Community Radiative Transfer Model (CRTM) Overview and Tutorial

Paul Van Delst

NCEP/EMC & JCSDA

Overview

- Some background
  - How does radiation interact with matter
  - Development of radiative transfer
- Additional CRTM components (not all, just the latest developments)
  - Non-LTE, Zeeman, emissivity models, etc
- CRTM Interface
  - Forward, Tangent-linear, Adjoint, and K-matrix models.
- Calling the CRTM
  - Step-by-step for the K-matrix model.
- Default available object methods
  - Not really OO yet, but we're heading that way.

Introduction

- CRTM is a library containing functions to compute satellite sensor radiances, and Jacobians.
  - Fortran95/2003.
  - Heavy use of language features made to ease code maintenance and reuse of common components.
- Geared towards data assimilation (i.e. GSI at NCEP) but used in other contexts.
- There are forward, tangent-linear, adjoint, and K-matrix functions.
  - Forward model is “built in” to the other functions so they are all stand-alone.

BACKGROUND

Electromagnetic spectrum

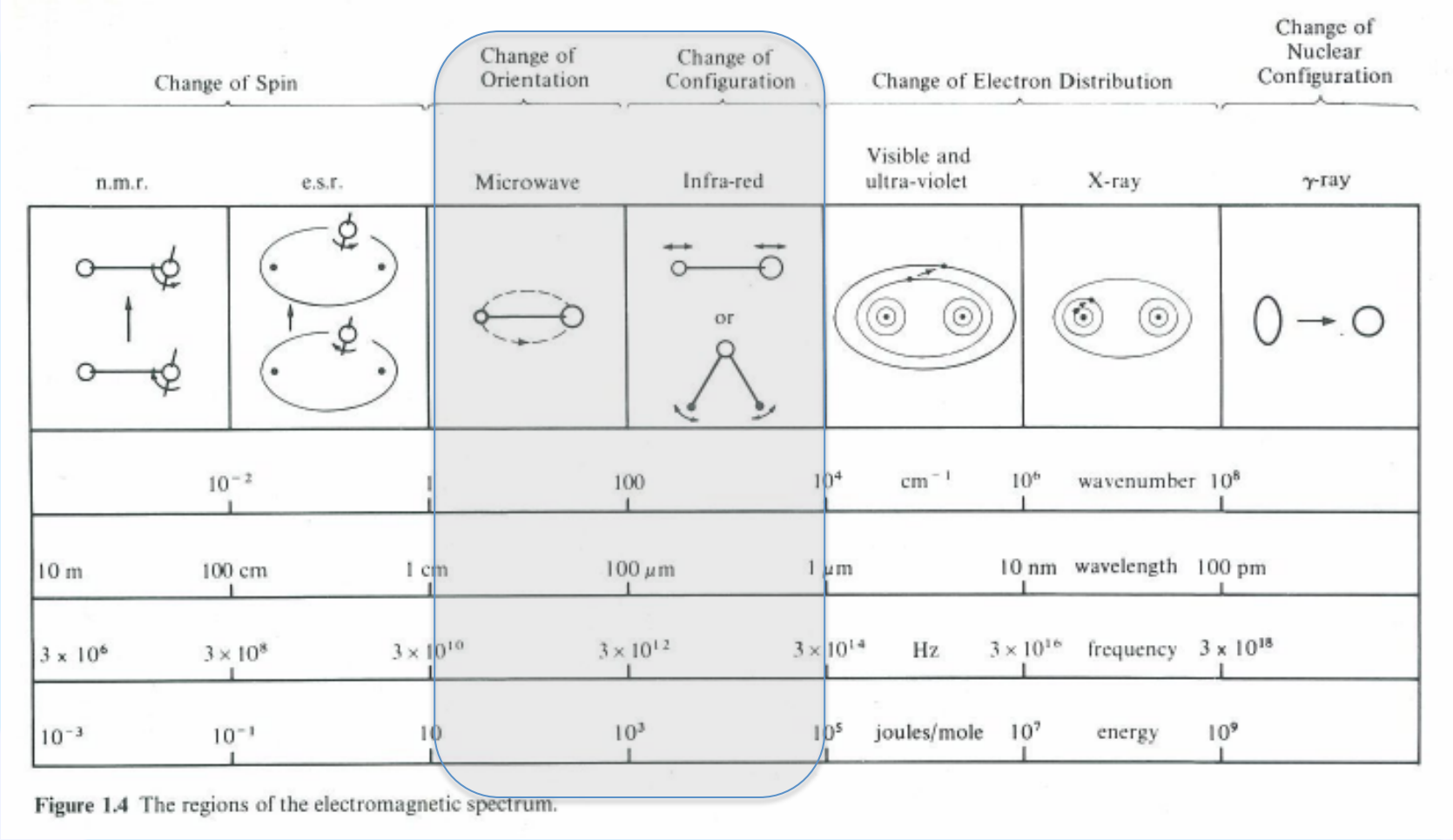


Figure 1.4 The regions of the electromagnetic spectrum.

From: Banwell, C.N., 1983, "Fundamentals of Molecular Spectroscopy", McGraw-Hill, London.

Microwave – rotation of molecules

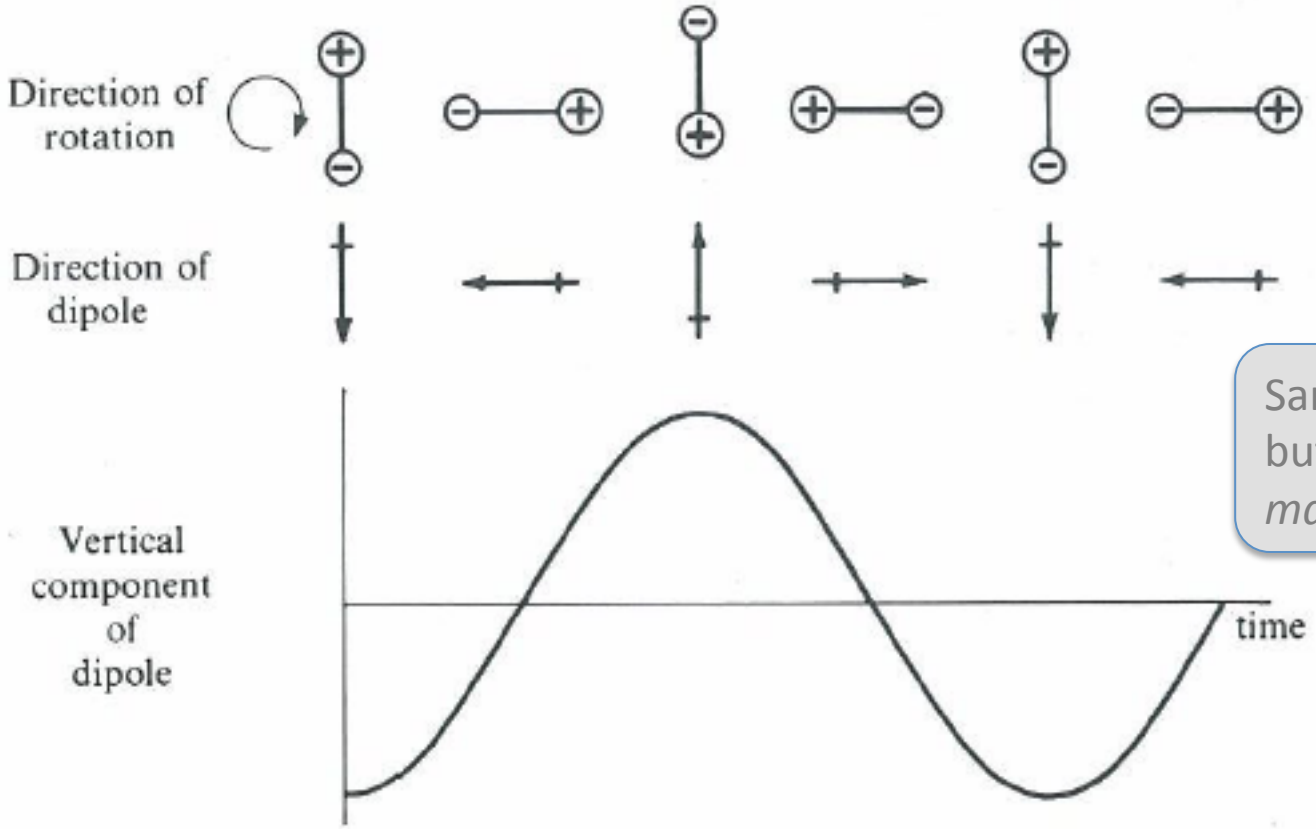


Figure 1.5 The rotation of a diatomic molecule, HCl, showing the fluctuation in the dipole moment measured in a particular direction.

Same process for oxygen, but due to its permanent *magnetic* dipole.

From: Banwell, C.N., 1983, "Fundamentals of Molecular Spectroscopy", McGraw-Hill, London.

Infrared – vibration of molecules. Symmetric stretching

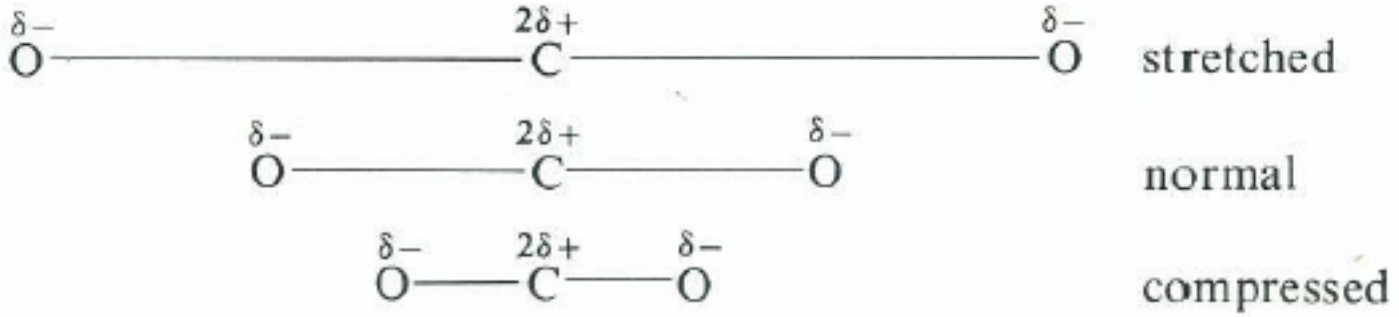


Figure 1.6 The symmetric stretching vibration of the carbon dioxide molecule.

- No change in dipole moment.
- Vibration is “infrared inactive”

From: Banwell, C.N., 1983, “Fundamentals of Molecular Spectroscopy”, McGraw-Hill, London.

Infrared – vibration of molecules. Asymmetric stretching

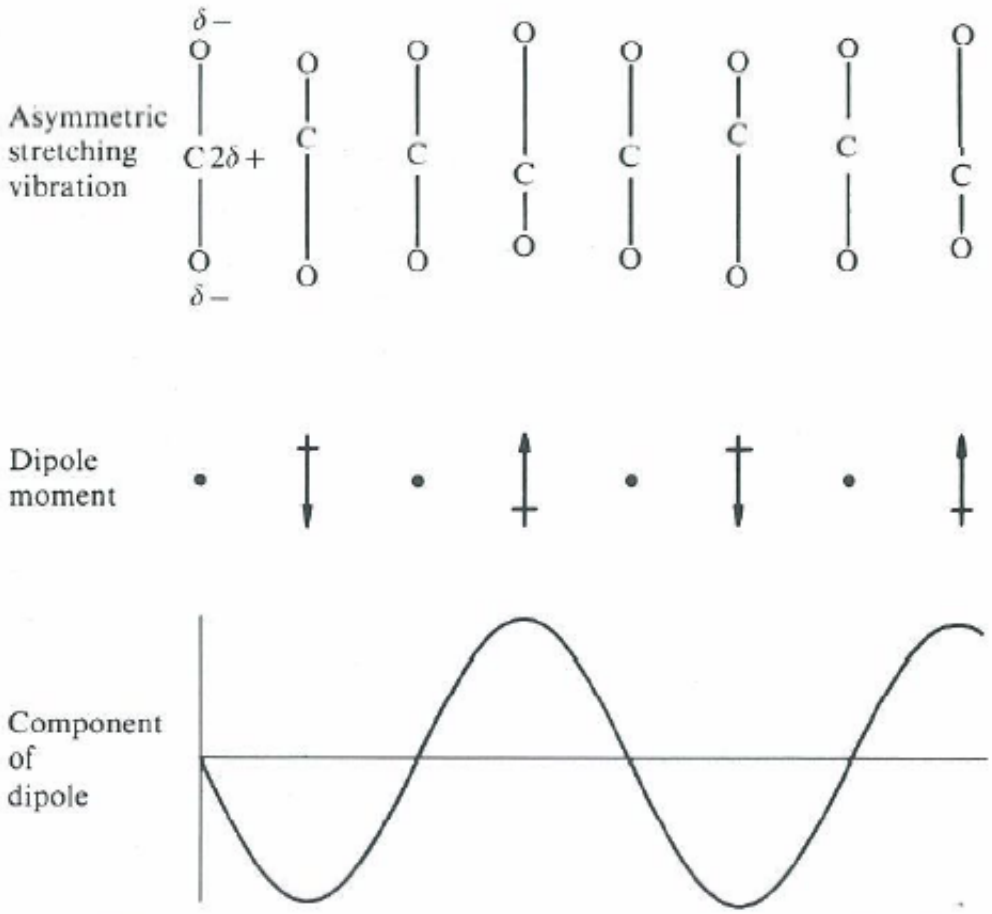


Figure 1.7 The asymmetric stretching vibration of the carbon dioxide molecule showing the fluctuation in the dipole moment.

From: Banwell, C.N., 1983, "Fundamentals of Molecular Spectroscopy", McGraw-Hill, London.

Infrared – vibration of molecules. Bending motion

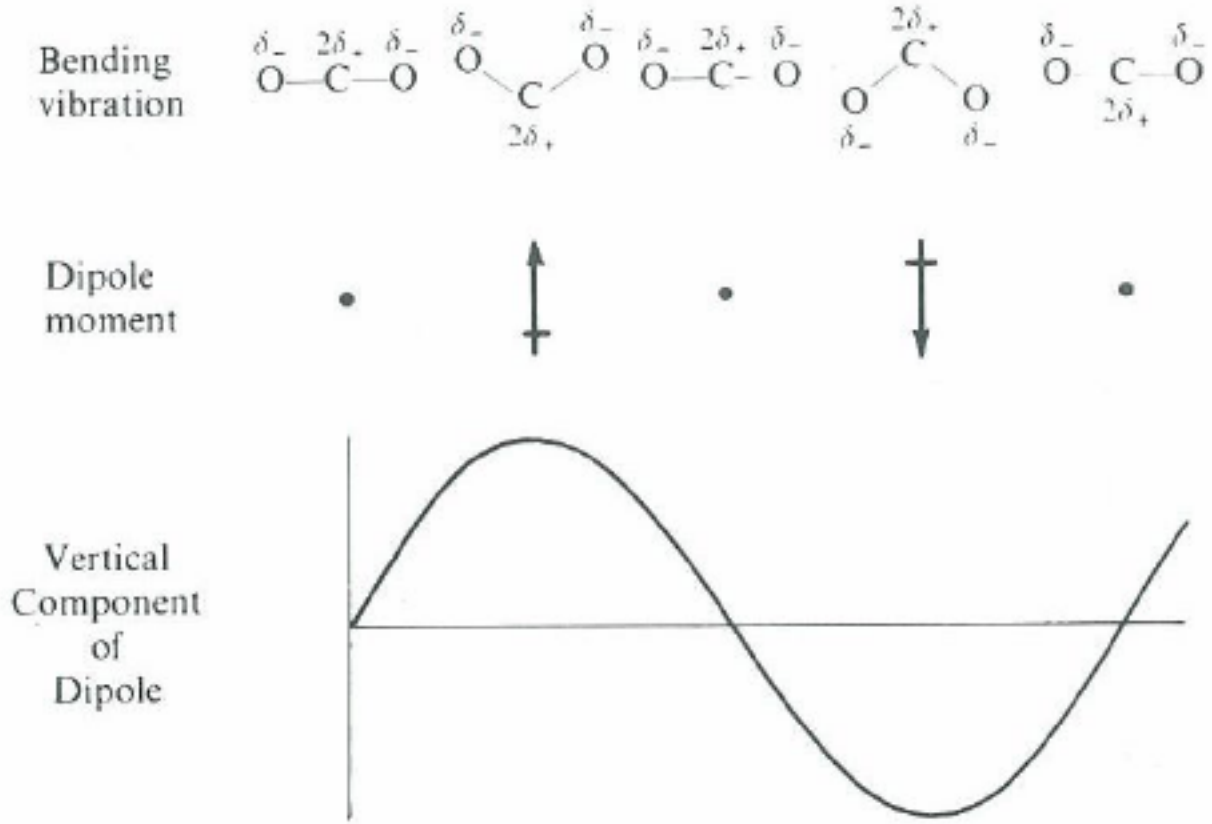


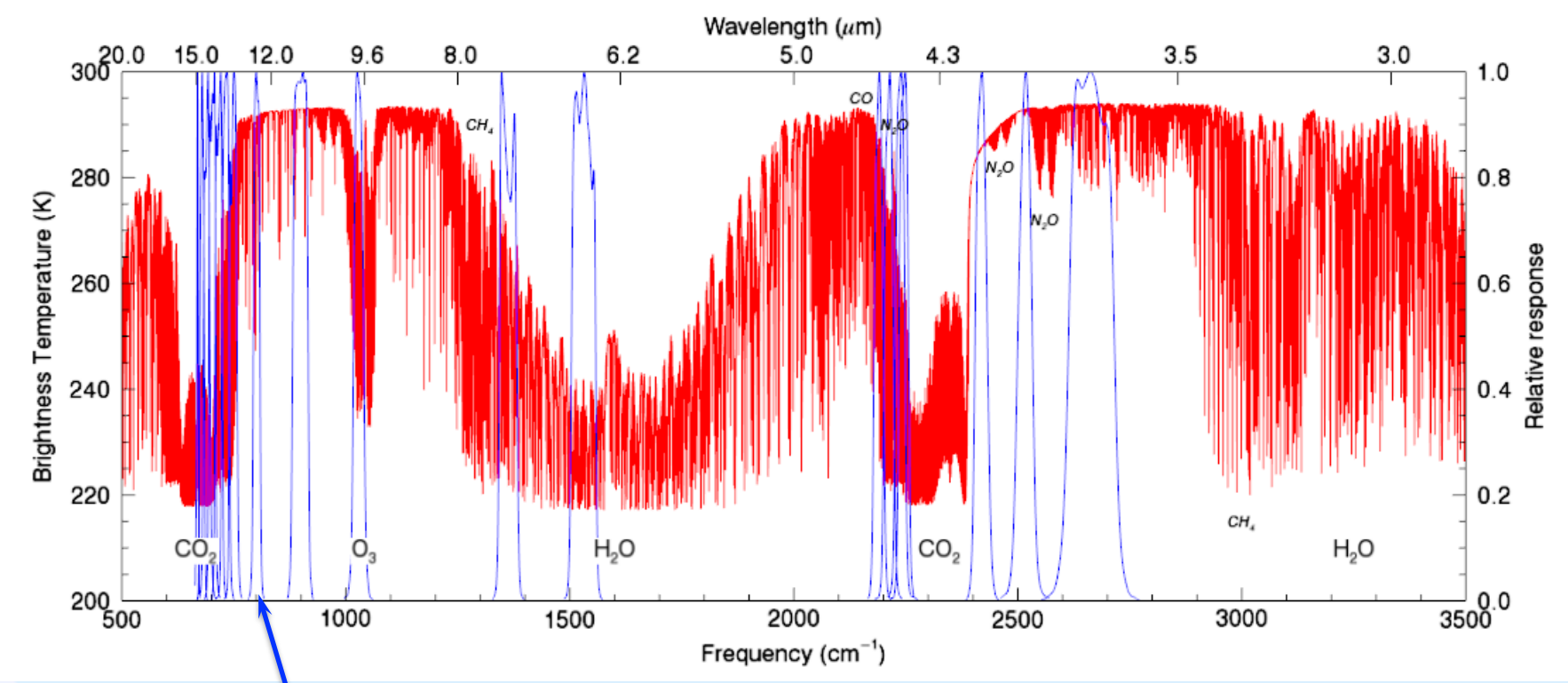
Figure 1.8 The bending motion of the carbon dioxide molecule and its associated dipole fluctuation.

From: Banwell, C.N., 1983, "Fundamentals of Molecular Spectroscopy", McGraw-Hill, London.

Mechanism behind the interaction of radiation and matter

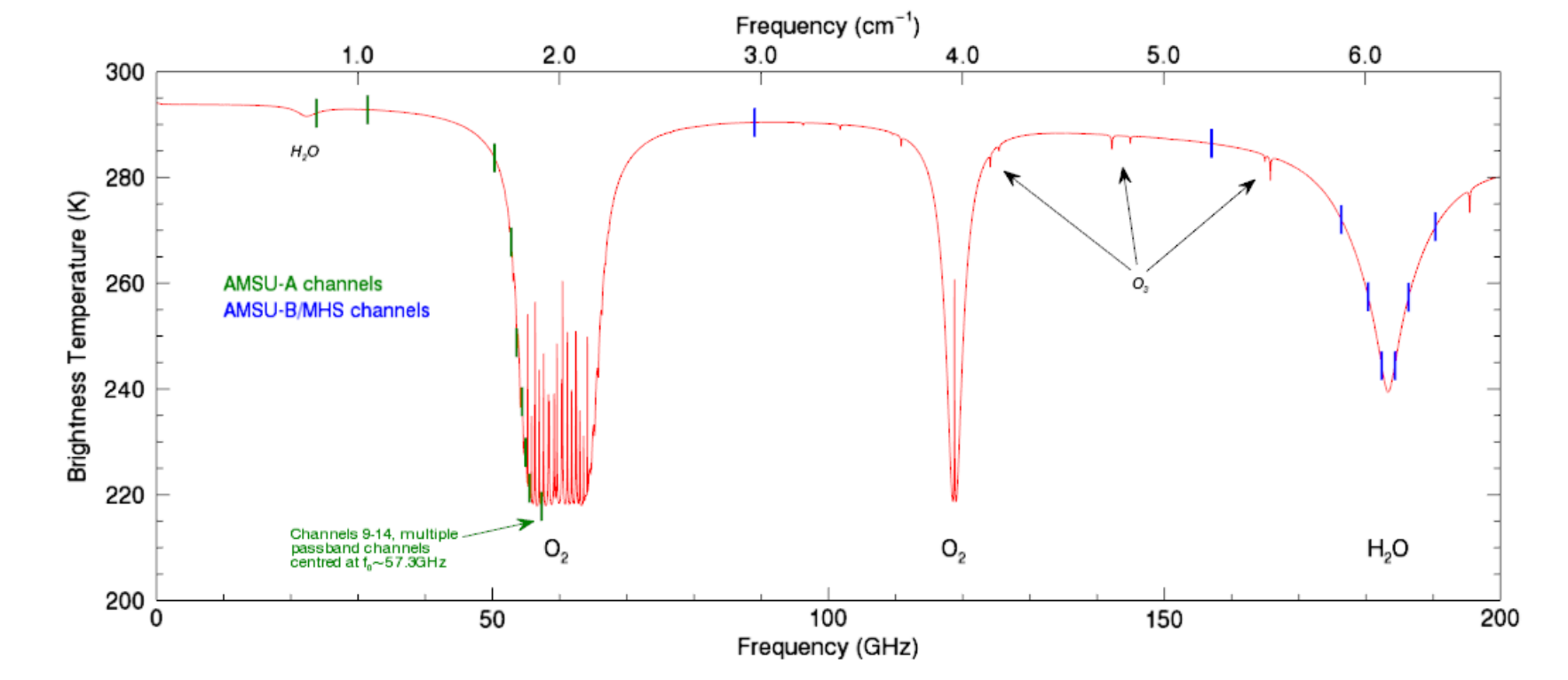
- The change in a molecule's dipole moment (electric or magnetic) as it rotates or vibrates is what allows it to interact with radiation.
- The energies at which molecules interact with radiation are determined by properties such as their shape (e.g. rotational inertia) and bond strength (e.g. elasticity of stretching and bending motions).
- Additionally, the rotational and vibrational energies of molecules are quantised. Thus, molecules interact with radiation at well defined frequencies.
- So what?

Infrared spectrum



HIRS spectral response functions

Microwave spectrum



Information in the measured spectra

- The measured spectra give us information about the state of the atmosphere and surface.
  - How much of molecule X?
  - Temperature at pressure Y?
  - Surface temperature?
- Extracting this information from radiance measurements is a classical inversion problem.
- But first we have to be able to simulate the transfer of radiation through the atmosphere.

Some definitions

- Absorption, extinction: radiance decreases.
- Emission: radiance increases.
- Kirchoff's Law: Under the conditions of local thermodynamic equilibrium (LTE), the absorptivity, α, of a medium is equal to its emissivity, ε.
- Lambert's (or Bouguet's) law: The absorption process is linear, independent of the radiant intensity and amount of matter, provided the physical state is held constant.

Interaction of radiation and matter

- From Lambert's law, the change of radiance, I, along a path ds due to extinction is proportional to the amount of matter in the path:

$$dI_{abs} = -\beta_a I \; ds$$

- Similarly for emission

$$dI_{emit} = \beta_a J \; ds$$

  - β_a = volume absorption coefficient
  - J = source function.

$$\beta_a = k_a \rho$$

k_a = mass absorption coefficient
ρ = density

Is the Planck function, $B(T)$, under LTE in a non-scattering medium

- Total change in radiance due to interaction of radiation and matter is then,

$$dI = dI_{abs} + dI_{emit} = k_a \rho \left(B(T) - I \right) ds$$

$$\therefore \frac{dI}{k_a \rho \; ds} = B(T) - I$$

With apologies to Max Planck, k_a encompasses all the interesting physics

- This is Schwarzchild's equation

Petty, G.W., 2004, "A First Course in Atmospheric Radiation", Sundog Publishing, Madison WI.
Goody, R.M. and Y.L. Yung, 1989, "Atmospheric Radiation: Theoretical Basis", 2nd Ed, Oxford University Press, New York.

Non-scattering solution

- This is where we define the optical thickness between two points along the path:

$$\tau(s \rightarrow s_{toa}) = \int_{s}^{s_{toa}} k\rho\, ds' \quad \Rightarrow \quad d\tau(s \rightarrow s_{toa}) = -k\rho ds$$

- Using this in Schwarzchild's equation and integrating the optical thickness thickness from 0 (s_{toa}) to it's value at the surface (s_{sfc}) :

$$I(0) = I(\tau_{sfc})e^{-\tau_{sfc}} + \int_{0}^{\tau_{sfc}} B(\tau)e^{-\tau}\, d\tau$$

- Manipulation of this expression leads to the more familiar (well, to me) non-scattering radiative transfer equation (RTE),

$$I(p_{toa}) = B[T_{sfc}]\varepsilon_{sfc}t_{sfc} + \int_{p_{sfc}}^{p_{toa}} B[T(p)]\frac{dt(p)}{dp}dp$$

  - where $t(p) = -\frac{\sec(\theta)}{g}e^{-\tau(p \rightarrow p_{toa})}$

Weighting function

Petty, G.W., 2004, "A First Course in Atmospheric Radiation", Sundog Publishing, Madison WI.

Transmittance calculation

- The gaseous absorption algorithm is the core of the CRTM (or any model).
- Regular regression model is used where frequency dependent regression coefficients, $c_{i,\nu}$, are used with atmospheric state predictors, X_i, to compute the channel absorption coefficients,

$$k_{a,\nu} = c_{0,\nu} + \sum_{i=1}^{N} c_{i,\nu} X_i$$

- Two algorithms for gaseous absorption
  - ODAS (**O**ptical **D**epth in **A**bsorber **S**pace).
    - A "compact" version of the OPTRAN model.
    - H_2O, O_3 absorption only.
  - ODPS (**O**ptical **D**epth in **P**ressure **S**pace).
    - Optical depths computed on a fixed pressure grid.
    - Better fitting statistics.
    - More trace gases: CO_2, CH_4, N_2O.
    - Enables incorporation of Zeeman model (requires fixed pressure grid).

RTE with scattering

- With scattering the change in intensity, dI, becomes:

$$dI = dI_{ext} + dI_{emit} + dI_{scat}$$

  - where $dI_{ext} = -\beta_e I \, ds$
  - and $\beta_e = \beta_a + \beta_s$

- The dI_{scat} term takes into account radiation from any direction being scattered into the sensor field-of-view (FOV),

$$dI_{scat} = \frac{\beta_s}{4\pi} \int_{4\pi} p(\hat{\Omega}', \hat{\Omega}) I(\hat{\Omega}) d\omega' \, ds$$

Scattering phase function

- The CRTM uses a look-up table to obtain scattering optical properties (e.g. extinction coefficient, single scatter albedo, etc) and reconstruct phase functions for the necessary number of streams (angles over which the zenith integration is done)

Petty, G.W., 2004, "A First Course in Atmospheric Radiation", Sundog Publishing, Madison WI.

OTHER CRTM COMPONENTS

Non-LTE model (1)

- At high altitudes, vibrational transitions depart from LTE.
- Planck function cannot be used as a source function.

Populations of NLTE and LTE high states

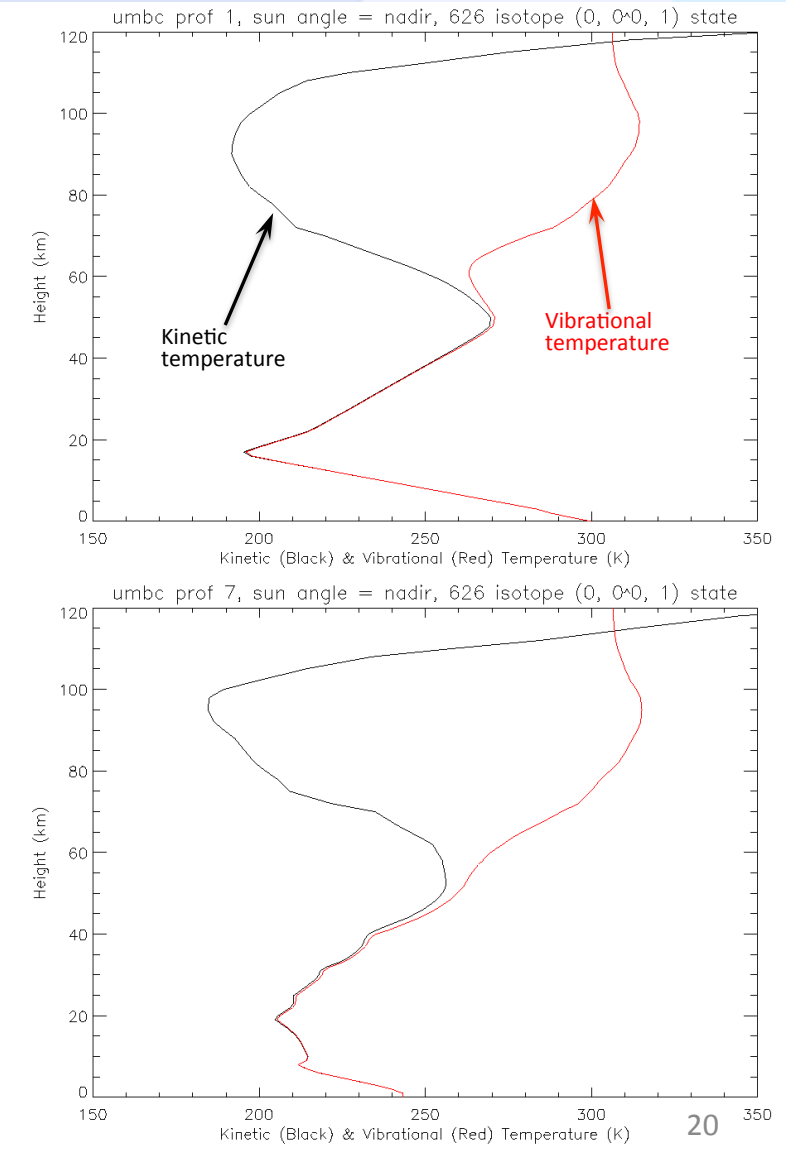
$$J_{\nu,i} = B_\nu \frac{n_{2,i}}{\bar{n}_{2,i}} \frac{\bar{\psi}_{\nu,i}}{\psi_{\nu,i}}$$

NLTE and LTE absorption coefficients

$$\frac{\bar{\psi}_{\nu,i}}{\psi_{\nu,i}} = r_{1,i} \frac{1-\Gamma_i r_{2,i}}{1-\Gamma_i}, \text{ where } \Gamma_i = \exp\left(-\frac{h\nu_0}{kT}\right)$$

$$r_{\alpha,i} = \exp\left[-\frac{E_\alpha}{k}\left(\frac{1}{T_\alpha} - \frac{1}{T}\right)\right], \text{ for } \alpha = 1 \text{ or } 2$$

Vibrational temperature



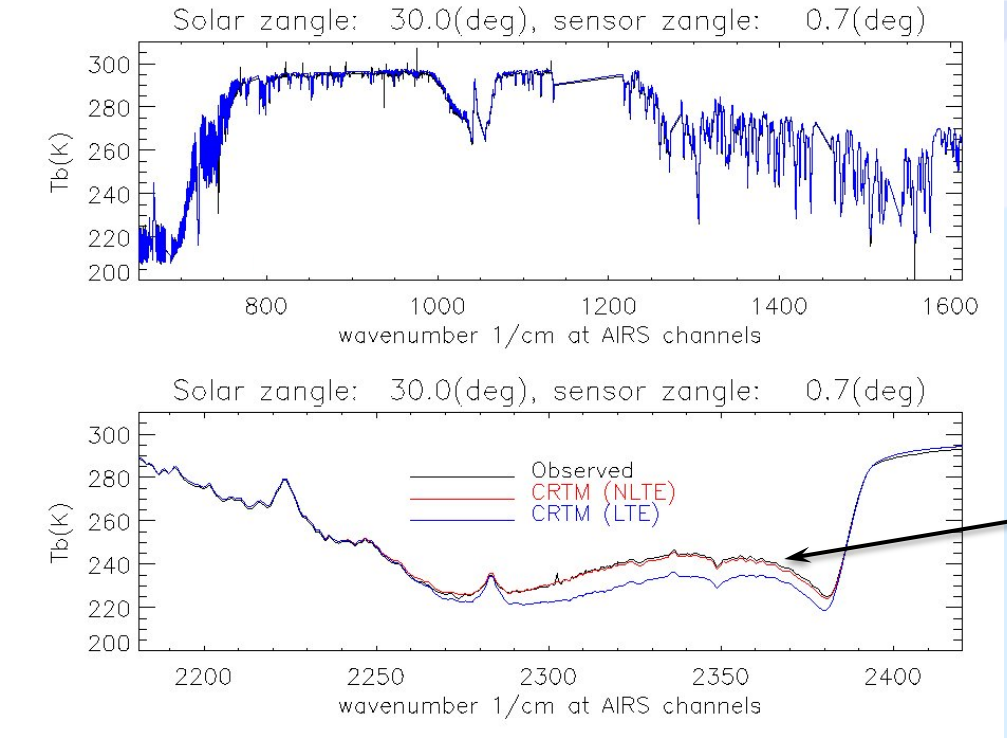
Non-LTE model (2)

- Simple regression model predicts the correction to LTE radiances,

$$R_\upsilon^{NLTE} = R_\upsilon^{LTE} + \Delta R_\upsilon^{NLTE}$$

$$\Delta R_\upsilon^{NLTE} = c_{0,\upsilon} + \sum_{i=1}^{4} c_{i,\upsilon} X_i$$

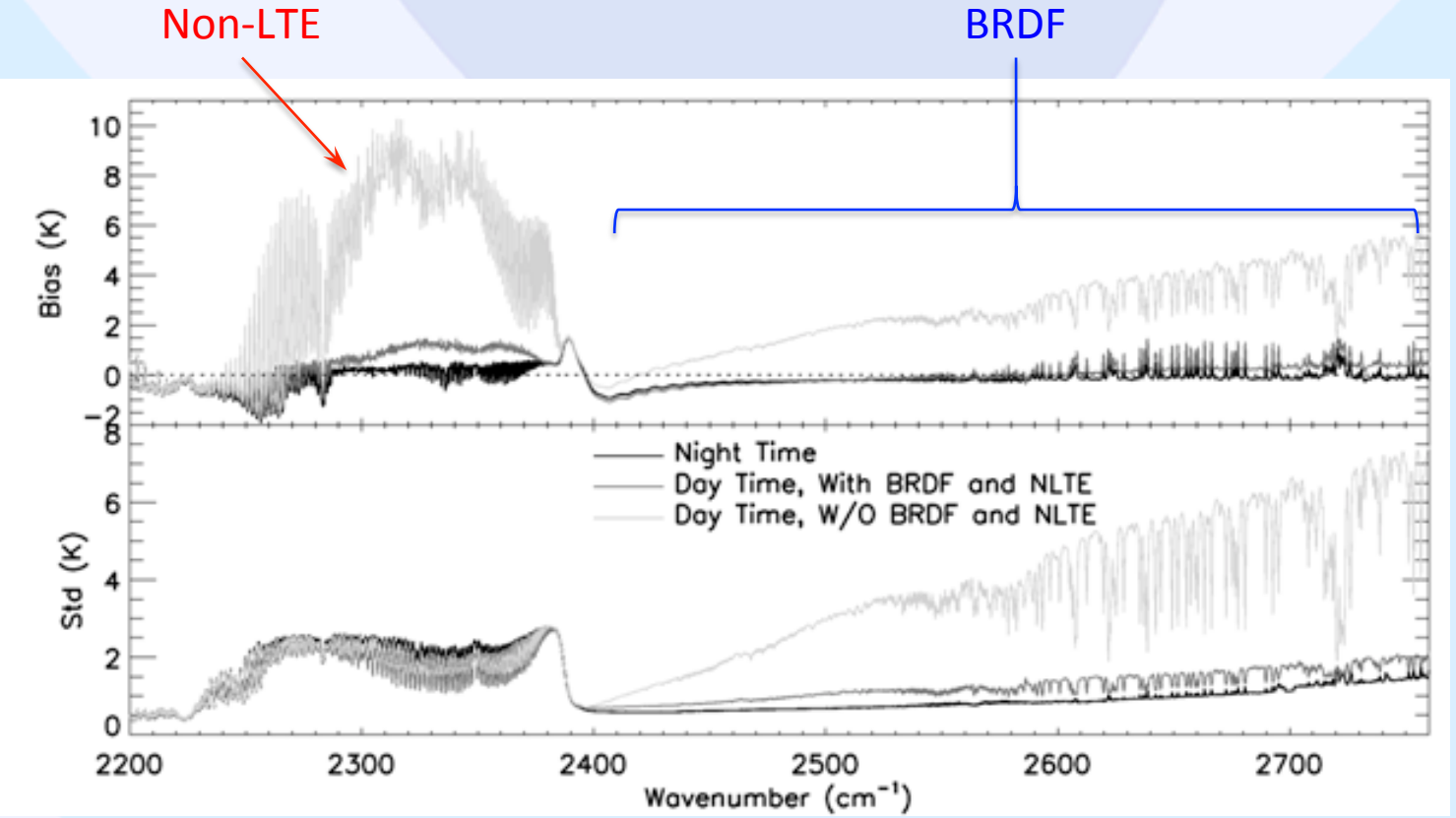
Predictors	Description
X_1	Solar zenith angle
X_2	Mean temperature from 0.005-0.2hPa
X_3	Mean temperature from 0.2-52hPa
X_4	Sensor zenith angle



Remaining differences between observations and NLTE simulations in the shortwave traced to certain CO_2 isotopologues not being present in the spectroscopic database.

Non-LTE model (3)

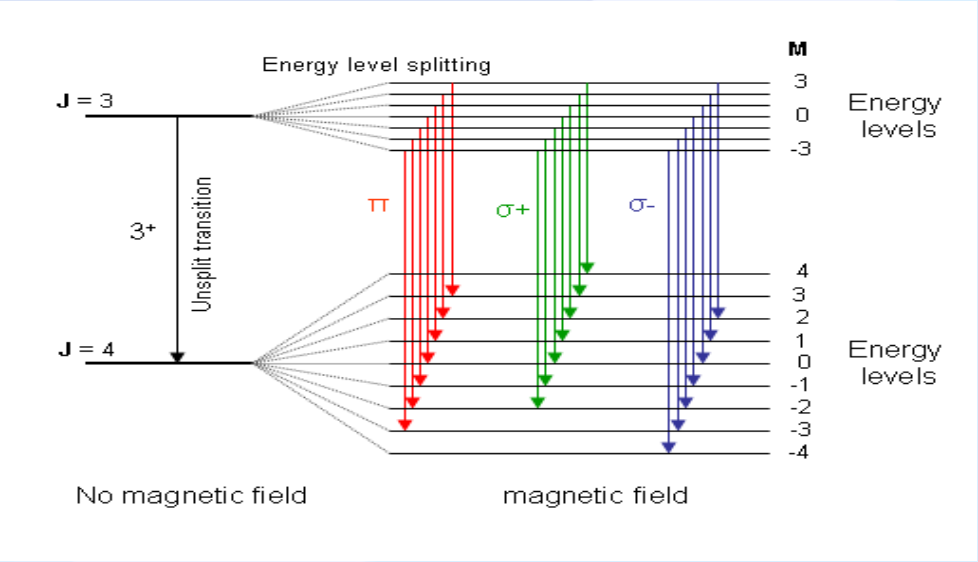
- Comparison of observations and CRTM calculations for IASI shortwave channels, both day and night.



Zeeman model (1)

Energy level splitting:

In the presence of an external magnetic field, each energy level associated with the total angular momentum quantum number J is split into 2J+1 levels corresponding to the azimuthal quantum number M = -J, …, 0, …,J



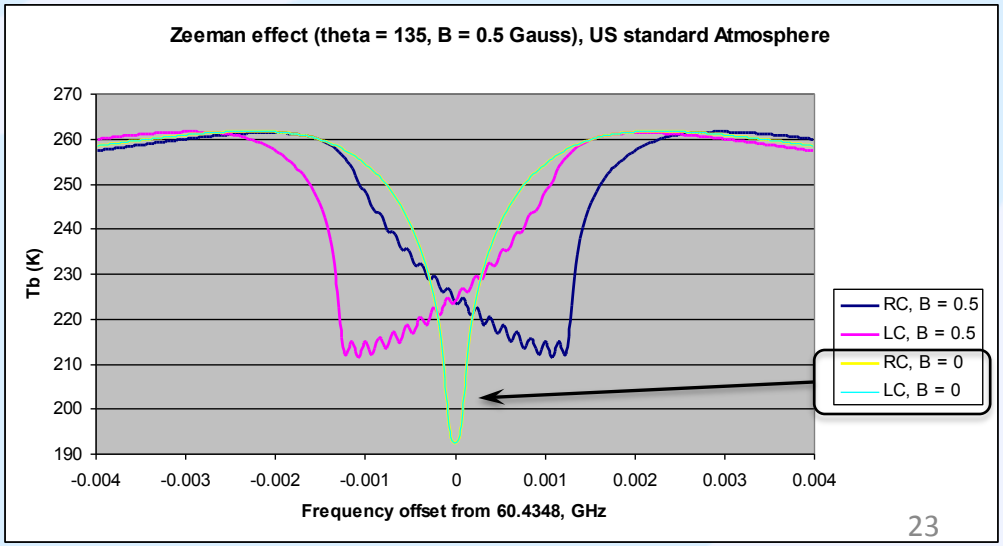
Transition lines (Zeeman components) :

The selection rules permit transitions with ΔJ = ±1 and ΔM = 0, ±1. For a change in J (e.g. J=3 to J=4, represented by 3^{+}), transitions with

ΔM = 0 are called π components,
ΔM = 1 are called σ+ components and
ΔM = -1 are called σ- components.

Polarization:

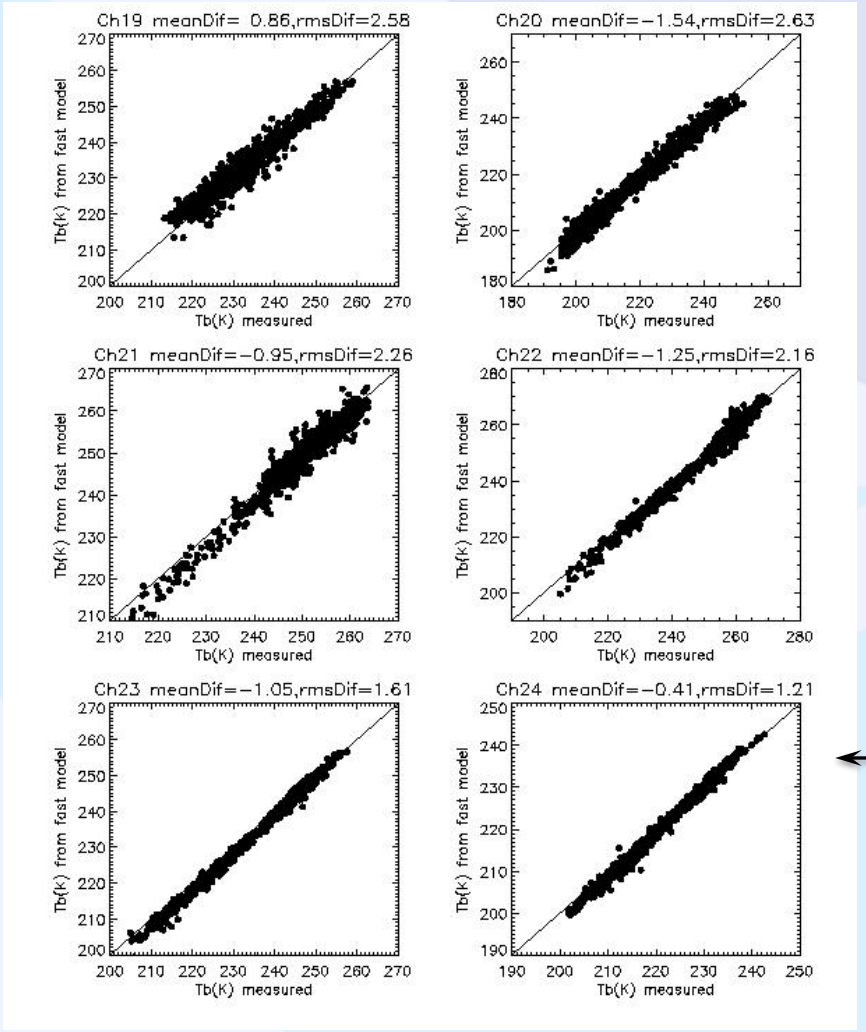
The three groups of Zeeman components also exhibit polarization effects with different characteristics. Radiation from these components received by a circularly polarized radiometer such as the SSMIS upper-air channels is a function of the magnetic field strength |**B**|, the angle θ_B between **B** and the wave propagation direction **k** as well as the state of atmosphere, not dependent on the azimuthal angle of **k** relative to **B**.



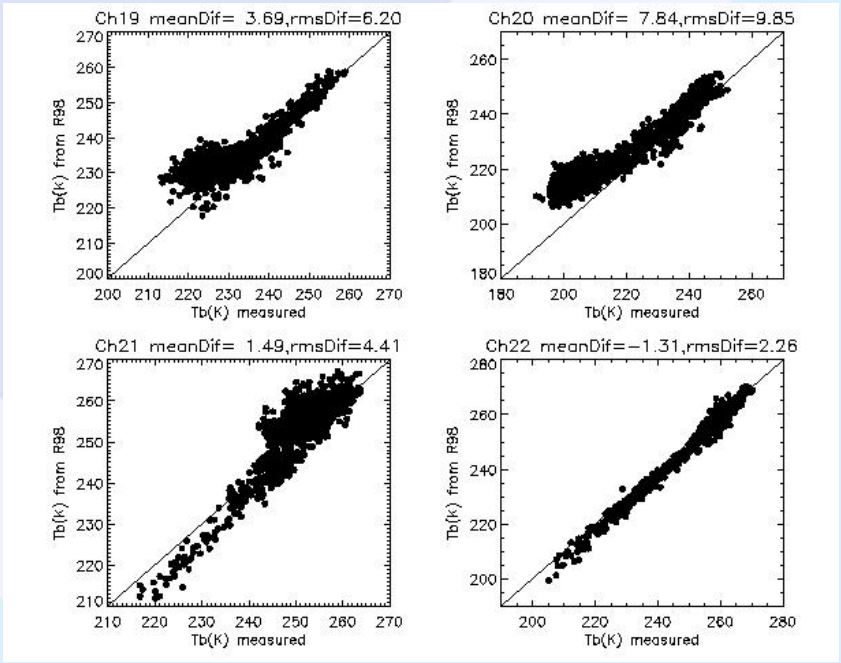
Zeeman model (2)

Comparison between SSMIS observations and CRTM calculations

Including Zeeman effect



Without including Zeeman effect

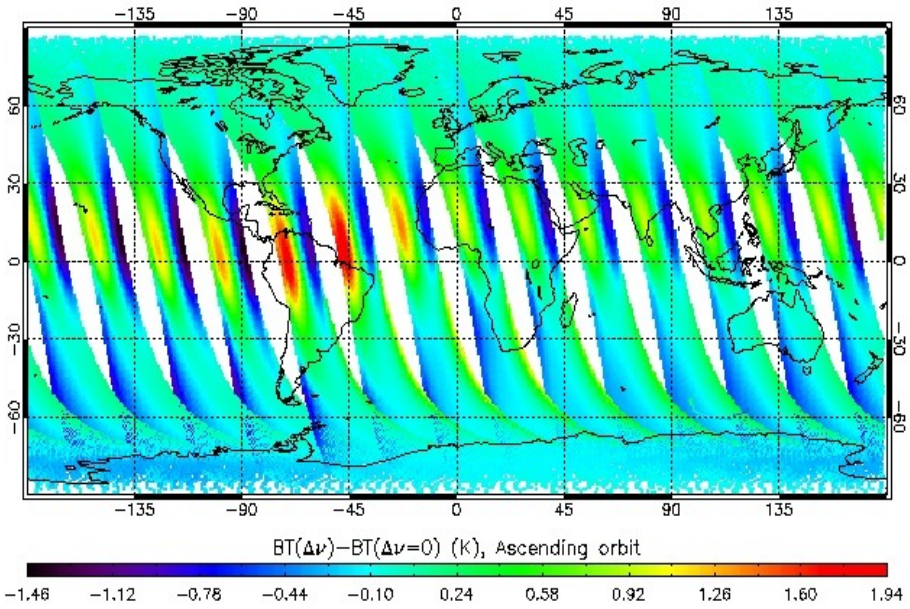


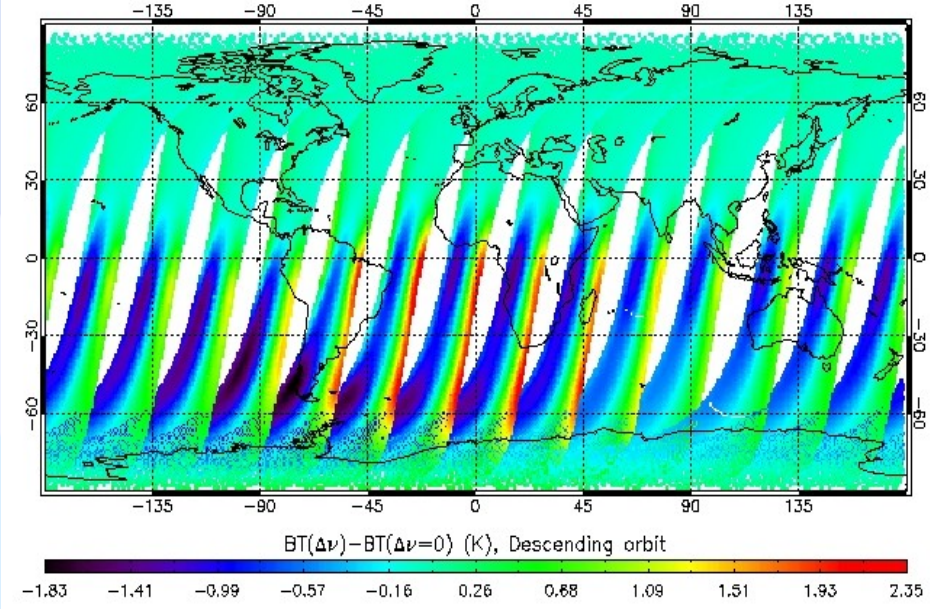
Channels 23 and 24 are not affected by Zeeman-splitting.

Colocated temperature profiles for model inputs are retrievals from the SABER experiment. 1097 samples.

Zeeman model (3)

- Frequency shift of radiation spectrum due to Earth's rotation also has to be taken into account.
- Doppler shift in frequencies can be as much as 80kHz in some regions and scan pixels.
- Impact of Doppler shift has a strong dependence on the angle between the Earth's magnetic field and the wave propagation direction, θ_B. Can reach 2K when θ_B = 0° or 180°, but becomes very small when θ_B=90°.

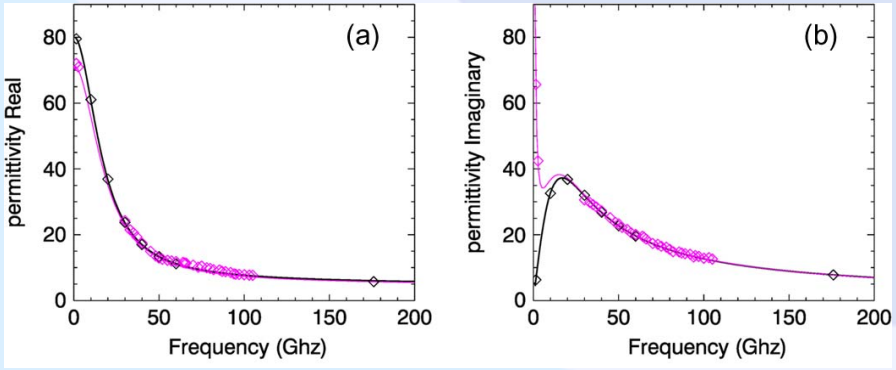




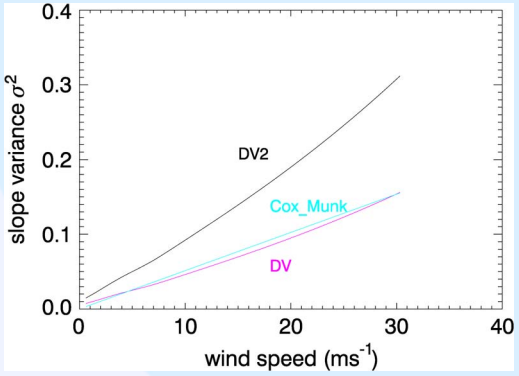
- Simulated brightness temperature differences for SSMIS channel 20 with and without the inclusion of the Doppler shift effect for observations on January 1, 2006. Temperature profiles obtained from the CIRA-88 model.

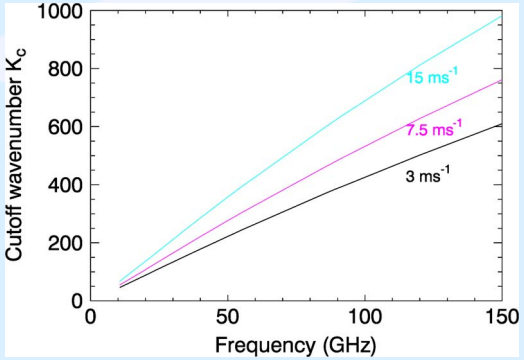
Microwave sea surface emissivity model (1)

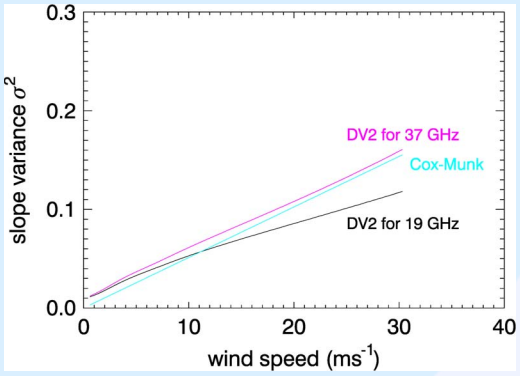
- Updated microwave sea surface emissivity model (FASTEM5).
- Revised double-Debye permittivity model with variable salinity term.



- Factor of two slope variance adjustment to the Durden and Vesecky[1] spectrum model, along with a variable cutoff wavenumber separating the small and large roughness.



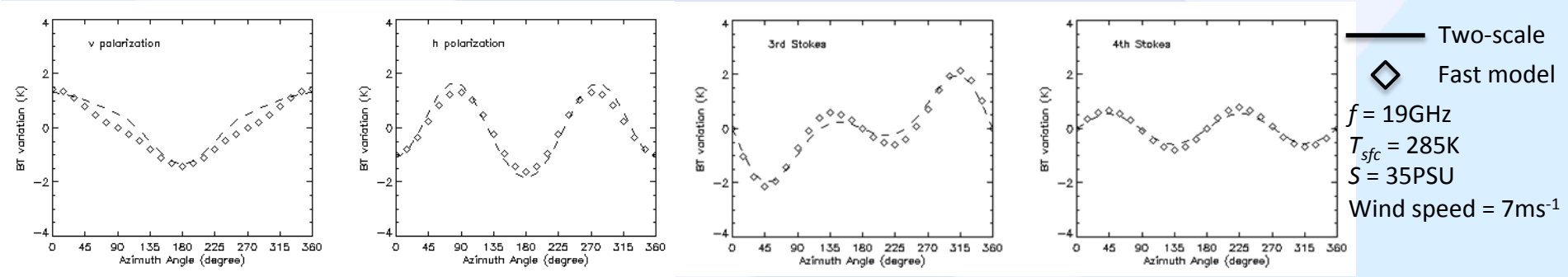




[1]Durden, S.P. and J.F. Vesecky (1985), *IEEE J. Ocean Eng.*, **OE-10**(4), pp445-451.

Microwave sea surface emissivity model (2)

- Azimuthal emissivity dependence using St. Germain and Poe[1] fast model.



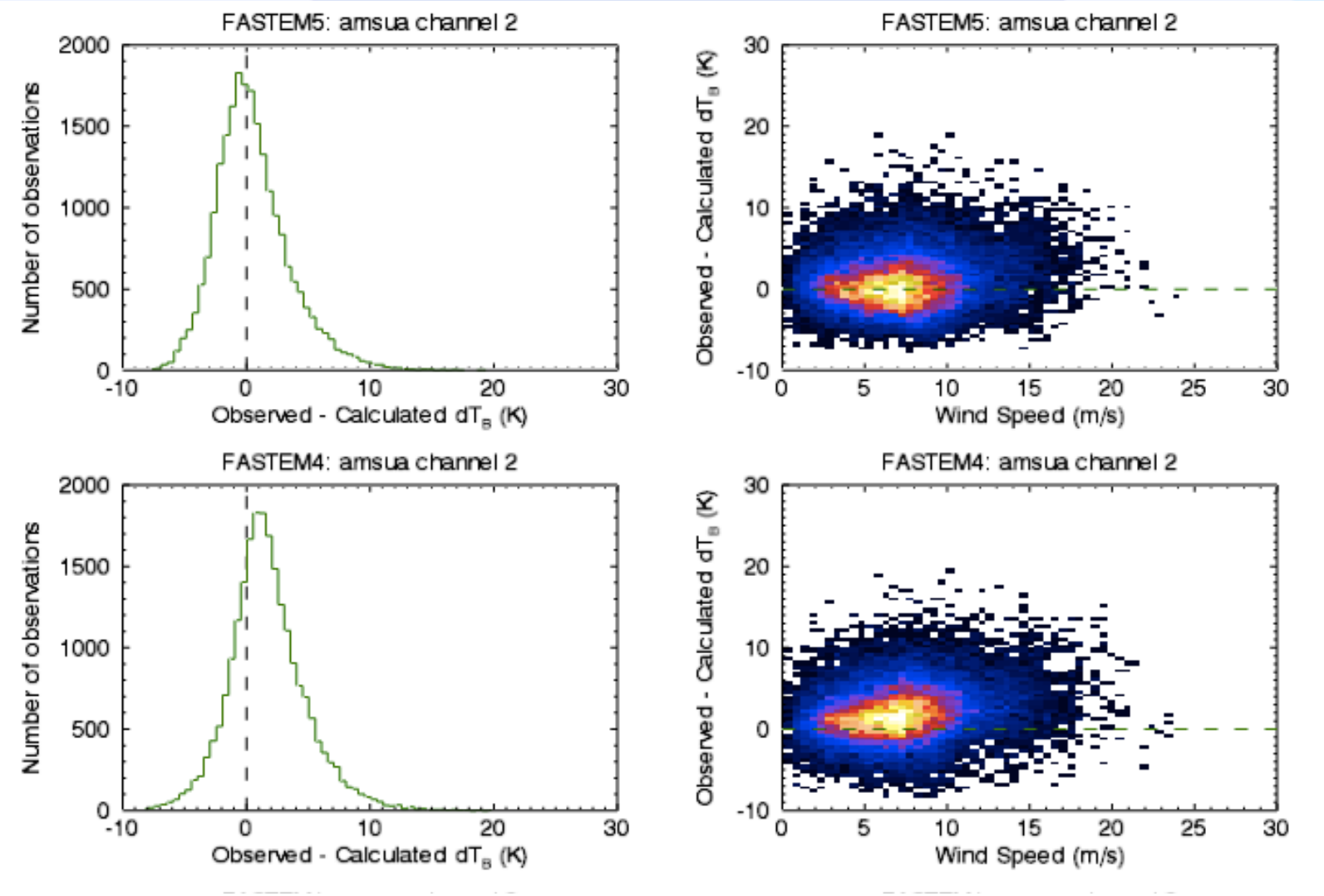
- Updates to the fitting coefficients for several components compared to FASTEM4
  - Foam coverage.
  - Downwelling reflected radiance correction.

- All the components are modeled separately via regression fits to data/models.
  - The same parameters are predictors (e.g. frequency, temperature, angle, wind speed, etc).

[1]St. Germain, K and G. Poe (1998), "Polarimetric emission model of the sea at microwave frequencies, Part II, Comparison with measurements", NRL, Washington, DC.

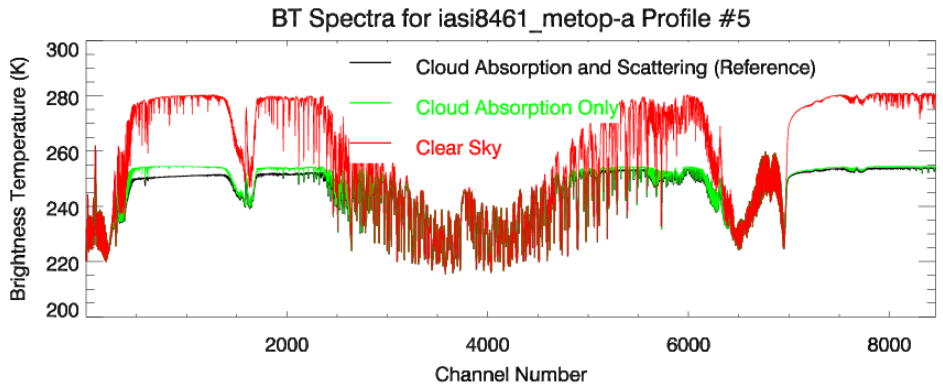
Microwave sea surface emissivity model (3)

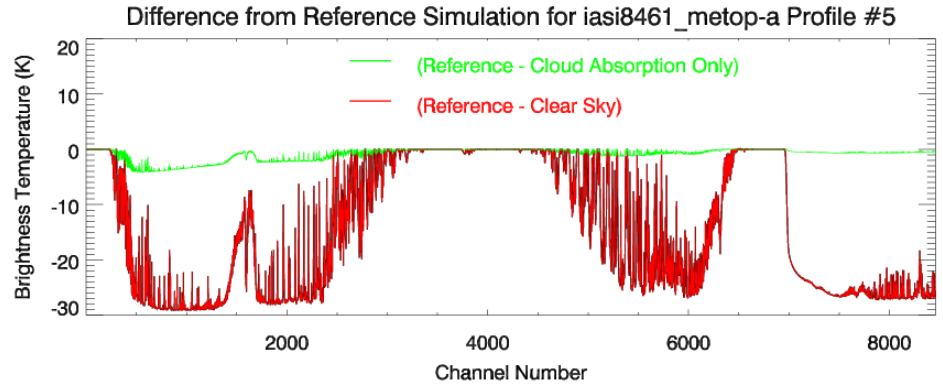
- Comparison of FASTEM5 and FASTEM4 for AMSU-A channel 2 from GSI single-cycle run

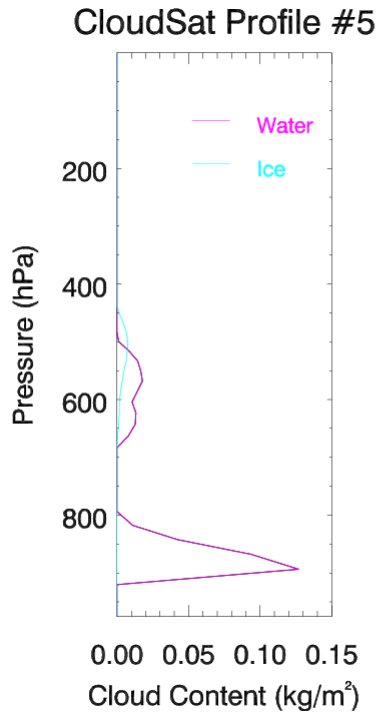


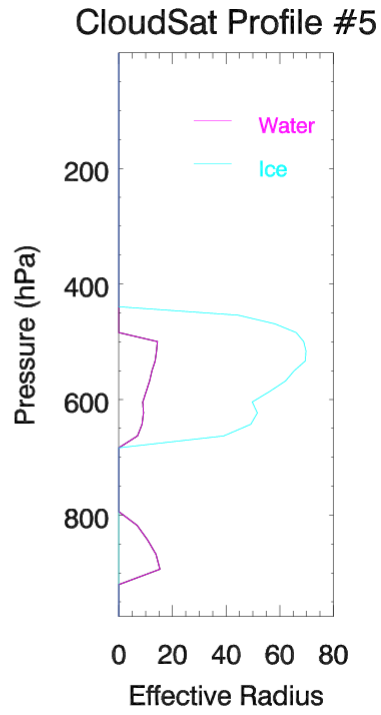
Scattering switch

- Implementation of a user-selectable switch to skip the scattering computations and only compute the cloud and aerosol *absorption*.
- Useful for initial cloud detection in assimilation system.
- Full characterisation (e.g. for various cloud scenarios) still ongoing.









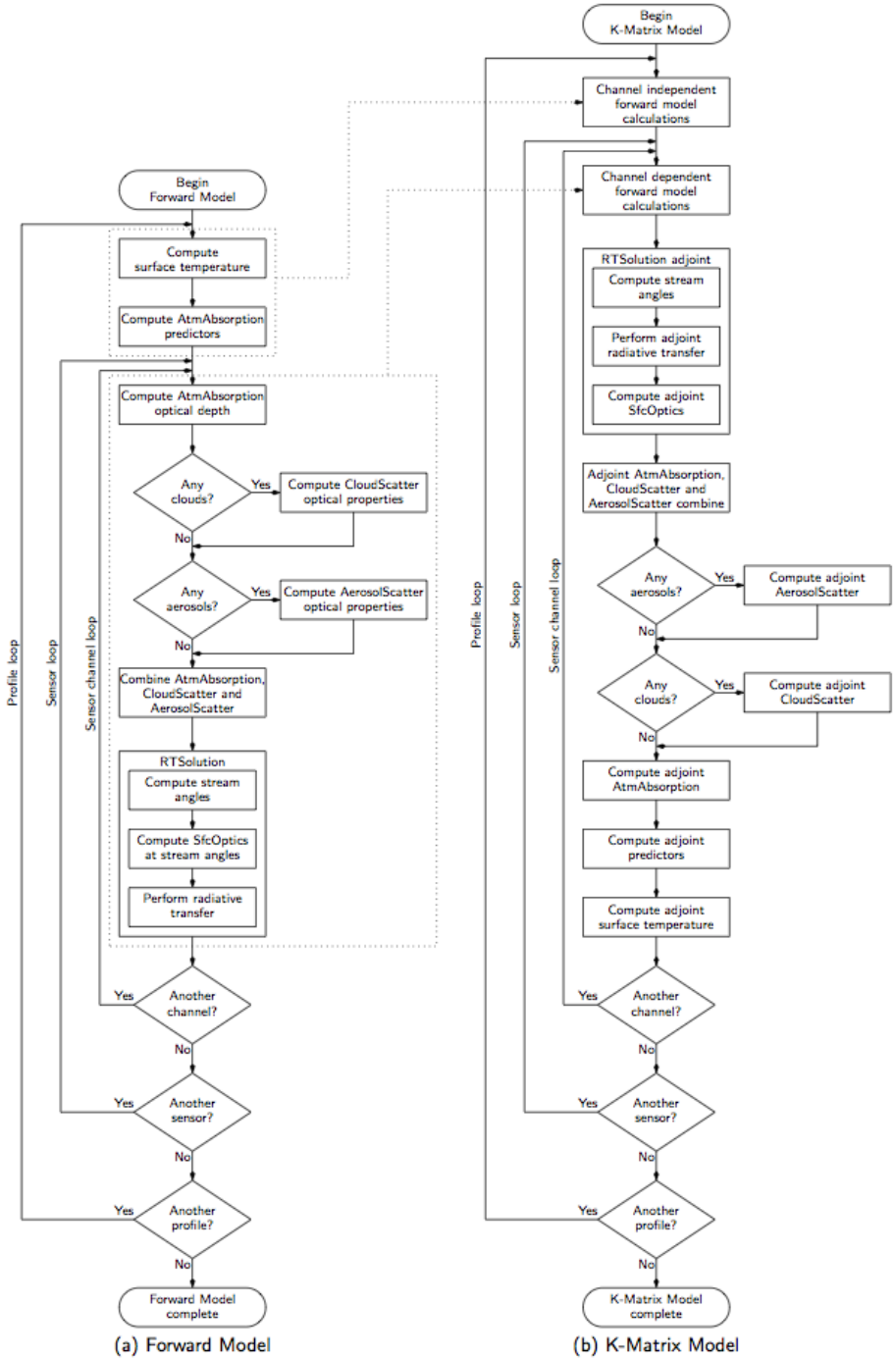
Other

- Implementation of additional RT solvers.
  - Successive Order of Interaction (SOI) algorithm implemented. Under review.
  - Additional algorithms being considered (VDISORT) for research use (fast, but too slow for operational use).
- AOD computation functions
  - Used by air quality investigators/forecasters.
- Aircraft model
  - User supplies an aircraft flight level pressure to turn this option on.
- Channel selection capability
  - Currently ALL channels for a given sensor are processed.
  - Selection capability allows users to turn off the processing of unwanted channels.
- User specified number of streams
  - Rather than the (currently unsophisticated) auto-select of the number of streams, users can input their own value.

CRTM INTERFACE

CRTM Interface

- Overview of procedure flow.
- Main function interface definitions
  - Dimensioning
  - Argument structure definitions
    - Atmosphere, Surface, Geometry, ChannelInfo, *Options*, RTSolution
- Initialisation interface definition
  - Argument definitions



Overview

- CRTM uses regression transmittance models
- Cloud and aerosol optical properties are read from a LUT
- Surface emissivity models are split into four “gross” surface types (land, water, snow, and ice) and three spectral regions (microwave, infrared, and visible). There is a great variety of model implementations, e.g. surface categories, emissivity atlases, empirical models, physical models.
- Radiative transfer is a simple emission model for clear sky; ADA, or SOI, for scattering.
- Inputs and outputs to the CRTM are packaged in their own structures.
- Tangent-linear, adjoint, and K-matrix functions are all constructed “line-by-line” from the forward model. Each are stand-alone, i.e. the forward model calls are incorporated in each.

Argument definitions

```
Error_Status = CRTM_Forward( &
  Atmosphere       , &
  Surface          , &
  Geometry         , &
  ChannelInfo      , &
  RTSolution       , &
  Options = Options  )
```

- Using the forward model as an example here.
- Structure definitions and dimensioning are appropriately similar for the tangent-linear, adjoint, and K-matrix functions.

Argument type definitions

```
TYPE(CRTM_Atmosphere_type),          INTENT(IN)      :: Atmosphere(:)    ! M
TYPE(CRTM_Surface_type),             INTENT(IN)      :: Surface(:)       ! M
TYPE(CRTM_Geometry_type),            INTENT(IN)      :: Geometry(:)      ! M
TYPE(CRTM_ChannelInfo_type),         INTENT(IN)      :: ChannelInfo(:)   ! N
TYPE(CRTM_RTSolution_type),          INTENT(IN OUT)  :: RTSolution(:,:)  ! L x M
TYPE(CRTM_Options_type), OPTIONAL,   INTENT(IN)      :: Options(:)       ! M
```

Dimension Key

L = number of channels
M = number of profiles
N = number of sensors

Atmosphere structure

```
Error_Status = CRTM_Forward( &
  Atmosphere        , &
  Surface           , &
  Geometry          , &
  ChannelInfo       , &
  RTSolution        , &
  Options = Options   )
```

```
TYPE :: CRTM_Atmosphere_type
  ! Dimension values
  INTEGER :: n_Layers     = 0  ! K dimension
  INTEGER :: n_Absorbers  = 0  ! J dimension
  INTEGER :: n_Clouds     = 0  ! Nc dimension
  INTEGER :: n_Aerosols   = 0  ! Na dimension
  ! Climatology model associated with the profile
  INTEGER :: Climatology = US_STANDARD_ATMOSPHERE
  ! Absorber ID and units
  INTEGER, ALLOCATABLE :: Absorber_ID(:)    ! J
  INTEGER, ALLOCATABLE :: Absorber_Units(:) ! J
  ! Profile LEVEL and LAYER quantities
  REAL(fp), ALLOCATABLE :: Level_Pressure(:)  ! 0:K
  REAL(fp), ALLOCATABLE :: Pressure(:)        ! K
  REAL(fp), ALLOCATABLE :: Temperature(:)     ! K
  REAL(fp), ALLOCATABLE :: Absorber(:,:)      ! K x J
  ! Clouds associated with each profile
  TYPE(CRTM_Cloud_type),   ALLOCATABLE :: Cloud(:)   ! Nc
  ! Aerosols associated with each profile
  TYPE(CRTM_Aerosol_type), ALLOCATABLE :: Aerosol(:) ! Na
END TYPE CRTM_Atmosphere_type
```

Cloud substructure

```
Error_Status = CRTM_Forward( &
  Atmosphere        , &
  Surface           , &
  Geometry          , &
  ChannelInfo       , &
  RTSolution        , &
  Options = Options   )
```

```
TYPE :: CRTM_Cloud_type
  ! Dimension values
  INTEGER :: n_Layers = 0  ! K dimension.
  ! Cloud type
  INTEGER :: Type = INVALID_CLOUD
  ! Cloud state variables
  REAL(fp), ALLOCATABLE :: Effective_Radius(:) ! K. um
  REAL(fp), ALLOCATABLE :: Water_Content(:)    ! K. kg/m^2
END TYPE CRTM_Cloud_type
```

```
TYPE :: CRTM_Atmosphere_type
  ! Dimension values
  INTEGER :: n_Layers      = 0  ! K dimension
                           = 0  ! J dimension
                           = 0  ! Nc dimension
                           = 0  ! Na dimension
sociated with the profile
 = US_STANDARD_ATMOSPHERE
s
: Absorber_ID(:)     ! J
: Absorber_Units(:)  ! J
YER quantities
                  :: Level_Pressure(:)  ! 0:K
  REAL(fp), ALLOCATABLE :: Pressure(:)        ! K
  REAL(fp), ALLOCATABLE :: Temperature(:)     ! K
  REAL(fp), ALLOCATABLE :: Absorber(:,:)      ! K x J
  ! Clouds associated with each profile
  TYPE(CRTM_Cloud_type),   ALLOCATABLE :: Cloud(:)    ! Nc
  ! Aerosols associated with each profile
  TYPE(CRTM_Aerosol_type), ALLOCATABLE :: Aerosol(:) ! Na
END TYPE CRTM_Atmosphere_type
```

Aerosol substructure

```
Error_Status = CRTM_Forward( &
  Atmosphere        , &
  Surface           , &
  Geometry          , &
  ChannelInfo       , &
  RTSolution        , &
  Options = Options   )
```

```
TYPE :: CRTM_Atmosphere_type
  ! Dimension values
  INTEGER :: n_Layers      = 0  ! K dimension
  INTEGER :: n_Absorbers   = 0  ! J dimension
  INTEGER :: n_Clouds      = 0  ! Nc dimension
                           = 0  ! Na dimension
sociated with the profile
 = US_STANDARD_ATMOSPHERE
s
: Absorber_ID(:)     ! J
: Absorber_Units(:)  ! J
YER quantities
:: Level_Pressure(:)  ! 0:K
:: Pressure(:)        ! K
:: Temperature(:)     ! K
  REAL(fp), ALLOCATABLE :: Absorber(:,:)       ! K x J
  ! Clouds associated with each profile
  TYPE(CRTM_Cloud_type),   ALLOCATABLE :: Cloud(:)    ! Nc
  ! Aerosols associated with each profile
  TYPE(CRTM_Aerosol_type), ALLOCATABLE :: Aerosol(:) ! Na
END TYPE CRTM_Atmosphere_type
```

```
TYPE :: CRTM_Aerosol_type
  ! Dimension values
  INTEGER :: n_Layers = 0  ! K dimension.
  ! Aerosol type
  INTEGER :: Type = INVALID_AEROSOL
  ! Aerosol state variables
  REAL(fp), ALLOCATABLE :: Effective_Radius(:) ! K. um
  REAL(fp), ALLOCATABLE :: Concentration(:)    ! K. kg/m^2
END TYPE CRTM_Aerosol_type
```

Surface structure

```
Error_Status = CRTM_Forward( &
  Atmosphere        , &
  Surface           , &
  Geometry          , &
  ChannelInfo       , &
  RTSolution        , &
  Options = Options   )
```

In v3.0, to accommodate multiple surface *subtypes* in a single FOV (in particular the land subtypes), the gross surface types will be split into their own structure arrays, e.g.

```
TYPE(LandSurface_type), ALLOCATABLE :: Land(:)
```

```
TYPE :: CRTM_Surface_type
  ! Gross type of surface determined by coverage
  REAL(fp) :: Land_Coverage  = ZERO
  REAL(fp) :: Water_Coverage = ZERO
  REAL(fp) :: Snow_Coverage  = ZERO
  REAL(fp) :: Ice_Coverage   = ZERO
  ! Land surface type data
  INTEGER  :: Land_Type = DEFAULT_LAND_TYPE
  ...other land surface inputs...
  ! Water type data
  INTEGER  :: Water_Type = DEFAULT_WATER_TYPE
  ...other water surface inputs...
  ! Snow surface type data
  INTEGER  :: Snow_Type = DEFAULT_SNOW_TYPE
  ...other snow surface inputs...
  ! Ice surface type data
  INTEGER  :: Ice_Type = DEFAULT_ICE_TYPE
  ...other ice surface inputs...
END TYPE CRTM_Surface_type
```

Geometry structure

```
Error_Status = CRTM_Forward( &
  Atmosphere       , &
  Surface          , &
  Geometry         , &
  ChannelInfo      , &
  RTSolution       , &
  Options = Options  )
```

```
TYPE :: CRTM_Geometry_type
  ! Field of view index (1-nFOV)
  INTEGER  :: iFOV = 0
  ! Earth location
  REAL(fp) :: Longitude        = ZERO
  REAL(fp) :: Latitude         = ZERO
  REAL(fp) :: Surface_Altitude = ZERO
  ! Sensor angle information
  REAL(fp) :: Sensor_Scan_Angle    = ZERO
  REAL(fp) :: Sensor_Zenith_Angle  = ZERO
  REAL(fp) :: Sensor_Azimuth_Angle = 999.9_fp
  ! Source angle information
  REAL(fp) :: Source_Zenith_Angle  = 100.0_fp
  REAL(fp) :: Source_Azimuth_Angle = ZERO
  ! Flux angle information
  REAL(fp) :: Flux_Zenith_Angle = DIFFUSIVITY_ANGLE
END TYPE CRTM_Geometry_type
```

Not all the data in the structure is always required. E.g.:

- `iFOV` is only used in antenna correction for the AMSU/MHS instruments,
- `Lat/Lon` is planned for use with atlas-type emissivity databases,
- Azimuth angles are only used in surface emissivity model for microwave polarimetric sensors, and in solar calculations for visible sensors. Recent VIIRS comparisons indicate it's also required for multi-detector SWIR and NIR channels.

`ChannelInfo` structure

```
Error_Status = CRTM_Forward( &
  Atmosphere       , &
  Surface          , &
  Geometry         , &
  ChannelInfo      , &
  RTSolution       , &
  Options = Options  )
```

Users do not have to worry about filling this data structure – it is done during the CRTM initialisation step.

CRTM v2.1 (re)introduces a channel selection capability. By default, all channels for a given sensor are processed. A call to the new channel subset function allows the user to control what channels are processed.

```
TYPE :: CRTM_ChannelInfo_type
  ! Dimensions
  INTEGER :: n_Channels = 0  ! L dimension
  ! Scalar data
  CHARACTER(STRLEN) :: Sensor_ID
  INTEGER           :: Sensor_Type
  INTEGER           :: WMO_Satellite_ID
  INTEGER           :: WMO_Sensor_ID
  INTEGER           :: Sensor_Index
  ! Array data
  LOGICAL, ALLOCATABLE :: Process_Channel(:)  ! L
  INTEGER, ALLOCATABLE :: Sensor_Channel(:)   ! L
  INTEGER, ALLOCATABLE :: Channel_Index(:)    ! L
END TYPE CRTM_ChannelInfo_type
```

RTSolution structure

```
Error_Status = CRTM_Forward( &
  Atmosphere       , &
  Surface          , &
  Geometry         , &
  ChannelInfo      , &
  RTSolution       , &
  Options = Options  )
```

```
TYPE :: CRTM_RTSolution_type
  ! Dimensions
  INTEGER :: n_Layers = 0  ! K
  ! Sensor information
  CHARACTER(STRLEN) :: Sensor_ID
  INTEGER           :: WMO_Satellite_ID
  INTEGER           :: WMO_Sensor_ID
  INTEGER           :: Sensor_Channel
  ! Forward intermediate results
  ! Not defined when they for TL, AD, and K.
  REAL(fp) :: Surface_Emissivity      = ZERO
  REAL(fp) :: Up_Radiance             = ZERO
  REAL(fp) :: Down_Radiance           = ZERO
  REAL(fp) :: Down_Solar_Radiance     = ZERO
  REAL(fp) :: Surface_Planck_Radiance = ZERO
  REAL(fp), ALLOCATABLE :: Upwelling_Radiance(:)
  ! The layer optical depths
  REAL(fp), ALLOCATABLE :: Layer_Optical_Depth(:)
  ! RT results for a single channel
  REAL(fp) :: Radiance               = ZERO
  REAL(fp) :: Brightness_Temperature = ZERO
END TYPE CRTM_RTSolution_type
```

The internal arrays do not *have* to be allocated – the code checks if they are allocated before attempting to assign them.

The channel dimension is handled by dimensioning the `RTSolution` argument to the required number of channels. We have considered putting the channel dimension inside the structure definition to make it congruent with the `ChannelInfo` input.

Options structure (optional)

```
Error_Status = CRTM_Forward( &
  Atmosphere        , &
  Surface           , &
  Geometry          , &
  ChannelInfo       , &
  RTSolution        , &
  Options = Options   )
```

This is where all the optional features are specified. (Note: not all options slated for v2.1 are shown here as they still reside in their respective branches in the repository.)

The emissivity/reflectivity input option was an early request (in v1.x). It will eventually be moved into its own structure, like the `SSU` and `Zeeman` components, but not until v3.0 (we restrict existing interface changes to the main version number updates).

Prior to v3.0, we've begun discussing implementation of a user-specified input reflectivity matrix.

```
TYPE :: CRTM_Options_type
  LOGICAL :: Check_Input              = .TRUE.
  LOGICAL :: Use_Old_MWSSEM           = .FALSE.
  LOGICAL :: Use_Antenna_Correction = .FALSE.
  LOGICAL :: Apply_NLTE_Correction  = .TRUE.
  LOGICAL :: Include_Scattering       = .TRUE.
  ! Aircraft flight level pressure
  ! Value > 0 turns "on" the aircraft option
  REAL(fp) :: Aircraft_Pressure = -ONE
  ! User defined emissivity/reflectivity
  INTEGER :: n_Channels = 0
  LOGICAL :: Use_Emissivity = .FALSE.
  REAL(fp), ALLOCATABLE :: Emissivity(:)
  LOGICAL :: Use_Direct_Reflectivity = .FALSE.
  REAL(fp), ALLOCATABLE :: Direct_Reflectivity(:)
  ! User defined number of RT streams
  LOGICAL :: Use_n_Streams = .FALSE.
  INTEGER :: n_Streams = 0
  ! SSU instrument input
  TYPE(SSU_Input_type) :: SSU
  ! Zeeman-splitting input
  TYPE(Zeeman_Input_type) :: Zeeman
END TYPE CRTM_Options_type
```

Other main functions

```
Error_Status = CRTM_Tangent_Linear( &
  Atmosphere       , &   ! M
  Surface          , &   ! M
  Atmosphere_TL    , &   ! M
  Surface_TL       , &   ! M
  Geometry         , &   ! M
  ChannelInfo      , &   ! N
  RTSolution       , &   ! L x M
  RTSolution_TL    , &   ! L x M
  Options = Options  )   ! M
```

- Similar interfaces for tangent-linear, adjoint, and K-matrix functions.
- Adjoint and K-matrix functions are similar
  - Adjoint is a strict transpose of the tangent-linear code, so atmosphere and surface adjoint results are a sum over channels.
  - K-matrix function simply moves all the channel independent code inside the channel loop to provide Jacobians for each channel.

```
Error_Status = CRTM_Adjoint( &
  Atmosphere       , &   ! M
  Surface          , &   ! M
  RTSolution_AD    , &   ! L x M
  Geometry         , &   ! M
  ChannelInfo      , &   ! N
  Atmosphere_AD    , &   ! M
  Surface_AD       , &   ! M
  RTSolution       , &   ! L x M
  Options = Options  )   ! M
```

```
Error_Status = CRTM_K_Matrix( &
  Atmosphere       , &   ! M
  Surface          , &   ! M
  RTSolution_K     , &   ! L x M
  Geometry         , &   ! M
  ChannelInfo      , &   ! N
  Atmosphere_K     , &   ! L x M
  Surface_K        , &   ! L x M
  RTSolution       , &   ! L x M
  Options = Options  )
```

Note dimension change

Initialisation function

```
Error_Status = CRTM_Init( &
  Sensor_ID                              , &   ! N
  ChannelInfo                            , &   ! N
  File_Path          = File_Path         , &   ! Scalar
  Load_CloudCoeff    = Load_CloudCoeff   , &   ! Scalar
  Load_AerosolCoeff = Load_AerosolCoeff, &   ! Scalar
  …optional specification of explicit coefficient filenames… )
```

`CHARACTER(*), INTENT(IN) :: Sensor_ID(:)`

- Character string identifying the sensor and platform for which radiances are to be computed.
- Filenames are constructed from this value.
- Standard specification of "**`sensor_platform`**"
  - **`abi_gr`**
  - **`amusa_metop-a`**
  - **`iasi_metop-b`**
  - **`cris_npp`**
  - Etc..
- Dimension, **`N`**, is the number of sensors. Most often this is just one.

Initialisation function

```
Error_Status = CRTM_Init( &
  Sensor_ID                                , &   ! N
  ChannelInfo                              , &   ! N
  File_Path          = File_Path           , &   ! Scalar
  Load_CloudCoeff    = Load_CloudCoeff     , &   ! Scalar
  Load_AerosolCoeff  = Load_AerosolCoeff,  &   ! Scalar
  …optional specification of explicit coefficient filenames… )
```

`TYPE(CRTM_ChannelInfo_type), INTENT(OUT) :: ChannelInfo(:)`

- Structure array (described previously) whose contents are populated based on the contents of the sensor coefficient files..
- Dimension, **`N`**, is the number of sensors. Must be the same as **`Sensor_Id`**.

Initialisation function

```
Error_Status = CRTM_Init( &
  Sensor_ID                              , &   ! N
  ChannelInfo                            , &   ! N
  File_Path          = File_Path         , &   ! Scalar
  Load_CloudCoeff    = Load_CloudCoeff   , &   ! Scalar
  Load_AerosolCoeff  = Load_AerosolCoeff , &   ! Scalar
  …optional specification of explicit coefficient filenames… )
```

`CHARACTER(*), OPTIONAL, INTENT(IN) :: File_Path`

- Character string specifying a file path for the various input coefficient files.
- If not specified, the current directory is the default.
- No fine-grain control of locations; all files in one place.

Initialisation function

```
Error_Status = CRTM_Init( &
  Sensor_ID                                , &   ! N
  ChannelInfo                              , &   ! N
  File_Path          = File_Path           , &   ! Scalar
  Load_CloudCoeff    = Load_CloudCoeff     , &   ! Scalar
  Load_AerosolCoeff = Load_AerosolCoeff,  &   ! Scalar
  …optional specification of explicit coefficient filenames… )
```

```
LOGICAL, OPTIONAL, INTENT(IN) :: Load_CloudCoeff
LOGICAL, OPTIONAL, INTENT(IN) :: Load_AerosolCoeff
```

- Logical switches to disable loading of cloud and aerosol optical properties look-up tables.
- Saves memory for clear-sky-only computations.
- Default is to load the data. Set to **`.FALSE.`** to disable.

CALLING THE CRTM

Example of calling the CRTM

- Will use K-matrix function.
- Dimensions to be defined:
  - `n_Profiles`
  - `n_Layers`
  - `n_Absorbers`
  - `n_Clouds`
  - `n_Aerosols`
- We'll set the number of sensor, `n_Sensors`, to just one.

Step 0: Define the dimensions

```
! This example processes TWO profiles of 100 layers and
!                                          2 absorbers and
!                                          2 clouds and
!                                          1 aerosol....
INTEGER, PARAMETER :: N_PROFILES  = 2
INTEGER, PARAMETER :: N_LAYERS    = 100
INTEGER, PARAMETER :: N_ABSORBERS = 2
INTEGER, PARAMETER :: N_CLOUDS    = 2
INTEGER, PARAMETER :: N_AEROSOLS  = 1
! ...but only ONE Sensor at a time
INTEGER, PARAMETER :: N_SENSORS = 1
```

Step 1: Define the variables

- Let's make everything allocatable.
- Plan for future expansion/changes.

```
CHARACTER(256)              , ALLOCATABLE :: sensor_id(:)
TYPE(CRTM_ChannelInfo_type), ALLOCATABLE :: chinfo(:)
TYPE(CRTM_Geometry_type)    , ALLOCATABLE :: geo(:)

! Define the FORWARD variables
TYPE(CRTM_Atmosphere_type) , ALLOCATABLE :: atm(:)
TYPE(CRTM_Surface_type)    , ALLOCATABLE :: sfc(:)
TYPE(CRTM_RTSolution_type) , ALLOCATABLE :: rts(:,:)

! Define the K-MATRIX variables
TYPE(CRTM_Atmosphere_type) , ALLOCATABLE :: atm_K(:,:)
TYPE(CRTM_Surface_type)    , ALLOCATABLE :: sfc_K(:,:)
TYPE(CRTM_RTSolution_type) , ALLOCATABLE :: rts_K(:,:)
```

Note that the FORWARD and K-MATRIX variables are the same type.

Done to minimise code maintenance, but means K-matrix variables cannot have extra internal dimensions, e.g. channels.

Step 2: Initialise the CRTM

- Let's initialise the CRTM for the GOES-R ABI.
- We'll use the Nalli IR sea surface emissivity model.

Allocate the sensor dependent arrays. Use `abi_gr` as example.

IR water EmisCoeff default filename is '`EmisCoeff.bin`', but we distribute the various files named by model for historical consistency only.

```
n_Sensors = 1
ALLOCATE( sensor_id(n_sensors), chinfo(n_sensors) )
sensor_id = 'abi_gr'

emiscoeff_file = 'Nalli.EK-PDF.W_W-RefInd.EmisCoeff.bin'

Error_Status = CRTM_Init( sensor_id, &
                          chinfo, &
                          EmisCoeff_File = emiscoeff_file )
IF ( Error_Status /= SUCCESS ) THEN
  …handle error…
END IF
```

Error status integer results are either SUCCESS, FAILURE, or WARNING.

Step 2b: Select channels

- Default is to process ALL channels for a sensor.
- New **`ChannelInfo`** procedures allow user selection.

Channel list is sorted into ascending order regardless of order specified in subset.

```
! Select the channel subset to be processed
Error_Status = CRTM_ChannelInfo_Subset( &
                 chinfo(1), &
                 Channel_Subset = (/ 16, 11, 13 /) )
IF ( Error_Status /= SUCCESS ) THEN
  …handle error…
END IF

! Determine the total number of channels to be processed
n_Channels = CRTM_ChannelInfo_n_Channels(chinfo(1))

! The list of channels to be processed
WRITE( *,'(1x,10i5)' ) CRTM_ChannelInfo_Channels(chinfo(1))
```

Step 3a: Allocate ARRAYS

- Don’t confuse structure arrays with the structure components.
- Both need to be allocated.

```
! Allocate the ARRAYS
ALLOCATE( atm(N_PROFILES), sfc(N_PROFILES), geo(N_PROFILES), &
          rts(n_channels, N_PROFILES), &
          atm_K(n_channels, N_PROFILES), &
          sfc_K(n_channels, N_PROFILES), &
          rts_K(n_channels, N_PROFILES), &
          STAT = Allocate_Status )
IF ( Allocate_Status /= 0 ) THEN
  …handle error…
END IF
```

Step 3b: Allocate STRUCTURES

- Recall that structure creation procedures are ELEMENTAL.
- Only allocation of the atmosphere structure is mandatory.

Elemental procedures make allocation of structure arrays simple(r).

```
! The input FORWARD structure
CALL CRTM_Atmosphere_Create( &
       atm, N_LAYERS, N_ABSORBERS, N_CLOUDS, N_AEROSOLS )
IF ( ANY(.NOT. CRTM_Atmosphere_Associated(atm)) ) THEN
  …handle error…
END IF

! The output K-MATRIX structure
CALL CRTM_Atmosphere_Create( &
       atm_K, N_LAYERS, N_ABSORBERS, N_CLOUDS, N_AEROSOLS )
IF ( ANY(.NOT. CRTM_Atmosphere_Associated(atm_K)) ) THEN
  …handle error…
END IF
```

Note the use of the `ANY()` intrinsic..

Step 4: Assign input data

- Once the structures are allocated, fill them with the necessary data.
- **`ChannelInfo`** is handled via initialisation and procedures.
- **`Atmosphere`** and **`Surface`** are assigned values as appropriate.

Surface data can be specified for multiple surface types: land, water, snow, and ice

```
sfc(1)%Land_Coverage    = 1.0_fp
sfc(1)%Land_Type        = SCRUB
sfc(1)%Land_Temperature = 318.0_fp
sfc(1)%LAI              = 1.70_fp
sfc(1)%Soil_Type        = MEDIUM
sfc(1)%Vegetation_Type  = GROUNDCOVER
```

- Climatology required for "profile extension".
- Absorber id is needed to identify the absorber.
- Absorber units are currently fixed: g/kg for H_2O, ppmv for everything else.

```
atm(1)%Climatology    = TROPICAL
atm(1)%Absorber_Id    = &
  (/ H2O_ID, O3_ID /)
atm(1)%Absorber_Units = &
  (/ MASS_MIXING_RATIO_UNITS, &
     VOLUME_MIXING_RATIO_UNITS /)
atm(1)%Level_Pressure = &
atm(1)%Pressure = &
atm(1)%Temperature = &
```

The order of absorber data must match the absorber identifier.

```
atm(1)%Absorber(:,1) = …H2O data…
atm(1)%Absorber(:,2) = …O3 data…
```

Step 4: Assign input data

- **`Atmosphere`** component **`Cloud`** and **`Aerosol`** are also assigned data as appropriate.

Other cloud types are ICE, RAIN, SNOW, GRAUPEL, and HAIL.

GO-CART aerosol species only. Work on CMAQ has been postponed indefinitely.

```
! Some cloud data
k1 = 75  ! Pressure[k1] = 650.104hPa
k2 = 79  ! Pressure[k2] = 741.693hPa
atm(1)%Cloud(1)%Type = WATER_CLOUD
atm(1)%Cloud(1)%Effective_Radius(k1:k2) = 20.0_fp ! Microns
atm(1)%Cloud(1)%Water_Content(k1:k2)    = 5.0_fp  ! kg/m^2

! Some pretend aerosol data
k1 = 83  ! Pressure[k1] = 840.016hPa
k2 = 85  ! Pressure[k2] = 891.679hPa
atm(1)%Aerosol(1)%Type = DUST_AEROSOL
atm(1)%Aerosol(1)%Effective_Radius(k1:k2) = 2.0_fp ! microns
atm(1)%Aerosol(1)%Concentration(k1:k2)    = 5.0_fp ! kg/m^2
```

Step 4: Assign input data

- **`Geometry`**. Structure also assigned values as appropriate.

Only used for AMSU-A/B and MHS antenna temperature correction.

Only used for emissivity database location matching.

These are (currently) the only "mandatory" elements – unless you always want to compute nadir radiances.

```
TYPE :: CRTM_Geometry_type
  ! Field of view index (1-nFOV)
  INTEGER  :: iFOV = 0
  ! Earth location
  REAL(fp) :: Longitude         = ZERO
  REAL(fp) :: Latitude          = ZERO
  REAL(fp) :: Surface_Altitude  = ZERO
  ! Sensor angle information
  REAL(fp) :: Sensor_Scan_Angle     = ZERO
  REAL(fp) :: Sensor_Zenith_Angle   = ZERO
  REAL(fp) :: Sensor_Azimuth_Angle  = 999.9_fp
  ! Source angle information
  REAL(fp) :: Source_Zenith_Angle   = 100.0_fp
  REAL(fp) :: Source_Azimuth_Angle  = ZERO
  ! Flux angle information
  REAL(fp) :: Flux_Zenith_Angle = DIFFUSIVITY_ANGLE
END TYPE CRTM_Geometry_type
```

Step 5: Initialise the K-matrix arguments

- Initialise the inputs for the sensor-dependent result.
- Zero out the outputs (just to be sure).

```
! Zero the K-matrix OUTPUT structures
CALL CRTM_Atmosphere_Zero( atm_K )
CALL CRTM_Surface_Zero( sfc_K )

! Initialize the K-matrix INPUT
DO l = 1, n_Channels
  IF ( ChannelInfo(1)%Sensor_Type == INFRARED_SENSOR .OR. &
       ChannelInfo(1)%Sensor_Type == MICROWAVE_SENSOR ) THEN
    RTSolution_K(l,:)%Brightness_Temperature = ONE
  ELSE
    RTSolution_K(l,:)%Radiance = ONE
  END IF

END DO
```

IR/MW Jacobians are dTb/dx.

Visible Jacobians are dR/dx.

Step 6: Call the K-matrix function

If it's a forward model output, it's a K-matrix input.

If it's a forward model input, it's a K-matrix output.

```
Error_Status = CRTM_K_Matrix( &
                      atm   , &
                      sfc   , &
                      rts_K , &
                      geo   , &
                      chinfo, &
                      atm_K , &
                      sfc_K , &
                      rts     )
IF ( Error_Status /= SUCCESS ) THEN
  …handle error…
END IF
```

Step 7: Destroy the CRTM

- This step deallocates memory allocated during initialisation.
- Note: deallocation of structure ARRAYS is sufficient to deallocate structure components.

```
Error_Status = CRTM_Destroy( chinfo )
IF ( Error_Status /= SUCCESS ) THEN
  …handle error…
END IF


DEALLOCATE( sensor_id, chinfo, &
            atm, sfc, geo, &
            rts_K, &
            atm_K, sfc_K, &
            rts )
```

No need to call the various structure `Destroy()` procedures explicitly.

CRTM OBJECT METHODS

Object methods

- Huh?
- Currently, no functional OO concepts are implemented in the CRTM.
- So, for our purposes here:
  - Object ≅ Derived type (aka structure)
  - Method ≅ Function or subroutine (aka procedure, or type-bound procedure) operating on an instance (aka variable) of that type.
- We've defined a list of list of default procedures that must be available for all CRTM objects.

Naming conventions

- Modules
  - All structures and their procedures are defined in their respective definition modules,

    CRTM_*structure*_**Define**
  - Currently, I/O procedures are defined in a separate module,

    CRTM_*structure*_**IO**

    but will eventually be moved to the definition module.
- Procedures are named following the convention

  CRTM_*structure_action*

Implemented default procedures

Action	Type	Description
`Associated`	Elemental function	Tests if the structure components have been allocated.
`Destroy`	Elemental subroutine	Deallocates any allocated structure components.
`Create`	Elemental subroutine	Allocates any allocatable structure components.
`Inspect`	Subroutine	Displays structure contents.
`OPERATOR(==)`	Elemental function	Tests the equality of two structure.

Example default procedures

```
MODULE CRTM_Atmosphere_Define
  …etc…
  PUBLIC :: OPERATOR(==)
 …etc…
  PUBLIC :: CRTM_Atmosphere_Associated
  PUBLIC :: CRTM_Atmosphere_Destroy
  PUBLIC :: CRTM_Atmosphere_Create
  PUBLIC :: CRTM_Atmosphere_Inspect
  …etc…
END MODULE CRTM_Atmosphere_Define
```

Example default procedures

```
MODULE CRTM_Atmosphere_Define
  …etc…
  PUBLIC :: OPERATOR(==)
 …etc…
  PUBLIC :: CRTM_Atmosphere_Associated
  PUBLIC :: CRTM_Atmosphere_Destroy
  PUBLIC :: CRTM_Atmosphere_Create
  PUBLIC :: CRTM_Atmosphere_Inspect
  …etc…
END MODULE CRTM_Atmosphere_Define
```

```
TYPE(CRTM_Atmosphere_type) :: atm1(10), atm2(10)  ! Many profiles

…etc…

IF ( .NOT. ALL(atm1 == atm2) ) THEN
  …handle non-equality…
END IF
```

Example default procedures

```
MODULE CRTM_Atmosphere_Define
  …etc…
  PUBLIC :: OPERATOR(==)
 …etc…
  PUBLIC :: CRTM_Atmosphere_Associated
  PUBLIC :: CRTM_Atmosphere_Destroy
  PUBLIC :: CRTM_Atmosphere_Create
  PUBLIC :: CRTM_Atmosphere_Inspect
  …etc…
END MODULE CRTM_Atmosphere_Define
```

```
TYPE(CRTM_Atmosphere_type) :: atm(100)   ! Many profiles

…etc…

IF ( .NOT. ALL(CRTM_Atmosphere_Associated(atm)) ) THEN
  …handle non-association…
END IF
```

Example default procedures

```
MODULE CRTM_Atmosphere_Define
  …etc…
  PUBLIC :: OPERATOR(==)
 …etc…
  PUBLIC :: CRTM_Atmosphere_Associated
  PUBLIC :: CRTM_Atmosphere_Destroy
  PUBLIC :: CRTM_Atmosphere_Create
  PUBLIC :: CRTM_Atmosphere_Inspect
…etc…
END MODULE CRTM_Atmosphere_Define
```

```
TYPE(CRTM_Atmosphere_type) :: atm(100)  ! Many profiles
INTEGER :: scalar_n_layers, rank1_n_layers(100)

…etc…

! All profiles have same layering
scalar_n_layers = 64
CALL CRTM_Atmosphere_Create( atm, scalar_n_layers, …etc… )

…etc…

! All profiles can have different layering
Rank1_n_layers = [28,64,91,…,75]
CALL CRTM_Atmosphere_Create( atm, rank1_n_layers, …etc… )
```

Example default procedures

```
MODULE CRTM_Atmosphere_Define
  …etc…
  PUBLIC :: OPERATOR(==)
 …etc…
  PUBLIC :: CRTM_Atmosphere_Associated
  PUBLIC :: CRTM_Atmosphere_Destroy
  PUBLIC :: CRTM_Atmosphere_Create
  PUBLIC :: CRTM_Atmosphere_Inspect
…etc…
END MODULE CRTM_Atmosphere_Define
```

```
TYPE(CRTM_Atmosphere_type) :: atm(100)  ! Many profiles

…allocation…etc…

! Destroy structure components
CALL CRTM_Atmosphere_Destroy( atm )
```

Example default procedures

```
MODULE CRTM_Atmosphere_Define
  …etc…
  PUBLIC :: OPERATOR(==)
 …etc…
  PUBLIC :: CRTM_Atmosphere_Associated
  PUBLIC :: CRTM_Atmosphere_Destroy
  PUBLIC :: CRTM_Atmosphere_Create
  PUBLIC :: CRTM_Atmosphere_Inspect
…etc…
END MODULE CRTM_Atmosphere_Define
```

```
TYPE(CRTM_Atmosphere_type) :: atm(100)  ! Many profiles

…etc…

DO i = 1, 100
  CALL CRTM_Atmosphere_Inspect( atm(i) )
END DO
```

I/O procedures

Action	Type	Description
`InquireFile`	Function	Inquires file for dimensions to use in allocations if necessary. As yet, doesn't actually act upon the structure.
`WriteFile`	Function	Write a structure instance to file.
`ReadFile`	Function	Loads a structure instance with data read from file.