The Monte Carlo Simulation Package: ColquhounHawkesMC

Load the package. You should start a fresh *Mathematica* session before loading this package.

```
<<ChannelKinetics`ColquhounHawkesMC`
```

```
C routine installation successful.
```

You can get a list of the functions defined in this package by typing

```
Help[]
```

```
The following functions are available:
      CSeedRandom[n]
      SingleTrace[q, sclist, initstate,
                  height, duration, u, {t, tmax, tstep}]
      TraceTable[q, sclist, initstate,
                 height, duration, n, u, {t, tmax, tstep}]
      MeanCurrent[q, sclist, initstate,
                  height, duration, n, u, {t, tmax, tstep}]
      CurrentVariance[option, q, sclist, initstate,
                      height, duration, n, u, {t, tmax, tstep}]
      CoefficientOfVariation[option, q, sclist, initstate,
                             height, duration, n, u,
                             {t, tmax, tstep}]
      CurrentThirdMoment[option, q, sclist, initstate,
                         height, duration, n, u, {t, tmax, tstep}]
      FilterTest[EnsembleSize -> m, q, sclist, initstate, height,
                 duration, u, {t, tmax, samplingfreq, fc}]
      PeakEPSCAmplitudeDistribution[
      EnsembleSize -> m, q, sclist, initstate, height, duration,
      n, u, {t, tmax, samplingfreq, fc, tint}]
      DistributionPlot[distr]
      Help[],

 where option is an optional argument of the form
      EnsembleSize -> m
 (m = 1 by default).

 You can get detailed information on each function by typing
      ?function
 followed by SHIFT-RETURN or ENTER.
 The definition of a function (Mathematica code) is available at
      ??function
 followed by SHIFT-RETURN or ENTER, as usual.
```

Copy **q** and **stateConductivityList** from the section above, and newly add in the variable **initialState** such that at **t = 0**, all channels are in state **C1**. Note that **initialState** is not the same type of expression as **initialOccupancy**.

```
q = {
      {-100000.*c[t], 100000.*c[t], 0, 0},
      {50., -64., 4., 10.},
      {0, 0.02, -0.02, 0},
      {0, 0.5, 0, -0.5}
    }; (* c[t] in M *)

stateConductivityList = { (* conductivity in pS *)
                          {C1, 0},
                          {C2, 0},
                          {C3, 0},
                          {O,  8}
                        };

initialState = C1;
```

Simulations in **ColquhounHawkesMC** are done using a random number generator. (The routine used is due to L'Ecuyer, "Efficient and Portable Combined Random Number Generators", Commun. ACM **31**, 742-774 (1988), implemented in C by Eberhard v. Kitzing.) When the package is loaded, it takes the time of day as the seed for the random number generator. Two different *Mathematica* sessions will therefore almost always give different sequences of random numbers and, therefore, different results of the simulations. If you want to make sure that you always get the same sequence of pseudorandom numbers, you can explicitly give an integer seed for the random number generator, using **CSeedRandom[]**.

```
?CSeedRandom
```

CSeedRandom[n] resets the pseudorandom number generator for the C routines, using the integer n as a seed. The sign of n is neglected, and it must not be zero.

An example call.

```
CSeedRandom[6328]
```

CSeedRandom[] gives a seed to the random number generator called by the C routines. It is different from *Mathematica*'s built-in **SeedRandom[]** which resets the seed for *Mathematica*'s own random number generator.

Now we turn to doing some simulations. First, we try simulation of the current flowing through a single channel.

```
?SingleTrace
```

SingleTrace[q, sclist, initstate, height, duration, u, {t, tmax, tstep}] calls a C routine that performs a Monte Carlo simulation of the state transitions of a single channel, from t = 0 to tmax milliseconds, using a sampling interval of tstep ms; q (ms^-1) is the transition rate matrix; sclist has entries {state, conductivity (pS)}; initstate is the initial state of the channel; height (M) and duration (ms) define the transmitter concentration square pulse function; u (V) is the voltage across the membrane minus the reversal potential. SingleTrace[] returns a list of real numbers of length Floor[tmax / tstep] + 1 that gives the single channel current (pA) at ascending integer multiples of tstep.

Prepare some variables for the simulation (alternatively, you might wish to give these values directly as arguments to **SingleTrace[]**):

```
height = 0.001; (* M *)
duration = 1;   (* ms *)
u = 0.07;       (* V *)
tmax = 10;      (* ms *)
tstep = 0.2;    (* ms *)
```

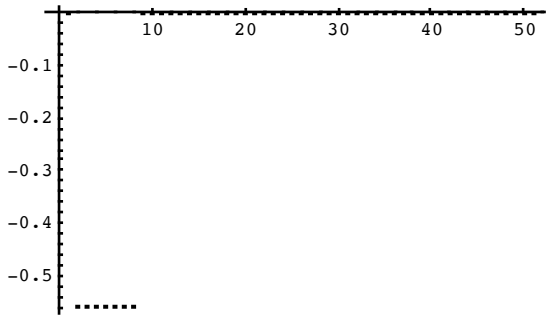
Invoke **SingleTrace[]**.

```
myTrace =
SingleTrace[q, stateConductivityList, initialState,
            height, duration, u, {t, tmax, tstep}]
```

```
{0, -0.56, -0.56, -0.56, -0.56, -0.56, -0.56, -0.56, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0}
```

This is a list of numbers that represents the single channel current at sample points with distance **tstep** milliseconds. We plot it using the built-in *Mathematica* function **ListPlot[]**.

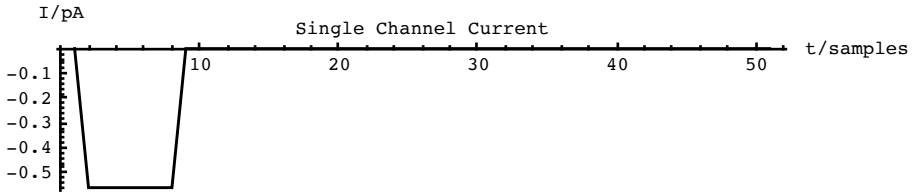
```
ListPlot[myTrace]
```




```
-Graphics-
```

We can make this look better by giving some options to **ListPlot[]**:

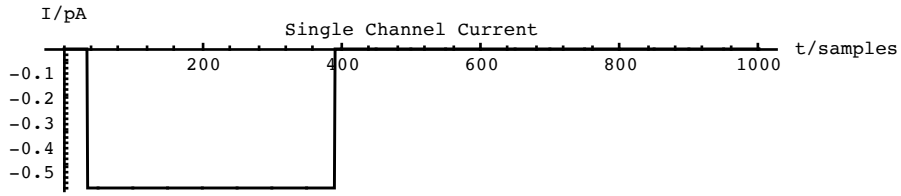
```
ListPlot[myTrace, PlotJoined -> True, PlotRange -> All,
         AspectRatio -> 1 / 5,
         AxesLabel -> {"t/samples", "I/pA"},
         PlotLabel -> "Single Channel Current"]
```




```
-Graphics-
```

A smaller setting for **tstep** will make the transitions steeper.

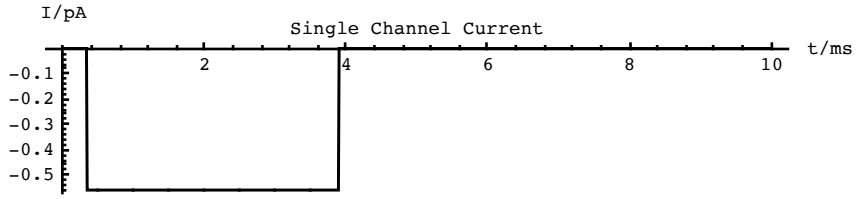
```
tstep = 0.01; (* ms *)
myTrace =
SingleTrace[q, stateConductivityList, initialState,
            height, duration, u, {t, tmax, tstep}];
ListPlot[myTrace, PlotJoined -> True, PlotRange -> All,
         AspectRatio -> 1 / 5,
         AxesLabel -> {"t/samples", "I/pA"},
         PlotLabel -> "Single Channel Current"]
```




```
-Graphics-
```

We can easily change the unit of the x-axis to be milliseconds.

```
ListPlot[Transpose[{Range[0, tmax, tstep], myTrace}],
         PlotJoined -> True, PlotRange -> All,
         AspectRatio -> 1 / 5,
         AxesLabel -> {"t/ms", "I/pA"},
         PlotLabel -> "Single Channel Current"]
```




```
-Graphics-
```

Using **`TraceTable[]`**, we generate a list of traces.

```
?TraceTable
```

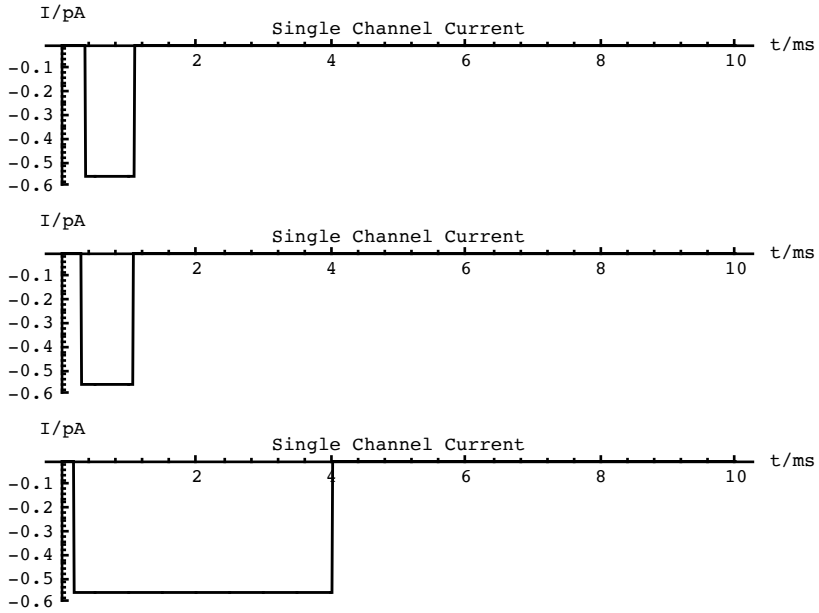
TraceTable[q, sclist, initstate, height, duration, n, u, {t, tmax, tstep}] returns a list of n independent single channel current functions of t. See ?SingleTrace for information on the other arguments.

```
n = 3; (* 3 traces *)

myTraces =
TraceTable[q, stateConductivityList, initialState,
           height, duration, n, u, {t, tmax, tstep}];
```

This error message is not always useful.

We map **`ListPlot[]`** on every element of the trace list.

```
Map[
    ListPlot[Transpose[{Range[0, tmax, tstep], #}],
             PlotJoined -> True, PlotRange -> {-0.6, 0},
             AspectRatio -> 1 / 5,
             AxesLabel -> {"t/ms", "I/pA"},
             PlotLabel -> "Single Channel Current"] &,
    myTraces];
```



Simulations of the current flowing through an ensemble of channels are done by

```
?MeanCurrent
```

MeanCurrent[q, sclist, initstate, height, duration, n, u, {t, tmax, tstep}] calls a C routine that adds up n independent single channel current functions (pA) and divides the sum by n. See ?SingleTrace for information on the other arguments.

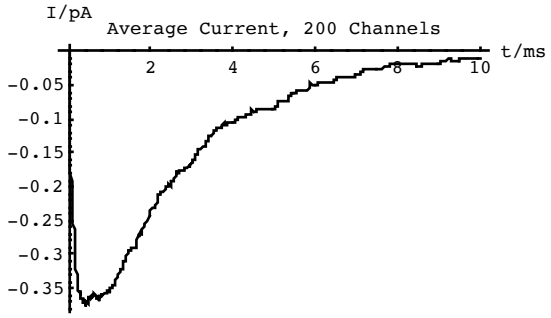
MeanCurrent[], as well as some other functions, take the same sequence of arguments as **TraceTable[]**.

The following computes a mean of 200 single channel traces.

```
n = 200;
myMean =
MeanCurrent[q, stateConductivityList, initialState,
            height, duration, n, u, {t, tmax, tstep}];
```

It looks like this.

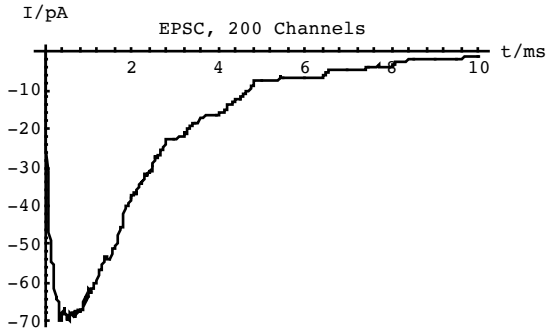
```
ListPlot[Transpose[{Range[0, tmax, tstep], myMean}],
         PlotJoined -> True, PlotRange -> All,
         AxesLabel -> {"t/ms", "I/pA"},
         PlotLabel -> "Average Current, 200 Channels"]
```



-Graphics-

This gives the sum of 200 single channel traces.

```
n = 200;
myEPSC =
n MeanCurrent[q, stateConductivityList, initialState,
              height, duration, n, u, {t, tmax, tstep}];

ListPlot[Transpose[{Range[0, tmax, tstep], myEPSC}],
         PlotJoined -> True, PlotRange -> All,
         AxesLabel -> {"t/ms", "I/pA"},
         PlotLabel -> "EPSC, 200 Channels"]
```




```
-Graphics-
```

Write the simulated EPSC to a file, with the numbers separated by the End-Of-Line character.

```
ColumnForm[myEPSC] >> file
```

The following three functions have close analogs in the **ColquhounHawkes** exact solution package. We won't discuss them here, but give their help texts.

```
?CurrentVariance
```

CurrentVariance[option, q, sclist, initstate, height, duration, n, u, {t, tmax, tstep}] gives the variance function (pA^2) of n independent single channel current functions of t. Division by n-1 (rather than n) is used. The option EnsembleSize -> m causes m-membered channel ensembles to be simulated instead of single channels. See ?SingleTrace for information on the other arguments.

```
?CoefficientOfVariation
```

CoefficientOfVariation[option, q, sclist, initstate, height, duration, n, u, {t, tmax, tstep}] gives the "coefficient of variation" (i.e. standard deviation / mean) of n independent single channel current functions of t in percent. Division by n-1 (rather than n) is used. The option EnsembleSize -> m causes m-membered channel ensembles to be simulated instead of single channels. See ?SingleTrace for information on the other arguments.

```
?CurrentThirdMoment
```

CurrentThirdMoment[option, q, sclist, initstate, height, duration, n, u, {t, tmax, tstep}] gives the third central moment function (pA^3) of n independent single channel current functions of t. Division by n is used. The option EnsembleSize -> m causes m-membered channel ensembles to be simulated instead of single channels. Since the current is negative by convention, the third moment undergoes the same sign change. See ?SingleTrace for information on

the other arguments.

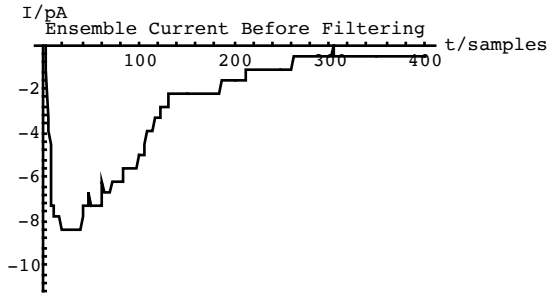
`FilterTest[]` is useful for determinimg how the chosen filter acts on a simulated EPSC. You can visually check the effects of different cutoff and sampling frequencies.

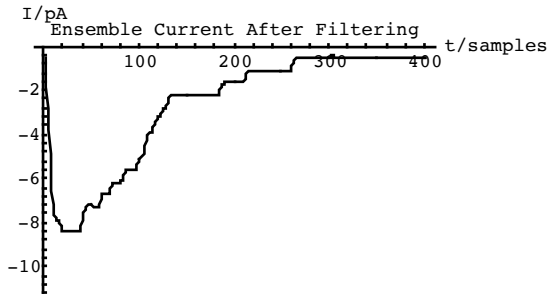
```
?FilterTest
```

```
FilterTest[EnsembleSize -> m, q, sclist, initstate,
  height, duration, u, {t, tmax, samplingfreq,
  fc}] plots the input and output function of a
  gaussian filter whose cutoff frequency is fc
  kilohertz. The input function is prepared by
  simulating a m-membered channel ensemble from 0
  to tmax milliseconds, sampling the current at
  samplingfreq kilohertz. See ?SingleTrace for
  information on the other arguments.
```

A good choice for **`samplingfreq`** and for the cutoff frequency **`fc`** is 40 kHz and 3 kHz, respectively.

```
height = 0.001;     (* M *)
duration = 1;       (* ms *)
u = 0.07;           (* V *)
tmax = 10;          (* ms *)
samplingfreq = 40; (* kHz *)
fc = 3;             (* kHz *)

FilterTest[EnsembleSize -> 20,
           q, stateConductivityList, initialState,
           height, duration, u,
           {t, tmax, samplingfreq, fc}]
```






```
{-Graphics-, -Graphics-}
```

The construction of a histogram for the distribution of the peak EPSC amplitudes is done by

```
?PeakEPSCAmplitudeDistribution
```

```
PeakEPSCAmplitudeDistribution[EnsembleSize -> m,
  q, sclist, initstate, height, duration, n, u,
  {t, tmax, samplingfreq, fc, tint}] gives the
```

unnormalized distribution of the peak currents
of n independently simulated m-membered channel
ensembles. Each simulation extends from 0 to
tmax milliseconds. The simulation results are
sampled at samplingfreq kilohertz, filtered
using a gaussian filter with cutoff frequency fc
kilohertz, and integrated in an interval of tint
milliseconds around the peak. The bin width is u
times the maximum single channel conductivity.
The distribution is returned in a form suitable
for DistributionPlot[]. See ?SingleTrace for
information on the other arguments.

`EnsembleSize -> m` defines the number of available channels, while **`n`** is the number of trials.

In the following two simulations we use an integration interval of length 0.5 ms around the peak, that is, from (time_of_the_peak) - 0.25 ms to (time_of_the_peak) + 0.25 ms.

```
tint = 0.5;  (* ms *)

m100n200 = (* 100 available channels, 200 trials *)
PeakEPSCAmplitudeDistribution[EnsembleSize -> 100,
q, stateConductivityList, initialState,
height, duration, 200, u,
{t, tmax, samplingfreq, fc, tint}]

m200n121 = (* 200 available channels, 121 trials *)
PeakEPSCAmplitudeDistribution[EnsembleSize -> 200,
q, stateConductivityList, initialState,
height, duration, 121, u,
{t, tmax, samplingfreq, fc, tint}]
```

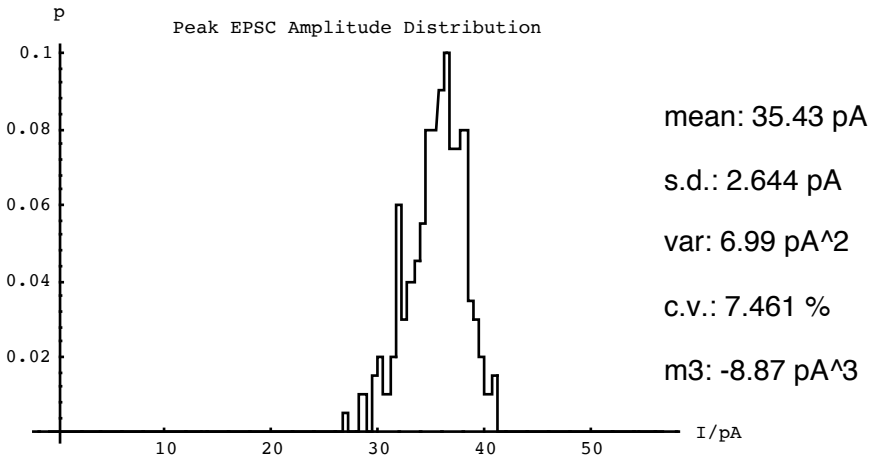
```
{0.56, 35.4316, 6.9881, -8.87153,
  {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 2, 0, 3,
   4, 2, 4, 12, 6, 8, 9, 11, 16, 16, 18, 20, 15, 15, 16,
   7, 6, 4, 2, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}}

{0.56, 70.445, 15.4601, 4.41487,
  {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   1, 0, 1, 0, 2, 2, 0, 1, 0, 6, 7, 5, 2, 6, 4, 8, 11, 7,
   4, 8, 5, 4, 7, 2, 6, 3, 6, 1, 5, 1, 1, 1, 0, 3, 0, 0,
   1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0}}
```

Once the distributions have been computed, you can plot them in various ways.

```
?DistributionPlot
```

DistributionPlot[distr] normalizes the distribution distr and plots it, giving the mean, variance etc. at the right side of the plot. DistributionPlot[distr1, distr2, ...] does the same for a superposition of the distri.

```
DistributionPlot[m100n200]
```



mean: 35.43 pA

s.d.: 2.644 pA

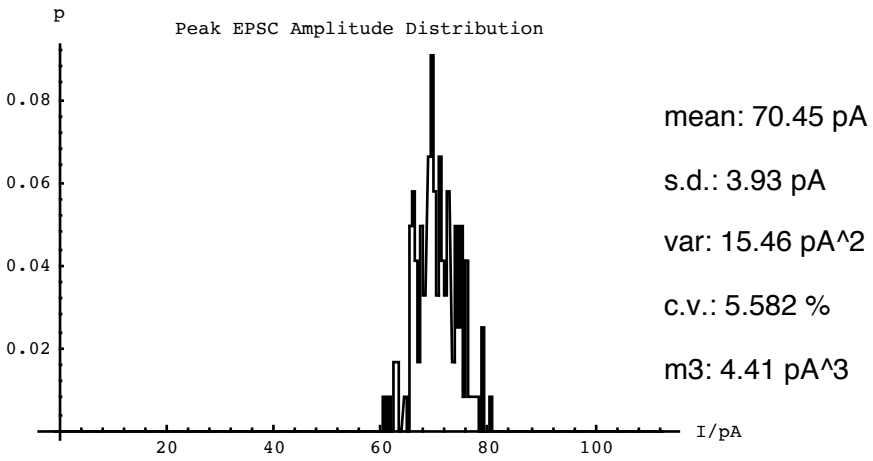
var: 6.99 pA^2

c.v.: 7.461 %

m3: -8.87 pA^3

```
-Graphics-
```

```
DistributionPlot[m200n121]
```



mean: 70.45 pA

s.d.: 3.93 pA

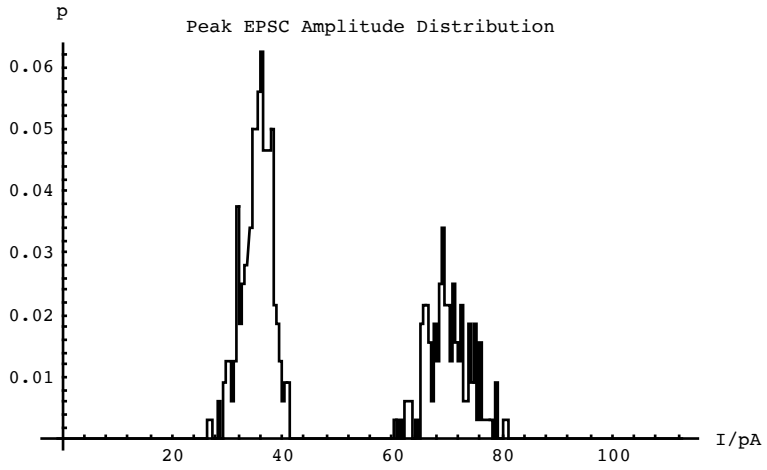
var: 15.46 pA^2

c.v.: 5.582 %

m3: 4.41 pA^3

```
-Graphics-
```

```
DistributionPlot[m100n200, m200n121]
```



-Graphics-