8 Further tips

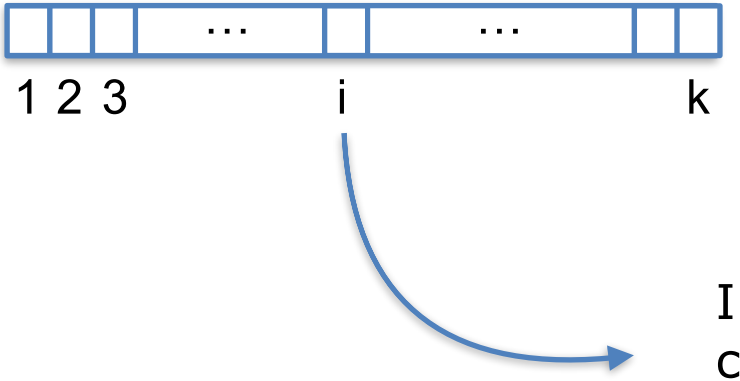
- Simulation to experiment comparison
- How to make your McStas more efficient

What is really the information content…?

- McStas sources generally provide “intensity” in units of neutrons/s (into a chosen solid angle)
- That intensity is carried through the instrument on a discrete set of “neutron rays”

In a histogram sense

- Imagine a histogram, e.g. I(λ)



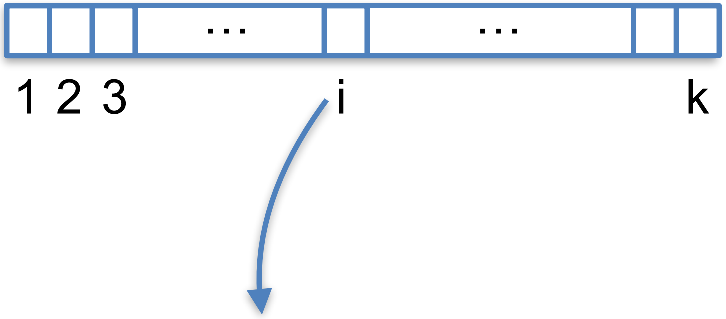
In bin *i*, *N* events each carrying a fractional intensity p_j so that

$$I = \sum_{N} p_j$$

- The RMS variance over that set becomes our statistical error bar *E*

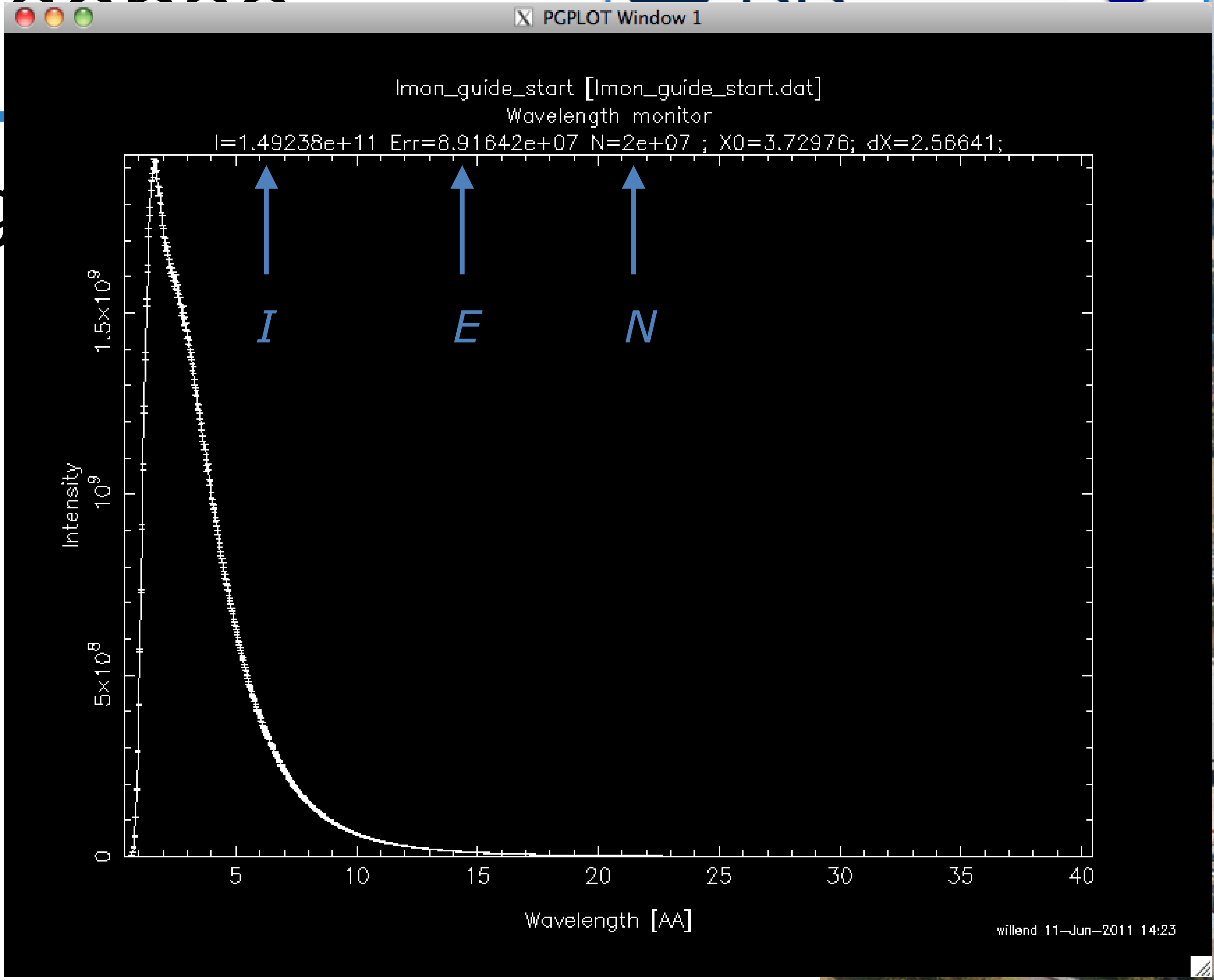
In a histogram

- Imagine a histog



In bin *i*, *N* events each carrying a fractional intensity p_j so that

$$I = \sum_N p_j$$



```
Imon_guide_start [Imon_guide_start.dat]
Wavelength monitor
I=1.49238e+11 Err=8.91642e+07 N=2e+07 ; X0=3.72976; dX=2.56641;
```

- The RMS variance over that set becomes our statistical error bar *E*

From "Virtual experiments - the ultimate aim of neutron ray-tracing simulations", K. Lefmann et al., Journal of Neutron Research 16, 97-111 (2008)

Let n be the number of neutron rays reaching the detector, and let the rays have (different) weights, w_i. The simulated intensity is then given by

$$\boxed{I = \sum_{i=1}^{n} w_i.} \qquad (1)$$

The estimate of the error on this number is calculated in the McStas manual [1], and the standard deviation is approximated by

$$\sigma^2(I) = \sum_{i=1}^{n} w_i^2. \qquad (2)$$

In real experiments, $w_i = 1$, whence we reach $I = n$ and $\sigma(I) = \sqrt{I}$ as expected (for counts exceeding 10). Let the virtual time be denoted by t. The simulated counts during this time becomes

$$\boxed{C = tI,} \qquad (3)$$

From "Virtual experiments - the ultimate aim of neutron ray-tracing simulations", K. Lefmann et al., Journal of Neutron Research 16, 97-111 (2008)

and its error bar estimate is

$$\sigma^2(C) = t^2\sigma^2(I). \tag{4}$$

However, to simulate a realistic counting statistics, we must fulfill

$$\sigma_{\mathrm{VE}}(C_{\mathrm{VE}}) = \sqrt{C_{\mathrm{VE}}}. \tag{5}$$

This is obtained by adding to (3) a Gaussian noise $E(\Sigma)$ of mean value zero and standard deviation Σ:

$$C_{\mathrm{VE}} = tI + E(\Sigma). \tag{6}$$

The standard deviation for the VE becomes

$$\sigma^2_{\mathrm{VE}}(C) = t^2\sigma^2(I) + \Sigma^2. \tag{7}$$

Now, the requirement (5) allows us to determine Σ:

$$\Sigma^2 = tI - t^2\sigma^2(I). \tag{8}$$

Since Σ^2 must remain positive, we reach an upper limit on t

$$\boxed{t_{\max} = \frac{I}{\sigma^2(I)}.} \tag{9}$$

Sketch of an algorithm…

1. On a given McStas histogram

2. For the non-zero bins, calculate

$$t_{\max} = \frac{I}{\sigma^2(I)}.$$

3. The *smallest* $t_{\mathtt{max}}$ defines the "maximal counting time" allowed by your statistics

4. Preferably a "background" should be added - use a "known experimental value" or an estimate…

Important points to remember

1. Your simulation will only contain elements you provided / defined
2. … to the precision you defined
3. Answers the questions you posed
4. Background essentially only from “sample”, or sample-near objects

Onto efficiency…

- Apply focusing techniques
  - At the source (spatially, temporally, in wavelength…)
  - At the sample, if possible
- (carefully!) Apply SPLIT - but only if immediately followed by Monte Carlo choices, e.g. in sample
- Alternatively use MCPL o/i which allows repetition - beware of biases!

Onto efficiency…

- Apply focusing techniques
  - At the source (spatially, temporally, in wavelength…)
  - At the sample, if possible
- (carefully!) Apply SPLIT - but only if immediately followed by Monte Carlo choices, e.g. in sample
- Alternatively use MCPL o/i which allows repetition - beware of biases!

All of this can be considered "variance reduction" or biasing

Onto efficiency…

- Use MPI parallelisation - included in macOS install from 2.4, easy to get on Linux…

- The Intel C compiler is known to give ~factor of 2 wrt. gcc in most cases

- - Still consider if you are asking the right question if runtimes reach days/weeks…

Onto efficiency…

- Use MPI parallelisation - included in macOS install from 2.4, easy to get on Linux…
- The Intel C compiler is known to give ~factor of 2 wrt. gcc in most cases
- \- Still consider if you are asking the right question if runtimes reach days/weeks…

Sledge-hammer / brute force!