TSpectrum package for ROOT

(Root 6)

TSpectrum: Package for analysis of 1-2-3 dimensional spectra containing peaks,
Author: Miroslav Morháč, Slovak Academy of Science. In ROOT it functions as a set of classes.

Help:

- root.cern/root/htmldoc/guides/spectrum/Spectrum.pdf : Manual for the package. Helps conceptually, but the implementation in ROOT is different...
- root.cern.ch/doc/master/group__Spectrum.html : Package implementation in ROOT
- root.cern.ch/doc/master/group__tutorial__spectrum.html : Demo macros
  www.fuw.edu.pl/~kpias/ctnp/tspectrum : Selection of macros *compiled didactically*
  $ROOTSYS/tutorials/spectrum/ : Selection of macros

What does TSpectrum aim to do?

- Smoothing of a spectrum
- Extraction of background under peaks (without assuming their shape)
- Search of peak positions
- Deconvolution (reversal of convolution of true distribution with the detector response function)
- Serial fit of many peaks (limitation: each peak has the same dispersion)
- Transforms (e.g Fourier)
- Visualisation

Notice:

The demo codes on the subsequent pages originate from authors of ROOT.
They were just adjusted didactically and placed on the subject's website.
Using these would require to download the following ROOT file with exemplary spectra:

```
> wget www.fuw.edu.pl/~kpias/ctnp/tspectrum/TSpectrum.root
```

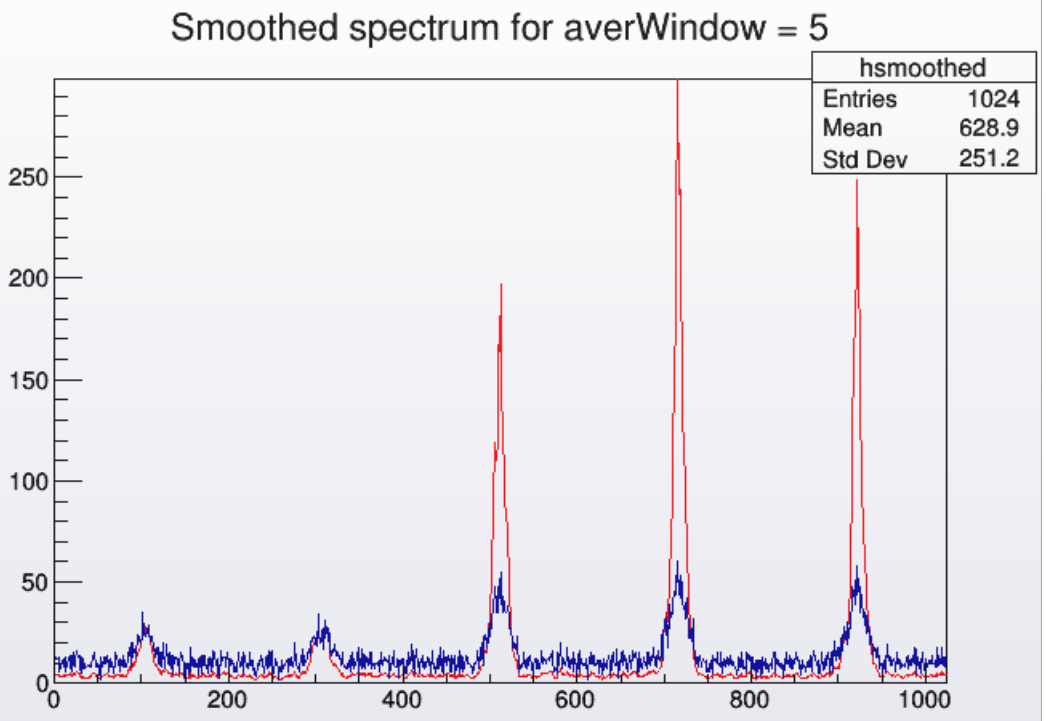
krzysztof.piasecki@fuw.edu.pl

Smooothing

- The basis of the algorithm is the discrete Markov chain[ref]. For subsequent bins, from a set of neighbouring points of width `averWindow` – it creates a distribution *U*(x), sensitive to peaks, and suppressing noise.
  A number of neighbouring points taken for averaging ∈ [3, 15].
  You can set it by giving constants: `kBackSmoothing3, 5, ..., 15 .`

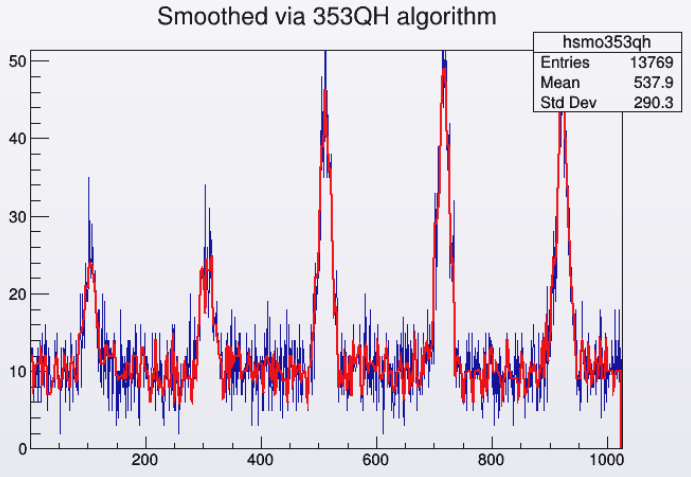
```
const char* TSpectrum::SmoothMarkov (
   Double_t* source,      :  pointer to input array with histogram values (after operation: output array)
   Int_t ssize,           :  number of histogram bins
   Int_t averWindow       :  width of smoothing window.  Possible values:
                                kBackSmoothing3, ...5, ..., ...15
);
```

Example: Smoothing.C



Notice. Also TH1 has a smoothing method.
It implements the 353QH algorithm [Ref, p. 295]

```
void TH1::Smooth   (
  Int_t ntimes = 1,
  Option_t* option = ""
);
```



Finding background under peaks

① For each point the algorithm sets a new value, being a minimum calculated from neighbours.
② Width of scope for neighbours is iterated from `1` to `numberIterations` or backwards. (*Width* = 2 ·*nIterations*)
③ A filter of order `filterOrder` is performed, that collects information from *N* = `filterOrder` neighbours.
For 2nd order the result is the min. between a given point and an average of neighbour pair at the window edges.
④ You can require the spectrum smoothing before the background search. Also here you can steer the window width.

```
const char* TSpectrum::Background (
  Double_t* spectrum,       : pointer to input array with histogram values (after operation: output array)
  Int_t ssize,              : number of histogram bins
  Int_t numberIterations,   : maximal width of window of neighbours (width = 2 ·nIterations)
  Int_t direction,          : Should window get widen or shrunk: kBackIn{De}creasingWindow
  Int_t filterOrder,        : Order of filter applied in each step: kBackOrder2, …4, …6, …8
  bool  smoothing,          : true if spectrum should be smoothed before the background search
  Int_t smoothWindow,       : width of smoothing window, kBackSmoothing3, …5, …, …15
  bool  compton             : true if the algorithm should account for Compton edges
);
```

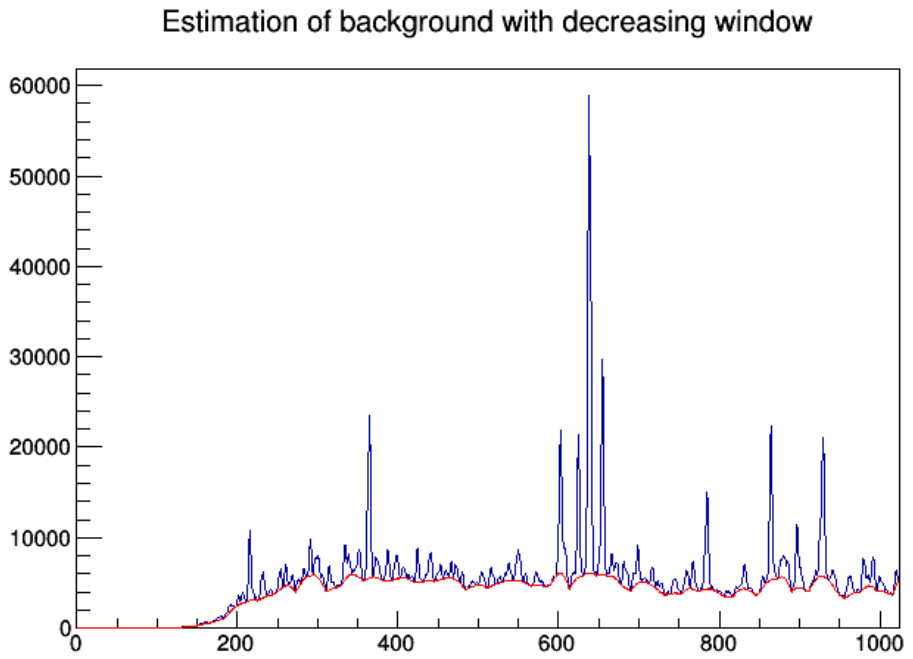
Examples:

1. Background_decr.C
2. Background_incr.C
3. Background_width.C

Note: Another TH1-based function is possible too:

```
TH1* TSpectrum::Background (
 const TH1* h,
 Int_t niter = 20,
 Option_t* option = ""
);
```

Modes of operation are given through the option.

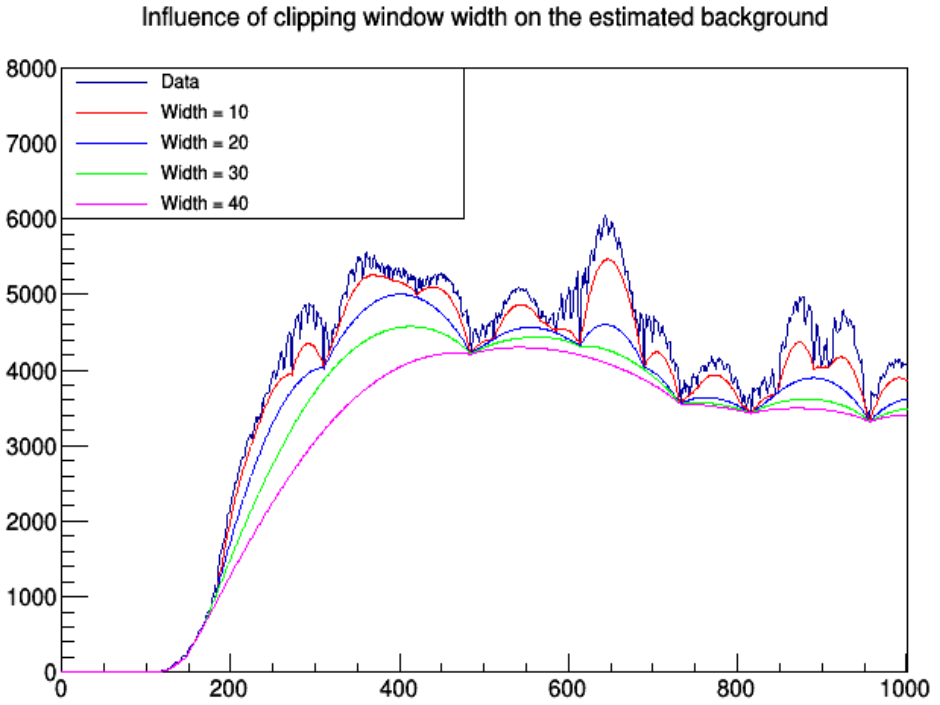


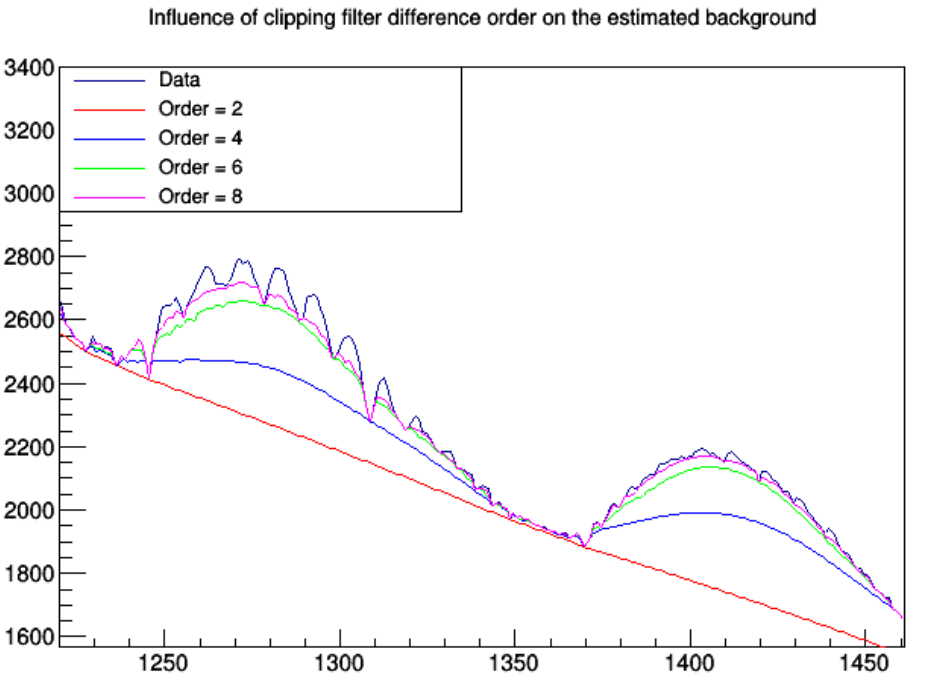
Fitting background under peaks , cont.

Sometimes the profile background is ambiguous, and/or depends on interpretation.

An exemplary Background_width2.C macro shows, how for the spectrum of unclear background interpretation, different widths of filtering window change the result.

In turn, the Background_order.C macro shows the results of filters of increasing order applied to another data with ambiguous background.





Peak searcher

```
Int_t TSpectrum::Search (
 Const TH1* hin,              :   pointer to the input histogram
 Double_t   sigma = 2,        :   estimated width of searched peaks
 Option_t*  option = ""       :   see below
 Double_t threshold = 0.05 :   [0..1]. Request to omit peaks whose  Amplitude  ≤  threshold*HighestAmp
);
```

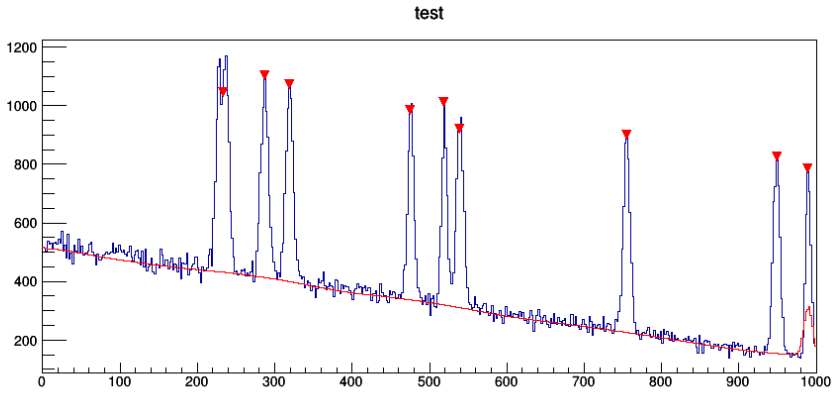
A number of found peaks and sets of their [X, Y] positions are retrievable via :

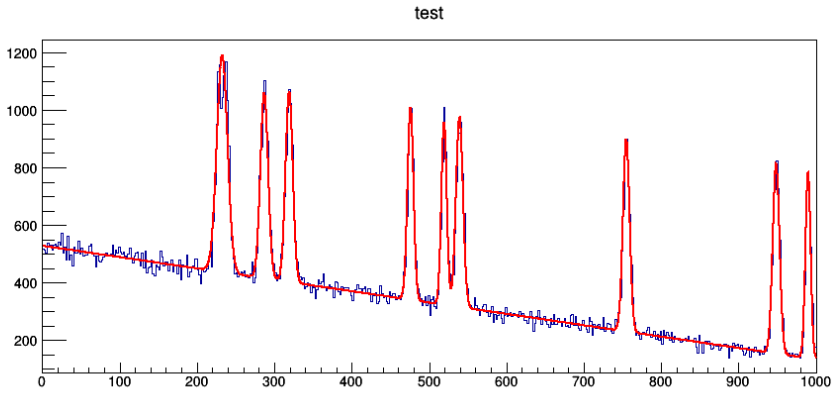
`int TSpectrum::GetNPeaks()` and `Double_t* GetPositionX () , Double_t* GetPositionY ()`

Internally, `Search` copies data into an array. For this array, it:

① removes the background
② smoothens the histogram (array, not the original object) ,
③ makes a deconvolution, response funct. is Gaus. with sigma.
④ adds the collection of markers with positions of peaks
⑤ draws the histogram with found peaks.

To stop these actions, inside `options` place, respectively,
① `nobackground`, ② `nomarkov` , ④ `goff` , ⑤ `nodraw` .





Example: peaks.C

Note: Search is a wrapper for this method:

```
TH1* TSpectrum::SearchHighRes (...);
```

with some parameters set as defaults.
For more manipulations it can be also accessed directly.

Deconvolution

If we measure some physics distribution $R(x)$ experimentally, usually our detector smears the original distribution due to its internal response function, $D(x)$. An obtained spectrum $W(x)$ is a following convolution of both:

$$W(x) = R(x) \circ D(x) = \int R(z) \cdot D(x - z)\, dz = \sum_z R(z) \cdot D(x - z)$$

If we know the detector response function, $D(x)$, the deconvolution allows to bring us back from the smeared measurement, $W(x)$, to an original distribution, $R(x)$.

```
Int_t TSpectrum::Deconvolution (
 Double_t* source,                 :  pointer to table with values of W(x) histogram
 const Double_t* response,         :  pointer to table with values of response function, D(x)
 Int_t ssize,                      :  number of bins of W(x) histogram [ will be the same for D(x) ]
 Int_t numberIterations,           :  in case of boost option [ref]: do boost after how many iterations
 Int_t numberRepetitions,          :  in case of boost option: how many repetitions of boost
 Double_t boost                    :  in case of boost option: the boost exponent
);
```

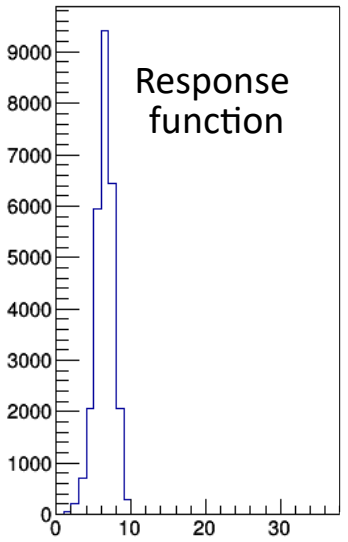
A resulting physics distribution, $R(x)$, is placed into the `source` pointer.

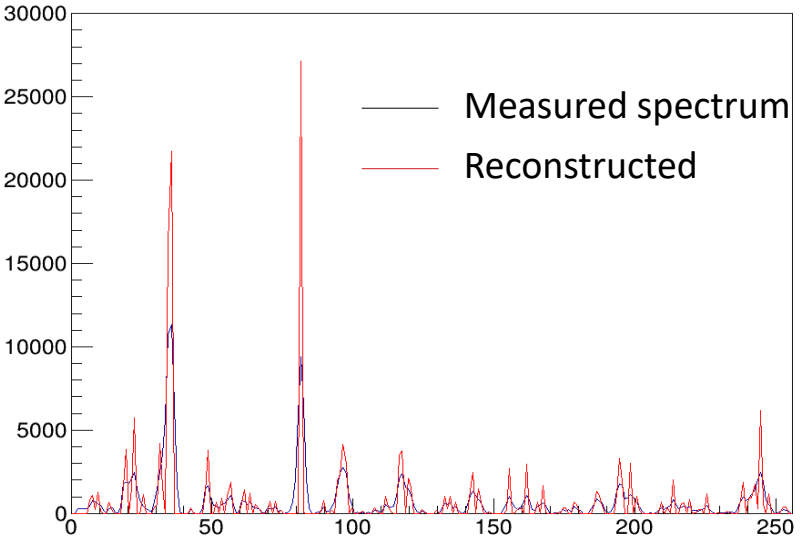
Examples:

1. Deconvolution.C
2. Deconvolution_wide.C
3. Deconvolution_wide_boost.C
4. DeconvolutionRL_wide.C
5. DeconvolutionRL_wide_boost.C

Note: ROOT offers an alternative algorithm, based on Richardson–Lucy method:

```
TH1* TSpectrum::DeconvolutionRL (...);
```





Serial peak fitter

TSpectrum offers a dedicated **AWMI** (Algorithm Without Matrix Inversion) tool [ref]. **Assumptions**:

① A sum of peaks is fitted. Each peak is described by the Gaussian function + possible "tail":

$$f(i,a) = \sum_{j=1}^{M} A(j) \left\{ exp\left[\frac{-(i-p(j))^2}{2\sigma^2}\right] + \frac{1}{2}T \cdot exp\left[\frac{(i-p(j))}{B\sigma}\right] \cdot erfc\left[\frac{(i-p(j))}{\sigma} + \frac{1}{2B}\right] + \frac{1}{2}S \cdot erfc\left[\frac{(i-p(j))}{\sigma}\right] \right\}$$

② All the peaks are characterized by the same dispersion (σ)
③ Parameters of the function above are not correlated together.

Proposed algorithm (⟶ see **FitAwmi.C** example)

⓪ Perform the initial peak search, using **TSpectrum::SearchHighRes** tool
① Create an instance of **TSpectrumFit** object, which manages the analyser.
② Set the analysis options by **TSpectrumFit::SetFitParameters**
③ Set the initial values of function parameters, using **TSpectrumFit::SetPeakParameters**
④ Perform the fit by invoking **TSpectrumFit::FitAwmi** method.
⑤ Access to the results (parameters and their uncertainties) is via getter of `TSpectrumFit` object: `GetPositions`, `GetAmplitudes`, `GetAreas`, `GetSigma`, `GetTailParameters` and `Get...Errors` .

Caution on the conventions:

① The analyser works on X-axis interpretted binwise. All the X-axis-related results work under this assumption. One should multiply these paramets (and uncertainties) by bin width, i.e. `TH1F::GetBinWidth (1)` .

② Values given by `GetSigma` must be additionally divided by $\sqrt{2}$, in order that they have the meaning of dispersion and its uncertainty.

Example: FitAwmi.C

① Generate histogram with series of peaks with pulled values of: σ (unique) and amplitudes (per peak)

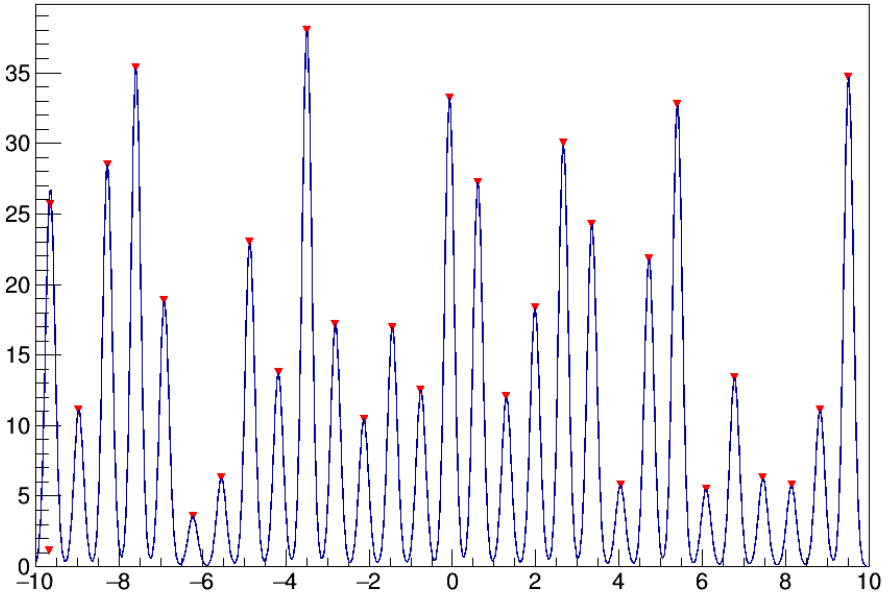
```
Created peak at -9.65769 with [Amp, Area]:  [26.7323 , 7.6458]
Created peak at -8.97307 with [Amp, Area]:  [11.0707 , 3.16637]
...
Total number of created peaks = 29 with sigma = 0.114103
```

② Initial search for a number of peaks and their centroids, using `TSpectrum::SearchHighRes`

```
* SearchHighRes has found: 30 peaks.
```

③ Set the method and starting values of parameters: dispersion σ , positions and amplitudes
④ Perform the fit by invoking `TspectrumFit::FitAwmi`
⑤ Extract the obtained parameters

```
* Overall sigma found by fit: 0.114101 (+-1.38523e-05)
Found: -3.49614 (+-9.10244e-05) 38.0106 (+-0.0299651) 10.8714 (+-0.000279681)
Found: -7.60384 (+-9.4716e-05) 35.3259 (+-0.0288984) 10.1035 (+-0.000269724)
...
```



Transforms

Transforms of signals are performed by the **TSpectrumTransform** class.
Possible types of transforms (decompositions into harmonics) :

Fourier, cosine, sine, Haar, Walsh, Hartley and their combinations.

General mode of operation:

① Create an object of **TSpectrumTransform** class, giving the size of signal sample.
② Set the desired type of transform via the **SetTransformType** method and the direction (→ / ←) using **SetDirection**.
③ Perform the transform via **Transform (Double_t* source, Double_t* dest)** method

Note for Fourier transform (and mixed ones incorporating Fourier) :

For the "direct" transform, the destination array (dest) should be of 2× larger size than that of source.
For the "backward" transform, the source array should be of 2× larger size than that of destination.

Example: FourierTransform.C

This macro creates the signal of a type:

$$S(N) = const + \sum_{R=1}^{R_{max}} \cos\left(\frac{2\pi}{T_0/R} \cdot N\right)$$

Input signal → Result of Fourier transform