Tangent Linear and Adjoint Coding Short Course
Day 3
Jacobian (or K) Coding

Thomas J. Kleespies
Room 810

Review of Day 2 Problem Set

TL Subroutine part 1

```
Do ichan = 1 , M

! Initialize integrator
 Sum_TL=0.

! Compute tl radiance for first level
 Call Planck_TL(Vnu(ichan),T(1),T_TL(1),B(1,Ichan),B_TL(1))

! Now compute radiances for the rest of the levels
 Do level=2,N
  Call Planck_TL(Vnu(ichan),T(level),T_TL(level),B(level,Ichan),B_TL(level))

   (B_TL(level-1)      +B_TL(level      ))*(Tau   (level-1,ichan)-Tau  (level,ichan))
 + (B   (level-1,ichan)+B   (level,ichan))*(Tau_TL(level-1,ichan)-Tau_TL(level,ichan)
                                         )
 EndDo
```

TL Subroutine part 2

```
! Surface term, ignoring downward reflected
 Call Planck_TL(Vnu(ichan),Tskin,Tskin_TL,Bs(Ichan),Bs_TL)

!Sum=Sum+Bs*Tau(N,ichan)*Emiss(ichan)  ! forward left commented in place for comparison
 Sum_TL = Sum_TL + Bs_TL       *Tau(N,ichan)   *Emiss(ichan)    &
                 + Bs(ichan)   *Tau_TL(N,ichan)*Emiss(ichan)    &
                 + Bs(ichan)   *Tau(N,ichan)   *Emiss_TL(ichan)

! Now brightness temperature
 Tb_TL(ichan) = 0
 If(TotalRad(Ichan).gt.0.) Then
   Tb_TL(ichan) = Bright_TL(Vnu(ichan),TotalRad(Ichan),Sum_TL,BC1(ichan),BC2(ichan))
 EndIf

EndDo ! ichan
```

AD Subroutine part 1

```
Do ichan = 1 , M  ! don't need to reverse this loop
                  ! because there is no interchannel dependence

 Sum_AD = 0.0  ! want to segregrate channels
 Bs_AD  = 0.0  ! so zero here
 B_AD   = 0.0
               ! Don't zero Tau_AD and Emiss_AD because they come in from outside
! Now brightness temperature
 If(TotalRad(Ichan).gt.0.) Then
   CallBright_AD(Vnu(ichan),TotalRad(Ichan),                &
                Sum_AD,BC1(ichan),BC2(ichan),Tb_AD(ichan))
 EndIf

 Bs_AD             = Bs_AD           + Sum_AD    *Tau(N,ichan)  *Emiss(ichan)
 Tau_AD(N,ichan)   = Tau_AD(N,ichan) + Bs(ichan) *Sum_AD        *Emiss(ichan)
 Emiss_AD(ichan)   = Emiss_AD(ichan) + Bs(ichan) *Tau(N,ichan)  *Sum_AD

! Surface term, ignoring downward reflected

! WARNING... WHAT IF WE NEED Bs_AD elsewhere... keep this in mind.
 Call Planck_AD(Vnu(ichan),Tskin,Tskin_AD,Bs(Ichan),Bs_AD)

! Now compute radiances for the rest of the levels
c
```

AD Subroutine part 2

```
 Do level = N , 2 , -1  ! this loop needs to be reversed because of vertical
                        ! dependencies

  B_AD(level-1) = B_AD(level-1) +.5* Sum_AD*(Tau(level-1,ichan)-Tau(level,ichan))
  B_AD(level  ) = B_AD(level  ) +.5* Sum_AD*(Tau(level-1,ichan)-Tau(level,ichan))

  Tau_AD(level-1,ichan) = Tau_AD(level-1,ichan)  &
                        +.5*(B(level,ichan)+B(level-1,ichan))*Sum_AD
  Tau_AD(level  ,ichan) = Tau_AD(level  ,ichan)  &
                        -.5*(B(level,ichan)+B(level-1,ichan))*Sum_AD

  Call Planck_AD(Vnu(ichan),T(level),T_AD(level),B(level,Ichan),B_AD(level))

 EndDo

! Compute AD for first level

 Call Planck_AD(Vnu(ichan),T(1),T_AD(1),B(1,Ichan),B_AD(1))


EndDo ! ichan
```

AD Testing Setup

```
Real Fin (940)
Real TLin(nTLin)
Real ADout(nADout)

! Forward input vector
Equivalence (Fin(1  ), T     )
Equivalence (Fin(47 ), Tau   )
Equivalence (Fin(921), Emiss )
Equivalence (Fin(940), Tskin )

! TL input vector
Equivalence (TLin(1  ), T_TL     )
Equivalence (TLin(47 ), Tau_TL   )
Equivalence (TLin(921), Emiss_TL )
Equivalence (TLin(940), Tskin_TL )

! AD output vector
Equivalence (ADout(1  ), T_AD     )
Equivalence (ADout(47 ), Tau_AD   )
Equivalence (ADout(921), Emiss_AD )
Equivalence (ADout(940), Tskin_AD )

Real Tb   (Nchan)   ! brightness temperature
Real Tb_TL(nTLout)  ! brightness temperature TL
Real Tb_AD(nADin)   ! brightness temperature AD

Real TLout(nTLout)
Real ADin (nADin)
Equivalence(TLout,Tb_TL)
Equivalence(ADin, Tb_AD)
Real TL(nTLin,nTLout)   ! TL Jacobian Matrix
Real AD(nADin,nADout)   ! AD Jacobian Matrix
```

AD Testing - Filling Matrices

```
! compute forward model values and variables here for use with TL and AD
 Call Compbright_Save(Vnu,T,Tau,Tskin,Emiss,BC1,BC2,Nlevel,Nchan,Tb)

! Form TL Jacobian matrix
 Do i = 1 , nTLin
  TLin = 0.0
  TLout = 0.0
  TLin(i) = 1.0
  Call Compbright_Save_TL(Vnu,                        &
                          T,    T_TL,                 &
                          Tau,  Tau_TL,               &
                          Tskin,Tskin_TL,             &
                          Emiss,Emiss_TL,             &
                          BC1,BC2,Nlevel,Nchan,       &
                          Tb,   Tb_TL)
  TL(i,1:nTLout) = TLout(1:nTLout)
 EndDo ! TL loop

! Form AD Jacobian matrix
 Do j = 1 , nADin
  ADin = 0.0
  ADout = 0.0
  ADin(j) = 1.0
  Call Compbright_Save_AD(Vnu,                        &
                          T,    T_AD,                 &
                          Tau,  Tau_AD,               &
                          Tskin,Tskin_AD,             &
                          Emiss,Emiss_AD,             &
                          BC1,BC2,Nlevel,Nchan,       &
                          Tb,   Tb_AD)
  AD(j,1:nADout) = ADout(1:nADout)
 EndDo ! TL loop
```

AD Testing – The Final Test

```
Do i = 1 , nTLout
 Do j = 1 , nADout

  If(TL(j,i) /= AD(i,j)) Then
   ktall = ktall + 1  ! counter for total unequal

   If(TL(j,i) == 0.0) Then
    Write(68,6192) i,j,TL(j,i), AD(i,j) , ' AD'
    KtAD = KtAD + 1
   EndIf

   If(AD(i,j) == 0.0) Then
    Write(68,6192) i,j,TL(j,i), AD(i,j) , ' TL'
    KtTL = KtTL + 1
   EndIf

   If(TL(j,i) /= 0.0 .and. AD(i,j) /= 0.0) Then
    If(abs((TL(j,i) - AD(i,j)) / TL(j,i)) < 1.e-12) Then
     Write(68,6192) i,j,TL(j,i), AD(i,j) , ' close',(TL(j,i) - AD(i,j)) / TL(j,i)
     KtClose = KtClose + 1
    Else
     Write(68,6192) i,j,TL(j,i), AD(i,j) , ' both ',(TL(j,i) - AD(i,j)) / TL(j,i)
     KtBad = KtBad + 1
    EndIf
   EndIf

  EndIf ! tl /= ad

 EndDo
EndDo
```

Adjoint Testing Results

```
1     2   0.1633989889760E-02   0.1633989889760E-02 close   0.1327061053780E-15
1     3   0.3174658014545E-02   0.3174658014545E-02 close  -0.4098213417073E-15
1     4   0.5125247936936E-02   0.5125247936936E-02 close  -0.1692331275796E-15
1     5   0.7920142828543E-02   0.7920142828543E-02 close   0.2190267920075E-15
1     6   0.1833180437177E-01   0.1833180437177E-01 close   0.3785166895296E-15
1     7   0.3748511350924E-01   0.3748511350924E-01 close  -0.1851106547189E-15
1     8   0.4503646633803E-01   0.4503646633803E-01 close  -0.1540727874125E-15
1    12   0.4086103064603E-01   0.4086103064603E-01 close  -0.1698169085361E-15
1    13   0.4823906210725E-01   0.4823906210725E-01 close   0.1438438808881E-15
                                        .
                                        .
                                        .


  1272 Total elements not matching
  1272 Elements that are close
     0 Elements AD /= 0, TL = 0
     0 Elements TL /= 0, AD = 0
     0 Elements just don't agree
```

Jacobian, or K code development and testing is the easiest of the three.

The Jacobian code is often called the K code. The reason is historical, and that’s all I know.

Our objective is the Jacobian

$$K(x)^T = \begin{bmatrix} \frac{\partial R_1}{\partial T_1} & \frac{\partial R_2}{\partial T_1} & \frac{\partial R_3}{\partial T_1} & \cdots & \frac{\partial R_m}{\partial T_1} \\ \frac{\partial R_1}{\partial T_2} & \frac{\partial R_2}{\partial T_2} & \frac{\partial R_3}{\partial T_2} & \cdots & \frac{\partial R_m}{\partial T_2} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{\partial R_1}{\partial T_n} & \frac{\partial R_2}{\partial T_n} & \frac{\partial R_3}{\partial T_n} & \cdots & \frac{\partial R_m}{\partial T_n} \\ \frac{\partial R_1}{\partial q_1} & \frac{\partial R_2}{\partial q_1} & \frac{\partial R_3}{\partial q_1} & \cdots & \frac{\partial R_m}{\partial q_1} \\ \frac{\partial R_1}{\partial q_2} & \frac{\partial R_2}{\partial q_2} & \frac{\partial R_3}{\partial q_2} & \cdots & \frac{\partial R_m}{\partial q_2} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{\partial R_1}{\partial q_n} & \frac{\partial R_2}{\partial q_n} & \frac{\partial R_3}{\partial q_n} & \cdots & \frac{\partial R_m}{\partial q_n} \end{bmatrix}$$

Adjoint Output

$$K(x)^T = \begin{bmatrix} \frac{\partial R}{\partial T_1} \\ \frac{\partial R}{\partial T_2} \\ \vdots \\ \frac{\partial R}{\partial T_n} \\ \frac{\partial R}{\partial q_1} \\ \frac{\partial R}{\partial q_2} \\ \vdots \\ \frac{\partial R}{\partial q_n} \end{bmatrix}$$

For a single call to AD, output is derivative of all channel radiances with respect to each element of the input state vector.

We need to distribute the adjoint level derivatives through the number of channels.

How to do this

- Examine the outputs of the AD.
- Add a channel dimension to those that do not have one.
- Add a channel loop if necessary.

- In general, low level routines K code is same as AD. Save channel loop for higher level routines.

Recommended Jacobian Naming Conventions

There is no ‘standard’ naming convention. Here is what I recommend:

- Keep forward model variable names the same
- Append “ _K” to forward model variable and routine names to describe Jacobian variables and routines

```
Subroutine Planck_AD(V,Temp,Temp_AD, B, B_AD)
! Computes Planck AD wavenumber V, Radiance B, temperature Temp and B_AD
Implicit None
Real V     ! Input wavenumber
Real B     ! Input Radiance mW m-2 sr-1 (cm-1)-1
Real Temp  ! Input Temperature (K)
Real Temp_AD ! Output AD Temperature
Real B_AD    !  Input AD Radiance
Real C1 /1.1905e-5/
Real C2 /1.4385/

Temp_AD = Temp_AD + C2*V*B*B*B_AD*exp(C2*V/Temp)/(C1*V*V*V*Temp*Temp)

 Return
End Subroutine Planck_AD

Subroutine Planck_K(V,Temp,Temp_K, B, B_K)
! Computes Planck K, wavenumber V, temperature Temp, Temp_K, Radiance B and B_K
Implicit None
Real V     ! Input wavenumber
Real B     ! Input Radiance mW m-2 sr-1 (cm-1)-1
Real Temp  ! Input Temperature (K)
Real Temp_K ! Output K Temperature
Real B_K    !  Input K Radiance
Real C1 /1.1905e-5/
Real C2 /1.4385/

Temp_K = Temp_K + C2*V*B*B*B_K*exp(C2*V/Temp)/(C1*V*V*V*Temp*Temp)

 Return
End Subroutine Planck_K
```

First examine AD outputs to see which variables need a channel dimension.

```
Subroutine CompBright_Save_AD(Vnu,T,T_AD,Tau,Tau_AD,  &
                        Tskin,Tskin_AD,Emiss,Emiss_AD,&
                        BC1,BC2,N,M,                 &
                        Tb,Tb_AD)
```

Active Variables

N levels

M channels

```
Real T_AD(N)      ! Need to expand by channel
Real Tau_AD(N,M) ! This is OK
Real Tskin_AD     ! Need to expand by channel
Real Emiss_AD(M) ! This is OK
```

Second: Add the channel dimension to these variables in declaration and assignment statements

This is within channel loop:

```
Call Planck_AD &
(Vnu(ichan),T(level),T_AD(level),B(level,Ichan),B_AD(level))

Call Planck_K  &
(Vnu(ichan),T(level),T_K(level,Ichan),B(level,Ichan),B_K(level))
```

If necessary, add channel loop.

You do not have do add a channel loop in your code.

Jacobian Testing

- Goal is to assure that Jacobian is consistent with adjoint.
- Done by making sure that the sum by channel of the Jacobian elements matches the corresponding Adjoint element.

$$K(x)^T = \begin{bmatrix} \frac{\partial R_1}{\partial T_1} & \frac{\partial R_2}{\partial T_1} & \frac{\partial R_3}{\partial T_1} & \cdots & \frac{\partial R_m}{\partial T_1} \\ \frac{\partial R_1}{\partial T_2} & \frac{\partial R_2}{\partial T_2} & \frac{\partial R_3}{\partial T_2} & \cdots & \frac{\partial R_m}{\partial T_2} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{\partial R_1}{\partial T_n} & \frac{\partial R_2}{\partial T_n} & \frac{\partial R_3}{\partial T_n} & \cdots & \frac{\partial R_m}{\partial T_n} \\ \frac{\partial R_1}{\partial q_1} & \frac{\partial R_2}{\partial q_1} & \frac{\partial R_3}{\partial q_1} & \cdots & \frac{\partial R_m}{\partial q_1} \\ \frac{\partial R_1}{\partial q_2} & \frac{\partial R_2}{\partial q_2} & \frac{\partial R_3}{\partial q_2} & \cdots & \frac{\partial R_m}{\partial q_2} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{\partial R_1}{\partial q_n} & \frac{\partial R_2}{\partial q_n} & \frac{\partial R_3}{\partial q_n} & \cdots & \frac{\partial R_m}{\partial q_n} \end{bmatrix}$$

$$K(x)^T = \begin{bmatrix} \frac{\partial R}{\partial T_1} \\ \frac{\partial R}{\partial T_2} \\ \vdots \\ \frac{\partial R}{\partial T_n} \\ \frac{\partial R}{\partial q_1} \\ \frac{\partial R}{\partial q_2} \\ \vdots \\ \frac{\partial R}{\partial q_n} \end{bmatrix}$$

$$\sum_{i=1}^{m} \frac{\partial R_m}{\partial x_j}_{K} = \frac{\partial R}{\partial x_j}_{AD}$$

K Code Testing

1. Call Adjoint with ALL inputs set to unity. Make sure you have zeroed all outputs.
2. Call K with ALL inputs set to unity. Make sure you have zeroed all outputs.
3. For each level/variable, sum the channels of K. This should be equal to AD to within machine precision.
4. If an element of both K and AD equal zero, this is probably OK.

K Code Testing- Exception

If the AD output variable has a channel dimension, don’t sum the K output by channel. Compare the AD and K variables element by element. In your code this is Tau_AD, Tau_K, Emiss_AD, Emiss_K.

Machine Precision Considerations

Test that

$\mathrm{Abs}(K-AD)/AD < MP$

Rule of thumb:

MP = 1.e-7 for Single precision

MP = 1.e-12 for Double precision

Pitfalls

If you compute Abs(K-AD)/AD and find that you get a number like 1, 2, ½ , (an integer or a fraction with one over an integer in the denominator) you probably are summing over channels too many times. Comment out the channel loop from a low level routine and repeat test.

```
! Ke is summed jacobian… should be equal to AD
Do j = 1 , nADout

   If(Ke(j) /= AD(j)) Then  ! Test if unequal
    ktall = ktall + 1  ! counter for total unequal
    Write(68,6192) j,Ke(j), AD(j) , ' found',(AD(j) - Ke(j)) / AD(j)
         If(Ke(j) == 0.0) Then  ! K = 0, AD not
          Write(68,6192) j,Ke(j), AD(j) , ' AD'
          KtAD = KtAD + 1
         EndIf

         If(AD(j) == 0.0) Then  ! AD = 0, K not
          Write(68,6192) j,Ke(j), AD(j) , ' K'
          KtKe = KtKe + 1
         EndIf

         If(K(j) /= 0.0 .and. AD(j) /= 0.0) Then  ! Both not eq 0
           If(abs((AD(j) - Ke(j)) / AD(j)) < 1.e-12) Then  ! But close enough
            Write(68,6192) j,Ke(j), AD(j) , ' close',(AD(j) - Ke(j)) / AD(j)
            KtClose = KtClose + 1
          Else                            ! BAD BAD BAD
            Write(68,6192) j,Ke(j), AD(j) , ' both ',(AD(j) - Ke(j)) / AD(j)
            KtBad = KtBad + 1
           EndIf
         EndIf

   EndIf ! Ke /= ad

EndDo
```

Assignment for Today

- Write COMPBRIGHT_SAVE_K.F90 based upon COMBRIGHT_SAVE_AD which I have provided.
- Test this routine using techniques we have learned today.
- Don’t dilly dally.
- We will meet back here at 4PM.

Review of Assignment

1: Identify the active K variables that need a dimension in channels

Real T(N)

Real T_K(N,M)

Real Tau(N,M)

Real Tau_K(N,M)

Real Tskin

Real Tskin_K(M)

Real Emiss(M)

Real Emiss_K(M)

2: Make sure that the K variables are accumulated by channel

```
Call Planck_K(Vnu(ichan),Tskin,Tskin_K(Ichan),Bs(Ichan),Bs_K)

Call Planck_K(Vnu(ichan),T(level),T_K(level,Ichan),  &
                    B(level,Ichan),B_K(level))

Call Planck_K(Vnu(ichan),T(1),T_K(1,Ichan),B(1,Ichan),B_K(1))
```

Other than re-naming the routine and variable names from _AD to _K, above is the only code changes necessary.

```
! Forward input vector
Equivalence (Fin(1  ), T     )
Equivalence (Fin(47 ), Tau   )
Equivalence (Fin(921), Emiss )
Equivalence (Fin(940), Tskin )

! K output vector
Equivalence (Kout(1  ), T_K     )
Equivalence (Kout(875), Tau_K   )
Equivalence (Kout(1749), Emiss_K )
Equivalence (Kout(1768), Tskin_K )

! AD output vector
Equivalence (ADout(1  ), T_AD     )
Equivalence (ADout(47 ), Tau_AD   )
Equivalence (ADout(921), Emiss_AD )
Equivalence (ADout(940), Tskin_AD )

! K equivalent output vector
Equivalence (Keout(1  ), T_Ke     )
Equivalence (Keout(47 ), Tau_Ke   )
Equivalence (Keout(921), Emiss_Ke )
Equivalence (Keout(940), Tskin_Ke )

Real Tb   (Nchan)   ! brightness temperature
Real Tb_K (nKin )   ! brightness temperature K
Real Tb_AD(nADin)   ! brightness temperature AD

Real Kin  (nKin)
Real ADin (nADin)

Equivalence(Kin , Tb_K)
Equivalence(ADin, Tb_AD)

Real K (nKout )
Real AD(nADout)
Real Ke(nKeout)
```

```
  Kin  = 1.0
  Kout = 0.0
  Keout= 0.0
  Call Compbright_Save_K (Vnu,                        &
                          T,    T_K,                  &
                          Tau,  Tau_K,                &
                          Tskin,Tskin_K,              &
                          Emiss,Emiss_K,              &
                          BC1,BC2,Nlevel,Nchan,       &
                          Tb,   Tb_K)
  Do i = 1 , Nchan
   Tskin_Ke = Tskin_Ke + Tskin_K(i)
   Do j = 1 , Nlevel
    T_Ke(j) = T_Ke(j) + T_K(j,i)
   EndDo
  EndDo

  Tau_Ke = Tau_K
  Emiss_Ke = Emiss_K ! Don’t compress Tau_k and emiss_k because they are naturally mxn in AD

  Ke = Keout

! Compute AD
  ADin = 1.0
  ADout = 0.0
  Call Compbright_Save_AD(Vnu,                        &
                          T,    T_AD,                 &
                          Tau,  Tau_AD,               &
                          Tskin,Tskin_AD,             &
                          Emiss,Emiss_AD,             &
                          BC1,BC2,Nlevel,Nchan,       &
                          Tb,   Tb_AD)

  AD = ADout
```

Single precision. Double precision reveals no differences.

```
  15   0.2418907582760E+00   0.2418907433748E+00 found  -0.6160286147860E-07
  18   0.2083575427532E+00   0.2083575576544E+00 found   0.7151725611720E-07
  22   0.2486892938614E+00   0.2486893236637E+00 found   0.1198375656486E-06
  27   0.2785069644451E+00   0.2785069942474E+00 found   0.1070074446829E-06
  28   0.3823396861553E+00   0.3823396563530E+00 found  -0.7794724155019E-07
  30   0.4918318092823E+00   0.4918317794800E+00 found  -0.6059454449314E-07
  32   0.4514432251453E+00   0.4514432549477E+00 found   0.6601565871733E-07
  41   0.5664035081863E+00   0.5664035677910E+00 found   0.1052335250051E-06
  46   0.9583471715450E-01   0.9583472460508E-01 found   0.7774406185490E-07
        9 Total elements not matching
        0 Elements that are close
        0 Elements AD /= 0, K = 0
        0 Elements Ke /= 0, AD = 0
        0 Elements just don't agree
```

We have now seen four ways to compute Jacobians

1. Finite differencing – 2N+1 -forward calcs
2. Tangent Linear – N -TL calcs
3. Adjoint – M -AD calcs
4. K - ~ 3 forward calcs

Using the code

- In real life, just the forward and K codes are used.
- The K code is called with just the brightness temperatures (or equivalent input) set to unity.
- DO NOT throw out the TL and AD codes. You will need them if you make any changes to the forward model. These changes MUST be propagated through the TL,AD and K and tested each step of the way.

Congratulations! You are now Adjoint Programmers, or you will be when you finish all of the assignments.