Reliability and Sensitivity Analysis in OpenSees

Michael H. Scott
Associate Professor
Civil and Construction Engineering
Oregon State University

Oregon State University

Presentation Outline

- Changes in Reliability Modules
- Parameter Objects in OpenSeees
- Sample Tcl Commands and Utilities
- Stand Alone Sensitivity Analysis
- Random Variables
- Random Variable Positioners
- Monte Carlo Analysis
- Performance Functions
- First Order Reliability (FORM)

Oregon State University

Changes in Reliability Modules

- Reliability modules of OpenSees have been undergoing extensive changes over the last three years
  - More flexible and extensible approach to "parameterizing" an FE model
  - Give user more control of input/output by taking advantage of native Tcl commands and introducing more low-level OpenSees/Tcl commands
  - Do away with sequential tagging and fixed-format output files
- Most modules "broken" by this re-engineering process will be fixed in the coming year

Parameter Objects

- Compute sensitivity with respect to, and map probability distributions to, uncertain properties (parameters) of an FE model
- Script-level mechanism for identifying and updating parameters
- Enable all methods of uncertainty modeling in OpenSees
  - FORM, SORM, etc.
  - Response Sensitivity
  - Importance Sampling
  - Optimization

The Parameter Command

- With OpenSees model defined, use the `parameter` command to identify uncertain element, material, load, and node properties

```
parameter $tag <specific object arguments>
```

- Just like objects of the FE domain
  - Each parameter has a unique tag
  - Parameter tags do not have to be sequential or start at one

Section/Material Parameters of Nonlinear Frame Elements

- Section/material model specific keywords, e.g., fy
  - See the *setParameter()* method of your model's source code
- Map parameter 5 to yield strength for all sections of element 3

```
parameter 5 element 3 fy
```

- Map parameter 18 to yield strength of only section 2 (end I to J: 1,…,Np) of frame element 6

```
parameter 18 element 6 section 2 fy
```

- Map parameter 9 to yield strength of only the fibers with material tag equal to 5 for all sections of element 23

```
parameter 9 element 23 material 5 fy
```

Load Parameters

- Nodal and member loads contained in load patterns

- Map parameter 14 to horizontal load at node 12 contained in load pattern 3

```
parameter 14 loadPattern 3 loadAtNode 12 1
```

Last argument: Global DOF, e.g., 2D (1=PX, 2=PY, 3=MZ)

- Map parameter 28 to uniform member load on frame element 13 in load pattern 5

```
parameter 28 loadPattern 5 elementLoad 13 wy
```

Last argument: uniform: wy wx; pointLoad: Px Py xOverL
in local coordinate system of the element

Node Parameters

- Nodal coordinates defined using the `node` command
- Map parameter 43 to X-coordinate of node 5

```
parameter 43 node 6 coord 1
```

Last argument: Global direction (1=X, 2=Y, 3=Z)

- Nodal mass defined using the `mass` command
- Map parameter 34 to X direction mass of node 12

```
parameter 34 node 12 mass 1
```

Last argument: Global DOF (1, 2, etc.); can also use XY and XYZ for all translational DOFs in one command, e.g.,

```
parameter 34 node 12 mass XY
```

Adding More Objects to a Parameter

- One-to-many relationship between a parameter object and objects of an FE model
- Invoke the `addToParameter` command on an existing parameter

```
addToParameter $tag <specific object arguments>
```

- Specific object arguments follow same syntax shown on previous slides
- Example: Have parameter 5 map to fy of element 4 (in addition to fy of element 3 a few slides back)

```
addToParameter 5 element 4 fy
```


Updating Parameter Values

- Update parameter values during a probabilistic analysis
- Invoke the `updateParameter` command on a parameter

```
updateParameter $tag $newValue
```

- Example: Set parameter 5 to 61.5

```
updateParameter 5 61.5
```

- Can also remove parameters from an analysis

```
removeParameter 5
```

Query Parameter Value

- Tcl command to obtain the current value of a parameter

```
[getParamValue $tag]
```

- Use a parameter as an r-value of a Tcl expression [be sure to use square brackets!]
  - Put to standard output stream

```
puts [getParamValue 5]
```

  - Put to file output stream

```
puts $filehandle [getParamValue 5]
```

  - Store in a Tcl variable

```
set x [getParamValue 5]
```

- Note: this command returns 0.0 if invoked before `updateParameter` (will be fixed)

Tcl List of Parameter Tags

- Tcl command to obtain a list of all parameter tags

```
[getParamTags]
```

- Iterate over all parameters with a `foreach` loop

```
foreach p [getParamTags] {
   puts      [getParamValue $p]
}
```

- Returns an unordered list, so use Tcl function to sort

```
foreach p [lsort –integer [getParamTags]] {
   puts      [getParamValue $p]
}
```

- Note: list length is total number of parameters

```
set Nparam [llength [getParamTags]]
```

Stand Alone Sensitivity Analysis

- Compute "raw" FE response sensitivity (gradients) with respect to parameters
  - Independent of gradient-based application
  - Direct differentiation method (DDM) calculations invoked
  - Useful for assessing change in response to design parameters

- With parameters mapped to FE model objects, issue the following commands:

```
reliability
sensitivityIntegrator –static  ;# or –transient
sensitivityAlgorithm –computeByCommand -parameters
```


Computing the Gradients

- Issue the `computeGradients` command
- After a specified number of analysis steps

```
analyze 20
computeGradients
```

- Inside an analysis loop

```
for {set i 1} {$i <= 20} {incr i} {
    analyze 1
    computeGradients
}
```

Nodal Response Sensitivity

- Issue the `sensNodeDisp` command

  `[sensNodeDisp $node $dof $param]`
  - Analogous to `nodeDisp` command with node tag and dof
  - Last argument is parameter tag
- Inside an analysis loop, for example, output sensitivity of node 2 horizontal displacement wrt all parameters

```
for {set i 1} {$i <= 20} {incr i} {
    analyze 1
    computeGradients
    foreach p [lsort –integer [getParamTags]] {
          puts “Parameter $p: [sensNodeDisp 2 1 $p]”
    }
}
```


Section Response Sensitivity

- Issue the `sensSectionForce` command

  `[sensSectionForce $ele $sec $dof $param]`
  - Analogous to `sectionForce` command with element tag, section number, and section dof (depends on section model)
  - Last argument is parameter tag
- Inside an analysis loop, for example, output sensitivity of bending moment (dof 2 for fiber sections) at section 1 of element 3 wrt all parameters

```
for {set i 1} {$i <= 20} {incr i} {
  analyze 1
  computeGradients
  foreach p [lsort –integer [getParamTags]] {
    puts “Parameter $p: [sensSectionForce 3 1 2 $p]”
  }
}
```

Random Variable Definition

- Associate parameters with random variables (probability distribution functions) in order to perform probabilistic finite element analysis

```
randomVariable $tag type <arguments>
```

- Similar to parameters and FE objects
  - Each RV has a unique integer tag
  - RV tags do not have to be sequential or start at one
- RV types: normal, lognormal, poisson, etc., and about ten others available
- Trailing command arguments are distribution parameters

Random Variable Utilities

- Probability density function, f(x)

```
[getPDF $tag $x]
```

- Cumulative density function, F(x)

```
[getCDF $tag $x]
```

- Inverse CDF, F^-1(p)

```
[getInverseCDF $tag $p]
```

- List of all RV tags

```
[getRVtags]
```

- Remove a RV

```
remove randomVariable $tag
```

Random Variable Positioners

- Map each random variable to a parameter

```
randomVariablePositioner $tag –rvTag $rvTag –parameter $pTag
```

- The RVP tags are inconsequential, so use a Tcl array and loop to construct the positioners automatically after RV and parameter definitions

```
    parameter 12 element 1 . . .
    parameter 4 loadPattern 2 . . .
    randomVariable 7 normal . . .; set param(7) 12
    randomVariable 15 lognormal . . .; set param(15) 4
    foreach rvTag [array names param] {
      randomVariablePositioner $rvTag –rvTag $rvTag \
           –parameter $param($rvTag)
}
```

- May seem awkward, but it beats typing out each RVP command

RVP Utilities

- Remove a random variable positioner

```
remove randomVariablePositioner $tag
```

- Return a list of (rvTag,pTag) pairs for all positioners

```
[getRVPositioners]
```

  – Construct performance function gradients (later)
  – Monte Carlo analysis (next slide)

Monte Carlo Analysis

- Use the positioner pairs list, Tcl's random number generator, and inverse CDF functions

```
set Ntrials 10000
set Nfail 0
expr srand([clock clicks])
for {set i 1} {$i <= $Ntrials} {incr i} {
   reset
   foreach {rvTag pTag} [getRVPositioners] {
     set p [expr rand()]
     set x [getInverseCDF $rvTag $p]
     updateParameter $pTag $x
   }
   analyze
   if {fail} {incr Nfail}
}
puts "pf = [expr double($Nfail)/$Ntrials]"
```

- Avoid integer division in the probability calculation!

Limit State Functions

- Also known as performance functions or g-functions, limit state functions define "failure" of a system
- Defined in terms of nodal and element response
  - Exceeding a displacement limit
  - Exceeding an internal force limit, etc.
- For most PBEE applications, usually a simple algebraic expression (capacity minus demand)

$$
\begin{aligned}
g_1(\mathbf{x}) &= 0.15 - u \\
g_2(\mathbf{x}) &= 1500.0 - M
\end{aligned}
$$

Limit State Function Definition

- Use a Tcl-expression for the limit state function

```
performanceFunction $tag "Tcl expression"
```

  - Arbitrary, non-sequential tag
  - Expression contains regular Tcl syntax and commands

```
performanceFunction 76 "0.15 - \[nodeDisp 2 1\]"
performanceFunction 23 "1500 - \[sectionForce 2 1 2\]"
```

- The "slash bracket" syntax \[\] defers evaluation of a Tcl command to the times of function evaluation during a probabilistic analysis rather than the one time of function definition in the Tcl script
- The `u_2_1` and `rec_element_2_section_1_dof_2` limit state function syntax is deprecated

Limit State Function Gradients

- Gradient of each LSF is required during search for design point $\frac{\partial g_i}{\partial x_j} = \frac{\partial g_i}{\partial \mathbf{U}_f}\frac{\partial \mathbf{U}_f}{\partial x_j} + \frac{\partial g_i}{\partial \mathbf{P}}\frac{\partial \mathbf{P}}{\partial x_j} + \ldots$
- $\partial \mathbf{U}_f / \partial x_j$ from gradient of FE response
  - Finite difference (inaccurate) -or-
  - Direct differentiation (preferred)
- $\partial g_i / \partial \mathbf{U}_f$ from derivative of LSF expression
  - Finite differences (inaccurate) – ONLY OPTION
  - Automatic differentiation (difficult)
  - Analytic expressions (user defined, exact) – NEW OPTION
- Identical issues for derivatives with respect to internal forces in $\frac{\partial g_i}{\partial \mathbf{P}}\frac{\partial \mathbf{P}}{\partial x_j}$

Analytic LSF Gradients

- User-defined Tcl expressions for analytic gradient, $\partial g_i / \partial x_j$, of each limit state function wrt random variables

```
gradPerformanceFunction $lsfTag $rvTag "Tcl expression"
```

- Define using (rvTag,pTag) pairs in conjunction with the `sensNodeDisp` and `sensSectionForce` commands

```
foreach {rvTag pTag} [getRVPositioners] {
    gradPerformanceFunction 76 $rvTag \
          "-\[sensNodeDisp 2 1 $pTag\]"
    gradPerformanceFunction 23 $rvTag \
          "-\[sensSectionForce 2 1 2 $pTag\]"
}
```

- Trivial for linear LSFs, but more efficient than finite diffs

FORM Analysis Results

- Previously, the `runFORMAnalysis` command piped all analysis results to a fixed-format text file
  - Reliability index
  - RV realizations at the design point
  - Importance vectors

```
##########################################################################
#  FORM ANALYSIS RESULTS, LIMIT-STATE FUNCTION NUMBER 1                  #
#                                                                        #
#  Limit-state function value at start point: ......... 11.964           #
#  Limit-state function value at end point: ........... 7.953e-08        #
#  Number of steps: ................................... 4                #
#  Number of g-function evaluations: .................. 8                #
#  Reliability index beta: ............................ 1.1222           #
#  FO approx. probability of failure, pf1: ............ 0.13089          #
#                                                                        #
# rv#     x*           u*          alpha    gamma     delta     eta      #
#  1    2.093e+05  -3.874e-02  -0.03453 -0.03453     -         -         #
#  2    3.411e+02  -7.734e-01  -0.68921 -0.68921     -         -         #
#  3    3.510e+05   8.122e-01   0.72374  0.72374     -         -         #
#                                                                        #
##########################################################################
```

FORM Analysis Tcl Commands

- Recent addition of Tcl commands to obtain FORM analysis results
  - Reliability index

    ```
    [betaFORM $lsfTag]
    ```
  - Importance vectors

    ```
    [alphaFORM $lsfTag $rvTag]
    [gammaFORM $lsfTag $rvTag]
    ```
- Commands coming soon:
  - delta and eta vectors (importance of mean and st.dev.)
  - Design point realizations for each LSF

Oregon State University

User Control of FORM Results

- Open an output file in Tcl or print to screen
- Iterate over the limit state functions

```
foreach lsf [getLSFTags] {
    puts “Limit State Function $lsf“
    puts "beta = [betaFORM $lsf]“
    foreach rv [getRVTags] {
      puts " alpha($rv) = [alphaFORM $lsf $rv], \
          gamma($rv) = [gammaFORM $lsf $rv]“
    }
}
```

```
Performance Function 23
beta = 2.2094816
          alpha(32) = 0.9814942, gamma(32) = 0.9814942
          alpha(41) = 0.1593380, gamma(41) = 0.1593380
          alpha(62) = 0.1062089, gamma(62) = 0.1062089
          alpha(89) = 0.0003440, gamma(89) = 0.0003440
Performance Function 76
beta = 2.4495696
          alpha(32) = 0.9011882, gamma(32) = 0.9011882
          alpha(41) = -0.0054455, gamma(41) = -0.0054455
          alpha(62) = -0.4333630, gamma(62) = -0.4333630
          alpha(89) = 0.0051653, gamma(89) = 0.0051653
```

Oregon State University

Concluding Remarks

- Work continues to make uncertainty modeling in OpenSees more extensible to wider range of problems
  - Bridge engineering
  - Fluid-structure interaction
- Many changes to underlying software architecture on the way, but should be transparent to users
- Documentation and updated reliability examples on OpenSees website

Questions, comments?

michael.scott@oregonstate.edu