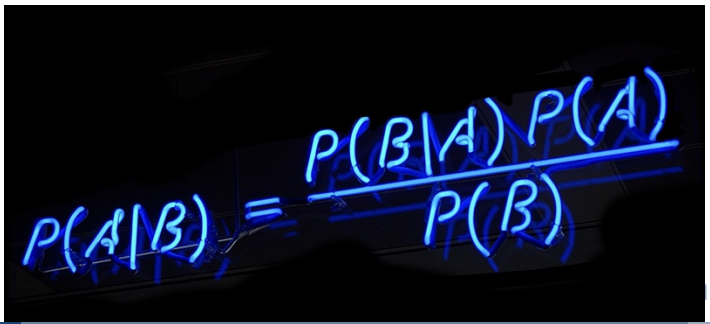
Statistics in Data Analysis

All you ever wanted to know about statistics but never dared to ask

part 7

Pawel Brückman de Renstrom
(pawel.bruckman@ifj.edu.pl)

April 15, 2026



Putting it to work...

1D fit of the signal yield

N data events have been collected in an experiment yielding a scalar random variable x:

- The sample consists of a mixture of *signal* and *background* events with known p.d.f.'s.
- Background p.d.f.: $f_B(x) = \frac{1}{\tau} e^{-x/\tau}$, $\tau = 5$,
- Signal p.d.f.: $f_S(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x-\mu)^2/2\sigma^2}$, $\mu = 10$, $\sigma = 3$,
- The varaible x has been recorded in the range $(0, 30)$.
- No assumption about background yield can be made: $N = N_S + N_B$.

Our task is to:

1. Estimate number of signal events N_S in the observed sample,
2. assess the error of the N_S estimate from the $\log L$ or χ^2 profile.

MIND: This is NOT an extended fit.

Putting it to work...

1D fit of the signal yield

We shall use three strategies:

1. the *Unbinned Maximum Likelihood* fit:
   `Unbinned ML Python notebook template in Colab`
2. the *Binned Maximum Likelihood* fit; 10 bins over (0, 30):
   `Binned ML Python notebook template in Colab`
3. the *Binned Least Squares* (NOT modified) fit; 10 bins over (0, 30):
   `Binned LS Python notebook template in Colab`

and four data samples:

1. `pickled data sample 1 from GitHub`
2. `pickled data sample 2 from GitHub`
3. `pickled data sample 3 from GitHub`
4. `pickled data sample 4 from GitHub`

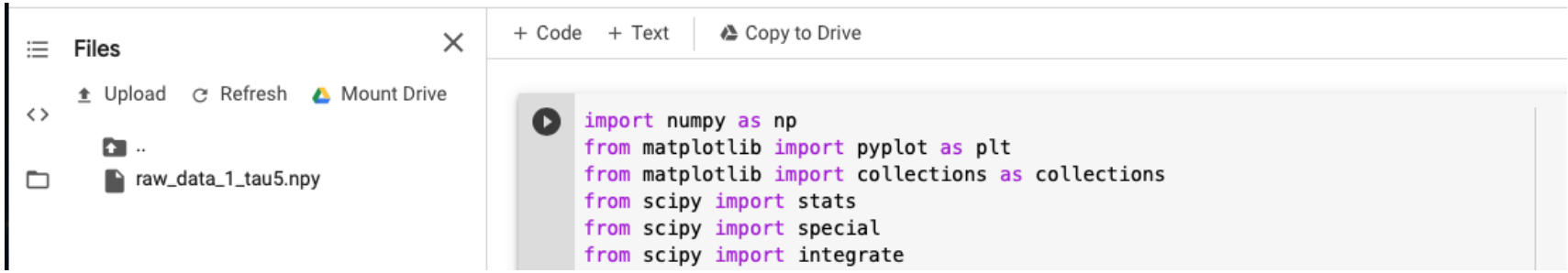
All shall be executed on the *Google Colaboratory* platform.

Detailed instructions

$\longrightarrow$ Click on one of the Python notebook links in order to open it in *Google Colaboratory*.

$\longrightarrow$ Click to download the assigned dataset file from GitHub.

$\longrightarrow$ Click on one of the *Files* icon on the left bar of your *Colab* interface. If you cannot see any datafiles, click on the *Upload* button and select previously downloaded file. As a result you should see:



Hints part 1: fit the N_s, $\log L$ or χ^2 function

$\longrightarrow$ You need p.d.f. normalization factors in the range (0,30). For this purpose calculate `scales` and `scaleb` just as it is done in the main part of the Python script. You will need `xmin`, `xmax`, `tau0`, `mu0` and `sigma0`.

$\longrightarrow$ You need the total number of collected events. For the unbinned ML this is just the length of the `data` vector (`len(data)`). For the binned methods loop over the `hist` array and sum all entries. `nbins=len(hist)` gives you the number of bins.

$\longrightarrow$ For the unbinned ML you need to loop over the `data` array, for each entry calculate the normalized p.d.f.'s (`gauss` & `decay`) and acumulate $\log L$ according to Eq. (25) of lecture 4 and using the combined S+B p.d.f.

$\longrightarrow$ For the binned methods you need to loop over the bins, the `hist` array (e.g. `for k in range(nbins)`). You need to get the prediction for the bin by integrating the normalized p.d.f., see Eq. (11) of lecture 5. Bin k is delimitted by `binsy[k]` and `binsy[k+1]`.

$\longrightarrow$ For the binned ML increment the $\log L$ using Eq. (13) of lecture 5 and the combined S+B p.d.f.

$\longrightarrow$ For the binned LS increment the χ^2 using Eq. (36) of lecture 5 and the combined S+B p.d.f.

$\longrightarrow$ NOTE: We are fitting just one free parameter, N_S (coded as `mus`). Make sure you properly define the combined S+B p.d.f. using N_S and the total number of collected events.

Hints part 2: estimate the error on N_s using $\log L$ or χ^2 profile

$\longrightarrow$ The code provides you with the `pl` & `ll_array` arrays which contain the estimated N_S and the corresponding value of either $\log L$ or χ^2, respectively.
$\longrightarrow$ Your task is to find values of N_S corresponding to $+1\sigma$ and -1σ about the fitted value (coded as `sigma_neg` & `sigma_pos`).
$\longrightarrow$ For the purpose, recall Eq. (7) and Eq. (32) of lecture 5.

Example solution

Unbinned Maximum Likelihood

$\log L$ function definition:

```
def logL(mus, data, xmin=0, xmax=99, tau0=5.0, mu0=10.0, sigma0=3.0):
    temp_ll = 0
    integrals = integrate.quad(gauss, xmin, xmax, args=(mu0, sigma0,))
    scales = 1/integrals[0]
    integralb = integrate.quad(decay, xmin, xmax, args=(tau0,))
    scaleb = 1/integralb[0]
    N = len(data)
    for k in data:
        ps = scales*gauss(k, mu0, sigma0)
        pb = scaleb*decay(k, tau0)
        temp_ll += -np.log((mus*ps + (N-mus)*pb)/N)
    return temp_ll
```

Uncertainties from the $\log L$ profile:

```
#  Estimate the error:
pl = np.linspace(plow, phig, ndistr)
ll_array = []
for xx in pl:
   ll_array.append(logL(xx, rndy, 0.0, xrange, tau, mu, sigma))
min_val = min(ll_array)
min_val_index = ll_array.index(min_val)
min_val_location = pl[min_val_index]
# search the -maxML+0.5:
i = min_val_index
while (i<ndistr-1 and ll_array[i] <= min_val + 0.5):
    i += 1
sigma_pos = pl[i]
i = min_val_index
while (i>0 and ll_array[i] <= min_val + 0.5):
    i += -1
sigma_neg = pl[i]
print("Profile extremum = %2.3f"%(min_val_location, ))
print("-sigma = %2.3f,  +sigma = %2.3f"%(sigma_neg, sigma_pos))
```

Results:

DS 1: $N_S = 21.52 + 7.82 - 7.47$

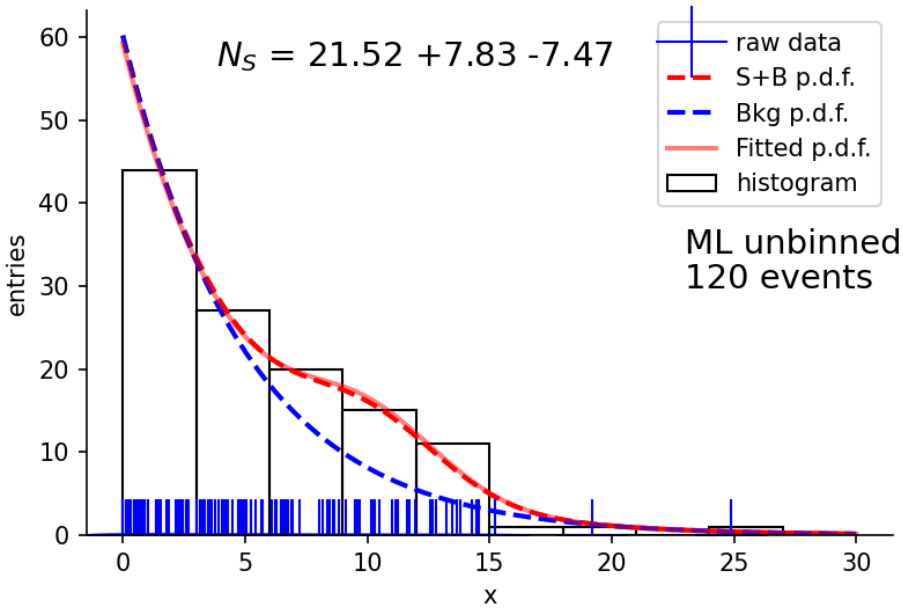
DS 2: $N_S = 29.99 + 7.93 - 7.69$

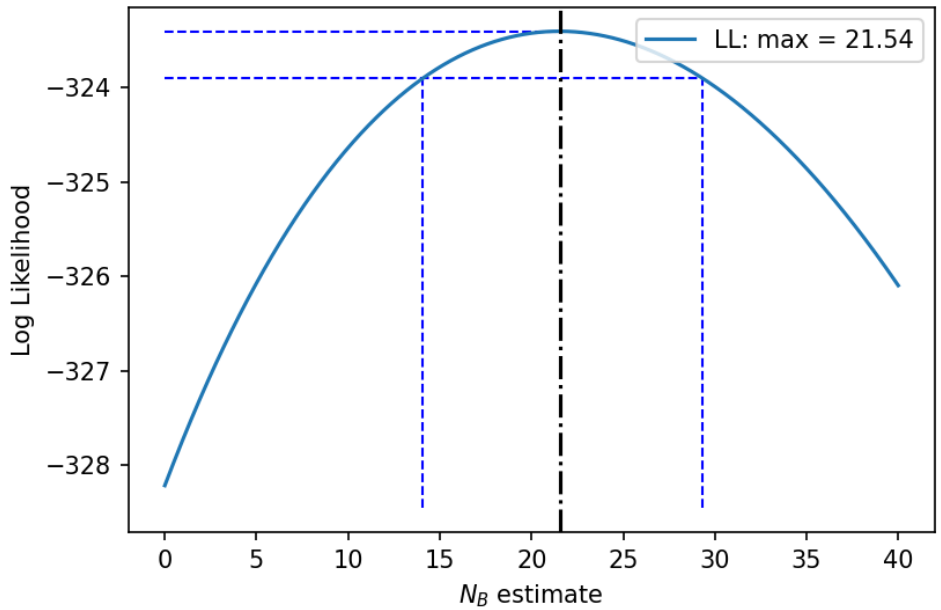
DS 3: $N_S = 18.29 + 7.69 - 7.32$

DS 4: $N_S = 21.09 + 7.94 - 7.55$

Example solution

Unbinned Maximum Likelihood





Fit uncertainty obtained from the $\log L$ profile at $\log L(N_S \pm \sigma_{N_S}) = \log L_{\max} - \frac{1}{2}$.

Example solution

Binned Maximum Likelihood

$\log L$ function definition:

```
def logLB(mus, hist, bins, xmin=0, xmax=99, tau0=5.0, mu0=10.0, sigma0=3.0):
    temp_ll = 0
    integrals = integrate.quad(gauss, xmin, xmax, args=(mu0, sigma0,))
    scales = 1/integrals[0]
    integralb = integrate.quad(decay, xmin, xmax, args=(tau0,))
    scaleb = 1/integralb[0]
    nbins = len(hist)
    N = 0
    for k in range(nbins):
        N += hist[k]
    for k in range(nbins):
        ps = scales*(integrate.quad(gauss, bins[k], bins[k+1], args=(mu0, sigma0,)))[0]
        pb = scaleb*(integrate.quad(decay, bins[k], bins[k+1], args=(tau0,)))[0]
        binexp  = mus*(ps-pb) + N*pb
        temp_ll += -hist[k]*np.log(binexp)
    return temp_ll
```

Uncertainties from the $\log L$ profile:

```
#  Estimate the error:
pl = np.linspace(plow, phig, ndistr)
ll_array = []
for xx in pl:
   ll_array.append(logL(xx, rndy, 0.0, xrange, tau, mu, sigma))
min_val = min(ll_array)
min_val_index = ll_array.index(min_val)
min_val_location = pl[min_val_index]
# search the -maxML+0.5:
i = min_val_index
while (i<ndistr-1 and ll_array[i] <= min_val + 0.5):
    i += 1
sigma_pos = pl[i]
i = min_val_index
while (i>0 and ll_array[i] <= min_val + 0.5):
    i += -1
sigma_neg = pl[i]
print("Profile extremum = %2.3f"%(min_val_location, ))
print("-sigma = %2.3f,  +sigma = %2.3f"%(sigma_neg, sigma_pos))
```

Results:

DS 1: $N_S = 23.73 + 8.07 - 7.75$

DS 2: $N_S = 29.90 + 8.05 - 7.80$

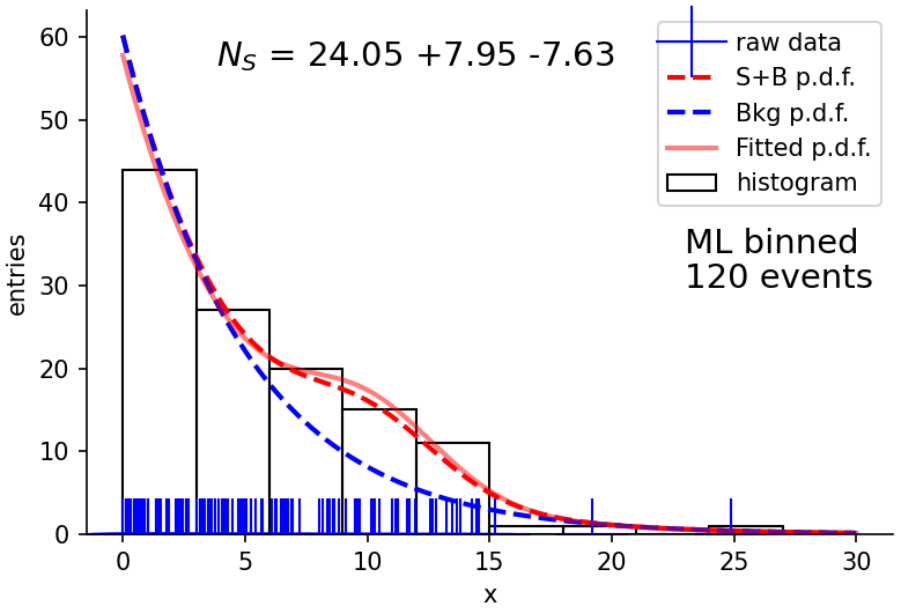
DS 3: $N_S = 17.45 + 8.02 - 7.60$

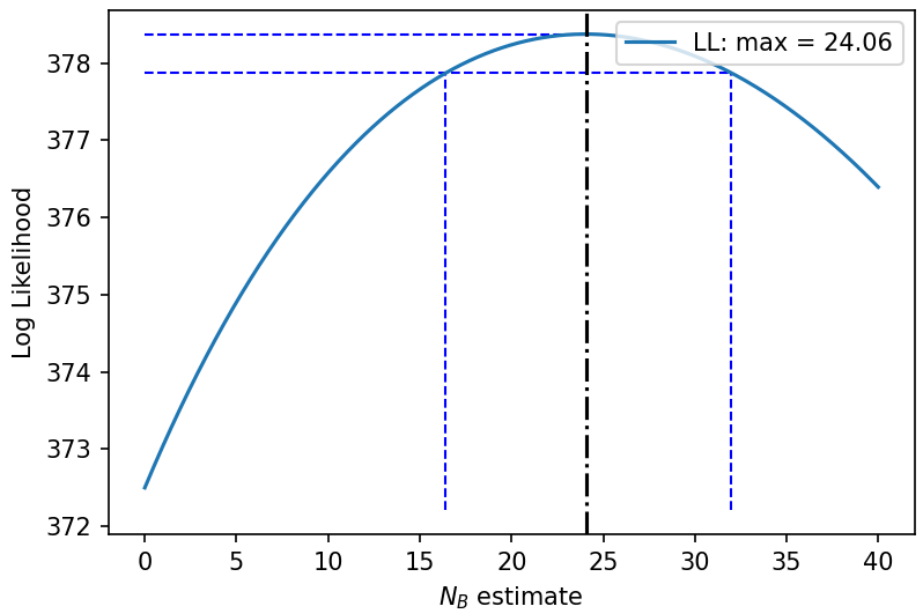
DS 4: $N_S = 22.76 + 8.19 - 7.86$

Example solution

Binned Maximum Likelihood

10 bins:



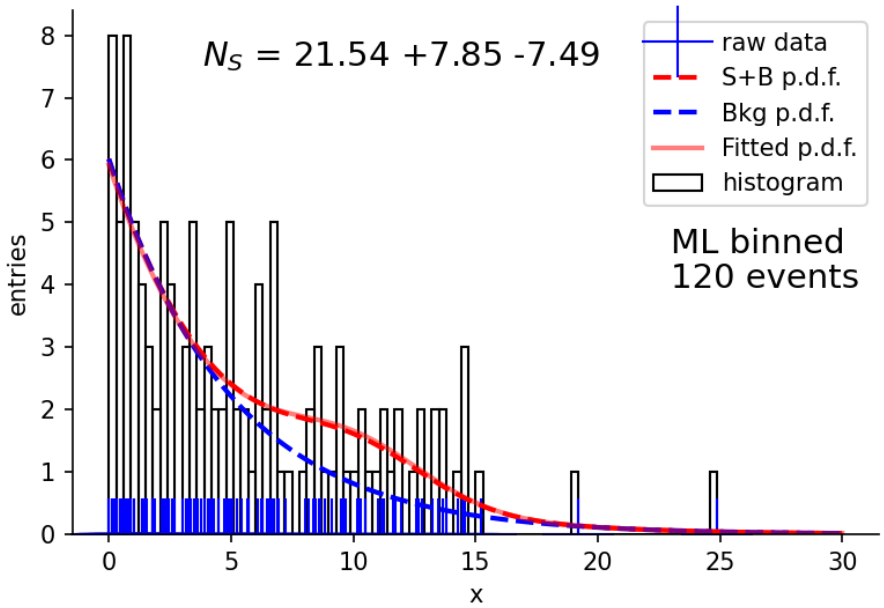


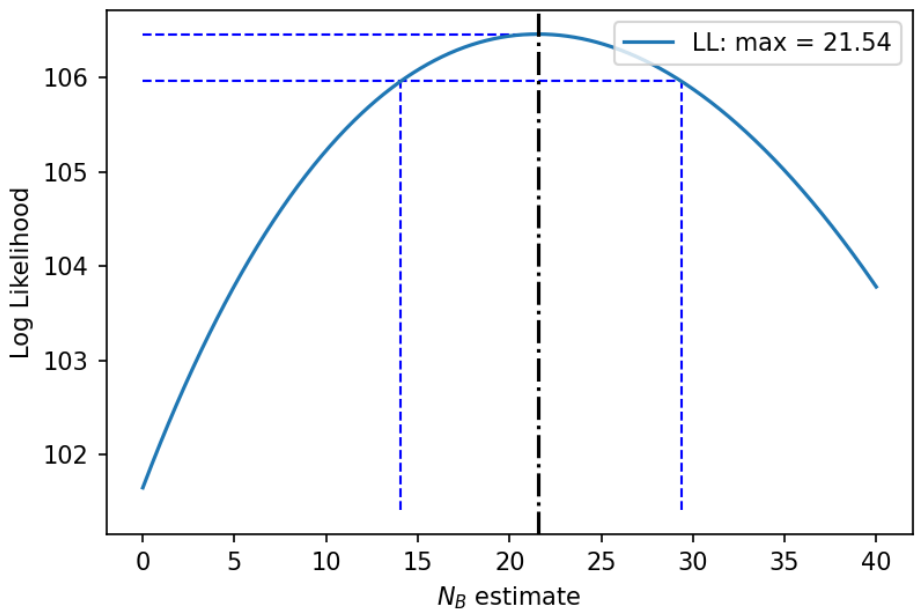
Fit uncertainty obtained from the $\log L$ profile at $\log L(N_S \pm \sigma_{N_S}) = \log L_{\max} - \frac{1}{2}$.

Example solution

Binned Maximum Likelihood

100 bins (for very fine binning result converges to the unbinned one!):





Fit uncertainty obtained from the $\log L$ profile at $\log L(N_S \pm \sigma_{N_S}) = \log L_{\max} - \frac{1}{2}$.

Example solution

Binned Least Squares

χ^2 function definition:

```
def chi2(mus, hist, bins, xmin=0, xmax=99, tau0=5.0, mu0=10.0, sigma0=3.0):
    temp_x2 = 0
    integrals = integrate.quad(gauss, xmin, xmax, args=(mu0, sigma0,))
    scales = 1/integrals[0]
    integralb = integrate.quad(decay, xmin, xmax, args=(tau0,))
    scaleb = 1/integralb[0]
    nbins = len(hist)
    N = 0
    for k in range(nbins):
        N += hist[k]
    for k in range(nbins):
        ps = scales*(integrate.quad(gauss, bins[k], bins[k+1], args=(mu0, sigma0,)))[0]
        pb = scaleb*(integrate.quad(decay, bins[k], bins[k+1], args=(tau0,)))[0]
        binexp  = mus*(ps-pb) + N*pb
        temp_x2 += (hist[k]-binexp)*(hist[k]-binexp)/binexp
        #temp_x2 += (hist[k]-binexp)*(hist[k]-binexp)/hist[k]
    return temp_x2
```

Uncertainties from the χ^2 profile:

```
#  Estimate the error:
pl = np.linspace(plow, phig, ndistr)
ll_array = []
for xx in pl:
   ll_array.append(chi2(xx, hy, binsy, 0.0, xrange, tau, mu, sigma))
min_val = min(ll_array)
min_val_index = ll_array.index(min_val)
min_val_location = pl[min_val_index]
# search the minLS+1:
i = min_val_index
while (i<ndistr-1 and ll_array[i] <= min_val + 1.0):
    i += 1
sigma_pos = pl[i]
i = min_val_index
while (i>0 and ll_array[i] <= min_val + 1.0):
    i += -1
sigma_neg = pl[i]
print("Profile extremum = %2.3f"%(min_val_location, ))
print("-sigma = %2.3f,  +sigma = %2.3f"%(sigma_neg, sigma_pos))
```

Results:

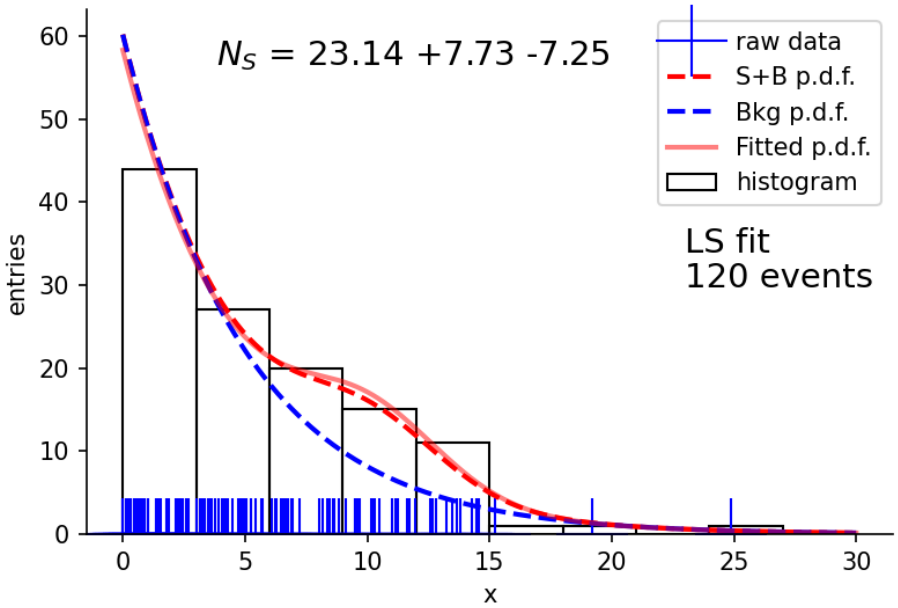
DS 1: $N_S = 23.74 + 7.93 - 7.49$
DS 2: $N_S = 28.32 + 7.40 - 7.05$

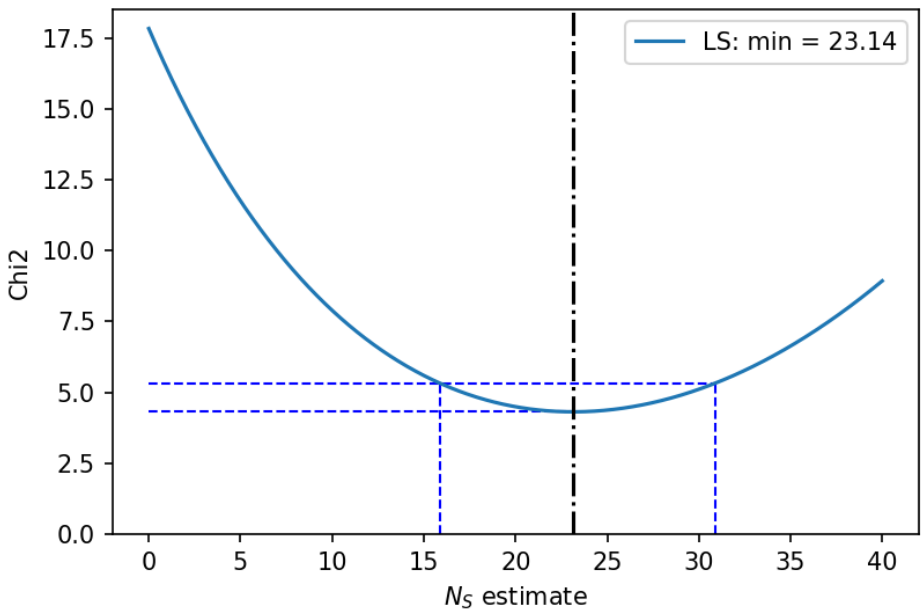
DS 3: $N_S = 17.11 + 7.95 - 7.42$
DS 4: $N_S = 23.04 + 8.19 - 7.67$

Example solution

Binned Least Squares

10 bins:



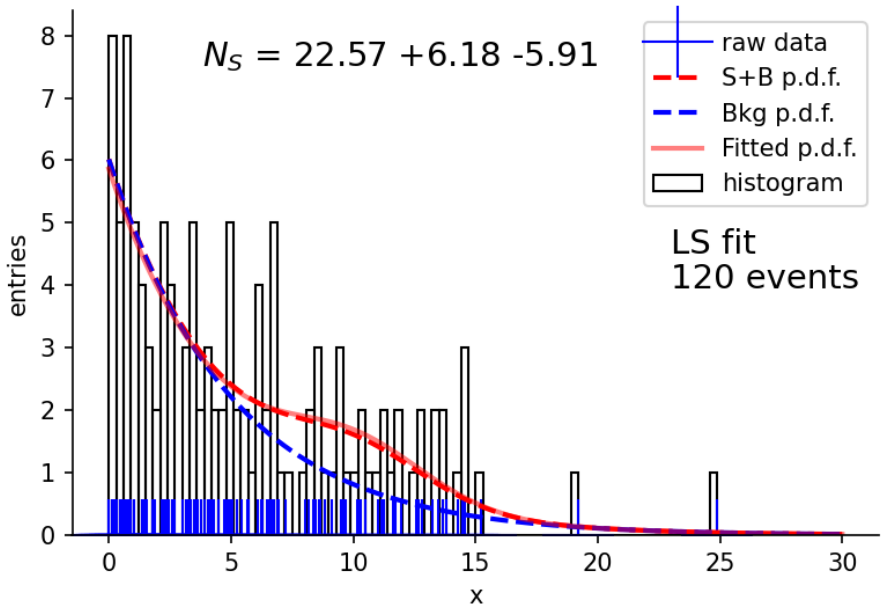


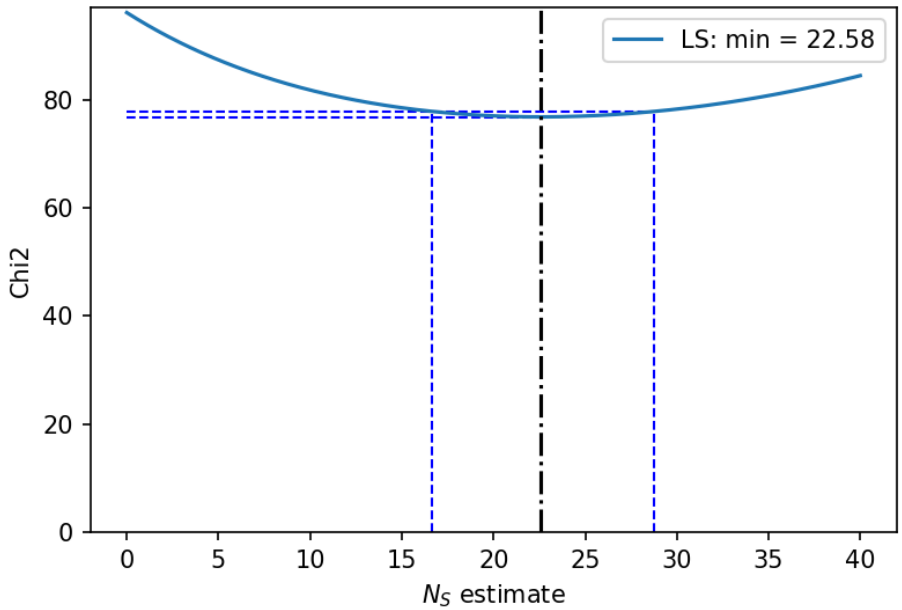
Fit uncertainty obtained from the χ^2 profile at $\chi^2(N_S \pm \sigma_{N_S}) = \chi^2_{\text{min}} + 1$.

Example solution

Binned Least Squares

100 bins (cannot reproduce the ML result exactly):
Small and empty bins make the uncertainty estimation less reliable.





Fit uncertainty obtained from the χ^2 profile at $\chi^2(N_S \pm \sigma_{N_S}) = \chi^2_{\min} + 1$.

Variable transformation

(revisited)

Let y be a function of a random variable $y = a(x)$ (or variables) which itself is a random variable.

x is distributed according to p.d.f. $f(x)$ How do we find $g(y)$, the p.d.f. of y?

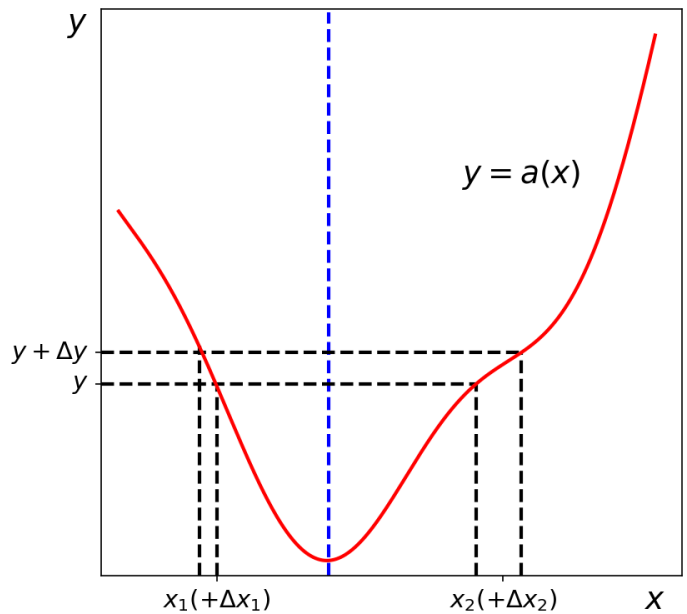
$$\Delta P = \int_{x' \in \{x, x+\Delta x\}} f(x')dx' = \int_{y' \in \{a(x), a(x+\Delta x)\}} g(y')dy' \tag{1}$$

Irregardles of the sign of the derivative we have:

$$\Delta x \to 0 \Rightarrow f(x)|dx| = g(y)|dy|$$

$$g(y) = f(x(y)) \left|\frac{dx}{dy}\right| = f(x(y))|a'(x))|^{-1} \tag{2}$$

where $x(y) \equiv a^{-1}(y)$.



If a not single-valued, contributions must be added:

$$g(y) = \sum_i f(x(y))_i \left|\frac{dx}{dy}\right|_i = \sum_i f(x(y))_i |a'(x_i))|^{-1} \tag{3}$$

Variable transformation

(revisited)

In a more general case of multidimensional joint p.d.f. one can define a one-to-one (invertable) transformation. We have:

$$g(y_1, ..., y_n) = f(x_1, ..., x_n)\,|J| \tag{4}$$

$$\text{where} \quad J \equiv \det \begin{pmatrix} \frac{\partial x_1}{\partial y_1} & \frac{\partial x_1}{\partial y_2} & \cdots & \frac{\partial x_1}{\partial y_n} \\ \frac{\partial x_2}{\partial y_1} & \cdots & \cdots & \frac{\partial x_2}{\partial y_n} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial x_n}{\partial y_1} & \frac{\partial x_n}{\partial y_2} & \cdots & \frac{\partial x_n}{\partial y_n} \end{pmatrix},$$

$$x_i(y_1, ..., y_n) \equiv a_i^{-1}(y_1, ..., y_n).$$

The Monte Carlo method

Cumulative function & uniform distribution $[0, 1]$

- Let y be the cumulative of an arbitray p.d.f. $f(x)$:

$$y = F(x) = \int_{-\infty}^{x} f(x')\mathrm{d}x' \tag{5}$$

- $g(y)$, the p.d.f. of y is given by the transformation:

$$g(y) = f(x(y)) \left| \frac{1}{F'(x)} \right| = f(x(y)) \frac{1}{f(x(y))} = \begin{cases} 1 \text{ for } y \in (0, 1) \\ 0 \text{ otherwise} \end{cases} \tag{6}$$

- For any continuous p.d.f. $f(x)$, $y = F(x)$ is distributed according to $[0, 1]$. Hence, p.d.f. of $x = F^{-1}(y)$ will be $f(x)$ if y has a uniform distribution $[0, 1]$.
- $f(x; 0, 1)$ (or simply $[0, 1]$) is commonly used in statistics as the base for random number generators.

The Monte Carlo method

Generating the uniform distribution $[0, 1]$

- In order to generate random variables according to an arbitrary p.d.f. one needs the $[0, 1]$ to start with.
- This task is usually acomplished by a **random number generator** algorithm. More specifically, computers can generate **pseudorandom** numbers (e.g. `RANMAR, RANLUX` from CernLib or `numpy.random.random()` in Python).
- Pseudorandom sequence behaves has a feel & touch of randomness, but given the initial **seed** remains entirely deterministic. It also has a cycle period (the longer the better, of course).
- There are various algorithms on the market. A common simple implementation generates the sequence $n_1, n_2, ...$ according to:

$$n_{i+1} = (a \times n_i) \bmod m.$$

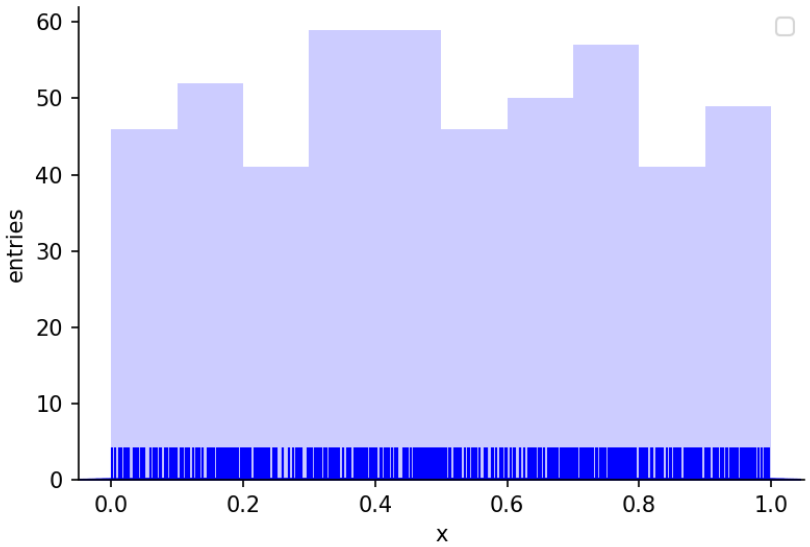
  The **multiplier** a and the **modulus** m are constants that determine the sequence, in particular its period (e.g. $a = 40692$ and $m = 2147483399$ were used on 32-bit machines giving the period of $\approx 2 \times 10^9$).
- Much better, more spohisticated algorithms are routinely used nowadays.

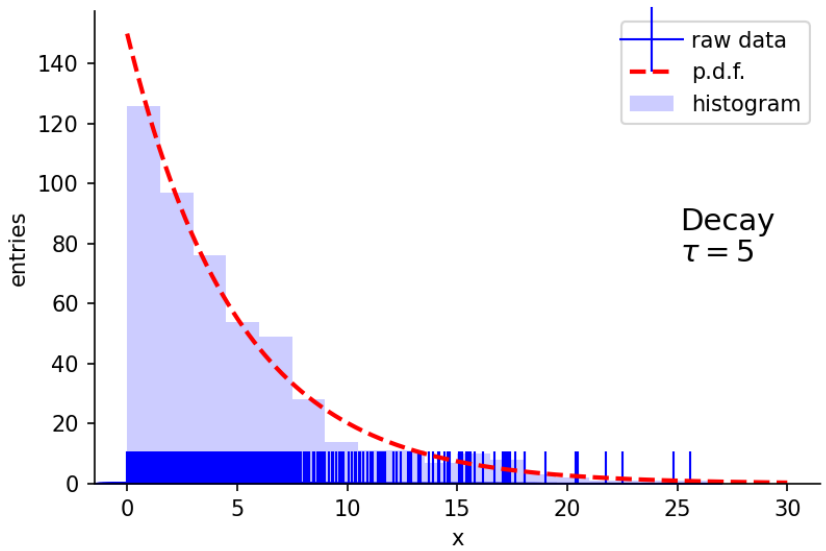
The Monte Carlo method

Using transformation for random number generation

- If the Eq. 5 can be solved for x ($x = F^{-1}(y)$) the transformation offers the most effective (efficient) way to generate random numbers.
- A simple example comes from the exponential decay:

$$y = F(x) = \int_0^{x(y)} \frac{1}{\tau} e^{-x'/\tau} \mathrm{d}x' \quad \Longrightarrow \quad x(y) = -\tau \log(1-y) \tag{7}$$



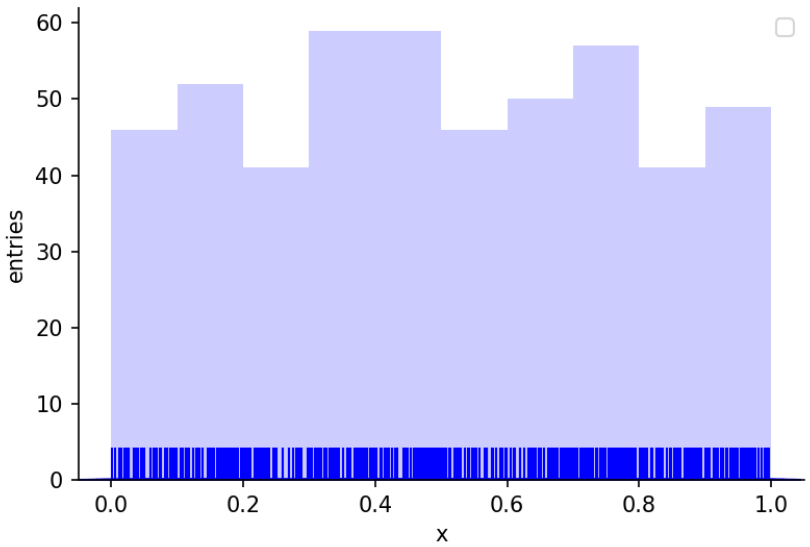


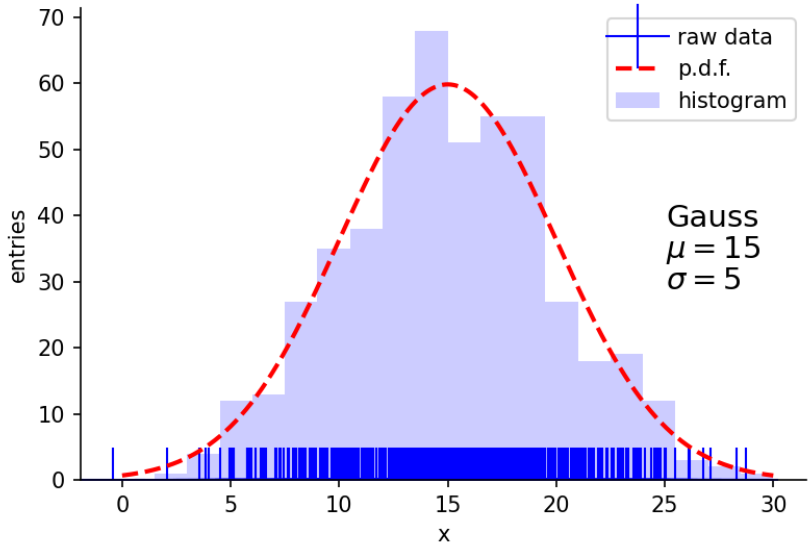
The Monte Carlo method

Using transformation for random number generation

- If the Eq. 5 can be solved for x ($x = F^{-1}(y)$) the transformation offers the most effective (efficient) way to generate random numbers.
- Numerically, it can also be done for a Gaussian:

$$y = F(x) = \int_{-\infty}^{x(y)} G(x';\mu,\sigma)\mathrm{d}x' \quad \Longrightarrow \quad x(y) = \sqrt{2}\sigma\ \mathrm{erf}^{-1}(2y-1) + \mu \tag{8}$$

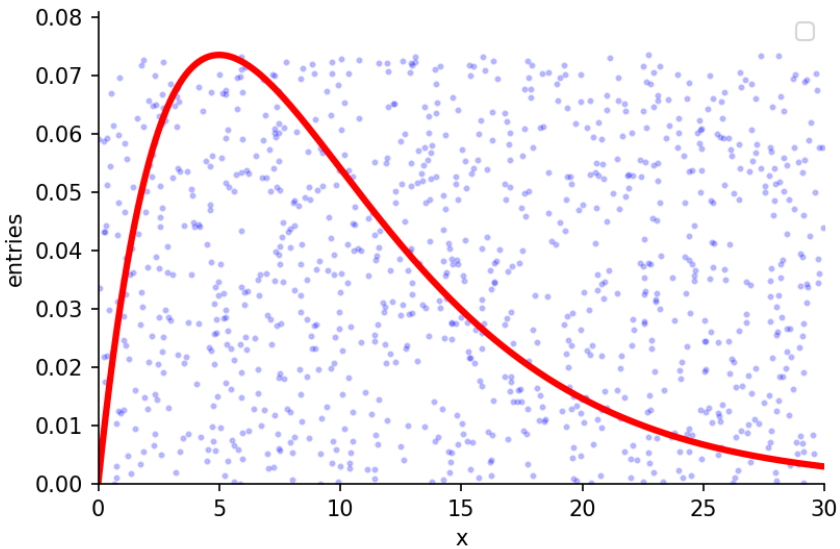


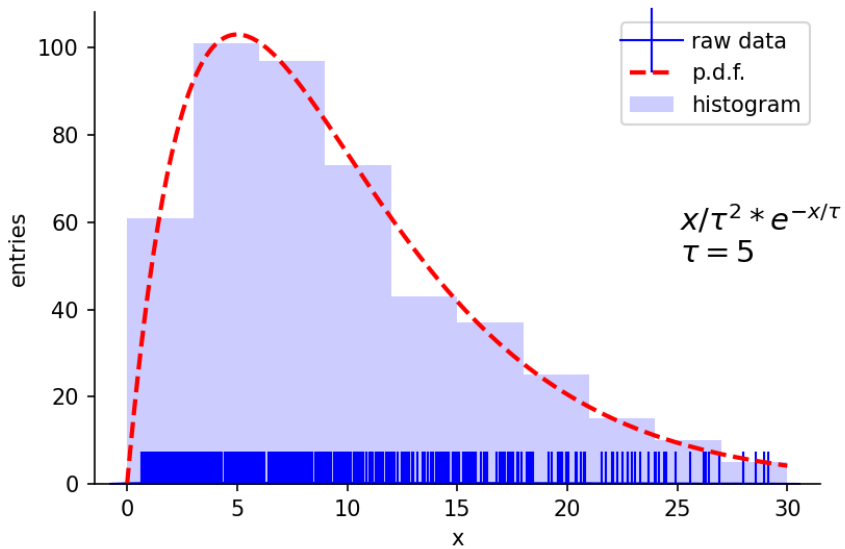


The Monte Carlo method

Acceptance-rejection for random number generation

- In more complex cases the acceptance-rejection method proves handy (also generally less efficient).
  1) Generate pairs of random numbers satisfying: $x = x_{\min} + r_1(x_{\max} - x_{\min})$, $u = r_2 f_{\max}$.
  2) If $u < f(x)$, then accept, otherwise reject.





Applications of the Monte Carlo method

- Various tests of estimators (we have seen multiple examples).
- Numerical integration. The accuracy improves as $1/\sqrt{N}$. Numerical integration using trapezoidal rule follows $1/N^{2/d}$ where d is the domain dimension. In 1D it is much faster but for $d > 4$ Monte Carlo comes into the game!
- Determination of p.d.f. of a function of random variables in more general case when $y(x_1, ..., x_n)$ when p.d.f.'s of x_i are known.
- More advanced examples come e.g. from particle physics when experimental data are simulated using **event generators** and subsequently **detector simulation** programs. In both cases randomness of the physics process is the paramount requirement.

From measured to true distributions

Unfolding - general concepts

In experimental physics (more generally, science) we usually take measurement and compare them to some theoretical models. What we observe (measure) in an experiment is never perfect. There are two basic ways around this problem:

1. Generate Monte Carlo events according to our model and add all experimental effects using simulation. Then, we can compare experimental data directly to the Monte Carlo prediction,
2. Given the observed data construct *estimator* for the underlying true distribution - called **unfolding**.

The first option is usually easier, but does not allow for comparison outside of the experimental context (e.g. at a later time) and does not allow for direct comparison or combination of different experiments. Unfolding, although more involved, gives considerably larger flexibility of interpretation.

From measured to true distributions

Unfolding - general concepts

Let us take a measurement of a spectrum with some features corresponding to measured properties (resonances). We have various *experimental effects* which impact the observed spectrum:

- resolution, which causes migration between bins: the distribution is "smeared out", peaks broadened,
- efficiency, which causes some events go undetected,
- background, which causes extra events due to spurious processes.

$$f_{\text{meas}}(x) = \int R(x|y) f_{\text{true}}(y) dy \tag{9}$$

Here we consider histogrammed (discretized) data $\mathbf{n} = (n_1, ..., n_N)$

$$\nu_i = E[n_i] = \sum_{j=1}^{M} R_{ij}\mu_j + \beta_i, \quad \mu_j = \mu_{\text{tot}} \underbrace{\int_{\text{bin}j} f_{\text{true}}(y) dy}_{p_j}, \quad i = 1, ..., N \tag{10}$$

From measured to true distributions

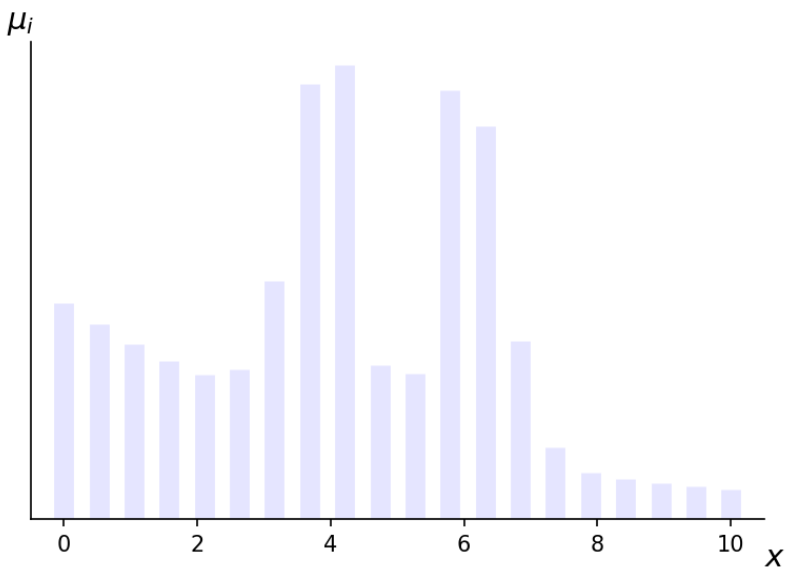
Unfolding - general concepts

$$R_{ij} = P\left(\text{observed in bin } i \mid \text{true in bin } j\right). \tag{11}$$

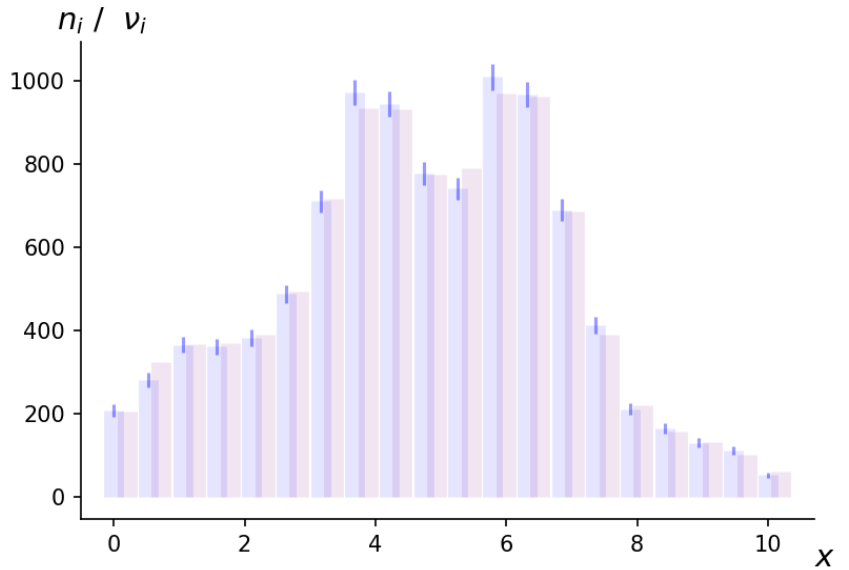
Note that:

$$\sum_{i=1}^{N} R_{ij} = \varepsilon_j \qquad \longleftarrow \text{efficiency} \tag{12}$$

"true" distribution histogram



"observed" events histogrammed



From measured to true distributions

Unfolding - general concepts

In the following we shall assume no background or known background contribution which can be subtracted prior to unfolding.
We have:

$$\boldsymbol{\nu} = R\boldsymbol{\mu} + \boldsymbol{\beta} \tag{13}$$

And assume it can be solved (inverted):

$$\boldsymbol{\mu} = R^{-1}\boldsymbol{\nu} \tag{14}$$

Suppose data are independent Poisson:

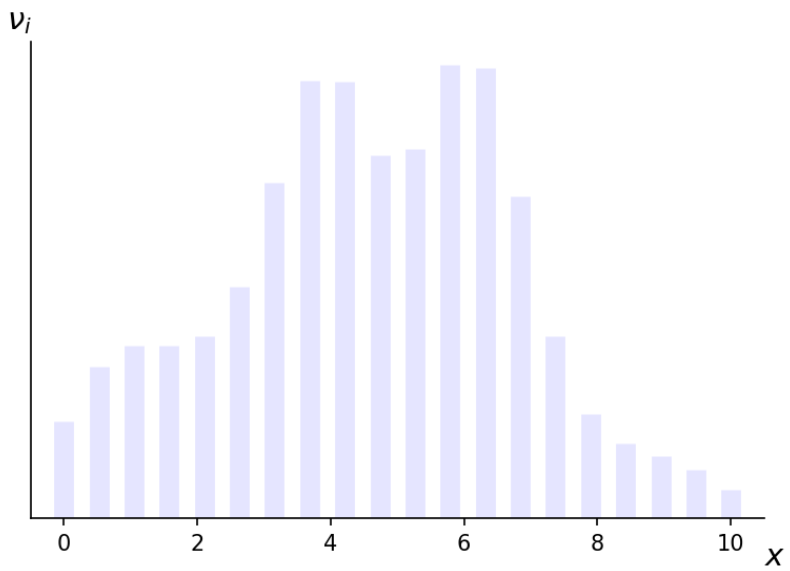
$$P(n_i;\nu_i) = \frac{\nu_i^{n_i}}{n_i!}e^{-\nu_i} \quad \Longrightarrow \quad \log L(\boldsymbol{\mu}) = \sum_{i=1}^{N}(n_i \ln \nu_i - \nu_i) \quad \Longrightarrow \quad \hat{\boldsymbol{\nu}} = \boldsymbol{n}$$
$$\longrightarrow \hat{\boldsymbol{\mu}} = R^{-1}\boldsymbol{n} \tag{15}$$

From measured to true distributions

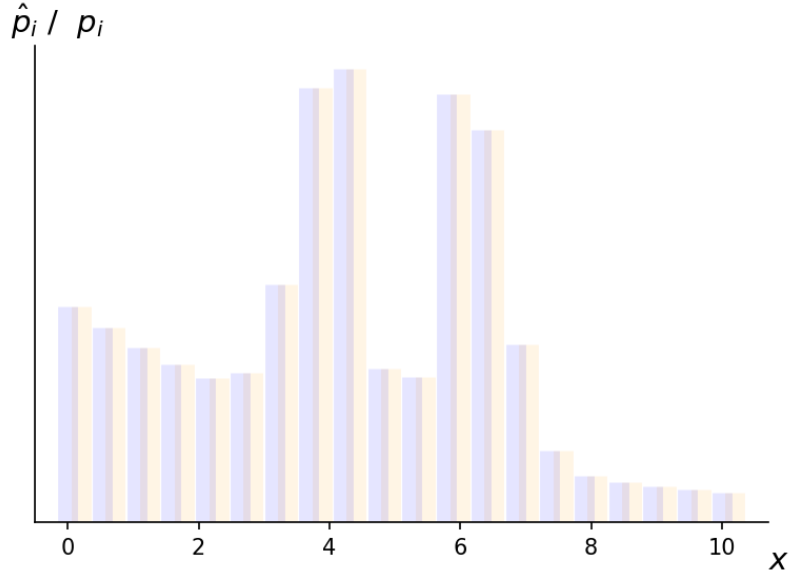
Unfolding - putting it to work

Let us take a finite Gaussian resolution ($\sigma = 0.6$) and an efficiency which linearly grows from 0.5 at 0 to 1 at 10.

"smeared" distribution histogram



"unfolded" histogram



$$\nu_i = \sum_{j=1}^{M} R_{ij}\mu_j$$

$$\hat{\mu}_i = \sum_{j=1}^{M} R_{ij}^{-1}\nu_j$$

From measured to true distributions

Unfolding - covariance matrix

For Poisson data the ML estimators are unbiased:

$$E[\hat{\boldsymbol{\mu}}] = R^{-1}E[\boldsymbol{n}] = \boldsymbol{\mu} \tag{16}$$

What is the covariance matrix of the unfolded histogram ($\hat{\boldsymbol{\mu}}$)?

$$U_{ij} = \mathrm{cov}[\hat{\mu}_i, \hat{\mu}_j] = \sum_{k,l=1}^{N} (R^{-1})_{ik}(R^{-1})_{jl}\,\mathrm{cov}[n_k, n_l] \tag{17}$$

$$\text{or in short: } U = R^{-1}V(R^{-1})^T$$

Recall the RCF bound:

$$(U^{-1})_{kl} = -E\left[\frac{\partial^2 \log L}{\partial\mu_k\mu_l}\right] = \sum_{i=1}^{N}\frac{R_{ik}R_{il}}{\nu_i}$$

$$\Longrightarrow U_{ij} = \sum_{k,l=1}^{N}(R^{-1})_{ik}(R^{-1})_{jk}\nu_k, \qquad \underbrace{(\mathrm{cov}[n_k, n_l] = \delta_{kl}\nu_k)}_{\text{independent Poisson}}. \tag{18}$$

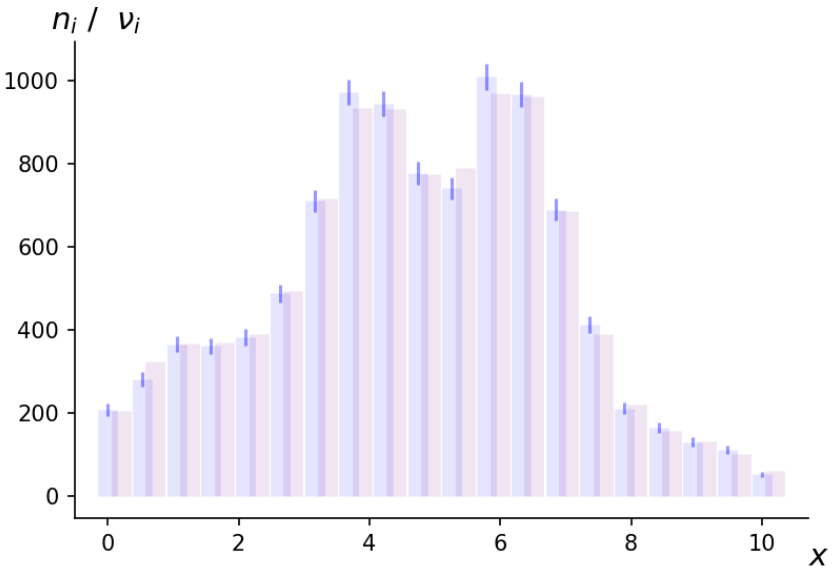
Unfolding realises the minimal variance among all unbiased estimators!

From measured to true distributions

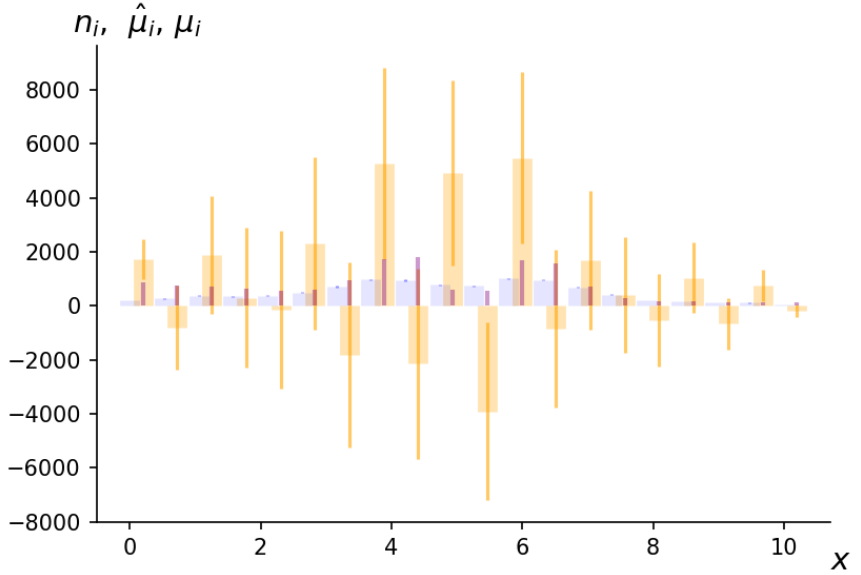
Unfolding - real life

Take the same Gaussian resolution ($\sigma = 0.6$) and an efficiency which linearly grows from 0.5 at 0 to 1 at 10 and generate 10,000 random events:

real data histogram



unfolded data histogram



Now, apply the same unfolding $\longrightarrow$

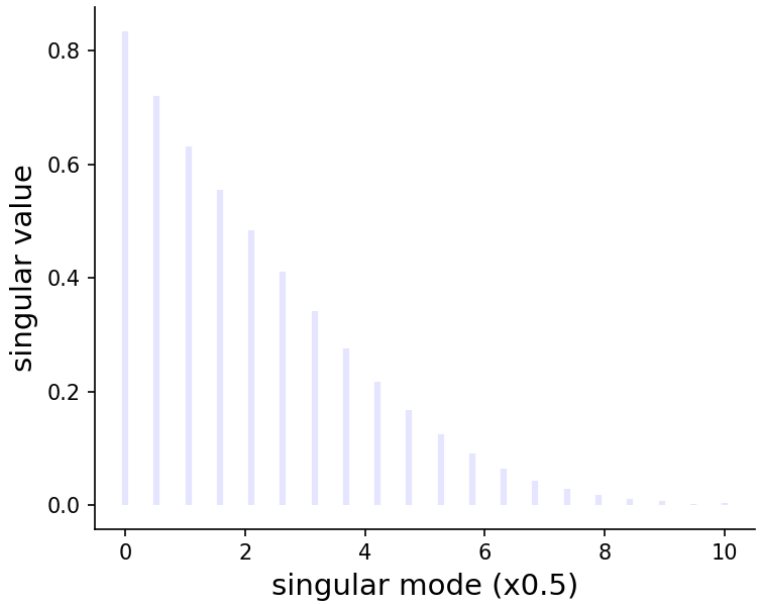
$$\hat{\mu}_i = \sum_{j=1}^{M} R_{ij}^{-1} n_j$$

Huston, we have a problem! ☹

From measured to true distributions

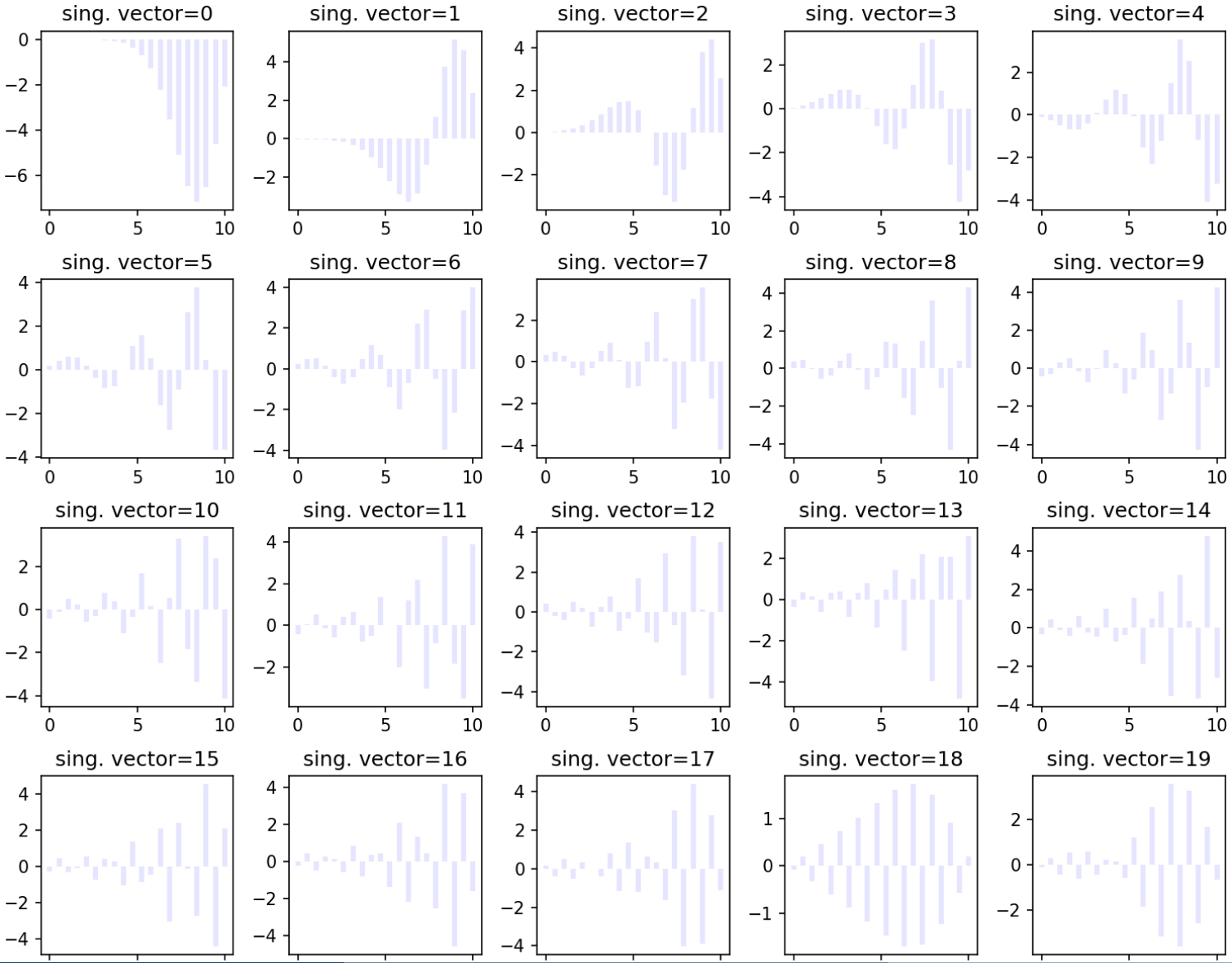
What has happened?

1. The perfect "unfolding" for the ν is assured by construction.
2. Data n seem very close but...
   there are Poisson fluctuations w.r.t. expectation ν.
3. Smearing is larger than the width of one bin. Information about fine structure at this level is lost. This corresponds to the fact that solutions corresponding to the hightest "frequency" modes are ill-defined.
4. Mathematically, such modes have very small singular values.



From measured to true distributions

The singular vectors (modes):



From measured to true distributions

Linear algebra - reminder

$$\mathcal{M}\boldsymbol{X} = \boldsymbol{Y} \quad \Longrightarrow \quad \boldsymbol{X} = \mathcal{M}^{-1}\boldsymbol{Y} \tag{19}$$

Assuming $\mathcal{M}$ a square matrix we can perform *Singular Value Decomposition* (SVD):

$$\mathcal{M} = S^T \mathcal{D} V \tag{20}$$

$$\underbrace{S\mathcal{M}V^T}_{\mathcal{D}}\underbrace{V\boldsymbol{X}}_{\tilde{\boldsymbol{X}}} = \underbrace{S\boldsymbol{Y}}_{\tilde{\boldsymbol{Y}}} \quad \Longrightarrow \quad \tilde{\boldsymbol{X}} = \mathcal{D}^{-1}\tilde{\boldsymbol{Y}}, \tag{21}$$

where $\mathcal{D}$ is a diagonal matrix with singular values λ_i on the diagonal and S (V) is a orthonormal matrix with eigenvectors $\boldsymbol{S_i}$ $(\boldsymbol{V_i})$ as rows.

$$\mathcal{D}\begin{pmatrix} \lambda_1 & \dots & 0 \\ \dots & \dots & \dots \\ 0 & \dots & \lambda_N \end{pmatrix}, \qquad U \propto \sum_i \frac{1}{\lambda_i^2} \tag{22}$$

From measured to true distributions

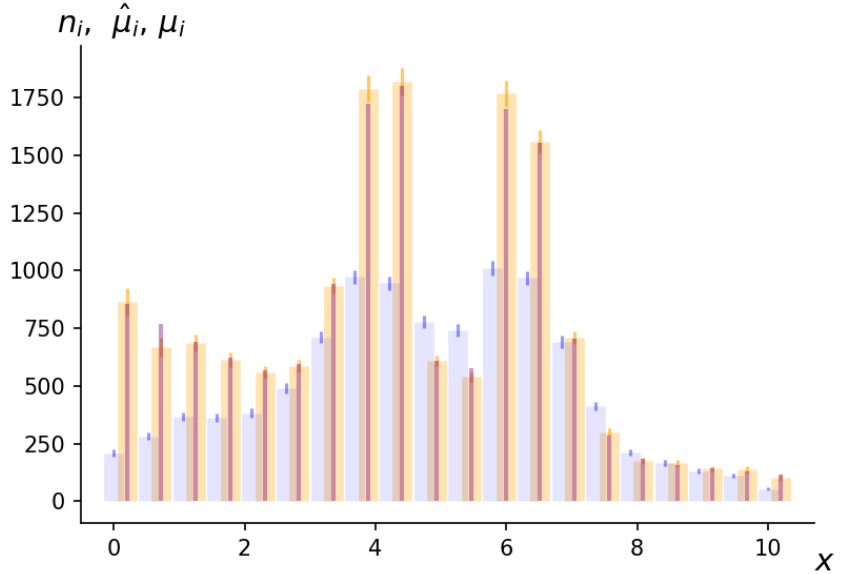
Correction factors

By far, the simplest method of correcting distributions is by applying **correction factors**.

$$\hat{\mu}_i = C_i n_i, \quad \text{with} \quad C_i = \frac{\mu_i^{\mathrm{MC}}}{\nu_i^{\mathrm{MC}}} \quad \longleftarrow \text{from MC.} \tag{23}$$

$$U_{ij} = \mathrm{cov}[\hat{\mu}_i, \hat{\mu}_j] = C_i C_j \,\mathrm{cov}[n_i, n_j] = \delta_{ij} C_i^2 n_i \quad \longleftarrow \text{small!.} \tag{24}$$

applying correction factors



We get a seamingly ideal response!
but...

From measured to true distributions

Correction factors

Now, real peaks have been shifted by one left.
This method is NOT unbiased. We actually have:

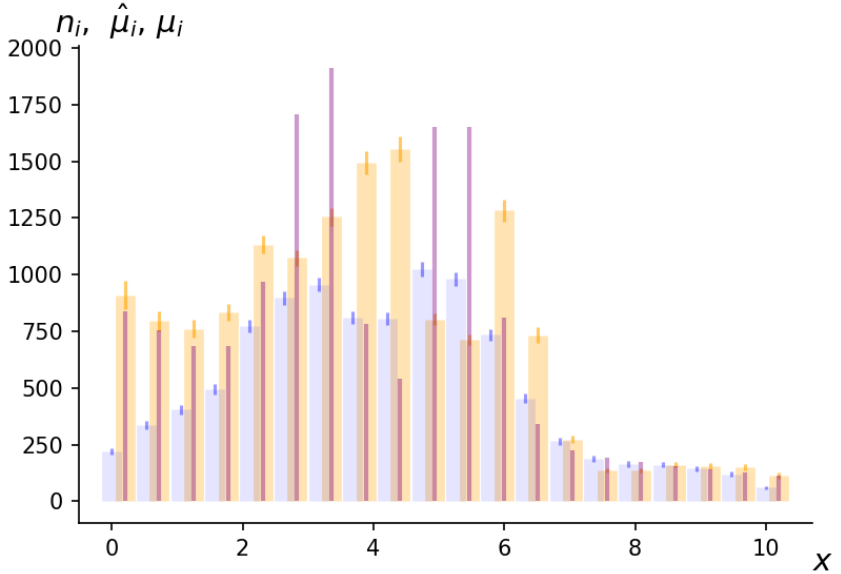
$$b_i = E[\hat{\mu}_i] - \mu_i \quad \longrightarrow b_i = \left(\frac{\mu_i^{\mathrm{MC}}}{\nu_i^{\mathrm{MC}}} - \frac{\mu_i}{\nu_i} \right) \tag{25}$$

When reality is different from the assumed model, we get it completely wrong ☹

In other words, correction factor method works ONLY if we are sure of the model!
In particular:

- bad if we don't know the exact shape of the signal.
- good if we want to correct for the known inefficiency.

applying correction factors

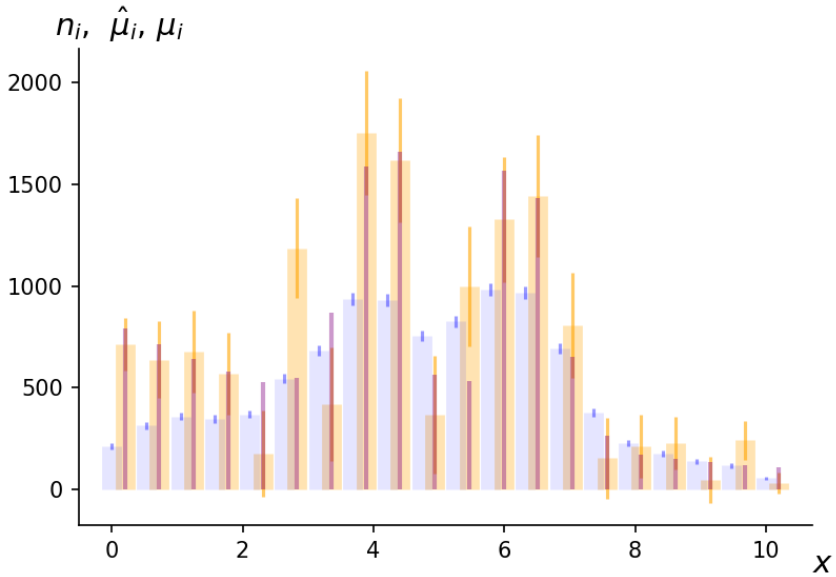


From measured to true distributions

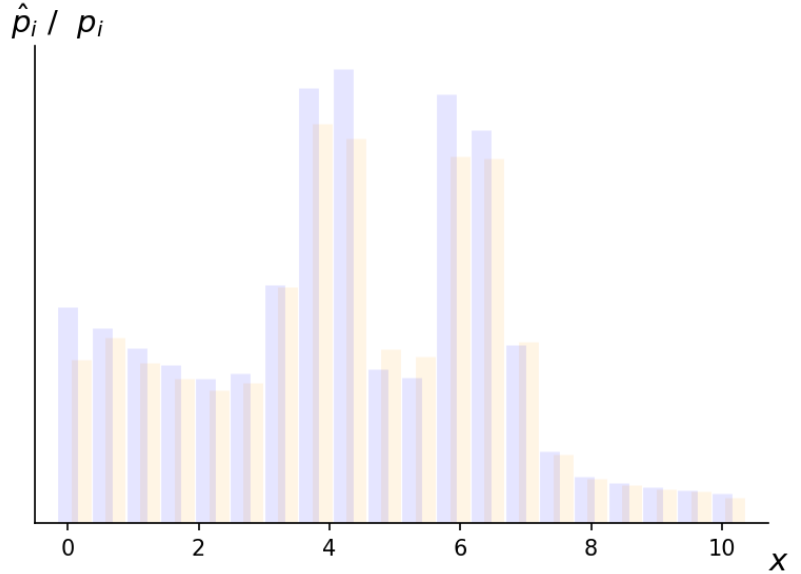
Unfolding - with conditioning

We execute the same unfolding but using the R^{-1} matrix with suppressed *weak modes*. This can be done in various ways. Here a very simple conditioning of the R is used, just to demonstrate the principle.

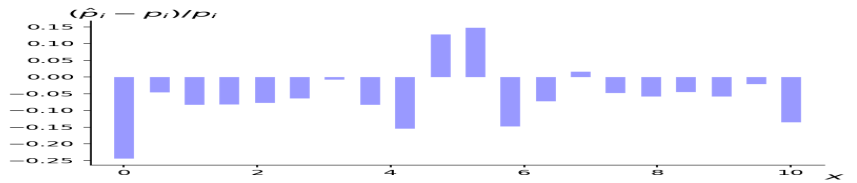
unfolded with conditioned R^{-1}



same applied to $\boldsymbol{\nu}$



Bias can be assessed from unfolding $\boldsymbol{\nu}$ $\longrightarrow$



That's way better! ☺

Regularized unfolding

The problem of unfolding are typically large uncertainties on "high frequency" modes which are unphysical and are driven by the statistical fluctuations.

Regularisation: maximize the following with respect to $\boldsymbol{\mu}$:

$$\Phi(\boldsymbol{\mu}) = \alpha \log L(\boldsymbol{\mu}) + S(\boldsymbol{\mu}) \tag{26}$$

$S(\boldsymbol{\mu})$ regularization function (measure of smoothness),
α regularization parameter (tradeoff between $\log L$ and S).

In addition require $\sum_i^N \nu_i = \sum_{i,j} R_{ij}\mu_j = n_{\text{tot}}$, i.e. maximize:

$$\phi(\boldsymbol{\mu}, \lambda) = \alpha \log L(\boldsymbol{\mu}) + S(\boldsymbol{\mu}) + \lambda \left[n_{\text{tot}} - \sum_i^N \nu_i \right], \tag{27}$$

where λ is a Lagrange multiplier: $\frac{\partial \phi}{\partial \lambda} = 0 \Rightarrow \sum_i^N \nu_i = n_{\text{tot}}$.

Regularized unfolding

One needs: a) regularization function $S(\boldsymbol{\mu})$, b) a prescription for choosing α.

1. Tikhonov regularization:

$$S[f_{\text{true}}(y)] = -\int \left(\frac{d^k f_{\text{true}}(y)}{dy^k} \right)^2, \quad \text{where } k = 1, 2, .. \tag{28}$$

E.g. for $k = 2$ and $\log L = -\frac{1}{2}\chi^2$ one gets:

$$\phi(\boldsymbol{\mu}, \lambda) = -\frac{\alpha}{2}\chi^2(\boldsymbol{\mu}) - \sum_{i=2}^{N-1} (-\mu_{i-1} + 2\mu_i - \mu_{i+1})^2, \tag{29}$$

which after setting derivatives equal zero, gives a system of linear equations solvable using e.g. SVD.

2. Entropy: $H = -\sum_{i=1}^{N} p_i \ln p_i$ (max when all p_i equal).
Use entropy-based regularization function:

$$S(\boldsymbol{\mu}) = H(\boldsymbol{\mu}) = -\sum_{i=1}^{N} \frac{\mu_i}{\mu_{\text{tot}}} \ln \frac{\mu_i}{\mu_{\text{tot}}}, \tag{30}$$

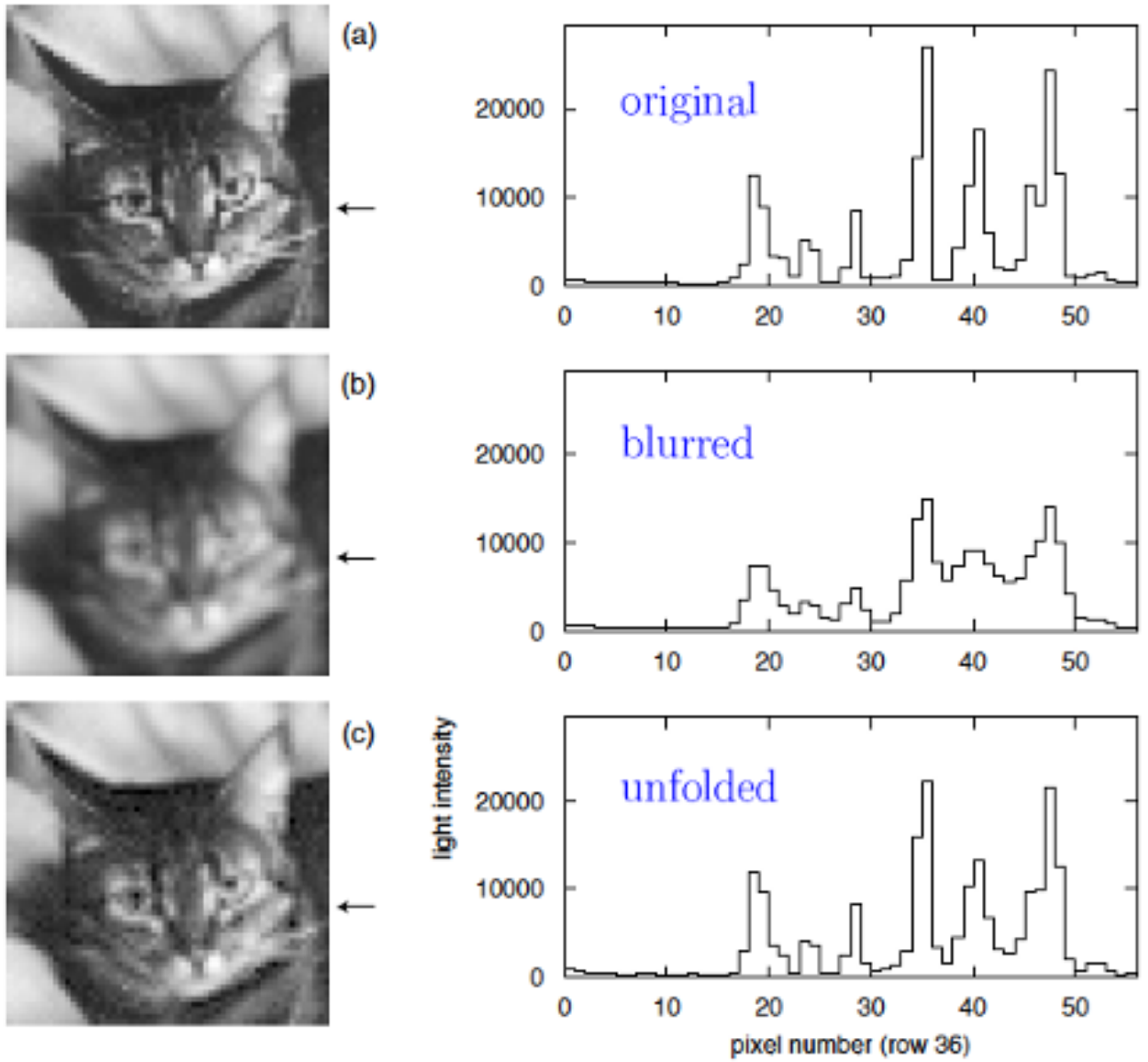
$\propto \ln(\text{number of ways to arrange } \mu_{\text{tot}} \text{ entries in } M \text{ bins})$

From measured to true distributions

Unfolding - image unblure

The entropy-based unfolding of a blured image.

credit: G. Cowan



Regularized unfolding

Choosing α parameter

One needs a trade-off between bias and variance:
Common choices include:

1. $$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(U_{ii} + \hat{b}_i^2), \quad \text{or} \quad \text{MSE'} = \frac{1}{N}\sum_{i=1}^{N}\frac{U_{ii} + \hat{b}_i^2}{\hat{\mu}_i}.$$

2. Allow for a "reasonable" change of the χ^2:

$$\Delta\chi^2 = 2\Delta \log L = N$$

3. Require bias be consistent with zero within its uncertainty:

$$\chi_b^2 = \sum_{i=1}^{N}\frac{\hat{b}_i^2}{W_{ii}} = N, \quad \text{where} \quad W_{ij} = \text{cov}[\hat{b}_i, \hat{b}_j].$$

Note: there is no optimal choice. All depends on the particular analysis context.

Thank you

Back-up