Biblioteka TSpectrum

(Root 6)

TSpectrum: pakiet do analizy widm 1-2-3 wymiarowych, posiadających maksima (“peaki”),
Autor: Miroslav Morháč [Słowacka Akademia Nauk]. W ROOT funkcjonuje jako zestaw klas.

Pomoc:

- root.cern/root/htmldoc/guides/spectrum/Spectrum.pdf : Manual dla pakietu. Pomaga koncepcyjnie, ale implementacja w ROOT jest inna...
- root.cern.ch/doc/master/group__Spectrum.html : Implementacja pakietu w ROOT
- root.cern.ch/doc/master/group__tutorial__spectrum.html : Makra demonstracyjne
  www.fuw.edu.pl/~kpias/nkfj/tspectrum : Wybór makr *opracowany dydaktycznie*
  $ROOTSYS/tutorials/spectrum/ : Wybór makr

Co stara się robić TSpectrum ?

- Wygładzanie widma (smoothing)
- Wyznaczanie tła pod peakami
- Znajdywanie maksimów peaków
- Dekonwolucja (odwrócenie splotu widma z funkcją odpowiedzi detektora)
- Grupowe dopasowanie peaków (fit; ograniczenie: każdy peak ma tę samą dyspersję)
- Transformaty (np. Fouriera)
- Wizualizacja

Uwaga:

Kody demonstracyjne na następnych stronach pochodzą od autorów środowiska ROOT.
Zostały usprawnione dydaktycznie i umieszczone na stronie www Wykładu.
Potrzebny będzie plik z przykładowymi widmami:

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

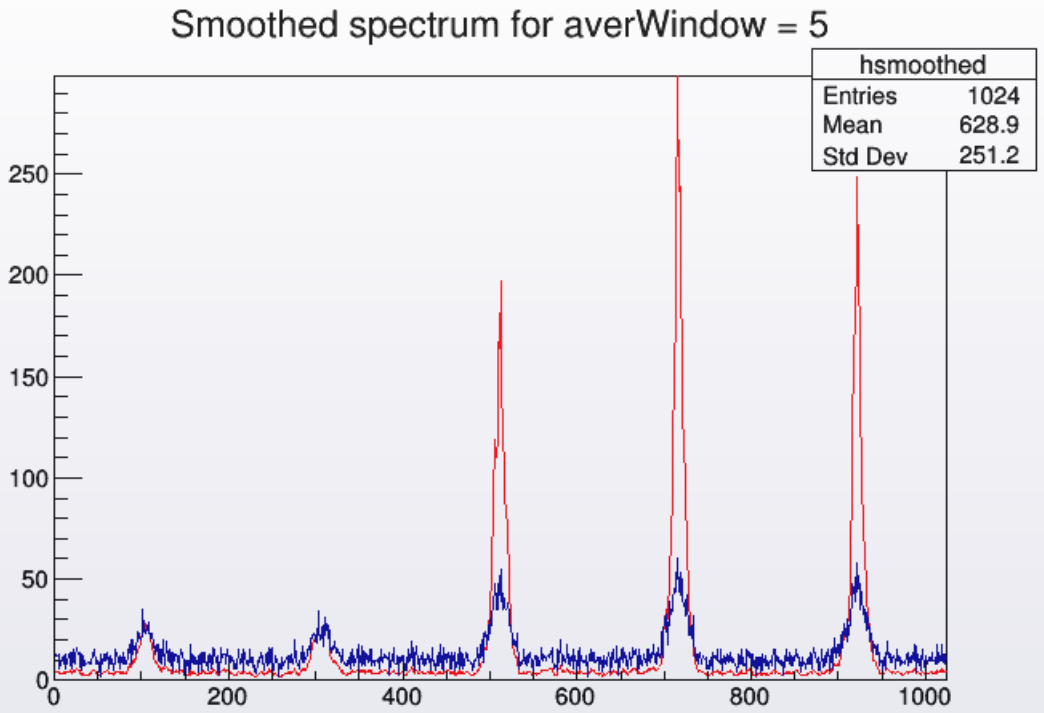
krzysztof.piasecki@fuw.edu.pl

Wygładzanie (smooothing)

- Zasadą działania jest dyskretny łańcuch Markova[ref]. Dla kolejnych słupków tworzy się z zestawu sąsiadów o rozpiętości `averWindow` rozkład *U*(x), czuły na peaki, a tłumiący szum.
  Ilość punktów sąsiednich branych do uśredniania ∈ [3, 15].
  Szerokością okna wygładzania sterujemy, podając stałe `kBackSmoothing3, 5, ..., 15`.

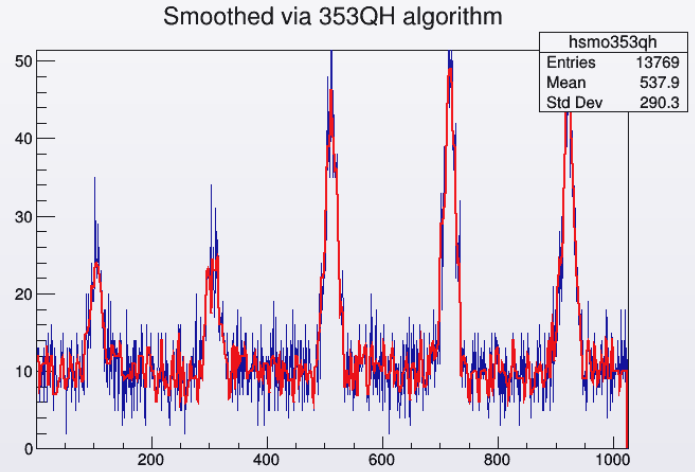
```
const char* TSpectrum::SmoothMarkov (
   Double_t* source,      : wskaźnik do tablicy z wartościami histogramu
   Int_t ssize,           : ilość słupków histogramu
   Int_t averWindow       : szerokość okienka wygładzania. Możliwe wartości:
                            kBackSmoothing3, ...5, ..., ...15
);
```

Przykład: Smoothing.C



Uwaga. W TH1 zaimplementowane jest wygładzanie metodą 353QH: [Ref, s. 295]

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



Wyznaczanie tła pod peakiem

① Algorytm dla każdego punktu wyznacza nową wartość w oparciu o sąsiadów.
② Szerokość okna sąsiadów jest iterowana od `1` do `numberIterations` lub odwrotnie. (*Szerokość = 2 · nIterations*)
③ Wykonywany jest filtr rzędu `filterOrder`, tzn. zbierający informację z liczby sąsiadów = `filterOrder`.
Dla rzędu 2 wynikiem jest minimum z punktu oraz średniej arytmetycznej z pary sąsiadów na brzegach okna.
④ Można zażądać, aby przed filtrem widmo było wygładzone. Tu również można sterować szerokością okna.

```
const char* TSpectrum::Background (
  Double_t* spectrum,        : wskaźnik do tablicy z wartościami histogramu
  Int_t ssize,               : ilość słupków histogramu
  Int_t numberIterations,    : maksymalna szerokość okna sąsiadów  (szerokość = 2 ·nIterations)
  Int_t direction,           : Poszerzanie/zawężanie okna = kBackIn{De}creasingWindow
  Int_t filterOrder,         : Rząd filtra w każdym kroku: kBackOrder2, …4, …6, …8
  bool  smoothing,           : true, jeżeli wyznaczanie tła ma najpierw wygładzać widmo.
  Int_t smoothWindow,        : szerokość okienka wygładzania, kBackSmoothing3, …5, …, …15
  bool  compton              : true, gdy algorytm ma uwzględniać krawędzie Comptona
);
```

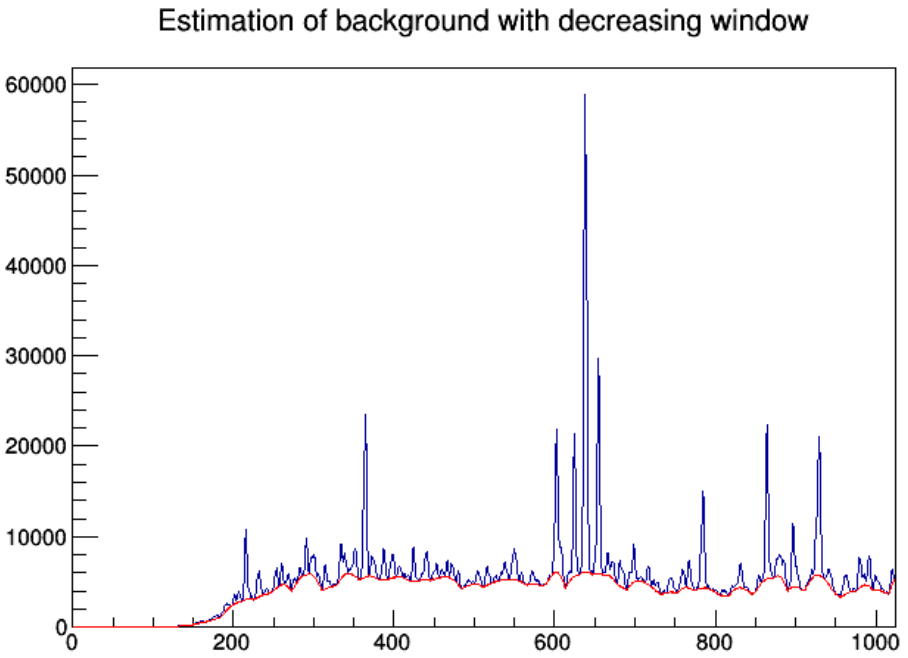
Przykłady:

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

Uwaga. Możliwa też inna funkcja, działająca na TH1:

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

Mody działania podawane są przez `option`.

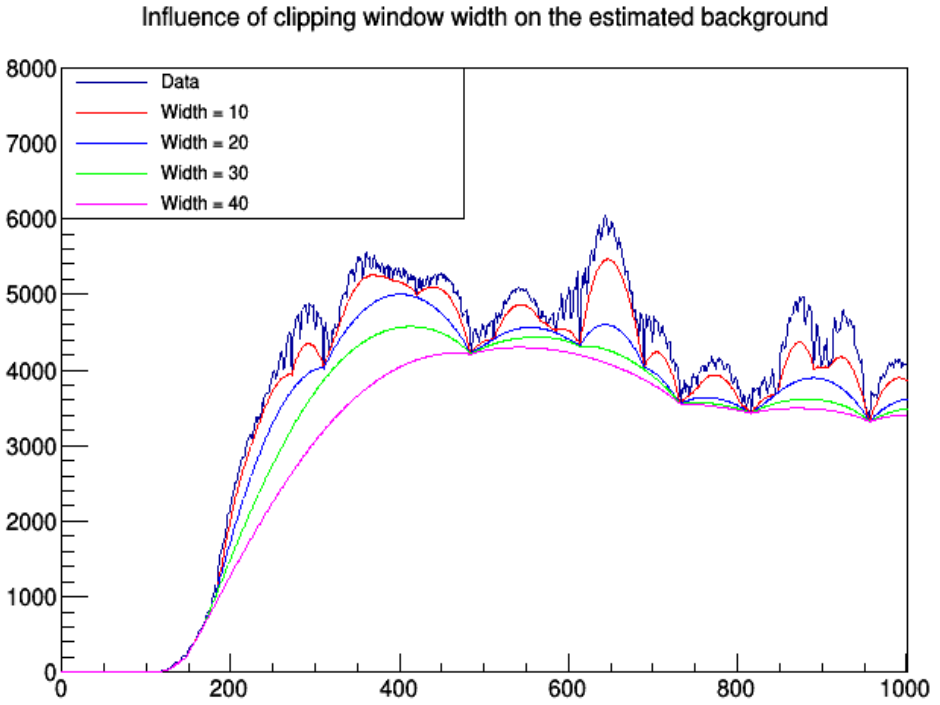


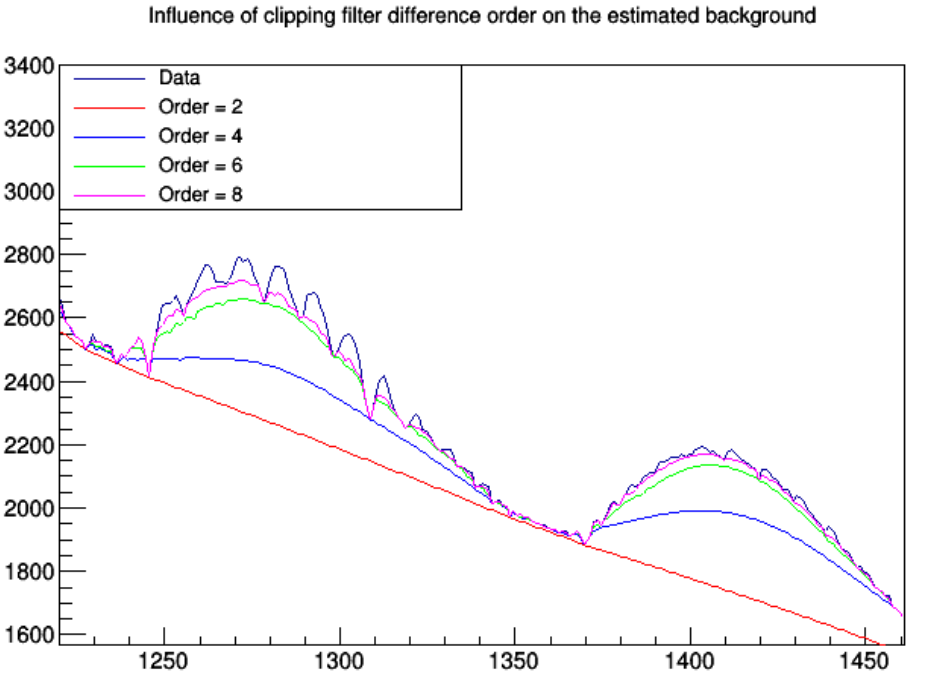
Wyznaczanie tła pod peakiem c.d.

Czasami przebieg tła jest niejednoznaczny, względnie zależny od interpretacji.

Przykładowe makro Background_width2.C pokazuje, jak dla widma o niejednoznacznej interpretacji tła działają różne szerokości okna filtrowania.

Natomiast makro Background_order.C pokazuje, jak dla podobnego widma tła działają filtry kolejnych rzędów.





Poszukiwanie maksimów (peak search)

```
Int_t TSpectrum::Search (
 Const TH1* hin,              :   wskaźnik do histogramu
 Double_t   sigma = 2,        :   założona szerokość poszukiwanych peaków
 Option_t*  option = ""       :   zob. poniżej
 Double_t threshold = 0.05 :   [0..1]. Pomija peaki, w których Amplituda ≤ threshold*NajwyższaAmp
);
```

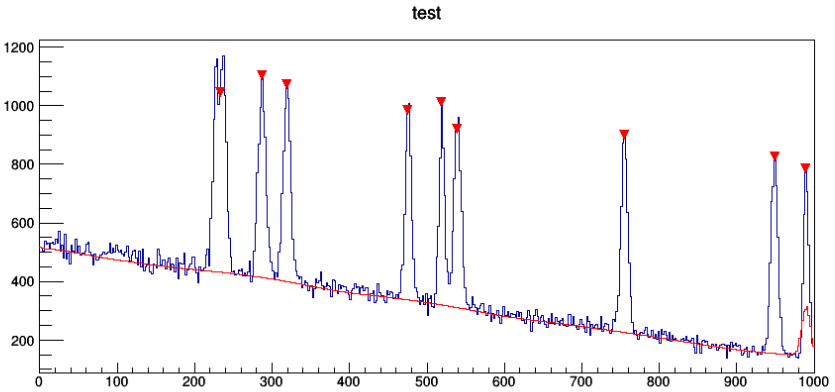
Liczbę znalezionych peaków i zestaw ich pozycji [X,Y] otrzymujemy przez :

`int TSpectrum::GetNPeaks()` oraz `Double_t* GetPositionX/Y ()`

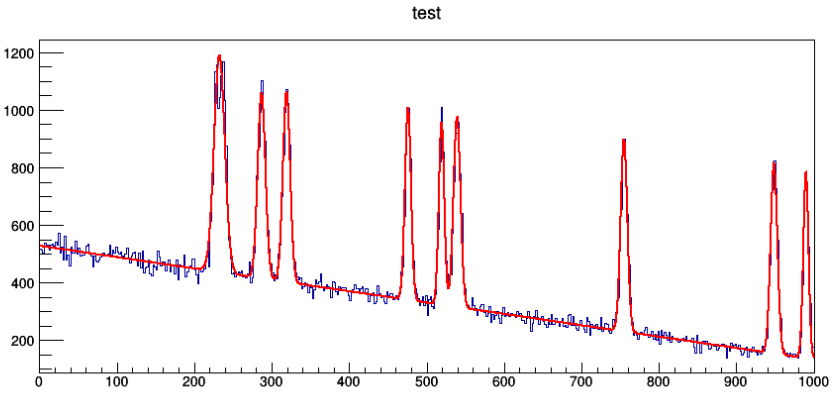
W środku `Search` kopiuje dane do tablicy. Następnie:

① transformacja osi Y wzmacniająca niskie krotności
② (w tablicy) wygładza histogram
③ dodaje zestaw markerów z pozycjami peaków
④ i rysuje histogram ze znalezionymi peakami.

Aby wstrzymać te działania, w `options` piszemy odpowiednio:
`"nobackground"`, `"nomarkov"`, `"goff"`, `"nodraw"`.



Przykład: peaks.C



Uwaga. Możliwa nieco bardziej zaawansowana funkcja:

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

gdzie istotną nowością jest możliwość dekonwolucji (odwrócenia splotu z funkcją odpowiedzi detektora).

Dekonwolucja

Gdy rozkład fizyczny *R*(x) jest mierzony doświadczalnie, to detektor zwykle rozmywa rozkład swoją funkcją odpowiedzi *D*(x). Otrzymane widmo *W*(x) jest splotem:

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

Jeżeli znamy funkcję odpowiedzi detektora D(x), to dekonwolucja pozwala na cofnięcie rozmycia i poznanie rozkładu oryginalnego W(x).

```
Int_t TSpectrum::Deconvolution (
 Double_t* source,                  :  wskaźnik na tablicę z wartościami histogramu W(x)
 const Double_t* response,          :  wskaźnik na tablicę z wartościami funkcji odpowiedzi D(x)
 Int_t ssize,                       :  ilość słupków histogramu W(x) [ tak samo D(x) ]
 Int_t numberIterations,            :  zob. Dokumentację lub [ref]
 Int_t numberRepetitions,           :  w przypadku opcji przyspieszania, ile powtórzeń przyspieszenia
 Double_t boost                     :  w przypadku opcji przyspieszania: wykładnik eksponensu
);
```

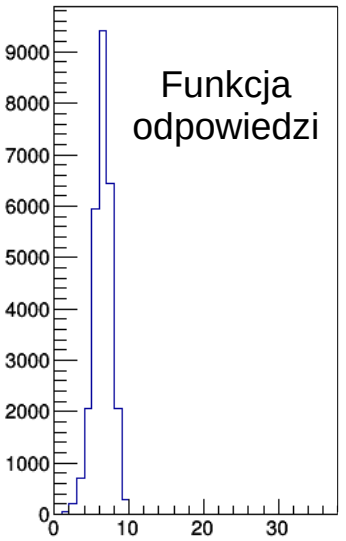
Wynikowy rozkład fizyczny R(x) zostaje umieszczony we wskaźniku `source`.

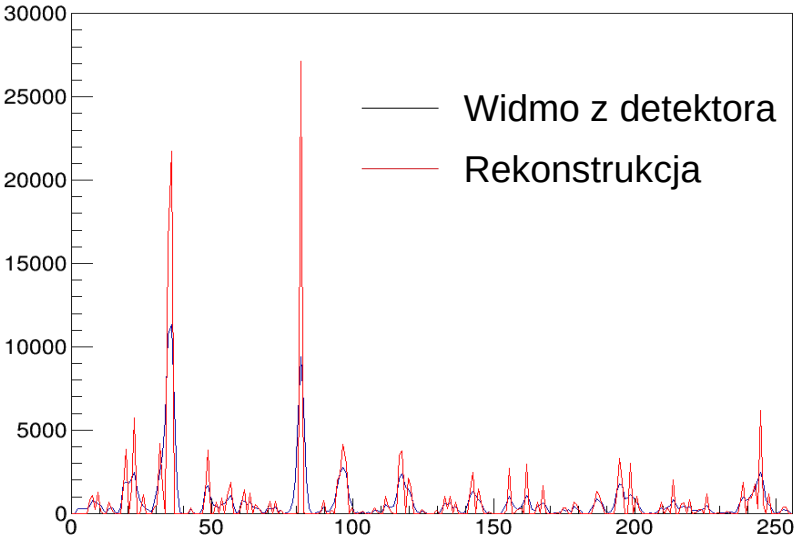
Przykłady:

1. Deconvolution.C
2. Deconvolution_wide.C
3. Deconvolution_wide_boost.C
4. Deconvolution**RL**_wide.C
5. Deconvolution**RL**_wide_boost.C

Uwaga. Możliwy alternatywny algorytm, Metoda Richardson–Lucy:

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





Grupowe dopasowanie peaków

Założenia zaimplementowanej metody **AWMI** (Algorithm Without Matrix Inversion) [ref]:

① Dopasowywana jest suma peaków. Każdy peak to funkcja Gaussa + możliwy "ogon":

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

② Wszystkie peaki mają taką samą dyspersję (σ)
③ Parametry dopasowywanej funkcji są nieskorelowane.

Proponowany algorytm (⟶ por. przykład **FitAwmi.C**)

⓪ Wstępne wyszukanie peaków, np. metodą **TSpectrum::SearchHighRes**
① Tworzymy obiekt analizujący klasy **TSpectrumFit**
② Ustawiamy opcje analizy poprzez **TSpectrumFit::SetFitParameters**
③ Ustawiamy wstępne wartości parametrów funkcji poprzez **TSpectrumFit::SetPeakParameters**

④ Wykonujemy dopasowanie poprzez **TSpectrumFit::FitAwmi** .
⑤ Dostęp do wyników (parametry i niepewności) poprzez gettery obiektu TSpectrumFit : `GetPositions`, `GetAmplitudes`, `GetAreas`, `GetSigma`, `GetTailParameters` oraz `Get...Errors` .

Uwaga na konwencje:

① wyniki podawane są przy założeniu, że szerokość słupka na osi X wynosi 1.
Należy przemnożyć parametry (i niepewności) przez szerokość, tj. TH1F::GetBinWidth (1) .

② Wartości podawane przez `GetSigma` należy podzielić przez √2, aby miały sens dyspersji i jej niepewności.

Przykład: FitAwmi.C

① Generacja rozkładu szeregu peaków o losowanych: σ (wspólnie) i amplitudzie (każdy osobno).

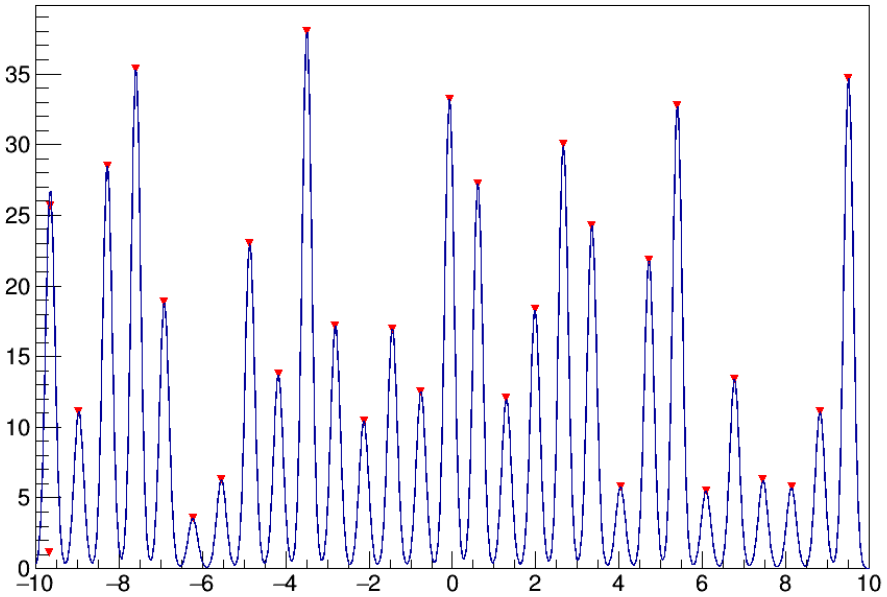
```
Created peak at -9.65769 with [Amp, Area]:  [26.7323 , 7.6458]
Created peak at -8.97307 with [Amp, Area]:  [11.0707 , 3.16637]
...
Total number of created peaks = 29 with sigma = 0.114103
```

② Wstępne wyszukanie liczby peaków i ich środków przez `TSpectrum::SearchHighRes`

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

③ Ustawienie metody i wartości wstępnych parametrów: dyspersji σ oraz szeregu pozycji i amplitud
④ Wykonanie dopasowania poprzez `TspectrumFit::FitAwmi`
⑤ Wydobycie znalezionych parametrów

```
* 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)
...
```



Transformaty

Transformaty sygnału wykonuje klasa **`TSpectrumTransform`** .
Możliwe rodzaje transformat (rozkładów na składowe):
Fouriera, cosinusowa, sinusowa, Haara, Walsha, Hartleya oraz łączone.

Ogólne **działanie**:

① Tworzymy obiekt klasy **`TSpectrumTransform`**, podając rozmiar sygnału.
② Ustawiamy pożądany rodzaj transformaty przez metodę **`SetTransformType`**.
oraz kierunek (→/←) przez **`SetDirection`**.
③ Wykonujemy transformatę: metoda **`Transform (Double_t* source, Double_t* dest)`**

Uwaga dla transformaty Fouriera (i pochodnych):

Przy transformacie “wprost” tablica wynikowa (dest) powinna mieć 2x większy wymiar, niż wejściowa
Przy transformacie odwrotnej tablica źródłowa (source) powinna mieć 2x większy wymiar, niż dest.

Przykład: FourierTransform.C

Tworzony sygnał jest typu:

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

Sygnał wejściowy → Wynik transformaty Fouriera