Tangent Linear and Adjoint Coding Short Course
Day 1
Overview and Tangent Linear Coding

Thomas J. Kleespies
Room 810

Class Rules

1. **If I start talking too fast, stop me!**
2. **Do the homework. This is a hands on class. If you don't finish an assignment, do it next week. It will be easy because I will have given my answer (which may be different from you're equally correct answer. There is no uniqueness theorem for these codes).**
3. **There will be a prize for the first to turn in each day's correct solution.**
4. **Come to all of the classes.**
5. **Auditors need to keep their questions until after class.**
6. **Be safe, not stupid! Practice safe computing. Never use software without a shrink wrap.**

Comments on Giering&Kaminski

+ useful for formal aspects of mathematics and coding

+ good coding examples

- combines tangent linear and adjoint steps

- does not differentiate between adjoint and Jacobian

--- does not discuss testing

What's it all about?

1DVAR / maximum probability solution is that which minimizes a 'cost' or 'penalty function:

$$\mathbf{J}(\mathbf{x}) = \left(\mathbf{x}-\mathbf{x}^{b}\right)^{T}\mathbf{B}^{-1}\left(\mathbf{x}-\mathbf{x}^{b}\right) + \left(\mathbf{y}^{o}-\mathbf{y}(\mathbf{x})\right)^{T}\mathbf{O}^{-1}\left(\mathbf{y}^{o}-\mathbf{y}(\mathbf{x})\right)$$

where $\mathbf{x}^{b}$ is an initial estimate given by the model state vector, $\mathbf{x}$ is the model state for which the solution is desired, $\mathbf{y}^{o}$ is the vector of observations, $\mathbf{y}(\mathbf{x})$ is an operator which transforms the model state vector into the same parameters as the observations, and $\mathbf{B}$ and $\mathbf{O}$ are the background and observational error covariance matrices respectively. For our purposes, $\mathbf{y}(\mathbf{x})$ is the radiative transfer operator. Note that $\mathbf{O}$ is a combination of observational errors and radiative transfer errors. (This is just a least squares problem)

What's it all about: part deux

How do we find the minimum? From first quarter Calculus: Take the first derivative and set it equal to zero.

$$\nabla \mathbf{J}(\mathbf{x}) = \mathbf{B}^{-1}\left(\mathbf{x} - \mathbf{x}^{b}\right) - \mathbf{K}(\mathbf{x})^{T}\ \mathbf{O}^{-1}\left(\mathbf{y}^{o} - \mathbf{y}(\mathbf{x})\right) = 0$$

where **K(x)** is the matrix of partial derivatives of **y(x)** with respect to the elements of **x**. (factor of 2 divides out)

What's it all about: part trois

It is evident that the solution requires both the forward radiative transfer operator **y(x)**, and the transpose of its derivative, $\mathbf{K(x)^T}$. $\mathbf{K(x)^T}$ is called the adjoint, or Jacobian.

$$\mathbf{x = \{ T_1, T_2, T_3, \ldots, T_n, q_1, q_2, q_3, \ldots, q_n, \ldots\}}$$

$$\mathbf{y(x) = \{R_1, R_2, R_3, \ldots, R_m\}^T}$$

What's it all about, part quatre

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

What's it all about, part cinq

In olden days (say 1990), computation of $\mathbf{K(x)^T}$ required N+1 forward model calculations using forward (or backward) finite differencing (centered required 2N+1). Thus these techniques were only used in limited studies

In these modern times, using adjoint coding techniques $\mathbf{K(x)^T}$ can be computed with the effort of about 3 forward model calculations.

Why all the models?

- The tangent linear model is derived from the forward model

  - gives the derivative of the radiance with respect to the state vector (vector output, m channels)

- The adjoint is derived from the tangent linear model

  - gives the transpose of the derivative of the radiance with respect to the state vector (vector output, N variables)

- The Jacobian is derived from the adjoint model

  - gives the transpose of the derivative of the radiance with respect to the state vector by channel (matrix output, Nxm)

- At NCEP, only the forward and the Jacobian models are actually used, but all models must be developed and maintained in order to assure a testing path, and to make sure the performance is correct.

Why can’t we just use the Tangent Linear Model

- **You can.**
- **However, it still takes N TL calculations.**
- **You avoid the finite differencing because the TL is the analytic derivative, but you just get a vector of radiances for each call. You still have to call it for each element of the input vector.**

Where we are going?

- **Today**
  - **What's it all about**
  - **Tangent linear coding and testing**
- **Tomorrow**
  - **Adjoint coding and testing**
- **Day Three**
  - **Jacobian coding and testing**

Nomenclature

- **Forward Model Variables: Those variables that contain values used or computed in the forward model (T,p,k,q,τ)**
- **Tangent Linear Variables: Those variables that contain differential values computed in the tangent linear model.**
- **Active variables: “Variables depending on the control variables and having an influence on the cost function are called active” (ex, temperature, moisture, OPTRAN absorber amounts in weighted predictors)**
- **Passive variable: those not active. (e.g. constants, OPTRAN absorber amount coordinate system, orbital elements, indices)**
- **Perturbed Forward Model: Forward model called with the input vector perturbed by the TL input vector (+-)**

Recommended TL Naming Conventions

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

- **Keep forward model variable names the same**
- **Append “ _TL” to forward model variable and routine names to describe tangent linear variables and routines**

Tangent Linear Coding Rules

- Only assignment statements need to be differentiated
- There **must** be one TL routine for each forward model routine
- Generally the full forward model is invoked before the TL
- Accommodations must be made to carry required forward model variables to the TL code

Tangent Linear Coding Example

Equation:

$y = a + bx + cx^2 + dx^3 + ez^{1/2}$

Differential:

$y' = bx' + 2cxx' + 3dx^2x' + 1/2ez^{-1/2}z'$

Forward Code:

```
Y = A + B*X + C*X**2. + D*X**3. + E*SQRT(Z)
```

TL Code:

```
Y_TL = B*X_TL + 2.*C*X*X_TL + 3.*D*X**2.*X_TL  &
     * .5*E*Z**(-.5)*Z_TL
```

Forward Model Variables in TL Code

```
Y_TL = B*X_TL + 2.*C*X*X_TL + 3.*D*X**2.*X_TL  &
     * .5*E*Z**(-.5)*Z_TL
```

Need X and Z to satisfy this code fragment.

Depending on speed/memory tradeoffs:

1) Re-compute forward variable (can make testing tricky if you create an error in re-coding the forward calculation)
2) Store on a temporary file (IO bound, bookkeeping)
3) Store in memory (cleanest method if not too many ACTIVE forward variables)

What the heck are these TL variables?

- **TL output variables are the derivative of the forward model output with respect to each element of the input variables. e.g.** $\partial \mathrm{Tb}_5 / \partial \mathrm{q}_{18}$

- **Internal TL variables represent local derivatives. The chain rule sorts the results out in the end.**

- **Don't ask me for a proof of this… see the literature. I'm just here to teach the coding, not the theory.**

Conditionals

- **Logical tests in the forward model MUST be carried to the TL model using the ACTIVE FORWARD MODEL VARIABLES**

Forward example:	TL example:
IF(T > 273.) THEN	IF(T > 273.) THEN ! NOT T_TL
Q = T**2.	Q_TL = 2.*T*T_TL
ELSE	ELSE
Q = 2.*T	Q_TL = 2.*T_TL
ENDIF	ENDIF

General TL Rule

- **If it has an equal sign in it, differentiate it.**
- **If it doesn’t, leave it alone.**
- **Only exception is subroutines & function names, etc.**

Testing TL Model consistency with Forward Model

$$\lim_{\Delta \mathbf{x}^{\pm} \to 0} \frac{\mathrm{FM}(\mathbf{x}+\Delta\mathbf{x}) - \mathrm{FM}(\mathbf{x})}{\mathrm{TLM}(\mathrm{x}, \Delta\mathbf{x})} = 1$$

This looks a lot like the definition of the derivative.

FM(x) = Forward model acting on x
FM(x+Δx)=perturbed Forward model acting on x+Δx
TL(x,Δx)= Tangent Linear model acting on Δx (at x)

Testing TL Model consistency with Forward Model, part zwei

- **It is best to write and test the TL of each routine before going to the next**
- **Start with the bottom most routine and work up**
- **Guidelines for limit test:**
  - **Call the forward model first. Keep results. (exception as TBD later)**
  - **Pick an initial increment of 10% of the forward model input values**
  - **Write an outer loop that halves the increment, 10-15 iterations**
  - **Apply the increment independently in both a positive and negative sense to both the perturbed forward model and the TL model. Do this for ALL variables/levels at a time.**
  - **Calculate the ratio and observe how it approaches unity from both sides. If it converges to something other than unity, or doesn't converge, start checking your code, and/or check precision.**

```
Subroutine Planck_TL(V,Temp,Temp_TL,B, B_TL)

! Computes Planck TL radiance at wavenumber V, Radiance B,
! temperature  Temp and Temp_TL

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_TL ! Input TL Temperature
Real B_TL    ! Output TL Radiance

Real C1 /1.1905e-5/
Real C2 /1.4385/

!forward model code included as a comment for reference
!B = (C1*v*v*v)/(Exp(C2*V/Temp) - 1.)
 B_TL = C2*V*B*B*Temp_TL*exp(C2*V/Temp)/(C1*V*V*V*Temp*Temp)

 Return
End Subroutine Planck_TL
```

! Code fragment from testing routine… most of the work done here.

```
 Call Planck(Vnu(Ichan),Temp,B) ! Compute forward model radiance

 Temp_TL(1) = -Temp / 10.  ! initial value  negative increment
 Temp_TL(2) =  Temp / 10.  ! initial value positive increment

 Write(6,*) ' HIRS channel ',Ichan ! This just prints a header
 Write(2,*) ' HIRS channel ',Ichan
 Write(6,6110) " Iter   Neg dx        Pos dx        Neg ratio     Pos ratio"
 Write(2,6110) " Iter   Neg dx        Pos dx        Neg ratio     Pos ratio"

 Do i = 1 , niter ! outer loop emulates taking the limit

! Compute TL values asymptotically
  Do isign = 1 , 2 ! inner loop delta x -> 0 +-
   Call Planck(Vnu(Ichan),Temp+Temp_TL(isign),BP(Isign))      ! perturbed forward model
   Call Planck_TL(Vnu(Ichan), Temp,Temp_TL(isign), B, B_TL(Isign)) ! tangent linear model
   Ratio(isign) = (BP(Isign) - B ) / B_TL(Isign)         ! ratio
  EndDo

  Write(6,6120) i, Temp_TL,Ratio(1),Ratio(2)
  Write(2,6120) i, Temp_TL,Ratio(1),Ratio(2)

  Temp_TL(1) = Temp_TL(1) * 0.5 ! now halve the input TL temperature and repeat
  Temp_TL(2) = Temp_TL(2) * 0.5

 EndDo
```

Example of Output of testing routine (Your results may vary)

HIRS channel 16

Iter	Neg dx	Pos dx	Neg ratio	Pos ratio
1	-25.000000000	25.000000000	0.590156871	1.729834400
2	-12.500000000	12.500000000	0.764005950	1.315447047
3	-6.250000000	6.250000000	0.873208093	1.146620850
4	-3.125000000	3.125000000	0.934268719	1.070686369
5	-1.562500000	1.562500000	0.966532956	1.034705682
6	-0.781250000	0.781250000	0.983113900	1.017195751
7	-0.390625000	0.390625000	0.991518526	1.008558887
8	-0.195312500	0.195312500	0.995749622	1.004269732
9	-0.097656250	0.097656250	0.997872396	1.002132442
10	-0.048828125	0.048828125	0.998935594	1.001065616
11	-0.024414062	0.024414062	0.999467646	1.000532657
12	-0.012207031	0.012207031	0.999733785	1.000266291
13	-0.006103516	0.006103516	0.999866883	1.000133136
14	-0.003051758	0.003051758	0.999933439	1.000066566
15	-0.001525879	0.001525879	0.999966719	1.000033282

1: ratio approaches 1 from both sides, but do not expect negative to approach from below.
2: as perturbation enters linear regime, ratio goes half the distance to the goal line each iteration
3: If you do too many iterations, ratio may get strange because of machine precision. Try DP.

Example of something ‘apparently’ going wrong

Test_Bright_TL output Single Precision

HIRS channel 16

Iter	Neg dx	Pos dx	Neg ratio	Pos ratio
1	-0.054812677	0.054812677	1.044734240	0.960472047
2	-0.027406339	0.027406339	1.021662235	0.979655027
3	-0.013703169	0.013703169	1.010679841	0.989705265
4	-0.006851585	0.006851585	1.005261421	0.994774103
5	-0.003425792	0.003425792	1.002581358	0.997454226
6	-0.001712896	0.001712896	1.001183033	0.998852491
7	-0.000856448	0.000856448	1.000250816	0.999318600
8	-0.000428224	0.000428224	1.001183033	0.999318600
9	-0.000214112	0.000214112	0.999318600	0.999318600
10	-0.000107056	0.000107056	0.999318600	0.999318600
11	-0.000053528	0.000053528	0.999318600	0.999318600
12	-0.000026764	0.000026764	1.014233828	1.014233828
13	-0.000013382	0.000013382	1.014233828	1.014233828
14	-0.000006691	0.000006691	0.954572976	0.954572976
15	-0.000003346	0.000003346	0.954572976	0.954572976

Switch to DP reveals problem is machine precision

Test_Bright_TL output Double Precision

HIRS channel 16

Iter	Neg dx	Pos dx	Neg ratio	Pos ratio
1	-0.054812675	0.054812675	1.044735123	0.960478588
2	-0.027406337	0.027406337	1.021643085	0.979654926
3	-0.013703169	0.013703169	1.010650418	0.989673749
4	-0.006851584	0.006851584	1.005283591	0.994797430
5	-0.003425792	0.003425792	1.002631531	0.997388722
6	-0.001712896	0.001712896	1.001313217	0.998691846
7	-0.000856448	0.000856448	1.000655973	0.999345292
8	-0.000428224	0.000428224	1.000327828	0.999672488
9	-0.000214112	0.000214112	1.000163875	0.999836205
10	-0.000107056	0.000107056	1.000081927	0.999918092
11	-0.000053528	0.000053528	1.000040961	0.999959044
12	-0.000026764	0.000026764	1.000020480	0.999979521
13	-0.000013382	0.000013382	1.000010240	0.999989761
14	-0.000006691	0.000006691	1.000005120	0.999994880
15	-0.000003346	0.000003346	1.000002560	0.999997440

Testing TL Model consistency with Forward Model, part drei

- **The previous simple example was easy to test because only one output TL variable**
- **Ex: If you have a TL input profile with surface variables:**
  - Perturb all input variables at once.
  - Sometimes it is useful to perturb only a part of the input vector at once, e.g. Temperature only, or surface parameters only. This will help isolate where is the error.
- **In general, test each TL output variable**
- **How this is done depends upon each routine.**
- **If you have stored forward model variables in COMMON,** be careful about getting them mixed with perturbed forward model variables**.(call perturbed forward first, then straight forward, then TL).**

Testing TL Model consistency with Forward Model, part vier

Tip: If you have a lot of TL inputs, string them into a vector for easy handling using F90 vector operations:

```
Real T(40),q(40),o3(40),tsfc,emiss
Real FMBuf(122)
Equivalence(FMBuf(1),   T )
Equivalence(FMBuf(41),  Q )
Equivalence(FMBuf(81),  O3 )
Equivalence(FMBuf(121), tsfc)
Equivalence(FMBuf(122), emiss)
Real T_TL(40),q_TL(40),o3_TL(40),tsfc_TL,emiss_TL
Real TLBuf(122)
Equivalence(TLBuf(1),   T_TL )
Equivalence(TLBuf(41),  Q_TL )
Equivalence(TLBuf(81),  O3_TL )
Equivalence(TLBuf(121), tsfc_TL)
Equivalence(TLBuf(122), emiss_TL)
TLBuf = FMBuf * .1
```

Then construct the ratio from the selected element of the output. This method will also make Adjoint testing easier, as we will see.

Problem Set for Tomorrow:

Construct routine COMPBRIGHT_TL.F90 from forward code COMPBRIGHT.F90 and test using techniques learned today.

Low level routines Planck, Bright, Planck_TL, and Bright_TL and their testing routines are provided for reference.

Problem Set for Tomorrow cont: Things to watch out for:

You will need forward model intermediate variables in you TL code. You may choose to:

1) Recompute them in the TL code (dangerous and expensive). This gets very dangerous in the Adjoint code because you have forward code in a backward environment. Expensive because you repeat forward work.

2) Carry them from the forward model code in COMMON, for example (advised if memory not an issue). This will require modification of the forward routine COMPBRIGHT.F90 that I provided. Analyze your derivatives to assess exactly what intermediate Forward variables are required (including dimensions). I recommend making a slight name change to COMPBRIGHT.F90 to track your changes. Make sure that the computed brightness temperatures are correct after your modifications.

If you choose (2), construct the testing routine so that the perturbed forward model is called 1st, then the forward model, then the TL model. If you call Forward, Perturbed Forward, TL, the TL will act with the Perturbed Forward variables, and not converge properly, even if TL is correct.

Problem Set for Tomorrow cont: Things to watch out for:

Hint: Get rid of the temporary variables B1,B2,Tau1,Tau2, and save the local radiances B(level),Bs, and total radiance

```
! Compute radiance for first level
 b1=Planck(Vnu(ichan),T(1))  ! Save this as B(1)
 tau1=tau(1,Ichan)

! Now compute radiances for the rest of the levels
 do level=2,N
  b2=Planck(Vnu(ichan),T(level))        ! Save this as B(level)
  TAU2 = TAU(level,ichan)
  Sum=Sum+.5*(b1+b2)*(tau1-tau2)   ! Use arrays b and tau
  b1=b2
  tau1=tau2
 EndDo
```

Remember what I said about conditionals.
You have to deal with one in this problem set.

Problem Set for Tomorrow cont: More things to watch out for:

If things don't converge, don't assume that it is in the TL code… it could be in the testing logic.

I find that at least half of the errors that I chase down are in the testing logic.

GOOD LUCK!

I am available for consultation, during this week. After the class you are on your own.

Room 810 all the way back to the left by the farthest window.
763-8136x126