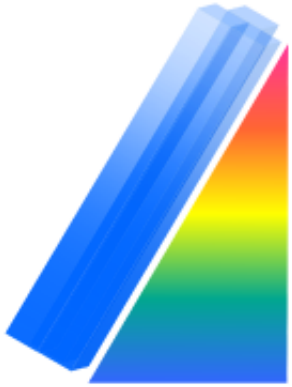


PRISM

NNSA Center for Prediction of Reliability, Integrity and Survivability of Microsystems

ME 597/AAE 590 : Introduction to Uncertainty Quantification

Lecture 2: Sensitivity Analysis

Sanjay Mathur
Purdue University

ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

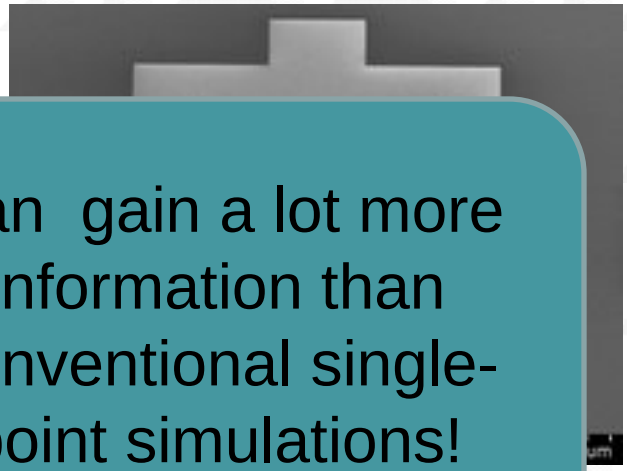
PURDUE
UNIVERSITY

THE UNIVERSITY of NEW MEXICO

Why Sensitivity and Adjoints?

What inputs yield desired outputs ?

Tangent Problem: How does a specific input affect all outputs?



Can gain a lot more information than conventional single-point simulations!

Adjoint Problem: How is a specific output affected by all inputs?

Determine effect of uncertainties and tolerances

Plan experiments better

Improve single-point simulations
- Mesh sensitivity
- Sensitivity to iteration parameters

Approaches

Jacobian $$\boldsymbol{J} = \left[\frac{\partial y_m}{\partial x_n} \right]_{mxn}$$

m outputs

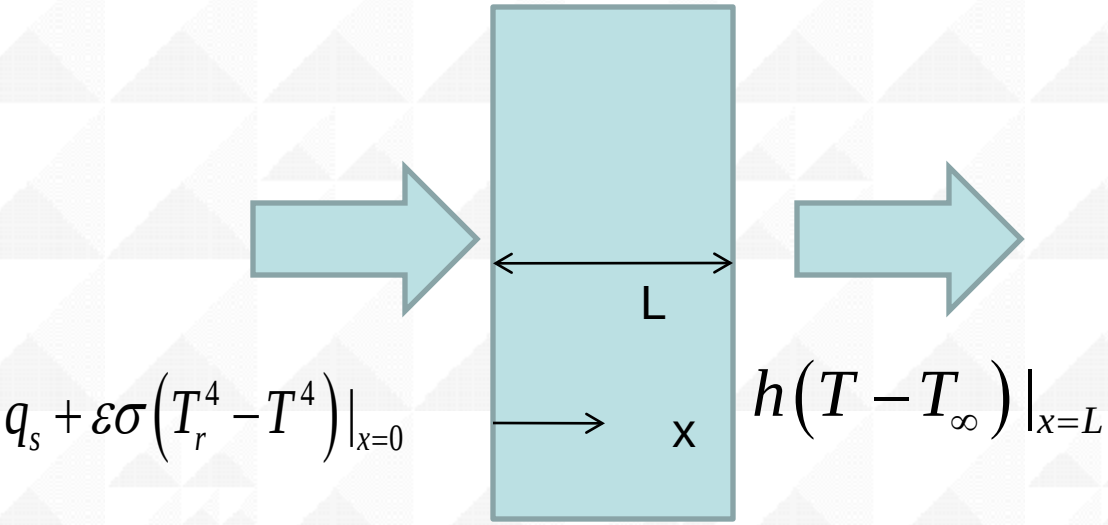
n inputs

- Continuous approach
  - Derive sensitivity/adjoint versions of pde
  - Discretize and solve
  - Need to repeat for each physical model
  - Generally more complex
- Discrete approach
  - Start with discrete pde and add sensitivity/adjoints
  - Easier to automate
  - Adjoints more difficult

Continuous Approach: Sensitivity Equation

- Governing equations are differentiated with respect to each parameter of interest
- New PDEs are derived for the sensitivity coefficients, as well as appropriate boundary conditions
- These are solved using numerical methods similar (but not identical) to the original PDE
- CPU times scale sub-linearly with the number of parameters

Example: Transient Heat Conduction in a 1D Slab



$$C\frac{\partial T}{\partial t} + \frac{\partial q}{\partial x} = 0 \qquad q = -k\frac{\partial T}{\partial x}$$

$$q|_{x=0} = -k\frac{\partial T}{\partial x}|_{x=0} = q_s + \varepsilon\sigma\left(T_r^4 - T^4\right)|_{x=0}$$

$$q|_{x=L} = -k\frac{\partial T}{\partial x}|_{x=L} = h\left(T - T_\infty\right)|_{x=L}$$

$$T(x,0) = T_i$$

Parameters: C, k, $q_s, \varepsilon, T_r, h, T_\infty, T_i$

PRISM
NNSA Center for Prediction of Reliability, Integrity and Survivability of Microsystems

Courtesy K. Dowding, Sandia

Sensitivity Equation for Heat Capacity

Differentiate energy equation wrt C:

$$\frac{\partial}{\partial C}\left[C\frac{\partial T}{\partial t}+\frac{\partial q}{\partial x}\right]=C\frac{\partial}{\partial t}\left[\frac{\partial T}{\partial C}\right]+\frac{\partial T}{\partial t}+\frac{\partial}{\partial x}\left[\frac{\partial q}{\partial C}\right]=0$$

Multiply by nominal value of C:

$$C\frac{\partial}{\partial t}\left[C\frac{\partial T}{\partial C}\right]+C\frac{\partial T}{\partial t}+\frac{\partial}{\partial x}\left[C\frac{\partial q}{\partial C}\right]=0$$

Scaled sensitivity of q wrt C:

$$q_C=C\frac{\partial q}{\partial C}=-k\frac{\partial}{\partial x}\left(C\frac{\partial T}{\partial C}\right)=-k\frac{\partial}{\partial x}(T_C)$$

PDE for new variable T_C :

$$C\frac{\partial T_C}{\partial t}-\frac{\partial}{\partial x}\left[k\frac{\partial T_C}{\partial C}\right]=-C\frac{\partial T}{\partial t}$$

PRISM
NNSA Center for Prediction of Reliability, Integrity and Survivability of Microsystems

Courtesy K. Dowding, Sandia

Boundary and Initial Conditions

At x=0

$$q_C\,|_{x=0} = C\frac{\partial q}{\partial C}\,|_{x=0} = -k\frac{\partial T_C}{\partial x}\,|_{x=0} = -4\varepsilon\sigma T^3 T_C\,|_{x=0}$$

i.e., $-k\frac{\partial T_C}{\partial x}\,|_{x=0} = -4\varepsilon\sigma T^3 T_C\,|_{x=0}$ (Linear in T_C)

At x=L

$$q_C\,|_{x=L} = C\frac{\partial q}{\partial C}\,|_{x=L} = -k\frac{\partial T_C}{\partial x}\,|_{x=L} = -hT_C\,|_{x=L}$$ (Linear in T_C)

Initial condition:

$T_C(x,0) = 0$ (Homogeneous)

Solution Procedure

- Need T to solve for T_C but not vice versa
- LHS of T_C equation identical to T equation (but not always true for all parameters)
  - Same coefficient matrix may be used
- Solution procedure would march T one time step, and then march T_C the same time step using just-computed T
- Even for a non-linear problem, the sensitivity equation is linear
  - Solution cost lower than for original PDE
- But we have to create a new sensitivity PDE for each new parameter of interest!
- Very intrusive – have to code this in, debug, parallelize etc!

Discrete Sensitivity Approaches

Jacobian $$\boldsymbol{J} = \left[\frac{\partial y_m}{\partial x_n} \right]_{mxn}$$

m outputs

n inputs

- Objective is to determine the derivative of all outputs to a single input
- Useful if number of output parameters is large (m>>n)
- Three main methods
  - Finite difference
  - Complex variables
  - Code differentiation

Finite Difference Method

$$\frac{\partial f_j}{\partial x_k} = \frac{f_j\left(x_k + \Delta x_k\right) - f_j\left(x_k\right)}{\Delta x_k} + O\left(\Delta x_k\right)$$

- No code modification – unintrusive!
- Run code N+1 times to find derivatives wrt N parameters
- Each solve is a non-linear solve if original PDE is non-linear
- Need to choose Δ carefully
  - too small -> roundoff errors
  - too large -> truncation errors
- Can yield very poor results for noisy functions
  - Most iterative solutions with finite termination criteria are noisy

Complex Variables

$$f_j\left(x_k + i\Delta x_k\right) = f_j\left(x_k\right) + i\frac{\partial f_j}{\partial x_k}\Delta x_k + O\left(i\Delta x_k\right)^2$$

- Replace all floating point variables with complex variables
- Set imaginary component to Δx_k for the variable we want derivatives with respect to
- No roundoff problems
- Still has truncation error
- Need to deal with relational operators etc.

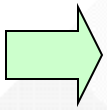
Code Differentiation

Consider code as a composition of unary and binary operations

$$z = f(u) \quad z' = \frac{\partial f}{\partial u} u'$$

$$z = f(u,v) \quad z' = \frac{\partial f}{\partial u} u' + \frac{\partial f}{\partial v} v'$$

We know how to propagate inputs to outputs for elementary operations



Apply chain rule

This process can be automated !

PRISM
NNSA Center for Prediction of Reliability, Integrity and Survivability of Microsystems

Simple Example

Original Functions:

$p = 3 * x^2 + \sin(y)$ $\quad q = p / y$

x,y: inputs p,q: outputs

Elemental Decomposition

$t_1 = x * x$

$t_2 = 3 * t_1$

$t_3 = \sin(y)$

$p = t_2 + t_3$

$q = p / y$

Elemental Derivatives

$t_1' = x * x' + x' * x$

$t_2' = 3 * t_1'$

$t_3' = \cos(y) * y'$

$p' = t_2' + t_3'$

$q' = (p' * y - y' * p) / y^2$

Simple Example (Cont'd)

Inputs:

x=10, y=π/3, x' =1, y'=0

Outputs:

$$p' = 60.0 = \left.\frac{\partial p}{\partial x}\right|_{x=10,\, y=\pi/3}$$

$$q' = -273.8795 = \left.\frac{\partial q}{\partial x}\right|_{x=10,\, y=\pi/3}$$

Derivatives are exact!

Inputs:

x=10, y=π/3, x' =0, y'=1

$$= \left.\frac{\partial p}{\partial y}\right|_{x=10,\, y=\pi/3}$$

$$q' = -273.8795 = \left.\frac{\partial q}{\partial y}\right|_{x=10,\, y=\pi/3}$$

What Did We Get?

- Exact value of derivative wrt variable whose prime was set to unity
  - Not subject to truncation as in finite difference
- Each variable and its prime must be stored
- Obtains numerical value of derivative, not symbolic
- Process works through loops, conditionals and iterations
- To evaluate derivative wrt another input, need to run code again
  - However, can exploit linearity of sensitivity calculation by turning on sensitivity only after basic single-point computation has converged
- Sensitivity is obtained at the nominal condition at which the single-point computation is done

Conventional C Implementation

Original C code

```
void myfunc (double x,  double y, double *p, double *q){

*p=3*x*x+sin(y);

*q=*p/y;  }
```

Differentiated code

```
void myfunc (double x,  double y, double *p, double *q,
double xp, double yp, double *pp, double *qp){
double t1, tp1,t2, t2p, t3, t3p;
t1 = x*x;          t1p = x*xp + xp*x;
t2 = 3*t1;         t2p = 3*t1p;
t3 = sin(y);       t3p = cos(y) * yp;
*p  =  t2+t3;      *pp = t2p + t3p;
*q = *p/y;         *qp  = ((*pp)*y – yp*(*p))/(y*y);
}
```

Conventional Code Differentiation

- For every variable we introduce a companion primed variable
- This process can be automated by tools like ADIFOR and ADIC
  - Parse FORTRAN/C code and produce new code with additional lines
- Need to deal with two versions of the code!
- Need to generate differentiated code every time original changes
- Often requires modifications of original code for special situations
- Generated code is not human readable
  - Harder to debug

Alternative: Let the Compiler Do It!

- Use C++ language features
  - User-defined data types (classes)
  - Operator overloading
  - Templates and template meta-programming
  - Static initialization
  - Shared libraries

We define a template for the whole code which can be instantiated for any well defined algebra

C++ Implementation

Original C++ code

```
void myfunc (const double& x, const double& y,
double& p, double& q){

p=3*x*x+sin(y);

q=p/y;}
```

Templated C++ code

```
Template <class T>

void myfunc (const T& x, const T& y,
T& p, T& q){

p=3*x*x+sin(y);

q=p/y;}
```

Tangent class T contains value and derivative

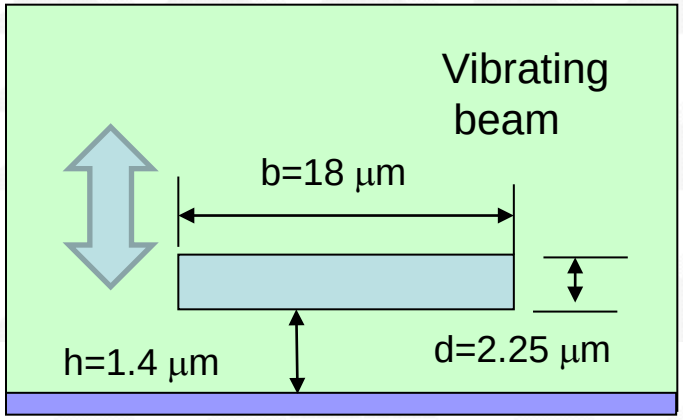
Operators are overloaded to operate on both

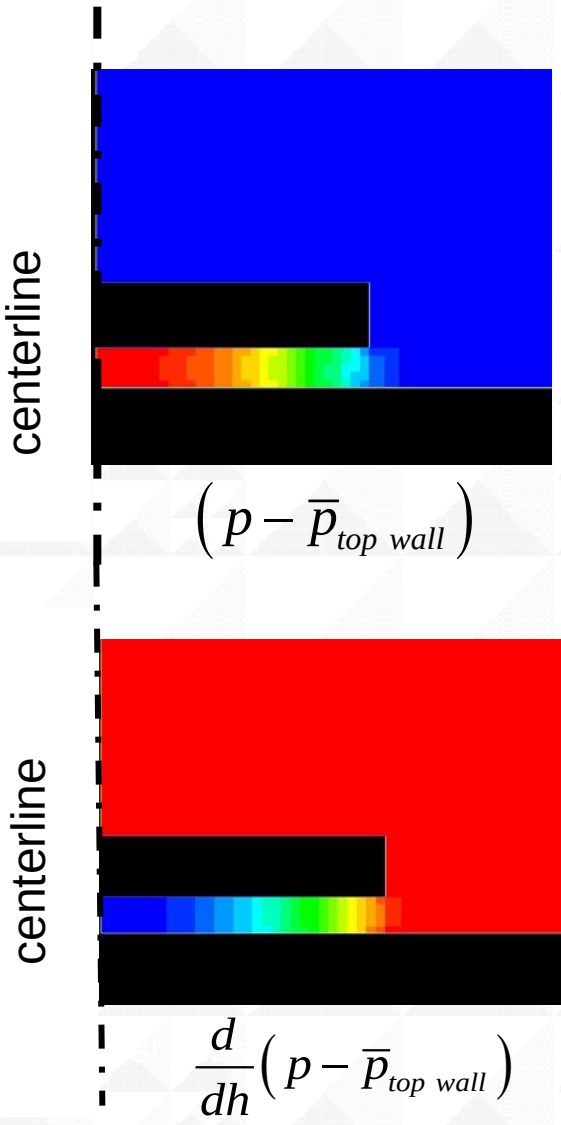
PRISM
NNSA Center for Prediction of Reliability, Integrity and Survivability of Microsystems

Advantages of C++ Approach

- Does not rely on external parser like ADIFOR
- One source code to maintain and debug
  - Instantiate sensitivity version when required
- No penalty for normal usage
- Can switch from double to Tangent mode as desired
  - Very useful for non-linear problems

Sensitivity Analysis of Squeeze Film Damping

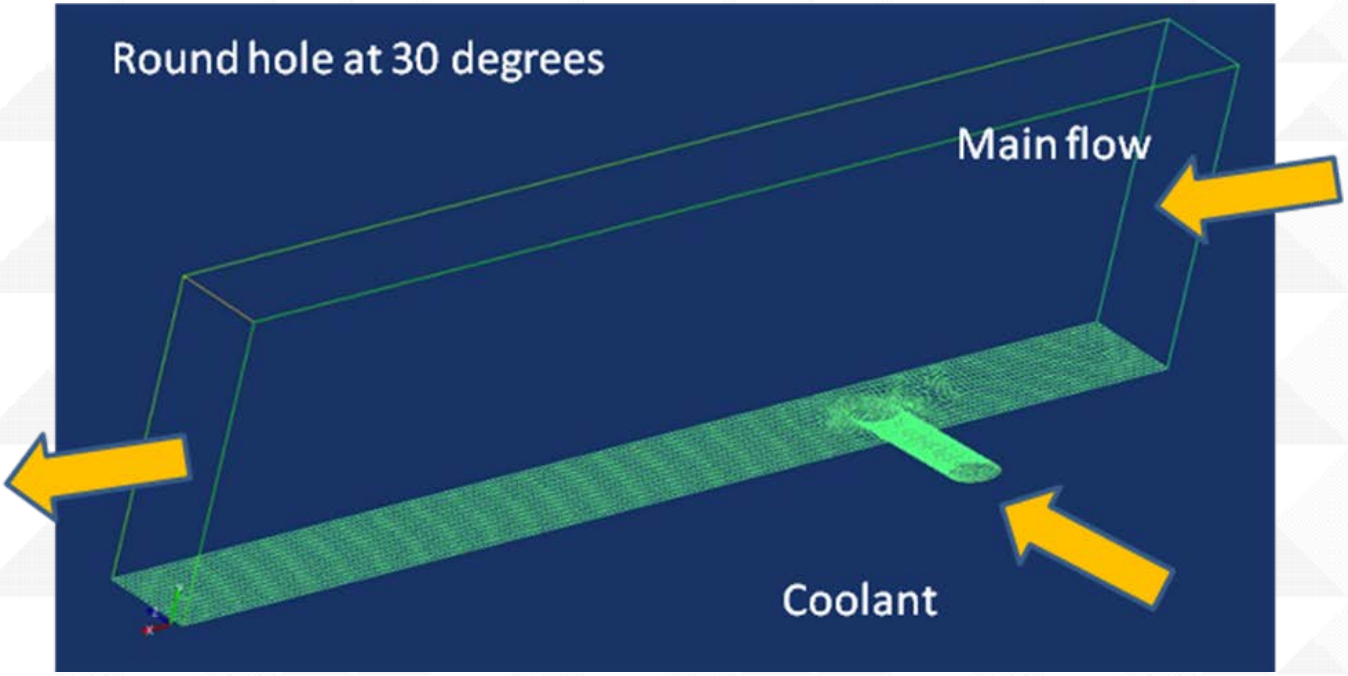




- $\Delta p=\left(p-\overline{p}_{top\ wall}\right)$ greatest at center and decreases outwards
- d (Δp)/dh most negative at center and increases to zero outward
- As h decreases, Δp increases;
  - Vertical force F on cantilever increases
  - dF/dh = -4460 N/m at h = 1.4 microns.

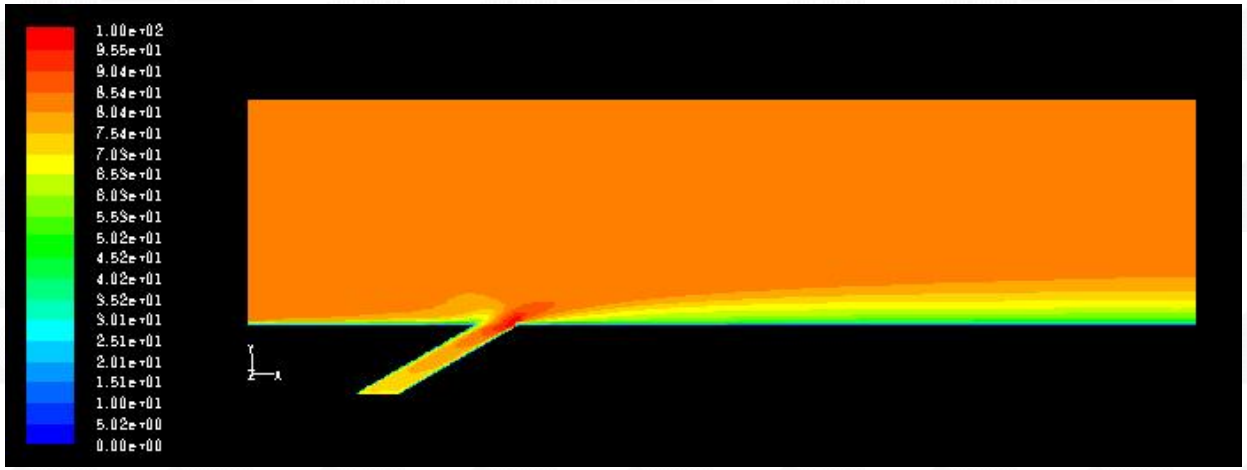
PRISM
NNSA Center for Prediction of Reliability, Integrity and Survivability of Microsystems

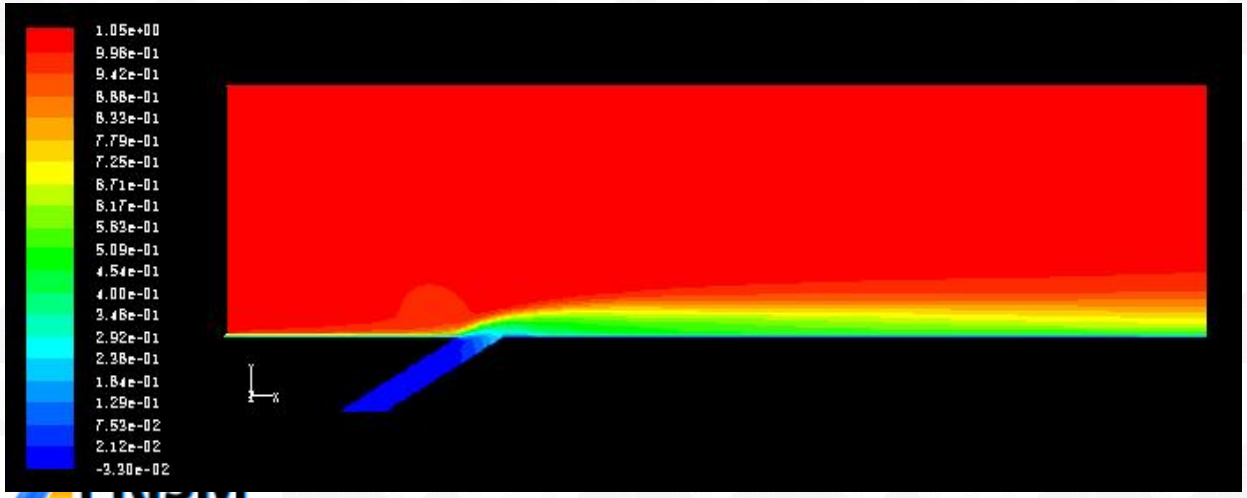
Sensitivity Analysis of Film Cooling



Compute sensitivity of flow and temperature field and area-averaged effectiveness to hot gas inlet velocity and temperature

Sensitivity of U-Velocity to Hot Gas Inlet Velocity



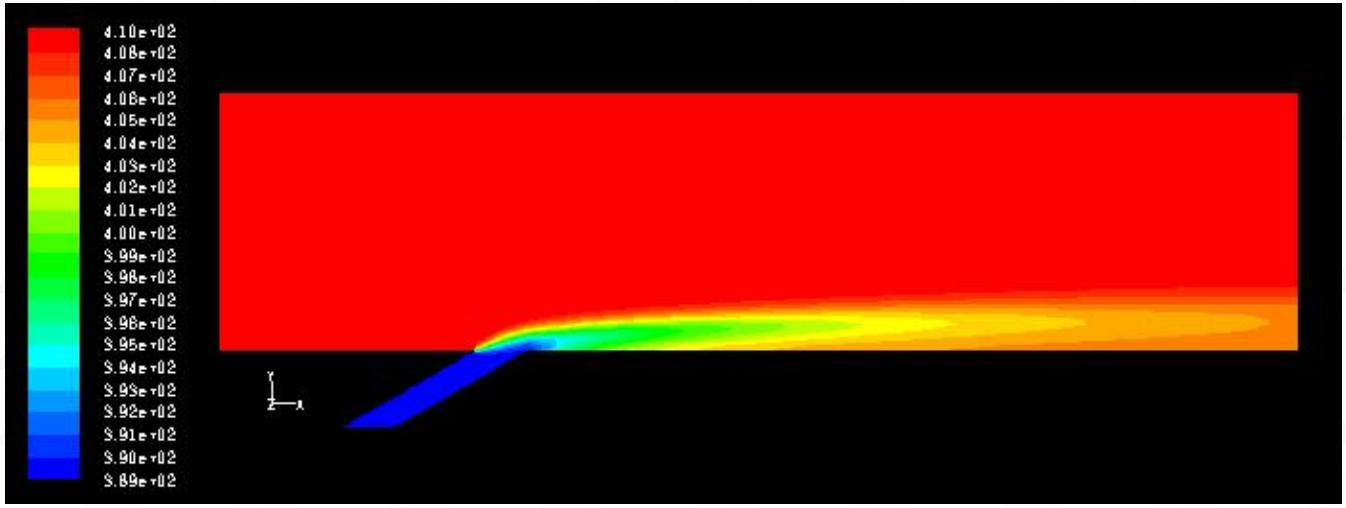


M=1.0
T ratio=1.05

u

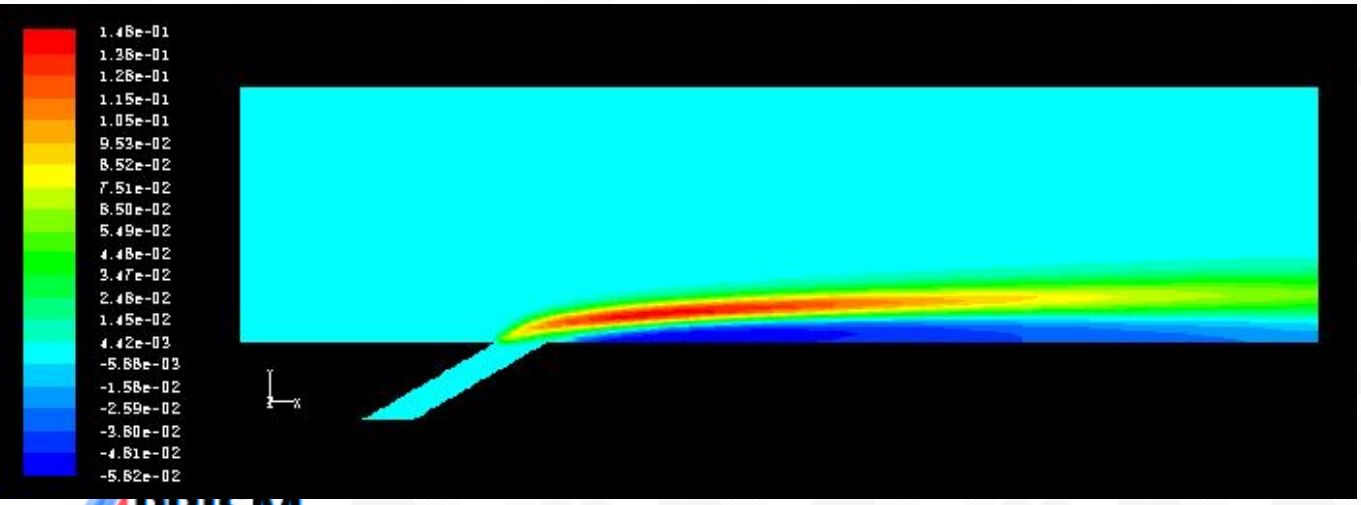
$$\frac{\partial u}{\partial U_{main}}$$

Sensitivity of Temperature to Hot Gas Inlet Velocity



M=1.0
T ratio=1.05

T



$$\frac{\partial T}{\partial U_{main}}$$

Summary

- Our objective is to find the local Jacobian (sensitivity) of outputs to inputs
- We described two ways of doing this
  - Continuous - sensitivity equation
  - Discrete – automatic code differentiation
- Sensitivities are useful in their own right, but have many other uses
  - Optimization, variance propagation
- In the next lecture, we will describe how these derivatives can be used to propagate variances from inputs to outputs