As a practical matter

Setting the stage

The function of interest is for\funwe.for, for\RANDOM.FOR

$$f\left(r\right)=Ae^{-\left(\frac{r-r_0}{w}\right)^2} \quad (1.1)$$

It has a Gaussian error with standard deviation β.

1. A look at the function

The relevant equation is 3.9 in Laurent.htm $I=\pi\int_0^1\left(1+r^2\left(t\right)\right)f\left(r\left(t\right)\right)dt$ (1.2)

Define

$$\hat{f}\left(t\right)=\pi\left(1+r^2\left(t\right)\right)f\left(r\left(t\right)\right) \quad (1.3)$$

Where

$$r\left(t\right)=\tan\left(\pi\left(t-1/2\right)\right) \quad (1.4)$$

In practice, we know nothing about the function and at this stage merely want to look at it. With the definitions in (1.3) and (1.4) the entire range of the function is from 0 to 1. Thus the plotting limits are obvious. Thus define (1.3) as for\fhat.for. Then slightly modify the BLI found in ..\interpolation\Bli.htm to start with NP/2 points for\bli2.for and make a main code for\tbli2.for as in **Bli.htm,** but with fhat instead of ftest. For the convenience of Watcom a project file use for\apm.wpj

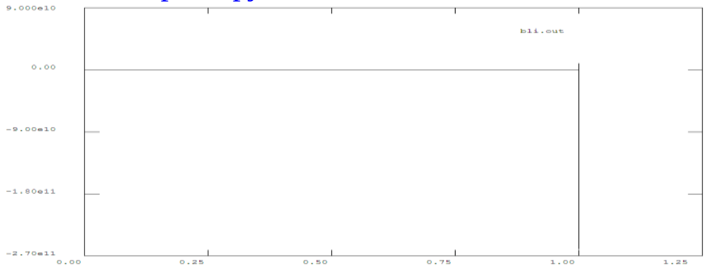
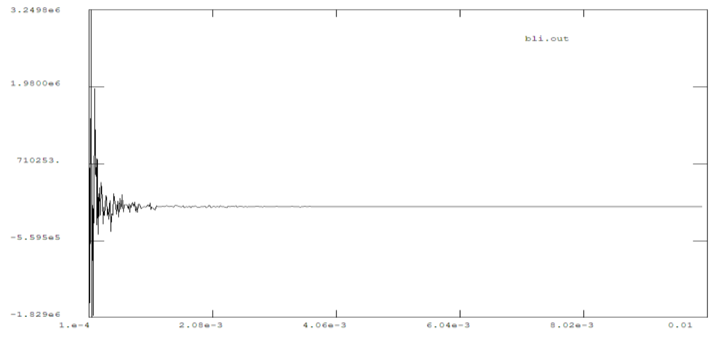


Figure 1 The first output BLI.out from the code.

Note that the error makes for potentially very large values near the origin.
Since nothing shows on the plot try the ends. .0001 (to stay just a bit away from the end) to 0.01



Then 0.99 (equals 1 to graphical accuracy) to 0.9999 to stay just away from the end.

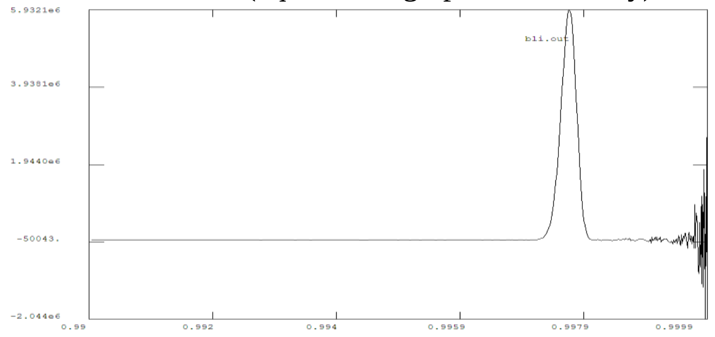
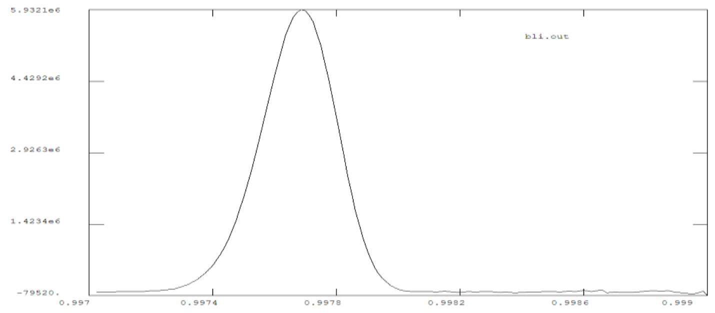


Figure 2 The upper end of BLI.OUT



The function is between 0.997 and 0.999

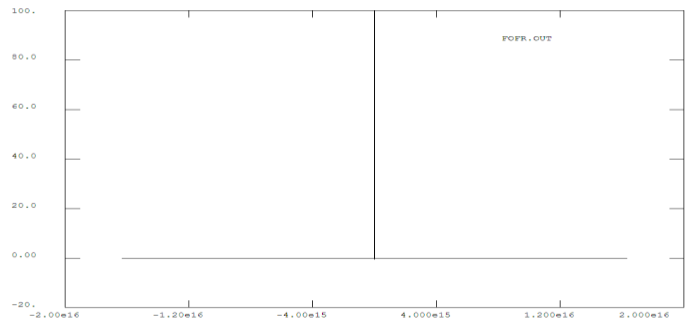


Figure 3 The function as located in t space.

Figure 4 The unknown function as a function of r.

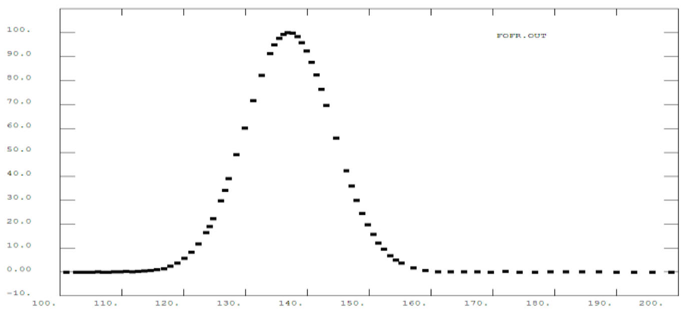


Figure 5 The function of r afer repeated expansions.

Be reasonable

The function that we have found has errors that can be easily seen by expanding the regions above and below the peak to y = +- 0.2. The function of interest to a physicist starts at R = 100 and ends by R = 200
The code for\TBLI3.FOR differs from **tbli2.for** only in the utilization of these beginning and ending points and the use of for\funwe.for directly.

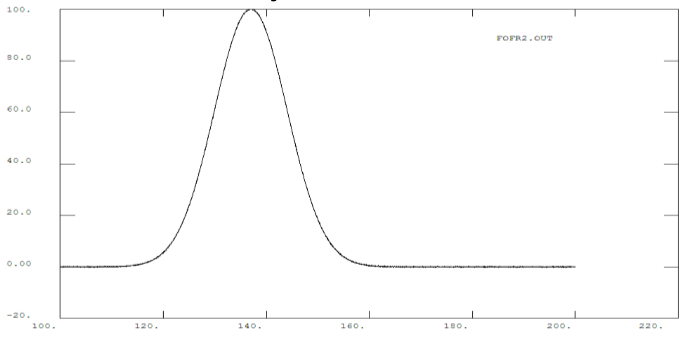
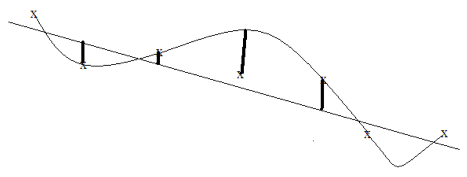


Figure 6 The function of R alone.

There is something missing in the file for\FOFR2.OUT. An error estimate for the points is needed. This is where the code for/TLag2.for, with include for\lag2.for, comes in for\errest.wpj. The code reads the file fofr2.out. Each point is interpolated using a polynomial through all but the point. – very similar to the BLI method itself except that the polynomial is large enough that the errors are not due to derivatives but to fluctuations in the data.



As the picture shows this is larger than the actual miss, but related. The actual error is 0.1 from for\funwe.for
The average error estimated in this way is 0.15. This could be corrected in an ad-hoc fashion.

2. The derivatives

The function has now been found at a large number of points. We need to find its derivative at these same points and the error in this value. A method for using the function values to find the derivatives is described in ..\interpolation\Derivatives.htm and given here as for\fders3.for. The derivative is "essentially"

$$f_A'(x_i) = \frac{f(x_{i+1}) + \varepsilon_{i+1} - f(x_{i-1}) + \varepsilon_{i-1}}{h} + \alpha h^2 \frac{f(x_i) + f(x_{i-1}) - 2f(x_i) + 4\varepsilon}{h^2} + \ldots \quad (2.1)$$

This becomes

$$f_A'(x_i) = f'(x_i) + \frac{2\varepsilon_i}{h} + 4\varepsilon_i + \alpha h^4 \quad (2.2)$$

The epsilon terms are random. They do not extrapolate well. If I use fders3 with half the points, the term in epsilon/h becomes smaller while the term in alpha h^4 becomes smaller. If the dominant error is random, the error with half the points will be smaller, if the dominant error is systematic, both sets of points will have the same very small errors. In either case, a good error estimate at every other point is the difference in the values.

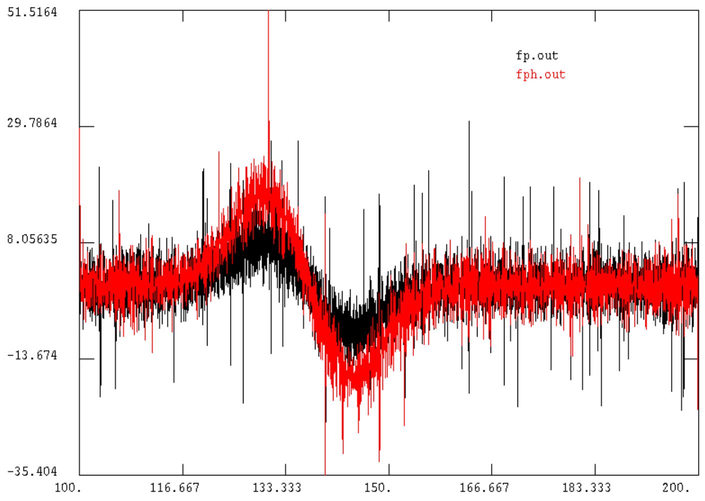


Figure 7 First derivatives returned by fders3

Note that since the "error" is significantly less with half points, that the epsilon dominates the calculation.
AVE ERROR = 6.698145003576840

Alternate set of derivatives

Start with the Lagrange Polynomial

$$P_N(x) = \sum_{m=1}^{N} f(x_m) L_m(x) \quad (2.3)$$

$$L_m(x) = \prod_{j \neq m}^{N} \frac{x - x_j}{x_m - x_j} \quad (2.4)$$

The fixed values f(x_m) have no derivatives, but L_m(x) does so the first derivative of P is given by

$$\frac{dP_N(x)}{dx} = \sum_{m=1}^{N} f(x_m) \frac{dL_m(x)}{dx} \quad (2.5)$$

$$\frac{dL_m(x)}{dx} = \sum_{\ell \neq i}^{N} \frac{1}{x_m - x_\ell} \prod_{\substack{j \neq \ell \\ j \neq m}}^{N} \frac{x - x_j}{x_m - x_j} \quad (2.6)$$

The code for\TLagder.for with include for\lagder.for treats this case.

ERRSAVE= 13.851268901421800

The doubling of the error estimate is real. The Lagrangian is inclined to jump around much more.

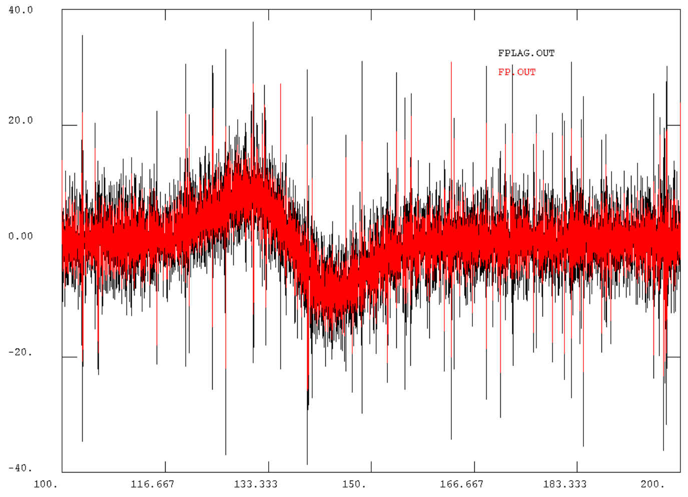


Figure 8 Lagrange derivatives (black) FDERS (red)

3. Integrals

Trap rule can be carried out in a manner exactly analogous to the FDER3 method above. A somewhat more interesting method is to use Lagrange interpolation between each set of points and to integrate the polynomial exactly using GaussQuadrature as mentioned two lectures ago.

for\lagiint.wpj for\tlagint.for for\lag4.for

ERRSAVE= 0.004859969557288