Software Tutorial

Streamlining PWA Workflow and Regularization

Lawrence Ng
Jefferson Lab

Regularization and PWA

$$\mathcal{I}(\theta, \xi)$$

Bin in mass describe only angular dependence
"Mass independent fit"

α_m "Production coefficients" in mass bin m

Intensity Distribution

θ = Angular dependence

ξ = (mass, ...) dependence

For a more detailed writeup see
GlueX Doc: 6826

Regularization and PWA

α_m Parameter vector (real/imaginary parts of amplitudes)
Dimensionality ~ (2 x $N_{\text{partial waves}}$)

$P(\alpha_m)$ We wish to infer the probability distribution

X_m Observed data samples

$P(X_m|\alpha_m) = \prod_j P(x_{m,j}|\alpha_{m,j})$ Likelihood (product of probs) of observing data { j }

$$\hat{\alpha}_m = \arg\min_{\alpha_m} \left[-\log P(X_m|\alpha_m)\right]$$

Best guess by maximizing the likelihood (or minimizing the negative log-likelihood)

If we believe some free parameters are small then we can apply pressure on the model -> Regularization

Lasso Regularization (L1 norm of the parameter vector) penalizing non-zero values

$$\hat{\theta} = \arg\min_{\theta} \left[-\log P(X|\theta) + \lambda \sum_{i=1}^{D} |\theta_i| \right]$$

So far, these concepts live in the Frequentist framework for Statistics

Alternatively we can apply pressure by multiplying the likelihood by a prior distribution

Bayesian Framework

$$P(\theta|X) \propto P(X|\theta)P(\theta)$$

Posterior Distribution | Likelihood | Prior Distribution

Normalization constant is generally difficult to evaluate as it is a high dimensional integral

$$Z = P(X) = \int P(X|\theta)P(\theta)d\theta$$

Markov Chain Monte Carlo performs monte carlo integration to approximate expectation values, i.e. marginalization

$$P(\theta_i|X) = \int P(\theta|X)d\theta_{-i} = \frac{1}{Z}\int P(\theta|X)\prod_{j\neq i} d\theta_j \approx \text{Histogram}\{\theta_i^{(s)}\}_{s=1}^{S}$$

Regularization and PWA

$$\mathcal{I}(\theta, \xi)$$

θ = Angular dependence

ξ = (mass, t, ...) dependence

Bin in mass describe only angular dependence
“Mass independent fit”

Alternatively we can describe mass dependence
“Mass dependent fit”

α_m “Production coefficients” in mass bin m

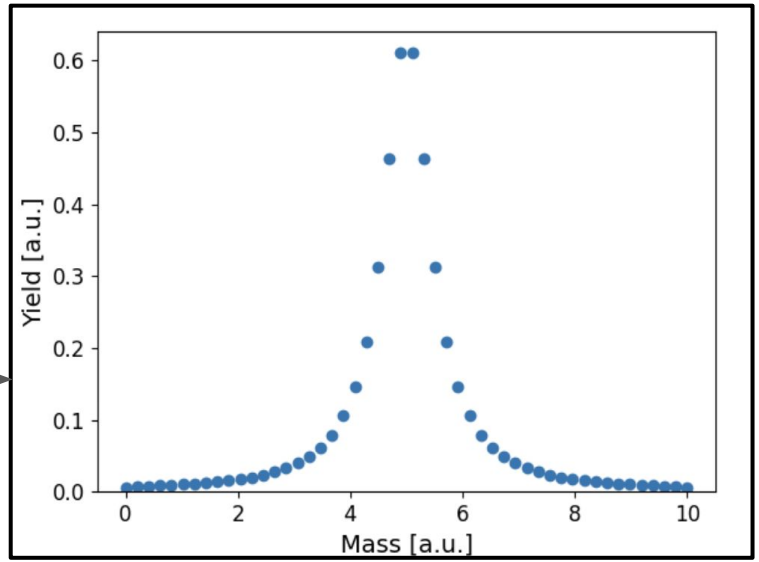
Regularization | Mass Dependent Fits

$$P(\alpha) = P(\alpha_0, \ldots, \alpha_m)$$

"mass independent" fits across all mass bins
Dimensionality ~ (M bins) x (2 x $N_{\text{partial waves}}$)

If isolated resonance than Breit-Wigner good description - needs only 2 parameters

$$\frac{1}{E - M_0 - i\Gamma_0/2}$$



More parameters needed if modeling background and approaches (M bins) x (2 x $N_{\text{partial waves}}$) if using piecewise description

Regularization | Bayesian Mass Dependence

Posterior distribution of production coefficients across all mass bins

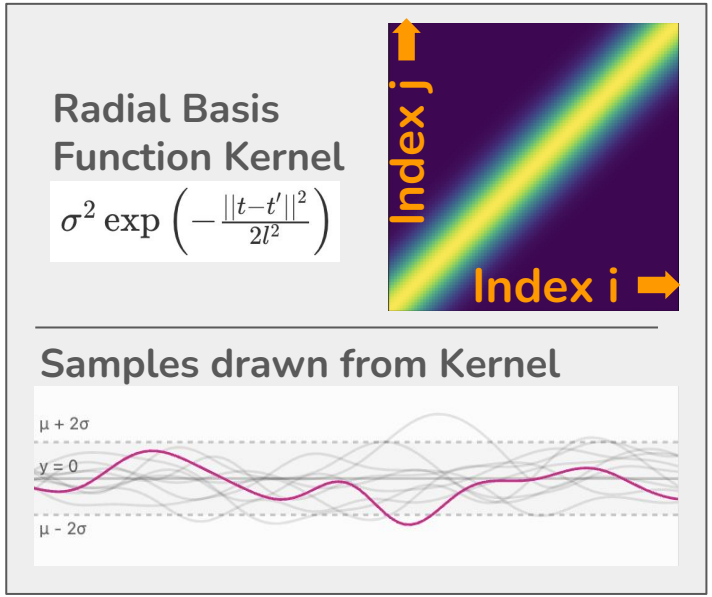
$$P(\alpha|X) \propto \prod_j P(x_j|\alpha)P(\alpha)$$

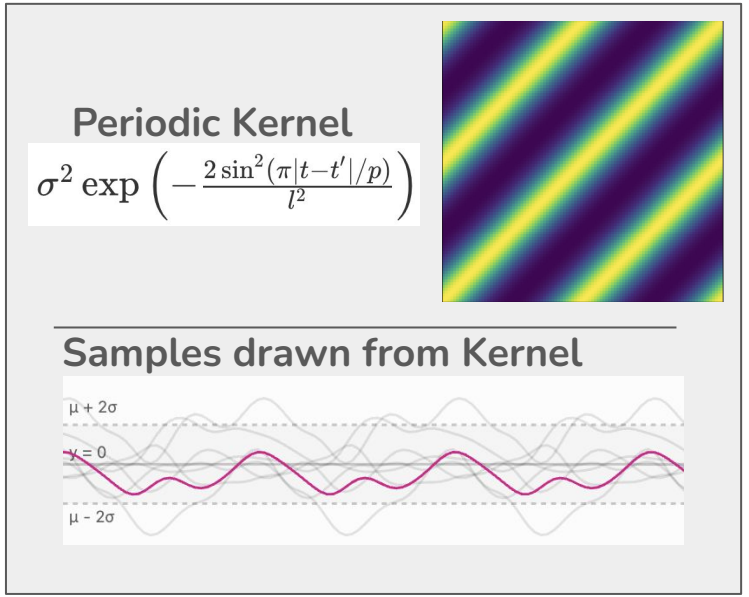
Physical Constraints

- Enforce smoothness by considering correlations between mass bins
- How might this be achieved?

Regularization | Information Field Theory

$k(\alpha, \alpha')$ Kernel function - correlation between production coefficients in a pair of kinematic bins





Distill.pub: A Visual Exploration of Gaussian Processes

Regularization | Information Field Theory

If bin spacing is uniform (By Wiener–Khinchin theorem) it is sufficient to infer only the power spectral density

Example:

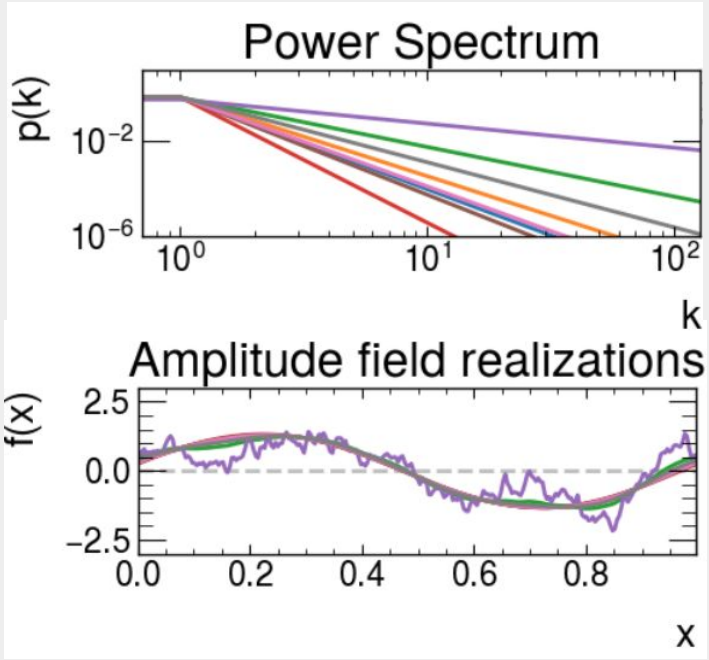
Power spectrum slope parameter

~

LogNormal(-6, 3)

Draw 8 samples

NIFTy Correlated Field Demo



Regularization | Information Field Theory

$$P(\alpha|X) \propto \prod_j P(x_j|\alpha)P(\alpha)$$

$P(\alpha)$ Smoothness enforced by the prior distribution of power spectral density for each partial wave

For a more detailed writeup see
GlueX Doc: 6826

PyAmpTools

Tutorial Link

Motivation

- Partial wave analysis can be complicated (fitting, bookkeeping, result handling, ...)
- PyAmpTools sacrifices flexibility for ease of use and focuses on Python as the core language
- **Features:**
  - Scientific Python ecosystem is massive (achieve most of the benefits of low level languages without the hassle)
  - Optimization uses gradients provided by JAX automatic differentiation (scales better than numerical diff.)
  - Hooks into several optimization frameworks spanning MLE, MCMC, to Variational Inference
    - iminuit, scipy, numpyro, iftpwa ← Information Field Theory framework
  - YAML file is used to configure the analysis for consistency and automation
  - Fitting is generally done through the command line

Loading the container

On the JLab farm type this in

```
apptainer exec --contain --writable-tmpfs \
    --bind /my/working/directory \
    --bind /scratch \
    --bind ~/.cache/fontconfig \
    --env BASH_ENV=/dev/null \
    /w/halld-scshelf2101/lng/WORK/PyAmpTools/pyamptools.sif bash
```

```
source /etc/bash.bashrc
```

PyAmpTools| Command line

- Fitting is generally done through the command line

```
bash$ pa -h
```

```
Dispatch pyamptools commands. Select a command from the Commands section below. Remaining arguments will be passed to
the selected command.

optional arguments:
  -h, --help   show this help message and exit
  -f, --files  show command file locations and exit


Commands:
  * nentries                Print entries in List[ROOT files] (* wildcard), flag for branch integration
  * fit                     (AmpTools) Perform a set of MLE fits given an amptools config file
  * fitfrac                 (AmpTools) Extract fit fractions from a given amptools FitResults file
  * gen_amp                 (AmpTools) Generate data for a given configuration file
  * gen_vec_ps              (AmpTools) Generate vector-pseduoscalar data for a given configuration file
  * ift_pkl_summary         Summarize the contents of an IFT results pickle file
  * calc_ps                 [In Development] Calculate the phase space factor for IFT fits

  ==== YAML based commands below (takes a YAML file argument to configure setup) ====
  * from_default            Copy default yaml to a given location
  * run_priorSim            Draw sample from NIFTy prior, generate simulated data, and split into kinematic bins
  * run_cfgGen              (AmpTools) Generate an AmpTools fit configuration file
  * run_divideData          Divide data into kinematic bins (separate folders)
  * run_processEvents       (AmpTools) Process bins: dump AmpVecs + NormInts to pkl files
  * run_fit                 (AmpTools) Run MLE fits over kinematic bins using AmpTools
  * run_mle                 Run MLE fits over kinematic bins using variety of optimizers (minuit, lbfgs, ...)
  * run_mcmc                Run MCMC fits over kinematic bins using numpyro NUTS sampler
  * run_ift                 Run IFT fit over kinematic bins
  * run_momentInverter      Run moment inverter
  * run_resultMan           Run result manager commands, make all default plots
  * dash_ift_cmp            Compare multiple IFT fits (intensity and phase) using dash package
```

YAML Commands of form
`pa run_ift main.yaml`

Generate/Format Data

Fit the Data

Plot the data

YAML file contains all the knobs

```yaml
defaults_location: null
base_directory: /w/halld-scshelf2101/lng/WORK/PyAmpTools9/demos/RESULTS # base directory for
data_folder: ${base_directory}/DATA_SOURCES # folder for data sources that will be binned
n_processes: 4 # global number of processes to generally use
polarizations:
  "000": 1.0 # polarization magnitude in each orientation
waveset: Sp0+_Sp0-_Dp2+_Dp2- # underscore separated list of waves to use
phase_reference: Sp0+_Sp0- # reference wave in each reflectivity sector
reaction: Beam Proton Pi0 Eta # amptools reaction scheme
daughters: # Daughter masses
  Pi0: 0.135
  Eta: 0.548
min_mass: 1.04 # minimum mass to consider
max_mass: 1.72 # maximum mass to consider
n_mass_bins: 17 # number of mass bins to use
min_t: 0.0 # minimum t to consider
max_t: 1.0 # maximum t to consider
n_t_bins: 1 # number of t bins to use
acceptance_correct: true # whether to apply acceptance corrections
datareader: ROOTDataReader # data reader to use
coordinate_system: cartesian # ['cartesian', 'polar'], dont use polar
bins_per_group: 1 # create group dirs grouping bins allowing nifty to handle finer bins
merge_grouped_trees: true # remerge the trees in each group
constrain_grouped_production: false # if not remerging, we can choose to constrain amplitude
real_waves: "" # same form as waveset, define which waves are purely real
fixed_waves: "" # same form as waveset, define which waves are fixed
add_amp_factor: "" # Add an amplitude factor to every amplitude. For example, OmegaDalitz 0.
append_to_decay: "" # append this string to the decay amplitude, i.e. 'omega3pi' can be appe
append_to_cfg: "" # append this string to the AmpTools configuration file
```

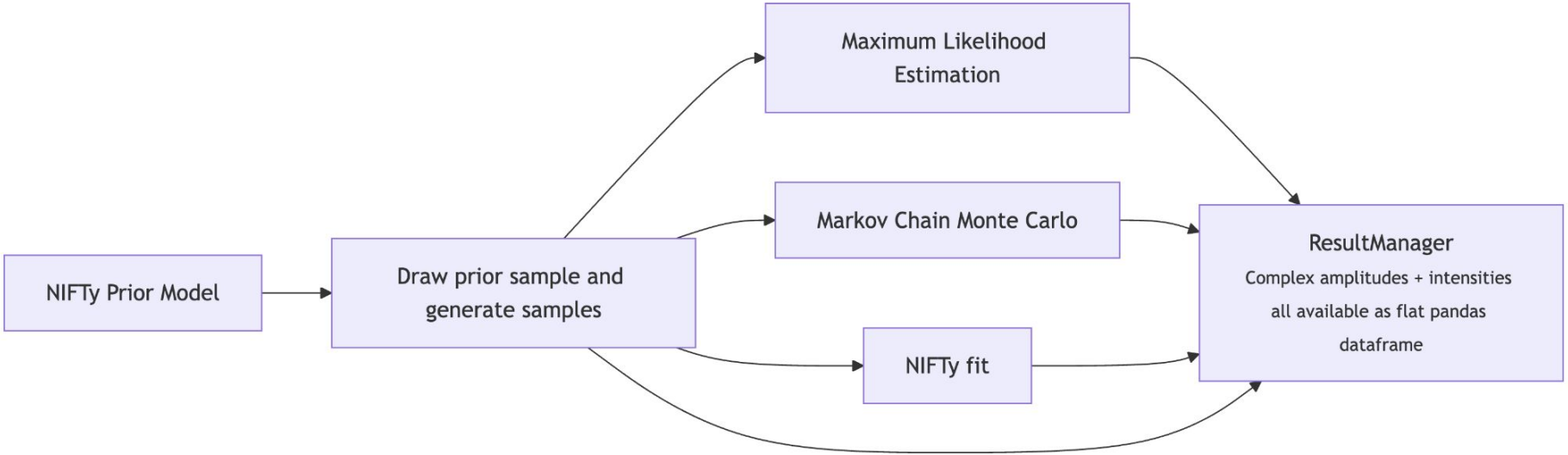
For vector pseudoscalar just change waves to another format:
1Sp1-
1Sp1+

Also see Here

Lots more knobs not shown but defaults can be pulled and most work decently well (of course you need dataset, waveset, binning, ...)

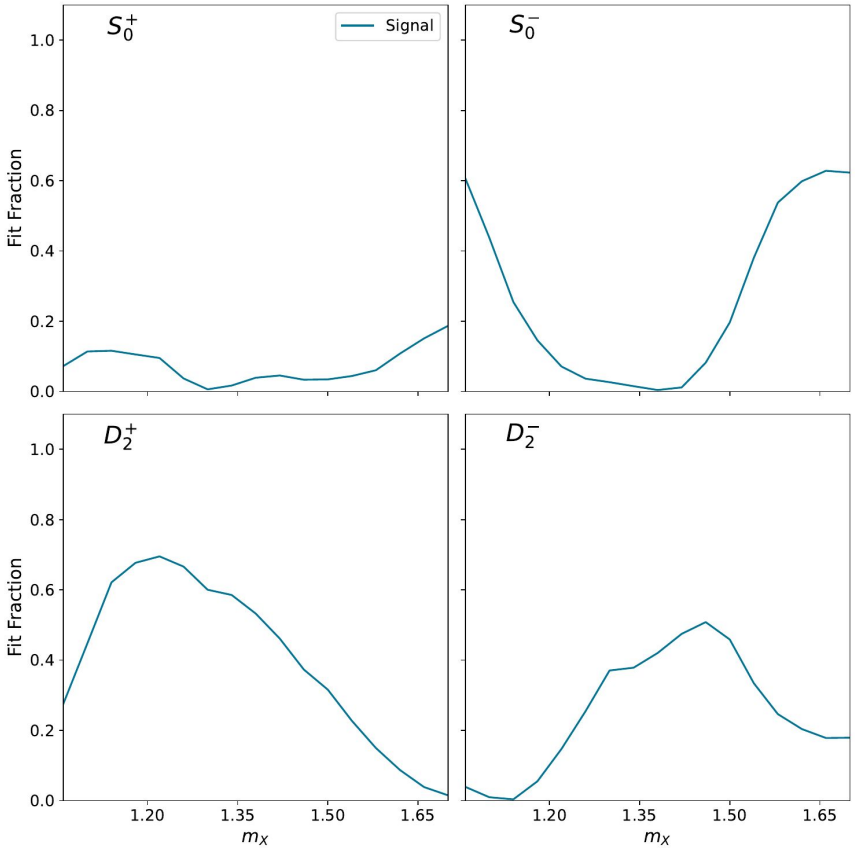
Tutorial workflow

Since IFT model is a Bayesian framework we can draw samples from the prior and generate MC simulations to rapidly iterate on Input/Output studies and compare optimization frameworks with this toolkit



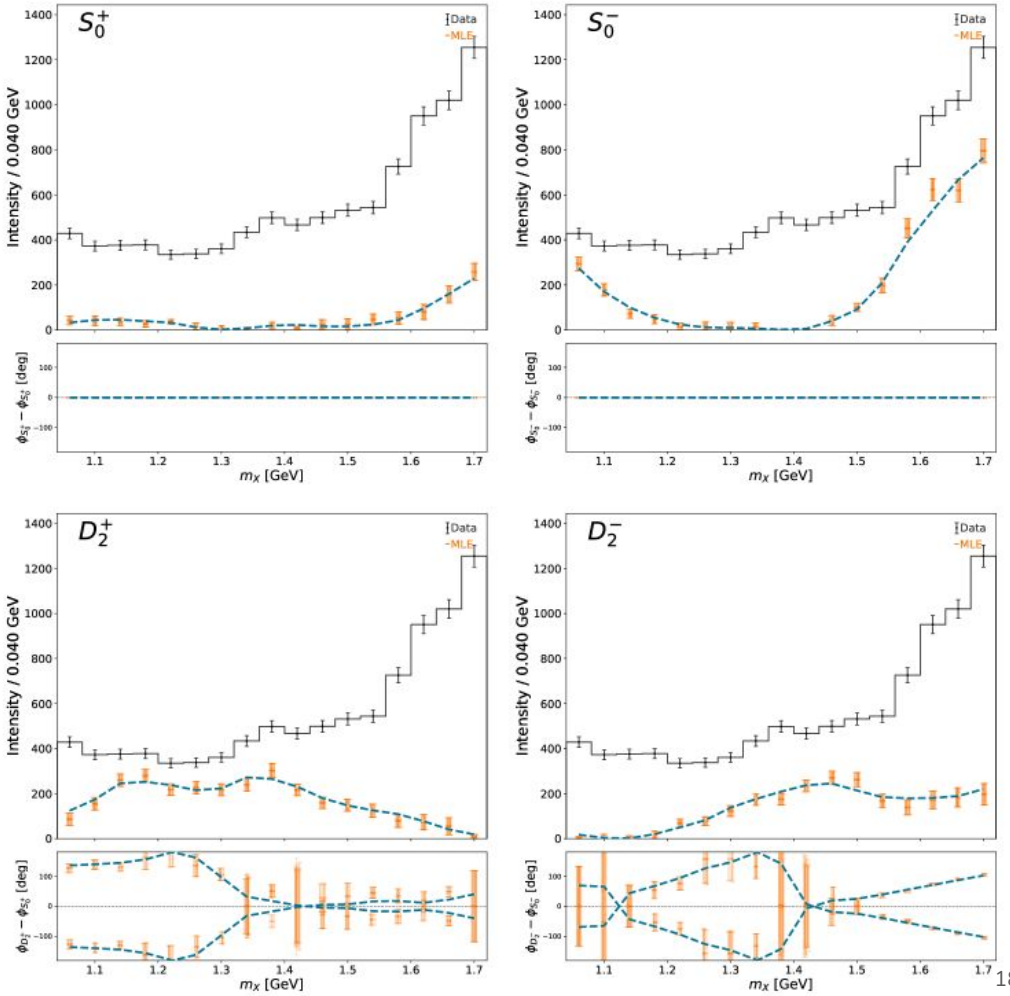
Draw prior sample and generate samples

```
pa run_priorSim <yaml_file>
```



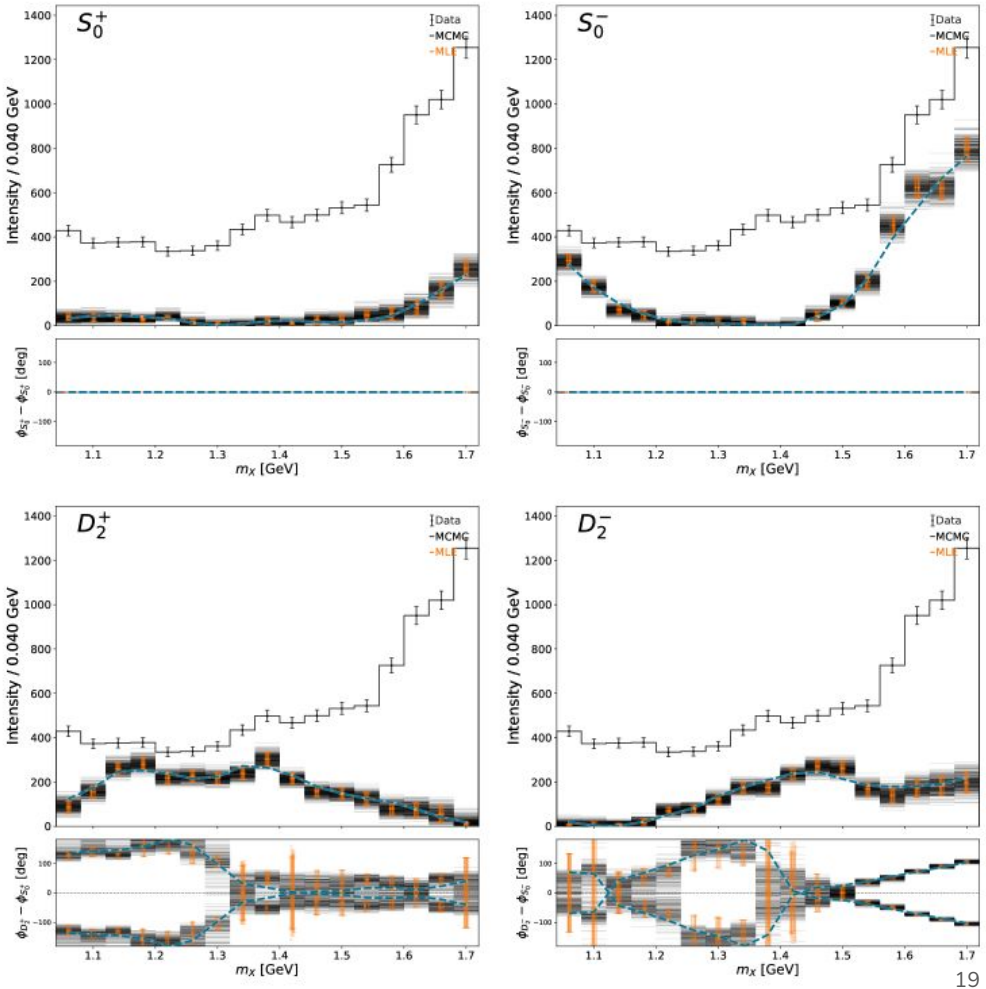
Maximum Likelihood Estimation

```
pa run_mle <yaml_file>
```



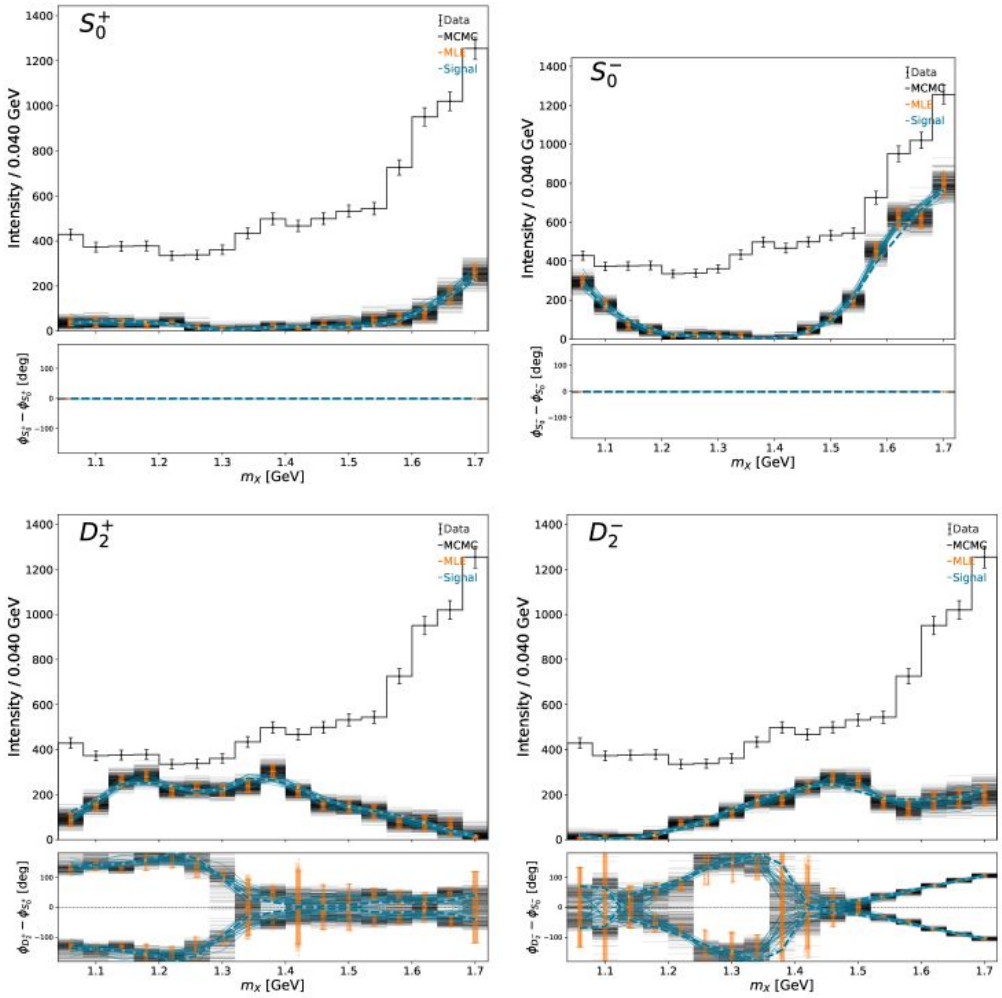
Markov Chain Monte Carlo

```
pa run_mcmc <yaml_file>
```



NIFTy fit

```
pa run_ift <yaml_file>
```



ResultManager

Complex amplitudes + intensities all available as flat pandas dataframe

```
from pyamptools.utility.resultManager import ResultManager
yaml_file = "/w/halld-scshelf2101/lng/WORK/PyAmpTools9/OTHE
resultManager = ResultManager(yaml_file)
resultManager.attempt_load_all()
```

All results loaded as a flat pandas dataframe (intensities, complex amplitudes, moments , etc…)

```
resultManager.mle_results.head(1)
```

✓ 0.1s Python

	mass	initial_likelihood	likelihood	sample	Sp0+_amp	Sp0-_amp	Dp2+_amp	Dp2-_amp	intensity	intensity_err	...	H2(1,1)	H2(2,1)	H2(2,2)	H2(3,1)	H2(3,2)	H
0	1.06	658.275784	361.181041	8	20.106399+ 0.000000j	59.770623+ 0.000000j	-25.079330+21.531924j	3.806061- 16.484035j	424.992813	29.15439	...	0.0	0.0	0.061104	0.0	0.0	

1 rows × 79 columns

+ Code + Markdown

```
resultManager.mcmc_results.head(1)
```

✓ 0.1s Python

	intensity	Sp0+	Sp0-	Dp2+	Dp2-	Sp0+_amp	Sp0-_amp	Dp2+_amp	Dp2-_amp	mass	...	H2(1,1)	H2(2,1)	H2(2,2)	H2(3,1)	H2(3,2)	H2(3,3)	H2(4,1)
0	447.726497	9.040142	299.052373	116.342458	23.328655	10.63292+ 0.00000j	61.450239+ 0.000000j	-37.406014- 8.087889j	3.974760- 16.600252j	1.06	...	0.0	0.0	0.050858	0.0	0.0	0.0	0.0

1 rows × 54 columns

```
resultManager.ift_results[0].head(1)
```

✓ 0.1s Python

	tprime	mass	sample	Sp0+_amp	Sp0-_amp	Dp2+_amp	Dp2-_amp	Sp0+	Sp0-	Dp2+	...	H2(1,1)	H2(2,1)	H2(2,2)	H2(3,1)	H2(3,2)	H2(3,3)	H2(4,1)
0	0.5	1.06	0	18.905427- 0.000000j	60.91196- 0.00000j	-32.059605- 13.029281j	1.409181+22.108157j	28.578744	293.836164	95.129854	...	0.0	0.0	0.05376	0.0	0.0	0.0	0.0

1 rows × 55 columns

Have time? Click this

Tutorial Link