Sensitivity Analysis in Computational Mechanics
Sanjay R. Mathur, Purdue University

What inputs yield the desired outputs?

**Tangent problem:** How does a specific input affect all outputs?

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

**Jacobian Computation**

\[
J = \begin{bmatrix}
\frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_n} \\
\vdots & \ddots & \vdots \\
\frac{\partial y_m}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_n}
\end{bmatrix}
\]

- **Continuous approach**
  - Derive sensitivity/adjoint pde; discretize and solve
  - Need to repeat for each pde – more complex

- **Discrete approach**
  - Start with discretized pde and add sensitivities and adjoints
  - Implement through automatic code differentiation
  - Easy to extend to new models

**Inputs**

\[x = 10, y = \pi/3, x' = 1, y' = 0\]

**Outputs**

\[p = 10, q = \pi/3\]

\[60.0 \quad 57.2958\]

\[p = x^2 + \sin(y), \quad q = p/y\]

**Inputs**

\[x = 10, y = \pi/3, x' = 0, y' = 1\]

**Outputs**

\[p' = 60.0 \quad q' = 57.2958\]

\[-273.88\]

\[p = 3x^2 + \sin(y); \quad q = p/y\]

**What did we get?**

- Exact derivative with respect to variable with unity prime
- Numerical, not symbolic value
- Works through loops and conditionals
- Must store value and derivative

**Code differentiation**

Decompose code into elementary unary and binary operations; propagate inputs to outputs

**Original Functions**

\[p = 3x^2 + \sin(y), \quad q = p/y\]

**Elementary Decomposition**

\[t_1 = x^2, \quad t_2 = 3t_1, \quad t_3 = \sin(y)\]

\[p = t_2 + t_3, \quad q = p/y\]

**Original C++ code**

```cpp
void myfunc(const double& x, const double& y, double& p, double& q) {
    p = 3 * x * x + sin(y);
    q = p / y;
}
```

**Templated C++ code**

```cpp
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 values and derivatives.

**Let the compiler do it!**

Exploit C++ features:

- **Tangent** class through user-defined data types
- Operator overloading
- Templates and template meta-programming

**Verification of sensitivity calculation**

Compute \( F \) for two different \( h \) values, \( F_1(h_1) \) and \( F_2(h_2+\Delta h) \). Verify that:

\[
F_2 - \left( F_1 + 0.5 \left( \frac{dF_1}{dh} \right) \Delta h \right) \text{is small.}
\]

We find a value of 2.678x10^-6 (0.096% error) for \( \Delta h/h_1 = 0.071 \) at \( h_1 = 1.4 \mu m \)

**Sensitivity to film height \( h \)**

- \( \Delta p = (p - p_{top\ wall}) \) greatest at center and decreases outwards
- \( d\Delta p/dh \) most negative at center and increases to zero outward, i.e., center region is most sensitive to \( h \).
- As \( h \) decreases, \( \Delta p \) increases; vertical force \( F \) on cantilever increases; \( dF/dh = -4460 \text{ N/m} \)

**Sensitivity of velocity field to viscosity changes**

The greatest sensitivity is at the cantilever edge. The vertical fluid velocity is the most sensitive to viscosity changes and increases with an increase in viscosity. We find \( dV/d\mu = 127.557 \text{ N m/s/kg} \).