1 Continuous piecewise linear least squares fit

1.1 Theory

We consider the problem of fitting N straight lines between points with x-coordinates $\{X_1, X_2, ...X_{N+1}\}$, which do not have to be evenly spaced. In other words, $\Delta X_J := X_{J+1} - X_J$, $J = 1 \ldots N$ is not necessarily constant. Within each segment we have n_J (again, not necessarily constant) data points $\{(x_J^i, y_J^i),\ i = 1 \ldots n_J\}$, which we imagine to be scattered about the desired piecewise linear fit. Fig 1.1 shows the idea, for $N = 4$.

The standard way of fitting the noisy data by straight line segments would be to calculate the slope m_J and intercept c_J of each segment independently, using the usual formulas

$$\begin{aligned} m_J &= (\overline{xy}_J - \bar{x}_J \bar{y}_J)/(\overline{x^2}_J - (\bar{x}_J)^2) && (1.1)\\ c_J &= \bar{y}_J - m_J \bar{x}_J = (\overline{x^2}_J \bar{y}_J - \bar{x}_J \overline{xy}_J)/(\overline{x^2}_J - (\bar{x}_J)^2), \quad \text{where} && (1.2)\\ \overline{f}_J &:= (n_J)^{-1} \sum_{i=1}^{n_J} f_J^i \quad \text{for general } f. && (1.3) \end{aligned}$$

This procedure minimises the sum of the squares in each segment, but the resulting piecewise linear fit is, naturally, discontinuous at the boundaries between the segments, as exemplified in Fig 1.1.

We could enforce continuity by insisting that the $(J-1)^{\text{th}}$ and J^{th} segments join at (X_J, Y_J), where the $\{Y_J\}$ are to be determined. Since the $\{X_J\}$ are known, the $\{Y_J\}$ would then determine the entire fit.

The $\{Y_J\}$ can be found by minimising the overall sum of squares S, which is given by

$$\begin{aligned} S &= \sum_{J=1}^{N} \frac{1}{2} \sum_{i=1}^{n_J} (y_J^i - m_J x_J^i - c_J)^2 && (1.4)\\ &= \sum_{J=1}^{N} \frac{1}{2} \sum_{i=1}^{n_J} \left[y_J^i - x_J^i \left(\frac{Y_{J+1} - Y_J}{X_{J+1} - X_J} \right) - \left(\frac{X_{J+1} Y_J - X_J Y_{J+1}}{X_{J+1} - X_J} \right) \right]^2. && (1.5) \end{aligned}$$

Setting $\partial S / \partial Y_J = 0$, for $J = 1 \ldots N+1$ gives

$$\begin{aligned} &\sum_{i=1}^{n_{J-1}} \left[y_{J-1}^i + Y_{J-1} \left(\frac{x_{J-1}^i - X_J}{\Delta X_{J-1}} \right) - Y_J \left(\frac{x_{J-1}^i - X_{J-1}}{\Delta X_{J-1}} \right) \right] \left[- \left(\frac{x_{J-1}^i - X_{J-1}}{\Delta X_{J-1}} \right) \right] + \\ &\sum_{i=1}^{n_J} \left[y_J^i + Y_J \left(\frac{x_J^i - X_{J+1}}{\Delta X_J} \right) - Y_{J+1} \left(\frac{x_J^i - X_J}{\Delta X_J} \right) \right] \left[+ \left(\frac{x_J^i - X_{J+1}}{\Delta X_J} \right) \right] = 0. \qquad (1.6) \end{aligned}$$

Eqn (1.6) holds for $J = 2 \ldots N$. If $J = 1$ the first sum is omitted; for $J = N+1$ the second one is.

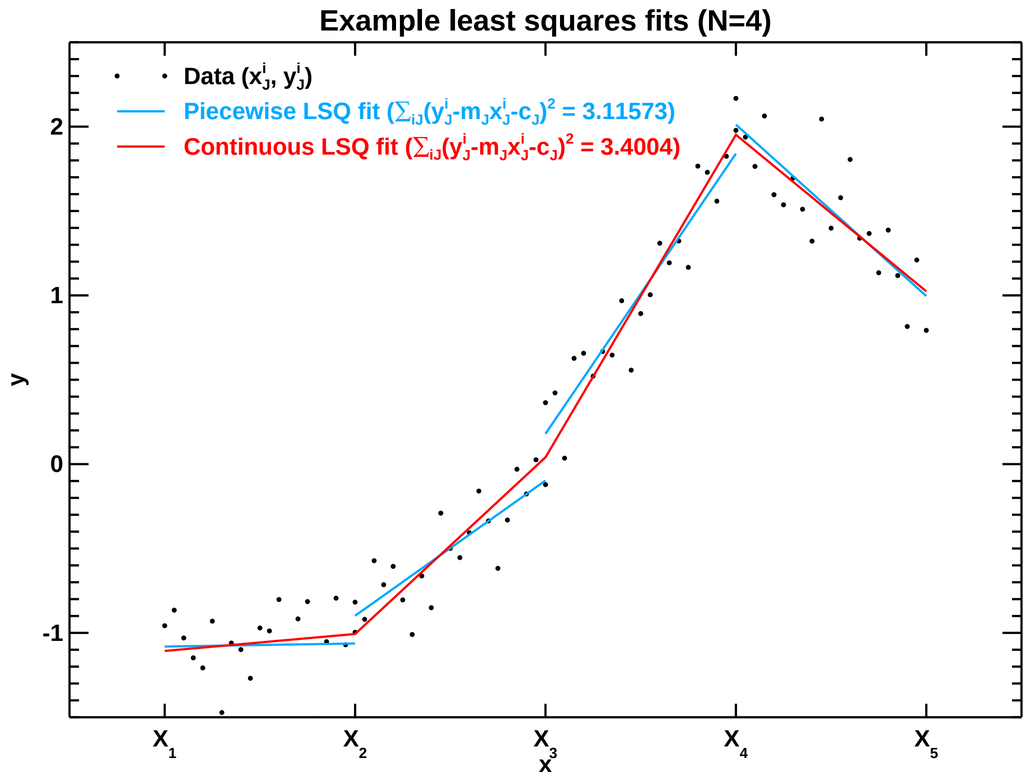

Figure 1.1: Example piecewise linear fits. Blue: least squares fit in each region calculated independently; red: least squares fit with enforced continuity between regions.

Rearranging Eqn (1.6) and collecting terms, we find

$$\begin{aligned}
Y_{J-1}\left\{-\frac{n_{J-1}}{(\Delta x_{J-1})^2}\overline{(x_{J-1}-X_{J-1})(x_{J-1}-X_J)}\right\} &+\\
Y_J\left\{\frac{n_{J-1}}{(\Delta x_{J-1})^2}\overline{(x_{J-1}-X_{J-1})^2}+\frac{n_J}{(\Delta x_J)^2}\overline{(x_J-X_{J+1})^2}\right\} &+\\
Y_{J+1}\left\{-\frac{n_J}{(\Delta x_J)^2}\overline{(x_J-X_J)(x_J-X_{J+1})}\right\} &=\\
\left\{\frac{n_{J-1}}{\Delta x_{J-1}}\left[\overline{xy}_{J-1}-X_{J-1}\overline{y}_{J-1}\right]-\frac{n_J}{\Delta x_J}\left[\overline{xy}_J-X_{J+1}\overline{y}_J\right]\right\} & \qquad (1.7)
\end{aligned}$$

where the averages ($\overline{x}_J$ etc) are defined by Eqn (1.3). Eqn (1.7) holds for $J=2\ldots N$. For $J=1$ the terms in n_{J-1} and Y_{J-1} are omitted; for $J=N+1$ the terms in n_J and Y_{J+1} are.

All the terms in curly brackets in Eqn (1.7) are calculable from the data $\{X_J,\{(x^i_J,y^i_J),i=1\ldots n_J\},J=1\ldots N\}$ and X_{N+1}. Eqn (1.7) is therefore a tridiagonal system of equations for the unknowns $\{Y_J\}$, for which efficient and robust solvers are readily available (e.g. IDL's TRISOL or the NAG library's F04BCF). Once the $\{Y_J\}$ are known, the straight line fit to the data in the J^{th} region $(J=1\ldots N)$ is given by

$$\begin{aligned}
y &= m_Jx+c_J, \quad \text{where} && (1.8)\\
m_J &= (Y_{J+1}-Y_J)/(X_{J+1}-X_J) \quad \text{and} && (1.9)\\
c_J &= (X_{J+1}Y_J-X_JY_{J+1})/(X_{J+1}-X_J). && (1.10)
\end{aligned}$$

Fig 1.1 shows the result of applying this method to the data whose piece-by-piece least squares linear fit was calculated earlier. As can be seen, the fit is now continuous, although the price of this continuity is a 10% increase in the overall sum of squares.

1.2 Code

There follows a Fortran-90 subroutine to implement the algorithm of Eqn (1.7) in the special but common case where n_J and Δx_J are constant. This allows the input data $\{\{(x^i_J, y^i_J), i = 1 \dots n_J\}, J = 1 \dots N\}$ to be held as 2D arrays of size (n, N), where n is the common value of n_J. The general case could be handled by holding these data as 1D arrays of length $\sum_{J=1}^{N} n_J$, and using F90 WHERE statements to select the data points within each interval $X_J \leq x^i_J < X_{J+1}$. Alternatively, the data could be held in masked 2D arrays of size $(\max_J n_J, N)$. Both solutions would incur costs in execution time and in recoding.

```
! ------------------------------------------------------------------------------
  SUBROUTINE cont_lsq_fit (x, y, bigx, bigy, yout)

! Make piecewise linear LSQ fit to {{(x^i_J, y^i_J), i=1, n}, J=1, bign)
! by constructing the ordinates Y_J at the given abscissae X_J.
! x^i_J and y^i_J are held in 2D arrays x(i, J) and y(i, J).
! Optionally, return the fitted values in the 2D array yout.
! wp is assumed to be the desired real working precision kind value.
! Reference:
! http://www.golovchenko.org/docs/ContinuousPiecewiseLinearFit.pdf

! I/O
    REAL(wp), INTENT(IN)                :: x(:, :), y(:, :), bigx(:)
    REAL(wp), INTENT(INOUT)             :: bigy(:)
    REAL(wp), OPTIONAL, INTENT(INOUT)   :: yout(:, :)

! Local
    INTEGER                             :: n, bign, J
    REAL(wp), ALLOCATABLE               :: xbar(:), ybar(:), xxbar(:), xybar(:)
    REAL(wp), ALLOCATABLE               :: dl(:), dd(:), du(:), rhs(:)
    REAL(wp)                            :: grad, intercept

! Sizes
    n    = SIZE(x, 1)
    bign = SIZE(x, 2)

! Means
    ALLOCATE ( xbar(1:bign), xxbar(1:bign), ybar(1:bign), xybar(1:bign) )
    xbar  = SUM(x,   DIM=1) / n
    xxbar = SUM(x*x, DIM=1) / n
    ybar  = SUM(y,   DIM=1) / n
    xybar = SUM(x*y, DIM=1) / n

! Define tridiagonal matrix
    ALLOCATE ( dl(1:bign+1), dd(1:bign+1), du(1:bign+1), rhs(1:bign+1) )

!! Top row of tridiag
    J = 1
    dl(J)  = 0.0_wp
    dd(J)  =   xxbar(J) - xbar(J)*2.0_wp*bigx(J+1) + bigx(J+1)**2
    du(J)  = -(xxbar(J) - xbar(J)*(bigx(J)+bigx(J+1)) + bigx(J)*bigx(J+1))
    rhs(J) = -(bigx(J+1)-bigx(J)) * (xybar(J) - ybar(J)*bigx(J+1))

!! Bottom row of tridiag
    J = bign + 1
    dl(J)  = -(xxbar(J-1) - xbar(J-1)*(bigx(J)+bigx(J-1)) + bigx(J)*bigx(J-1))
    dd(J)  =   xxbar(J-1) - xbar(J-1)*2.0_wp*bigx(J-1) + bigx(J-1)**2
    du(J)  = 0.0_wp
    rhs(J) =  (bigx(J)-bigx(J-1)) * (xybar(J-1) - ybar(J-1)*bigx(J-1))

!! Other rows of tridiag
    DO J=2,bign
      dl(J)  = -(xxbar(J-1) - xbar(J-1)*(bigx(J)+bigx(J-1)) + bigx(J)*bigx(J-1))
      dd(J)  =   xxbar(J-1) - xbar(J-1)*2.0_wp*bigx(J-1) + bigx(J-1)**2 + &
                 xxbar(J)   - xbar(J)*2.0_wp*bigx(J+1) + bigx(J+1)**2
      du(J)  = -(xxbar(J)   - xbar(J)*(bigx(J)+bigx(J+1)) + bigx(J)*bigx(J+1))
      rhs(J) = (bigx(J)-bigx(J-1)) * (xybar(J-1) - ybar(J-1)*bigx(J-1)) - &
               (bigx(J+1)-bigx(J)) * (xybar(J) - ybar(J)*bigx(J+1))
    END DO

! Tridiagonal inversion
    CALL TRISOL(dl, dd, du, rhs, bigy)

! Generate fit at original x values
    IF ( PRESENT(yout) ) THEN
      DO J=1,bign
        grad = (bigy(J+1) - bigy(J)) / (bigx(J+1) - bigx(J))
        intercept = bigy(J+1) - grad * bigx(J+1)
        yout(:, J) = grad * x(:, J) + intercept
      END DO
    END IF
```

```
! Clean up
    DEALLOCATE ( rhs, du, dd, dl )
    DEALLOCATE ( xybar, ybar, xxbar, xbar )

  END SUBROUTINE cont_lsq_fit
! -----------------------------------------------------------------------------
```

Tridiagonal solvers abound, but, for completeness, here is a simple one:

```
! -----------------------------------------------------------------------------
  SUBROUTINE TRISOL(a, b, c, d, x)

! Solves
!      _                         _   _  _     _  _
!     | b1 c1                     | |x1  | = |d1  |
!     | a2 b2 c2                  | |x2  | = |d2  |
!     |     . . .                 | |.   | = |.   |
!     |        ai bi ci           | |xi  | = |di  |
!     |           . . .           | |.   | = |.   |
!     |             an-1 bn-1 cn-1| |xn-1| = |dn-1|
!     |                      an bn| |xn  | = |dn  |
!      -                         -   -  -     -  -
! by Gaussian elimination without pivoting.
!
! Refererence:
! https://en.wikipedia.org/w/index.php?title=Tridiagonal_matrix_algorithm.
!
! a1 and cn must be supplied (i.e. all vectors must have dimension (1:n)),
! even though they are not used.

! I/O
    REAL(wp), INTENT(IN)     :: a(:), b(:), c(:), d(:)
    REAL(wp), INTENT(INOUT)  :: x(:)

! Local
    INTEGER                  :: i, n
    REAL(wp), ALLOCATABLE    :: c1(:), d1(:)

! Sizes
    n = SIZE(a)

    ALLOCATE ( c1(n), d1(n) )
    c1(1) = c(1) / b(1)
    DO i=2,n-1
      c1(i) = c(i) / (b(i) - a(i)*c1(i-1))
    END DO

    d1(1) = d(1) / b(1)
    DO i=2,n
      d1(i) = (d(i) - a(i)*d1(i-1)) / (b(i) - a(i)*c1(i-1))
    END DO

    x(n) = d1(n)
    DO i=n-1,1,-1
      x(i) = d1(i) - c1(i)*x(i+1)
    END DO

    DEALLOCATE ( d1, c1 )

  END SUBROUTINE TRISOL
! ------------------------------------------------------------------------------
```

1.3 Results

This work was prompted by the inclusion of a wave optics propagator in version 9.0 of the ROM SAF Radio Occultation Processing Package, ROPP (http://www.romsaf.org/ropp/index.php). The new tool simulates the propagation of radio waves between transmitting and receiving satellites as they pass through the earth's refractivity field, which is modelled as a series of vertical refractivity 'screens'. The radio signal at each point of the receiving satellite's orbit is calculated by a Fresnel integral of the phase and amplitude of the signal on the final screen (http://www.romsaf.org/romsaf_ropp_ug_pp.pdf). Currently these final screen phases and amplitudes are represented by a set of piecewise linear fits to inherently high resolution ($\sim 10^6$ points) source data, like the blue curves in Fig 1.1. (Linear fits allow the Fresnel integrals to be evaluated analytically.) It is natural to ask whether a continuous piecewise linear fit, like red curve in Fig 1.1, might give a better solution — for example, a less noisy one.

In fact, the use of a continuous least squares fit generally makes little difference to the bending angles and refractivities derived from the excess phase data by means of the ROPP pre-processing tool `ropp_pp_occ_tool`, even if we increase the number of points in each region (i.e. n_J) from its default of 100 to 200. For example, Fig 1.2 compares the bending angles and refractvities produced by the wave optics propagator when it runs through a refractivity profile with some large vertical refractive index gradients. The bending angles produced by the continuous and discontinuous least squares fits are both noisy below 25 km (where `ropp_pp_occ_tool` uses wave optics processing to generate bending angles), but are very similar on average throughout the profile. In addition, the refractivies derived from these bending angles are compared to each other and to the input refractivity profile. Again, the two models produce very similar results. If anything, the discontinuous fit looks slightly closer to the input refractivity below 25 km.

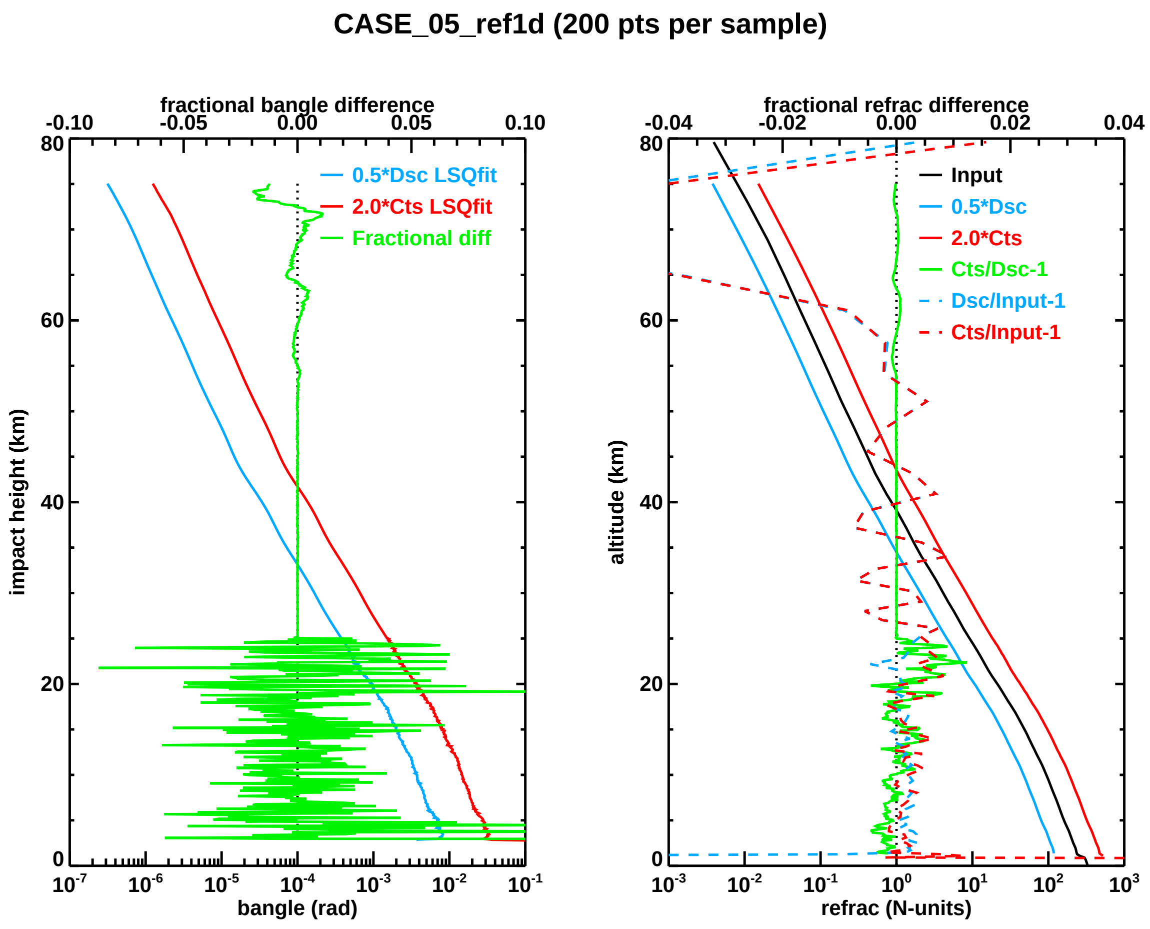

Figure 1.2: Comparison of bending angles and refractivities derived from excess phase data produced by the wave optics propagator when the signal on the final screen modelled by continuous and discontinuous piecewise linear fits. This uses 200 points per sample, which is twice the default. Left: bending angles and their fractional differences. Right: refractivities and their fractional differences with respect to each other and to the input refractivity.

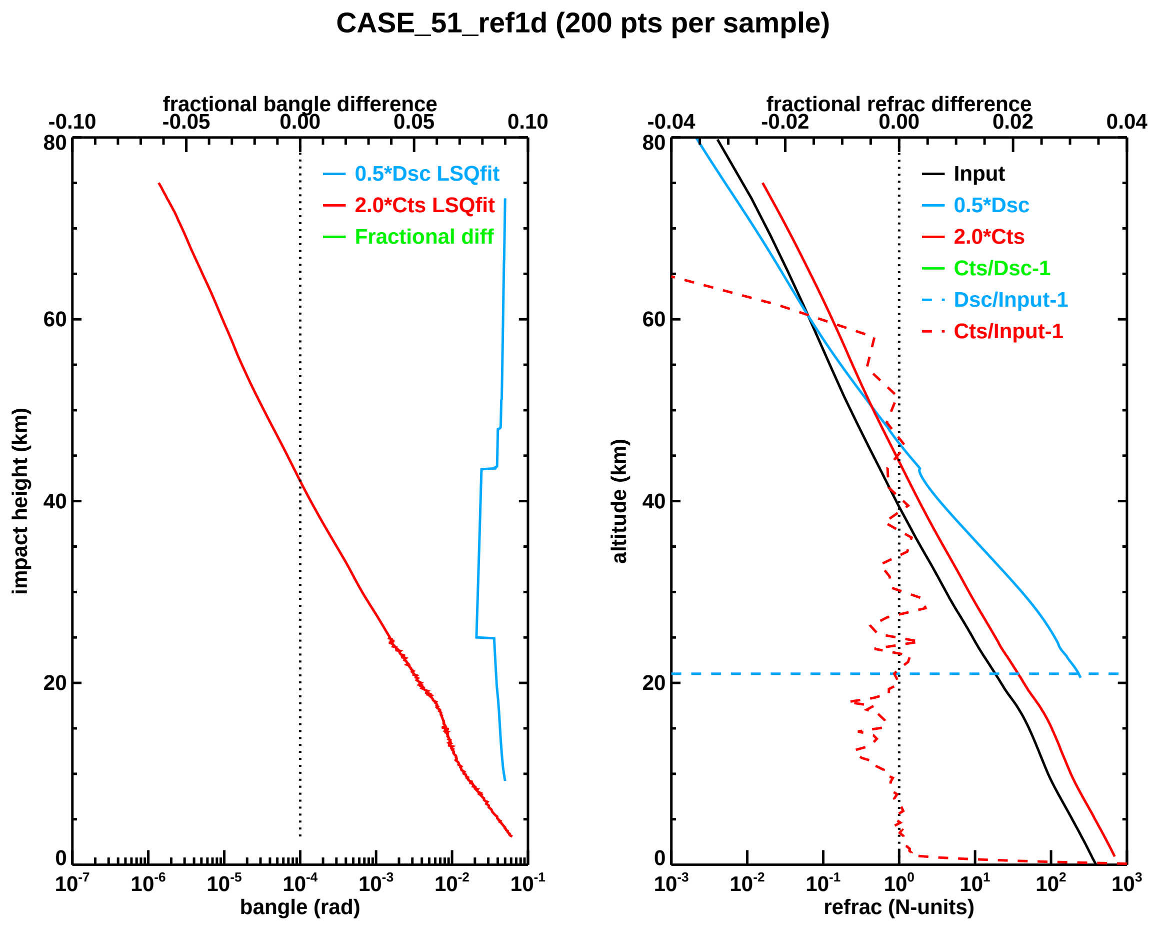

Figure 1.3: As Fig 1.2, but for a different input refractivity profile (one with lower vertical gradients).

There is one case in which the continuous least squares fit is better, and this is shown in Fig 1.3. In this example, which is derived from a refractivity profile having modest vertical gradients, `ropp_pp_occ_tool` fails to produce any sort of meaningful bending angle when presented with excess phases generated from discontinuous linear fits to the signal on the final screen. Excess phases produced by continuous fits, however, generate reasonable bending angles and thence refractivities. The continuous fit is therefore more robust in this case. But it should be remembered that both runs use values of n_J (namely, 200) that are twice as large as usual. In the default case there is very little difference between the two. So a better solution in this case would be to use $n_J = 100$. (The implication for the wave optics propagator is therefore that n_J should never be larger than 100.)

A possible reason for the clear difference between the two methods in this case is shown in Fig 1.4, which is a zoomed-in section of the amplitude of the signal near the top of the final screen (where the signal is 'apodized'), and the continuous and discontinuous linear fits to it. Large gaps in the amplitude are produced by the discontinuous method in regions where the gradient of the amplitude is changing rapidly. Naturally, these gaps disappear when continuous linear fits are used. If $n_J = 100$ the corresponding curves (not shown) are almost indistinguishable by eye, which is why there is little difference in the bending angles derived from them.

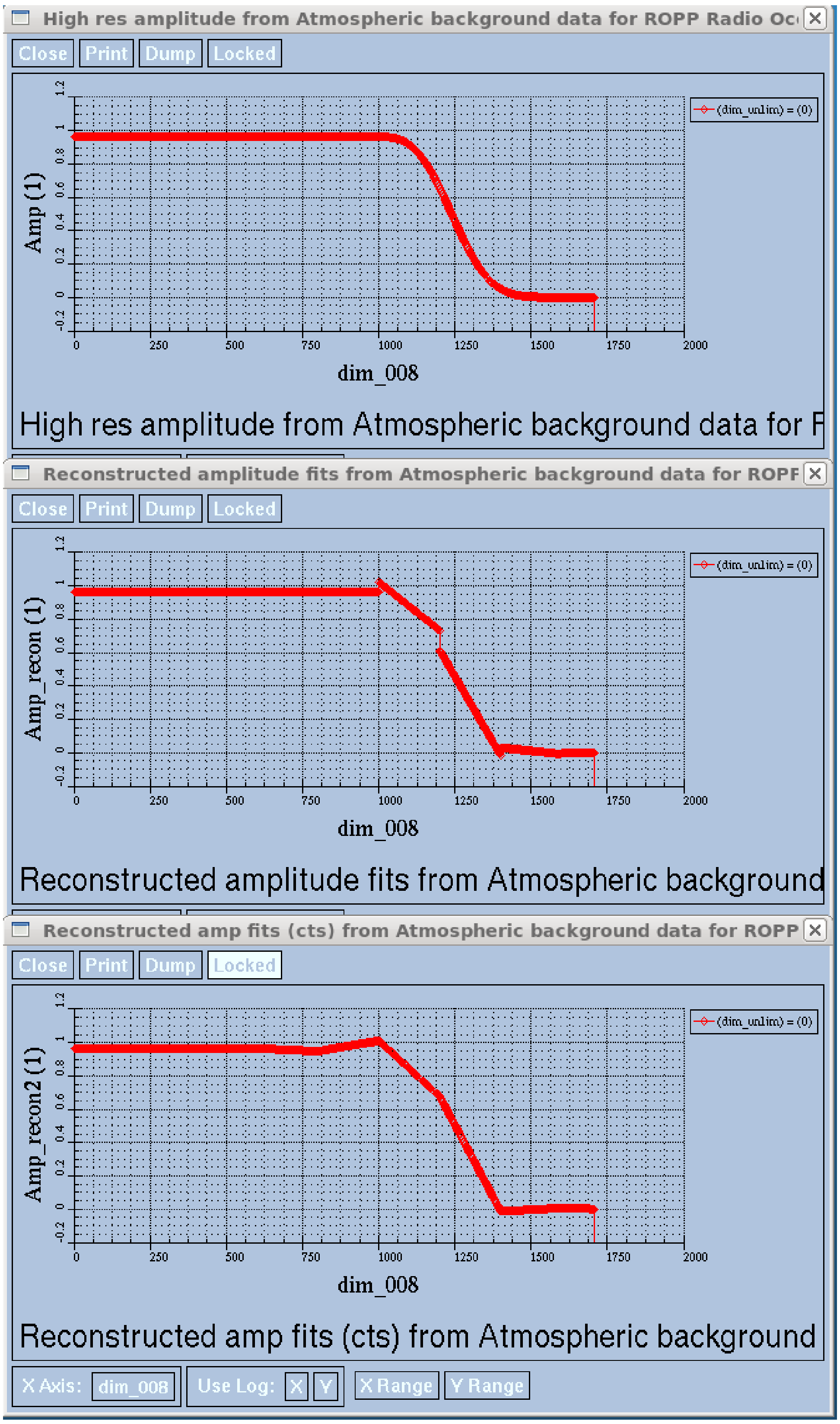

Figure 1.4: Zoomed-in signal amplitudes on the final refractivity screen, for the case used in Fig 1.3. Top: high resolution source data. Middle: Discontinuous linear fits (200 points per sample). Bottom: continuous linear fits (200 points per sample).

1.4 Conclusions

This note has described the means to generate a continuous piecewise linear fit to a set of data points $\{(x_i, y_i)\}$. The calculation involves the solution of a tridiagonal system of equations for the y-values at the boundaries of the fitting regions. Such systems can be solved simply and quickly, and Fortran-90 codes to do so have been included. The price of this continuity is an increase in the overall sum of squares above that produced by the usual piece-by-piece least squares fitting procedure.

In one example the new fitting method was not found to deliver significant advantages over the usual method. It is hoped, however, that the theory may prove beneficial in other situations.

1.5 Acknowledgements

Useful discussions on this subject with Sean Healy (ECMWF) and Chris Burrows (Met Office) are gratefully acknowledged.