NOAA Technical Memorandum NOS NGS 84

Biquadratic Interpolation

Dru Smith

Silver Spring, MD
September 2020

Versions

Date	Changes
September 2, 2020	Original Release

1 Introduction

The National Geodetic Survey (NGS) has coded software based tools for both internal and external use since the late 1960s (Coast and Geodetic Survey, 1967; Coast and Geodetic Survey 1970.) Throughout much of that history, the language of choice was FORTRAN, as this was the de facto standard used around the world for scientific computing. As such, scientists within NGS often had a strong combination of both geodetic skills and computer coding skills, and these same scientists were almost always the authors of the software which found its way into the public domain.

As part of that scientific software development, it was often important for NGS scientists to creatively design individual computer tasks, from simple subroutines up through programs of hundreds of thousands of lines of code. Within the team of geoid scientists, one small routine was developed in the 1990s which has served in dozens of programs for the last 20+ years, though it was never formally documented: **biquadratic interpolation**.

Because biquadratic interpolation is being implemented in numerous NGS programs as part of the modernized NSRS, and because it appears to be somewhat less well known than either bilinear interpolation or bi-cubic splines, it was felt that a brief explanation should be provided.

2 Biquadratic Interpolation

As its name implies, biquadratic interpolation is the use of quadratic functions in two dimensions, to interpolate a value at any location, from some given grid of values. In order to provide a unique solution therefore, biquadratic interpolation relies upon the nearest 3×3 set of grid points to the point of interpolation (POI). A simple comparison of other interpolation methods (from a grid), for interests sake, is given below. Because splines are a more complicated interpolation mechanism, and not relevant to this report, they will not be discussed.

Type of interpolation	**Type of functions used**	**Number of points used**
Bilinear	Linear	2×2
Biquadratic	Quadratic	3×3
Bi-cubic[1]	Cubic	4×4

Like all non-spline two-dimensional grid interpolators, the actual interpolation is done first by defining the window of grid points to be used in the interpolation. After that, the interpolation is done as follows: fit functions in one direction and then, as a final step, fit one final function in the cross-direction. That is, for biquadratic interpolation, one first fits quadratic functions row-wise through 3 grid points in 3 different rows, evaluating each function at the X value of the POI. Then a final quadratic function is fit through those three functional values, column-wise and this final quadratic function is evaluated at the Y value of the POI, yielding the interpolated value at the POI. Incidentally, one would arrive at the identical value if one first fit through 3 columns, and finished with a row-wise evaluation. Similar descriptions exist for bilinear (2 rows of linear

[1] Not the same as "bi-cubic splines"

functions followed by 1 column of linear function, or vice versa) or bi-cubic (4 rows of cubic functions followed by 1 column of cubic function, or vice versa).

2.1 Defining the window

The first step in biquadratic interpolation is defining the 3 × 3 window of grid points to be used. This will always be the nearest 3 × 3 points to the POI. In the space surrounded by any four grid points, four quadrants can be identified. The 3 × 3 window used depends on whether the POI falls in quadrant 1, 2, 3 or 4. See Figure 1 below.

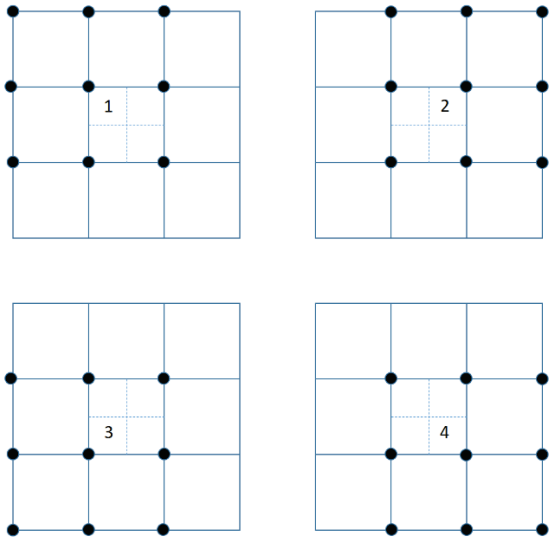

Figure 1: The four possible 3 × 3 windows for biquadratic interpolation, depending on the location of the point of interpolation.

Once the 3 × 3 window is identified, the interpolation begins. This discussion will presume that row-wise quadratic fits will be done first, followed by column-wise. The same arguments can be made (but are not included herein) if column-wise followed by row-wise is performed.

Before proceeding, let us first point out that the method of 2-D interpolation works whether the coordinates are Cartesian or geographic. That is, the following can be (and has been, successfully) implemented if *X* is replaced with longitude and *Y* is replaced with latitude. However the subroutine as encoded uses the Newton-Gregory forward polynomial fit which requires that the column spacing be identical and that row spacing be identical, though they need not be identical to one another. One can set up biquadratic (or any windowing) interpolation without even row and column spacing, but as this is so rarely found in NGS grids it is not addressed.

So, for this method to work, one must have:

$$\Delta X = X_{i+1} - X_i = constant \quad (1)$$

$$\Delta Y = Y_{j+1} - Y_j = constant \quad (2)$$

but ΔX does not need to equal ΔY.

2.2 Quadratic fit across rows

Consider the figure below. The POI is green, and is point "0" with Cartesian coordinates (X_0,Y_0). This POI is surrounded by its 9 nearest grid points, in red. Each of these grid points has a gridded value, Z, so that $Z(X_1, Y_2)$ is the gridded value for the point at (X_1,Y_2) etc.

For the purpose of providing a step-by-step numerical example, some example values of the X and Y coordinates, as well as the gridded Z values are provided in Figure 2 below.

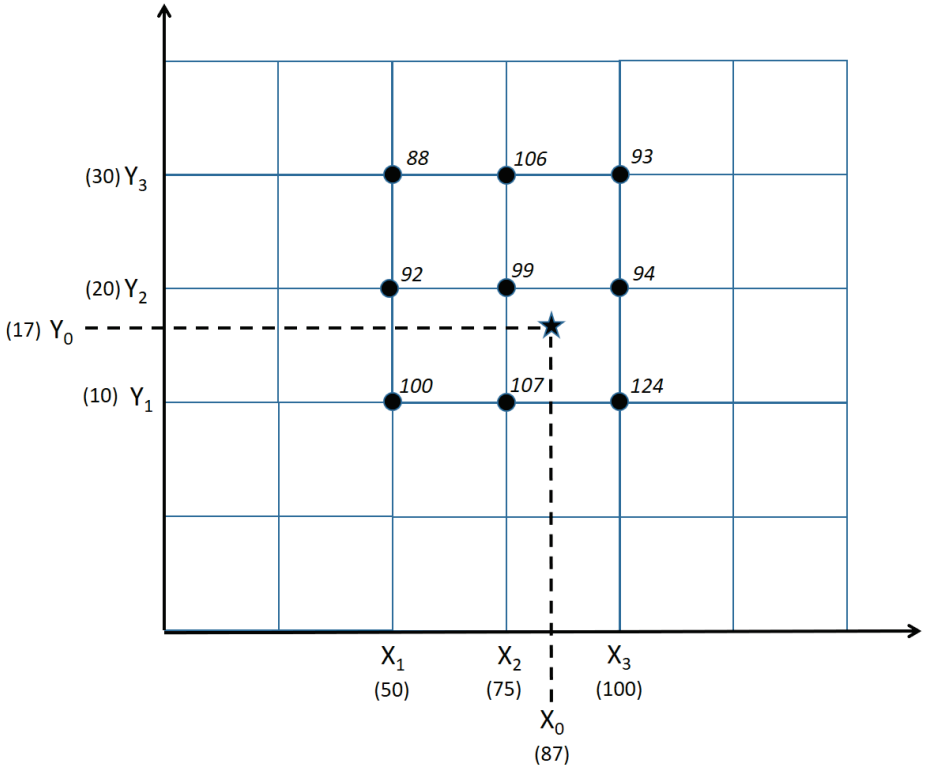

Figure 2: An example problem showing the 3 × 3 window of values and point of interpolation.

The goal of this interpolation is to find $Z(X_0, Y_0)$, located at the star.

The first step is to fit a quadratic function through the top row of values, using the changing X values as the abscissa, and the gridded Z values as the ordinate. That is, a quadratic function is first fit through these three coordinate pairs: (50, 88), (75, 106) and (100, 93). This function is then evaluated at X_0 and stored in memory. The same procedure occurs for the middle and bottom rows. This means finding the three values in the figure below represented by squares.

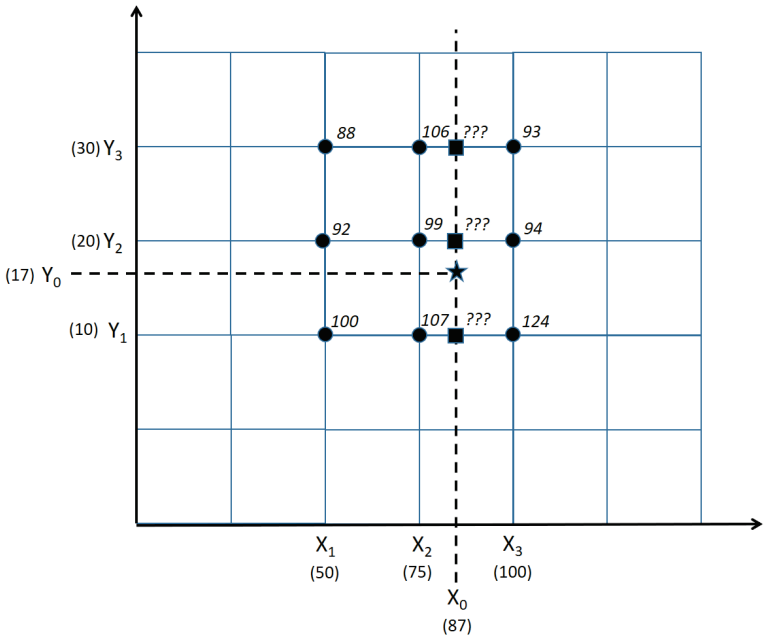

Figure 3: Identifying the first quadratically interpolated values in three rows.

These three quadratically interpolated values will then be used in the final, column-wise quadratic fit in order to arrive at the final biquadratically interpolated value, at the star in Figure 3.

Within NGS software, the quadratic fit is performed first by normalizing the values X_1, X_2, X_3 from their original values to 0, 1 and 2 within the greater subroutine **`biquad.f`**, which are then sent to function **`qterp.f`** for the quadratic function determination (see chapter 4.) This quadratic function is then evaluated at X_0 (also normalized to fall between 0 and 2) and returned. In our numerical example, the quadratic function and its evaluation at X_0 are shown below. The normalization has been skipped in the figure below, just so the numbers line up with the actual data.

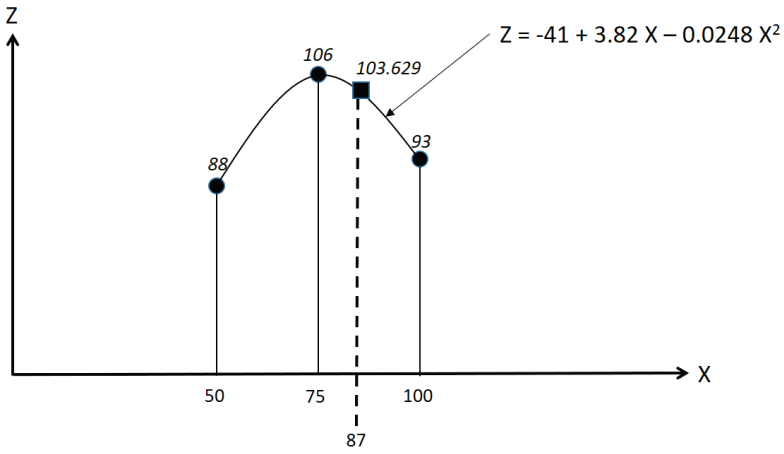

Figure 4: The first (of three) quadratically interpolated values

In this case, the top row has yielded a quadratically interpolated value of 103.629 at X=87. Thus we gain our first coordinate pair (30, 103.629) for the eventual column-wise final interpolation step, where 30 is the Y value of the top row.

This process is repeated for the middle and bottom rows, each yielding its own quadratic function which is then evaluated at $X = 87$, yielding two additional coordinate pairs. For the middle row it is (20, 98.098) and for the bottom row it is (10, 113.912). Thus our information content has changed to that in Figure 5.

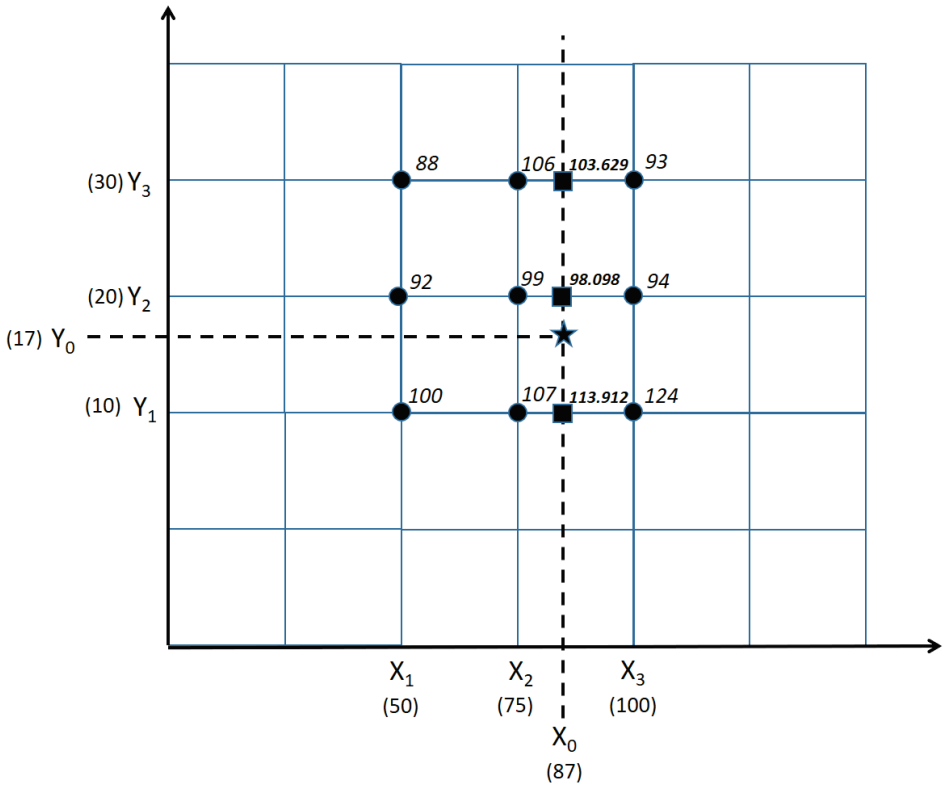

Figure 5: Information content after three row-wise quadratic interpolations.

2.3 Final quadratic fit across columns

Having found quadratically interpolated values in each of three rows, these values are then used in a column-wise quadratic interpolation, using the same procedure of normalizing values and sending to subroutine **`qterp.f`**. This mix of row-wise quadratic interpolations and a column-wise quadratic interpolation is why the entire process is called "biquadratic" interpolation.

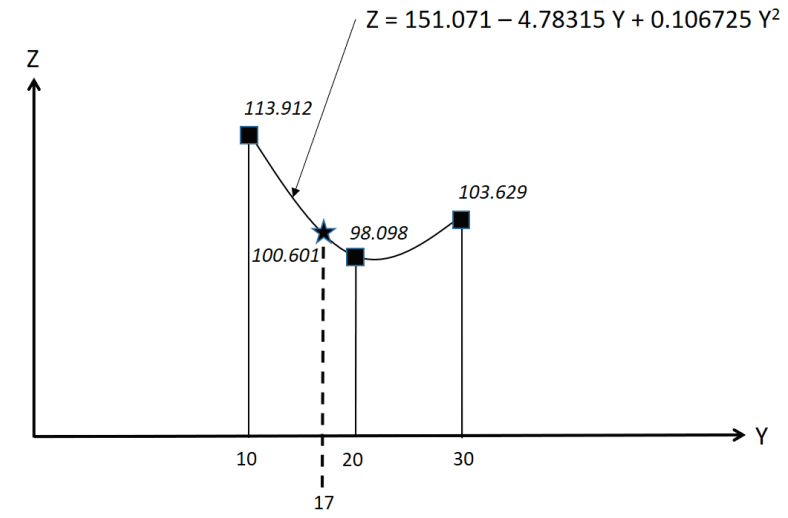

Figure 6: The final quadratic interpolation, this time column-wise

As such, we conclude that for our example the biquadratically interpolated value of Z at (X_0,Y_0 = 17,87) is 100.601.

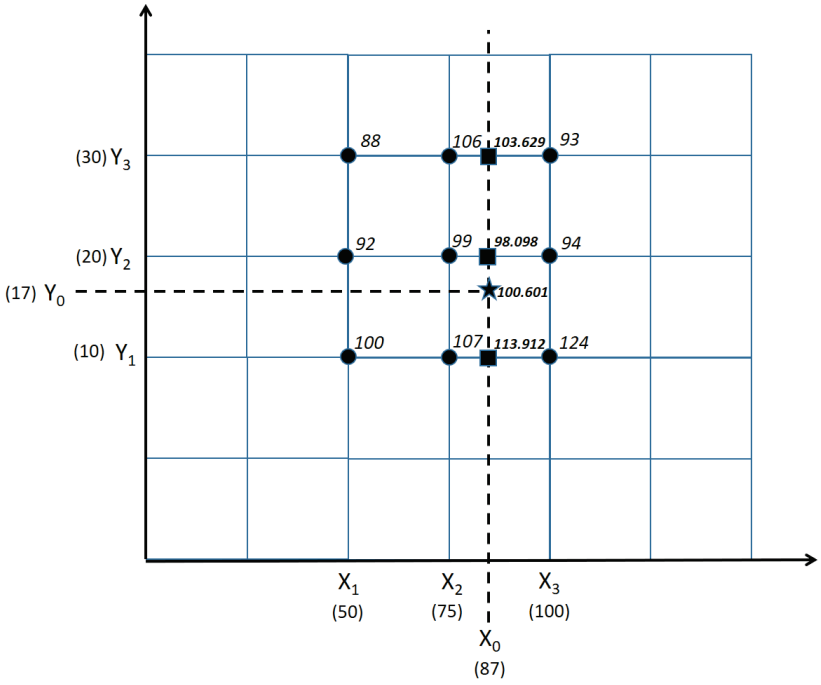

Figure 7: The final results of biquadratic interpolation

3 Discussion

Biquadratic interpolation is just one of many choices of interpolation, and comes with advantages and disadvantages. Unless one uses splines, then windowing interpolators (bilinear, biquadratic, bi-cubic) will always come with the potential for small step functions between

interpolated values at neighboring POIs as the window shifts with the location of each POI. However, the use of splines requires a significantly larger commitment of RAM than a window interpolator. As NGS considered the modernization of the NSRS, tools were built with ease-of-use in mind, and therefore most grid-based interpolation schemes (whether it be from GEOID models or NADCON and VERTCON modules built within other NGS tools) will generally rely upon biquadratic interpolation.

4 Code

Although most of the tools being developed for the modernized NSRS are being coded in JAVA or other modern languages, the origin of the biquadratic interpolator was a FORTRAN subroutine called **`biquad.f`**, created by Dr. Dennis Milbert in the 1990s, with its companion function **`qterp.f`**, both of which are provided below.

4.1 Subroutine biquad.f

```
      subroutine biquad(z,glamn,glomn,dla,dlo,
     *nla,nlo,maxla,maxlo,xla,xlo,val)

c - Subroutine to perform a 2-D quadratic ("biquadratic")
c - interpolation at location "xla,xlo" off of grid "z", whose
c - header information is the standard ".b" header
c - information of
c -    glamn = minimum latitude   (real*8 decimal degrees)
c -    glomn = minimum longitude  (real*8 decimal degrees)
c -    dla   = latitude spacing   (real*8 decimal degrees)
c -    dlo   = longitude spacing  (real*8 decimal degrees)
c -    nla   = number of lat rows (integer*4)
c -    nlo   = number of lon cols (integer*4)
c -    maxla = actual dimensioned size of "z" in rows
c -    maxlo = actual dimensioned size of "z" in cols
c - Further input:
c -    xla   = lat of pt for interpolation (real*8 dec. deg)
c -    xlo   = lon of pt for interpolation (real*8 dec. deg)
c - Output:
c -    val   = interpolated value (real*8)
c -
c - Method:
c -   Fit a 3x3 window over the random point.  The closest
c -   2x2 points will surround the point.  But based on which
c -   quadrant of that 2x2 cell in which the point falls, the
c -   3x3 window could extend NW, NE, SW or SE from the 2x2
cell.
```

```
c - Data is assumed real*4
      implicit real*8 (a-h,o-z)
      real*4 z(maxla,maxlo)

c - Internal use
      real*4 x,y,fx0,fx1,fx2,qterp

c - Find which row should be LEAST when fitting
c - a 3x3 window around xla.
      difla = (xla - glamn)
      ratla = difla / (dla/2.d0)
      ila = int(ratla)+1
      if(mod(ila,2).ne.0)then
        jla = (ila+1)/2 - 1
      else
        jla = (ila  )/2
      endif
c - Fix any edge overlaps
      if(jla.lt.1)jla = 1
      if(jla.gt.nla-2)jla = nla-2

c - Find which column should be LEAST when fitting
c - a 3x3 window around xlo.
      diflo = (xlo - glomn)
      ratlo = diflo / (dlo/2.d0)
      ilo = int(ratlo)+1
      if(mod(ilo,2).ne.0)then
        jlo = (ilo+1)/2 - 1
      else
        jlo = (ilo  )/2
      endif
c - Fix any edge overlaps
      if(jlo.lt.1)jlo = 1
      if(jlo.gt.nlo-2)jlo = nlo-2

c - In the range of 0(westernmost) to
c - 2(easternmost) col, where is our
c - random lon value?  That is, x must
c - be real and fall between 0 and 2.

      x=(xlo-dlo*(jlo-1)-glomn)/dlo

      if(x.lt.0.0)then
        val = 1.d30
```

```
        write(6,100)x,val
        return
      endif
  100 format(
     *'FATAL in biquad:  x<0 : ',f20.10,/,
     *' --> Returning with val = ',f40.1)

      if(x.gt.2.0)then
        val = 1.d30
        write(6,101)x,val
        return
      endif
  101 format(
     *'FATAL in biquad:  x>2 : ',f20.10,/,
     *' --> Returning with val = ',f40.1)

c - In the range of 0(southernmost) to
c - 2(northernmost) row, where is our
c - random lat value?  That is, x must
c - be real and fall between 0 and 2.

      y=(xla-dla*(jla-1)-glamn)/dla

      if(y.lt.0.0)then
        val = 1.d30
        write(6,102)y,val
        return
      endif
  102 format(
     *'FATAL in biquad:  y<0 : ',f20.10,/,
     *' --> Returning with val = ',f40.1)

      if(y.gt.2.0)then
        val = 1.d30
        write(6,103)y,val
        return
      endif
  103 format(
     *'FATAL in biquad:  y>2 : ',f20.10,/,
     *' --> Returning with val = ',f40.1)

c - Now do the interpolation.  First, build a paraboloa
c - east-west the southermost row and interpolate to longitude
c - "xlo" (at "x" for 0<x<2).  Then do it in the middle
```

```
c - row, then finally the northern row.  The last step
c - is to fit a parabola north-south at the three previous
c - interpolations, but this time to interpolate to
c - latitude "xla" (at "y" for 0<y<2).  Obviously we
c - could reverse the order, doing 3 north-south parabolas
c - followed by one east-east and we'd get the same answer.

      fx0=qterp(x,z(jla  ,jlo),z(jla  ,jlo+1),z(jla  ,jlo+2))
      fx1=qterp(x,z(jla+1,jlo),z(jla+1,jlo+1),z(jla+1,jlo+2))
      fx2=qterp(x,z(jla+2,jlo),z(jla+2,jlo+1),z(jla+2,jlo+2))
      val=dble(qterp(y,fx0,fx1,fx2))

      return
      end
      include '/home/dru/Subs/qterp.f'
```

4.2 Function qterp.f

```
      real function qterp(x,f0,f1,f2)

c - x = real*4
c - f0,f1,f2 = real*4

c - This function fits a parabola (quadratic) through
c - three points, *equally* spaced along the x-axis
c - at indices 0, 1 and 2.  The spacing along the
c - x-axis is "dx"
c - Thus:
c -
c -    f0 = y(x(0))
c -    f1 = y(x(1))
c -    f2 = y(x(2))
c -       Where
c -       x(1) = x(0) + dx
c -       x(2) = x(1) + dx

c - The input value is some value of "x" that falls
c - between 0 and 2.  The output value (qterp2) is
c - the parabolic function at x.
c -
c - This function uses Newton-Gregory forward polynomial

      df0 =f1 -f0
```

```
df1 =f2 -f1
d2f0=df1-df0

qterp=f0 + x*df0 + 0.5*x*(x-1.0)*d2f0

return
end
```

5 Bibliography

Coast and Geodetic Survey, 1967: Block Analytic Aerotriangulation, *ESSA Technical Report*, C&GS 35.

Coast and Geodetic Survey, 1970: A Comparison of Methods for Computing Gravitational Potential Derivatives, *ESSA Technical Report*, C&GS 40.