MODIS Airborne Simulator Infrared Calibration

Chris Moeller and Liam Gumley, SSEC UW-Madison
November 11, 1999

Algorithm

The total observed radiance R_{tot} during the view of a MAS blackbody, taking into account non-unit blackbody emissivity, is given by

$$R_{tot} = \varepsilon_b.(R_b + \Delta R_b) + (1 - \varepsilon_b).R_r \qquad (1)$$

where

R_b and ε_b are the MAS blackbody radiance and emissivity, and
R_r is the background radiance reflected by the blackbody.

The ΔR_b term represents radiance error resulting from blackbody temperature measurement error. The reflected radiance R_r is dominated by radiance from the MAS instrument scan cavity hardware. A contribution, as yet unquantified and unaccounted for in the MAS L1B production code, originates from the scene mirror and the earth scene below the ER-2 aircraft. Provisions for these contributions may be added at a later date as MAS performance characterization improves.

Using eqn (1) a linear calibration equation for MAS emissive bands can be constructed. The linear calibration expressly includes correction terms for non-unit blackbody emissivity and background radiance reflected by the blackbodies, and assumes that both blackbodies have the same emissivity. The blackbody temperature measurement error ΔR_b has now been incorporated into the blackbody emissivity correction term ε_b, making that term an *effective* emissivity. The form of the linear calibration equation is

$$R_s = S_b.C_s + I_b \qquad (2)$$

where

R_s and C_s refer to scene radiance and digital counts, and
S_b and I_b refer to calibration slope and intercept.

The calibration slope S_b and the intecept I_b are given by the equations

$$S_b = \varepsilon_b.\left(\frac{R_w - R_a}{C_w - C_a}\right)$$

$$I_b = R_a + (R_r - R_a).(1 - \varepsilon_b) - S.C_a$$

where

R_w and R_a refer to warm and ambient blackbody measured radiance,
C_w and C_a refer to warm and ambient blackbody digital counts.

Eqn (2) is the fundamental emissive band linear calibration equation applied in the MAS L1B algorithm (module `mascal`).

To implement eqn (2), the MAS blackbody temperature measurements must be converted to equivalent MAS broadband radiances. This is accomplished by converting temperature to radiance in a Planck function for each MAS band (spectral characteristics based on MAS laboratory measured SRF). Because the Planck function is monochromatic, it is necessary to apply monochromaticity correction coefficients to produce a broadband radiance. The monochromaticity correction coefficients are based on an integration of the monochromatic Planck function over the bandwidth of a MAS band. In practice this is accomplished by performing this integration for a wide range of scene temperatures and forming a linear relationship between the monochromatic Planck temperature and the broadband Planck temperature for each MAS band. The monochromaticity correction coefficients (a slope and offset) need only be calculated once for each spectral band and are applicable as long as the spectral shape and position of a MAS band remains constant. In the calibration procedure, the MAS blackbody measured temperature is first corrected for monochromaticity effects, and then the Planck function is applied to obtain a band radiance for each blackbody. The ambient and warm blackbody radiances are then used in eqn (2) to represent R_a and R_w respectively.

Another piece of information required in eqn (2) is the background reflected radiance (TBACK in MAS L1B code). Physically, this is the radiance that is reflected by each MAS blackbody into the MAS fore optics when the scan mirror faces each blackbody. The majority of this radiance originates in the scan cavity and thus an estimate of the scan cavity temperature conditions is necessary to estimate R_r. The scan cavity temperature estimate is produced for each scan by calibrating digital counts when the MAS scan mirror views upwards into the scan head. Typically, band 45 digital counts ("THEAD2" for band 45) are used however any of the MAS low noise bands would suffice. Since MAS earth scene calibration sensitivity to background radiance is fairly small, a rough estimate (within 1 to 2°C) of scan cavity temperature is quite sufficient (a general rule of thumb is that error in the scan cavity temperature scales by 0.10 to error in MAS earth scene temperature). For the calculation of TBACK, eqn (2) is applied with an assumption of effective emissivity ε=1.0. Once TBACK is known, it is then converted to an equivalent Planck function radiance for each MAS band via the process explained in the previous paragraph. This is the estimate of R_r for each MAS band.

Estimates of the blackbody effective emissivity (based on laboratory measurements) are necessary in eqn (2). Historically, the effective emissivities are stable inflight and over the timeframe of months to years. However, provisions for updates to these values when the blackbodies are recoated or after being used in marine environments should be planned.

MAS earth scene digital counts (C_s) are collected for each of 716 pixels at 50m resolution on a scan. MAS blackbody digital counts (C_w and C_a) are the result of averaging 25 consecutive views of each blackbody on each scan. With this information, eqn (2) can be solved for an earth scene radiance for each MAS pixel of every emissive band.

Figure 1, which shows a typical MAS linear calibration plot, demonstrates that the effective emissivity reduces the calibration slope from the unit emissivity condition (compare solid and

dashed line slope). The effect on the offset is to shift the entire calibration line up or down depending on the reflected radiance contribution (compare the two dashed lines).

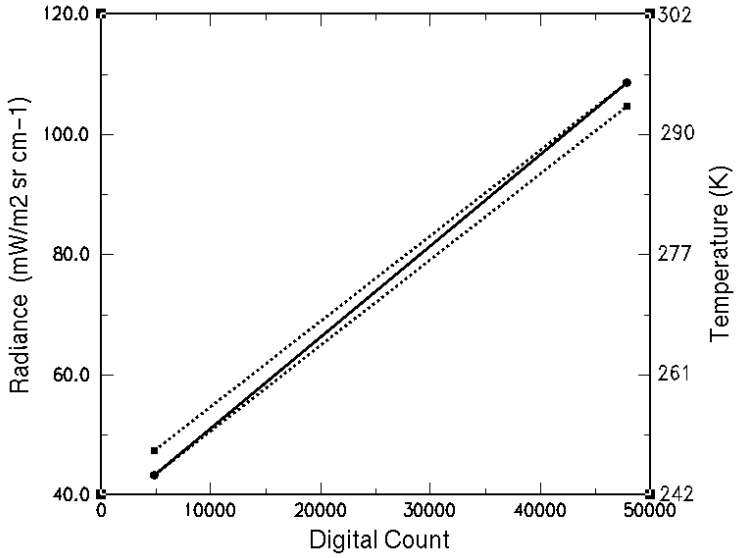

Figure 1. MAS thermal calibration relationship. Non-unit blackbody emissivity calibration slope (dotted lines) is reduced from unit emissivity calibration slope (solid line) by the emissivity factor. Calibration line moves up or down as a function of reflected radiation term.

For more information:
King, Michael D., W. Paul Menzel, Patrick S. Grant, Jeffrey S. Myers, G. Thomas Arnold, Steven E. Platnick, Liam E. Gumley, Si-Chee Tsay, Christopher C. Moeller, Michael Fitzgerald, Kenneth S. Brown, and Fred G. Osterwisch, 1996: Airborne scanning spectrometer for remote sensing of cloud, aerosol, water vapor, and surface properties. Journal of Atmospheric and Oceanic Technology, 13, 777-794.

Software Overview

The MAS emissive band calibration is implemented by the attached FORTRAN-77 modules. The primary modules that are called by the main program are `tembck` and `mascal`. The `tembck` module estimates the MAS head temperature given the head count for an IR window band (usually band 45). The `mascal` module computes MAS IR band calibration slope and intercept with correction for non-unit blackbody emissivity. The software is available at the following URL:

ftp://origin.ssec.wisc.edu/pub/MAS/mascal/src

Inputs Required for each Scanline

For estimating the radiance from the MAS scan cavity (head temperature) by module tembck:
- Band number (45 is the usual value; use 46 or 47 if 45 is bad)
- Digital counts for blackbody 1
- Digital counts for blackbody 2
- Digital counts from MAS scan cavity (head) view

- Temperature for blackbody 1
- Temperature for blackbody 2

For calibrating each MAS band by module mascal:
- Band number (26-50)
- Digital counts for blackbody 1
- Digital counts for blackbody 2
- Temperature for blackbody 1
- Temperature for blackbody 2
- Estimated head temperature

Outputs produced for each Scanline

- Calibration slope (Watts per square meter per steradian per micron per count)
- Calibration intercept (Watts per square meter per steradian per micron)

Calling Sequence

Each indent level represents one level in the calling tree.

```
calling routine
    tembck    (compute head temperature)
        initcf    (store temperature correction coefficients for IR bands)
            getcof    (compute temperature correction coefficients)
                cenwav    (compute effective central wavelength)
                hmwsrf    (convert spectral response to 1 wavenumber spacing)
                    getlun    (get a logical unit)
                    interp    (interpolate a linear sequence)
                bndfit    (compute temperature correction slope, intercept)
                    wbrite    (compute brightness temp, wavenumber units)
                    wplanc    (compute Planck radiance, wavenumber units)
                    linreg    (compute line of best fit)
        masrad    (compute MAS radiance)
            planck    (compute Planck radiance, wavelength units)
        masbrt    (compute MAS brightness temperature)
            bright    (compute brightness temp, wavelength units)
    mascal    (compute IR calibration slope, intercept)
        masrad (see above)
        initcf (see above)
```

Error Checking

Each module checks it's input for appropriate values. The modules are designed to print a warning and continue execution if a recoverable error occurs. If a fatal error occurs, then a warning message is printed and execution stops.

```
c-----------------------------------------------------------------------
c
c  Name:
c      SUBROUTINE BNDFIT
c
c  Purpose:
c      Compute monochromaticity temperature correction coefficients
c      given the spectral response for an infrared band.
c
c  Usage:
c      CALL BNDFIT( N, W, R, WC, T1, T2, TCS, TCI )
c
c  Input:
c      N       Number of points in wavenumber and spectral response
c              arrays
c      W       Array of wavenumbers where spectral response is defined,
c              typically at 0.1 wavenumber resolution
c              (inverse centimeters)
c      R       Spectral response values (normalized to 1.0 or
c              un-normalized are both acceptable)
c      WC      Effective central wavenumber for the band
c              (see function CENWAV)
c      T1      Lower limit of expected earth scene temperature range (K)
c      T2      Upper limit of expected earth scene temperature range (K)
c
c  Output:
c      TCS     Temperature correction slope (no units)
c      TCI     Temperature correction intercept (K)
c
c  Revised:
c      Liam.Gumley@ssec.wisc.edu
c      $Id: bndfit.f,v 1.4 1999/11/11 20:04:43 gumley Exp $
c
c  Notes:
c      The temperature correction coefficients are applied in the
c      Planck function as
c
c      R = B( WC, TCS*T+TCI )
c
c      where R is the Planck radiance in the infrared band,
c            WC is the effective central wavenumber,
c            T is the blackbody temperature,
c            B( W, TCS*T+TCI ) is the Planck function.
c
c-----------------------------------------------------------------------

      subroutine bndfit( n, w, r, wc, t1, t2, tcs, tci )

      implicit none

c ... arguments

      integer n
      real w( * ), r( * ), wc, t1, t2, tci, tcs

c ... local variables
```

```
      integer npts, i, j
      parameter( npts = 100 )
      real r1, r2, rad, te( npts ), ti( npts ), a, b, dw, rx, wx,
     &  wbrite, wplanc
      double precision xbar, ybar, xsig, ysig, yint, slope,
     &  corr, evar, serr
      external wbrite, wplanc

c ... establish 100 point equal spacing in radiance between t1 and t2,
c ... and then convert radiances to brightness temperatures

      r1 = wplanc( wc, t1 )
      r2 = wplanc( wc, t2 )
      do i = 1, npts
        rad = ( r2 - r1 ) * real( i - 1 ) / real( npts - 1 ) + r1
        te( i ) = wbrite( wc, rad )
      end do

c ... compute true response weighted radiance for each element of te,
c ... and compute corresponding brightness temperature

      do j = 1, npts
        a = 0.0
        b = 0.0
        do i = 1, n - 1
            dw = w( i + 1 ) - w( i )
            wx = w( i ) + 0.5 * dw
            rx = 0.5 * ( r( i ) + r( i + 1 ) )
            a = a + dw * rx * wplanc( wx, te( j ) )
            b = b + dw * rx
         enddo
         rad = a / b
         ti( j ) = wbrite( wc, rad )
      enddo

c ... compute linear fit for ti vs. te,
c ... and return slope and intercept

      call linreg( npts, te, ti,
     &  xbar, ybar, xsig, ysig, yint, slope, corr, evar, serr )
      tcs = sngl( slope )
      tci = sngl( yint )

      end
```

```
c------------------------------------------------------------------------
c
c  Name:
c      FUNCTION BRIGHT
c
c  Purpose:
c      Compute brightness temperature in Kelvin given monochromatic
c      Planck radiance in Watts per square meter per steradian per
c      micron.
c
c  Usage:
c      RESULT = BRIGHT( WV, RAD )
c
c  Input:
c      WV      Wavelength (microns)
c      RAD     Planck radiance (Watts per square meter per
c              steradian per micron)
c
c  Output:
c      RESULT  Brightness temperature (Kelvin)
c
c  Revised:
c      Liam.Gumley@ssec.wisc.edu
c      $Id: bright.f,v 1.3 1999/11/11 20:04:43 gumley Exp $
c
c------------------------------------------------------------------------

      real function bright( wv, rad )

      implicit none

c ... arguments

      real wv, rad

c ... local variables

      double precision h, c, b, c1, c2, ws

c ... Planck function constants

      parameter( h = 6.6260755d-34, c = 2.9979246d+8,
     &  b = 1.380658d-23, c1 = 2.0d0 * h * c**2, c2 = h * c / b )

c ... scale wavelength from microns to meters

      ws = 1.0d-6 * dble( wv )

c ... compute brightness temperature

      bright = sngl( c2 /
     &  ( ws * log( c1 / ( 1.0d6 * dble( rad ) * ws**5 ) + 1.0d0 ) ) )

      end
```

```
c------------------------------------------------------------------------
c
c  Name:
c      FUNCTION CENWAV
c
c  Purpose:
c      Compute an effective central wavenumber from an array of
c      spectral response values.
c
c  Usage:
c      RESULT = CENWAV( N, W1, W2, SRF )
c
c  Input:
c      N       Number of points in spectral response array
c      W1      Lower wavenumber limit of spectral response array
c              (inverse centimeters)
c      W2      Upper wavenumber limit of spectral response array
c              (inverse centimeters)
c      SRF     Spectral response data, typically at 0.1 inverse
c              centimeter resolution (normalized to 1.0 or
c              un-normalized are both acceptable).
c
c  Output:
c      CENWAV  Effective central wavenumber (inverse centimeters)
c
c  Revised:
c      Liam.Gumley@ssec.wisc.edu
c      $Id: cenwav.f,v 1.4 1999/11/12 15:51:18 gumley Exp $
c
c------------------------------------------------------------------------

      real function cenwav( n, w1, w2, srf )

      implicit none

c ... arguments

      integer n
      real w1, w2, srf( * )

c ... local variables

      double precision sumw, sumr
      real w
      integer i

c ... compute central wavenumber

      sumw = 0.0d0
      sumr = 0.0d0

      do i = 1, n
        w = ( w2 - w1 ) * real( i - 1 ) / real( n - 1 ) + w1
        sumw = sumw + dble( srf( i ) ) * dble( w )
        sumr = sumr + dble( srf( i ) )
      enddo
```

```
cenwav = sngl( sumw / sumr )

end
```

```
c-----------------------------------------------------------------------
c
c  Name:
c      SUBROUTINE GETCOF
c
c  Purpose:
c      Get effective central wavenumber and temperature correction
c      coefficients for a MAS IR band, given a range of expected earth
c      scene temperatures.
c
c  Usage:
c      CALL GETCOF( BAND, T1, T2, WC, TCS, TCI )
c
c  Input:
c      BAND    MAS IR band number (26-50)
c      T1      Lower limit of expected earth scene temperature range (K)
c      T2      Upper limit of expected earth scene temperature range (K)
c
c  Output:
c      WC      Effective central wavenumber (inverse centimeters)
c      TCS     Temperature correction slope (no units)
c      TCI     Temperature correction intercept (K)
c
c  Revised:
c      Liam.Gumley@ssec.wisc.edu
c      $Id: getcof.f,v 1.3 1999/11/11 20:04:44 gumley Exp $
c
c  Notes:
c      The temperature correction coefficients are applied in the
c      Planck function as
c
c      R = B( WC, TCS*T+TCI )
c
c      where R is the Planck radiance in the infrared band,
c            WC is the effective central wavenumber,
c            T is the blackbody temperature,
c            B( W, TCS*T+TCI ) is the Planck function.
c
c-----------------------------------------------------------------------

      subroutine getcof( band, t1, t2, wc, tcs, tci )

      implicit none

c ... arguments

      integer band
      real t1, t2, tcs, tci

c ... local variables

      character*20 file
      integer n, i
      real w1, w2, r( 10000 ), wc, cenwav, w( 10000 )
      external cenwav

c ... check band number
```

```
      if( band .lt. 26 .or. band .gt. 50 ) then
        write(*,*) 'Fatal Error in subroutine GETCOF'
        write(*,*) 'MAS IR band number was out of the range [26-50]'
        write(*,*) 'Offending value was ', band
        stop
      endif

c ... get interpolated spectral response

      file = 'srf-'//char(band/10+48)//char(mod(band,10)+48)//'.asc'
      call hmwsrf( file, n, w1, w2, r )

c ... get central wavenumber

      wc = cenwav( n, w1, w2, r )

c ... make wavenumber array

      do i = 1, n
        w( i ) = ( w2 - w1 ) * real( i - 1 ) / real( n - 1 ) + w1
      end do

c ... get temperature correction slope and intercept

      call bndfit( n, w, r, wc, t1, t2, tcs, tci )

      end
```

```
c-----------------------------------------------------------------------
c
c  Name:
c      FUNCTION GETLUN
c
c  Purpose:
c      Get a logical unit number that is not in use.
c
c  Usage:
c      RESULT = GETLUN()
c
c  Input:
c      Nothing
c
c  Output:
c      RESULT  Logical unit number (use in OPEN statement for example)
c
c  Revised:
c      Liam.Gumley@ssec.wisc.edu
c      $Id: getlun.f,v 1.3 1999/11/11 20:04:44 gumley Exp $
c
c-----------------------------------------------------------------------

      integer function getlun()

      implicit none

c ... local variables

      integer i
      logical od

      getlun = -1

c ... search unit numbers from 20 onwards for one that is not in use

      i = 20
10    od = .false.
      inquire( i, opened = od )
      if( od ) then
        i = i + 1
        goto 10
      endif

      getlun = i

      end
```

```
c-----------------------------------------------------------------------
c
c  Name:
c      SUBROUTINE HMWSRF
c
c  Purpose:
c      Read a spectral response data file and linearly interpolate
c      the spectral response to 0.1 inverse centimeter resolution.
c
c  Usage:
c      CALL HMWSRF( FILE, N, W1, W2, SRF )
c
c  Input:
c      FILE    Name of spectral response data file
c              The input spectral response data file should be ASCII
c              format with two columns. The first column should be
c              wavelength in microns, and the second column should
c              be spectral response (normalized to 1.0 or un-normalized
c              are both acceptable).
c              Wavelength values in the range [0.4-20.0] are accepted.
c              Wavelength values outside this range will cause an exit.
c              Negative response values in the range [-0.01,0.0] will be
c              set to zero. Negative values less than -0.01 will cause
c              an exit.
c
c  Output:
c      N       Number of points in interpolated output array
c      W1      Lower wavenumber limit of interpolated output array
c              (inverse centimeters)
c      W2      Upper wavenumber limit of interpolated output array
c              (inverse centimeters)
c      SRF     Linearly interpolated spectral response at
c              0.1 inverse centimeter resolution, normalized so that
c              maximum respose is 1.0
c
c  Revised:
c      Liam.Gumley@ssec.wisc.edu
c      $Id: hmwsrf.f,v 1.3 1999/11/11 20:04:44 gumley Exp $
c
c  Notes:
c      (1) Calls the linear interpolation subroutine INTERP.
c
c-----------------------------------------------------------------------

      subroutine hmwsrf( file, n, w1, w2, srf )

      implicit none

c ... arguments

      character*(*) file
      integer n
      real w1, w2, srf( * )

c ... local variables

      real xold( 10000 ), yold( 10000 ), xnew( 10000 ), temp( 10000 ),
```

```
     &     big
      integer nold, ios, i, j, lun, getlun
      external getlun

c ... open input file

      lun = getlun()
      if( lun .le. 0 ) then
        write(*,*) 'Fatal Error in subroutine HMWSRF'
        write(*,*) 'Could not get a free logical unit number'
        stop
      endif
      open( unit = lun, file = file, status = 'old', iostat = ios )
      if( ios .ne. 0 ) then
        write(*,*) 'Fatal Error in subroutine HMWSRF'
        write(*,*) 'Could not open input file'
        write(*,*) 'Input filename is ', file
        stop
      endif

c ... read wavelength and spectral response values and check ranges

      nold = 0
20    continue
      read( unit = lun, fmt = *, iostat = ios )
     &  xold( nold + 1 ), yold( nold + 1 )
      if( ios .ne. 0 ) goto 40

      nold = nold + 1

      if( xold( nold ) .le. 0.4 .or. xold( nold ) .gt. 20.0 ) then
        write(*,*) 'Fatal Error in subroutine HMWSRF'
        write(*,*) 'Input wavelength is outside the range [0.4,20.0]'
        write(*,*) 'Offending value is at element ', nold,
     &             ', value is ', xold( nold )
        write(*,*) 'Input filename is ', file
        stop
      endif

      if( yold( nold ) .lt. -0.01 ) then
        write(*,*) 'Fatal Error in subroutine HMWSRF'
        write(*,*) 'Input response is less than -0.01'
        write(*,*) 'Offending value is at element ', nold,
     &             ', value is ', yold( nold )
        write(*,*) 'Input filename is ', file
        stop
      endif
      yold( nold ) = max( yold( nold ), 0.0 )

      goto 20

40    continue

      close( unit = lun )

c ... reverse order of spectral response data
```

```
      j = 1
      do i = nold, 1, -1
        temp( j ) = xold( i )
        j = j + 1
      end do
      do i = 1, nold
        xold( i ) = temp( i )
      end do

      j = 1
      do i = nold, 1, -1
        temp( j ) = yold( i )
        j = j + 1
      end do
      do i = 1, nold
        yold( i ) = temp( i )
      end do

c ... convert wavelength (microns) to wavenumber

      do i = 1, nold
        xold( i ) = 1.0e4 / xold( i )
      end do

c ... construct new wavenumber array at 0.1 inverse centimeter intervals

      w1 = xold( 1 ) - mod( xold( 1 ), 0.1 ) - 0.05
      w2 = xold( nold ) - mod( xold( nold ), 0.1 ) + 0.15
      n = int( ( w2 - w1 ) / 0.1 ) + 1
      do i = 1, n
        xnew( i ) = ( w2 - w1 ) * real( i - 1 ) / real( n - 1 ) + w1
      end do

c ... interpolate to new wavenumber array spacing

      call interp( nold, xold, yold, n, xnew, srf )
      srf( 1 ) = 0.0
      srf( n ) = 0.0

c ... normalize to maximum value of 1.0

      big = -1.0e8
      do i = 1, n
        big = max( big, srf( i ) )
      end do
      do i = 1, n
        srf( i ) = srf( i ) / big
      end do

      end
```

```
c-----------------------------------------------------------------------
c
c  Name:
c      SUBROUTINE INITCF
c
c  Purpose:
c      Compute temperature correction coefficients for all MAS IR bands
c      and store in common block which is accessed by MASRAD and MASBRT.
c
c  Usage:
c      CALL INITCF()
c
c  Input:
c      Nothing

c  Output:
c      The following variables are stored in COMMON block MASCOF
c      INIT    Flag indicating that initialization is required
c      WC      Effective central wavenumber (inverse centimeters)
c      TCS     Temperature correction slope (no units)
c      TCI     Temperature correction intercept (K)
c
c  Revised:
c      Liam.Gumley@ssec.wisc.edu
c      $Id: initcf.f,v 1.3 1999/11/11 20:04:44 gumley Exp $
c
c-----------------------------------------------------------------------

      subroutine initcf()

c ... local variables

      integer band
      real t1( 50 ), t2( 50 )

c ... common block to be read later by MASRAD and MASBRT

      integer init
      real wc( 50 ), tcs( 50 ), tci( 50 )
      common / mascof / init, wc, tcs, tci

c ... lower and upper limits of expected earth scene temperature range

      data t1 / 50 * 180.0 /, t2 / 50 * 320.0 /

c ... get coefficients for all IR bands

      if( init .ne. 1 ) then

        do band = 26, 50

          call getcof( band, t1( band ), t2( band ),
     &      wc( band ), tcs( band ), tci( band ) )

        end do

c ...   set initialization flag to be read later by MASRAD and MASBRT
```

```
  init = 1
endif
end
```

```
c-----------------------------------------------------------------------
c
c  Name:
c      SUBROUTINE INTERP
c
c  Purpose:
c      Linearly interpolate an input array.
c
c  Usage:
c      CALL INTERP( NOLD, XOLD, YOLD, NNEW, XNEW, YNEW )
c
c  Input:
c      NOLD    Number of elements in the input arrays
c      XOLD    Abscissa values for the input array (must be monotonic)
c      YOLD    Values of the input array
c      NNEW    Number of elements in the output arrays
c      XNEW    Abscissa values for the output array
c
c  Output:
c      YNEW    Linearly interpolated values of the output array
c
c  Revised:
c      Liam.Gumley@ssec.wisc.edu
c      $Id: interp.f,v 1.3 1999/11/11 20:04:45 gumley Exp $
c
c  Notes:
c      Values outside the range of the input abscissae are obtained
c      via extrapolation.
c
c-----------------------------------------------------------------------

      subroutine interp( nold, xold, yold, nnew, xnew, ynew )

      implicit none

      integer nold, nnew
      real xold( nold ), yold( nold ), xnew( nnew ), ynew (nnew )

      real slope, intercept
      integer lo, hi, j, init

      lo = 1
      hi = 2
      init = 1

      do j = 1, nnew

20      continue

c ...   check for monotonicity

        if( xold( lo ) .ge. xold( hi ) ) then
          write(*,*) 'Fatal Error in subroutine INTERP'
          write(*,*) 'Input abscissa array XOLD is not monotonic'
          write(*,*) 'Offending XOLD values are', xold( lo ), xold( hi )
          stop
        endif
```

```
c ...   check if output point falls between current input points

        if( xnew( j ) .gt. xold( hi ) ) then
          if( hi .lt. nold ) then
            lo = lo + 1
            hi = hi + 1
            init = 1
            goto 20
          endif
        endif

c ...   compute slope and intercept only when necessary

        if( init .eq. 1 ) then
          slope = ( yold( hi ) - yold( lo ) ) /
     &      ( xold( hi ) - xold( lo ) )
          intercept = yold( lo ) - slope * xold( lo )
          init = 0
        endif

c ...   compute output value

        ynew( j ) = slope * xnew( j ) + intercept

      end do

      end
```

```
c-----------------------------------------------------------------------
c
c  Name:
c      SUBROUTINE LINREG
c
c  Purpose:
c      Compute line of best fit to a dataset defined in X and Y.
c
c  Usage:
c      CALL LINREG( N, X, Y, XBAR, YBAR, XSIG, YSIG, YINT,
c     &   SLOPE, CORR, EVAR, SERR )
c
c  Input:
c      N       Number of points in X and Y arrays
c      X       X ordinate array
c      Y       Y coordinate array
c
c  Output:
c      XBAR    Mean of X values
c      YBAR    Mean of Y values
c      XSIG    Standard deviation of X values
c      YSIG    Standard deviation of Y values
c      YINT    Intercept on the Y axis of the line of best fit
c      SLOPE   Slope of the line of best fit
c      CORR    Correlation coefficient
c      EVAR    Variance
c      SERR    Standard error
c
c  Revised:
c      Liam.Gumley@ssec.wisc.edu
c      $Id: linreg.f,v 1.3 1999/11/11 20:04:45 gumley Exp $
c
c  Notes:
c      All output parameters are DOUBLE PRECISION.
c
c-----------------------------------------------------------------------

      subroutine linreg( n, x, y, xbar, ybar, xsig, ysig, yint,
     &                   slope, corr, evar, serr )

      implicit none

c ... arguments

      integer n
      real x( * ), y( * )
      double precision xbar, ybar, xsig, ysig, yint, slope, corr,
     &  evar, serr

c ... local variables

      integer i
      double precision sn, cvar, xp, xvar, yp, yvar

      sn = dble( n )
      xbar = 0.0d0
      ybar = 0.0d0
```

```
do i = 1, n
   xbar = xbar + dble( x( i ) )
   ybar = ybar + dble( y( i ) )
end do
xbar = xbar / sn
ybar = ybar / sn

xvar = 0.0d0
yvar = 0.0d0
cvar = 0.0d0
do i = 1, n
   xp = dble( x( i ) ) - xbar
   yp = dble( y( i ) ) - ybar
   xvar = xvar + xp * xp
   yvar = yvar + yp * yp
   cvar = cvar + xp * yp
end do

xvar = xvar / sn
xsig = sqrt( xvar )
yvar = yvar / sn
ysig = sqrt( yvar )
cvar = cvar / sn
slope = cvar / xvar
yint = ybar - slope * xbar
corr = cvar / ( xsig * ysig )
evar = slope * cvar
serr = sqrt( abs( yvar - evar ) )
evar = evar / yvar

end
```

```
c------------------------------------------------------------------------
c
c  Name:
c      FUNCTION MASBRT
c
c  Purpose:
c      Compute brightness temperature for a MAS IR band given response
c      weighted Planck radiance.
c
c  Usage:
c      RESULT = MASBRT( BAND, RAD )
c
c  Input:
c      BAND    MAS IR band number (26-50)
c      RAD     Planck radiance (Watts per square meter per
c              steradian per micron)
c
c  Output:
c      RESULT  Brightness temperature (Kelvin)
c
c  Revised:
c      Liam.Gumley@ssec.wisc.edu
c      $Id: masbrt.f,v 1.3 1999/11/11 20:04:45 gumley Exp $
c
c  Notes:
c      SUBROUTINE INITCF must be called before calling MASRAD or MASBRT
c
c------------------------------------------------------------------------

      real function masbrt( band, rad )

      implicit none

c ... arguments

      integer band
      real rad

c ... local variables

      real t, w, bright
      external bright

c ... common block of coefficients which is initialized in INITCF
c ... (INITCF must be called before MASRAD or MASBRT)

      integer init
      real wc( 50 ), tcs( 50 ), tci( 50 )
      common / mascof / init, wc, tcs, tci

c ... check that coefficients were created by INITCF

      if( init .ne. 1 ) then
        write(*,*) 'Fatal Error in subroutine MASBRT'
        write(*,*) 'Subroutine INITCF must be called before MASBRT'
        stop
      endif
```

```
c ... check input arguments

      if( band .lt. 26 .or. band .gt. 50 ) then
        write(*,*) 'Fatal Error in subroutine MASBRT'
        write(*,*) 'Input argument BAND was out of the range [26,50]'
        write(*,*) 'Offending value was ', band
        stop
      endif

      if( rad .lt. 0.0 .or. rad .ge. 20.0 ) then
        write(*,*) 'Fatal Error in subroutine MASBRT'
        write(*,*) 'Input argument RAD was out of the range [0.0,20.0]'
        write(*,*) 'Offending value was ', rad
        stop
      endif

c ... check the coefficients just to be sure

      if( wc( band ) .le. 600.0 .or. wc( band ) .gt. 3500.0 ) then
        write(*,*) 'Fatal Error in subroutine MASBRT'
        write(*,*) 'Variable WC was out of the range [600,3500]'
        write(*,*) 'Offending value was at element', band,
     &    ', value was ', wc( band )
        stop
      endif

      if( tcs( band ) .le. 0.98 .or. tcs( band ) .gt. 1.02 ) then
        write(*,*) 'Fatal Error in subroutine MASBRT'
        write(*,*) 'Variable TCS was out of the range [0.98,1.02]'
        write(*,*) 'Offending value was at element', band,
     &    ', value was ', tcs( band )
        stop
      endif

      if( tci( band ) .le. -2.0 .or. tci( band ) .gt. 2.0 ) then
        write(*,*) 'Fatal Error in subroutine MASBRT'
        write(*,*) 'Variable TCI was out of the range [-2.0,2.0]'
        write(*,*) 'Offending value was at element', band,
     &    ', value was ', tci( band )
        stop
      endif

c ... convert effective central wavenumber to wavelength in microns

      w = 1.0e4 / wc( band )

c ... compute brightness temperature

      t = bright( w, rad )

c ... apply temperature correction

      masbrt = ( t - tci( band ) ) / tcs( band )

      end
```

```
c-----------------------------------------------------------------------
c
c  Name:
c      SUBROUTINE MASCAL
c
c  Purpose:
c      Compute MAS IR band calibration slope and intercept with
c      correction for non-unit blackbody emissivity.
c
c  Usage:
C      CALL MASCAL( BAND, BB1C, BB2C, BB1T, BB2T, HEADT,
C     &   SLOPE, INTERCEPT )
c
c  Input:
c      BAND      MAS IR band number (26-50)
c      BB1C      Digital counts for blackbody 1
c      BB2C      Digital counts for blackbody 2
c      BB1T      Temperature for blackbody 1 (Kelvin)
c      BB2T      Temperature for blackbody 2 (Kelvin)
c      HEADT     Estimated head temperature (Kelvin) [function TEMBCK]
c
c  Output:
c      SLOPE     Calibration slope (Watts per square meter per
c                steradian per micron per count)
c      INTERCEPT Calibration intercept (Watts per square meter per
c                steradian per micron)
c
c  Revised:
c      Liam.Gumley@ssec.wisc.edu
c      $Id: mascal.f,v 1.5 1999/11/12 16:39:35 gumley Exp $
c
c-----------------------------------------------------------------------

      subroutine mascal( band, bb1c, bb2c, bb1t, bb2t, headt,
     &  slope, intercept )

      implicit none

c ... arguments

      integer band, bb1c, bb2c
      real bb1t, bb2t, headt, slope, intercept

c ... local variables

      real emis( 50 ), masrad, rad1, rad2, radbck
      external masrad

c ... effective blackbody emissivity for all 50 MAS bands

      data emis / 25 * 1.0,
     &  0.98095, 0.98104, 0.98115, 0.98124, 0.98134,
     &  0.98145, 0.98155, 0.98164, 0.98175, 0.98184,
     &  0.98194, 0.98203, 0.98213, 0.98223, 0.98233,
     &  0.98243, 0.94590, 0.94102, 0.93890, 0.93782,
     &  0.93668, 0.93668, 0.93700, 0.93764, 0.93858 /
```

```
c ... set default return values

      slope = 0.0
      intercept = 0.0

c ... check input arguments

      if( band .lt. 26 .or. band .gt. 50 ) then
        write(*,*) 'Warning in subroutine MASCAL'
        write(*,*) 'Input argument BAND was out of the range [26,50]'
        write(*,*) 'Offending value was ', band
        write(*,*) 'Slope and Intercept set to zero in MASCAL'
        return
      endif

      if( bb1c .le. 0 .or. bb1c .ge. 100000 ) then
        write(*,*) 'Warning in subroutine MASCAL'
        write(*,*) 'Input argument BB1C was out of the range [0,100000]'
        write(*,*) 'Offending value was ', bb1c
        write(*,*) 'Slope and Intercept set to zero in MASCAL'
        return
      endif

      if( bb2c .le. 0 .or. bb2c .ge. 100000 ) then
        write(*,*) 'Warning in subroutine MASCAL'
        write(*,*) 'Input argument BB2C was out of the range [0,100000]'
        write(*,*) 'Offending value was ', bb2c
        write(*,*) 'Slope and Intercept set to zero in MASCAL'
        return
      endif

      if( bb1c .ge. bb2c ) then
        write(*,*) 'Warning in subroutine MASCAL'
        write(*,*) 'Input argument BB1C was GE than BB2C'
        write(*,*) 'Offending values were ', bb1c, bb2c
        write(*,*) 'Slope and Intercept set to zero in MASCAL'
        return
      endif

      if( bb1t .le. 180.0 .or. bb1t .ge. 320.0 ) then
        write(*,*) 'Warning in subroutine MASCAL'
        write(*,*) 'Input argument BB1T was out of the range ' //
     &    '[180.0,320.0]'
        write(*,*) 'Offending value was ', bb1t
        write(*,*) 'Slope and Intercept set to zero in MASCAL'
        return
      endif

      if( bb2t .le. 180.0 .or. bb2t .ge. 320.0 ) then
        write(*,*) 'Warning in subroutine MASCAL'
        write(*,*) 'Input argument BB2T was out of the range ' //
     &    '[180.0,320.0]'
        write(*,*) 'Offending value was ', bb2t
        write(*,*) 'Slope and Intercept set to zero in MASCAL'
        return
      endif
```

```
      if( bb1t .ge. bb2t ) then
        write(*,*) 'Warning in subroutine MASCAL'
        write(*,*) 'Input argument BB1T was GE than BB2T'
        write(*,*) 'Offending values were ', bb1t, bb2t
        write(*,*) 'Slope and Intercept set to zero in MASCAL'
        return
      endif

      if( headt .le. 180.0 .or. headt .ge. 320.0 ) then
        write(*,*) 'Warning in subroutine MASCAL'
        write(*,*) 'Input argument HEADT was out of the range ' //
     &    '[180.0,320.0]'
        write(*,*) 'Offending value was ', headt
        write(*,*) 'Slope and Intercept set to zero in MASCAL'
        return
      endif

c ... initialize planck function temperature correction coefficients
c ... (must be done before calling MASRAD or MASBRT)

      call initcf()

c ... convert blackbody and head temperatures to radiance

      rad1 = masrad( band, bb1t )
      rad2 = masrad( band, bb2t )
      radbck = masrad( band, headt )

c ... compute emissivity corrected calibration slope and intercept

      slope = emis( band ) * ( ( rad2 - rad1 ) / real( bb2c - bb1c ) )
      intercept = rad1 + ( radbck - rad1 ) * ( 1.0 - emis( band ) ) -
     &  slope * real( bb1c )

      end
```

```
c-----------------------------------------------------------------------
c
c  Name:
c      FUNCTION MASRAD
c
c  Purpose:
c      Compute response weighted Planck radiance for a MAS IR band given
c      brightness temperature.
c
c  Usage:
c      RESULT = MASRAD( BAND, TEM )
c
c  Input:
c      BAND    MAS IR band number (26-50)
c      TEM     Brightness temperature (Kelvin)
c
c  Output:
c      RESULT  Planck radiance (Watts per square meter per
c              steradian per micron)
c
c  Revised:
c      Liam.Gumley@ssec.wisc.edu
c      $Id: masrad.f,v 1.3 1999/11/11 20:04:45 gumley Exp $
c
c  Notes:
c      SUBROUTINE INITCF must be called before calling MASRAD or MASBRT
c
c-----------------------------------------------------------------------

      real function masrad( band, tem )

      implicit none

c ... arguments

      integer band
      real tem

c ... local variables

      real t, w, planck
      external planck

c ... common block of coefficients which is initialized in INITCF
c ... (INITCF must be called before MASRAD or MASBRT)

      integer init
      real wc( 50 ), tcs( 50 ), tci( 50 )
      common / mascof / init, wc, tcs, tci

c ... check that coefficients were created by INITCF

      if( init .ne. 1 ) then
        write(*,*) 'Fatal Error in subroutine MASRAD'
        write(*,*) 'Subroutine INITCF must be called before MASRAD'
        stop
      endif
```

```
c ... check input arguments

      if( band .lt. 26 .or. band .gt. 50 ) then
        write(*,*) 'Fatal Error in subroutine MASRAD'
        write(*,*) 'Input argument BAND was out of the range [26,50]'
        write(*,*) 'Offending value was ', band
        stop
      endif

      if( tem .le. 100.0 .or. tem .ge. 1000.0 ) then
        write(*,*) 'Fatal Error in subroutine MASRAD'
        write(*,*) 'Input argument TEM was out of the range [100,1000]'
        write(*,*) 'Offending value was ', tem
        stop
      endif

c ... check the coefficients just to be sure

      if( wc( band ) .le. 600.0 .or. wc( band ) .gt. 3500.0 ) then
        write(*,*) 'Fatal Error in subroutine MASRAD'
        write(*,*) 'Variable WC was out of the range [600,3500]'
        write(*,*) 'Offending value was at element', band,
     &    ', value was ', wc( band )
        stop
      endif

      if( tcs( band ) .le. 0.98 .or. tcs( band ) .gt. 1.02 ) then
        write(*,*) 'Fatal Error in subroutine MASRAD'
        write(*,*) 'Variable TCS was out of the range [0.98,1.02]'
        write(*,*) 'Offending value was at element', band,
     &    ', value was ', tcs( band )
        stop
      endif

      if( tci( band ) .le. -2.0 .or. tci( band ) .gt. 2.0 ) then
        write(*,*) 'Fatal Error in subroutine MASRAD'
        write(*,*) 'Variable TCI was out of the range [-2.0,2.0]'
        write(*,*) 'Offending value was at element', band,
     &    ', value was ', tci( band )
        stop
      endif

c ... apply temperature correction

      t = tem * tcs( band ) + tci( band )

c ... convert effective central wavenumber to wavelength in microns

      w = 1.0e4 / wc( band )

c ... compute Planck radiance

      masrad = planck( w, t )

      end
```

```
c------------------------------------------------------------------------
c
c  Name:
c      FUNCTION PLANCK
c
c  Purpose:
c      Compute monochromatic Planck radiance in Watts per square meter
c      per steradian per micron given brightness temperature in Kelvin.
c
c  Usage:
c      RESULT = PLANCK( WV, TEM )
c
c  Input:
c      WV      Wavelength (microns)
c      TEM     Brightness temperature (Kelvin)
c
c  Output:
c      RESULT  Planck radiance (Watts per square meter per
c              steradian per micron)
c
c  Revised:
c      Liam.Gumley@ssec.wisc.edu
c      $Id: planck.f,v 1.3 1999/11/11 20:04:46 gumley Exp $
c
c------------------------------------------------------------------------

      real function planck( wv, tem )

      implicit none

c ... arguments

      real wv, tem

c ... local variables

      double precision h, c, b, c1, c2, ws

c ... Planck function constants

      parameter( h = 6.6260755d-34, c = 2.9979246d+8,
     &  b = 1.380658d-23, c1 = 2.0d0 * h * c**2, c2 = h * c / b )

c ... scale wavelength from microns to meters

      ws = 1.0d-6 * dble( wv )

c ... compute radiance

      planck = sngl( 1.0d-6 * c1 /
     &  ( ws**5 * ( exp( c2 / ( ws * dble( tem ) ) ) - 1.0d+0 ) ) )

      end
```

```
c-----------------------------------------------------------------------
c
c  Name:
c      FUNCTION TEMBCK
c
c  Purpose:
c      Estimate the MAS head temperature given the head count for an
c      IR window band (usually band 45, or band 47 if band 45 is bad).
c
c  Usage:
c      RESULT = TEMBCK( BAND, BB1C, BB2C, HEADC, BB1T, BB2T )
c
c  Input:
c      BAND    MAS IR band number (26-50, 45 is the usual value)
c      BB1C    Digital counts for blackbody 1
c      BB2C    Digital counts for blackbody 2
c      HEADC   Digital counts from MAS head view
c      BB1T    Temperature for blackbody 1 (Kelvin)
c      BB2T    Temperature for blackbody 2 (Kelvin)
c
c  Output:
c      RESULT  MAS head temperature estimate (Kelvin)
c
c  Revised:
c      Liam.Gumley@ssec.wisc.edu
c      $Id: tembck.f,v 1.3 1999/11/11 20:04:46 gumley Exp $
c
c-----------------------------------------------------------------------

      real function tembck( band, bb1c, bb2c, headc, bb1t, bb2t )

      implicit none

c ... arguments

      integer band, bb1c, bb2c, headc
      real bb1t, bb2t

c ... local variables

      real rad1, rad2, slope, intercept, radh, masrad, masbrt
      external masrad, masbrt

c ... set default return value

      tembck = -1.0

c ... check input arguments

      if( band .lt. 26 .or. band .gt. 50 ) then
        write(*,*) 'Warning in function TEMBCK'
        write(*,*) 'Input argument BAND was out of the range [26,50]'
        write(*,*) 'Offending value was ', band
        write(*,*) 'Head temperature set to -1.0 in TEMBCK'
        return
      endif
```

```
 if( bb1c .le. 0 .or. bb1c .ge. 100000 ) then
   write(*,*) 'Warning in function TEMBCK'
   write(*,*) 'Input argument BB1C was out of the range [0,100000]'
   write(*,*) 'Offending value was ', bb1c
   write(*,*) 'Head temperature set to -1.0 in TEMBCK'
   return
 endif

 if( bb2c .le. 0 .or. bb2c .ge. 100000 ) then
   write(*,*) 'Warning in function TEMBCK'
   write(*,*) 'Input argument BB2C was out of the range [0,100000]'
   write(*,*) 'Offending value was ', bb2c
   write(*,*) 'Head temperature set to -1.0 in TEMBCK'
   return
 endif

 if( bb1c .ge. bb2c ) then
   write(*,*) 'Warning in function TEMBCK'
   write(*,*) 'Input argument BB1C was GE than BB2C'
   write(*,*) 'Offending values were ', bb1c, bb2c
   write(*,*) 'Head temperature set to -1.0 in TEMBCK'
   return
 endif

 if( headc .le. 0 .or. headc .ge. 100000 ) then
   write(*,*) 'Warning in function TEMBCK'
   write(*,*) 'Input argument HEADC was out of the range ' //
&    '[0,100000]'
   write(*,*) 'Offending value was ', headc
   write(*,*) 'Head temperature set to -1.0 in TEMBCK'
   return
 endif

 if( bb1t .le. 180.0 .or. bb1t .ge. 320.0 ) then
   write(*,*) 'Warning in function TEMBCK'
   write(*,*) 'Input argument BB1T was out of the range ' //
&    '[180.0,320.0]'
   write(*,*) 'Offending value was ', bb1t
   write(*,*) 'Head temperature set to -1.0 in TEMBCK'
   return
 endif

 if( bb2t .le. 180.0 .or. bb2t .ge. 320.0 ) then
   write(*,*) 'Warning in function TEMBCK'
   write(*,*) 'Input argument BB2T was out of the range ' //
&    '[180.0,320.0]'
   write(*,*) 'Offending value was ', bb2t
   write(*,*) 'Head temperature set to -1.0 in TEMBCK'
   return
 endif

 if( bb1t .ge. bb2t ) then
   write(*,*) 'Warning in function TEMBCK'
   write(*,*) 'Input argument BB1T was GE than BB2T'
   write(*,*) 'Offending values were ', bb1t, bb2t
   write(*,*) 'Head temperature set to -1.0 in TEMBCK'
   return
```

```
      endif

c ... initialize planck function temperature correction coefficients
c ... (must be done before calling MASRAD or MASBRT)

      call initcf()

c ... convert blackbody temperatures to radiance

      rad1 = masrad( band, bb1t )
      rad2 = masrad( band, bb2t )

c ... compute calibration slope and intercept

      slope = ( rad2 - rad1 ) / real( bb2c - bb1c )
      intercept = rad1 - slope * real( bb1c )

c ... compute head radiance and brightness temperature

      radh = slope * real( headc ) + intercept
      tembck = masbrt( band, radh )

      end
```

```
c------------------------------------------------------------------------
c
c  Name:
c      FUNCTION WBRITE
c
c  Purpose:
c      Compute brightness temperature in Kelvin given monochromatic
c      Planck radiance in milliWatts per square meter per steradian per
c      wavenumber.
c
c  Usage:
c      RESULT = WBRITE( WN, RAD )
c
c  Input:
c      WN      Wavenumber (inverse centimeters)
c      RAD     Planck radiance (milliWatts per square meter per
c              steradian per wavenumber)
c
c  Output:
c      RESULT  Brightness temperature (Kelvin)
c
c  Revised:
c      Liam.Gumley@ssec.wisc.edu
c      $Id: wbrite.f,v 1.3 1999/11/11 20:04:46 gumley Exp $
c
c------------------------------------------------------------------------

      real function wbrite( wn, rad )

      implicit none

c ... arguments

      real wn, rad

c ... local variables

      double precision h, c, b, c1, c2

c ... Planck function constants

      parameter( h = 6.6260755d-27, c = 2.9979246d+10,
     &  b = 1.380658d-16, c1 = 2.0d0 * h * c**2, c2 = h * c / b )

      wbrite = sngl( c2 * dble( wn ) /
     &  log( c1 * dble( wn )**3 / dble( rad ) + 1.0d0 ) )

      end
```

```
c------------------------------------------------------------------------
c
c  Name:
c      FUNCTION WPLANC
c
c  Purpose:
c      Compute monochromatic Planck radiance in milliWatts per square
c      meter per steradian per wavenumber given brightness temperature
c      in.
c
c  Usage:
c      RESULT = WBRITE( WN, TEM )
c
c  Input:
c      WN      Wavenumber (inverse centimeters)
c      TEM     Brightness temperature (Kelvin)
c
c  Output:
c      RESULT  Planck radiance (milliWatts per square meter per
c              steradian per wavenumber)
c
c  Revised:
c      Liam.Gumley@ssec.wisc.edu
c      $Id: wplanc.f,v 1.3 1999/11/11 20:04:47 gumley Exp $
c
c------------------------------------------------------------------------

      real function wplanc( wn, tem )

      implicit none

c ... arguments

      real wn, tem

c ... local variables

      double precision h, c, b, c1, c2

c ... Planck function constants

      parameter( h = 6.6260755d-27, c = 2.9979246d+10,
     &  b = 1.380658d-16, c1 = 2.0d0 * h * c**2, c2 = h * c / b )

      wplanc = sngl( c1 * dble( wn )**3 /
     &  ( exp( c2 * dble( wn ) / dble( tem ) ) - 1.0d0 ) )

      end
```

```
c ... a quick hack to test the MAS IR calibration routines

c ... Tree of subprogram calls:
c ... MAIN
c ...     TEMBCK
c ...         INITCF
c ...             GETCOF
c ...                 CENWAV
c ...                 HMWSRF
c ...                     GETLUN
c ...                     INTERP
c ...                 BNDFIT
c ...                     WBRITE
c ...                     WPLANC
c ...                     LINREG
c ...         MASRAD
c ...             PLANCK
c ...         MASBRT
c ...             BRIGHT
c ...     MASCAL
c ...         MASRAD (see above)
c ...         INITCF (see above)

      implicit none

      integer band, bb1c( 50 ), bb2c( 50 ), count( 50 ), headc
      real bb1t, bb2t, headt, slope, intercept, rad, brt, masbrt,
     &  temp( 50 ), tembck
      character answer*1
      external masbrt, tembck

c ... the following numbers are from Chris Moellers EMISCAL.f
c ... test input dataset

c ... blackbody 1 counts (for all 50 bands)

      data bb1c / 25 * 0,
     &    2119,  2066,  2359,  2163,  2342,  2042,  2491,  2213,  2140,
     &    2415,  2311,  2243,  2083,  2421,  2070,  2429,  2429,  2436,
     &    2422,  2400,  2436,  2356,  2199,  2297,  6492 /

c ... blackbody 2 counts

      data bb2c / 25 * 0,
     &    2175,  2169,  2619,  2646,  3174,  3312,  4291,  4646,  5322,
     &    6506,  7160,  7646,  8618,  9606,  9712,  9304,  8666, 16578,
     &   24144, 25486, 15082, 23782, 32745, 26221, 19857 /

c ... earth scene counts

      data count / 25 * 0,
     &    2163,  2134,  2511,  2534,  3091,  3149,  3520,  3971,  2454,
     &    2220,  3925,  6263,  6360,  6835,  6159,  4859,  8108, 13568,
     &   23243, 24666, 14549, 21436, 24512,  7883,  3373 /

c ... earth scene temperatures computed in Chris Moellers test program
```

```
      data temp / 25 * 0.0,
     &  291.198, 288.018, 285.583, 290.225, 293.159,
     &  292.414, 283.792, 288.217, 257.119, 234.307,
     &  272.331, 288.150, 285.046, 283.416, 280.148,
     &  271.592, 291.064, 285.888, 292.434, 292.600,
     &  292.241, 289.403, 282.485, 258.376, 230.289 /

c ... Data for computing head temperature

      band = 45
      headc = 10612
      bb1t = 243.70
      bb2t = 295.61

c ... Test error handling if required

      write(*,'(''Test error handling (y/n)'')')
      read(*,'(bn,a1)') answer
      if( answer .eq. 'y' .or. answer .eq. 'Y') then
        headc = -1
        bb2c( 31 ) = -1
      endif

c ... Estimate head temperature using band 45 head count,
c ... or band 47 head count if band 45 fails

      headt = tembck( band, bb1c( band ), bb2c( band ), headc, bb1t,
     &  bb2t )
      if( headt .eq. -1.0 ) then
        band = 47
        headc = 10200
        headt = tembck( band, bb1c( band ), bb2c( band ), headc, bb1t,
     &    bb2t )
      endif

      write(*,'(''BB1 TEMP (K) = '',f6.2)') bb1t
      write(*,'(''BB2 TEMP (K) = '',f6.2)') bb2t
      write(*,'(''BAND '',i2,'' HEAD COUNT    = '',i5)') band, headc
      write(*,'(''BAND '',i2,'' HEAD TEMP (K) = '',f6.2)') band, headt
      write(*,'(a)') 'BAND   BB1C   BB2C  COUNT    SLOPE       ' //
     &  'INTERCEPT   RADIANCE   TEMP    REFTEMP    DIFF'

c ... Compute earth scene temperature for each band

      do band = 26, 50

        call mascal( band, bb1c( band ), bb2c( band ), bb1t, bb2t,
     &    headt, slope, intercept )
        rad = slope * real( count( band ) ) + intercept
        brt = masbrt( band, rad )

        write(*,fmt='(1x,i2,1x,3i7,2x,e12.6,2f11.6,3f9.3)')
     &    band, bb1c( band ), bb2c( band ), count( band ),
     &    slope, intercept, rad, brt, temp( band ), brt - temp( band )

      end do
```

```
end
```