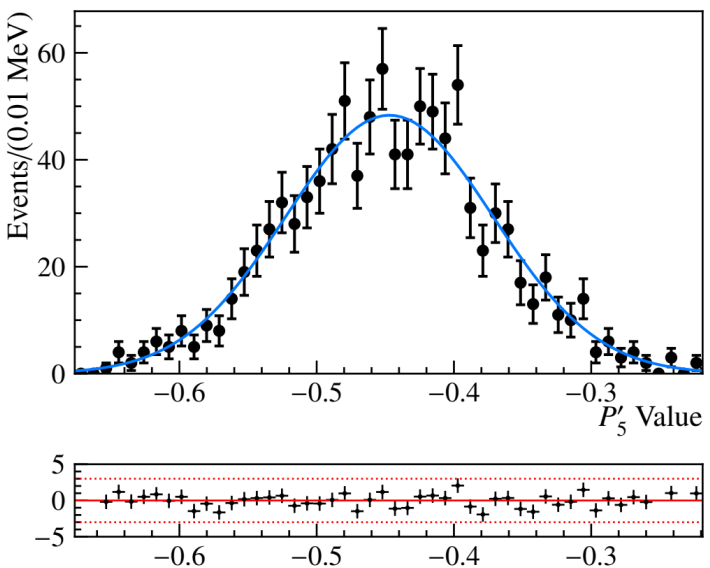
- draw toys (sample) from PDF
- Fit to sample

```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:
        ...
```

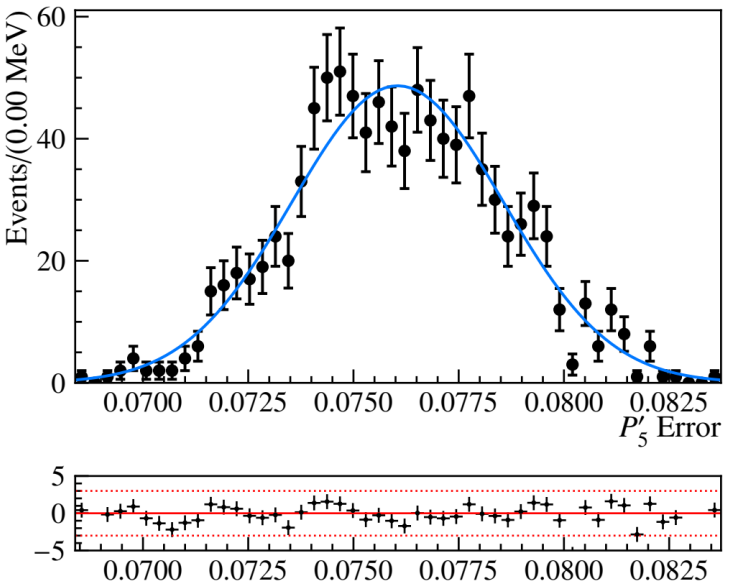
$\mathrm{B}^0 \to K^{*0} l^+ l^-$ angular: toy study

Result of toy study

P5' value



P5' error



Extending with a mass shape

```
# 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
```

Build model

$\mathrm{B}^0 \to K^{*0}\mu^+\mu^-$ full amplitude

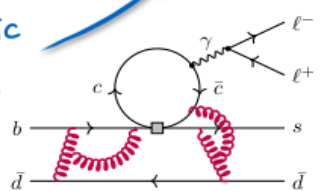
- Measuring the full differential decay ratio [1, 2]
  - Angular, q^2 distribution $\dfrac{\mathrm{d}^4\Gamma}{\mathrm{d}q^2\,\mathrm{d}\cos\theta_\ell\,\mathrm{d}\cos\theta_K\,\mathrm{d}\phi} \propto \sum J_i(q^2) f(\cos\theta_\ell, \cos\theta_K, \phi)$
  - Branching ratio information

Form Factors

$$\mathcal{A}_\lambda^{L,R} = \mathcal{N}_\lambda \left\{ \left[(\mathcal{C}_9 \pm \mathcal{C}'_9) \mp (\mathcal{C}_{10} \pm \mathcal{C}'_{10}) \right] \mathcal{F}_\lambda(q^2) + \frac{2m_b M_B}{q^2} \left[(\mathcal{C}_7 \pm \mathcal{C}'_7) \mathcal{F}_\lambda^T(q^2) - 16\pi^2 \frac{M_B}{m_b} \mathcal{H}_\lambda(q^2) \right] \right\}$$

Wilson coeff.

non-local hadronic matrix elements "charm-loop"



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,…)

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

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

Non-HEP

Scipy, lmfit, 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

```
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 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

zfit: the project

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

scalable pythonic fitting

zfit: the project

build *the* stable model fitting ecosystem for HEP

...the time has come

zfit: the project

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: API & Workflow

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

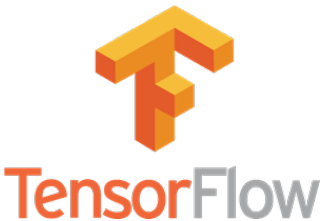
Many discussions with community
to avoid splitting/duplication

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

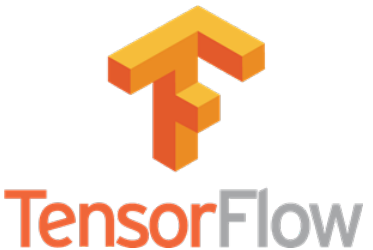
Scalable

- 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

Minimize Python function

```
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
```

From classical

to more TensorFlow

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

up to pure TensorFlow

Model building

```
obs = zfit.Space("x", limits=(-10, 10))

mu =    zfit.Parameter("mu",          1,  -4,  6)
sigma = zfit.Parameter("sigma",       1, 0.1, 10)
lambd = zfit.Parameter("lambda",     -1,  -5,  0)
frac =  zfit.Parameter("fraction", 0.5,   0,  1)

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

parameters

models

Simultaneous fit

```
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)
```

shared parameters

```
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
```

Completely equivalent