ROOT Project - Part 2: Unbinned vs. Binned Fits

Project Goal: The objective of this analysis is to compare the unbinned maximum-likelihood estimate for an exponential decay to several binned (histogram) fits, utilizing ROOT's binned-likelihood option. This approach aims to illustrate the impact of binning on the estimator and to demonstrate how sufficiently fine binning can recover the result obtained from the unbinned case.

Data: The following 40 decay times were analyzed:

$$\{4.99, 4.87, 2.59, 3.04, 3.39, 6.20, 10.61, 7.64, 3.92, 5.33, 4.85, 2.39, 4.16, 6.74, 3.53, 5.86, 5.41, 26.25, 4.40, 10.79, 7.08, 2.86, 33.9$$

The total number of points is $N = 40$, and the sum of decay times is $\sum_i t_i = 270.4$.

Code reference (data container, N, and sum):

```
// vector of times and N
std::vector<double> times = { 4.99, 4.87, /* ... */ 7.69, 4.93 };
int N_data = times.size();

// compute sum of decay times
double sum_t = 0.0;
for (double t_val : times) { sum_t += t_val; }
std::cout << "Sum of decay times (Sigma t_i): " << sum_t << std::endl;
```

Unbinned Analysis (Analytical ML)

For the exponential probability density function $f(t|\lambda) = \lambda e^{-\lambda t}$, the log-likelihood is given by

$$\ln L(\lambda) = N \ln \lambda - \lambda \sum_i t_i,$$

with the maximum-likelihood estimator (MLE) expressed as $\hat{\lambda} = N/\sum_i t_i$.

Computed values (analytical):

- $\hat{\lambda}_{\mathrm{ML}} = \dfrac{40}{270.4} = 0.147929.$

  Code reference (MLE):

  ```
  // analytical MLE
  double lambda_ml_unbinned = 0.0;
  if (sum_t > 1e-9) {
      lambda_ml_unbinned = static_cast<double>(N_data) / sum_t;
  }
  std::cout << "Unbinned ML estimate for lambda (lambda_hat_ML): " << lambda_ml_unbinned <<
       std::endl;
  ```

- Analytical standard error (approximate): $\sigma_{\hat{\lambda}} \approx \dfrac{\hat{\lambda}}{\sqrt{N}} = 0.0233896.$

  Code reference (analytical error):

  ```
  // approximate analytical error
  double err_lambda_ml_unbinned = 0.0;
  if (N_data > 0 && lambda_ml_unbinned > 1e-9) {
      err_lambda_ml_unbinned = lambda_ml_unbinned / TMath::Sqrt(static_cast<double>(N_data)
      );
  }
  std::cout << "Analytical error on lambda_hat_ML: " << err_lambda_ml_unbinned << std::endl
      ;
  ```

Log-Likelihood Plot and Graphical 1-Sigma

The function $\ln L(\lambda) = 40 \ln \lambda - 270.4\lambda$ was evaluated and plotted around its maximum, and the graphical 1σ interval (where $\Delta \ln L = -0.5$) was determined.

Code reference (log-likelihood TF1 definition and evaluation):

```
// define log-likelihood as a ROOT TF1-compatible function
auto logLikelihoodFunc = [&](double* x_lambda_ptr, double* /*params*/) {
    double current_lambda = x_lambda_ptr[0];
    if (current_lambda <= 1e-9) return -1e10;
    return static_cast<double>(N_data) * TMath::Log(current_lambda) - current_lambda * sum_t;
};
double lambda_plot_min = lambda_ml_unbinned > 3 * err_lambda_ml_unbinned ? lambda_ml_unbinned
    - 3 * err_lambda_ml_unbinned : 0.01;
if (lambda_plot_min <= 0) lambda_plot_min = 0.001;
double lambda_plot_max = lambda_ml_unbinned + 3 * err_lambda_ml_unbinned;
if (lambda_plot_max <= lambda_plot_min) lambda_plot_max = lambda_plot_min + 0.1;

TF1* fLogL = new TF1("fLogL", logLikelihoodFunc, lambda_plot_min, lambda_plot_max, 0);
fLogL->SetNpx(500);

// evaluate maximum at analytic MLE and determine target for 1-sigma
double logL_max = fLogL->Eval(lambda_ml_unbinned);
double target_logL_for_sigma = logL_max - 0.5;
```

To find the graphical bounds where $\ln L(\lambda) = \ln L_{\max} - 0.5$ we use ROOT's 'GetX' method (searching left / right of the MLE):

```
// find left/right 1-sigma intersections graphically
double lambda_sigma_minus = lambda_ml_unbinned;
double lambda_sigma_plus  = lambda_ml_unbinned;

if (fLogL->Eval(lambda_plot_min) < target_logL_for_sigma && lambda_ml_unbinned >
    lambda_plot_min) {
     lambda_sigma_minus = fLogL->GetX(target_logL_for_sigma, lambda_plot_min,
    lambda_ml_unbinned - 1e-6);
} else {
     std::cout << "Warning: Could not find lower sigma bound reliably with GetX.\n";
}
if (fLogL->Eval(lambda_plot_max) < target_logL_for_sigma && lambda_ml_unbinned <
    lambda_plot_max) {
     lambda_sigma_plus = fLogL->GetX(target_logL_for_sigma, lambda_ml_unbinned + 1e-6,
    lambda_plot_max);
} else {
     std::cout << "Warning: Could not find upper sigma bound reliably with GetX.\n";
}
```

Values obtained from the log-likelihood analysis:

- $\ln L_{\max} = -116.441$.
- Target for 1σ: $\ln L_{\max} - 0.5 = -116.941$.
- Graphically determined bounds: $\lambda_- = 0.125756$, $\lambda_+ = 0.172567$.
- Asymmetric errors: -0.0221735 and $+0.0246383$. Average (approximate symmetric) error: 0.0234059.
- The graphical results are found to be consistent with the analytical error of 0.0233896.

Comment: The peak of the likelihood and the graphical 1σ band exhibit close agreement with the analytical estimate, supporting the robustness of the unbinned ML result.

Binned Fits (Histograms) and Diagnostics

Three binned-likelihood fits were performed using the following configurations (employing ROOT with option "L"):

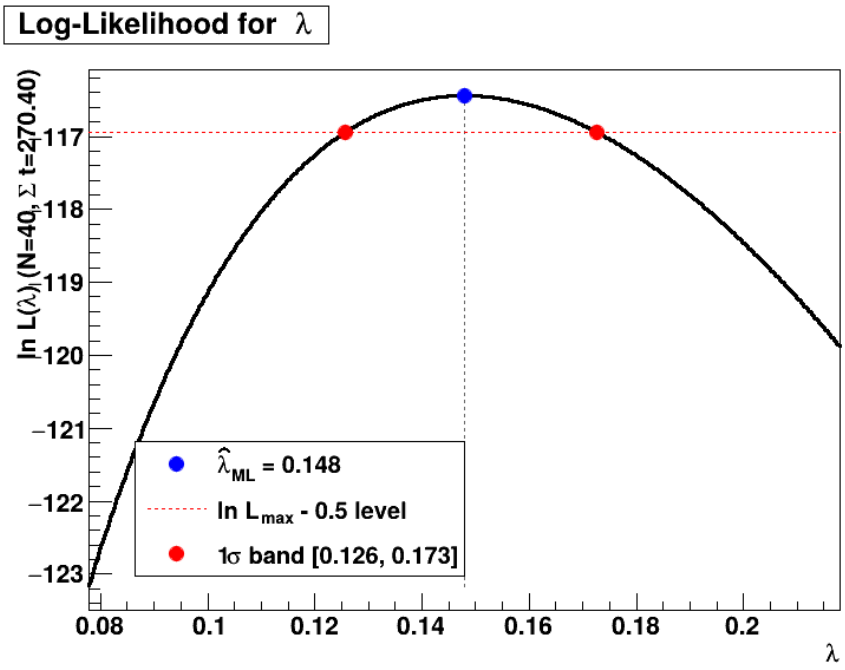

Figure 1: Profile log-likelihood $\ln L(\lambda)$ as a function of λ. The maximum and the horizontal line at $\ln L_{\max} - 0.5$ are indicated.

1. **3 bins, range** $0 - 15$ (bin width $= 5$)
2. **5 bins, range** $0 - 15$
3. **350 bins, range** $0 - 35$ (bin width $= 0.1$; includes all data)

3-bin Fit (0–15, 3 bins)

Code reference (3-bin histogram creation, fill, model, binned-likelihood fit):

```
// create 3-bin histogram and fill
double t_min_hist1 = 0.0;
double t_max_hist1 = 15.0;
int n_bins_hist1 = 3;
TH1F* h1 = new TH1F("h1", Form("Binned Decay (N_bins=%d);Time (t);Entries / (%.1f time units)"
    , n_bins_hist1, (t_max_hist1-t_min_hist1)/n_bins_hist1),
                   n_bins_hist1, t_min_hist1, t_max_hist1);
for (double t_val : times) { h1->Fill(t_val); }

// exponential model and initial params
TF1* fitFunc1 = new TF1("fitFunc1", "[0]*exp(-[1]*x)", t_min_hist1, t_max_hist1);
fitFunc1->SetParName(0, "Amplitude"); fitFunc1->SetParName(1, "#lambda");
fitFunc1->SetParameter(1, lambda_ml_unbinned);
double approx_norm1 = (double)N_data * ( (t_max_hist1-t_min_hist1)/n_bins_hist1 ) *
    lambda_ml_unbinned;
fitFunc1->SetParameter(0, approx_norm1 > 0 ? approx_norm1 : N_data);

// fit with binned likelihood ('L')
TFitResultPtr r1 = h1->Fit(fitFunc1, "L Q S");
```

Results and Diagnostics:

- Fitted $\lambda_{\text{binned},3} = 0.16122 \pm 0.0468606$.

  Code reference (extract param, chi2/ndf):

  ```
  if (r1->IsValid()) {
      double lambda_binned1 = r1->Parameter(1);
      double err_lambda_binned1 = r1->ParError(1);
  ```

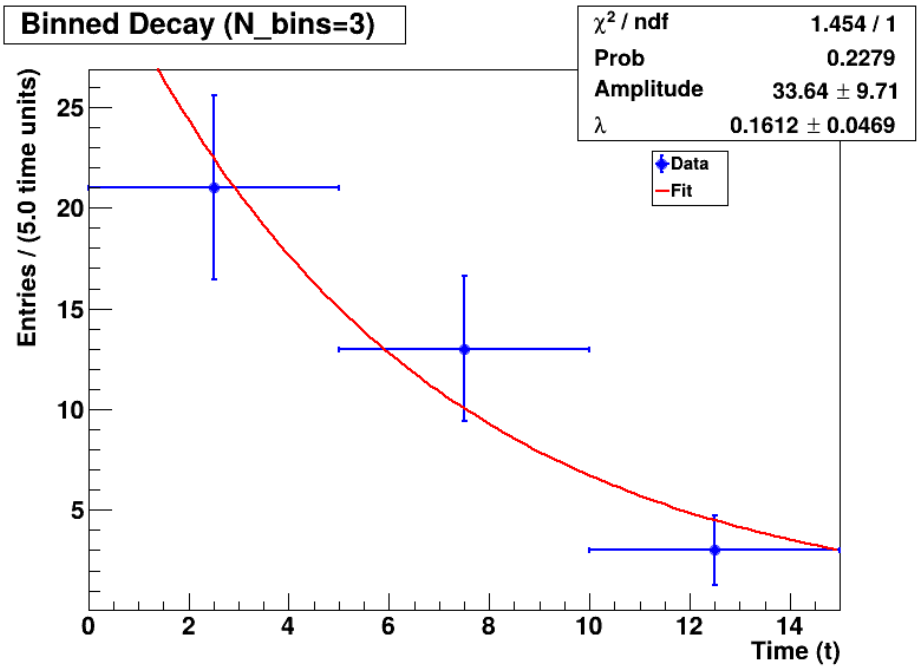

Figure 2: Binned ML fit to the histogram with 3 bins covering $t \in [0, 15]$.

```
    std::cout << "  lambda_binned (3 bins) = " << lambda_binned1 << " +/- " <<
    err_lambda_binned1 << std::endl;
    std::cout << "  Fit Chi2/NDF = " << r1->Chi2() << "/" << r1->Ndf() << std::endl;
}
```

- Integral of fitted function over [0,15]: 190.047.
- Bin width = 5. Therefore Q = Integral/bin width = 190.047/5 = 38.0094.

  Code reference (integral, bin width, Q, counts):

```
double integral_fit1 = fitFunc1->Integral(t_min_hist1, t_max_hist1);
double bin_width1 = h1->GetXaxis()->GetBinWidth(1);
double Q1 = (bin_width1 > 1e-9) ? (integral_fit1 / bin_width1) : 0.0;
double sum_bin_contents1 = h1->GetSumOfWeights();
std::cout << "Integral of fitted function = " << integral_fit1 << std::endl;
std::cout << "Bin width = " << bin_width1 << ", Q = " << Q1 << std::endl;
std::cout << "Total entries in histogram = " << h1->GetEntries() << std::endl;
```

- The sum of histogram bin contents in the fitted range is 37 (out of 40 total entries).

Observation: The normalization of the 3-bin fit is consistent with the observed counts ($Q \approx 38$ vs. 37 in range). The binned value for λ is somewhat higher than the unbinned estimate and exhibits a larger uncertainty, as is expected when coarse binning is used.

5-bin Fit (0–15, 5 bins)

Code reference (5-bin histogram fit):

```
int n_bins_hist2 = 5;
TH1F* h2 = new TH1F("h2", Form("Binned Decay (N_bins=%d);Time (t);Entries / (%.1f time units)"
    , n_bins_hist2, (t_max_hist1-t_min_hist1)/n_bins_hist2),
                  n_bins_hist2, t_min_hist1, t_max_hist1);
for (double t_val : times) { h2->Fill(t_val); }
TF1* fitFunc2 = new TF1("fitFunc2", "[0]*exp(-[1]*x)", t_min_hist1, t_max_hist1);
fitFunc2->SetParName(0, "Amplitude"); fitFunc2->SetParName(1, "#lambda");
fitFunc2->SetParameter(1, lambda_ml_unbinned);
```

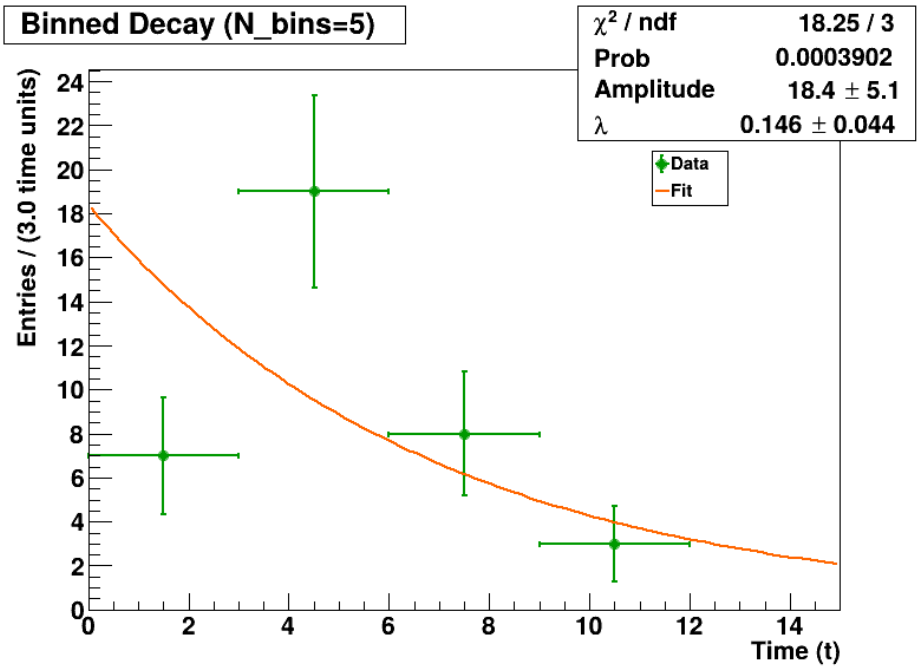

Figure 3: Binned ML fit to the histogram with 5 bins covering $t \in [0, 15]$.

```
double approx_norm2 = (double)N_data * ( (t_max_hist1-t_min_hist1)/n_bins_hist2 ) *
    lambda_ml_unbinned;
fitFunc2->SetParameter(0, approx_norm2 > 0 ? approx_norm2 : N_data);
TFitResultPtr r2 = h2->Fit(fitFunc2, "L Q S");
if (r2->IsValid()) {
    double lambda_binned2 = r2->Parameter(1);
    double err_lambda_binned2 = r2->ParError(1);
}
```

Result:

- $\lambda_{\text{binned},5} = 0.14603 \pm 0.0435447$.

Observation: Increasing the number of bins from 3 to 5 leads the estimate to move closer to the unbinned ML value (0.147929), and the uncertainty decreases slightly. This is consistent with expectations: moderate bin refinement can recover more information from the data.

350-bin Fit (0–35, 350 bins)

Code reference (350-bin histogram fit):

```
double t_min_hist3 = 0.0; double t_max_hist3 = 35.0; int n_bins_hist3 = 350;
TH1F* h3 = new TH1F("h3", "Binned Decay (350 bins, t_{max}=35);Time (t);Entries / bin",
                    n_bins_hist3, t_min_hist3, t_max_hist3);
for (double t_val : times) { h3->Fill(t_val); }

TF1* fitFunc3 = new TF1("fitFunc3", "[0]*exp(-[1]*x)", t_min_hist3, t_max_hist3);
fitFunc3->SetParName(0, "Amplitude"); fitFunc3->SetParName(1, "#lambda");
fitFunc3->SetParameter(1, lambda_ml_unbinned);
fitFunc3->SetParameter(0, N_data * lambda_ml_unbinned > 0 ? N_data * lambda_ml_unbinned :
    N_data);
TFitResultPtr r3 = h3->Fit(fitFunc3, "L Q S");
if (r3->IsValid()) {
    double lambda_binned3 = r3->Parameter(1);
    double err_lambda_binned3 = r3->ParError(1);
}
```

Result:

- $\lambda_{\text{binned},350} = 0.142986 \pm 0.0248022$.

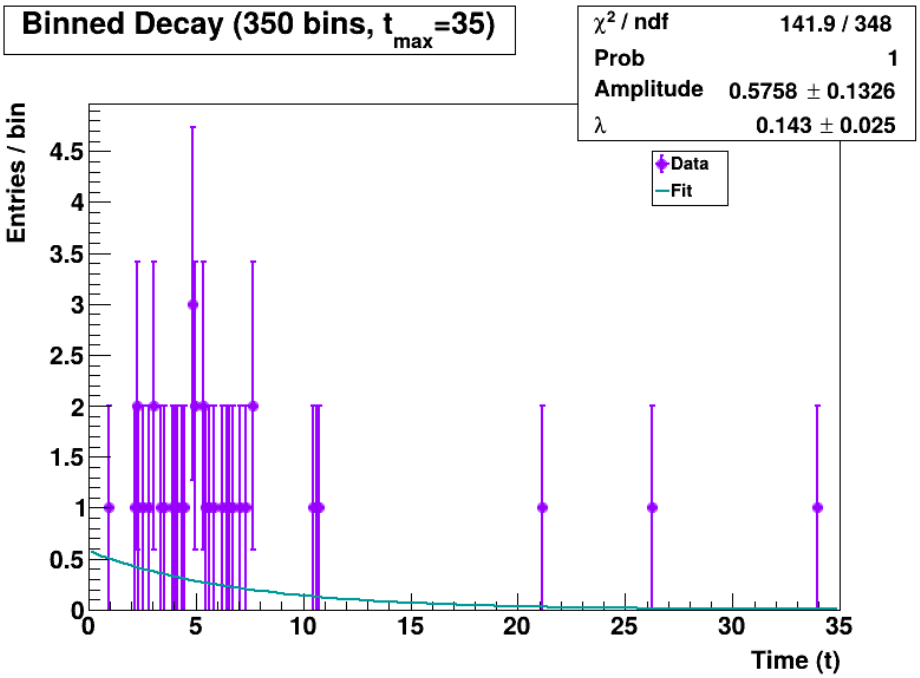

Figure 4: Binned ML fit to the histogram with 350 bins over $t \in [0, 35]$. Most bins are empty or contain a single event.

Observation: When very fine binning is employed and the full range is included, the binned ML result converges toward the unbinned result (0.147929), with an uncertainty comparable to the unbinned analytical error. This demonstrates the expected limiting behavior: as bin width decreases, the binned likelihood approaches the unbinned likelihood.

Conclusions

- The preferred estimate is provided by the unbinned MLE:

$$\boxed{\hat{\lambda} = 0.147929 \pm 0.02339.}$$

- The graphical log-likelihood interval (0.125756, 0.172567) yields an average error ≈ 0.02341, which is in very good agreement with the analytical calculation.
- Coarse binning tends to increase the uncertainty and can result in a shift of the central value, while finer binning (or use of the unbinned ML method) allows for more accurate recovery of the underlying information in the sample.
- The normalization check for the 3-bin fit ($Q = 38.0094$ vs. observed 37) indicates consistency between the fit and the observed data in the fitted range.

Appendix: Code Snippets (provided for reproducibility)

```
// compute analytical unbinned ML (actual code used)
double sum_t = 0.0;
for (double t_val : times) sum_t += t_val;
double lambda_ml_unbinned = static_cast<double>(N_data) / sum_t;
double err_lambda_ml_unbinned = lambda_ml_unbinned / TMath::Sqrt(static_cast<double>(N_data));
```

```
// create log-likelihood TF1 (actual code used) and find 1-sigma band
auto logLikelihoodFunc = [&](double* x_lambda_ptr, double* /*params*/) {
    double current_lambda = x_lambda_ptr[0];
    if (current_lambda <= 1e-9) return -1e10;
    return static_cast<double>(N_data) * TMath::Log(current_lambda) - current_lambda * sum_t;
};
TF1* fLogL = new TF1("fLogL", logLikelihoodFunc, lambda_plot_min, lambda_plot_max, 0);
fLogL->SetNpx(500);
double logL_max = fLogL->Eval(lambda_ml_unbinned);
double target_logL_for_sigma = logL_max - 0.5;
double lambda_sigma_minus = fLogL->GetX(target_logL_for_sigma, lambda_plot_min,
    lambda_ml_unbinned - 1e-6);
double lambda_sigma_plus  = fLogL->GetX(target_logL_for_sigma, lambda_ml_unbinned + 1e-6,
    lambda_plot_max);
```