DAO Office Note 96–08

Office Note Series on Global Modeling and Data Assimilation

Richard B. Rood, Head
Data Assimilation Office
Goddard Space Flight Center
Greenbelt, Maryland

The GLA TOVS Rapid Algorithm Forward Radiance Modules and Jacobian Version 1.0

Meta Sienkiewicz

Data Assimilation Office, Goddard Laboratory for Atmospheres
General Sciences Corporation, Laurel, Maryland

Goddard Space Flight Center
Greenbelt, Maryland 20771
September 1996

Abstract

This document describes the rapid transmittance algorithm and forward radiative transfer calculations used in the Goddard Laboratory for Atmospheres (GLA) TOVS processing and the forward model and Jacobian program modules that have been derived from the TOVS processing routines for use by the Data Assimilation Office. An overview of the physical basis of the rapid algorithm is given, followed by descriptions of the routines for forward radiance calculation and routines for Jacobian calculation (derivative of the forward model with respect to input parameters). The source code for the modules is listed in the Appendix.

An on-line version of this document can be obtained from

`ftp://dao.gsfc.nasa.gov/pub/office_notes/on9608.ps.Z (postscript)`

Also, see the Data Assimilation Office's Home Page at

`http://dao.gsfc.nasa.gov/`

Revision of September 5, 1996

Contents

List of Tables

List of Symbols

ν frequency (cm^{-1} or GHz)

$\phi_i(\nu)$ filter response function as a function of frequency for channel i

$\tau_\nu(p, \theta)$ transmittance, see (3)

$\tau_{iS}(\theta, \theta_S)$ atmospheric transmittance for solar radiation term

θ zenith angle

Θ_i brightness temperature, see , page 11

ε_ν surface emissivity

φ solar zenith angle

$B_\nu(T)$ Planck function, see (2)

H_i solar radiation at the top of atmosphere

$k_\nu(P)$ absorption per unit pressure

R_ν radiance, see (1)

R_i radiance for channel i, see (5)

$R_i\downarrow$ effective downward flux of radiance from atmospheric emission

T_g surface skin (ground) temperature

T_{BB} cosmic background brightness temperature

$R_i'(\theta)$ radiance reflected from surface, see (6)

1 Introduction

One of the goals of the Data Assimilation Office (DAO) is to make better use of observations from new instruments, such as satellite-based sensors, which do not directly measure traditional atmospheric variables such as temperature and moisture. Formerly, height profiles (retrieved from satellite radiance measurements) were used in our data assimilation system in an *ad hoc* fashion without regard to the error covariance structure of the retrievals. New assimilation methodologies being evaluated by the DAO for using satellite measurements require knowledge of the forward radiative transfer model and its Jacobian (derivative with respect to input parameters). To this end, the code for the radiative transfer forward model for the TOVS (TIROS Operational Vertical Sounder) was extracted from the Goddard Laboratory for Atmospheres (GLA) TOVS retrieval program used by the Satellite Data Utilization Office[1] to process the TOVS Pathfinder Path A retrievals. This code was used to write a new set of Fortran modules for the forward model and corresponding Jacobian, which will be incorporated in future DAO assimilation systems.

The purpose of this office note is to document the program modules for the TOVS forward model and Jacobian. Section 2 describes the physics used in the forward model. In Section 3 the finite-difference representations used in the code are reviewed. These first sections, which give the scientific and mathematical background for the forward model and Jacobian, will be of particular interest to users who would wish to modify the code or to implement similar code using other forward model parameterizations. General users (those who wish to calculate TOVS brightness temperatures or Jacobians from input profiles of temperature, moisture and ozone) will find the description of the program interface in the latter sections useful: in section 4 the forward model program modules are discussed, and section 5 discusses the calculation of the Jacobian. Methods which use the forward radiative transfer model and Jacobian for assimilation of satellite measurements are discussed in Office Note 96–06 (Joiner and daSilva 1996).

2 Theoretical development

Physically-based retrievals and radiance assimilation methods use forward radiance calculations, *i.e.* calculation of radiance that would be measured in a satellite channel under given atmospheric conditions. The most accurate forward problem calculations are performed via *line-by-line* calculations to obtain high-resolution radiance spectra, which are convolved with the instrument response function to obtain channel radiances. (See section 2.2). Use of such computationally intensive calculations is not practical for operational retrievals or for data assimilation. Thus, rapid algorithms for forward calculation have been developed which use simple models, with coefficients fitted to results of line-by-line caluculation with representative atmospheric profiles.

The methodology for the forward calculation of radiance and brightness temperature using the GLA TOVS rapid algorithm is given in Susskind, *et al.* 1983, 1984 and references contained therein. A summary of the main points presented in those papers is given here. The purpose is to give users of the TOVS modules an idea of the physics underlying the code and to bring together information about the GLA forward model (as implemented in these modules) into one convenient location for easy reference.

[1] now the GLA Sounder Research Team

2.1 Forward radiance calculation

The basic radiative transfer equation for radiation emitted at the top of the atmosphere at a single frequency ν can be written

$$R_\nu(\theta) = \varepsilon_\nu(\theta) B_\nu(T_g) \tau_\nu(p_s, \theta) + \int_{p_s}^{0} B_\nu[T(p)] \frac{\partial \tau_\nu(p, \theta)}{\partial p} dp + R_\nu'(\theta) \tag{1}$$

where $\varepsilon_\nu(\theta)$ is the surface emissivity at zenith angle θ and frequency ν, T_g and $T(p)$ are surface skin (ground) temperature and temperature at pressure level p respectively, p_s is surface pressure and $R_\nu'(\theta)$ is the contribution from radiance reflected from the surface (discussed in section 2.3).

$B_\nu(T)$ is the Planck function or emitted radiance from a blackbody at temperature T for frequency ν under conditions of local thermodynamic equilibrium:

$$B_\nu(T) = \frac{2h\nu^3}{c^2(e^{\frac{h\nu}{kT}} - 1)} \tag{2}$$

where h is Planck's constant, c is the speed of light, and k is Boltzmann's constant. For microwave frequencies, blackbody radiation varies nearly linearly with temperature. Also, microwave measurements are expressed in terms of brightness temperatures, so $B(T)$ is replaced by temperature T in the radiative transfer calculations.

The transmittance $\tau_\nu(p, \theta)$ from pressure p to the top of the atmosphere at zenith angle θ is given by

$$\tau_\nu(p, \theta) = \exp\left[-\int_0^p k_\nu(P) \sec(\theta) dP\right] \tag{3}$$

where $k_\nu(P)$ is the atmospheric absorption per unit pressure at frequency ν and pressure p.

2.2 Calculations for satellite channels

Each satellite channel represents an observation taken over a range of frequencies

$$R_i(\theta) = \int_\nu \phi_i(\nu) R_\nu(\theta) d\nu \tag{4}$$

where $\phi_i(\nu)$ is the filter response function for satellite channel i. For the GLA rapid algorithm Susskind *et al.* 1983 write this in a form similar to (1):

$$R_i(\theta) = \varepsilon_i(\theta) B_i(T_g) \tau_i(p_s, \theta) + \int_{p_s}^{0} B_i[T(p)] \frac{\partial \tau_i(p, \theta)}{\partial p} dp + R_i'(\theta) \tag{5}$$

where now $B_i(T) = B_{\nu_i}(T)$, $\varepsilon_i(\theta) = \varepsilon_{\nu_i}(\theta)$ are evaluated at a prescribed central channel frequency ν_i and the transmittance τ_i is a modeled transmittance fitted to channel-averaged transmittances obtained from line-by-line calculations. Although the Planck function $B_{\nu_i}(T)$ changes with frequency ν it is assumed that the width of the channel response function ϕ is small enough that the Planck function at the central frequency ν_i can be used without modification.

2.3 Reflected radiance from surface

Susskind, *et al.* 1984 describe $R_i'(\theta)$, the contribution from reflected radiance, as:

$$R_i'(\theta) = (1 - \varepsilon_i)\, R_i{\downarrow}\, \tau_i(p_s, \theta) + \rho_i H_i \tau_{iS}(\theta, \theta_S) \tag{6}$$

where $R_i{\downarrow}$ is the effective downward flux of radiation from atmospheric emission for channel i, ρ_i is the bidirectional reflectance of the surface to the satellite of solar radiation from the sun with zenith angle θ_S, H_i is the solar radiation at the top of the atmosphere, and $\tau_{iS}(\theta, \theta_S)$ is the atmospheric transmittance along the entire path of incident and reflected solar radiation. The first term in the above equation represents the contribution of downwelling atmospheric radiation, and the second term represents the contribution of reflected solar radiation. The second term is calculated only for HIRS channels with frequencies greater than 2000 cm^{-1} (HIRS 13 - 19), for solar zenith angles less than 85°.

For the first term, there are several ways that the effective downward flux is calculated, depending on whether the channel is an infrared (IR) or microwave (MW) channel. For HIRS IR channels 8,10,18,19 where the atmosphere is optically thin, the effective downward flux $R_i{\downarrow}$ is calculated as:

$$R_i{\downarrow} = 2\cos\theta \int_0^{\tau_i(p_s)} B_i(T)\, d\tau \tag{7}$$

As Susskind, *et al.* 1984 explain, this equation is based on the assumption of (a) an optically thin atmosphere, in which the downward flux at any zenith angle χ is proportional to $\sec(\chi)$, and (b) a Lambertian surface, for which the angular reflectance at any angle χ is a constant given by $(1 - \varepsilon)/\pi$.

For the other HIRS infrared channels, the effective downward flux $R_i{\downarrow}$ is modeled according to Kornfield and Susskind (1977):

$$R_i{\downarrow} = F_i B_i(T_s)(1 - \tau_i(p_s, \theta)) \tag{8}$$

where T_s is the surface air temperature and F_i is a channel-dependent constant.

For the MSU microwave channels, the downward flux $R_i{\downarrow}$ is calculated explicitly:

$$R_i{\downarrow} = T_{BB}\tau_i(p_s, \theta) + \int_{p_s}^{0} T(p) \frac{\partial \tau_i(p, p_s, \theta)}{\partial p} dp \tag{9}$$

where T_{BB} is the background radiation from space from the big bang, and $\tau_i(p, p_s, \theta)$ is the (downward) transmittance between pressure level p and the surface. The downward transmittance can be related to the transmittance upward to the satellite as:

$$\tau_i(p, p_s, \theta) = \tau_i(p_s, \theta)/\tau_i(p, \theta) \tag{10}$$

which enables us to write (9) as:

$$R_i{\downarrow} = T_{BB}\tau_i(p_s, \theta) + \tau_i(p_s, \theta) \int_{p_s}^{0} T(p) \frac{\partial}{\partial p} \frac{1}{\tau_i(p, \theta)} dp \tag{11}$$

Strictly speaking, the relation in (10) is applicable only to monochromatic radiances; but as Weinreb *et al.* 1981 explain the microwave response functions are narrow and the transmittance across the channel varies slowly with frequency, so this formula may be sufficiently accurate for use.

2.4 Rapid algorithm for transmittance

The rapid transmittance algorithm has the form

$$\tau_i(p_l,\theta) = \prod_{j=1}^{l} \tilde{\tau}_i(p_j,p_{j-1},\theta) \tag{12}$$

$$= \prod_{j=1}^{l} \tilde{\tau}_{iF}(p_j,p_{j-1},\theta)\tilde{\tau}_{iO}(p_j,p_{j-1},\theta)\tilde{\tau}_{iW}(p_j,p_{j-1},\theta) \tag{13}$$

The total transmittance $\tau_i(p_l,\theta)$ between satellite level and a level l, is the product of effective layer transmittances $\tilde{\tau}_i(p_j,p_{j-1},\theta)$ for all layers j between satellite level and the level l. These effective layer transmittances are modeled as the product of terms representing absorption due to atmospheric gases having a fixed mixing ratio ($\tilde{\tau}_{iF}(p_j,p_{j-1},\theta)$), absorption due to ozone ($\tilde{\tau}_{iO}(p_j,p_{j-1},\theta)$) and absorption due to water vapor ($\tilde{\tau}_{iW}(p_j,p_{j-1},\theta)$). Coefficients for these terms are obtained by fitting transmittances obtained from line-by-line calculations. Additional factors for the water vapor continuum absorption and aerosol scattering and absorption (not included in the line-by-line calculations) are also applied; these are presented at the end of this discussion.

As explained in Susskind *et al.* 1983, $\tilde{\tau}_{iO}(p_j,p_{j-1},\theta)$ and $\tilde{\tau}_{iW}(p_j,p_{j-1},\theta)$ are not single species layer transmittances calculated using ozone absorption or water vapor absorption alone. Since the channel radiance measurements are not monochromatic a simple product relation for transmittances does not apply. Given line-by-line transmittance calculations for $\tau_{iF}(p_l,\theta)$ (fixed gases only), $\tau_{iFO}(p_l,\theta)$ (fixed gases plus ozone), and $\tau_{iFOW}(p_l,\theta)$ (fixed gases, ozone, and water vapor), the effective layer transmittances are calculated according to

$$\tilde{\tau}_i(p_j,p_{j-1},\theta) \equiv \frac{\tau_i(p_j,\theta)}{\tau_i(p_{j-1},\theta)} \tag{14}$$

$$\tilde{\tau}_{iO}(p_j,p_{j-1},\theta) \equiv \frac{\tilde{\tau}_{iFO}(p_j,p_{j-1},\theta)}{\tilde{\tau}_{iF}(p_j,p_{j-1},\theta)} \tag{15}$$

$$\tilde{\tau}_{iW}(p_j,p_{j-1},\theta) \equiv \frac{\tilde{\tau}_{iFOW}(p_j,p_{j-1},\theta)}{\tilde{\tau}_{iFO}(p_j,p_{j-1},\theta)} \tag{16}$$

The next two sections describe the models used for the effective mean layer transmittances in the GLA TOVS rapid algorithm. The values of the coefficients used with these models are derived by least-squares fitting to transmittances from line-by-line calculations performed using representative atmospheric profiles. The coefficients are provided for 71 layers in the vertical (see Table 1 for pressure levels bounding the layers) and at three zenith angles (0°, 50°, and 75°); values at other zenith angles are obtained by linear interpolation with respect to $\sec(\theta)$:

$$\tilde{\tau}_i(p_j,p_{j-1},\theta) = \tilde{\tau}_i(p_j,p_{j-1},\theta_1) + \frac{\sec(\theta)-\sec(\theta_1)}{\sec(\theta_2)-\sec(\theta_1)}(\tilde{\tau}_i(p_j,p_{j-1},\theta_2) - \tilde{\tau}_i(p_j,p_{j-1},\theta_1)) \tag{17}$$

where θ_1 and θ_2 are the rapid algorithm angles bracketing the zenith angle θ.

Table 1: GLA algorithm pressures, standard temperatures and scattering coefficients

Pressure (mb)	Temperature (K)	Scattering Coefficient	Pressure (mb)	Temperature (K)	Scattering Coefficient
0.10	221.6	0.000	240.0	222.9	0.00300
0.40	254.8	0.000	260.0	226.2	0.00300
0.70	264.4	0.000	280.0	229.3	0.00300
1.00	266.5	0.000	300.0	232.1	0.00300
1.30	265.0	0.000	320.0	235.2	0.00300
1.70	261.8	0.000	340.0	238.5	0.00300
2.00	259.7	0.000	360.0	241.6	0.00300
3.00	251.4	0.000	380.0	244.5	0.00300
4.00	246.1	0.000	400.0	247.3	0.00300
5.00	242.2	0.000	425.0	250.2	0.00300
6.00	239.2	0.000	450.0	253.3	0.00300
7.00	236.6	0.000	475.0	256.3	0.00300
8.00	234.4	0.000	500.0	259.0	0.00306
9.00	232.5	0.000	525.0	261.6	0.00312
10.0	230.8	0.000	550.0	263.9	0.00318
15.0	228.0	0.000	575.0	266.1	0.00325
20.0	224.5	0.000	600.0	268.2	0.00331
30.0	221.0	0.000	625.0	270.3	0.00340
40.0	217.0	0.000	650.0	272.2	0.00354
50.0	213.5	0.000	675.0	274.1	0.00368
60.0	210.6	0.00022	700.0	275.9	0.00381
70.0	208.1	0.00058	725.0	277.5	0.00409
80.0	206.8	0.00091	750.0	278.9	0.00438
90.0	206.5	0.00121	775.0	280.3	0.00446
100.0	206.1	0.00143	800.0	281.6	0.00501
110.0	206.6	0.00167	825.0	282.9	0.00560
120.0	207.9	0.00186	850.0	284.2	0.00619
130.0	209.0	0.00207	875.0	285.3	0.00678
140.0	210.1	0.00222	900.0	286.3	0.00741
150.0	211.0	0.00238	925.0	287.2	0.00864
160.0	212.2	0.00254	950.0	288.2	0.00988
170.0	213.4	0.00271	975.0	289.1	0.01112
180.0	214.7	0.00292	1000.0	290.0	0.01236
190.0	215.8	0.00300	1025.0	290.8	0.01359
200.0	216.9	0.00300	1050.0	291.7	0.01483
220.0	219.3	0.00300			

2.4.1 Ozone and water vapor contributions

The effective layer transmittances due to ozone and those due to water vapor are both modelled using an equation of this form:

$$\tilde{\tau}_{ic}(p_j, p_{j-1}, \theta) = \exp\left(-A_{i,j,c}(\theta)[1 - B_{i,c}(\bar{T}_j - 273)]u_c(j, j-1)^{N_{i,c}}\right) \tag{18}$$

where c is the constituent (either ozone or water vapor), $u_c(j, j-1)$ is the integrated column density of the species in the layer between levels j and $j-1$, $N_{i,c}$ is a channel and species dependent constant between 0.5 and 1, $A_{i,j,c}$ is an effective channel, species, pressure, and angle dependent absorption coefficient, $B_{i,c}$ is a channel and species dependent constant representing percent change per degree, and $\bar{T}_j$ is the mean temperature in the layer between p_{j-1} and p_j.

Susskind *et al.* 1983 give the rationale for this formulation as follows: the effective layer transmittances $\tilde{\tau}_{iO}$ and $\tilde{\tau}_{iW}$ can be treated as having the transmittance properties of a gas in a homogeneous layer with mean temperature T, and pressure p of the atmospheric layer, and vertical column density u of the absorbing gas in the layer. The use of (14) removes most of the dependence of the mean layer transmittance on the properties of the atmosphere above the layer. Thus we would expect that the log of the mean layer transmittances would be proportional to u for weakly absorbing lines and $u^{1/2}$ for strong lines, with an exponent of intermediate value applicable to a composite of lines.

2.4.2 Fixed gas contribution

The definition of effective mean layer transmittances in (14) reduces the dependence of the layer transmittances on the profile above the layer, but this effect still must be taken into account in the model for fixed gas contribution. The transmittance in layer j for two profiles having the same layer temperature T_j but different temperature profiles above p_j will still differ because the profile with more attenuation above the level will have less effective attenuation within the level. Susskind *et al.* 1983 note that temperatures in regions of the atmosphere which do not contribute to the channel radiance are irrelevant – either no attenuation is taking place in those levels or the total transmittance is already small enough that second-order effects are not important. Thus, they define an effective mean temperature above pressure p_j as the average temperature weighted by the weighting function for each channel i:

$$\tilde{T}_{ij}(\theta) = \frac{1}{1 - \tau_i^\circ(p_j, \theta)} \int_0^{p_j} T(p) \frac{\partial \tau_i^\circ(p_j, \theta)}{\partial p} dp \tag{19}$$

where $\tau_i^\circ(p_j, \theta)$ is the transmittance of channel i for the standard temperature profile.

The effective mean layer transmittance is modeled according to

$$\tilde{\tau}_{iF}(p_j, p_{j-1}, \theta) = A_{ij}(\theta) + B_{ij}(\theta)(\bar{T}_j - \bar{T}_j^\circ) + C_{ij}(\theta)(\tilde{T}_{ij}(\theta) - \tilde{T}_{ij}^\circ(\theta)) \tag{20}$$

where $\bar{T}_j$ is the mean temperature in layer j between p_j and p_{j-1} for the given profile, $\bar{T}_j^\circ$ is the mean layer temperature for a standard temperature profile for the layer j and $\tilde{T}_{ij}$ and $\tilde{T}_{ij}^\circ$ are the effective mean layer temperatures as defined in (19) for the given temperature profile and for the standard temperature profile, respectively.

2.4.3 Water vapor continuum

We now discuss the transmittance terms that are not included in the line-by-line calculations; the water vapor continuum and aerosol scattering terms. The water vapor continuum term is modeled as described in Susskind and Searl (1978), based on a representation given by Bignell (1970). The net atmospheric absorption in layer j due to the e–type component of the water vapor continuum is written

$$K_{H_2O}(\nu, j, j-1) = k(\nu, \bar{T}_j)\bar{e}_j \hat{u}_{H_2O}(j, j-1) \tag{21}$$

where $k(\nu, \bar{T}_j)$ is the absorption coefficient (g^{-1} cm^2 atm^{-1}) for the e–type H_2O continuum component for layer j, $\bar{e}_j$ is the mean H_2O partial pressure for layer j, and $\hat{u}_{H_2O}(j, j-1)$ is the H_2O column density (g cm^{-1}). The coefficient $k(\nu, \bar{T}_j)$ is assumed to vary linearly with temperature.

2.4.4 Scattering and absorption by aerosols

The term for aerosol scattering and absorption used with the GLA rapid transmittance algorithm is written as (Susskind *et al.* 1983):

$$\tau_{a_i}(p, \theta) = \exp(-k_{a_i}(p) \sec\theta) \tag{22}$$

where $k_{a_i}(p)$ is the sum of the aerosol scattering and aerosol absorption optical thickness from pressure p to the top of the atmosphere for the given channel i. For the 15-μm channels, the total optical depth of the tropospheric aerosols is taken as 0.01 for nadir viewing, falling off with a 1.2-km scale height. There is an additional stratospheric aerosol layer between 12 and 20 km with a total optical depth 30% of that of the tropospheric layer. For the 4.3-μm channels, the optical depths are taken to be a factor of 3 greater than the 15-μm channels (*i.e.* a total nadir optical depth of 0.039). The values of scattering coefficients used are listed in Table 1.

3 Finite-difference representations

The purpose of this section is to relate the physical background of the forward model (as discussed in Section 2 to the finite difference representations that are used in the code for the forward radiative transfer calculation. From the finite-difference forward model representation we can perform derivative operations on the forward model to obtain the Jacobians for temperature, moisture, and ozone.

3.1 Finite-difference forward calculation

The description of the finite-difference forward model calculation in this section will follow the same steps as in the forward module code. Effective layer transmittances are calculated at the rapid algorithm zenith angles and interpolated to the satellite zenith angle. The product of the layer transmittances is taken to obtain the transmittance between the satellite and the rapid algorithm levels. The radiance corresponding to the input profiles is then

calculated by taking vertical integral of the Planck function weighted by the derivative of the transmittance.

The notation convention used in this section is relatively simple. The subscripts of terms are ordered as (channel, level or layer, angle) if applicable. The channel index is usually i, and the angle index is k. Level indices used include j, l, and m. Thus, T_m is temperature at pressure p_m, $\bar{T}_m$ is mean layer temperature for the layer m between p_{m-1} and p_m, $\tilde{\tau}_{i,j,k}$ is the effective layer transmittance for channel i, layer j, and rapid algorithm zenith angle θ_k, and $\tau_{i,l}(\theta)$ is transmittance for channel i between the satellite and pressure level l for zenith angle θ.

Fixed gas contribution In the forward model code, we first calculate the contributions to effective layer transmittance due to fixed gases, water vapor, and ozone. We consider the fixed gas contribution first. The expression for effective mean temperature defined in (19) can be written in finite difference form as:

$$\tilde{T}_{i,j,k} = \frac{1}{1 - \tau^{\circ}_{i,j,k}} \sum_{m=1}^{j} \bar{T}_m \left(\tau^{\circ}_{i,m-1,k} - \tau^{\circ}_{i,m,k} \right). \tag{23}$$

where $\tilde{T}_{i,j,k}$ is the effective mean temperature for channel i at pressure level j for rapid algorithm zenith angle k. The transmittance for the standard temperature profile $\tau^{\circ}_{i,j,k}$ is derived from (13) and (20):

$$\tau^{\circ}_{i,j,k} = \prod_{l=1}^{j} \tilde{\tau}_{F_{i,l,k}} = \prod_{l=1}^{j} A_{i,l.k} \tag{24}$$

This can be substituted into (23) to get

$$\begin{aligned} \tilde{T}_{i,j,k} &= \frac{1}{1 - \prod_{l=1}^{j} A_{i,l,k}} \sum_{m=1}^{j} \bar{T}_m (1 - A_{i,m,k}) \prod_{l=1}^{m-1} A_{i,l,k} \\ &= \frac{1}{1 - \prod_{l=1}^{j} A_{i,l,k}} \sum_{m=1}^{j} W_{i,m,k} \bar{T}_m \end{aligned} \tag{25}$$

where $W_{i,m,k} = (1 - A_{i,m,k}) \prod_{l=1}^{m-1} A_{i,l,k}$ is a coefficient which combines the constant terms of the vertical integration. Then (20) can be rewritten as

$$\begin{aligned} \tilde{\tau}_{F_{i,j,k}} &= A_{i,j,k} + B_{i,j,k}(\bar{T}_j - \bar{T}^{\circ}_j) + \frac{C_{i,j,k}}{1 - \prod_{l=1}^{j} A_{i,l,k}} \sum_{m=1}^{j} W_{i,m,k} \left(\bar{T}_m - \bar{T}^{\circ}_m \right) \\ &= A_{i,j,k} + B_{i,j,k}(\bar{T}_j - \bar{T}^{\circ}_j) + C^{*}_{i,j,k} \sum_{m=1}^{j} W_{i,m,k} \left(\bar{T}_m - \bar{T}^{\circ}_m \right) \end{aligned} \tag{26}$$

where we now define a new coefficient $C^{*}_{i,j,k} = \frac{C_{i,j,k}}{1 - \prod_{l=1}^{j} A_{i,l,k}}$.

Water vapor contribution The water vapor contribution to transmittance is made up of two terms, the modeled water vapor transmittance from (18) fitted from line-by-line transmittance calculations, and the water vapor continuum term given in (21). Since the coefficient $k(\nu, \bar{T}_j)$ is assumed to be a linear function of temperature we write the absorption for channel i in layer (p_{j-1}, p_j) as

$$K_{H_2O_{i,j}} = k_i^\circ \left(1 + \frac{dk}{dT}(\bar{T}_j - T^c)\right) \bar{e}_j \hat{u}_{H_2O_j} \tag{27}$$

The contribution to transmittance from the water vapor continuum term would then be

$$\tilde{\tau}_{\text{cont}_{i,j,k}} = \exp(-\sec(\theta_k) K_{H_2O_{i,j}}) \tag{28}$$

The input water vapor profile is in terms of specific humidity (kg H_2O/kg air) so conversions to mean layer vapor pressure $\bar{e}$ (atm) and column density $\hat{u}$ (g cm^{-2}) must be performed. For column density, the conversion is:

$$\hat{u}_{H_2O_j} = \rho_{H_2O_j} \Delta z_j = \rho_{\text{air}_j} q_j \Delta z_j = \frac{10 \Delta p_j}{g} q_j = \frac{10(p_{j-1} - p_j)}{g} q_j \tag{29}$$

For H_2O vapor pressure the conversion is:

$$\begin{aligned} \bar{e}_j &= \frac{\text{air mol. wt}}{H_2O \text{ mol. wt.}} \frac{\bar{p}_j(\text{mb})}{1013.25 \frac{\text{mb}}{\text{atm}}} q_j \\ &= \frac{1}{\epsilon} \frac{\bar{p}_j(\text{mb})}{1013.25 \frac{\text{mb}}{\text{atm}}} q_j \end{aligned} \tag{30}$$

For the transmittance modeled from line-by-line calculations (18), the coefficients are adjusted on input to be used with water vapor column density. We may write

$$\tilde{\tau}_{W_{i,j,k}} = \exp\left(-D_{i,j,k}[1 - E_{i,k}(\bar{T}_j - 273)]\{(\hat{u}_{H_2O_j})^{M_i}\}\right) \tag{31}$$

and this may be combined with the water vapor continuum term to obtain a term for the total water vapor contribution to effective layer transmittance

$$\begin{aligned} \tilde{\tau}_{H_2O_{i,j,k}} &= \tilde{\tau}_{W_{i,j,k}} \tilde{\tau}_{\text{cont}_{i,j,k}} \\ &= \exp\left(-D_{i,j,k}[1 - E_{i,k}(\bar{T}_j - 273)]\{(\hat{u}_{H_2O_j})^{M_i}\} - \sec(\theta_k) K_{H_2O_{i,j}}\right) \end{aligned} \tag{32}$$

Ozone contribution For the ozone contribution the only factor is from the modeled transmittance (18). Since the coefficients for ozone are adjusted to be used with the ozone column density input to the routine, no conversions are necessary. Thus, we may write

$$\tilde{\tau}_{O_{3_{i,j,k}}} = \exp\left(-F_{i,j,k}[1 - G_{i,k}(\bar{T}_j - 273)]\{(\hat{u}_{O_{3_j}})^{N_i}\}\right) \tag{33}$$

Total transmittance When the contributions to effective layer transmittance from the fixed gases, water vapor, and ozone have been calculated, the effective layer transmittance is formed from their product:

$$\tilde{\tau}_{i,j,k} = \tilde{\tau}_{F_{i,j,k}} \tilde{\tau}_{H_2O_{i,j,k}} \tilde{\tau}_{O_{3_{i,j,k}}} \tag{34}$$

The effective layer transmittances for the rapid algorithm angles that bracket the satellite zenith angle are interpolated linearly with respect to $\sec(\theta)$ as shown in (17). The product of the effective layer transmittances gives the total transmittance between the satellite and the rapid algorithm levels. For infrared channels, this is also multiplied by a factor $\tau_{a_{i,j}}$ (from (22)) to account for scattering. Thus, the total transmittance is:

$$\begin{aligned}\tau_{i,j}(\theta) &= \tau_{a_{i,j}}(\theta) \prod_{l=1}^{j} \tilde{\tau}_{i,l}(\theta) \\ &= \tau_{a_{i,j}}(\theta) \prod_{l=1}^{j} \left\{ \tilde{\tau}_{i,l,k} + \frac{\sec(\theta) - \sec(\theta_k)}{\sec(\theta_{k+1}) - \sec(\theta_k)} (\tilde{\tau}_{i,l,k+1} - \tilde{\tau}_{i,l,k}) \right\}\end{aligned} \tag{35}$$

where θ_k and θ_{k+1} are the rapid algorithm angles bracketing the desired angle θ.

The transmittance for solar radiation reflected from the surface is calculated in the same way as the total transmittance in (35) except that transmittance is interpolated to the effective solar zenith angle $\theta_{EFF} = \sec^{-1}(\sec(\varphi) + \sec(\theta))$ (where φ is the solar zenith angle). Thus

$$\tau_{iS}(\theta_{EFF}) = \tau_{a_{i,N}}(\theta_{EFF}) \prod_{l=1}^{N} \left\{ \tilde{\tau}_{i,l,k} + \frac{\sec(\theta_{EFF}) - \sec(\theta_k)}{\sec(\theta_{k+1}) - \sec(\theta_k)} (\tilde{\tau}_{i,l,k+1} - \tilde{\tau}_{i,l,k}) \right\} \tag{36}$$

Forward radiance calculation The forward radiance calculation from (5) can be written in finite difference form as

$$R_i(\theta) = \varepsilon_i(\theta) B_i(T_g) \tau_{i,N}(\theta) + \sum_{l=1}^{N} B_i(\bar{T}_l) \left(\tau_{i,l-1}(\theta) - \tau_{i,l}(\theta)\right) + R_i'(\theta) \tag{37}$$

where ε_i is the surface emissivity for channel i and T_g is the surface skin temperature. Note again that for microwave channels we use $\bar{T}_l$ in place of $B_i(\bar{T}_l)$

The expressions for reflected radiance $R_i'(\theta)$ are discussed in section 2.3. The basic equation (6) expressed this as the sum of terms for contribution from the downward flux of radiation from the atmosphere and a solar term (where appropriate).

$$R_i'(\theta) = (1 - \varepsilon_i)\, R_i{\downarrow}\, \tau_{i,N}(\theta) + \rho_i H_i \tau_{iS}(\theta_{EFF}) \tag{38}$$

The method for calculating the effective downward flux of radiation $R_i{\downarrow}$ is channel dependent, so there are three expressions to be written in finite-difference form. For the microwave channels, the finite-difference representation of (11) is

$$R_i{\downarrow} = T_{BB} \tau_{i,N}(\theta) + \tau_{i,N}(\theta) \sum_{l=1}^{N} \bar{T}_l \left(\frac{1}{\tau_{i,l}(\theta)} - \frac{1}{\tau_{i,l-1}(\theta)} \right) \tag{39}$$

For IR channels where the atmosphere is optically thin, we follow (7) and write

$$R_i{\downarrow} = 2 \cos\theta \sum_{l=1}^{N} B_i(\bar{T}_l) \left(\tau_{i,l-1}(\theta) - \tau_{i,l}(\theta)\right) \tag{40}$$

Other IR channels use the modeled downward flux as in (8)

$$R_i\downarrow = F_i B_i(T_s)(1 - \tau_{iN}(\theta)) \tag{41}$$

where T_s is the surface air temperature.

For the solar contribution term, the bi-directional reflectance ρ_i is taken as an input parameter since it (along with cloud-cleared radiances) is available from the level 2 Pathfinder data sets. The solar radiation H_i is

$$H_i = \pi \left(\frac{r_{\text{sun}}}{R_e} \right)^2 \cos(\varphi) B_i(5600\text{K}) \tag{42}$$

where r_{sun} is the Sun's radius, R_e is the radius of Earth's orbit and φ is the solar zenith angle.

3.2 Finite-difference Jacobian calculation

The calculation of the Jacobian, or derivative of brightness temperature with respect to input parameters is accomplished through operation on the finite-difference representation of the forward problem as expressed in the Fortran code. For our current work, we are only interested in the dependence of brightness temperature on changes in the atmospheric profiles of temperature, moisture and ozone, so derivatives with respect to other parameters (*e.g.* surface and skin temperature, surface emissivity, reflectance, *etc.*) will not be considered.

This discussion begins with derivation of a few general expressions needed for the Jacobian calculation. Then, the derivative of the atmospheric term (vertical summation) with respect to ozone and water vapor is obtained. The derivative for temperature is discussed separately since its form is different from the ozone and water vapor derivative. We finish by discussing the derivative of the reflected radiance term, which makes use of terms derived in earlier parts of this section.

General expressions The expressions needed for the Jacobian calculation can be derived by repeated applications of the chain rule for derivatives. Brightness temperature is defined as $\Theta_i = B^{-1}(R_i)$, where B^{-1} is the inverse Planck function. Thus, for some input quantity Q:

$$\frac{\partial \Theta}{\partial Q} = \frac{\partial \Theta_i}{\partial R_i} \frac{\partial R_i}{\partial Q} \tag{43}$$

$$= \frac{\partial \Theta_i}{\partial R_i} \frac{\partial}{\partial Q} \left[\varepsilon_i(\theta) B_i(T_g) \tau_{i,N}(\theta) + \sum_{l=1}^{N} B_i(\bar{T}_l) \left(\tau_{i,l-1}(\theta) - \tau_{i,l}(\theta) \right) + R_i'(\theta) \right] \tag{44}$$

$$= \frac{\partial \Theta_i}{\partial R_i} \left[\frac{\partial}{\partial Q} \{ \varepsilon_i(\theta) B_i(T_g) \tau_{i,N}(\theta) \} + \right.$$
$$\left. \frac{\partial}{\partial Q} \sum_{l=1}^{N} B_i(\bar{T}_l) \left(\tau_{i,l-1}(\theta) - \tau_{i,l}(\theta) \right) + \frac{\partial R_i'(\theta)}{\partial Q} \right] \tag{45}$$

The derivative of the Planck function with respect to temperature is

$$\frac{\partial B(T)}{\partial T} = \frac{2h\nu^3}{c^2(e^{\frac{h\nu}{kT}}-1)^2} e^{\frac{h\nu}{kT}} \frac{h\nu}{kT^2} \tag{46}$$

At brightness temperature Θ_i we can write the derivative $\partial\Theta_i/\partial R_i$ as

$$\frac{\partial \Theta_i}{\partial R_i} = \frac{\partial}{\partial R} B^{-1}(R_i) = \left[\frac{\partial B_i(T)}{\partial T}\right]^{-1}_{T=\Theta} \tag{47}$$

Water vapor and ozone derivatives The next step in obtaining the Jacobian is to find the derivative of the term in the vertical sum, which is the product of brightness temperature and transmittance. If Q_m is water vapor or ozone at a particular level m, we need only consider the transmittance derivatives. From (45) we may write:

$$\frac{\partial \Theta_i}{\partial Q_m} = \frac{\partial \Theta_i}{\partial R_i}\left[\varepsilon_i B_i(T_g)\frac{\partial \tau_{i,N}(\theta)}{\partial Q_m} + \sum_{l=1}^{N} B_i(\bar{T}_l)\frac{\partial}{\partial Q_m}(\tau_{i,l-1}(\theta) - \tau_{i,l}(\theta)) + \frac{\partial R_i'}{\partial Q_m}\right] \tag{48}$$

From (35) we may write (from the product rule or by logarithmic differentiation)

$$\begin{aligned}\frac{\partial \tau_{i,l}(\theta)}{\partial Q_m} &= \tau_{a_j}(\theta)\frac{\partial}{\partial Q_m}\prod_{j=1}^{l}\tilde{\tau}_{i,j}(\theta) \\ &= \tau_{i,l}(\theta)\sum_{j=1}^{l}\frac{1}{\tilde{\tau}_{i,j}(\theta)}\frac{\partial \tilde{\tau}_{i,j}(\theta)}{\partial Q_m}\end{aligned} \tag{49}$$

Recall that the layer transmittance $\tilde{\tau}_{i,j}(\theta)$ is obtained by linear interpolation from transmittances calculated at rapid algorithm angles; thus, we have:

$$\frac{\partial \tilde{\tau}_{i,j}(\theta)}{\partial Q_m} = (1-\alpha)\frac{\partial \tilde{\tau}_{i,j,k}}{\partial Q_m} + \alpha\frac{\partial \tilde{\tau}_{i,j,k+1}}{\partial Q_m} \tag{50}$$

where θ_k and θ_{k+1} are the rapid algorithm zenith angles which bracket the desired angle θ and $\alpha = (\sec(\theta) - \sec(\theta_k))/(\sec(\theta_{k+1}) - \sec(\theta_k))$. Substituting into (49), we have

$$\frac{\partial \tau_{i,l}(\theta)}{\partial Q_m} = \tau_{i,l}(\theta)\sum_{j=1}^{l}\frac{1}{\tilde{\tau}_{i,j}(\theta)}\left\{(1-\alpha)\frac{\partial \tilde{\tau}_{i,j,k}}{\partial Q_m} + \alpha\frac{\partial \tilde{\tau}_{i,j,k+1}}{\partial Q_m}\right\} \tag{51}$$

Let's concentrate now on a specific quantity such as ozone column density for layer m ($Q_m = u_m$). We know that

$$\tilde{\tau}_{i,j,k} = \tilde{\tau}_{F_{i,j,k}}\tilde{\tau}_{O_{i,j,k}}\tilde{\tau}_{W_{i,j,k}} \tag{52}$$

Since by definition $\tilde{\tau}_{F_i}$ and $\tilde{\tau}_{W_i}$ do not depend on ozone we need only consider $\tilde{\tau}_{O_i}$ when calculating the derivative of $\tilde{\tau}_{i,j,k}$. We can write:

$$\begin{aligned}\frac{\partial \tilde{\tau}_{O_{i,j,k}}}{\partial u_m} &= -\tilde{\tau}_{O_{i,j,k}}F_{i,j,k}[1 - G_{i,k}(\bar{T}_j - 273)]N_i u_j^{N_i-1}\frac{\partial u_j}{\partial u_m} \\ &= \frac{\partial \tilde{\tau}_{O_{i,m}}}{\partial u_m}\delta_{jm}\end{aligned} \tag{53}$$

since the ozone column density in layer m enters into the calculation of the effective layer transmittance only at layer m. We can write similar expressions for the derivatives with respect to specific humidity. The derivative $\partial\tau_{W_{i,j,k}}/\partial q_m$ is almost the same as $\partial\tau_{O_{i,j,k}}/\partial u_m$ but with an added term for the water vapor continuum.

Then we can also write:

$$\begin{aligned}\frac{\partial\tilde{\tau}_{i,j,k}}{\partial u_m} &= \tilde{\tau}_{F_{i,m,k}}\tilde{\tau}_{W_{i,m,k}}\frac{\partial\tilde{\tau}_{O_{i,m,k}}}{\partial u_m}\delta_{jm} \\ &= \frac{\tilde{\tau}_{i,m,k}}{\tilde{\tau}_{O_{i,m,k}}}\frac{\partial\tilde{\tau}_{O_{i,m,k}}}{\partial u_m}\delta_{jm} \qquad (54)\\ \frac{\partial\tilde{\tau}_{i,j,k}}{\partial q_m} &= \tilde{\tau}_{F_{i,m,k}}\tilde{\tau}_{O_{i,m,k}}\frac{\partial\tilde{\tau}_{W_{i,m,k}}}{\partial u_m}\delta_{jm} \\ &= \frac{\tilde{\tau}_{i,m,k}}{\tilde{\tau}_{W_{i,m,k}}}\frac{\partial\tilde{\tau}_{W_{i,m,k}}}{\partial u_m}\delta_{jm} \qquad (55)\end{aligned}$$

Since $\partial\tilde{\tau}_{i,j,k}/\partial Q_m = 0$ for $j \neq m$ (for $Q_m = q_m$ or u_m) we may also drop the summation over j in (49) and (51) and write, for $l > m$

$$\frac{\partial\tau_{i,l}(\theta)}{\partial Q_m} = \frac{\tau_{i,l}(\theta)}{\tilde{\tau}_{i,m}(\theta)}\frac{\partial\tilde{\tau}_{i,m}(\theta)}{\partial Q_m} = \frac{\tau_{i,l}(\theta)}{\tilde{\tau}_{i,m}(\theta)}\left\{(1-\alpha)\frac{\partial\tilde{\tau}_{i,m,k}}{\partial Q_m} + \alpha\frac{\partial\tilde{\tau}_{i,m,k+1}}{\partial Q_m}\right\} \qquad (56)$$

$$\begin{aligned}\frac{\partial}{\partial Q_m}(\tau_{i,l-1}(\theta) - \tau_{i,l}(\theta)) &= \frac{(\tau_{i,l-1}(\theta) - \tau_{i,l}(\theta))}{\tilde{\tau}_{i,m}(\theta)}\frac{\partial\tilde{\tau}_{i,m}(\theta)}{\partial Q_m} \\ &= \frac{(\tau_{i,l-1}(\theta) - \tau_{i,l}(\theta))}{\tilde{\tau}_{i,m}(\theta)}\left\{(1-\alpha)\frac{\partial\tilde{\tau}_{i,m,k}}{\partial Q_m} + \alpha\frac{\partial\tilde{\tau}_{i,m,k}}{\partial Q_m}\right\}\end{aligned} \qquad (57)$$

These expressions can be substituted into (48) to compute the ozone and water vapor Jacobians.

$$\begin{aligned}\frac{\partial\Theta}{\partial Q_m} = \frac{\partial\Theta}{\partial R}\Bigg(&\frac{1}{\tilde{\tau}_{i,m}(\theta)}\left\{(1-\alpha)\frac{\partial\tilde{\tau}_{i,m,k}}{\partial Q_m} + \alpha\frac{\partial\tilde{\tau}_{i,m,k+1}}{\partial Q_m}\right\} \\ &\left[\varepsilon_i B_i(T_g)\tau_{i,N}(\theta) - B_i(\bar{T}_m)\tau_{i,m}(\theta) + \sum_{l=m+1}^{N} B_i(\bar{T}_l)(\tau_{i,l-1}(\theta) - \tau_{i,l}(\theta))\right] + \frac{\partial R_i'}{\partial Q_m}\Bigg)\end{aligned} \qquad (58)$$

Temperature derivative For the temperature derivative, there are additional terms for the derivative of the Planck function. For convenience, we calculate the derivative with respect to mean layer temperature $\bar{T}_m = 0.5(T_{m-1} + T_m)$

$$\begin{aligned}\frac{\partial\Theta_i}{\partial\bar{T}_m} = \frac{\partial\Theta_i}{\partial R_i}\Bigg[&\varepsilon_i B_i(T_g)\frac{\partial\tau_{iN}}{\partial\bar{T}_m} + \frac{\partial B_i(\bar{T}_m)}{\partial\bar{T}_m}(\tau_{i,m-1} - \tau_{i,m}) \\ &+ \sum_{l=1}^{N} B_i(\bar{T}_l)\frac{\partial}{\partial\bar{T}_m}(\tau_{i,l-1} - \tau_{i,l}) + \frac{\partial R_i'}{\partial\bar{T}_l}\Bigg]\end{aligned} \qquad (59)$$

and then use that derivative in calculating the level temperature Jacobian

$$\frac{\partial\Theta_i}{\partial T_j} = \sum_{m=0}^{N}\frac{\partial\Theta_i}{\partial\bar{T}_m}\frac{\partial\bar{T}_m}{\partial T_j} = 0.5\left[\frac{\partial\Theta}{\partial\bar{T}_{j-1}} + \frac{\partial\Theta}{\partial\bar{T}_j}\right] \qquad (60)$$

The temperature derivative has terms for the derivative of the Planck function and the derivative of the transmittances with respect to temperature. In practice, the transmittance derivative is often neglected because it is assumed to be much smaller than the derivative of the Planck function. The Planck function derivative was given in (46). The derivation of the transmittance derivative is similar to that in the previous section, up to (51). Unlike ozone and water vapor, there is a temperature contribution in all three components ($\tau_{F_i}, \tau_{O_i}, \tau_{W_i}$) of the effective mean layer transmittance. Thus, by logarithmic differentiation we write:

$$\frac{\partial \tilde{\tau}_{i,j,k}}{\partial \bar{T}_m} = \tilde{\tau}_{i,j,k} \left(\frac{1}{\tilde{\tau}_{F_{i,j,k}}} \frac{\partial \tilde{\tau}_{F_{i,j,k}}}{\partial \bar{T}_m} + \frac{1}{\tilde{\tau}_{O_{i,j,k}}} \frac{\partial \tilde{\tau}_{O_{i,j,k}}}{\partial \bar{T}_m} + \frac{1}{\tilde{\tau}_{W_{i,j,k}}} \frac{\partial \tilde{\tau}_{W_{i,j,k}}}{\partial \bar{T}_m} \right) \tag{61}$$

The derivatives of the water vapor and ozone contributions to transmittance with respect to temperature are relatively simple:

$$\frac{\partial \tilde{\tau}_{W_{i,j,k}}}{\partial \bar{T}_m} = \left[D_{i,j,k} E_{i,k} (\bar{T}_j - 273) u_{q_j}^{M_i} - \sec(\theta_k) \frac{\partial K_{H_2O_{ij}}}{\partial \bar{T}_m} \right] \tau_{W_{i,j,k}} \delta_{jm} \tag{62}$$

$$\frac{\partial \tilde{\tau}_{O_{i,j,k}}}{\partial \bar{T}_m} = \left(F_{i,j,k} G_{i,k} (\bar{T}_j - 273) u_{O_{3j}}^{N_i} \right) \tau_{O_{i,j,k}} \delta_{jm} \tag{63}$$

For these derivatives (as with the transmittance derivatives with respect to ozone and water vapor) the mean temperature for a given layer affects only the effective transmittance contribution for that same layer.

The temperature derivative of the fixed gas contribution is complicated by the effective mean temperature term. Since we calculate effective mean temperature for a layer through a weighted sum of temperatures from all layers above, the temperature in a given layer affects the effective mean layer transmittance for every layer below that one.

$$\begin{aligned} \frac{\partial \tilde{\tau}_{F_{i,j,k}}}{\partial \bar{T}_m} &= B_{i,j,k} \frac{\partial \bar{T}_j}{\partial T_m} + C^*_{i,j,k} \sum_{l=1}^{j} W_{i,l,k} \frac{\partial \bar{T}_l}{\partial \bar{T}_j} \\ &= B_{i,j,k} \delta_{jm} + C^*_{i,j,k} \sum_{l=1}^{j} W_{i,l,k} \delta_{lm} \end{aligned} \tag{64}$$

Thus

$$\frac{\partial \tilde{\tau}_{F_{i,j,k}}}{\partial \bar{T}_m} = \begin{cases} 0 & j < m \\ B_{i,j,k} + C^*_{i,j,k} W_{i,m,k} & j = m \\ C^*_{i,j,k} W_{i,m,k} & j > m \end{cases} \tag{65}$$

and

$$\frac{\partial \tilde{\tau}_{i,j,k}}{\partial \bar{T}_m} = \begin{cases} 0 & j < m \\ \tilde{\tau}_{i,j,k} \left(\frac{1}{\tilde{\tau}_{F_{i,j,k}}} \frac{\partial \tilde{\tau}_{F_{i,j,k}}}{\partial \bar{T}_m} + \frac{1}{\tilde{\tau}_{O_{i,j,k}}} \frac{\partial \tilde{\tau}_{O_{i,j,k}}}{\partial \bar{T}_m} + \frac{1}{\tilde{\tau}_{W_{i,j,k}}} \frac{\partial \tilde{\tau}_{W_{i,j,k}}}{\partial \bar{T}_m} \right) & j = m \\ \frac{\tilde{\tau}_{i,j,k}}{\tilde{\tau}_{F_{i,j,k}}} \frac{\partial \tilde{\tau}_{F_{i,j,k}}}{\partial \bar{T}_m} & j > m \end{cases} \tag{66}$$

Also

$$\frac{\partial \tilde{\tau}_{i,j}(\theta)}{\partial \bar{T}_m} = \begin{cases} 0 & j < m \\ (1-\alpha) \frac{\partial \tilde{\tau}_{i,j,k}}{\partial \bar{T}_m} + \alpha \frac{\partial \tilde{\tau}_{i,j,k+1}}{\partial \bar{T}_m} & j \geq m \end{cases} \tag{67}$$

and since the temperature in a given layer only affects the effective mean layer transmittance in layers at and below that layer we can write for $l \geq m$

$$\frac{\partial \tau_{i,l}(\theta)}{\partial \bar{T}_m} = \tau_{i,l}(\theta) \sum_{j=m}^{l} \frac{1}{\tilde{\tau}_{i,j}(\theta)} \frac{\partial \tilde{\tau}_{i,j}(\theta)}{\partial \bar{T}_m} \tag{68}$$

Since $\partial \tau_{i,l}(\theta)/\partial \bar{T}_m = 0$ for $l < m$ we can also rewrite (59) to get:

$$\begin{aligned} \frac{\partial \Theta_i}{\partial \bar{T}_m} = \frac{\partial \Theta_i}{\partial R_i} \Bigg[& \varepsilon_i B_i(T_g) \frac{\partial \tau_{iN}}{\partial \bar{T}_m} + \frac{\partial B_i(\bar{T}_m)}{\partial \bar{T}_m} (\tau_{i,m-1} - \tau_{i,m}) - B_i(\bar{T}_m) \frac{\partial \tau_{i,m}}{\partial \bar{T}_m} \\ & + \sum_{l=m+1}^{N} B_i(\bar{T}_l) \left(\frac{\partial \tau_{i,l-1}}{\partial \bar{T}_m} - \frac{\partial \tau_{i,l}}{\partial \bar{T}_m} \right) + \frac{\partial R_i'}{\partial \bar{T}_l} \Bigg] \end{aligned} \tag{69}$$

Derivative of reflected radiance term Obtaining the derivatives of the reflected radiance term is the final step in the calculation of the Jacobians. From (38) we can write:

$$\frac{\partial R_i'(\theta)}{\partial Q} = (1 - \varepsilon_i) \frac{\partial R_i \downarrow}{\partial Q} \tau_{i,N}(\theta) + (1 - \varepsilon_i) R_i \downarrow \frac{\partial \tau_{i,N}}{\partial Q} + \rho_i H_i \frac{\partial \tau_{iS}(\theta_{EFF})}{\partial Q} \tag{70}$$

The derivatives of the transmittances $\partial \tau_{i,N}/\partial Q$ and $\partial \tau_{iS}(\theta_{EFF})/\partial Q$ can be obtained as in (56) and (68). This leaves us only with the derivative of the downward flux $\partial R_i \downarrow /\partial Q$ to consider.

The expressions for downward flux were given in (39)–(41). For microwave channels, the form of the downward flux derivative is the most complex. For the water vapor and ozone derivatives, we have:

$$\begin{aligned} \frac{\partial R_i \downarrow}{\partial Q_m} = \frac{\partial \tau_{i,N}(\theta)}{\partial Q_m} & \left[T_{BB} + \sum_{l=1}^{N} \bar{T}_l \left(\frac{1}{\tau_{i,l}(\theta)} - \frac{1}{\tau_{i,l-1}(\theta)} \right) \right] + \\ & \frac{\tau_{i,N}(\theta)}{\tilde{\tau}_{i,m}(\theta)} \left\{ \frac{\bar{T}_m}{\tau_{i,m}(\theta)} + \sum_{l=m+1}^{N} \bar{T}_l \left[\left(\frac{1}{\tau_{i,l-1}(\theta)} \right) - \left(\frac{1}{\tau_{i,l}(\theta)} \right) \right] \right\} \frac{\partial \tilde{\tau}_{i,m}(\theta)}{\partial Q_m} \end{aligned} \tag{71}$$

The expression for the temperature derivative is more complicated:

$$\begin{aligned} \frac{\partial R_i \downarrow}{\partial \bar{T}_m} = & \frac{\partial \tau_{i,N}(\theta)}{\partial \bar{T}_m} \left[T_{BB} + \sum_{l=1}^{N} \bar{T}_l \left(\frac{1}{\tau_{i,l}(\theta)} - \frac{1}{\tau_{i,l-1}(\theta)} \right) \right] \\ & + \tau_{i,N}(\theta) \left[\left(\frac{1}{\tau_{i,m-1}(\theta)} \right) - \left(\frac{1}{\tau_{i,m}(\theta)} \right) \right] - \tau_{i,N}(\theta) \bar{T}_m \left(\frac{1}{\tau_{i,m}(\theta)} \right)^2 \frac{\partial \tau_{i,m}(\theta)}{\partial \bar{T}_m} \\ & + \tau_{i,N}(\theta) \sum_{l=m+1}^{N} \bar{T}_l \left[\left(\frac{1}{\tau_{i,l-1}(\theta)} \right)^2 \frac{\partial \tau_{i,l-1}(\theta)}{\partial \bar{T}_m} - \left(\frac{1}{\tau_{i,l}(\theta)} \right)^2 \frac{\partial \tau_{i,l}(\theta)}{\partial \bar{T}_m} \right] \end{aligned} \tag{72}$$

For IR channels where the atmosphere is optically thin, Eq. (40) for $R_i \downarrow$ involves a vertical sum–the atmospheric component of the radiance calculation for which we have already

derived (and in the modules have calculated) appropriate expressions. Thus we have

$$\frac{\partial R_i\downarrow}{\partial Q_m} = 2\cos\theta \frac{\partial}{\partial Q_m} \sum_{l=m}^{N} B_i(\bar{T}_l)\,(\tau_{i,l-1}(\theta) - \tau_{i,l}(\theta)) \tag{73}$$

For the other IR channels, the expression for the derivative of the downward flux is much simpler. The only term affected by changes in the atmospheric profile is the surface transmittance, thus:

$$\frac{\partial R_i\downarrow}{\partial Q_m} = -F_i B_i(T_s) \frac{\partial \tau_{iN}(\theta)}{\partial Q_m} \tag{74}$$

4 Modules for forward radiance calculation

The last two sections of this office note discuss the program modules used to implement the GLA forward model and Jacobian. This discussion is aimed primarily at the end user who may wish to call the forward model or Jacobian code from their program. The entry points to the main routines are given, along with a complete description of the input and output parameters. The program flow is documented by listing the subroutines called by the main routines and describing the calculations performed. Some relevant variables in the subroutines are described using mathematical formulas.

In this section, the Fortran modules used to implement the GLA forward model are discussed. The program flow is very simple to follow. The major steps are initialization of `COMMON` blocks with rapid algorithm coefficients, calculation of transmittances, and the forward model calculation of radiance (or brightness temperature). The routines responsible for these steps are **`initialGLA`**, **`tovs_tau`**, and **`hirsrad`**, respectively.

4.1 Initialization: `initialGLA`

Input parameters:

`itrfdat`	`INTEGER` unit number (previously opened) of transmittance data file
`satelite`	`CHARACTER*2` satellite identifier (see Table 2)

Output parameters:

`bland`	`LOGICAL (360,180)`	array of land/water flags
`ierr`	`INTEGER`	error flag for transmittance coefficients
	0	successful read
	1	error reading dataset
	2	less than 24 channels found

Table 2: Satellite identifiers

satellite	id
TIROS–N	TN
NOAA–6	NA
NOAA–7	NC
NOAA–8	NE
NOAA–9	NF
NOAA–10	NG
NOAA–11	NH
NOAA–12	ND

Subroutines called:

`matcon`	Sets math constants for retrievals
`phycon`	Sets physical constants for retrievals
`grdcon`	Sets vertical grid constants
`chacon`	Sets channel constants for TOVS channels
`dattrf`	Reads rapid algorithm transmittance coefficients

Before the transmittance and radiance calculations can be performed, coefficients and physical and mathematical constants must be loaded into the TOVS `COMMON` blocks. The `initialGLA` routine is an example driver for this; the input variables are the unit number of the transmittance coefficient file (previously opened by the main program) and a 2-letter character string indicating which satellite is being used. The current configuration only accomdates the use of coefficients from one satellite at a time. The `initialGLA` routine calls routines to initialize physical (`phycon`), mathematical (`matcon`) and vertical grid (`grdcon`) constants. This version of the `initialGLA` routine opens and reads in a dataset with land/water flags; this may not be necessary in future versions of the code if land/water flags are provided with the input profiles. After the land/water flags are set, routines are called to initialize channel-related constants (`chacon`) and to read in the rapid algorithm transmittance coefficients (`dattrf`). Each of the initialization routines `phycon`, `matcon`, `grdcon`, `chacon` and `dattrf` has an associated `COMMON` block defined in `INCLUDE` files `phycons.h`, `matcons.h`, `grdcons.h`, `chacons.h` and `taucoef.h`, respectively. Most of these initialization routines are straightforward, except for `dattrf` which is summarized below.

4.1.1 Transmittance coefficients: `dattrf`

The purpose of `dattrf` is to read transmittance coefficients, perform some calculations to convert the coefficients to the form which they will be used in the radiance calculation, and store the coefficients into the `taucoef` `COMMON` block. In the first step the zenith angles for the rapid algorithm coefficients are set up. Then, the coefficients for the rapid algorithm are read into temporary arrays. These coefficients were fitted to the original 66-level rapid algorithm; so the level-dependent coefficients at and above the 2 mb level are adjusted to fit the current 71-level vertical integration. In the final section of the routine, the coefficients are copied into the proper 'slots' in the common block, based on the input channel id.

In the last section of **dattrf** we also perform calculations to adjust the units of the transmittance coefficients to match those of the profiles used in the forward radiance calculation. The original transmittance coefficients were fitted for water vapor and ozone column densities expressed in terms of molecules/cm^2 while the current calculation uses densities of g/cm^2. The conversion factors are raised to the same power as the column density (see (18)).

Some coefficients used in the fixed gas transmittance calculation are also set in the last section of **dattrf**. Eq. (26) showed how the coefficients associated with the fixed gas calculation could be arranged in a compact form; the calculations in the last section of **dattrf** are performed to obtain these new coefficients. The variables from **dattrf** associated with (26) are:

$$\begin{aligned}
\texttt{ZQ} &= 1 - \tau^{\circ}_{ij} = 1 - \prod_{l=1}^{j} A_{il}(\theta) \\
\texttt{TAUCFW} &= (1 - A_{ik}(\theta)) \prod_{l=1}^{k-1} A_{il}(\theta) \\
\texttt{TAUCFC} &= C^{*}_{ij} = \frac{C_{ij}(\theta)}{1 - \prod_{l=1}^{j} A_{il}(\theta)} = \frac{C_{ij}(\theta)}{1 - \tau^{\circ}_{ij}}.
\end{aligned}$$

The **dattrf** routine was copied almost verbatim from the GLA TOVS retrieval code, to ensure the compatibility of the transmittance coefficients. Future versions of this code could be simplified if the coefficient database were reformatted, but the link to the original physics of the rapid algorithm would then be lost. Any revision of this routine should be done with care.

4.2 Transmittance calculation: `tovs_tau`

Input parameters:

tair	REAL (MAXLEV)	array of temperatures (K) at rapid algorithm levels
tmptop	REAL	temperature for top level of integration
h2omid	REAL (MAXLEV)	array of layer specific humidities (kg H_2O / kg air)
ozomid	REAL (MAXLEV)	array of layer ozone column densities (g / cm^2)
ps	REAL	surface pressure (mb)
nchan	INTEGER	number of channels to calculate
chanarr	INTEGER (MAXCHA)	array of channels to calculate; 1–20 = HIRS, 21–24 = MSU
nlev	INTEGER	index of surface level (also number of levels to use in calculation)
mw_zen	REAL	satellite zenith angle (degrees) for microwave channels
ir_zen	REAL	satellite zenith angle (degrees) for IR channels
dayflag	LOGICAL	true if "day" (sun angle > 85 degrees)
secsun	REAL	1. / cos(sun angle)

Output parameters:

`tau`	`REAL (MAXLEV,MAXCHA)`	transmittance along path to satellite
`tausun`	`REAL (MAXCHA)`	transmittance along path from sun to surface to satellite
`errflag`	`LOGICAL`	returns `.true.` if error in calculation

Subroutines called:

`tovsfix_tau`	transmittance contribution due to fixed gases
`tovsh2o_tau`	transmittance contribution due to water vapor
`tovsozo_tau`	transmittance contribution due to ozone
`tovs_tauang`	interpolate layer transmittance to satellite zenith angle; take products over layers

The `tovs_tau` routine is the driver for the transmittance calculation. The input to this routine includes profiles of temperature at the rapid algorithm pressure levels and specific humidity and ozone column density for layers bounded by the pressure levels. The strategy used is to calculate variables common to all channel calculations (such as mean layer temperature, moisture variables, and zenith angle factors), then loop through the input channel list to calculate transmittances.

For the moisture variables, unit conversions are performed, since the input units of specific humidity (q: kg H_2O / kg air) are different from the units needed for the rapid algorithm (column density u_q: g cm^{-2}) and the continuum model (vapor pressure e: atm). These conversions were discussed in (29) and (30).

$$\texttt{effh2o} = \hat{u}_{H_2O_j} = \rho_{H_2O_j}\Delta z_j = \rho_{air} q_j \Delta z_j = \frac{10\Delta p}{g} q_j$$

$$\texttt{e} = e_j(\text{atm}) = \frac{\text{mol. air}}{\text{mol. } H_2O} \frac{\bar{p}(\text{mb})}{1013.25 \frac{\text{mb}}{\text{atm}}} q_j$$

The routines `tovsfix_tau`, `tovsh2o_tau` and `tovsozo_tau` are called to calculate effective layer transmittances for the fixed gas contribution, $\tau_{iF}(p_l, \theta)$, water vapor contribution $\tau_{iW}(p_l, \theta)$, and ozone contribution $\tau_{iO}(p_l, \theta)$ (see Section 2.4 for discussion). The product of these effective layer transmittances calculated at the rapid algorithm angles is taken to obtain the layer transmittance at the rapid algorithm angles. These layer transmittances are interpolated linearly with respect to sec θ to the observed satellite zenith angle, and the product of the layer transmittances is taken to obtain the total transmittance between the satellite and each layer.

4.2.1 Fixed gas contribution: `tovsfix_tau`

In `tovsfix_tau`, the fixed gas contribution to effective layer transmittances is calculated using the model described in Section 2.4.2, with the coefficients modified as in Eq. (26).

The variables in **tovsfix_tau** associated with (26) are

$$\begin{aligned} \texttt{effTdif} &= \sum_{k=1}^{j} W_{ik}(\theta)(\bar{T}_k - \bar{T}_k^{\circ}) \\ \texttt{taufix} &= \tilde{\tau}_{iF}(p_j, p_{j-1}, \theta) \end{aligned}$$

The effective temperature **effTdif** is carried as a running sum as the calculation of **taufix** is carried from the top of the atmosphere down to the surface.

4.2.2 Water vapor contribution: tovsh2o_tau

The contribution of water vapor to transmittance is calculated in **tovsh2o_tau** as a combination of the modeled water vapor transmittance $\tilde{\tau}_{iW}$ (18) and the water vapor continuum term K_{H_2O} (21).

Key variables in **tovsh2o_tau** include:

$$\begin{aligned} \texttt{degchn} &= -\frac{dk}{dT} \\ \texttt{cont} &= K_{H_2O_{i,j}} = k_i \left(1 + \frac{dk}{dT}(\bar{T}_j - T_{cont,j})\right) e_j \hat{u}_{H_2O_j} \\ \texttt{tauh2o} &= \tilde{\tau}_{H_2O_{i,j,k}} = \exp\left[\left(-D_{i,j}(\theta)[1 - E_i(\bar{T}_j - 273)]u_{q_j}^{M_i}\right) - \sec(\theta) K_{H_2O_{i,j}}\right] \end{aligned}$$

4.2.3 Ozone contribution: tovsozo_tau

The calculation of the ozone transmittance contribution is relatively simple. Since the input profile and transmittance coefficients are already expressed in terms of the same units, we need only perform the calculation for layer ozone transmittance contribution given in (18).

$$\texttt{tauozo} = \tilde{\tau}_{O_{3i,j,k}} = \exp\left(-F_{i,j,k}[1 - G_{i,k}(\bar{T}_j - 273)]\{(\hat{u}_{O_{3j}})^{N_i}\}\right)$$

4.2.4 Zenith angle interpolation: tovs_tauang

After the product of the fixed gas, water vapor, and ozone contributions is taken to obtain the effective layer transmittance, a call is made to **tovs_tauang** to interpolate the layer transmittances to the proper zenith angle. The transmittance between each rapid algorithm level and the satellite is calculated as the product of all the layer transmittances above that level (12). For the IR channels, the transmittances are multiplied by the scattering factor described in Section 2.4.4. Finally, if solar effects will need to be calculated (*i.e.* daytime observation for IR channels 13–19), the transmittance is also interpolated to the effective solar zenith angle

$$\theta_{EFF} = \sec^{-1}(\sec(\varphi) + \sec(\theta))$$

(where φ is the solar zenith angle) along with the appropriate scattering factors.

Some relevant variables are

$$\texttt{tauang} = \tilde{\tau}_{i,j,k} = \tilde{\tau}_{F_{i,j,k}} \tilde{\tau}_{H_2O_{i,j,k}} \tilde{\tau}_{O_{3_{i,j,k}}}$$

$$\texttt{tau} = \tau_{i,j}(\theta) = \tau_{a_{i,j}}(\theta) \prod_{l=1}^{j} \left\{ \tilde{\tau}_{i,l,k} + \frac{\sec(\theta) - \sec(\theta_k)}{\sec(\theta_{k+1}) - \sec(\theta_k)} (\tilde{\tau}_{i,l,k+1} - \tilde{\tau}_{i,l,k}) \right\}$$

4.3 Radiance calculation: `hirsrad`

Input parameters:

`tau`	`REAL (MAXLEV,MAXCHA)`	transmittance along path to satellite
`tausun`	`REAL (MAXCHA)`	transmittance along path from sun to surface to satellite
`nchan`	`INTEGER`	number of channels to calculate
`chanarr`	`INTEGER (MAXCHA)`	array of channels to calculate; 1–20 = HIRS, 21–24 = MSU
`tair`	`REAL (MAXLEV)`	array of temperatures (K) at rapid algorithm levels
`tmptop`	`REAL`	temperature for top level of integration
`ir_zen`	`REAL`	satellite zenith angle (degrees) for IR channels
`emissmw`	`REAL`	microwave surface emissivity
`land`	`LOGICAL`	land/water flag (`.true.` if over land)
`nlev`	`INTEGER`	index of surface level (also number of levels to use in calculation)
`rho`	`REAL`	bidirectional surface reflectance
`sunzang`	`REAL`	solar zenith angle (degrees)
`dayflag`	`LOGICAL`	flag for solar calculation, `.true.` if day
`tground`	`REAL`	skin (ground) temperature (K)
`tsurfair`	`REAL`	surface air temperature (K)

Output parameters:

`calc_rad`	`REAL (MAXCHA)`	IR channels: calculated radiance (w/ cm^2 sr) MW channels: calculated brightness temperature (K)

The forward radiance calculation is carried out as described in Sections 2.1 and 3. The vertical sums are carried out separately for infrared and microwave channels, with surface terms and reflected radiance added afterwards.

The contribution from reflected radiance is calculated as described in Section 2.3, according to (38)–(42). For channels 8, 10, 18, and 19 (`use_refl = .true.`) where (40) applies, the atmospheric integral part of $R_i\downarrow$ is simply the partial sum in `crad` at that point (since surface terms have not yet been added). The solar contribution (second term in (38)) is only used for daytime calculations for IR channels 13–19.

5 Modules for Jacobian calculation

5.1 Brightness temperature Jacobian: `tovs_tauK`

Input parameters:

`tair`	`REAL (MAXLEV)`	array of temperatures (K) at rapid algorithm levels
`tmptop`	`REAL`	temperature for top level of integration
`h2omid`	`REAL (MAXLEV)`	array of layer specific humidities (kg H_2O / kg air)
`ozomid`	`REAL (MAXLEV)`	array of layer ozone column densities (g / cm^2)
`ps`	`REAL`	surface pressure (mb)
`nchan`	`INTEGER`	number of channels to calculate
`chanarr`	`INTEGER (MAXCHA)`	array of channels to calculate; 1–20 = HIRS, 21–24 = MSU
`nlev`	`INTEGER`	index of surface level (also number of levels to use in calculation)
`mw_zen`	`REAL`	satellite zenith angle (degrees) for microwave channels
`ir_zen`	`REAL`	satellite zenith angle (degrees) for IR channels
`dayflag`	`LOGICAL`	true if "day" (sun angle > 85 degrees)
`secsun`	`REAL`	1. / cos(sun angle)
`emissmw`	`REAL`	microwave surface emissivity
`land`	`LOGICAL`	land/water flag - `.true.` if measurement over land
`rho`	`REAL`	bi-directional surface reflectance
`tground`	`REAL`	skin (ground) temperature (K)
`tsurfair`	`REAL`	surface air temperature (K)

Output parameters:

`tau`	`REAL (MAXLEV,MAXCHA)`	transmittance along path to satellite
`tausun`	`REAL (MAXCHA)`	transmittance along path from sun to surface to satellite
`bt`	`REAL (MAXCHA)`	brightness temperatures (K)
`dTbdT`	`REAL (MAXLEV,MAXCHA)`	temperature Jacobian $\partial\Theta_i/\partial T_j$
`dTbdq`	`REAL (MAXLEV,MAXCHA)`	water vapor Jacobian $\partial\Theta_i/\partial q_j$
`dTbdu`	`REAL (MAXLEV,MAXCHA)`	ozone Jacobian $\partial\Theta_i/\partial u_j$
`errflag`	`LOGICAL`	returns `.true.` if error in calculation

Subroutines called:

`tovsfix_tauK`	transmittance contrib. by fixed gases and derivative
`tovsh2o_tauK`	transmittance contrib. by water vapor and derivative
`tovsozo_tauK`	transmittance contrib. by ozone and derivative
`tovs_tauprodK`	take product of fixed, water vapor and ozone contributions and derivatives
`tovs_tauangK`	interpolate transmittance to obs. zenith angle and calculate consistent derivative terms
`dTb_dTQU`	calculate brightness temperature and Jacobian

The calculation of the brightness temperature Jacobian follows essentially the same steps as forward problem calculation of radiance. (Actually, transmittances are set up and the forward radiance calculation is performed during the process of obtaining the Jacobian.) The mapping between forward problem modules and Jacobian modules is indicated by adding the suffix `K` to the module name, *i.e.* **`tovsozo_tauK`** is the Jacobian corresponding to **`tovsozo_tau`** in the forward model.

The finite-difference representations of the equations used were given in Section 3.2. The strategy for the Jacobian calculation is to first calculate the derivatives of the effective layer transmittance contributions from fixed gases, water vapor, and ozone (in **`tovsfix_tauK`**, **`tovsh2o_tauK`** and **`tovsozo_tauK`**). These derivatives are combined to obtain the derivatives of effective layer transmittance (**`tovs_tauprodK`**). The layer transmittance derivatives are interpolated to the appropriate zenith angle and combined to form the total transmittance derivatives (**`tovs_tauangK`**) , which are used to calculate the brightness temperature Jacobians (**`dTb_dTQU`**). Since many of the transmittance terms involve calculation of products, it is convenient to calculate and store the logarithmic derivative of such terms (*i.e.* $\frac{1}{F}\frac{\partial F}{\partial Q}$ rather than $\frac{\partial F}{\partial Q}$) since, if $F = abc$, then by the product rule

$$\frac{\partial F}{\partial Q} = \frac{\partial (abc)}{\partial Q} = F\left(\frac{1}{a}\frac{\partial a}{\partial Q} + \frac{1}{b}\frac{\partial b}{\partial Q} + \frac{1}{c}\frac{\partial c}{\partial Q}\right)$$

Another thing to note about the derivatives is that when the transmittance is set to zero, we also set the derivative to zero. The GLA retrieval code (and also this forward radiance code) contains tests to prevent negative transmittance values; if a transmittance goes negative it is set to zero. Then, since the transmittance has been set to a constant value, its derivative must also be zero.

5.1.1 Fixed gas contribution: `tovsfix_tauK`

The fixed gas contribution to transmittance is a function only of temperature. We can use the derivative formula (65) to obtain the logarithmic derivative

$$\mathtt{dttaufix} = \frac{1}{\tilde{\tau}_{F_{i,j,k}}}\frac{\partial \tilde{\tau}_{F_{i,j,k}}}{\partial \tilde{T}_m} = \begin{cases} 0 & j < m \\ \frac{1}{\tilde{\tau}_{F_{i,j,k}}}\left(B_{i,j,k} + C^*_{i,j,k} W_{i,m,k}\right) & j = m \\ \frac{C^*_{i,j,k}}{\tilde{\tau}_{F_{i,j,k}}} W_{i,m,k} & j > m \end{cases}$$

Since this derivative is zero for transmittance layers $j < m$, it is efficient to use a packed matrix to store **`dttaufix`**. (In fact, the time required to calculate the Jacobian was reduced about 20 % when packed notation is used compared with a full matrix representation.) If we take $\mathtt{dttaufix} = \frac{1}{\tilde{\tau}_{F_{i,j,k}}}\frac{\partial \tilde{\tau}_{F_{i,j,k}}}{\partial \tilde{T}_m} = a_{mj}$ then the matrix has the form

$$\mathtt{dttaufix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & \dots & a_{1N} \\ 0 & a_{22} & a_{23} & a_{24} & \dots & a_{2N} \\ 0 & 0 & a_{33} & a_{34} & \dots & a_{3N} \\ 0 & 0 & 0 & a_{44} & \dots & a_{4N} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & \dots & a_{NN} \end{bmatrix}$$

where we have taken the temperature layer as the first index and the transmittance layer as the second index. In the program, this upper triangular matrix is stored as

$$\texttt{dttaufix} = [a_{11}, a_{12}, a_{22}, a_{13}, a_{23}, a_{33}, a_{14}, a_{24}, a_{34}, a_{44}, \ldots]$$

and the index of $(m, j) = (j-1)(j/2) + m$ for $1 < m < j$.

5.1.2 Water vapor contribution: tovsh2o_tauK

The derivatives of the water vapor and ozone contributions to effective layer transmittance are much simpler than the fixed gas term, since layer temperatures and column densities of ozone and water vapor only affect the layer transmittance for that same layer. (See (53) and (62)). Some expressions used in this module are:

$$\texttt{celsius} = \bar{T}_j - 273$$

$$\texttt{deffh2o} = \frac{\partial u_{q_j}}{\partial q_m} = \frac{10\Delta p_j}{g}$$

$$\texttt{dqcont} = \frac{\partial K_{H_2O_{i,j}}}{\partial q_m} = 2k_i \left(1 + \frac{dk}{dT}(\bar{T}_j - T_{cont,j})\right) \frac{1}{\epsilon} \frac{\bar{p}(\text{mb})}{1013.25 \frac{\text{mb}}{\text{atm}}} \frac{10\Delta p}{g} q_m \delta_{jm}$$

$$\texttt{dtcont} = \frac{\partial K_{H_2O_{i,j}}}{\partial \bar{T}_m} = k_i \frac{dk}{dT} \frac{1}{\epsilon} \frac{\bar{p}(\text{mb})}{1013.25 \frac{\text{mb}}{\text{atm}}} \frac{10\Delta p}{g} q_k^2 \delta_{jm}$$

$$\texttt{dqtauh2o} = \frac{1}{\tilde{\tau}_{H_2O_{i,j,k}}} \frac{\partial \tilde{\tau}_{H_2O_{i,j,k}}}{\partial q_m}$$

$$= -D_{i,j,k}\{1 - E_{i,k}(\bar{T}_j - 273)\} M_i u_{q_j}^{M_i - 1} \delta_{jm} - \sec(\theta_k) \frac{\partial K_{H_2O_{i,j}}}{\partial T_m}$$

$$\texttt{dttauh2o} = \frac{1}{\tilde{\tau}_{H_2O_{i,j,k}}} \frac{\partial \tilde{\tau}_{H_2O_{i,j,k}}}{\partial \bar{T}_m} = D_{i,j,k} E_{i,k} u_{q_j}^{M_i} \delta_{jm} - \sec(\theta_k) \frac{\partial K_{H_2O_{im}}}{\partial \bar{T}_m}$$

5.1.3 Ozone contribution: tovsozo_tauK

The calculation of the derivatives of the ozone contribution to mean layer transmittance is similar to that for the water vapor contribution. The derivatives calculated are:

$$\texttt{dutauozo} = \frac{1}{\tilde{\tau}_{O_{3i,j,k}}} \frac{\partial \tilde{\tau}_{O_{3i,j,k}}}{\partial q_m} = -F_{i,j,k}\{1 - G_{i,k}(\bar{T}_j - 273)\} N_i u_{O_{3_j}}^{N_i - 1} \delta_{jm}$$

$$\texttt{dttauozo} = \frac{1}{\tilde{\tau}_{O_{3i,j,k}}} \frac{\partial \tilde{\tau}_{O_{3i,j,k}}}{\partial \bar{T}_m} = F_{i,j,k} G_{i,k} u_{O_{3_j}}^{N_i} \delta_{jm}$$

5.1.4 Transmittance product: tovs_tauprodK

In the modules for the forward radiance problem, the product of the effective layer transmittance contributions from fixed gases, water vapor and ozone is calculated in the main driver to obtain the layer transmittances. We calculate the derivative of that product in

this module. The derivatives for the layer transmittances were given in (54), (55) and (66). The relevant variables are

$$\texttt{dqtauang} = \frac{1}{\tilde{\tau}_{i,j,k}} \frac{\partial \tilde{\tau}_{i,j,k}}{\partial q_m} = \frac{1}{\tilde{\tau}_{W_{i,m,k}}} \frac{\partial \tilde{\tau}_{W_{i,m,k}}}{\partial u_m} \delta_{jm}$$

$$\texttt{dutauang} = \frac{1}{\tilde{\tau}_{i,j,k}} \frac{\partial \tilde{\tau}_{i,j,k}}{\partial u_{O_{3m}}} = \frac{1}{\tilde{\tau}_{O_{i,m,k}}} \frac{\partial \tilde{\tau}_{O_{i,m,k}}}{\partial u_m} \delta_{jm}$$

$$\texttt{dttauang} = \frac{1}{\tilde{\tau}_{i,j,k}} \frac{\partial \tilde{\tau}_{i,j,k}}{\partial \bar{T}_m}$$

$$= \begin{cases} 0 & j < m \\ \frac{1}{\tilde{\tau}_{F_{i,j,k}}} \frac{\partial \tilde{\tau}_{F_{i,j,k}}}{\partial \bar{T}_m} + \frac{1}{\tilde{\tau}_{O_{i,j,k}}} \frac{\partial \tilde{\tau}_{O_{i,j,k}}}{\partial \bar{T}_m} + \frac{1}{\tilde{\tau}_{W_{i,j,k}}} \frac{\partial \tilde{\tau}_{W_{i,j,k}}}{\partial \bar{T}_m} & j = m \\ \frac{1}{\tilde{\tau}_{F_{i,j,k}}} \frac{\partial \tilde{\tau}_{F_{i,j,k}}}{\partial \bar{T}_m} & j > m \end{cases}$$

5.1.5 Zenith angle interpolation: `tovs_tauangK`

In this routine the layer transmittances at the rapid algorithm zenith angles are interpolated to the desired zenith angle, and the product of the layer transmittances is taken to obtain the total transmittance between a given level and the satellite. The derivative of the interpolated layer transmittance is obtained by interpolating the derivatives of transmittance at the rapid algorithm angles. The derivative of the product of the layer transmittances is the sum of the logarithmic derivatives of layer transmittance. See equations (56) and (68). The variables used in the code are:

$$\texttt{tauint} = \frac{\partial \tilde{\tau}_{i,j}(\theta)}{\partial Q_m} = (1-\alpha) \frac{\partial \tilde{\tau}_{i,j,k}}{\partial Q_m} + \alpha \frac{\partial \tilde{\tau}_{i,j,k+1}}{\partial Q_m}$$

$$\texttt{dqtau} = \frac{1}{\tau_{i,l}(\theta)} \frac{\partial \tau_{i,l}(\theta)}{\partial q_m} = \frac{1}{\tilde{\tau}_{i,m}(\theta)} \left\{ (1-\alpha) \frac{\partial \tilde{\tau}_{i,m,k}}{\partial q_m} + \alpha \frac{\partial \tilde{\tau}_{i,m,k+1}}{\partial q_m} \right\}$$

$$\texttt{dutau} = \frac{1}{\tau_{i,l}(\theta)} \frac{\partial \tau_{i,l}(\theta)}{\partial u_m} = \frac{1}{\tilde{\tau}_{i,m}(\theta)} \left\{ (1-\alpha) \frac{\partial \tilde{\tau}_{i,m,k}}{\partial u_m} + \alpha \frac{\partial \tilde{\tau}_{i,m,k+1}}{\partial u_m} \right\}$$

$$\texttt{dttau} = \frac{1}{\tau_{i,l}(\theta)} \frac{\partial \tau_{i,l}(\theta)}{\partial \bar{T}_m} = \sum_{j=m}^{l} \frac{1}{\tilde{\tau}_{i,j}(\theta)} \left\{ (1-\alpha) \frac{\partial \tilde{\tau}_{i,j,k}}{\partial \bar{T}_m} + \alpha \frac{\partial \tilde{\tau}_{i,j,k+1}}{\partial \bar{T}_m} \right\}$$

5.1.6 Jacobian calculation: `dTb_dTQU`

The final step in the Jacobian calculation is to sum the transmittance derivatives (and for temperature, the derivatives of the Planck function) in the vertical and add derivatives of surface terms to obtain derivatives of radiance and brightness temperature with respect to the input temperature, water vapor, and ozone profiles. Relevant equations include (58) and (69), as well as the reflected radiance derivative terms discussed in (3.2)–(3.2).

The calculation of the vertical sums involves terms such as

`dTb_dq(l) = dTb_dq(l) -`	`btav *`	`tau(l) *`	`dqtau(l)`
	$B_i(\bar{T}_l)$	$\tau_{i,l}(\theta)$	$\frac{1}{\tau_{i,l}(\theta)} \frac{\partial \tau_{i,l}(\theta)}{\partial q_l}$

```
dTb_dTav(m) = dTb_dTav(m) +    btav *   ( tau(l-1) *      dttau(m+nm1)
```

$$B_i(\bar{T}_l) \qquad \tau_{i,l-1}(\theta) \qquad \frac{1}{\tau_{i,l-1}(\theta)} \frac{\partial \tau_{i,l-1}(\theta)}{\partial \bar{T}_m}$$

```
                             -  tau(l) *    dttau(m+n) )
```

$$\tau_{i,l}(\theta) \qquad \frac{\partial \tau_{i,l}(\theta)}{\partial \bar{T}_m}$$

Remember that terms such as **dttau** and **dqtau** are logarithmic derivatives and thus have the derivative multiplied by $1/\tau$.

Acknowledgments

Thanks are due to Paul Piraino and Dr. Joel Susskind of the Sounder Research Team for supplying the GLA retrieval code and other help. Joanna Joiner and Dave Lamich offered advice and guidance while the coding of the forward modules and Jacobians was being performed. Thanks are also due to Arlindo daSilva for LaTeX help.

A formal technical review of this document and code was conducted on 11 July 1996. My thanks to Jing Guo (review leader), Dave Ledvina (recorder), and Runhua Yang and Alice Trenholme (reviewers) for their suggestions and corrections.

References

Bignell, K.J., 1970: The water-vapour infra-red continuum. *Quart. J. Roy. Meteor. Soc.*,**96**, 390–403.

Eyre, J.R., and H.M. Woolf, 1988: Transmittance of atmospheric gases in the microwave region: a fast model. *Applied Optics*, **27**, 3244–3249.

Joiner, J., and A.M. daSilva, 1996: Efficient methods to assimilate satellite retrievals based on information content. *DAO Office Note 96-06.* Data Assimilation Office, Goddard Space Flight Center, Greenbelt, MD 20771.

Kornfield, J., and J. Susskind, 1977: On the effect of surface emissivity on temperature retrievals. *Mon. Wea. Rev.*, **105**, 1605–1608.

McMillan, L.M., and H.E. Fleming, 1976: Atmospheric transmittance of an absorbing gas: a computationally fast and accurate transmittance model for absorbing gases with constant mixing ratios in inhomogeneous atmospheres. *Applied Optics*, **15**, 358–363.

Susskind, J., and J.E. Searl, 1977: Synthetic atmospheric transmittance spectra near 15 and 4.3 μm. *J. Quant. Spectrosc. Radiat. Transfer*,**19**, 195–215.

Susskind, J., J. Rosenfield, and D. Reuter, 1983: An accurate radiative transfer model for use in the direct physical inversion of HIRS2 and MSU temperature sounding data. *J. Geophys. Res.*, **88**, 8550–8568.

Susskind, J., J. Rosenfield, and D. Reuter, 1984: Remote sensing of weather and climate parameters from HIRS2/MSU on TIROS–N. *J. Geophys. Res.*,**89**, 4677–4697.

Weinreb, M.P., H.E. Fleming, L.M. McMillin and A.C. Neuendorffer, 1981: Transmittances for the TIROS Operational Vertical Sounder. *NOAA Tech. Report* NESS 85, National Oceanic and Atmospheric Administration, Washington D.C.

A Forward radiance modules prologues and source code

Sample program

```
      program sample

      implicit none
c
c  sample program for GLA forward model and Jacobian
c
      include "tovsparam.h"

      integer      l                ! loop counter

      character*80 trcoefs          ! transmittance coef. file name

c  inputs/outputs for initialGLA
      integer      itrfdat          ! transmittance coef. unit number
      character*2  satelite         ! satellite ID
      logical      bland(360,180)   ! array of land/water flags
      integer      ierr             ! error flag


c  inputs for tovs_tau
c  atmospheric values are at the 71 (maxlev) rapid algorithm pressure levels
c
      real      tair(maxlev)        !  temperature (K)
      real      tmptop              !  temperature at "top of atmosphere"
      real      h2omid(maxlev)      !  layer specific humidity (kg/kg)
      real      ozomid(maxlev)      !  ozone column density (g/cm**2)
      real      ps                  !  surface pressure (mb)
      integer   nchan               !  number of channels to calculate
      integer   chanarr(maxcha)     !  array of channel numbers
      integer   nlev                !  number of levels in input sounding
      real      mw_zen              !  zenith angle for microwave channels
      real      ir_zen              !  zenith angle for infrared channels
      logical   dayflag             !  'true' if daytime
      real      secsun              !  1. / cos( solar zenith angle )

c  outputs from tovs_tau, inputs to hirsrad
      real      tau(maxlev, maxcha) !  transmittance along path to satellite
      real      tausun(maxcha)      !  transmittance along path from sun
      logical   errflag             !  error return flag

c  other inputs to hirsrad
      real      emissmw             !  microwave emissivity
      logical   land                !  land/water flag
      real      rho                 !  bidirectional reflectance
      real      sunzang             !  solar zenith angle
      real      tground             !  surface skin temperature
      real      tsurfair            !  surface air temperature
```

```
c  output from hirsrad
      real       calc_rad(maxcha)     !  calculated radiances

c  arrays for Jacobian calculation
      real dTbdt(maxlev,maxcha)       !  deriv. wrt. input temperature
      real dTbdq(maxlev,maxcha)       !  deriv. wrt. input specific humidity
      real dTbdu(maxlev,maxcha)       !  deriv. wrt. input column density

      real btnew(maxcha)              !  calculated brightness temperature

      data itrfdat /23/

c
c  set satellite parameters, read in transmittance coefficient dataset
c    (in initialGLA)

      satelite = 'NH'
      trcoefs = 'trcoefs8.nh.ieee'

      open(itrfdat,file=trcoefs,status='old',form='unformatted')

      call initialGLA(itrfdat,satelite,bland,ierr)

      close(itrfdat)

      if (ierr .ne. 0 ) then
         print *,' error returned from initialGLA ',ierr
         stop
      endif

c
c  input values for forward (and Jacobian) calculation

      print *,'input zenith angles to try'
      read(5,*) ir_zen,mw_zen

      print *,'input nchan (number of channels) and channel array'
      read(5,*) nchan,(chanarr(l),l=1,nchan)

      print *,'input number of levels in sounding, then T, H2O and O3'

      read(5,*) nlev

      do l = 1,nlev
         read(5,*) tair(l), h2omid(l), ozomid(l)
      enddo

      print *,'input surface pressure and top level temperature'
      read(5,*) ps, tmptop

      print *,'input surface air and ground (skin) temperature'
      read(5,*) tsurfair, tground

      print *,'input solar zenith angle, and land/water flag'
```

```
 read(5,*) sunzang, land

 if (sunzang .lt. 89.) dayflag = .true.
 secsun = 1. / cos (sunzang * acos(0.) / 90.)

 print *, 'input bidirectional reflectance and sfc. MW emissivity'
 read(5,*)  rho, emissmw

 call tovs_tau(tair, tmptop, h2omid, ozomid, ps, nchan,
&        chanarr, nlev, mw_zen, ir_zen, dayflag, secsun, tau,
&        tausun, errflag)

 call hirsrad(tau,tausun,nchan,chanarr,tair,tmptop,ir_zen,
&        emissmw, land, nlev, rho, sunzang, dayflag, tground,
&        tsurfair, calc_rad)

 call tovs_tauK(tair, tmptop, h2omid, ozomid, ps, nchan,
&     chanarr, nlev, mw_zen, ir_zen, dayflag, secsun,
&     emissmw, land, rho,  tground, tsurfair,
&     tau, tausun, btnew, dTbdT, dTbdq, dTbdu, errflag)

 stop
 end
```

Sample Makefile

```
#  sample Makefile  --- shows dependencies of files

INITIAL = chacon.o dattrf.o grdcon.o matcon.o phycon.o
TAUCALC = tovs_tau.o tovsfix_tau.o tovsh2o_tau.o tovsozo_tau.o tovs_tauang.o

MODULES = sunang.o hirsrad.o brtemp.o planck.o $(INITIAL) $(TAUCALC)

KMODULES = tovsh2o_tauK.o tovsozo_tauK.o tovsfix_tauK.o tovs_tauK.o \
                tovs_tauprodK.o tovs_tauangK.o dplanck.o dTb_dTQU.o

librad.a         : $(MODULES)
ar -r librad.a $(MODULES)

libradK.a         : $(KMODULES)
ar -r libradK.a $(KMODULES)

chacon.o       : tovsparam.h chacons.h phycons.h
dattrf.o       : tovsparam.h chacons.h matcons.h phycons.h taucoef.h
grdcon.o       : tovsparam.h grdcons.h
hirsrad.o      : tovsparam.h phycons.h chacons.h matcons.h
matcon.o       : matcons.h
phycon.o       : matcons.h phycons.h
sunang.o       : phycons.h matcons.h
tovs_tau.o     : tovsparam.h matcons.h grdcons.h chacons.h \
                  tovsfix_tau.o tovsh2o_tau.o tovsozo_tau.o tovs_tauang.o
tovs_tauang.o  : tovsparam.h taucoef.h grdcons.h chacons.h
tovsfix_tau.o  : tovsparam.h taucoef.h
tovsh2o_tau.o  : tovsparam.h taucoef.h phycons.h chacons.h grdcons.h
tovsozo_tau.o  : tovsparam.h taucoef.h

tovs_tauK.o  : tovsparam.h matcons.h grdcons.h chacons.h taucoef.h \
                  tovsfix_tauK.o tovsh2o_tauK.o tovsozo_tauK.o \
                  tovs_tauangK.o
tovsfix_tauK.o : tovsparam.h taucoef.h
tovsh2o_tauK.o : tovsparam.h taucoef.h phycons.h chacons.h grdcons.h
tovsozo_tauK.o : tovsparam.h taucoef.h
tovs_tauprodK.o : tovsparam.h
dTb_dTQU.o      : tovsparam.h phycons.h chacons.h matcons.h \
                   planck.o dplanck.o brtemp.o
```

"Include" files

Parameter definitions

tovsparam.h – parameter definitions

```
!      beginning of include file   tovsparam.h
!
!    used in nearly all modules
!
!  parameters  for GLA TOVS rapid algorithm
!
      integer maxlev       !  maximum number of levels
      integer maxangs      !  maximum number of angles
      integer maxcha       !  maximum number of channels

      parameter( maxlev=71, maxangs=3, maxcha=24)

!      end of include file   tovsparam.h
```

Constants

chacons.h – channel constants

```
!      beginning of include file   chacons.h
!-----------------------------------------------------------------------
!     initialized in:   chacon
!     used in:   dattrf hirsrad tovs_tau tovs_tauang tovsh2o_tau
!                tovs_tauK  tovsh2o_tauK dTb_dTQU
!
!     channel constants used in GLA TOVS retrievals
!-----------------------------------------------------------------------
      character*8   chanid       !  channel identifier
      logical       irchan       !  true if ir channel
      logical       mwchan       !  true if mw channel
      logical       use_refl     !  true if use integrated reflected IR in
                                 !       downward flux calculation
      real          abscof       !  absorp. coeff for H2O continuum
      real          freq         !  frequency (cm^-1 or ghz)
      real          chscatt      !  aerosol scattering & absorp. coeff.
      real          contmp       !  reference T for H2O continuum
      real          degchn       !  dK/dT  for H2O continuum
      real          dfxcon       !  downward flux coefficient
      real          em_land      !  ir emissivity, land
      real          em_water     !  ir emissivity, water
      real          planck1      !  const for planck function
      real          planck2      !  const for planck function

      common /chacons/  chanid(maxcha), irchan(maxcha), mwchan(maxcha)
```

```
     &,use_refl(maxcha), abscof(maxcha), freq(maxcha), chscatt(maxcha)
     &,contmp(maxcha), degchn(maxcha), dfxcon(maxcha), em_land(maxcha)
     &,em_water(maxcha), planck1(maxcha), planck2(maxcha)
!
!      end of include file   chacons.h
```

grdcons.h – vertical grid constants

```
!      beginning of include file  grdcons.h

!  ------------------------------------------------------------------
!       initialized in:  grdcon
!       used in:   tovs_tau tovs_tauang tovsh2o_tau
!           tovs_tauK  tovsh2o_tauK
!
!       vertical grid constants used in GLA retrievals
!
!    18May96  Meta S.  added 'delprs' to common
!  ------------------------------------------------------------------

      real  pres          !  pressures at level boundaries for rapid alg.
      real  presa         !  average layer pressures for rapid algorithm
      real  presln        !  logarithm of pressure
      real  delprs        !  layer difference in pressure * 10 / g
      real scat           !  coeff. for scattering calculation
      real stntmp         !  standard temperature profile
      real tstd_av        !  average layer temperature, std. profile

      common /grdcons/
     &     pres( maxlev ), presa( maxlev ), presln( maxlev )
     & ,   delprs( maxlev), scat( maxlev ), stntmp( maxlev )
     & ,   tstd_av   ( maxlev )

!      end of include file  grdcons.h
```

matcons.h – mathematical constants

```
!      beginning of include file   matcons.h
!*********************************************************************
!
!   initialized in:  matcon
!   used in:   dattrf hirsrad sunang tovs_tau tovs_tauang
!              tovs_tauK  dTb_dTQU
!
!  mathematical constants as used in gla retrievals
!*********************************************************************
!
      real         deg2rad      !  convert degrees to radians
      real         pi           !   pi
      real         pid2         !   pi/2
```

```
      real         rad2deg      !  convert radians to degrees
      real         twopi        !  2 * pi

      common /matcons/  deg2rad, pi, pid2, rad2deg, twopi

!
!*********************************************************************
!      end of include file   matcons.h
```

phycons.h – physical constants

```
!      beginning of include file   phycons.h
!*********************************************************************
!
!   initialized in:  phycon
!   used in:   chacon dattrf hirsrad sunang tovs_h2o_tau
!              tovsh2o_tauK  dTb_dTQU
!
!   physical constants used in GLA retrievals
!
!   18May96  Meta S.  added 'gravcon', 'econ' to common
!*********************************************************************
      real         avogad       !  Avogadro's number
      real         boltzmns     !  Boltzmann's constant (joule/K)
      real         plancks      !  Planck constant (joule sec)
      real         atmosph      !  standard atmosphere (mb)
      real         bigbang      !  Cosmic background bright. temp. (K)
      real         grav         !  acceleration from gravity (m/s)
      real         clight       !  velocity of light (cm/s)
      real         daymaxd      !  date of maximum solar declin. angle
      real         days_per_yr  !  no. of days per year (non-leap year)
      real         declmax      !  max solar declination angle
      real         plcon1       !  planck funct. const #1 (see in
      real         plcon2       !  planck funct. const #2    phycon.f)
      real         scon         !  pi * (Rsun/Rearth-orbit)^2  steradians
      real         shmin        !  Minimum specific humid. for rapid alg
      real         sunset       !  limiting solar zen angle for sun calc.
      real         tsun         !  Sun temperature
      real         airmol       !  molecular weight of air
      real         h2omol       !  molecular weight of H2O
      real         ozomol       !  molecular weight of O3
      real         gravcon      !  10 / grav  - used in thickness calc.
      real         econ         !  (airmol/h2omol) /1013.25

      common /phycons/  avogad,  boltzmns, plancks, atmosph,  bigbang
     &     , grav , clight , daymaxd , days_per_yr
     &     , declmax , plcon1 , plcon2 ,  scon, shmin, sunset
     &     , tsun , airmol , h2omol , ozomol, gravcon, econ

!      end of include file   phycons.h
```

taucoef.h – transmittance coefficients

```
!      beginning of include file   taucoef.h
!------------------------------------------------------------------------
!       initialized in:  dattrf
!       used in: tovs_tauang tovsfix_tau tovsh2o_tau tovsozo_tau
!          tovs_tauK  tovsfix_tauK  tovsh2o_tauK  tovsozo_tauK
!
!       coefs for rapid algorithm computation of trans funs
!------------------------------------------------------------------------
      common /taucoef/
     $   taucfa  ( maxlev,  maxangs, maxcha )
     $ , taucfb  ( maxlev,  maxangs, maxcha )
     $ , taucfc  ( maxlev,  maxangs, maxcha )
     $ , taucfd  ( maxlev,  maxangs, maxcha )
     $ , taucfe           ( maxangs, maxcha )
     $ , taucff  ( maxlev,  maxangs, maxcha )
     $ , taucfg           ( maxangs, maxcha )
     $ , taucfm                    ( maxcha )
     $ , taucfn                    ( maxcha )
     $ , taucfw  ( maxlev,  maxangs, maxcha )
     $ , numangs
     $ , angle            ( maxangs )
     $ , secang           ( maxangs )
      double precision taucfa, taucfb, taucfc, taucfd, taucfe, taucff,
     $       taucfg, taucfm, taucfn, taucfw
      real angle
      real secang
      integer numangs
!**************************************************************************
!
!  tau_fixed = a + b (t - to) + c (t* - t*o)
!  tau_h2o   = exp ( - ( d (1 - e (t - 273)) h2omid**m) + continuum term
!  tau_ozone = exp ( - ( f (1 - g (t - 273)) ozomid**n)
!
!**************************************************************************
!      end of include file   taucoef.h
```

Fortran code

brtemp – Calculate brightness temperature

```
!***********************************************************************
!!BRTEMP
!***********************************************************************
!
!!ROUTINE: brtemp
!
!!DESCRIPTION:  COMPUTES BRIGHTNESS TEMPERATURE FROM RADIANCE VIA
!               PLANCK FUNCTION.  Derived from GLA retrieval code
!
! SYSTEM ROUTINES USED: none
!
! SUBROUTINES CALLED: none
!
!!INPUT PARAMETERS:
!     RAD     real RADIANCE FOR TEMPERATURE COMPUTATION
!                     UNITS WATTS/(CM**2-SR)
!     PLANK1  real WAVE NUMBER DEPENDENT CONSTANT
!                     PLANK1 = 2 * VELLIGHT**2 * PLANCKCON / LAMDA**3
!                     PLANK1 = 1.193E-12/LAMDA**3  WATTS / (CM**2 - SR)
!                        WHERE
!                     VELLIGHT = 3.00E+10 CM/SEC
!                     PLANKCON = 6.63E-34  JOULES-SEC
!                     LAMDA    = WAVE LENGTH IN CM
!     PLANK2  real WAVE NUMBER DEPENDENT CONSTANT
!                     PLANK2 = PLANCKCON*VELLIGHT / (BOLTZMAN*LAMDA)
!                     PLANK2 = 1.441/LAMDA   DEG.K
!                        WHERE
!                     BOLTZMAN = 1.38E-23 JOULE / DEG.K
!!OUTPUT PARAMETERS:
!     BT      real  Brightness temperature (K)
!
!!REVISION HISTORY:
!     25OCT84 JIMPF   original program in GLA TOVS retrievals
!     11DEC89 SYLEE   UNVECTORIZED FULL PRECISION VERSION
!     26SEP95 Meta S. converted to DAO standard (implicit none)
!     10OCT95 Meta S. revised prologue to DAO standard
!
!***********************************************************************
      subroutine brtemp ( rad, plank1, plank2, bt )
      implicit none
      real     bt
      real     plank1
      real     plank2
      real     rad
!***********************************************************************
!  calculate brightness temperature
!***********************************************************************
         bt        = plank1 / rad
```

```
   bt          = bt + 1.0
   bt          = log ( bt )
   bt          = plank2 / bt
return
end
```

chacon – Set channel constants

```
!=======================================================================
!!CHACON
!=======================================================================
!
!!ROUTINE:  chacon
!
!!DESCRIPTION:  Set channel constants for TOVS (HIRS/MSU) channels
!               Based on GLA TOVS retrieval code (chacon,
!                  addcha, and related subroutines)
!
! SYSTEM ROUTINES USED: none
!
! SUBROUTINES CALLED: none
!
!!INPUT PARAMETERS:
!     satelite - character*2 identifier for satellite
!
!!OUTPUT PARAMETERS: none
!
!!REVISION HISTORY:
!  25sep95  Meta S.  original routine derived from subroutines
!                    CHACON and ADDCHA in GLA retrieval code
!  20feb96  Meta S.  added central frequencies for NOAA-12 (ND)
!                    (Note, IR20 given dummy frequency - not listed
!                      in NOAA12 docs... wouldn't use IR20 anyway.)
! NOTES:
!        will need some revision to restore multiple satellites
!=======================================================================
      subroutine chacon (satelite)
!=======================================================================
      implicit none

!=======================================================================
!     input variables
      character*2 satelite      ! identifier for satellite to use

!=======================================================================

      include "tovsparam.h"
      include "chacons.h"
      include "phycons.h"

!=======================================================================
!
!     local variables
!
!=======================================================================
      integer ic
      real tplcon1, tplcon2

!=======================================================================
```

```
! values for HIRS channels
!
!  satellite dependent central frequencies of channels
      real    freqTN(20), freqNA(20), freqNC(20), freqNE(20) !frequency
      real    freqNF(20), freqNG(20), freqNH(20), freqND(20) ! cm^-1

      real    freqHRS (20)        !   frequency (cm^-1)
      real    abscof_hrs(20)      !   absorp. coeff for H20 continuum
      real    degchn_hrs(20)      !   dk/dT  for H20 continuum
      real    contemp_hrs(20)     !   ref T  for H20 continuum
      real    chscat_hrs(20)      !   aerosol scattering & absorp. coeff.
      real    em_land_hrs(20)     !   emissivity for land
      real    em_water_hrs(20)    !   emissivity for water
      real    dfx_hrs(20)         !   downward flux coefficient

! values for MSU channels
      real    freqMSU(4)          !   frequency (GHz)
      real    abscof_msu(4)       !   absorp. coeff for H20 continuum
      real    degchn_msu(4)       !   dk/dT  for H20 continuum
      real    contemp_msu(4)      !   ref T  for H20 continuum
      real    chscat_msu(4)       !   aerosol scattering & absorp. coeff.
      real    em_land_msu(4)      !   emissivity for land  (not used?)
      real    em_water_msu(4)     !   emissivity for water (not used?)
      real    dfx_msu(4)


      character*6 chans(24)       !   channel identifier

!-----------------------------------------------------------------------
!                  HIRS/2  FREQUENCIES AND CHANNEL ID
!                 TN(TIROS-N), NA(NOAA-6), NC(NOAA-7),
!                 NE(NOAA-8),  NF(NOAA-9), NG(NOAA-10),
!                 NH(NOAA-11), ND(NOAA-12)
!-----------------------------------------------------------------------
      DATA   freqTN
     *         /  668.70,  679.05,  689.70,  703.80,  716.70,
     *            731.85,  749.50,  900.00, 1030.00, 1225.00,
     *           1365.00, 1488.00, 2192.50, 2211.65, 2237.35,
     *           2271.20, 2308.85, 2512.00, 2660.00,14500.00/
      DATA   freqNA
     *         /  667.95,  679.34,  689.99,  704.63,  717.75,
     *            732.30,  749.16,  899.94, 1027.55, 1222.98,
     *           1368.50, 1481.08, 2190.48, 2210.91, 2238.62,
     *           2269.65, 2360.94, 2515.28, 2649.97,14500.00/
      DATA   freqNC
     *         /  668.70,  681.00,  692.00,  705.00,  718.00,
     *            734.00,  753.00,  904.00, 1028.00, 1229.00,
     *           1360.00, 1483.00, 2181.00, 2211.00, 2241.00,
     *           2260.00, 2360.00, 2515.00, 2660.00,14500.00/
      DATA   freqND
     *         /  667.58,  680.18,  690.01,  704.22,  716.32,
     *            732.81,  751.92,  900.45, 1026.66, 1223.44,
     *           1368.68, 1478.59, 2190.37, 2210.51, 2236.62,
```

```
*          2267.62, 2361.64, 2514.68, 2653.48,14500.00/
 DATA   freqNE
*         /  667.41,  679.45,  690.90,  702.97,  717.56,
*            732.97,  747.90,  901.08, 1027.11, 1224.05,
*           1366.17, 1486.92, 2189.28, 2211.71, 2238.06,
*           2271.43, 2357.11, 2515.53, 2661.85,14355.00/
 DATA   freqNF
*         /  667.67,  679.84,  691.46,  703.37,  717.16,
*            732.64,  749.40,  898.53, 1031.61, 1224.74,
*           1365.12, 1483.24, 2189.97, 2209.18, 2243.14,
*           2276.46, 2359.05, 2518.14, 2667.80,14549.27/
 DATA   freqNG
*         /  667.70,  680.23,  691.15,  704.33,  716.30,
*            733.13,  750.72,  899.50, 1029.01, 1224.07,
*           1363.32, 1489.42, 2191.38, 2208.74, 2237.49,
*           2269.09, 2360.00, 2514.58, 2665.38,14453.14/
 DATA   freqNH
*         /  668.99,  678.89,  689.70,  703.25,  716.83,
*            732.11,  749.48,  900.51, 1031.19,  795.69,
*           1361.10, 1479.86, 2189.94, 2209.66, 2239.26,
*           2267.80, 2416.32, 2511.83, 2664.07,14453.14/

!*********************************************************************
!---------------------------------------------------------------------
!                        HIRS/2 CHANNEL CONSTANTS
!---------------------------------------------------------------------

 DATA   abscof_hrs
*/30.32232,  28.83525,  27.34284,  25.63885,  24.13291,
* 22.40358,  20.61069,   8.87156,   6.77743,   5.533819,
*  0.0,       0.0,       0.2740922, 0.2529629, 0.2265023,
*  0.1949478, 0.0,       0.06503328,0.06404681,0.0      /
 DATA   degchn_hrs
*/ .02,       .02,       .02,       .02,       .02,
*  .02,       .02,       .02,       .02,       .02,
*  .0,        .0,        .010404,   .0103655, .01030173,
*  .01019153, .009672762,.007432,   .0055279, .0        /
 DATA   contemp_hrs
*         /  303.00,  303.00,  303.00,  303.00,  303.00,
*            303.00,  303.00,  303.00,  296.00,  296.00,
*              0.00,    0.00,  338.00,  338.00,  338.00,
*            338.00,    0.00,  338.00,  338.00,    0.00/
 DATA   chscat_hrs
*         /    1.00,    1.00,    1.00,    1.00,    1.00,
*              1.00,    1.00,    1.00,    1.00,    1.00,
*              1.00,    1.00,    3.00,    3.00,    3.00,
*              3.00,    3.00,    3.00,    3.00,    0.00/
 DATA   em_land_hrs
*         /    0.95,    0.95,    0.95,    0.95,    0.95,
*              0.95,    0.95,    0.95,    0.95,    0.90,
*              0.90,    0.90,    0.85,    0.85,    0.85,
*              0.85,    0.85,    0.85,    0.85,    0.00/
 DATA   em_water_hrs
*         /    0.98,    0.98,    0.98,    0.98,    0.98,
```

```
     *             0.98,    0.98,    0.98,    0.98,    0.98,
     *             0.98,    0.98,    0.95,    0.95,    0.95,
     *             0.95,    0.95,    0.95,    0.95,    0.00/
      DATA   dfx_hrs
     *        /    1.00,    1.00,    1.00,    1.00,    1.00,
     *             0.71,    0.49,    1.00,    1.00,    1.00,
     *             1.00,    1.00,    0.42,    0.55,    1.00,
     *             1.00,    1.00,    0.95,    0.95,    1.00/
!**********************************************************************
!----------------------------------------------------------------------
!                        MSU  CHANNEL CONSTANTS
!----------------------------------------------------------------------

      DATA   freqMSU        / 50.30,   53.74,   54.96,   57.95  /
      DATA   abscof_msu     / 4*0.00 /
      DATA   degchn_msu     / 4*0.00 /
      DATA   contemp_msu     / 4*0.00 /
      DATA   chscat_msu      / 4*0.00 /
      DATA   em_land_msu      / 4*1.00 /
      DATA   em_water_msu      / 4*1.00 /
      DATA   dfx_msu     / 4*0.95 /

      data chans / 'IR(01)','IR(02)','IR(03)','IR(04)','IR(05)',
     &  'IR(06)','IR(07)','IR(08)','IR(09)','IR(10)','IR(11)',
     &  'IR(12)','IR(13)','IR(14)','IR(15)','IR(16)','IR(17)','IR(18)',
     &  'IR(19)','IR(20)','MW(01)','MW(02)','MW(03)','MW(04)'/


!----------------------------------------------------------------------
!    set up channel id's (used in identifying coefficients in 'dattrf')
!----------------------------------------------------------------------

      do ic = 1,24
         write(chanid(ic),'(a2,a6)') satelite,chans(ic)
      enddo

!----------------------------------------------------------------------
!               SET THE INSTRUMENTS ( TOVS ONLY FOR NOW )
!----------------------------------------------------------------------

         IF ( satelite .EQ. 'TN' ) THEN
            DO ic   = 1, 20
               freqHRS(ic)  = freqTN(ic)
            enddo
         ELSE IF ( satelite .EQ. 'NA' ) THEN
            DO ic   = 1, 20
               freqHRS(ic)  = freqNA(ic)
            enddo
         ELSE IF ( satelite .EQ. 'NC' ) THEN
            DO ic   = 1, 20
               freqHRS(ic)  = freqNC(ic)
            enddo
         ELSE IF ( satelite .EQ. 'ND' ) THEN
            DO ic   = 1, 20
```

```
            freqHRS(ic)  = freqND(ic)
         enddo
      ELSE IF ( satelite .EQ. 'NE' ) THEN
         DO ic   = 1, 20
            freqHRS(ic)  = freqNE(ic)
         enddo
      ELSE IF ( satelite .EQ. 'NF' ) THEN
         DO ic   = 1, 20
            freqHRS(ic)  = freqNF(ic)
         enddo
      ELSE IF ( satelite .EQ. 'NG' ) THEN
         DO ic   = 1, 20
            freqHRS(ic)  = freqNG(ic)
         enddo
      ELSE IF ( satelite .EQ. 'NH' ) THEN
         DO ic   = 1, 20
            freqHRS(ic)  = freqNH(ic)
         enddo
      ELSE
         PRINT *, 'CHACON: UNKNOWN SATELLITE'
         STOP 12
      END IF
!
!-----------------------------------------------------------------------
!
!    FILL IN VALUES FOR INFRARED CHANNELS , slots 1-20
!
!-----------------------------------------------------------------------
      DO ic   = 1, 20
         freq(ic)= freqHRS(ic)
         IRCHAN(ic)    = .true.
         MWCHAN(ic)    = .false.
      enddo

      TPLCON1      = PLCON1
      TPLCON2      = PLCON2

      do ic   = 1, 20
         planck2(ic) = freq(ic)
         planck1(ic) = planck2(ic) * planck2(ic)
         planck1(ic) = tplcon1 * planck2(ic) * planck1(ic)
         planck2(ic) = tplcon2 * planck2(ic)
         abscof(ic)  = abscof_hrs(ic)
         degchn(ic)  = degchn_hrs(ic)
         contmp(ic)  = contemp_hrs(ic)
         chscatt(ic) = chscat_hrs(ic)
         em_land(ic)  = em_land_hrs(ic)
         em_water(ic)  = em_water_hrs(ic)
         dfxcon(ic)  = dfx_hrs(ic)
         use_refl(ic) = .false.
      enddo

c  "optically thin" HIRS channels which use integrated radiance (reflected
c     from surface) in downward flux calculation.
```

```
      use_refl(8) =.true.
      use_refl(10) =.true.
      use_refl(18) =.true.
      use_refl(19) =.true.

!-----------------------------------------------------------------------
!
!    FILL IN VALUES FOR MICROWAVE CHANNELS , slots 21-24
!
!-----------------------------------------------------------------------

      DO ic   = 1, 4
         freq(ic+20)= freqMSU(ic)
         IRCHAN(ic+20)    = .false.
         MWCHAN(ic+20)    = .true.
      enddo

!
!  GLA rets had remark about 'old gridding program' and filled in
!  different constants for PLCON1 and PLCON2;  this has the same
!  constants as for IR, but they are not used anyway.

      TPLCON1      = PLCON1
      TPLCON2      = PLCON2

      do ic   = 21, 24
         planck2(ic) = freq(ic)
         planck1(ic) = planck2(ic) * planck2(ic)
         planck1(ic) = tplcon1 * planck2(ic) * planck1(ic)
         planck2(ic) = tplcon2 * planck2(ic)
         abscof(ic)  = abscof_msu(ic-20)
         degchn(ic)  = degchn_msu(ic-20)
         contmp(ic)  = contemp_msu(ic-20)
         chscatt(ic) = chscat_msu(ic-20)
         em_land(ic)  = em_land_msu(ic-20)
         em_water(ic)  = em_water_msu(ic-20)
         dfxcon(ic)  = dfx_msu(ic-20)
         use_refl(ic) = .false.
      enddo
 100  continue

      return
!-----------------------------------------------------------------------
      end
```

dattrf.f – Set up transmittance coefficients

```
!*****************************************************************
!!DATTRF
!*****************************************************************
!
!!ROUTINE:  dattrf
!
!!DESCRIPTION:  Read rapid algorithm transmittance function coefficients
!               Fills common block /taucoef/
!
!               Based on GLA TOVS retrieval code (dattrf and
!                  related subroutines)
!
! SYSTEM ROUTINES USED:  dexp, dlog, cos
!
! SUBROUTINES CALLED: none
!
!  INPUT/OUTPUT FILES USED
!     TRCOEF  ITRFDAT  RAPID ALGORITH TRANS. FUN. DATA SET
!
!!INPUT PARAMETERS:
!  itrfdat   - integer unit number for input file
!
!!OUTPUT PARAMETERS:
!  ierr      - integer error flag    0 = successful
!                                    1 = error reading dataset
!                                    2 = fewer than 24 channels found
!
!!REVISION HISTORY:
!  22sep95  Meta S.  original routine derived from subroutine dattrf
!                    in GLA retrieval code
!  16apr96  Meta S.  slight modification, clarification, add error flag
!  13jul96  Meta S.  changed do-loops to remove statement numbers
!
!  REMARKS:
!     some changes will need to be made to accomodate multiple
!     satellites
!*****************************************************************
      subroutine dattrf(itrfdat, ierr)

      implicit none
      integer itrfdat
      integer ierr
!*****************************************************************
      include "tovsparam.h"

      include "chacons.h"

      include "matcons.h"

      include "phycons.h"
```

```
      include "taucoef.h"
!*********************************************************************
!  coefficients for rapid transmittance algorithm
!  zao, zbo, zco, zdo, zfo : "old" values read from file from OLDLEV level fit
!  za, zb, zc, zd, zf      :  transformed to maxlev level
!  ze, zg                  :  other coefficients
!
!  tau_fixed = A + B (T - To) + C (T* - T*o)              (ON-9608, eq26)
!  tau_h2o   = exp ( - ( D (1 - E (T - 273)) h2omid**M)   (ON-9608, eq32)
!  tau_ozone = exp ( - ( F (1 - G (T - 273)) ozomid**N)   (ON-9608, eq33)
!
!*********************************************************************
      integer oldlev
      parameter (oldlev = 66)
      CHARACTER*8       ZCHANID
      DIMENSION  ZA    (  maxlev  , maxangs  )
      DIMENSION zZA    (  maxlev  , maxangs  )
      DIMENSION  ZB    (  maxlev  , maxangs  )
      DIMENSION  ZC    (  maxlev  , maxangs  )
      DIMENSION  ZD    (  maxlev  , maxangs  )
      DIMENSION  ZE          ( maxangs  )
      DIMENSION  ZF    (  maxlev  , maxangs  )
      DIMENSION  ZG          ( maxangs  )
      DIMENSION  ZAO   (  OLDLEV  , maxangs  )
      DIMENSION  ZBO   (  OLDLEV  , maxangs  )
      DIMENSION  ZCO   (  OLDLEV  , maxangs  )
      DIMENSION  ZDO   (  OLDLEV  , maxangs  )
      DIMENSION  ZFO   (  OLDLEV  , maxangs  )
      DOUBLE PRECISION  ZA, ZB, ZC, ZD, ZE, ZM, ZF, ZG, ZN, ZQ,
     *                  ZAO, ZBO, ZCO, ZDO, ZFO,zza, zzm, zzn
      integer ii,jj,n,l,k
      integer nfound ! counter for no. of channels
      integer yang ! number of coeff. input angles
      integer ylev ! number of coeff. input levels


      ierr = 0

!*********************************************************************
!   set up angles used in rapid algorithm
!*********************************************************************
      NUMANGS      = 3

      ANGLE(1)     =  0.0
      ANGLE(2)     = 50.0
      ANGLE(3)     = 75.0

      DO  II   = 1, NUMANGS
         ANGLE(II)    = ANGLE(II) * DEG2RAD
         SECANG(II)   = COS( ANGLE(II) )
         SECANG(II)   = 1.0 / SECANG(II)
      enddo

!---------------------------------------------------------------------
```

```
!                  READ DATA FOR NEXT CHANNEL FROM TRCOEF
!-------------------------------------------------------------------------
      REWIND       ITRFDAT

      nfound       = 0

  100 CONTINUE

         READ  ( ITRFDAT, END=600,ERR=500) ZCHANID, N, YANG, N, YLEV,
     $                ZM, ( ZE(N), N = 1, YANG ),
     $                ZN, ( ZG(N), N = 1, YANG ),
     $             ( ( ZAO(L,N), ZBO(L,N), ZCO(L,N), ZDO(L,N), ZFO(L,N),
     $                                   L = 1, YLEV ), N = 1, YANG )

         if  ((YANG .NE. 3 ) .or.
     $        (YLEV .NE.  OLDLEV  ) )  go to 100
!
!  we have proper number of coefficients; convert from old 66 levels
!  to new 71 levels
!
!-------------------------------------------------------------------------
!      SHIFT OLDLEV(66) TO NEW MAXLEV(71) FOR ZA ZB ZC ZD ZF
!      coefficients zao, zbo, zco, zdo, zfo are at old 66 levels
!-------------------------------------------------------------------------
! the formula ln ZA(1) = .1 ln ZAO(1) for 0.1 MB
! the formula ln ZA(ii) = .3 ln ZAO(1) for 0.4, 0.7, 1.0 MB
! the formula ln ZA(5) = .3 ln ZAO(2) for 1.3 MB
! the formula ln ZA(6) = .4 ln ZAO(2) for 1.7 MB
! the formula ln ZA(7) = .3 ln ZAO(2) for 2.0 MB
! set ZB, ZC, ZD, ZF = 0.0 from 0.1 MB to 2.0 MB
! make this change for new MAXLEV  jack maa 4/27/92

       do jj = 1, yang
         if (ZAO(1,jj) .ne. 0.0) then
       ZA(1,jj) = DEXP(0.1 * DLOG(ZAO(1,jj)))
       do ii=2,4
          ZA(ii,jj) = DEXP(0.3 * DLOG(ZAO(1,jj)))
             end do
         else
       do ii=1,4
          ZA(ii,jj) = 0.0
             end do
         end if
         if (ZAO(2,jj) .ne. 0.0) then
       ZA(5,jj) = DEXP(0.3 * DLOG(ZAO(2,jj)))
       ZA(6,jj) = DEXP(0.4 * DLOG(ZAO(2,jj)))
       ZA(7,jj) = DEXP(0.3 * DLOG(ZAO(2,jj)))
         else
       do ii=5,7
          ZA(ii,jj) = 0.0
             end do
         end if
         do ii=1,7
      ZB(ii,jj) = 0.0
```

```
      ZC(ii,jj) = 0.0
      ZD(ii,jj) = 0.0
      ZF(ii,jj) = 0.0
         end do
         do ii=1, OLDLEV  - 2
      ZA(ii+7,jj) = ZAO(ii+2,jj)
      ZB(ii+7,jj) = ZBO(ii+2,jj)
      ZC(ii+7,jj) = ZCO(ii+2,jj)
      ZD(ii+7,jj) = ZDO(ii+2,jj)
      ZF(ii+7,jj) = ZFO(ii+2,jj)
         end do
       end do

!
! look for matching 'chanid' to find
! which slot to copy the coefficients into

         DO K      = 1, maxcha

            IF ( ZCHANID .EQ. CHANID(K)) THEN

               nfound       = nfound + 1
!-----------------------------------------------------------------------
!      CHANGE UNITS AND TRANSFER COEFFICIENTS TO COMMON /TRCOEF/
!
!     zZM, zZN  conversion factors from g/cm^2 to molecules/cm^2
!        raised to the power M or N as appropriate
!-----------------------------------------------------------------------

               TAUCFM(K)     = ZM
               IF ( ZM .LT. 0.99 ) THEN
                 zZM           = ( AVOGAD / H2OMOL )**ZM
               ELSE
                 zZM           = ( AVOGAD / H2OMOL )
               ENDIF
               TAUCFN(K)     = ZN
               IF ( ZN .LT. 0.99 ) THEN
                 zZN           = ( AVOGAD / OZOMOL )**ZN
               ELSE
                 zZN           = ( AVOGAD / OZOMOL )
               ENDIF
!
!  in code portion below...
!       zZA -> TAUCFW  calculation of d tau^o/dp (not fct. of temperature)
!                      part of equation (A6) in Susskind et al 1983
!
!       the 1/(1-tau^o) is included in TAUCFC  (i.e. Cij of eqn A7)
!
               DO N      = 1, NUMANGS
                  TAUCFE(N,K)  = ZE(N)
                  TAUCFG(N,K)  = ZG(N)
                  ZQ           = 0.0
                  DO L      = 1,  maxlev
                     TAUCFA(L,N,K)= ZA(L,N)
```

```
                  TAUCFB(L,N,K)= ZB(L,N)
                  zZA(L,N)      = (1.0 - ZQ) * (1.0 - ZA(L,N))
                  TAUCFW(L,N,K)= zZA(L,N)
                  ZQ            = ZQ + zZA(L,N)
                  IF ( ZQ .GT. 0.0001 ) THEN
                     TAUCFC(L,N,K)= ZC(L,N) / ZQ
                  ELSE
                     TAUCFC(L,N,K)= 0.0
                  ENDIF
                  TAUCFD(L,N,K)= ZD(L,N) * zZM
                  TAUCFF(L,N,K)= ZF(L,N) * zZN
               ENDDO
            ENDDO

! channel slot was found, go back and read another record
!
            GOTO      100

         ENDIF

      enddo

!-----------------------------------------------------------------------
!           GO BACK TO STMT 100 AND READ ANOTHER RECORD
!-----------------------------------------------------------------------
   GOTO        100
!-----------------------------------------------------------------------
!           READ ERROR OR UNEXPECTED END-FILE ON TRFDAT
!-----------------------------------------------------------------------
  500 print *, 'error reading trfdat'
   ierr = 1
!-----------------------------------------------------------------------
!              EOF ON TRFDAT TRYING TO READ NEXT HEADER
!-----------------------------------------------------------------------
  600 CONTINUE
   IF ( nfound .LT. 24 ) then
      print *, 'error reading dataset'
      ierr = 2
   endif
   close ( itrfdat )
   RETURN
   END
```

grdcon.f – Set vertical grid constants

```
!*************************************************************************
!!GRDCON
!*************************************************************************
!
!!ROUTINE:  grdcon
!
!!DESCRIPTION:  SETS VERTICAL GRID CONSTANTS for 71 level
!               GLA rapid transmittance algorithm
!               initializes common block /GRDCONS/
!                Adapted from GLA TOVS retrieval code
!                (subroutines grdcon and retini)
!
! SYSTEM ROUTINES USED: alog
!
! SUBROUTINES CALLED:  none
!
!!INPUT PARAMETERS: none
!
!!OUTPUT PARAMETERS: none  - sets common block GRDCONS
!
!!REVISION HISTORY:
!  22Sep95  Meta S.  original routine adapted from GLA TOVS grdcon and retini
!  18May96  Meta S.  Add calculation for delprs - moved from tovsh2o_tau
!
!*************************************************************************
      subroutine  grdcon
      implicit none

      include "tovsparam.h"
      include "phycons.h"
      include "grdcons.h"

      integer l

!*************************************************************************

!  71 pressure levels
      data                pres
     $        /  0.1,  0.4,  0.7,  1.0,  1.3,  1.7,
     $                              2.,   3.,   4.,
     $             5.,   6.,   7.,   8.,   9.,  10.,
     $            15.,  20.,  30.,  40.,  50.,  60.,
     $            70.,  80.,  90., 100., 110., 120.,
     $           130., 140., 150., 160., 170., 180.,
     $           190., 200., 220., 240., 260., 280.,
     $           300., 320., 340., 360., 380., 400.,
     $           425., 450., 475., 500., 525., 550.,
     $           575., 600., 625., 650., 675., 700.,
     $           725., 750., 775., 800., 825., 850.,
     $           875., 900., 925., 950., 975.,1000.,
     $          1025.,1050.                          /
```

```
!
! reference temperature for 71 levels
      data              stntmp
     $        /221.6, 254.8, 264.4, 266.5, 265.0, 261.8,
     $                               259.7, 251.4, 246.1,
     $         242.2, 239.2, 236.6, 234.4, 232.5, 230.8,
     $         228.0, 224.5, 221.0, 217.0, 213.5, 210.6,
     $         208.1, 206.8, 206.5, 206.1, 206.6, 207.9,
     $         209.0, 210.1, 211.0, 212.2, 213.4, 214.7,
     $         215.8, 216.9, 219.3, 222.9, 226.2, 229.3,
     $         232.1, 235.2, 238.5, 241.6, 244.5, 247.3,
     $         250.2, 253.3, 256.3, 259.0, 261.6, 263.9,
     $         266.1, 268.2, 270.3, 272.2, 274.1, 275.9,
     $         277.5, 278.9, 280.3, 281.6, 282.9, 284.2,
     $         285.3, 286.3, 287.2, 288.2, 289.1, 290.0,
     $         290.8, 291.7                                    /
!
! aerosol scattering factor for 71 levels
      data              scat
     $        /0.00000, 0.00000, 0.00000, 0.00000, 0.00000, 0.00000,
     $                                    0.00000, 0.00000, 0.00000,
     $         0.00000, 0.00000, 0.00000, 0.00000, 0.00000, 0.00000,
     $         0.00000, 0.00000, 0.00000, 0.00000, 0.00000, 0.00022,
     $         0.00058, 0.00091, 0.00121, 0.00143, 0.00167, 0.00186,
     $         0.00207, 0.00222, 0.00238, 0.00254, 0.00271, 0.00292,
     $         0.00300, 0.00300, 0.00300, 0.00300, 0.00300, 0.00300,
     $         0.00300, 0.00300, 0.00300, 0.00300, 0.00300, 0.00300,
     $         0.00300, 0.00300, 0.00300, 0.00306, 0.00312, 0.00318,
     $         0.00325, 0.00331, 0.00340, 0.00354, 0.00368, 0.00381,
     $         0.00409, 0.00438, 0.00446, 0.00501, 0.00560, 0.00619,
     $         0.00678, 0.00741, 0.00864, 0.00988, 0.01112, 0.01236,
     $         0.01359, 0.01483                                  /


!-----------------------------------------------------------------------
!
!  calculate layer average of standard temperature and of rapid alg. pressures
!  and layer difference in pressure (times a constant)
!
!  NOTE: gravcon is set in phycon.f which must be called prior to this
!          routine
!

      tstd_av(1)    = stntmp(1)

      presa(1)     = 0.50*pres(1)
      delprs(1) = pres(1) * gravcon

      do l   = 2, maxlev
         presa(l)    = 0.5 * ( pres(l-1) + pres(l) )
         tstd_av(l)   = 0.5 * ( stntmp(l-1) + stntmp(l) )
         delprs(l) = (pres(l) - pres(l-1)) * gravcon
      enddo
```

```
!
!  calculate logarithm of rapid algorithm pressure level values
!
      do l = 1, maxlev
         presln(l) = alog ( pres (l) )
      enddo

      return
      END
```

hirsrad.f – Calculate radiance

```
!*************************************************************************
!!HIRSRAD
!*************************************************************************
!
!!ROUTINE:  hirsrad
!
!!DESCRIPTION:  Calculate radiance from input temperature, specific
!               humidity, ozone using input transmittance functions
!               Based on GLA TOVS retrieval code (vecrad and
!                  related subroutines)
!
! SYSTEM ROUTINES USED:  cos
!
! SUBROUTINES CALLED: planck
!
!!INPUT PARAMETERS:
!  tau      - real array (of size maxlev by maxcha) of transmittance
!  tausun   - real array (of length maxcha) of transmittance along
!                solar path to Earth's surface
!  nchan    - integer number of channels to calculate
!  chanarr  - integer array (of length maxcha) of channels to calculate
!                1-20 -> HIRS2 ch 1-20;  21-24 -> MSU 1-4
!  tair     - real array (of length maxlev) with temperature profile (K)
!  tmptop   - real temperature (K) at "top of atmosphere"
!  ir_zen   - real     satellite zenith angle for infrared channels (degrees)
!  emissmw  - real microwave emissivity
!  land     - logical flag  (.true. if land, .false. if water)
!  nlev     - real number of levels to use in profiles
!  rho      - real bidirectional reflectance
!  sunzang  - real solar zenith angle (degrees)
!  dayflag  - logical flag for day/night (.true. if day, .false. if night)
!  tground  - real skin (ground) temperature (K)
!  tsurfair - real surface air temperature (K)
!
!!OUTPUT PARAMETERS:
!
!  calc_rad  -  real array (of length maxcha) of radiance (or brightness
!                  temperatures, for MSU channels ) calculated from profile
!                  for channels input in 'chanarr'
!
!!REVISION HISTORY:
!  08sep95  Meta S.  original routine derived from subroutine vecrad
!                    in GLA retrieval code
!  03oct95  Meta S.  make solar calculation (which was borrowed from
!                    GLA routine grtemp) more consistent with
!                    solar calculations in  tovs_tauang.  Revised prologue.
!  15dec95  Meta S.  ensure rad_down is initialized
!  17apr96  Meta S.  *Input arguments changed* so multiple channels can
!                    be calculated with a single subroutine call
!                    (requested "optimization" change)
!*************************************************************************
```

```
      subroutine hirsrad(tau,tausun,nchan,chanarr,tair,tmptop,ir_zen,
     &     emissmw, land, nlev, rho, sunzang, dayflag, tground,
     &     tsurfair, calc_rad)

      implicit none

      include "tovsparam.h"

!  input variables
      real       tau(maxlev,maxcha)
      real       tausun(maxcha)
      integer    nchan
      integer    chanarr(maxcha)
      real       tair(maxlev)
      real       tmptop
      real       ir_zen
      real       emissmw
      logical    land
      integer    nlev
      real       rho
      real       sunzang
      logical    dayflag
      real       tground
      real       tsurfair

! output variable
      real       calc_rad(maxcha)

! function types
      real       planck

!  parameters from common

      include "phycons.h"
      include "chacons.h"
      include "matcons.h"

! local variables
      real       angsun
      real       btav          ! blackbody temp. of avg layer temperature
      real       btgnd         ! blackbody temp. of ground temperature
      real       btsurf        ! blackbody temp. of surface air temperature
      integer    chan          !  channel counter
      real       crad          ! partial sum for radiance
      real       deltau        ! difference of transmittance in layer
      logical    dosun
      real       dtauinv       ! inverse transmittance difference
      real       emiss         !  sfc emissivity for this sounding
      integer    i,l           !  counter
      real       pcon1         !
      real       pcon2         !
      real       rad_down      !  atmos. emission downward flux
      real       rad_sun       !  solar radiation at TOA (?)
```

```
      real       rhosec
      real       sunterm       !  solar contribution to calc. radiance
      real       tav(maxlev)   !  mean layer temperature array


      tav(1) = 0.5 * (tair(1) + tmptop)
      do l = 2,nlev
         tav(l) = 0.5 * (tair(l-1) + tair(l))
      enddo

      do i = 1,nchan

         chan = chanarr(i)

         dosun = ( freq(chan) .gt. 2000. ) .and. dayflag

         pcon1 = planck1(chan)
         pcon2 = planck2(chan)
!

         if (irchan(chan)) then

!  initialize radiance sum
            deltau = 1.0 - tau(1,chan)
            btav = planck(tav(1), pcon1, pcon2)
            crad = btav * deltau

!  integrate through atmosphere (from top down)
            do l = 2,nlev
               btav = planck(tav(l), pcon1, pcon2)
               deltau = tau(l-1,chan) - tau(l,chan)
               crad = crad + btav * deltau
            enddo

!  prepare to calculate surface contribution
            btsurf = planck(tsurfair, pcon1, pcon2)
            btgnd  = planck(tground, pcon1, pcon2)
            rad_down = dfxcon(chan) * btsurf * (1. - tau(nlev, chan))
            if (land) then
               emiss = em_land(chan)
            else
               emiss = em_water(chan)
            endif

         else if (mwchan(chan)) then

!  initialize radiance sum
            btav = tav(1)
            deltau = 1.0 - tau(1,chan)
            rad_down = 0.0
            if (tau(1,chan) .ne. 0.0) then
               dtauinv = (1. / tau(1,chan)) - 1.
               rad_down = dtauinv * btav
            endif
```

```
            crad = btav * deltau

!   integrate through atmosphere (from top down)
            do l = 2,nlev
               btav = tav(l)
               deltau = tau(l-1,chan) - tau(l,chan)
               if ((tau(l,chan) .ne. 0.0 ) .and.
     &              (tau(l-1,chan) .ne. 0.0) )  then
                  dtauinv = ( 1. / tau(l,chan)) - ( 1./ tau(l-1,chan))
                  rad_down = rad_down + dtauinv * btav
               endif
               crad = crad + btav * deltau
            enddo

!   prepare to calculate surface contribution
            btsurf = tsurfair
            btgnd  = tground
            rad_down = (rad_down + bigbang) * tau(nlev,chan)
            emiss = emissmw

         endif


         if (use_refl(chan)) then
            crad = crad + tau(nlev, chan) *
     &        (cos(ir_zen*deg2rad)*crad*(1.-emiss)*(dfxcon(chan)*2.0)
     &           + emiss * btgnd)
         else
            crad = crad + tau(nlev, chan) *
     &           (rad_down + emiss * ( btgnd - rad_down ))
         endif

!   Add solar contribution for wavenumber greater than 2000.0

         if (dosun) then
            if (sunzang .le. sunset) then
               angsun = sunzang * deg2rad
            else
               angsun = 89.9e0 * deg2rad
            endif
            rhosec = rho * cos(angsun)
            rad_sun = scon * planck(tsun, pcon1, pcon2)
            sunterm = rhosec * rad_sun * tausun(chan)

            crad = crad + sunterm
         endif

         calc_rad(chan) = crad
      enddo

      return
      end
```

initialGLA.f – Sample initialization code

```
!*****************************************************************
!!INITIALGLA
!*****************************************************************
!
!!ROUTINE:  initialGLA
!
!!DESCRIPTION:  *Sample* routine to carry out initialization
!               of variables ... derived from code fragments in
!               test routines
!
!
! SYSTEM ROUTINES USED:
!
! SUBROUTINES CALLED:
!    matcon - SETS MATH CONSTANTS FOR RETRIEVALS
!    phycon - SETS PHYSICAL CONSTANTS FOR RETRIEVALS
!    grdcon - SETS VERTICAL GRID CONSTANTS
!    chacon - Set channel constants for TOVS (HIRS/MSU) channels
!    dattrf - Read rapid algorithm transmittance function coefficients
!
! FILES NEEDED:  "bland"  land/water flags
!
!!INPUT PARAMETERS:
!  itrfdat   - integer unit number for input file
!  satelite  - character*2 identifier for satellite
!
!
!!OUTPUT PARAMETERS:
!  bland     -  logical array (of size 360 by 180) of 1X1 degree flags
!               for land and water
!  ierr      -  integer error flag for transmittance coeffs
!                                  0 - successful read
!                                  1 - error reading dataset
!                                  2 - less than 24 channels found
!
!
!!REVISION HISTORY:
!  29nov95  Meta S.  original routine
!  28mar96  Meta S.  added prologue
!  19may96  Meta S.  shifted order of subroutine calls: matcon before
!                      phycon (before grdcon)
!
!*****************************************************************

      subroutine initialGLA(itrfdat,satelite,bland,ierr)
!
      character*2 satelite
      integer itrfdat
      logical bland(360,180)
      integer ierr
```

```
      integer i,j

!  initialize constants, physical, mathematical, and vertical grid related

      call matcon
      call phycon
      call grdcon

!     read in land/water flag dataset

!  the following statement will need to be changed for the operational
!  system to access a more general input dataset

      open(31,file='/ford1/local/new-data-types/gla-tovs/data/bland',
     &      form='formatted',status='old')

      do i =1,360
        read(31,'(18011)') (bland(i,j),j=1,180)
      enddo

!  initialize channel-related constants, read in transmittance coefficients

      call chacon(satelite)
      call dattrf(itrfdat,ierr)

      return
      end
```

matcon.f – Set up mathematical constants

```
!*****************************************************************
!!MATCON
!*****************************************************************
!
!!ROUTINE:  MATCON
!
!!DESCRIPTION:  SETS MATH CONSTANTS FOR RETRIEVALS
!
! SYSTEM ROUTINES USED: none
!
! SUBROUTINES CALLED: none
!
!!INPUT PARAMETERS: none
!
!!OUTPUT PARAMETERS: none
!
!!REVISION HISTORY:
!             1nov84 Jim Pf.  original routine in GLA retrival code
!            15sep95 Meta S. change common /matcons/ to include
!                            only trigonometric constants
!*****************************************************************
      SUBROUTINE MATCON

      implicit none

      include "matcons.h"
!-----------------------------------------------------------------
!    SET MATHEMATICAL CONSTANTS AS USED IN GLA RETS
!-----------------------------------------------------------------
      PI           = 3.141592654
      DEG2RAD      = PI / 180.0
      PID2         = PI / 2.0
      RAD2DEG      = 180.0 / PI
      TWOPI        = PI + PI
      RETURN
      END
```

phycon.f – Set up physical constants

```
!*****************************************************************
!!PHYCON
!*****************************************************************
!
!!ROUTINE:  PHYCON
!
!!DESCRIPTION:   SETS PHYSICAL CONSTANTS FOR RETRIEVALS
!                fills GLA common 'phycons'
!
! SYSTEM ROUTINES USED:    NONE
!
! SUBROUTINES CALLED:      NONE
!
!!INPUT PARAMETERS:  none
!
!!OUTPUT PARAMETERS: none, fills GLA common 'phycons'
!
!!REVISION HISTORY:
!             2Apr85 Jim Pf.   original routien in GLA retrieval code
!            15sep95 Meta S.   reformatted original GLA routine
!            28mar95 Meta S.   removed some unsed constants,
!                                 added some comments
!            18may95 Meta S.   move some "constants" from tovsh2o_tau
!
!*****************************************************************
      subroutine phycon
      implicit none

      include "matcons.h"
      include "phycons.h"

!     local variables

      real radsun      ! Sun's radius
      real radeorb     ! radius of Earth's orbit around Sun
!
!*****************************************************************
!-----------------------------------------------------------------
!  set physical constants as used in GLA retrievals
!-----------------------------------------------------------------
      avogad       = 6.022045e+23          ! molecules/mole
      boltzmns     = 8.317e0 / avogad      ! Joule / Kelvin
      clight       = 2.9979e+10            ! cm s^{-1}
      declmax      = 23.45                 ! degrees
      plancks      = 6.626176e-34          ! Joule-sec
      radeorb      = 149.57e+09            ! meters
      radsun       = 0.69595e+09           ! meters
      atmosph      = 1013.25               ! millibars
      bigbang      = 2.7                   ! Kelvin
      grav         = 9.80665               ! m s^{-2}
      daymaxd      = 173.0
```

```
      days_per_yr  = 365.0
      shmin        = 2.0e-06              ! kg air / kg H2O
      sunset       = 85.0e0               ! degrees
      tsun         = 5600.0e0             ! Kelvin
      airmol       = 28.8e0               ! g mol^{-1}
      h2omol       = 18.0e0               ! g mol^{-1}
      ozomol       = 48.0e0               ! g mol^{-1}

!-----------------------------------------------------------------------
!                          derived constants
!-----------------------------------------------------------------------
      scon         = pi * ( radsun / radeorb )**2     ! steradian
      plcon1       = 2.0 * plancks * clight * clight
      plcon2       = plancks * clight / boltzmns
      gravcon      = 10. / grav
      econ         = airmol / (h2omol * atmosph)

      RETURN
!-----------------------------------------------------------------------
      END
```

planck.f – Calculate Planck function

```
!*****************************************************************
!!PLANCK
!*****************************************************************
!
!!ROUTINE:  PLANCK
!
!!DESCRIPTION:  COMPUTES RADIANCE FROM TEMPERATURE VIA PLANCK FUNCTION
!
! SYSTEM ROUTINES USED: none
!
! SUBROUTINES CALLED: none
!
!!INPUT PARAMETERS:
!     temp      real  Temperature (K)
!     PLANK1    real WAVE NUMBER DEPENDENT CONSTANT
!                     PLANK1 = 2 * VELLIGHT**2 * PLANCKCON / LAMDA**3
!                     PLANK1 = 1.193E-12/LAMDA**3  WATTS / (CM**2 - SR)
!                        WHERE
!                     VELLIGHT = 3.00E+10 CM/SEC
!                     PLANKCON = 6.63E-34  JOULES-SEC
!                     LAMDA    = WAVE LENGTH IN CM
!     PLANK2    real WAVE NUMBER DEPENDENT CONSTANT
!                     PLANK2 = PLANCKCON*VELLIGHT / (BOLTZMAN*LAMDA)
!                     PLANK2 = 1.441/LAMDA   DEG.K
!                        WHERE
!                     BOLTZMAN = 1.38E-23 JOULE / DEG.K
!!OUTPUT PARAMETERS:
!     PLANCK  real blackbody radiance for given temperature
!                  UNITS WATTS/(CM**2-SR)
!
!!REVISION HISTORY:
!     25oct1984  Jim Pf.   original routine VPLANCK in GLA retrieval code
!     11oct1995  Meta S.   added DAO prologue
!
!*****************************************************************

      real FUNCTION PLANCK(temp,PLANCK1,PLANCK2)
      implicit none
!
! STATEMENT FUNCTION FOR PLANCK FUNCTION
!
      real    temp
      real    planck1
      real    planck2

      PLANCK = PLANCK1/(EXP(PLANCK2/temp)-1.0)

      RETURN
      END
```

tovsfix_tau.f – Fixed gas transmittance contribution

```
!*************************************************************************
!!TOVSFIX_TAU
!*************************************************************************
!
!!ROUTINE:  tovsfix_tau
!
!!DESCRIPTION:  Calculate contribution to transmittance by fixed gases
!               using rapid transmittance algorithm coefficients
!               - reference Susskind, et al. 1983 JGR, p 8565
!
! CALLED FROM:   tovs_tau
!
! SYSTEM ROUTINES USED:  none
!
! SUBROUTINES CALLED: none
!
!!INPUT PARAMETERS:
!  tdif     - real array (of length nlev) with difference between
!               current best estimate of layer mean temperatures and
!               standard atmosphere layer mean temperatures (K)
!  chan     - integer channel number
!  nlev     - integer surface level (also number of levels to use)
!
!
!!OUTPUT PARAMETERS:
!  taufix   - real array of effective mean layer transmittance
!              for fixed gases
!
!!REVISION HISTORY:
!  07sep95  Meta S.  original routine derived from subroutine tmptau
!                    in GLA retrieval code
!  18may96  Meta S.  effTdif need not be saved as fct. of angle
!
!*************************************************************************
      subroutine tovsfix_tau(tdif, chan, nlev, taufix)

      implicit none

      include "tovsparam.h"
!
!  variables passed into routine
!
      integer nlev, chan
      real tdif(maxlev)
!
!  variables output from routine
      real taufix(maxlev,maxangs)

!  constants, quasi-constants, and coefficients
      include "taucoef.h"
```

```
!  local variables
      integer l,k
      real effTdif

!
!  Susskind et al 1983  algorithm for transmittance from fixed gases
!  Effective mean temperature  above P -> \tilde{T} in equation (A6)
!
!  taucfw: precomputed non-varying part of \tilde{T} integral
!  taucfc: includes term with division by (1 - \tau^o)
!
!    here effTdif ->  \tilde{T} - \tilde{T}^o  or
!       difference between effective mean temperatures for the
!       current temperature profile and the standard temperature profile
!  Then:
!
!  tau = A + B (T - T^o) + C ( \tilde{T} - \tilde{T}^o)      (A7)
!
      do k = 1, maxangs
         effTdif = 0.
         do l = 1,nlev
            effTdif = effTdif + taucfw(l,k,chan) * tdif(l)

            taufix(l,k) = taucfa(l,k,chan) + taucfb(l,k,chan) * tdif(l)
     &           + taucfc(l,k,chan) * effTdif
            taufix(l,k) = amax1( taufix(l,k), 0.0)
            taufix(l,k) = amin1( taufix(l,k), 1.0)
         enddo
      enddo

      return
      end
```

tovsh2o_tau.f – Water vapor transmittance contribution

```
!*****************************************************************
!!TOVSH2O_TAU
!*****************************************************************
!
!!ROUTINE:  tovsh2o_tau
!
!!DESCRIPTION:  Calculate contribution to transmittance by water vapor
!               using rapid transmittance algorithm coefficients
!               - reference Susskind, et al. 1983 JGR, p 8564
!
! CALLED FROM:   tovs_tau
!
! SYSTEM ROUTINES USED:  exp
!
! SUBROUTINES CALLED: none
!
!!INPUT PARAMETERS:
!  h2ocd    - real array (of length maxlev) with water vapor layer
!                column density ( g cm^-2)
!  e        - real array (of length maxlev) of H2O vapor pressure (atm)
!  chan     - integer channel number
!  nlev     - integer surface level (also number of levels to use)
!  tav   - real array (of length maxlev) with temperature profile (K)
!  celsius  - real array (of length maxlev) with (T - 273) (approx.
!                 temperature in celsius)
!
!!OUTPUT PARAMETERS:
!  tauh2o   - real array of effective h2o transmittance
!
!!REVISION HISTORY:
!  06sep95  Meta S.  original routine derived from subroutine h2otau
!                    in GLA retrieval code
!  17may96  Meta S.  moved pressure calculations to main routine,
!                    rewrote continuum expression
!*****************************************************************

      subroutine tovsh2o_tau( h2ocd, e, chan, nlev, tav,
     &           celsius, tauh2o)

      implicit none

      include "tovsparam.h"

!
!  variables passed into routine
      real     h2ocd(maxlev)
      real     e(maxlev)
      integer  chan
      integer  nlev
      real     tav(maxlev)
      real     celsius(maxlev)
```

```
!  variable output from routine
      real     tauh2o(maxlev,maxangs)

!  constants, quasi-constants, and coefficients

      include "taucoef.h"
      include "chacons.h"
      include "grdcons.h"

!  local variables
      real     cont(maxlev)
      real     effh2o(maxlev)
      integer  l, k
      real     taulog

!  statement function definitions

      real     quikexp
      real     X

      quikexp(X) = 1.0 + X * (1.0 + X * ( .5 + X * .16666667 ) )


! for IR channel, calculate continuum (except for angle contribution)
!    see Susskind and Searl (1978) J. Quant. Spect. Radiat. Transfer
!
!  kH2O(\nu,i) = \sum_l k2(\nu,T) e(l) U(l)
!
!  U(l) (as above) -->  h2ocd = delprs * h2omid
!
! degchn = (- dK/dT) for channel; k2(T) = k2(contmp)*(1-degchn)(T-contmp)
!
!
      if (irchan(chan)) then

         do l = 1,nlev
            cont(l) = e(l) * abscof(chan) * h2ocd(l) *
     &           (1.0 - degchn(chan) * (tav(l) - contmp(chan)))
         enddo

      else
         do l = 1,nlev
            cont(l) = 0.0
         enddo
      endif

      if ( taucfm(chan) .lt. 0.99) then
         do l = 1,nlev
            effh2o(l) = h2ocd(l) ** taucfm(chan)
         enddo
      else
         do l = 1,nlev
            effh2o(l) = h2ocd(l)
```

```
         enddo
       endif

!
!  From (A5) in Susskind, et al. 1983  (page 8564)
!
! tau(H2O) = exp ( - ( D (1 - E (T - 273)) h2ocd**M)
!         {   * exp(-sec \theta * KcontinuumH2O) of course }
!
!  see also ON-9608, eq32
!
!  From GLA retrieval code:
!    TAULOG(II,K)  = (-1.0 - TAUCFE(K,YCHA) * TMPXCES(II))
!                    * TAUCFD(LEVL,K,YCHA) * EFFH2O(II)
!
!    so perhaps TAUCFE = -E
!
       do k = 1, numangs
          do l = 1,nlev
             taulog = ( -1.0 - taucfe(k,chan) * celsius(l))
     &            * taucfd(l,k,chan) * effh2o(l)
             taulog = taulog - secang(k) * cont(l)
             taulog = amin1(taulog, 0.0)
             tauh2o(l,k) = quikexp( taulog )
          enddo
       enddo

       return
       end
```

tovsozo_tau.f – Ozone transmittance contribution

```
!*************************************************************************
!!TOVSOZO_TAU
!*************************************************************************
!
!!ROUTINE:  tovsozo_tau
!
!!DESCRIPTION:  Calculate contribution to transmittance by ozone
!               using rapid transmittance algorithm coefficients
!               - reference Susskind, et al. 1983 JGR, p 8564
!
! CALLED FROM:   tovs_tau
!
! SYSTEM ROUTINES USED:  exp
!
! SUBROUTINES CALLED: none
!
!!INPUT PARAMETERS:
!  ozomid   - real array (of length mlev) with ozone column
!                densities (g/cm**2)
!  chan     - integer channel number
!  nlev     - integer number of levels (top to surface)
!  celsius  - real array (of length mlev) with profile of (T-273) (K)
!
!!OUTPUT PARAMETERS:
!  tauozo   - real array (of size maxlev by maxangs) of effective ozone
!             transmittance
!
!!REVISION HISTORY:
!  06sep95  Meta S.  original routine derived from subroutine ozotau
!                    in GLA retrieval code
!
!*************************************************************************

      subroutine tovsozo_tau(ozomid, chan, nlev, celsius, tauozo)

      implicit none
!
      include "tovsparam.h"
!
!  variables passed into routine
      real     ozomid(maxlev)
      integer  chan
      integer  nlev
      real     celsius(maxlev)
!
!  variable output from routine
      real     tauozo(maxlev,maxangs)
!
!  constants, quasi-constants, and coefficients

      include "taucoef.h"
```

```
!  local storage
      real     eff03(maxlev)
      integer  l, k
      real     taulog

!  statement function definitions
      real     quikexp
      real     X

      quikexp(X) = 1.0 + X * (1.0 + X * ( .5 + X * .16666667 ) )
!
!  calculate 'eff03' as (ozone conc.) ** taucfN
!
!  following is formula used in GLA code:
!      if (taucfn(chan) .lt. 0.99) then
!         do l = 1,nlev
!            eff03(l) = taucfn(chan) * alog( ozomid(l))
!            eff03(l) = exp( eff03 )
!         enddo
!      else
!         do l = 1,nlev
!            eff03(l) = ozomid(l)
!         enddo
!      endif

      do l = 1, nlev
         eff03(l) = ozomid(l) ** taucfn(chan)
      enddo

!
!  From (A5) in Susskind, et al. 1983  (page 8564)
!
! tau(ozone) = exp ( - ( F (1 - G (T - 273)) ozomid**N)
!                                              see also ON-9608  eq 33
!
! from GLA retrieval code:
! TAULOG(II,K) = (-1.0 - TAUCFG(K,YCHA) * TMPXCES(II))
!   * TAUCFF(LEVL,K,YCHA) * EFF03(II)* TAUCFF(LEVL,K,YCHA) * EFF03(II)
!
!  so sign of TAUCFG = -G , perhaps...
!

      do k = 1,numangs
         do l = 1, nlev
            taulog = ( -1.0 - taucfg(k,chan) * celsius(l))
     &           * taucff(l,k,chan) * eff03(l)
            taulog = amin1 (taulog, 0.0)
            tauozo(l,k) = quikexp( taulog )
         enddo
      enddo

      return
      end
```

tovs_tau.f – Driver for transmittance calculation

```
!*************************************************************************
!!TOVS_TAU
!*************************************************************************
!
!!ROUTINE:  tovs_tau
!
!!DESCRIPTION:  Calculate transmittance for given channel numbers
!               for a specified temperature, moisture, and ozone profile
!
!               Based on GLA TOVS retrieval code (comtau and
!                  related subroutines)
!
! SYSTEM ROUTINES USED:  abs, cos
!
! SUBROUTINES CALLED:
!   tovsfix_tau   - transmittance contribution due to fixed gases
!   tovsh2o_tau   - transmittance contribution due to H2O
!   tovsozo_tau   - transmittance contribution due to ozone
!   tovs_tauang   - interpolate layer transmittance to satellite zenith
!                     angle and take products over layers
!
!!INPUT PARAMETERS:
!  tair     - real array (of length maxlev) with temperature profile (K)
!  tmptop   - real temperature (K) at "top of atmosphere"
!  h2omid   - real array (of length maxlev) with water vapor layer specific
!               humidities (KG/KG)
!  ozomid   - real array (of length maxlev) with ozone column densities
!               (g/cm**2)
!  ps       - real surface pressure (mb)
!  nchan    - integer number of channels to calculate
!  chanarr  - integer array (of length maxcha) of channels to calculate
!               1-20 -> HIRS2 ch 1-20;  21-24 -> MSU 1-4
!  nlev     - integer surface level (also number of levels to use)
!  mw_zen   - real     satellite zenith angle for microwave channels
!  ir_zen   - real     satellite zenith angle for infrared channels
!  dayflag  - logical  true if day ( sun angle < SUNSET = 85.0E0)
!  secsun   - real  1. / cos( sun angle)
!
!
!!OUTPUT PARAMETERS:
!
!  tau    - real array (of size maxlev by maxcha) of transmittance along
!             path to satellite.
!  tausun - real array (of size maxcha) of transmittance along path from
!             sun
!  errflag - logical returns .true. if error
!
!!REVISION HISTORY:
!  11sep95  Meta S.  original routine derived from subroutine comtau
!                    in GLA retrieval code
!  04oct95  Meta S.  clarify arrangement of channels in tau, tausun
```

```
!  04dec95  Meta S.  update prologue, add comments describing local
!                      variables
!  17may96  Meta S.  moved calculation of column H2O density and vapor
!                      pressure from tovsh2o_tau into this routine
!*********************************************************************
      subroutine tovs_tau(tair, tmptop, h2omid, ozomid, ps, nchan,
     &     chanarr, nlev, mw_zen, ir_zen, dayflag, secsun, tau, tausun,
     &     errflag)

      implicit none

      include "tovsparam.h"
!
!  variables passed into routine
!
      real     tair(maxlev)
      real     tmptop
      real     h2omid(maxlev)
      real     ozomid(maxlev)
      real     ps
      integer  nchan
      integer  chanarr(maxcha)
      integer  nlev
      real     mw_zen
      real     ir_zen
      logical  dayflag
      real     secsun

!
!  variables output from routine
      real     tau(maxlev, maxcha)
      real     tausun(maxcha)
      logical  errflag

!  constants, quasi-constants, and coefficients

      include "matcons.h"
      include "grdcons.h"
      include "chacons.h"
      include "phycons.h"

!  local variables
      integer l,k                    !  counters
      integer ichan
      integer chan

      real tav(maxlev)               ! mean layer temperature
      real tdif(maxlev)              ! difference from rapid alg std. temp.
      real celsius(maxlev)           ! difference from 273


      real effangmw                  !  microwave and
      real seceffmw                  !
      real effangir                  !  IR channel zenith angles and secants
```

```
      real seceffir                !
      real effang                  !
      real seceff                  !

      real h2ocd (maxlev)          !  H2O column density
      real e(maxlev)               !  H2O vapor pressure

! effective mean layer transmittance from fixed gas contribution
      real taufix(maxlev,maxangs)

! effective mean layer transmittance from water vaper contribution
      real tauh2o(maxlev,maxangs)

! effective mean layer transmittance from ozone contribution
      real tauozo(maxlev,maxangs)

! effective mean layer transmittances at rapid algorithm angles
      real tauang(maxlev,maxangs)

!  calculated transmittance along path for satellite zenith angle
!  and sun angle

      real ctau(maxlev), ctausun

      errflag = .false.

!
!  calculate quantities which are common for all channels
!  tav - mean layer temperature for RT calculation
!  tdif   - difference from rapid algorithm's std. atmos. mean temp.
!  celsius - difference from 273 K
!     (note this is not Celsius; 273 K is used in rapid algorithm)

      tav(1) = 0.5 * ( tmptop + tair(1))
      tdif(1) = tav(1) - tstd_av(1)
      celsius(1) = tav(1) - 273.
      do l = 2,nlev
          tav(l) = (tair(l-1)+tair(l)) * 0.5
          tdif(l) = tav(l) - tstd_av(l)
          celsius(l) = tav(l) - 273.
      enddo

!  compute mid-level h2o column densities (g/cm**2)
!
!   delprs = {\delta p } * 10 / grav  -->  density(air) * {\delta z}
!
!   delprs * h2omid(spec. humid) --> column density effh2o  U(l)

      do l = 1,nlev-1
        h2omid(l) = amax1( h2omid(l), shmin)
        h2ocd(l) = delprs(l) * h2omid(l)
      enddo
      h2ocd(nlev) = (ps - pres(nlev-1)) * gravcon * h2omid(nlev)
```

```
!  e(l) = H2O partial pressure (in atm)
!          =  (airmol/h2omol) * P(l) / (1013.25 mb per atm)
!                    * h2omid (spec. humidity)
!
!     econ = airmol / (h2omol * atmosph)  !  set in phycons.h

      do l = 1,nlev-1
         e(l) = presa(l) * econ * h2omid(l)
      enddo
      e(nlev) = (ps + pres(nlev-1)) * 0.5 * econ * h2omid(nlev)

!  zenith angle factors for microwave and IR channels

      effangmw = abs( deg2rad * mw_zen )
      seceffmw = 1. / cos( effangmw )      ! secant of MW zenith angle
      effangir = abs( deg2rad * ir_zen )
      seceffir = 1. / cos( effangir )      ! secant of IR zenith angle

!  loop over requested angles

      do ichan = 1,nchan ! begin channel loop
         chan = chanarr(ichan)
         if (chan .gt. maxcha) then
            print *,' tovs_tau:  bad channel requested'
            errflag = .true.
            return
         else

!  set appropriate zenith angle (effang) and secant for IR or MW channel

            if (irchan(chan)) then
               effang = effangir
               seceff = seceffir
            else if (mwchan(chan)) then
               effang = effangmw
               seceff = seceffmw
            endif

!  calculate contribution to transmittance from fixed gases

            call tovsfix_tau(tdif, chan, nlev, taufix)

!  calculate contribution to transmittance from water vapor
            call tovsh2o_tau( h2ocd, e, chan, nlev, tav,
     &           celsius, tauh2o)

!  calculate contribution to transmittance from ozone
            call tovsozo_tau( ozomid, chan, nlev, celsius, tauozo)
!
!  mean layer tramsmittance is product of contributions from fixed gases,
!    H2O and O3; calculate for angles used in rapid algorithm
            do k = 1,maxangs
               do l = 1,nlev
                  tauang(l,k) = taufix(l,k)*tauh2o(l,k)*tauozo(l,k)
```

```
            enddo
         enddo

!  interpolate transmittance from rapid algorthim angles to observed
!     zenith angle and to solar zenith angle
         call tovs_tauang(tauang, chan, nlev, effang, seceff,
     &           dayflag, secsun, ctau, ctausun)

!
!  fill arrays with calculated tau and tausun
!
         do l = 1,nlev
            tau(l,chan) = ctau(l)
         enddo

         tausun(chan) = ctausun

      endif
   enddo                                  ! end of channel loop

   return
   end
```

tovs_tauang.f – Interpolate transmittance to zenith angle

```
!*****************************************************************
!!TOVS_TAUANG
!*****************************************************************
!
!!ROUTINE:  tovs_tauang
!
!!DESCRIPTION:  Interpolate computed transmittances to satellite
!                 zenith angle
!
! CALLED FROM:   tovs_tau
!
! SYSTEM ROUTINES USED:  exp, cos, acos
!
! SUBROUTINES CALLED: none
!
!!INPUT PARAMETERS:
!  tauang   - real matrix (of size maxlev by maxangs) of mean layer
!               transmittance calculated for different zenith angles
!  chan     - integer channel number
!  nlev     - integer surface level (also number of levels to use)
!  effang   - real abs(zenith angle) in radians
!  seceff   - real 1. / cos(effang)
!  dayflag  - logical flag  day = .true.  night = .false.
!  secsun   - real secant of solar angle for this observation
!
!!OUTPUT PARAMETERS:
!  tau      - real array of mean layer transmittance
!                 for satellite zenith angle
!  tausun   - real mean layer transmittance
!                 for solar radiation
!
!!REVISION HISTORY:
!  08sep95  Meta S.  original routine derived from subroutine tauang
!                    and tscatt in GLA retrieval code
!   1dec95  Meta S.  set tausun = 0 to cover when sun term not used
!                    (avoid uninitialized variable)
!
!*****************************************************************

      subroutine tovs_tauang(tauang, chan, nlev, effang, seceff,dayflag,
     &           secsun, tau, tausun)

      implicit none

      include "tovsparam.h"

!  variables passed into routine
      real     tauang(maxlev,maxangs)
      integer  chan
      integer  nlev
      real     effang
```

```
      real     seceff
      logical  dayflag
      real     secsun

!  variables output from routine
      real     tau(maxlev)
      real     tausun

!  constants, quasi-constants, and coefficients
!  (in common block or passed into routine)

      include "taucoef.h"
      include "grdcons.h"
      include "chacons.h"

!  local variables
      real     anglsun
      logical  dosun
      real     efcos
      real     efsun
      integer  k, l, nfang, nfsun, nangm1
      real     sat_factor
      real     sfactor
      real     sun_factor
      real     tauscat

!  statement functions
      real quikexp
      real X
      quikexp(X) = 1.0 + X * (1.0 + X * ( .5 + X * .16666667 ) )

!  set up some variables

      nangm1 = numangs - 1
      dosun = ( freq(chan) .gt. 2000. ) .and. dayflag
      tausun = 0.       ! set this in case of .not.irchan or .not.dosun
!
! find which angles of rapid algorithm bracket the observed zenith angle
!  (nfang and nfang+1) and calculate factor (sat_factor) for interpolation
!  in sec(zenith angle)
!
      nfang = 1
      do k = 2,nangm1
         if (effang .gt. angle(k) ) nfang = k
      enddo

      sat_factor = ( seceff - secang(nfang) ) /
     &                 (secang(nfang+1) - secang(nfang) )

!
!  calculate transmittance as product of transmittance through layers above
!   and transmittance in current layer (interpolated to sat. zenith angle)
!
      tau(1) = amax1( 0.0, (tauang(1, nfang)
```

```
     &     + sat_factor * (tauang(1,nfang+1)-tauang(1,nfang)) ) )

      do l = 2,nlev
         tau(l) = tau(l-1) *
     &        amax1( 0.0, (tauang(l, nfang)
     &          + sat_factor * (tauang(l,nfang+1)-tauang(l,nfang))))
      enddo

      if (irchan(chan)) then

!  for IR channels, include factor for scattering (from GLA routine tscatt)
         do l = 1,nlev
            sfactor = chscatt(chan) * scat(l)
            tauscat = quikexp( -sfactor * seceff )
            tau(l) = tau(l) * tauscat
         enddo

         if (dosun)  then

! if solar contribution needed, calculate effective transmittance for sun

! find which angles of rapid algorithm bracket the effective sun angle
! (nfsun and nfsun+1)  and calculate factor (sun_factor) for interpolation
! in sec(sun angle)

            anglsun = seceff + secsun
            efcos = 1. / anglsun
            efsun = acos( efcos )

            nfsun = 1
            do k = 2, nangm1
               if ( efsun .gt. angle(k) ) nfsun = k
            enddo

            sun_factor = ( anglsun - secang(nfsun) ) /
     &                   (secang(nfsun+1) - secang(nfsun) )

!  calculate product of transmittance in layers along solar path
            tausun = amax1( 0.0, (tauang(1,nfsun)
     &        + sun_factor * (tauang(1,nfsun+1) - tauang(1,nfsun)) ))

            do l = 2,nlev
               tausun = tausun *
     &           amax1( 0.0, (tauang(l, nfsun)
     &           + sun_factor * (tauang(l,nfsun+1)-tauang(l,nfsun))))
            enddo
            sfactor = chscatt(chan) * scat(nlev)
            tauscat = quikexp( -sfactor * anglsun )
            tausun = tausun * tauscat             !!! check this...
         endif

      endif
      return
      end
```

B Jacobian modules prologues and source code

dplanck.f – Calculate derivative of Planck function

```
!*******************************************************************
!!DPLANCK
!*******************************************************************
!
!!ROUTINE:  dPLANCK
!
!!DESCRIPTION:  derivative of PLANCK FUNCTION wrt. temperature
!
! CALLED FROM:  dTb_dTQU
!
! SYSTEM ROUTINES USED: none
!
! SUBROUTINES CALLED: none
!
!!INPUT PARAMETERS:
!     temp      real  Temperature (K)
!     PLANK1  real WAVE NUMBER DEPENDENT CONSTANT
!                     PLANK1 = 2 * VELLIGHT**2 * PLANCKCON / LAMDA**3
!                     PLANK1 = 1.193E-12/LAMDA**3  WATTS / (CM**2 - SR)
!                        WHERE
!                     VELLIGHT = 3.00E+10 CM/SEC
!                     PLANKCON = 6.63E-34  JOULES-SEC
!                     LAMDA    = WAVE LENGTH IN CM
!     PLANK2  real WAVE NUMBER DEPENDENT CONSTANT
!                     PLANK2 = PLANCKCON*VELLIGHT / (BOLTZMAN*LAMDA)
!                     PLANK2 = 1.441/LAMDA   DEG.K
!                        WHERE
!                     BOLTZMAN = 1.38E-23 JOULE / DEG.K
!!OUTPUT PARAMETERS:
!     PLANCK  real blackbody radiance for given temperature
!                  UNITS WATTS/(CM**2-SR)
!
!!REVISION HISTORY:
!     27nov1995  Meta S.   original routine
!
!*******************************************************************

      real FUNCTION dPLANCK(temp,PLANCK1,PLANCK2)
      implicit none
!
! STATEMENT FUNCTION FOR PLANCK FUNCTION
!
      real temp
      real planck1
      real planck2

      real d1              ! temporary variable
```

```
d1 = EXP(PLANCK2/temp)
dPLANCK = PLANCK1 / (d1-1.)**2 * d1 * planck2 / (temp*temp)

RETURN
END
```

dTb_dTQU.f – Calculate brightness temperature Jacobian

```
!*************************************************************************
!!DTB_dTQU
!*************************************************************************
!
!!ROUTINE:  dTb_dTQU
!
!!DESCRIPTION:  Calculate Jacobian or derivative of brightness
!               temperature with respect to input temperature,
!               water vapor and ozone profiles
!               dTb/dT,  dTb/dq  and dTb/du
!
!               Based on GLA TOVS retrieval code (vecrad and
!                  related subroutines)
!
! CALLED FROM:  tovs_tauK
!
! SYSTEM ROUTINES USED:  cos
!
! SUBROUTINES CALLED: planck,dplanck
!
!!INPUT PARAMETERS:
!  tau      - real array (of size maxlev) of transmittance
!  tausun   - real transmittance along solar path to Earth's surface
!  chan     - integer channel index for current calculation
!  tair     - real array (of length maxlev) with profile of 71-level
!                temperatures (K)
!  ir_zen   - real     satellite zenith angle for infrared channels (degrees)
!  emissmw  - real microwave emissivity
!  land     - logical flag  (.true. if land, .false. if water)
!  nlev     - real number of levels to use in profiles
!  rho      - real bidirectional reflectance
!  secsun   - real secant of solar angle for this observation
!  dosun    - logical flag  .true. if solar calculations needed
!  tground  - real skin (ground) temperature (K)
!  tsurfair - real surface air temperature (K)
!  dttau    - real array (of size maxpack) of derivatives of
!                tau with respect to changes in temperature at given layers
!  packed matrix:     first dimension: temperature layer    (see below)
!                    second dimension: transmittance layer
!  dqtau    - real array (of size maxlev) of derivatives of
!                tau with respect to changes in moisture at given layers
!                 -these apply to taus at all layers below the given layer
!  dutau    - real array (of size maxlev) of derivatives of
!                tau with respect to changes in ozone at given layers
!                 -these apply to taus at all layers below the given layer
!  dtsun    - real array (of size maxlev) of derivatives of
!                suntau with respect to changes in temperature at given layers
!  dqsun    - real array (of size maxlev) of derivatives of
!                suntau with respect to changes in moisture at given layers
!  dusun    - real array (of size maxlev) of derivatives of
!                suntau with respect to changes in ozone at given layers
```

```
!
!
!!OUTPUT PARAMETERS:
!
!  bt        -  real brightness temp for channel chan calculated from profile
!  dTb_dT    -  real array (of size maxlev) of d(Tb)/dT
!  dTb_dq    -  real array (of size maxlev) of derivative of brightness
!                  temperature with respect to moisture at given layers
!  dTb_du    -  real array (of size maxlev) of derivative of brightness
!                  temperature with respect to ozone at given layers
!
!!REVISION HISTORY:
!  20Dec95  Meta S.  original routine from combining dTb/dT(simple) and
!                    dTb/dq2 modules
!  02jan96  Meta S.  change to calculate values for 1 channel, call
!                     from tovs_tauK routine
!
!  06Jan96  Meta S.  modify to use tau-level dependent derivatives for
!                       temperature (dttau), to include C term in
!                        fixed gas temperature derivative.
!  12Jan96  Meta S.  edited prologue
!
!  29mar96  Meta S.  minor optimization -  change dttau to packed array
!                       format
!
! Matrix Packing                    (based on SPPTRF manual page)
!  The j-th column of A is stored in the array AP as follows:
!  AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j
!
!     Two-dimensional storage of the matrix A:
!
!       a11 a12 a13 a14    1st dimension: affecting temperature layer
!        0  a22 a23 a24    2nd dimension: affected transmittance layer
!        0   0  a33 a34
!        0   0   0  a44    a12: influence of T(1) on \hat(\tau)(2)
!        ---------
!        | zeros because temperatures in a given layer don't affect
!            transmittance layers above that layer
!
!    Packed storage of the upper triangle of A:
!
!    AP = [ a11, a12, a22, a13, a23, a33, a14, a24, a34, a44 ]
!
!*****************************************************************************

      subroutine dTb_dTQU(tau,tausun,chan,tair,tmptop,ir_zen,
     &     emissmw, land, nlev, rho, secsun, dosun, tground,
     &     tsurfair, dttau, dqtau, dutau, dtsun, dqsun, dusun,
     &     bt, dTb_dT, dTb_dq, dTb_du)

      implicit none

      include "tovsparam.h"
```

```
      integer maxpack
      parameter (maxpack = (maxlev*(maxlev+1))/2 )

!  input variables
      real tau(maxlev)
      real tausun
      integer chan
      real tair(maxlev)
      real tmptop
      real ir_zen
      real emissmw
      real rhosec
      logical land
      integer nlev
      real rho
      real secsun
      logical dosun
      real tground
      real tsurfair
      real dttau(maxpack)
      real dqtau(maxlev)
      real dutau(maxlev)
      real dtsun(maxlev)
      real dqsun(maxlev)
      real dusun(maxlev)

!  output variables
      real bt
      real dTb_dT(maxlev)
      real dTb_dq(maxlev)
      real dTb_du(maxlev)

!  parameters from common

      include "phycons.h"
      include "chacons.h"
      include "matcons.h"

!  function types
      real planck
      real dplanck

!  local variables
      real       btav          ! blackbody temp. of avg layer temperature
      real       btsurf        ! blackbody temp. of surface air temperature
      real       btgnd         ! blackbody temp. of ground temperature
      real       calc_rad      ! calculated radiance
      real       dbtav         !  d(btav) / dT
      real       deltau        ! difference of transmittance in layer
      real       dtauinv       ! inverse transmittance difference
      real       dRd_dTav(maxlev)  !  d (Rdown)/ d Tavg
      real       dRd_dq(maxlev)    !  d (Rdown)/ d H2O
      real       dRd_du(maxlev)    !  d (Rdown)/ d ozone
      real       dTb_dTav(maxlev)  !  d (bright. T)/ d Tavg
```

```
      real       dTb_dR              !  d (bright. T)/ d (radiance)
      real       emiss               !  sfc emissivity for this sounding
      integer    l, m, n, nm1        !  counters
      real       pcon1               !
      real       pcon2               !
      real       rad_down            !  atmos. emission downward flux
      real       rad_sun             !  solar radiation at TOA (?)
      real       sunterm             !  solar contribution to calc. radiance
      real       tav(maxlev)         !  mean layer temperature
!
!  check the following to see if using double makes a difference
      double precision factor, dqfactor, dufactor, dtfactor

      pcon1 = planck1(chan)
      pcon2 = planck2(chan)
!
!  initialize arrays
      do l = 1,maxlev
        dTb_dTav(l) = 0.
        dRd_dTav(l) = 0.       ! derivative from rad_down; work array
        dTb_dq(l) = 0.0
        dTb_du(l) = 0.0
        dRd_dq(l) = 0.0
        dRd_du(l) = 0.0
      enddo

      tav(1) = 0.5 * ( tair(1) + tmptop)
      do l = 2,nlev
        tav(l) = 0.5 * ( tair(l) + tair(l-1))
      enddo
!

      if (irchan(chan)) then

!

!  initialize radiance sum
          deltau = 1.0 - tau(1)
          btav = planck(tav(1), pcon1, pcon2)
          calc_rad = btav * deltau

!**********************************************************************
!  derivs wrt tau
          dTb_dq(1) =  dTb_dq(1) - btav * tau(1) * dqtau(1)
          dTb_du(1) =  dTb_du(1) - btav * tau(1) * dutau(1)
          dTb_dTav(1) =  dTb_dTav(1) - btav * tau(1) * dttau(1)

!  derivs wrt. planck function
          dbtav = dplanck(tav(1), pcon1, pcon2)
          dTb_dTav(1) = dTb_dTav(1) + dbtav * deltau

!  integrate through atmosphere (from top down)
!
          do l = 2,nlev
```

```
            n = (l-1)*l/2          ! set index for packed array
            nm1 = (l-2)*(l-1)/2    !  for levels l and l-1


            btav = planck(tav(l), pcon1, pcon2)
            deltau = tau(l-1) - tau(l)
            calc_rad = calc_rad + btav * deltau

!   derivative terms at l and for levels above l (which contribute to tau)

!   first, derivative for current layer (level) 'l'
            dTb_dq(l) = dTb_dq(l) - btav * tau(l) * dqtau(l)
            dTb_du(l) = dTb_du(l) - btav * tau(l) * dutau(l)
            dTb_dTav(l)=dTb_dTav(l) - btav * tau(l) * dttau(l+n)

!   then, derivatives for layers above 'l' (which contribute to both
!     tau(l) and tau(l-1))  -  loop over temperature, constituent layers
!      to add contribution for current layer of vertical integral

            do m = 1,l-1
              dTb_dq(m) = dTb_dq(m) + btav * deltau * dqtau(m)
              dTb_du(m) = dTb_du(m) + btav * deltau * dutau(m)
              dTb_dTav(m)=dTb_dTav(m) + btav *
     &           ( tau(l-1) * dttau(m+nm1) -
     &             tau(l)   * dttau(m+n) )
            enddo
!
!   then derivs wrt. planck function

            dbtav = dplanck(tav(l),pcon1,pcon2)
            dTb_dTav(l) = dTb_dTav(l) + dbtav * deltau
          enddo

!   prepare to calculate surface contribution
!
          btsurf = planck(tsurfair, pcon1, pcon2)
          btgnd  = planck(tground, pcon1, pcon2)
          rad_down = dfxcon(chan) * btsurf * (1. - tau(nlev))

          n = (nlev-1)*nlev/2       ! index for nlev
!
!   calculate derivative of rad_down wrt. changes in t,q,u at levels above
!      (derivatives of tau(nlev) wrt changes in tau above)
          do m = 1,nlev
            dRd_dq(m) = -dfxcon(chan) * btsurf * tau(nlev)
     &                              * dqtau(m)
            dRd_du(m) = -dfxcon(chan) * btsurf * tau(nlev)
     &                              * dutau(m)
            dRd_dTav(m) = -dfxcon(chan) * btsurf * tau(nlev)
     &                              * dttau(m+n)
          enddo

          if (land) then
```

```
            emiss = em_land(chan)
         else
            emiss = em_water(chan)
         endif

      else if (mwchan(chan)) then

!  initialize radiance sum
         btav = tav(1)
         deltau = 1.0 - tau(1)
         if (tau(1) .ne. 0.0) then
            dtauinv = (1. / tau(1)) - 1.
            rad_down = dtauinv * btav

            dRd_dTav(1) =  dRd_dTav(1) + dtauinv
         endif
         calc_rad = btav * deltau

!  first, deriv wrt. tau
         dTb_dq(1) =  dTb_dq(1) - btav * tau(1) * dqtau(1)
         dTb_du(1) =  dTb_du(1) - btav * tau(1) * dutau(1)
         dTb_dTav(1) =  dTb_dTav(1) - btav * tau(1) * dttau(1)
!
!  for derivative, note that "dqtau", "dutau" are
!    1/tau dtau/dq  1/tau dtau/du   - so need to multiply by tau
!      (i.e. not divide by tau^2 but only by tau)
         if (tau(1) .ne. 0.0) then
            dRd_dq(1) =  dRd_dq(1)
     &                 - btav * 1./tau(1) * dqtau(1)
            dRd_du(1) =  dRd_du(1)
     &                 - btav * 1./tau(1) * dutau(1)
            dRd_dTav(1) =  dRd_dTav(1)
     &                 - btav * 1./tau(1) * dttau(1)
         endif

!  now add contribution from deriv wrt. temperature
         dTb_dTav(1) =  dTb_dTav(1) + deltau

!  integrate through atmosphere (from top down)
         do l = 2,nlev

            n = (l-1)*l/2          ! set index for packed array
            nm1 = (l-2)*(l-1)/2    !  for levels l and l-1

            btav = tav(l)
            deltau = tau(l-1) - tau(l)
            if ((tau(l) .ne. 0.0 ) .and.
     &          (tau(l-1) .ne. 0.0) )  then
               dtauinv = ( 1. / tau(l)) - ( 1./ tau(l-1))
               rad_down = rad_down + dtauinv * btav

            endif
            calc_rad = calc_rad + btav * deltau
```

```
!  tau derivative terms at l and for levels above l (which contribute to tau)
!**********************************************************************
!  first, tau derivative for current layer (level) 'l'
           dTb_dq(l) = dTb_dq(l) - btav * tau(l) * dqtau(l)
           dTb_du(l) = dTb_du(l) - btav * tau(l) * dutau(l)
           dTb_dTav(l) = dTb_dTav(l) -
     &                              btav * tau(l) * dttau(l+n)

!  then, tau derivatives for layers above 'l' (which contribute to both
!    tau(l) and tau(l-1)) -- loop over temperature, constituent layers
!     to add contribution for current layer of vertical integral

           do m = 1,l-1
             dTb_dq(m) = dTb_dq(m) + btav * deltau * dqtau(m)
             dTb_du(m) = dTb_du(m) + btav * deltau * dutau(m)
             dTb_dTav(m) = dTb_dTav(m) + btav *
     &          ( tau(l-1) * dttau(m+nm1) -
     &            tau(l)   * dttau(m+n) )
           enddo

!  then, tau derivatives for rad_down terms (if applicable)
!  for derivative, note that "dqtau", "dutau" are
!    1/tau dtau/dq  1/tau dtau/du   - so need to multiply by tau
!      (i.e. not divide by tau^2 but only by tau)
           if ((tau(l) .ne. 0.0 ) .and.
     &         (tau(l-1) .ne. 0.0) )  then
!
!  tau derivative for current layer 'l'
             dRd_dq(l) =  dRd_dq(l)
     &                  - btav * 1./tau(l) * dqtau(l)
             dRd_du(l) =  dRd_du(l)
     &                  - btav * 1./tau(l) * dutau(l)
             dRd_dTav(l) =  dRd_dTav(l)
     &                  - btav * 1./tau(l) * dttau(l+n)

!**********************************************************************
!  then, tau derivatives for layers above 'l'... note the tau derivative terms
!      in "dtauinv" are the same for each level (needn't be recalculated)


             do m = 1,l-1
                dRd_dq(m) =  dRd_dq(m) -
     &                     btav * dtauinv * dqtau(m)
                dRd_du(m) =  dRd_du(m) -
     &                     btav * dtauinv * dutau(m)
                dRd_dTav(m) =  dRd_dTav(m) - btav *
     &                 (1./tau(l) * dttau(m+n) -
     &                  1./tau(l-1) *  dttau(m+nm1))
             enddo
!
!  then, contribution from temperature term to rad_down derivative

             dRd_dTav(l) =  dRd_dTav(l) + dtauinv
```

```
          endif

!
!  finally, add contribution from temperature term to Tb derivative

          dTb_dTav(l) =  dTb_dTav(l) + deltau

        enddo

!  prepare to calculate surface contribution
        btsurf = tsurfair
        btgnd  = tground
        rad_down = (rad_down + bigbang) * tau(nlev)
        emiss = emissmw

        n = (nlev-1)*nlev/2       ! index for nlev

        do m = 1,nlev
           dRd_dq(m) = dRd_dq(m) * tau(nlev) +
     &                       rad_down * dqtau(m)
           dRd_du(m) = dRd_du(m) * tau(nlev) +
     &                       rad_down * dutau(m)
           dRd_dTav(m) = dRd_dTav(m) * tau(nlev) +
     &                       rad_down * dttau(m+n)
        enddo
       endif


       if (use_refl(chan)) then

        factor =
     &     (cos(ir_zen*deg2rad)*calc_rad*(1.-emiss)*(dfxcon(chan)*2.0)
     &        + emiss * btgnd)

        calc_rad = calc_rad + tau(nlev) * factor

        n = (nlev-1)*nlev/2       ! index for nlev

        do m = 1,nlev
           dqfactor =  (cos(ir_zen*deg2rad)*dTb_dq(m) *
     &                      (1.-emiss) * (dfxcon(chan)*2.0) )
           dTb_dq(m) = dTb_dq(m) +
     &          tau(nlev) * dqtau(m) * factor  +
     &          tau(nlev) * dqfactor

           dufactor =  (cos(ir_zen*deg2rad)*dTb_du(m) *
     &                      (1.-emiss) * (dfxcon(chan)*2.0) )
           dTb_du(m) = dTb_du(m) +
     &          tau(nlev) * dutau(m) * factor  +
     &          tau(nlev) * dufactor

           dtfactor =  (cos(ir_zen*deg2rad)*dTb_dTav(m) *
     &                      (1.-emiss) * (dfxcon(chan)*2.0) )
           dTb_dTav(m) = dTb_dTav(m) +
```

```
     &           tau(nlev) * dttau(m+n) * factor  +
     &           tau(nlev) * dtfactor
         enddo

       else
         calc_rad = calc_rad + tau(nlev) *
     &        (rad_down + emiss * ( btgnd - rad_down ))

         n = (nlev-1)*nlev/2       ! index for nlev

         factor = (rad_down + emiss * ( btgnd - rad_down ))

         do m = 1,nlev
            dTb_dq(m) = dTb_dq(m) +
     &          tau(nlev) * dqtau(m) * factor +
     &          tau(nlev) * (1.-emiss) * dRd_dq(m)
            dTb_du(m) = dTb_du(m) +
     &          tau(nlev) * dutau(m) * factor +
     &          tau(nlev) * (1.-emiss) * dRd_du(m)
            dTb_dTav(m) = dTb_dTav(m) +
     &          tau(nlev) * dttau(m+n) * factor +
     &          tau(nlev) * (1.-emiss) * dRd_dTav(m)
         enddo

       endif

!    Add solar contribution for wavenumber greater than 2000.0

       if (dosun) then
          rhosec = rho / secsun
          rad_sun = scon * planck(tsun, pcon1, pcon2)
          sunterm = rhosec * rad_sun * tausun

          calc_rad = calc_rad + sunterm

          do m = 1,nlev
             dTb_dq(m) = dTb_dq(m) + rhosec * rad_sun * dqsun(m)
             dTb_du(m) = dTb_du(m) + rhosec * rad_sun * dusun(m)
             dTb_dTav(m) = dTb_dTav(m) + rhosec * rad_sun * dtsun(m)
          enddo

       endif

!    now calculate brightness temperature & derivative

       if (irchan(chan)) then
          call brtemp(calc_rad, pcon1, pcon2, bt)
          dTb_dR = 1.0 / dplanck(bt,pcon1,pcon2)
          do m = 1, nlev
            dTb_dq(m) = dTb_dq(m) *  dTb_dR
            dTb_du(m) = dTb_du(m) *  dTb_dR
            dTb_dTav(m) = dTb_dTav(m) *  dTb_dR
          enddo
```

```
      else if (mwchan(chan)) then
!
!   no dTb/dR factor for microwave channels

         bt = calc_rad

      endif

!  now, calculate dTb/dT = dTb_dTav * dTav/dT

      do l = 1,nlev-1
        dTb_dt(l) = 0.5 * ( dTb_dTav(l) + dTb_dTav(l+1))
      enddo
      dTb_dt(nlev) = 0.5 * dTb_dTav(nlev)

      return
      end
```

tovsfix_tauK.f – Fixed gas transmittance contribution

```
!*************************************************************************
!!TOVSFIX_TAUK
!*************************************************************************
!
!!ROUTINE:  tovsfix_tauK
!
!!DESCRIPTION:  Calculate contribution to transmittance by fixed gases
!               using rapid transmittance algorithm coefficients
!               - reference Susskind, et al. 1983 JGR, p 8565
!               and derivative of fixed contrib w.r.t. mean layer temperature
!
! CALLED FROM:   tovs_tauK
!
! SYSTEM ROUTINES USED:  none
!
! SUBROUTINES CALLED: none
!
!!INPUT PARAMETERS:
!  tdif     - real array (of length nlev) with difference between
!                current best estimate of layer mean temperatures and
!                standard atmosphere layer mean temperatures (K)
!  chan     - integer channel number
!  nlev     - integer surface level (also number of levels to use)
!
!
!!OUTPUT PARAMETERS:
!  taufix   - real array of effective mean layer transmittance
!              for fixed gases
!  dttaufix - real array (of size maxpack by maxangs) of
!                   derivative of taufix with respect to temperature
!                   ( 1/taufix d(taufix)/dT)
!  packed matrix:     first dimension: temperature layer    (see below)
!                    second dimension: transmittance layer
!
!!REVISION HISTORY:
!  17jan96  Meta S.  original routine derived from module tovsfix_tau
!                    in GLA retrieval code - move derivative code into
!                    this routine, and vertical sum from tovs_tauprodK
!                    back into tovs_tauangK
!  16feb96  Meta S.  minor optimization:  swap dimensions of dttaufix
!  29mar96  Meta S.  another minor optimization: change dttaufix
!                    into packed array ((maxlev*(maxlev+1))/2,maxangs)
!  18may96  Meta S.  effTdif need not be saved as fct. of angle
!
! Matrix Packing                    (based on SPPTRF manual page)
!  The j-th column of A is stored in the array AP as follows:
!  AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j
!
!     Two-dimensional storage of the matrix A:
!
!       a11 a12 a13 a14    1st dimension: affecting temperature layer
```

```
!        0  a22 a23 a24    2nd dimension: affected transmittance layer
!        0   0  a33 a34
!        0   0   0  a44    a12: influence of T(1) on \hat(\tau)(2)
!        ^^^^^^^^^
!        | zeros because temperatures in a given layer don't affect
!            transmittance layers above that layer
!
!    Packed storage of the upper triangle of A:
!
!    AP = [ a11, a12, a22, a13, a23, a33, a14, a24, a34, a44 ]
!
!*****************************************************************
      subroutine tovsfix_tauK(tdif, chan, nlev, taufix, dttaufix)

      implicit none

      include "tovsparam.h"

      integer maxpack
      parameter (maxpack = (maxlev*(maxlev+1))/2 )
!
!  variables passed into routine
!
      integer nlev, chan
      real tdif(maxlev)
!
!  variables output from routine
      real taufix(maxlev,maxangs)
      real dttaufix(maxpack,maxangs)

!  constants, quasi-constants, and coefficients
      include "taucoef.h"

!  local variables
      integer k,l,m,n

! difference of effective mean temperature above given level
!   from rapid algorithm standard temperature value
      real effTdif
      real cltau

!
!  Susskind et al 1983  algorithm for transmittance from fixed gases
!  Effective mean temperature  above P -> \tilde{T} in equation (A6)
!
!  taucfw: precomputed non-varying part of \tilde{T} integral
!  taucfc: includes term with division by (1 - \tau^o)
!
!    here effTdif ->  \tilde{T} - \tilde{T}^o  or
!      difference between effective mean temperatures for the
!      current temperature profile and the standard temperature profile
!  Then:
!     tau = A + B (T - T^o) + C ( \tilde{T} - \tilde{T}^o)      (A7)
!
```

```
      do k = 1, maxangs
         effTdif = 0.
         do l = 1,nlev

            n = (l-1)*l/2     ! set index for packed array

            effTdif = effTdif + taucfw(l,k,chan) * tdif(l)

            taufix(l,k) = taucfa(l,k,chan) + taucfb(l,k,chan) * tdif(l)
     &           + taucfc(l,k,chan) * effTdif

! layer transmittance bounded between 0 and 1
            taufix(l,k) = amax1( taufix(l,k), 0.0)
            taufix(l,k) = amin1( taufix(l,k), 1.0)
!
!  if taufix is zero, then set temperature derivative also to zero
!
            if (taufix(l,k) .eq. 0.0) then
               do m = 1,l
                  dttaufix(m+n,k) = 0.0
               enddo
            else
!
!  effTdif is weighted sum of temperatures over all layers above
!   current layer - so need to loop over those layers to fill in
!   their contribution to this layer's effective transmittance
!
               cltau = taucfc(l,k,chan) / taufix(l,k)
               do m = 1,l
                  dttaufix(m+n,k) = cltau * taucfw(m,k,chan)
               enddo
!
!  temperature contribution in 'taucfb' term is only for transmittance
!  in the same layer
               dttaufix(l+n,k) = dttaufix(l+n,k) +
     &            (taucfb(l,k,chan) / taufix(l,k))

            endif

         enddo
         do l = nlev+1,maxlev

            n = (l-1)*l/2     ! set index for packed array
            taufix(l,k) = 0.0
!           do m = 1,l
!              dttaufix(m+n,k) = 0.0
            do m = n+1,n+l
               dttaufix(m,k) = 0.0
            enddo
         enddo
      enddo

      return
      end
```

tovsh2o_tauK.f – Water vapor transmittance contribution

```
!*************************************************************************
!!TOVSH2O_TAUK
!*************************************************************************
!
!!ROUTINE:  tovsh2o_tauK
!
!!DESCRIPTION:  Calculate contribution to transmittance by water vapor
!               using rapid transmittance algorithm coefficients
!               and derivatives with respect to H2O and temperature
!               - reference Susskind, et al. 1983 JGR, p 8564
!
! CALLED FROM:   tovs_tauK
!
! SYSTEM ROUTINES USED:  exp
!
! SUBROUTINES CALLED: none
!
!!INPUT PARAMETERS:
!  h2ocd    - real array (of length maxlev) with water vapor layer
!                column densities (g cm^-2)
!  e        - real array (of length maxlev) of H2O vapor pressure (atm)
!  dqh2ocd  - real array (of length maxlev) of derivative of H2O
!               column density with respect to specific humidity
!  dqe      - real array (of length maxlev) of derivative of H2O
!               vapor pressure with respect to specific humidity
!  chan     - integer channel number
!  nlev     - integer surface level (also number of levels to use)
!  tav   - real array (of length maxlev) with temperature profile (K)
!  celsius  - real array (of length maxlev) with (T - 273) (approx.
!                temperature in celsius)
!
!!OUTPUT PARAMETERS:
!  tauh2o   - real array (of size maxlev by maxangs) of effective h2o
!                transmittance
!  dqtauh2o - real array (of size maxlev by maxangs) of logarithmic
!                derivative of tauh2o with respect to H2O specific humidity
!  dttauh2o - real array (of size maxlev by maxangs) of logarithmic
!                derivative of tauh2o with respect to temperature
!
!!REVISION HISTORY:
!  30nov95  Meta S.  original routine derived from module tovsh2o_tau
!  07dec95  Meta S.  added loop to fill in zeros below sfc level for
!                       derivative
!  19dec95  Meta S.  add 'factor' in calculation of 'cont', 'dqcont'
!                    (fix bug where 'cont' was reset before being used
!                       in calculation of 'dqcont')
!  20Dec95  Meta S.  this version calculates temperature derivative
!                       wrt 'tav'  (simplifies coding)
!  15Mar96  Meta S.  amplified comments - tiny fix with 'dtcont'
!  18may96  Meta S.  revisions to move column density and vapor pressure
!                      calculations up to tovs_tauK
```

```
!  20Jul96  Meta S.  corrected description of output parameters
!
!**********************************************************************

      subroutine tovsh2o_tauK( h2ocd, e, dqh2ocd, dqe, chan, nlev,
     &          tav, celsius, tauh2o, dqtauh2o, dttauh2o)

      implicit none

      include "tovsparam.h"

!
!  variables passed into routine
      real     h2ocd(maxlev)
      real     e(maxlev)
      real     dqh2ocd(maxlev)
      real     dqe(maxlev)
      integer  chan
      integer  nlev
      real     tav(maxlev)
      real     celsius(maxlev)

!  variable output from routine
      real     tauh2o(maxlev,maxangs)
      real     dqtauh2o(maxlev,maxangs)
      real     dttauh2o(maxlev,maxangs)

!  constants, quasi-constants, and coefficients

      include "taucoef.h"
      include "chacons.h"
      include "grdcons.h"

!  local storage
      real     cont(maxlev)
      real     deffh2o(maxlev)
      real     dqcont(maxlev)
      real     dtcont(maxlev)
      real     dqtaulog
      real     dttaulog
      real     effh2o(maxlev)
      integer  k, l
      real     taulog

!  statement function definitions

      real     quikexp
      real     X

      quikexp(X) = 1.0 + X * (1.0 + X * ( .5 + X * .16666667 ) )



! for IR channel, calculate continuum (except for angle contribution)
```

```
! and derivatives with respect to specific humidity and temperature
!
!    see Susskind and Searl (1978) J. Quant. Spect. Radiat. Transfer
!
!  kH2O(\nu,i) = \sum_l k2(\nu,T) e(l) U(l)
!
!  U(l) (as above) -->  h2ocd = delprs * h2omid
!
!  degchn = - dk2/dT for channel k2(T) = k2(contmp)*(1-degchn)(T-contmp)
!

      if (irchan(chan)) then

         do l = 1,nlev
            cont(l) = e(l) * abscof(chan) * h2ocd(l) *
     &           (1.0 - degchn(chan) * (tav(l) - contmp(chan)))
            dtcont(l) = - e(l) * abscof(chan) * h2ocd(l) *
     &          degchn(chan)
            dqcont(l) = abscof(chan) *
     &             ( e(l) * dqh2ocd(l) + h2ocd(l) * dqe(l) ) *
     &           (1.0 - degchn(chan) * (tav(l) - contmp(chan)))

         enddo


       else
          do l = 1,nlev
             cont(l) = 0.0
             dqcont(l) = 0.0
             dtcont(l) = 0.0
          enddo
       endif

       if ( taucfm(chan) .lt. 0.99) then
          do l = 1,nlev
             deffh2o(l) =  taucfm(chan) * (h2ocd(l)**(taucfm(chan)-1.))
     &              * dqh2ocd(l)
             effh2o(l) = h2ocd(l) ** taucfm(chan)
          enddo
       else
          do l = 1,nlev
             effh2o(l) = h2ocd(l)
             deffh2o(l) = dqh2ocd(l)
          enddo
       endif

!
!  From (A5) in Susskind, et al. 1983  (page 8564)
!
! tau(H2O) = exp ( - ( D (1 - E (T - 273)) h2ocd**M)
!         {   * exp(-sec \theta * KcontinuumH2O) of course }
!
!     see also ON-9608, eq32
!
```

```
!  From GLA retrieval code:
!    TAULOG(II,K)  = (-1.0 - TAUCFE(K,YCHA) * TMPXCES(II))
!                    * TAUCFD(LEVL,K,YCHA) * EFFH2O(II)
!
!    so perhaps TAUCFE = -E
!
      do k = 1, numangs
         do l = 1,nlev
            taulog = ( -1.0 - taucfe(k,chan) * celsius(l))
     &           * taucfd(l,k,chan) * effh2o(l)
            taulog = taulog - secang(k) * cont(l)
!           taulog = amin1(taulog, 0.0) !<-- modified for derivative calc.

            if (taulog .gt. 0.0) then
!
              dqtauh2o(l,k) = 0.0       ! if taulog set to zero, derivatives
              dttauh2o(l,k) = 0.0       ! should be zero also
!             taulog = 0.0
              tauh2o(l,k) = 1.0
!
            else
!
!  set derivatives w.r.t. ozone and temperature of mean layer transmittance
!
              dqtaulog = ( -1.0 - taucfe(k,chan) * celsius(l))
     &           * taucfd(l,k,chan) * deffh2o(l)
              dqtaulog = dqtaulog - secang(k) * dqcont(l)
              dttaulog = - taucfe(k,chan)
     &           * taucfd(l,k,chan) * effh2o(l)
              dttaulog = dttaulog - secang(k) * dtcont(l)
              tauh2o(l,k) = quikexp( taulog )
!
!  note on derivatives:  the factor exp(taulog) is not included here
!                        since we would divide by exp(taulog)
!                        when creating full tau derivative
!  we are calculating logarithmic derivative here ->
!   1/tauh2o  d(tauh2o)/dq  and 1/tauh2o d(tauh2o)/dt

              dqtauh2o(l,k) = dqtaulog
              dttauh2o(l,k) = dttaulog
            endif
         enddo
         do l = nlev+1,maxlev
            dqtauh2o(l,k) = 0.0
            dttauh2o(l,k) = 0.0
            tauh2o(l,k) = 1.0      ! shouldn't be used, anyway
         enddo
      enddo

      return
      end
```

tovsozo_tauK.f – Ozone transmittance contribution

```
!************************************************************************
!!TOVSOZO_TAUK
!************************************************************************
!
!!ROUTINE:  tovsozo_tauK
!
!!DESCRIPTION:  Calculate contribution to transmittance by ozone
!               using rapid transmittance algorithm coefficients
!               - reference Susskind, et al. 1983 JGR, p 8564
!               and derivatives with respect to ozone and temperature
!
! CALLED FROM:   tovs_tauK
!
! SYSTEM ROUTINES USED:  exp
!
! SUBROUTINES CALLED: none
!
!!INPUT PARAMETERS:
!  ozomid   - real array (of length mlev) with ozone column
!                densities (g/cm**2)
!  chan     - integer channel number
!  nlev     - integer number of levels (top to surface)
!  celsius  - real array (of length mlev) with profile of (T-273) (K)
!
!!OUTPUT PARAMETERS:
!  tauozo   - real array (of size maxlev by maxangs) of effective
!             ozone transmittance
!  dutauozo - real array (of size maxlev by maxangs) of derivative
!                of tauozo with respect to ozone
!  dttauh2o - real array (of size maxlev by maxangs) of derivative
!                of tauh2o with respect to temperature
!
!!REVISION HISTORY:
!  01dec95  Meta S.  original routine derived from module tovsozo_tau
!  11dec95  Meta S.  bug fixes: some typos (using H20 coeffs for deriv.
!                      instead of 03 coeffs)
!  20Dec95  Meta S.  this version calculates temperature derivative
!                       wrt 'tav'  (simplifies coding)
!  14Mar96  Meta S.  added more comments
!  10Jul96  Meta S.  corrected description of output parameters
!
!************************************************************************

      subroutine tovsozo_tauK(ozomid, chan, nlev, celsius, tauozo,
     &              dutauozo, dttauozo)

      implicit none
!
      include "tovsparam.h"
!
!  variables passed into routine
```

```
      real     ozomid(maxlev)
      integer  chan
      integer  nlev
      real     celsius(maxlev)
!
!  variables output from routine
      real     tauozo(maxlev,maxangs)
      real     dttauozo(maxlev,maxangs)
      real     dutauozo(maxlev,maxangs)
!
!  constants, quasi-constants, and coefficients

      include "taucoef.h"

!  local storage
      real     deff03(maxlev)
      real     dttaulog
      real     dutaulog
      real     eff03(maxlev)
      integer  k, l
      real     taulog


!  statement function definitions
      real     quikexp
      real     X

      quikexp(X) = 1.0 + X * (1.0 + X * ( .5 + X * .16666667 ) )
!
!  calculate 'eff03' as (ozone conc.) ** taucfN
!
!  following is formula used in GLA code:
!      if (taucfn(chan) .lt. 0.99) then
!         do l = 1,nlev
!            eff03(l) = taucfn(chan) * alog( ozomid(l))
!            eff03(l) = exp( eff03 )
!         enddo
!      else
!         do l = 1,nlev
!            eff03(l) = ozomid(l)
!         enddo
!      endif
!
!  deff03 is derivative of eff03 with respect to ozone

      do l = 1, nlev
         eff03(l) = ozomid(l) ** taucfn(chan)
         deff03(l) = taucfn(chan) * (ozomid(l)**(taucfn(chan)-1.))
      enddo

!
!  From (A5) in Susskind, et al. 1983  (page 8564)
!
! tau(ozone) = exp ( - ( F (1 - G (T - 273)) ozomid**N)
```

```
!                                                  see also ON-9608  eq 33
!
! from GLA retrieval code:
! TAULOG(II,K) = (-1.0 - TAUCFG(K,YCHA) * TMPXCES(II))
!   * TAUCFF(LEVL,K,YCHA) * EFFO3(II)* TAUCFF(LEVL,K,YCHA) * EFFO3(II)
!
!  so sign of TAUCFG = -G , perhaps...
!

      do k = 1,numangs
         do l = 1, nlev
            taulog = ( -1.0 - taucfg(k,chan) * celsius(l))
     &           * taucff(l,k,chan) * effO3(l)
!           taulog = amin1 (taulog, 0.0) !<-- modified for derivative calc.
!

            if (taulog .gt. 0.0) then
!
              dutauozo(l,k) = 0.0       ! if taulog set to zero, derivatives
              dttauozo(l,k) = 0.0       ! should be zero also.
!             taulog = 0.0
              tauozo(l,k) = 1.0

            else
!
!  set derivatives w.r.t. ozone and temperature of mean layer transmittance
!
              dutaulog = ( -1.0 - taucfg(k,chan) * celsius(l))
     &           * taucff(l,k,chan) * deffO3(l)
              dttaulog = -taucfg(k,chan)
     &           * taucff(l,k,chan) * deffO3(l)
              tauozo(l,k) = quikexp( taulog )
!
!  note on derivatives:  the factor exp(taulog) is not included here
!                         since we would divide by exp(taulog)
!                         when creating full tau derivative
!  we are calculating logarithmic derivative  1/tauozo d(tauozo)/dT
!    and 1/(tauozo) d(tauozo)/du
!
              dutauozo(l,k) = dutaulog
              dttauozo(l,k) = dttaulog
            endif
         enddo
         do l = nlev+1,maxlev
            dutauozo(l,k) = 0.0
            dttauozo(l,k) = 0.0
            tauozo(l,k) = 1.0       !shouldn't be used, anyway
         enddo
      enddo

      return
      end
```

tovs_tauK.f – Driver for transmittance derivative calculation

```
!*************************************************************************
!!TOVS_TAUK
!*************************************************************************
!
!!ROUTINE:  tovs_tauK
!
!!DESCRIPTION:  Calculate transmittance for given channel numbers
!               for a specified temperature, moisture, and ozone profile
!               and derivatives with respect to temperature, moisture
!               and ozone to be used in Jacobian calculation
!               adding in angle calculation
!               Based on module tovs_tau
!
!
! SYSTEM ROUTINES USED:  abs, cos
!
! SUBROUTINES CALLED:
!     tovsfix_tauK  - transmittance contrib. by fixed gases and derivative
!     tovsh2o_tauK  - transmittance contrib. by water vapor and derivative
!     tovsozo_tauK  - transmittance contrib. by ozone and derivative
!     tovs_tauprodK - take product of fixed, water vapor & ozone
!                        contribs and derivatives
!     tovs_tauangK  - interpolate transmittance to obs. zenith angle &
!                          calculate consistent derivative terms
!     dTb_dTQU      - calculate brightness temperature and Jacobian
!
!
!!INPUT PARAMETERS:
!  tair     - real array (of length maxlev) with temperature profile (K)
!  tmptop   - real temperature (K) at "top of atmosphere"
!  h2omid   - real array (of length maxlev) with water vapor layer specific
!               humidities (KG/KG)
!  ozomid   - real array (of length maxlev) with ozone column densities
!               (g/cm**2)
!  ps       - real surface pressure (mb)
!  nchan    - integer number of channels to calculate
!  chanarr  - integer array (of length maxcha) of channels to calculate
!  nlev     - integer surface level (also number of levels to use)
!  mw_zen   - real     satellite zenith angle for microwave channels
!  ir_zen   - real     satellite zenith angle for infrared channels
!  dayflag  - logical  true if day ( sun angle > SUNSET = 85.0E0)
!  secsun   - real  1. / cos( sun angle)
!  emissmw  - real microwave surface emissivity
!  land     - logical flag  (.true. if land, .false. if water)
!  rho      - real bidirectional reflectance
!  tground  - real skin (ground) temperature (K)
!  tsurfair - real surface air temperature (K)
!
!!OUTPUT PARAMETERS:
!
!  tau    - real array (of size maxlev by maxcha) of transmittance along
```

```
!              path to satellite.
!  tausun - real array (of size maxcha) of transmittance along path from
!              sun
!  bt         -  real array (of size maxcha) of brightness temperatures for
!                  for channels in chanarr, calculated from profile
!  dTbdT      -  real array (of size maxlev by maxcha) of d(Tb)/dT
!                  derivative of brightness temperature with respect to
!                  temperature at given levels
!  dTbdq      -  real array (of size maxlev by maxcha) of derivative
!                   of brightness temperature with respect to specific
!                   humidity at given layers
!  dTbdu      -  real array (of size maxlev by maxcha) of derivative
!                   of brightness temperature with respect to ozone at
!                   given layers
!  errflag - logical returns .true. if error
!
!!REVISION HISTORY:
!  04dec95  Meta S.  original routine derived from subroutine tovs_tau
!  07dec95  Meta S.  carry derivatives through to  tovs_tauNK
!  15dec95  Meta S.  replace non-interpolated tovs_tauNK with
!                       interpolating tovs_tauangK
!  20dec95  Meta S.  add changes to create temperature derivatives
!  02jan96  Meta S.  fold in rest of jacobian calculation - put in call
!                        to dTb_dTQU2
!  08jan96  Meta S.  changes for C term dependence of temperature
!
!  29mar96  Meta S.  minor optimization: 1-d packed array for temperature
!                     derivative terms
!  18may96  Meta S.  moved calculation of column H2O density and vapor
!                     pressure from tovsh2o_tau into this routine
!
!*************************************************************************
      subroutine tovs_tauK(tair, tmptop, h2omid, ozomid, ps, nchan,
     &     chanarr, nlev, mw_zen, ir_zen, dayflag, secsun,
     &     emissmw, land, rho,  tground, tsurfair,
     &     tau, tausun, bt, dTbdT, dTbdq, dTbdu, errflag)

      implicit none

      include "tovsparam.h"

      integer maxpack
      parameter (maxpack = (maxlev*(maxlev+1))/2 )

!
!  variables passed into routine
!
      real     tair(maxlev)
      real     tmptop
      real     h2omid(maxlev)
      real     ozomid(maxlev)
      real     ps
      integer  nchan
      integer  chanarr(maxcha)
```

```
      integer  nlev
      real     mw_zen
      real     ir_zen
      logical  dayflag
      real     secsun
      real     emissmw
      logical  land
      real     rho
      real     tground
      real     tsurfair

!
!  variables output from routine
      real     tau(maxlev, maxcha)
      real     tausun(maxcha)
      real     bt(maxcha)
      real     dTbdT(maxlev,maxcha)
      real     dTbdq(maxlev,maxcha)
      real     dTbdu(maxlev,maxcha)
      logical  errflag

!  constants, quasi-constants, and coefficients

      include "matcons.h"
      include "grdcons.h"
      include "chacons.h"
      include "phycons.h"

!  local variables
      integer l                      ! counters
      integer ichan
      integer chan

      real tav(maxlev)               ! mean layer temperature
      real tdif(maxlev)              ! difference from rapid alg std. temp.
      real celsius(maxlev)           ! difference from 273


      real effangmw                  !  microwave  and
      real seceffmw                  !
      real effangir                  !  IR channel zenith angles and secants
      real seceffir                  !
      real effang                    !
      real seceff                    !

      real h2ocd (maxlev)            !  H2O column density
      real e(maxlev)                 !  H2O vapor pressure

      real dqh2ocd(maxlev)           ! deriv. of column density wrt spec.humid.
      real dqe(maxlev)               ! deriv. of vapor press wrt spec.humid.

! effective mean layer transmittance from fixed gas contribution
!  and logarithmic derivative w.r.t. temperature
      real taufix(maxlev,maxangs)
```

```
      real dttaufix(maxpack,maxangs)

! effective mean layer transmittance from water vapor contribution
!  and logarithmic derivative w.r.t. temperature, specific humidity
      real tauh2o(maxlev,maxangs)
      real dqtauh2o(maxlev,maxangs), dttauh2o(maxlev,maxangs)

! effective mean layer transmittance from ozone contribution
!  and logarithmic derivative w.r.t. temperature, ozone column density
      real tauozo(maxlev,maxangs)
      real dutauozo(maxlev,maxangs), dttauozo(maxlev,maxangs)

! effective mean layer transmittance at rapid algorithm angles and
! logarithmic derivatives w.r.t temperature, specific humidity,
! and ozone column density
      real tauang(maxlev,maxangs)
      real dttauang(maxpack,maxangs)
      real dqtauang(maxlev,maxangs)
      real dutauang(maxlev,maxangs)

! derivatives (w.r.t. temperature, specific humidity, and ozone) of
! effective mean layer transmittances along path from satellite zenith
! angle and sun angle
      real dttau(maxpack), dtsun(maxlev)
      real dqtau(maxlev), dqsun(maxlev)
      real dutau(maxlev), dusun(maxlev)

! calculated tau, tausun, derivatives and brightness temperature
! for current channel
      real ctau(maxlev), ctausun
      real cdTbdT(maxlev), cdTbdq(maxlev), cdTbdu(maxlev)
      real cbt

! flag for solar calculation
      logical dosun

      errflag = .false.
!
!
!  calculate quantites which are common for all channels
!  tav - mean layer temperature for RT calculation
!  tdif   - difference from rapid algorithm's std. atmos. mean temp.
!  celsius - difference from 273 K
!     (note this is not Celsius; 273 K is used in rapid algorithm)

      tav(1) = 0.5 * ( tmptop + tair(1))
      tdif(1) = tav(1) - tstd_av(1)
      celsius(1) = tav(1) - 273.
      do l = 2,nlev
          tav(l) = (tair(l-1)+tair(l)) * 0.5
          tdif(l) = tav(l) - tstd_av(l)
          celsius(l) = tav(l) - 273.
      enddo
```

```
!  compute mid-level h2o column densities (g/cm**2)
!
!   delprs = {\delta p } * 10 / grav  -->  density(air) * {\delta z}
!
!   delprs * h2omid(spec. humid) --> column density effh2o  U(l)

      do l = 1,nlev-1
        h2omid(l) = amax1( h2omid(l), shmin)
        h2ocd(l) = delprs(l) * h2omid(l)
        dqh2ocd(l) = delprs(l)
      enddo

      h2ocd(nlev) = (ps - pres(nlev-1)) * gravcon * h2omid(l)
      dqh2ocd(nlev) = (ps - pres(nlev-1)) * gravcon

!  e(l) = H2O partial pressure (in atm)
!          =  (airmol/h2omol) * P(l) / (1013.25 mb per atm)
!                    * h2omid (spec. humidity)
!
!     econ = airmol / (h2omol * atmosph)  !  set in phycons.h

      do l = 1,nlev-1
         e(l) = presa(l) * econ * h2omid(l)
         dqe(l) = presa(l) * econ
      enddo
      e(nlev) = (ps + pres(nlev-1)) * 0.5 * econ * h2omid(nlev)
      dqe(nlev) = (ps + pres(nlev-1)) * 0.5 * econ


!  set up zenith angle factors for microwave and IR channels

      effangmw = abs( deg2rad * mw_zen )
      seceffmw = 1. / cos( effangmw )
      effangir = abs( deg2rad * ir_zen )
      seceffir = 1. / cos( effangir )

!  loop over requested channels

      do ichan = 1,nchan
         chan = chanarr(ichan)
         if (chan .gt. maxcha) then
            print *,' tovs_tau:  bad channel requested'
            errflag = .true.
            return
         else

!  set appropriate zenith angle (effang) and secant for IR or MW channel

            if (irchan(chan)) then
               effang = effangir
               seceff = seceffir
            else if (mwchan(chan)) then
               effang = effangmw
               seceff = seceffmw
```

```
            endif

!   set flag to show whether solar radiation calculations are performed
            dosun = ( freq(chan) .gt. 2000. ) .and. dayflag

!   Calculate logarithmic derivatives of effective mean layer transmittances
!   for fixed gas, water vapor and ozone contributions.  The logarithmic
!   derivatives are used because total transmittance from top of atmosphere
!   to a level is the -product- of layer values of fixed gas, water vapor
!   and ozone layer contributions.

!   calculate contribution to effective layer transmittances from fixed gases
!    (taufix) and logarithmic derivative of taufix w.r.t.
!     layer temperature     1/taufix d(taufix)/dTavg  : dttaufix

            call tovsfix_tauK(tdif, chan, nlev, taufix, dttaufix)

!   calculate contribution to effective layer transmittances from water vapor
!   (tauh2o) and logarithmic derivatives of tauh2o w.r.t.
!      specific humidity    1/tauh2o d(tauh2o)/dq     : dqtauh2o   and
!      layer temperature    1/tauh2o d(tauh2o)/dTavg  : dttauh2o
!
            call tovsh2o_tauK( h2ocd, e, dqh2ocd, dqe, chan, nlev,
     &          tav, celsius, tauh2o, dqtauh2o, dttauh2o)

!   calculate contribution to effective layer transmittances from ozone
!   (tauozo) and logarithmic derivative of tauozo w.r.t.
!      ozone column density   1/tauozo d(tauozo)/d
!
            call tovsozo_tauK( ozomid, chan, nlev, celsius, tauozo,
     &             dutauozo, dttauozo)

! take product of fixed, h2o, and ozone contributions to get
!    mean layer transmittances & logarithmic derivatives for
!    rapid algorithm angles
!
            call tovs_tauprodK(nlev, taufix, tauh2o, tauozo,
     &           dqtauh2o, dttauh2o, dutauozo, dttauozo, dttaufix,
     &           tauang, dqtauang, dutauang, dttauang )


!   interpolate transmittance to observed zenith angle
!   and calculate consistent logarithmic derivative terms
!
            call tovs_tauangK(tauang, chan, nlev, effang, seceff,
     &         dosun, secsun, dqtauang, dutauang, dttauang,
     &         ctau, ctausun, dutau, dusun, dqtau, dqsun,
     &         dttau, dtsun)

!   calculate temperature, moisture and ozone Jacobians
!    and brightness temperature
!
            call dTb_dTQU(ctau,ctausun,chan,tair,tmptop,ir_zen,
     &         emissmw, land, nlev, rho, secsun, dosun, tground,
```

```
     &          tsurfair, dttau, dqtau, dutau, dtsun, dqsun, dusun,
     &          cbt, cdTbdT, cdTbdq, cdTbdu)

!
!  fill arrays with calculated brightness temperature, tau and tausun
!   and derivatives
!
            do l = 1,nlev
               tau(l,chan) = ctau(l)
               dTbdT(l,chan) = cdTbdT(l)
               dTbdq(l,chan) = cdTbdq(l)
               dTbdu(l,chan) = cdTbdu(l)
            enddo
            do l = nlev+1,maxlev
               tau(l,chan) = ctau(nlev)
               dTbdT(l,chan) = 0.0
               dTbdq(l,chan) = 0.0
               dTbdu(l,chan) = 0.0
            enddo
            tausun(chan) = ctausun
            bt(chan) = cbt

         endif


      enddo                                  ! end of channel loop

      return
      end
```

tovs_tauangK.f – Interpolate transmittance to zenith angle

```
!*************************************************************************
!!TOVS_TAUANGK
!*************************************************************************
!
!!ROUTINE:  tovs_tauangK
!
!!DESCRIPTION:  Interpolate computed transmittances to satellite
!                zenith angle and combine transmittance derivatives
!                consistent with angle interpolation and tau calculation
!
! CALLED FROM:   tovs_tauK
!
! SYSTEM ROUTINES USED:  exp, cos, acos
!
! SUBROUTINES CALLED: none
!
!!INPUT PARAMETERS:
!  tauang   - real matrix (of size maxlev by maxangs) of mean layer
!               transmittance calculated for different zenith angles
!  chan     - integer channel number
!  nlev     - integer surface level (also number of levels to use)
!  effang   - real abs(zenith angle) in radians
!  seceff   - real 1. / cos(effang)
!  dosun    - logical flag  .true. if calculations for sun are to be done
!  secsun   - real secant of solar angle for this observation
!  dqtauang - real array (of size maxlev by maxangs) of derivative of
!               tauang wrt. H2O specific humidity
!  dutauang - real array (of size maxlev by maxangs) of derivative of
!               tauang wrt. ozone
!  dttauang - real array (of size maxpack by maxangs) of derivative of
!               tauang wrt. mean layer temperature
!  packed matrix:     first dimension: temperature layer    (see below)
!                    second dimension: transmittance layer
!
!!OUTPUT PARAMETERS:
!  tau      - real array (of length maxlev) of mean layer transmittance
!                for satellite zenith angle
!  tausun   - real mean layer transmittance
!                for solar radiation
!  dqtau    - real array (of size maxlev) of logarithmic derivatives of
!                tau with respect to changes in moisture at given layers
!                 -these apply to taus at all levels below the given layer
!  dutau    - real array (of size maxlev) of logarithmic derivatives of
!                tau with respect to changes in ozone at given layers
!                 -these apply to taus at all levels below the given layer
!  dttau    - real array (of size maxpack) of logarithmic derivatives of
!                tau with respect to changes in temperature at given layers
!  packed matrix:     first dimension: temperature layer    (see below)
!                    second dimension: transmittance layer
!  dqsun    - real array (of size maxlev) of logarithmic derivatives of
!               suntau with respect to changes in moisture at given layers
```

```
!  dusun    - real array (of size maxlev) of logarithmic derivatives of
!               suntau with respect to changes in ozone at given layers
!  dtsun    - real array (of size maxlev) of logarithmic derivatives of
!               suntau with respect to changes in temperature at given layers
!
!!REVISION HISTORY:
!  15dec95  Meta S.  original routine derived from subroutine tauang
!                    (derivs. based on earlier routine w/o interpolation)
!  20dec95  Meta S.  add changes to pass in temperature derivatives
!                      consider only temperature deriv from B term
!                      of tovsfix_tau (which would be 1/taufix *taucfb)
!  08jan96  Meta S.  include 'C' term derivative; dttaufix and dttau are
!                      now dependent on tau level also
!  12jan96  Meta S.  move derivative products to tovstau_prod.f (for
!                      clarity, mostly), edit prologue
!  17jan96  Meta S.  array notation in dttauang, dttau now consistent;
!                      sum of temp. derivatives over transmittance layers
!                      now back in this routine after interpolation (where
!                      it should be!)
!  16feb96  Meta S.  minor optimization: swap dimensions of dttau*
!  29mar96  Meta S.  another minor optimization: change dttaufix
!                    into packed array ((maxlev*(maxlev+1))/2,maxangs)
!
! Matrix Packing                    (based on SPPTRF manual page)
!  The j-th column of A is stored in the array AP as follows:
!  AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j
!
!     Two-dimensional storage of the matrix A:
!
!       a11 a12 a13 a14    1st dimension: affecting temperature layer
!        0  a22 a23 a24    2nd dimension: affected transmittance layer
!        0   0  a33 a34
!        0   0   0  a44    a12: influence of T(1) on \hat(\tau)(2)
!        ^^^^^^^^^^
!        | zeros because temperatures in a given layer don't affect
!             transmittance layers above that layer
!
!    Packed storage of the upper triangle of A:
!
!    AP = [ a11, a12, a22, a13, a23, a33, a14, a24, a34, a44 ]
!
!
!*****************************************************************

      subroutine tovs_tauangK(tauang, chan, nlev, effang, seceff,
     &         dosun, secsun, dqtauang, dutauang, dttauang,
     &         tau, tausun, dutau, dusun, dqtau, dqsun,
     &         dttau, dtsun)

      implicit none

      include "tovsparam.h"

      integer maxpack
```

```
      parameter (maxpack = (maxlev*(maxlev+1))/2 )

!  variables passed into routine
      real tauang(maxlev,maxangs)
      integer  chan
      integer  nlev
      real     effang
      real     seceff
      logical  dosun
      real     secsun
      real     dqtauang(maxlev,maxangs)
      real     dutauang(maxlev,maxangs)
      real     dttauang(maxpack,maxangs)

!  variables output from routine
      real     tau(maxlev)
      real     tausun
      real     dutau(maxlev)
      real     dusun(maxlev)
      real     dqtau(maxlev)
      real     dqsun(maxlev)
      real     dttau(maxpack)
      real     dtsun(maxlev)

!  constants, quasi-constants, and coefficients
!  (in common block or passed into routine)

      include "taucoef.h"
      include "grdcons.h"
      include "chacons.h"

!  local variables
      real     anglsun
      real     efcos
      real     efsun
      integer  k, l, m, n, nm1, nnm1
      integer  nangm1
      integer  nfang
      integer  nfsun
      real     sat_factor
      real     sfactor
      real     sun_factor
      real     tauscat
      real     tauint              ! interpolated tau: \tilde{tau}(\theta)
      real     tauinti             !   1. / \tilde{tau}(\theta)

!  statement functions
      real     quikexp
      real     X
      quikexp(X) = 1.0 + X * (1.0 + X * ( .5 + X * .16666667 ) )

      nangm1 = numangs - 1
      tausun = 0.       ! set this in case of .not.irchan or .not.dosun
      do l = 1,maxlev
```

```
        dusun(1) = 0.
        dqsun(1) = 0.
        dtsun(1) = 0.
      enddo

!
! find which angles of rapid algorithm bracket the observed zenith angle
!  and calculate factor for interpolation in sec(zenith angle)
!
!  first consider transmittance layer 1, then loop over rest of layers
!
      nfang = 1
      do k = 2,nangm1
         if (effang .gt. angle(k) ) nfang = k
      enddo

      sat_factor = ( seceff - secang(nfang) ) /
     &                 (secang(nfang+1) - secang(nfang) )

      tau(1) = amax1( 0.0, (tauang(1, nfang)
     &     + sat_factor * (tauang(1,nfang+1)-tauang(1,nfang)) ) )

      if ( tau(1) .eq. 0.0) then
         dqtau(1) = 0.0
         dutau(1) = 0.0
         dttau(1) = 0.0
      else
         tauinti = 1. / tau(1)
         dqtau(1) =  tauinti *
     &        ((1. - sat_factor)*dqtauang(1,nfang)
     &             + sat_factor*dqtauang(1,nfang+1))

         dutau(1) =  tauinti *
     &        ((1. - sat_factor)*dutauang(1,nfang)
     &             + sat_factor*dutauang(1,nfang+1))
         dttau(1) =  tauinti *
     &          ((1. - sat_factor)*dttauang(1,nfang)
     &               + sat_factor *dttauang(1,nfang+1))
      endif

!  loop over transmittance layers

      do l = 2,nlev

         n = (l-1)*l/2          ! set index for packed array
         nm1 = (l-2)*(l-1)/2    !  for levels l and l-1
         nnm1 = n - nm1

         tauint = amax1( 0.0, (tauang(l, nfang)
     &        + sat_factor * (tauang(l,nfang+1)-tauang(l,nfang))))
         tau(l) = tau(l-1) * tauint
!
!
!  calculate factors dqtau ( 1/tau d(tau)/dq) and dutau (1/tau d(tau)/du)
```

```
            if (tauint .eq. 0.0 ) then
               dqtau(l) = 0.0
               dutau(l) = 0.0
               do m = 1,l
                  dttau(m+n) = 0.0
               enddo
            else
               tauinti = 1. / tauint
               dqtau(l) =  tauinti *
     &           ((1. - sat_factor)*dqtauang(l,nfang)
     &                + sat_factor *dqtauang(l,nfang+1))
               dutau(l) =  tauinti *
     &           ((1. - sat_factor)*dutauang(l,nfang)
     &                + sat_factor *dutauang(l,nfang+1))

!              do m = 1,l-1
!                 dttau(m+n) =  dttau(m+nm1) + tauinti *
!    &              ((1. - sat_factor)*dttauang(m+n,nfang)
!    &                   + sat_factor *dttauang(m+n,nfang+1))
               do m = n+1,n+l-1
                  dttau(m) =  dttau(m-nnm1) + tauinti *
     &              ((1. - sat_factor)*dttauang(m,nfang)
     &                   + sat_factor *dttauang(m,nfang+1))
               enddo
               dttau(l+n) = tauinti *
     &            ((1. - sat_factor)*dttauang(l+n,nfang)
     &                 + sat_factor *dttauang(l+n,nfang+1))

            endif

         enddo

         if (irchan(chan)) then

            do l = 1,nlev
               sfactor = chscatt(chan) * scat(l)
               tauscat = quikexp( -sfactor * seceff )
               tau(l) = tau(l) * tauscat
            enddo

            if (dosun)  then

! find which angles of rapid algorithm bracket the effective sun angle
!  and calculate factor for interpolation in sec(sun angle)

               anglsun = seceff + secsun
               efcos = 1. / anglsun
               efsun = acos( efcos )

               nfsun = 1
               do k = 2, nangm1
                  if ( efsun .gt. angle(k) ) nfsun = k
               enddo
```

```
            sun_factor = ( anglsun - secang(nfsun) ) /
     &                   (secang(nfsun+1) - secang(nfsun) )

!    calculate product of transmittance in layers along solar path
            tausun = amax1( 0.0, (tauang(1,nfsun)
     &        + sun_factor * (tauang(1,nfsun+1) - tauang(1,nfsun)) ))

            if (tausun .eq. 0.0) then
               dqsun(1) = 0.0
               dusun(1) = 0.0
               dtsun(1) = 0.0
            else
               tauinti = 1. / tausun
               dqsun(1) = tauinti *
     &          ((1.-sun_factor)*dqtauang(1,nfang)
     &              +sun_factor *dqtauang(1,nfang+1))

               dusun(1) = tauinti *
     &          ((1.-sun_factor)*dutauang(1,nfang)
     &              +sun_factor*dutauang(1,nfang+1))

               dtsun(1) = tauinti *
     &          ((1.-sun_factor)*dttauang(1,nfang)
     &              + sun_factor*dttauang(1,nfang+1))

            endif

            do l = 2,nlev

               n = (l-1)*l/2          ! set index for packed array

               tauint = amax1( 0.0, (tauang(l, nfsun)
     &           + sun_factor * (tauang(l,nfsun+1)-tauang(l,nfsun))))
               tausun = tausun * tauint
               if (tauint .eq. 0.0) then
                  dqsun(l) = 0.0
                  dusun(l) = 0.0
                  dtsun(l) = 0.0
               else
                  tauinti = 1. / tauint
                  dqsun(l) = tauinti *
     &             ((1. - sun_factor)*dqtauang(l,nfang)
     &                  + sun_factor *dqtauang(l,nfang+1))
                  dusun(l) =  tauinti *
     &             ((1. - sun_factor)*dutauang(l,nfang)
     &                  + sun_factor *dutauang(l,nfang+1))

                  do m = 1,l-1
                     dtsun(m) =  dtsun(m) + tauinti *
     &                 ((1. - sat_factor)*dttauang(m+n,nfang)
     &                      + sat_factor *dttauang(m+n,nfang+1))
                  enddo
                  dtsun(l) = tauinti *
     &                ((1. - sat_factor)*dttauang(l+n,nfang)
```

```
&                    + sat_factor *dttauang(l+n,nfang+1))

          endif
        enddo
        sfactor = chscatt(chan) * scat(nlev)
        tauscat = quikexp( -sfactor * anglsun )
        tausun = tausun * tauscat              !!! check this...
      endif

 endif
 return
 end
```

tovs_tauprodK.f – Take products of transmittance derivatives

```
!*************************************************************************
!!TOVS_TAUPRODK
!*************************************************************************
!
!!ROUTINE:  tovs_tauprodK
!
!!DESCRIPTION:  Calculate product of fixed, h2o and ozone contributions
!                 to transmittances and derivatives of tau product
!                     ** with minor optimization
!
! CALLED FROM:   tovs_tauK
!
! SYSTEM ROUTINES USED:  none
!
! SUBROUTINES CALLED: none
!
!!INPUT PARAMETERS:
!  nlev     - integer surface level (also number of levels to use)
!  taufix   - real array (of size maxlev by maxangs) of effective
!                mean layer transmittance for fixed gases
!  tauh2o   - real array (of size maxlev by maxangs) of effective
!                mean layer transmittance for moisture
!  tauozo   - real array (of size maxlev by maxangs) of effective
!                mean layer transmittance for ozone
!  dqtauh2o - real array (of size maxlev by maxangs) of derivative
!                of tauh2o with respect to H2O specific humidity
!  dttauh2o - real array (of size maxlev by maxangs) of derivative
!                of tauh2o with respect to temperature
!  dutauozo - real array (of size maxlev by maxangs) of derivative
!                of tauozo with respect to ozone
!  dttauozo - real array (of size maxlev by maxangs) of derivative
!                of tauozo with respect to temperature
!  dttaufix - real array (of size maxpack by maxangs) of
!                derivative of taufix with respect to temperature
! packed matrix with  first dimension: temperature layer    (see below)
!                    second dimension: transmittance layer
!
!!OUTPUT PARAMETERS:
!  tauang   - real matrix (of size maxlev by maxangs) of mean layer
!               transmittance calculated for different zenith angles
!  dqtauang - real array (of size maxlev by maxangs) of derivative of
!               tauang wrt. H2O specific humidity
!  dutauang - real array (of size maxlev by maxangs) of derivative of
!               tauang wrt. ozone
!  dttauang - real array (of size maxpack by maxangs) of
!               derivative of tauang wrt. mean layer temperature
! packed matrix:      first dimension: temperature layer    (see below)
!                    second dimension: transmittance layer
!
!!REVISION HISTORY:
!  12jan96  Meta S.  original routine
```

```
!  17jan96  Meta S.  sum of temperature deriv over layers moved to
!                    tovs_tauangK, *after* interpolation to zenith angle
!                    removed TAUCFB, TAUCFC - added dttaufix input
!  16feb96  Meta S.  minor optimization:  Switched dimensions of dttaufix,
!                        dttauang
!  29mar96  Meta S.  another minor optimization: change dttaufix, dttauang
!                    into packed array ((maxlev*(maxlev+1))/2,maxangs)
!
! Matrix Packing                    (based on SPPTRF manual page)
!  The j-th column of A is stored in the array AP as follows:
!  AP(i + (j-1)*j/2) = A(i,j) for 1<=i<=j
!
!     Two-dimensional storage of the matrix A:
!
!       a11 a12 a13 a14    1st dimension: affecting temperature layer
!        0  a22 a23 a24    2nd dimension: affected transmittance layer
!        0   0  a33 a34
!        0   0   0  a44    a12: influence of T(1) on \hat(\tau)(2)
!        ^^^^^^^^^
!        | zeros because temperatures in a given layer don't affect
!             transmittance layers above that layer
!
!    Packed storage of the upper triangle of A:
!
!    AP = [ a11, a12, a22, a13, a23, a33, a14, a24, a34, a44 ]
!
!*****************************************************************************
      subroutine tovs_tauprodK( nlev, taufix, tauh2o, tauozo,
     &           dqtauh2o, dttauh2o, dutauozo, dttauozo, dttaufix,
     &           tauang, dqtauang, dutauang, dttauang )

      implicit none

      include "tovsparam.h"

      integer maxpack
      parameter (maxpack = (maxlev*(maxlev+1))/2 )

!  variables passed into routine
      integer nlev
      real taufix(maxlev,maxangs)
      real tauh2o(maxlev,maxangs)
      real tauozo(maxlev,maxangs)
      real dqtauh2o(maxlev,maxangs)
      real dttauh2o(maxlev,maxangs)
      real dutauozo(maxlev,maxangs)
      real dttauozo(maxlev,maxangs)
      real dttaufix(maxpack,maxangs)

!  variables passed out of routine
      real tauang(maxlev,maxangs)
      real dqtauang(maxlev,maxangs)
      real dutauang(maxlev,maxangs)
      real dttauang(maxpack,maxangs)
```

```
!  local variables
      integer k,l,m,n

      do k = 1,maxangs
         do l = 1,nlev

            n = (l-1)*l/2      ! set index for packed array

            tauang(l,k) = taufix(l,k)*tauh2o(l,k)*tauozo(l,k)
            dqtauang(l,k) = tauang(l,k) * dqtauh2o(l,k)
            dutauang(l,k) = tauang(l,k) * dutauozo(l,k)

!           do m = 1,l
!              dttauang(m+n,k) = tauang(l,k) * dttaufix(m+n,k)
            do m = n+1,n+l
               dttauang(m,k) = tauang(l,k) * dttaufix(m,k)
            enddo
            dttauang(l+n,k) =  dttauang(l+n,k) +
     &                tauang(l,k) * (dttauh2o(l,k) + dttauozo(l,k))

         enddo
         do l = nlev+1,maxlev

            n = (l-1)*l/2      ! set index for packed array

            tauang(l,k) = 0.
            dqtauang(l,k) = 0.
            dutauang(l,k) = 0.
            do m = n+1,n+l
               dttauang(m,k) = 0.
            enddo
         enddo
      enddo
      return
      end
```