Statistical Data Analysis PH4515
Problem Sheet 7 Solutions

1(a) [4 marks] The exponentially distributed time measurements, $t_1, \ldots, t_n$, and the Gaussian distributed calibration measurement y are all independent, so the likelihood is simply the product of the corresponding pdfs:

$$L(\tau, \lambda) = \prod_{i=1}^{n} \frac{1}{\tau + \lambda} e^{-t_i/(\tau+\lambda)} \frac{1}{\sqrt{2\pi}\sigma} e^{-(y-\lambda)^2/2\sigma^2} .$$

The log-likelihood is therefore

$$\ln L(\tau, \lambda) = -n \ln(\tau + \lambda) - \frac{1}{\tau + \lambda} \sum_{i=1}^{n} t_i - \frac{(y - \lambda)^2}{2\sigma^2} + C ,$$

where C represents terms that do not depend on the parameters and therefore can be dropped. Differentiating $\ln L$ with respect to the parameters gives

$$\begin{aligned} \frac{\partial \ln L}{\partial \tau} &= -\frac{n}{\tau + \lambda} + \frac{\sum_{i=1}^{n} t_i}{(\tau + \lambda)^2} \\ \frac{\partial \ln L}{\partial \lambda} &= -\frac{n}{\tau + \lambda} + \frac{\sum_{i=1}^{n} t_i}{(\tau + \lambda)^2} + \frac{y - \lambda}{\sigma^2} . \end{aligned}$$

Setting the derivatives to zero and solving for τ and λ gives the ML estimators,

$$\begin{aligned} \hat{\tau} &= \frac{1}{n} \sum_{i=1}^{n} t_i - y \\ \hat{\lambda} &= y . \end{aligned}$$

1(b) [4 marks] The variances of $\hat{\lambda}$ and $\hat{\tau}$ and their covariance are

$$\begin{aligned} V[\hat{\lambda}] &= V[y] = \sigma^2 , \\ V[\hat{\tau}] &= V\left[\frac{1}{n} \sum_{i=1}^{n} t_i - y\right] = \frac{1}{n^2} \sum_{i=1}^{n} V[t_i] + V[y] = \frac{(\tau + \lambda)^2}{n} + \sigma^2 \\ \mathrm{cov}[\hat{\tau}, \hat{\lambda}] &= \mathrm{cov}\left[\frac{1}{n} \sum_{i=1}^{n} t_i - y, y\right] = -V[y] = -\sigma^2 , \end{aligned}$$

For the covariance we used the fact that t_i and y are independent and thus have zero covariance.

1(c) [3 marks] The standard deviations of $\hat{\tau}$ and $\hat{\lambda}$ can be determined from the contour of $\ln L(\tau, \lambda) = \ln L_{\max} - 1/2$, as shown in Fig. 1. The standard can be approximated by the distance from the maximum of $\ln L$ to the tangent line to the contour (in either direction).

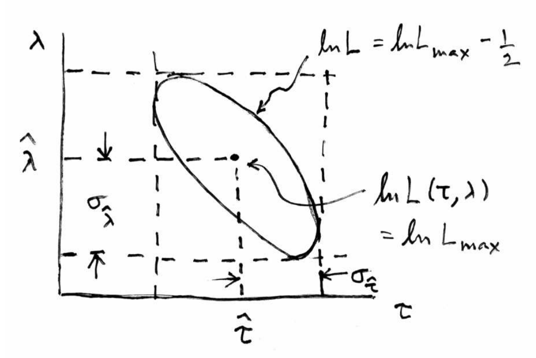


Figure 1: Illustration of the method to find $\sigma_{\hat{\tau}}$ and $\sigma_{\hat{\lambda}}$ from the contour of $\ln L(\tau, \lambda) = \ln L_{\max} - 1/2$ (see text).

If λ were to be known exactly, then the standard deviation of $\hat{\tau}$ would be less. This can be seen from Fig. 1, for example, since the distance one need to move τ away from the maximum of $\ln L$ to get to $\ln L_{\max} - 1/2$ would be less if λ were to be fixed at $\hat{\lambda}$.

1(d) [5 marks] The second derivatives of lnL are

$$
\begin{aligned}
\frac{\partial^2 \ln L}{\partial \tau^2} &= \frac{n}{(\tau+\lambda)^2} - \frac{2\sum_{i=1}^n t_i}{(\tau+\lambda)^3} , \\
\frac{\partial^2 \ln L}{\partial \lambda^2} &= \frac{n}{(\tau+\lambda)^2} - \frac{2\sum_{i=1}^n t_i}{(\tau+\lambda)^3} - \frac{1}{\sigma^2} , \\
\frac{\partial^2 \ln L}{\partial \tau \partial \lambda} &= \frac{n}{(\tau+\lambda)^2} - \frac{2\sum_{i=1}^n t_i}{(\tau+\lambda)^3} .
\end{aligned}
$$

Using $E[t_i] = \tau + \lambda$ we find the expectation values of the second derivatives,

$$
\begin{aligned}
E\left[\frac{\partial^2 \ln L}{\partial \tau^2}\right] &= \frac{n}{(\tau+\lambda)^2} - \frac{2n(\tau+\lambda)}{(\tau+\lambda)^3} = -\frac{n}{(\tau+\lambda)^2} , \\
E\left[\frac{\partial^2 \ln L}{\partial \lambda^2}\right] &= -\frac{n}{(\tau+\lambda)^2} - \frac{1}{\sigma^2} , \\
E\left[\frac{\partial^2 \ln L}{\partial \tau \partial \lambda}\right] &= -\frac{n}{(\tau+\lambda)^2} .
\end{aligned}
$$

The inverse covariance matrix of the estimators is given by

$$
V_{ij}^{-1} = -E\left[\frac{\partial^2 \ln L}{\partial \theta_i \partial \theta_j}\right]
$$

where here we can take, e.g., $\theta_1 = \tau$ and $\theta_2 = \lambda$. We are given the formula for the inverse of the corresponding 2×2 matrix, and by substituting in the ingredients we find

$$
V = \begin{pmatrix} \frac{(\tau+\lambda)^2}{n} + \sigma^2 & -\sigma^2 \\ -\sigma^2 & \sigma^2 \end{pmatrix}
$$

which are the same as what was found in (c).

Ex 2

```
// A simple program to generate exponential random numbers and
// store them in a histogram; also optionally writes the individual
// values to a file.

// Glen Cowan
// RHUL Physics
// 2 December 2006

#include <iostream>
#include <fstream>
#include <cstdlib>
#include <string>
#include <cmath>

#include <TFile.h>
#include <TH1D.h>
#include <TRandom3.h>

using namespace std;

int main(int argc, char **argv) {

// Set up output files, book histograms, add to list of histograms.

  TFile* histFile = new TFile("expData.root", "RECREATE");
  TList* hList = new TList();      // list of histograms to store
  TH1D* h1 = new TH1D("h1", "mixture of exponentials", 100, 0.0, 10.0);
  hList->Add(h1);

  string answer;
  ofstream dataFile;
  //  cout << "Also store individual values in a file? (y/n)" << endl;
  //  cin >> answer;
  answer = "y";
  bool makeDataFile = (answer == "y" || answer == "Y");
  if ( makeDataFile ) { dataFile.open("expData2.txt"); }

// Create a TRandom object to generate random numbers uniform in ]0,1]
// Use the "Marsenne Twister" algorithm of TRandom3

  int seed = 12345;
  TRandom* ran = new TRandom3(seed);

//  Fill with exponential random numbers.

  const double xi1 = 1.0;              // mean value of the exponential
  const double xi2 = 5.0;
  const double alpha = 0.2;
  int numVal = 0;
  // cout << "Enter number of values to generate: ";
  // cin >> numVal;
  numVal = 200;

  for (int i = 0; i<numVal; ++i){
    double r1 = ran->Rndm();
    double r2 = ran->Rndm();
    double x;
    if ( r1 < alpha ) {
      x = - xi1 * log(r2);
    }
    else {
      x = - xi2 * log(r2);
```

new

```
    }
    h1->Fill(x);
    if ( makeDataFile ) { dataFile << x << endl; }
  }

// Save all histograms and close up.

  hList->Write();
  histFile->Close();
  if ( makeDataFile ) { dataFile.close(); }

  return 0;

}
```

```
// A simple C++ program to illustrate the use of ROOT class TMinuit
// for function minimization.  The example shows a Maximum Likelihood
// fit for the mean of an exponential pdf in which TMinuit
// minimizes - 2*log(L).   The user must specify what to minimize in the
// function fcn, shown in this file below.

// fcn passes back f = -2*ln L by reference; this is the function to minimize.
// The factor of -2 allows MINUIT to get the errors using the same
// recipe as for least squares, i.e., go up from the minimum by 1.

// TMinuit does not allow fcn to be a member function, and the function
// arguments are fixed, so the one of the only ways to bring the data
// into fcn is to declare a pointer to the data (xVecPtr) as global.

// For more info on TMinuit see root.cern.ch/root/html/TMinuit.html .

// Glen Cowan
// RHUL Physics
// 4 December 2006

#include <iostream>
#include <fstream>
#include <cstdlib>
#include <cmath>
#include <string>
#include <vector>

#include <TMinuit.h>
#include <TApplication.h>
#include <TCanvas.h>
#include <TStyle.h>
#include <TROOT.h>
#include <TF1.h>
#include <TAxis.h>
#include <TLine.h>

using namespace std;

// Declare pointer to data as global (not elegant but TMinuit needs this).

vector<double>* xVecPtr = new vector<double>();

// The pdf to be fitted, here an exponential.
// First argument needs to be a pointer in order to plot with the TF1 class.

double expPdf2(double* xPtr, double par[]){
  double x = *xPtr;
  double xi1 = par[0];
  double xi2 = par[1];
  double alpha = par[2];
  double f = 0;
  if ( x >= 0 && xi1 > 0. && xi2 > 0. ) {
    f = alpha * (1.0/xi1) * exp(-x/xi1)  +
      (1.-alpha)*(1.0/xi2) * exp(-x/xi2);
  }                                              new

  //  if ( f <= 0. ) {
  //  cout << "expPdf2: " <<  x << "  " << xi << "  " << f << endl;
  //  }

  return f;
}
```

```
//----------------------------------------------------------------------------

//  function to read in the data from a file

void getData(vector<double>* xVecPtr){

  string infile;
  //  cout << "Enter name of input data file: ";
  // cin >> infile;
  infile = "../makeData2/expData2.txt";

  ifstream f;
  f.open(infile.c_str());
  if ( f.fail() ){
    cout << "Sorry, couldn't open file" << endl;
    exit(1);
  }

  double x ;
  bool acceptInput = true;
  while ( acceptInput ) {
    f >> x;
    acceptInput = !f.eof();
    if ( acceptInput ) {
      xVecPtr->push_back(x);
    }
  }
  f.close();

}

//----------------------------------------------------------------------------

// fcn passes back f = - 2*ln(L), the function to be minimized.

void fcn(int& npar, double* deriv, double& f, double par[], int flag){

  vector<double> xVec = *xVecPtr;             // xVecPtr is global
  int n = xVec.size();

  double lnL = 0.0;
  for (int i=0; i<n; i++){
    double x = xVec[i];
    double pdf = expPdf2(&x, par);
    if ( pdf > 0.0 ) {
      lnL += log(pdf);     // need positive f
    }
    else {
      cout << "WARNING -- pdf is negative!!!" << endl;
    }
  }
  f = -2.0 * lnL;            // factor of -2 so minuit gets the errors right

}                            // end of fcn

//----------------------------------------------------------------------------

int main(int argc, char **argv) {

  TApplication theApp("App", &argc, argv);
  TCanvas* canvas = new TCanvas();
```

```
// Set a bunch of parameters to make the plot look nice

  canvas->SetFillColor(0);
  canvas->UseCurrentStyle();
  canvas->SetBorderMode(0);        // still leaves red frame bottom and right
  canvas->SetFrameBorderMode(0);   // need this to turn off red hist frame!
  gROOT->SetStyle("Plain");
  canvas->UseCurrentStyle();
  gROOT->ForceStyle();

  gStyle->SetOptStat(0);
  gStyle->SetTitleBorderSize(0);
  gStyle->SetTitleSize(0.04);
  gStyle->SetTitleFont(42, "hxy");      // for histogram and axis titles
  gStyle->SetLabelFont(42, "xyz");      // for axis labels (values)
  gROOT->ForceStyle();

// Read in the data.  xVecPtr is global.

  getData(xVecPtr);

// Initialize minuit, set initial values etc. of parameters.

  const int npar = 3;              // the number of parameters
  TMinuit minuit(npar);
  minuit.SetFCN(fcn);

  double par[npar];               // the start values
  double stepSize[npar];          // step sizes
  double minVal[npar];            // minimum bound on parameter
  double maxVal[npar];            // maximum bound on parameter
  string parName[npar];

  par[0] = 1.0;            // a guess
  par[1] = 5.0;
  par[2] = 0.2;
  stepSize[0] = 0.1;       // take e.g. 0.1 of start value
  stepSize[1] = 0.5;
  stepSize[2] = 0.02;
  minVal[0] = 0.001;   // if min and max values = 0, parameter is unbounded.
  maxVal[0] = 100000000;
  minVal[1] = 0.001;   // if min and max values = 0, parameter is unbounded.
  maxVal[1] = 100000000;
  minVal[2] = 0.;
  maxVal[2] = 1.;
  parName[0] = "xi1";
  parName[1] = "xi2";
  parName[2] = "alpha";

  for (int i=0; i<npar; i++){
    minuit.DefineParameter(i, parName[i].c_str(),
      par[i], stepSize[i], minVal[i], maxVal[i]);
  }

// Do the minimization!

  minuit.Migrad();       // Minuit's best minimization algorithm
  double outpar[npar], err[npar];
  for (int i=0; i<npar; i++){
    minuit.GetParameter(i,outpar[i],err[i]);
  }

  cout << "fitted values and errors using mnpout..." << endl;
```

new

```
  for (int i=0; i<npar; i++){
    TString nam;
    double val;
    double err;
    double xlolim, xuplim;
    int iuint;
    minuit.mnpout(i, nam, val, err, xlolim, xuplim, iuint);
    cout << i << "   " << nam << "   " << val << "   " << err << endl;
  }
  cout << endl;

  cout << "covariance and correlation coefficients..." << endl;
  double covmat[npar][npar];
  minuit.mnemat(&covmat[0][0], npar);
  for (int i=0; i<npar; i++){
    for (int j=0; j<npar; j++){
      double sigma_i = sqrt(covmat[i][i]);
      double sigma_j = sqrt(covmat[j][j]);
      double rho = covmat[i][j]/(sigma_i*sigma_j);
      cout << i << "   " << j << "   " << covmat[i][j] << "   " << rho << endl;
    }
  }

// Plot the result.  For this example plot x values as tick marks.

  double xmin = 0.0;
  double xmax = 20.0;
  TF1* func = new TF1("funcplot", expPdf2, xmin, xmax, npar);
  func->SetMinimum(0.);
  func->SetParameters(outpar);
  func->Draw();

  func->SetLineStyle(1);              //  1 = solid, 2 = dashed, 3 = dotted
  func->SetLineColor(1);              //  black (default)
  func->SetLineWidth(1);

  func->GetXaxis()->SetTitle("x");
  func->GetYaxis()->SetTitle("f(x;#xi)");

  vector<double> xVec = *xVecPtr;
  const double tickHeight = 0.03;
  TLine* tick = new TLine();
  for (int i=0; i<xVec.size(); i++){
    tick->DrawLine(xVec[i], 0, xVec[i], tickHeight);
  }

  cout << "To exit, quit ROOT from the File menu of the plot" << endl;
  theApp.Run(true);
  canvas->Close();

  delete canvas, tick, xVecPtr;
  return 0;

}
```

new

```
PARAMETER DEFINITIONS:
   NO.   NAME          VALUE       STEP SIZE      LIMITS
     1 xi1          1.00000e+00  1.00000e-01    1.00000e-03  1.00000e+08
MINUIT WARNING IN PARAM DEF
============== LIMITS ON PARAM1 TOO FAR APART.
MINUIT WARNING IN PARAMETR
============== VARIABLE1 IS AT ITS LOWER ALLOWED LIMIT.
PARAMETER DEFINITIONS:
   NO.   NAME          VALUE       STEP SIZE      LIMITS
     2 xi2          5.00000e+00  5.00000e-01    1.00000e-03  1.00000e+08
MINUIT WARNING IN PARAM DEF
============== LIMITS ON PARAM2 TOO FAR APART.
PARAMETER DEFINITIONS:
   NO.   NAME          VALUE       STEP SIZE      LIMITS
     3 alpha        2.00000e-01  2.00000e-02    0.00000e+00  1.00000e+00
**********
**    1 **MIGRAD
**********
FIRST CALL TO USER FUNCTION AT NEW START POINT, WITH IFLAG=4.
MINUIT WARNING IN MIGrad
============== VARIABLE1 IS AT ITS LOWER ALLOWED LIMIT.
START MIGRAD MINIMIZATION.  STRATEGY  1.  CONVERGENCE WHEN EDM .LT. 1.00e-04
FCN=948.685 FROM MIGRAD    STATUS=INITIATE       26 CALLS          27 TOTAL
                    EDM= unknown      STRATEGY= 1      NO ERROR MATRIX
 EXT PARAMETER                CURRENT GUESS       STEP         FIRST
 NO.   NAME        VALUE          ERROR          SIZE      DERIVATIVE
  1  xi1          4.56366e-01   1.00000e-01  -2.08809e-03** at limit **
  2  xi2          5.00000e+00   5.00000e-01   0.00000e+00** at limit **
  3  alpha        2.00000e-01   2.00000e-02   0.00000e+00   1.24327e+01
MIGRAD MINIMIZATION HAS CONVERGED.
MIGRAD WILL VERIFY CONVERGENCE AND ERROR MATRIX.
COVARIANCE MATRIX CALCULATED SUCCESSFULLY
FCN=943.751 FROM MIGRAD    STATUS=CONVERGED      87 CALLS          88 TOTAL
                    EDM=1.99405e-06    STRATEGY= 1      ERROR MATRIX ACCURATE
 EXT PARAMETER                                    STEP         FIRST
 NO.   NAME        VALUE          ERROR          SIZE      DERIVATIVE
  1  xi1          9.25421e-01   3.12504e-01   7.49106e-07** at limit **
  2  xi2          5.17565e+00   6.03676e-01   7.48797e-07** at limit **
  3  alpha        2.68665e-01   9.12680e-02   1.93827e-03   4.65967e-03
EXTERNAL ERROR MATRIX.    NDIM=  25    NPAR=  3    ERR DEF=1
 9.766e-02 -8.195e-02 -1.925e-02
-8.195e-02  3.644e-01  3.635e-02
-1.925e-02  3.635e-02  8.450e-03
PARAMETER  CORRELATION COEFFICIENTS
      NO.  GLOBAL      1      2      3
       1  0.67011   1.000 -0.434 -0.670
       2  0.65506  -0.434  1.000  0.655
       3  0.78248  -0.670  0.655  1.000
fitted values and errors using mnpout...
0  xi1  0.925421  0.312504
1  xi2  5.17565  0.603676
2  alpha  0.268665  0.091268

covariance and correlation coefficients...
0  0  0.0976589  1
0  1  -0.0819479  -0.434388
0  2  -0.0192495  -0.67008
1  0  -0.0819479  -0.434388
1  1  0.364424  1
1  2  0.0363498  0.655032
2  0  -0.0192495  -0.67008
2  1  0.0363498  0.655032
```

should have $\hat{\xi}_1 \approx 1$

$\hat{\xi}_2 \approx 5$

$\hat{\alpha} \approx 0.2$

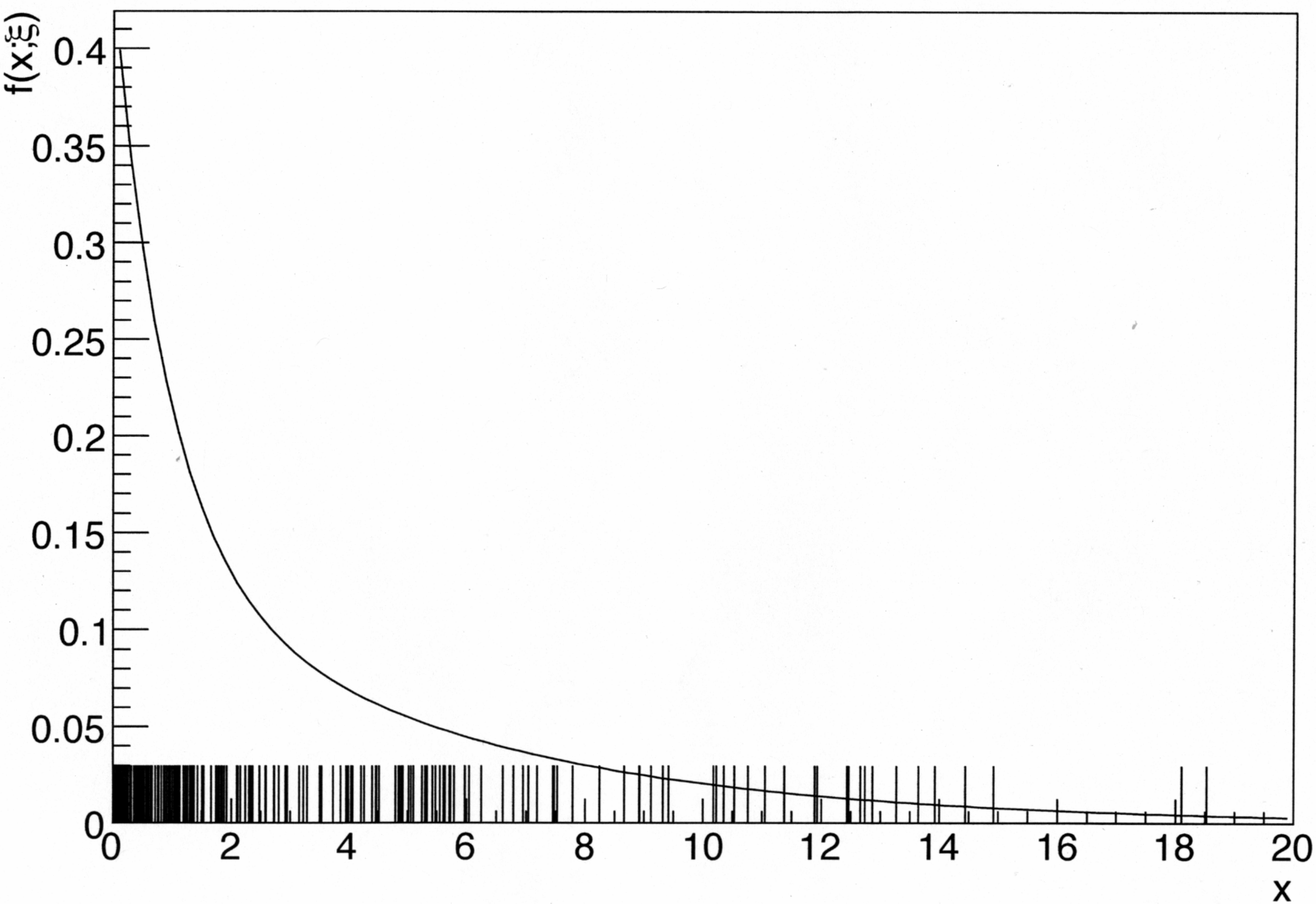
But might find

$\hat{\xi}_1 \approx 5$

$\hat{\xi}_2 \approx 1$

$\hat{\alpha} \approx 0.8$

2 2 0.00845031 1
To exit, quit ROOT from the File menu of the plot



f(x;ξ)
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0
2
4
6
8
10
12
14
16
18
20
x