Introduction to Analysis Commands

Frank McKenna

OpenSees Users Workshop
Berkeley, CA
August 14, 2006

Sponsored by the National Science Foundation
through the Pacific Earthquake Engineering Research Center
and through NEESit
Main Abstractions in OpenSees

ModelBuilder
- Constructs the objects in the model and adds them to the domain.

Recorder
- Monitors user defined parameters in the model during the analysis

Domain
- Holds the state of the model at time t and \((t + dt)\)

Analysis
- Moves the model from state at time t to state at time \(t + dt\)

In this presentation we focus on ANALYSIS GENERATION
Analysis

**Solver**

- numberer type? args…
- **algorithm type? args…**
- integrator type? args…
- system type? args…
- analysis type? args…
- analyze args …
analysis command:

- Static Analysis
- Transient Analysis

- both incremental solution strategies

```
for (int i=0; i<numIncr; i++) {
  theIntegrator->newStep();
  theAlgorithm->solveCurrentStep();
  theModel->commit();
}
```

```
for (int i=0; i<numIncr; i++) {
  theIntegrator->newStep(dt);
  theAlgorithm->solveCurrentStep();
  theModel->commit();
}
```

- Eigenvalue
  - general eigenvalue problem
    
    \[(K-\lambda M)\Phi=0\]

  - standard eigenvalue problem
    
    \[(K-\lambda)\Phi=0\]
integrator command:
- determines the predictive step for time t+\(\delta t\)
- specifies the tangent matrix and residual vector at any iteration
- determines the corrective step based on \(\Delta U\)

• Transient Integrator for Use in Transient Analysis

Nonlinear equation of the form:

\[ R(U, \dot{U}, \ddot{U}) = P(t) - F_i(\ddot{U}) - F_R(U, \dot{U}) \]

- Newmark Method

\[ integrator \ Newmark \ \gamma \beta \]

- Hilbert-Hughes-Taylor Method

\[ integrator \ Newmark \ \alpha \]
• Static Integrators for Use in Static Analysis

Nonlinear equation of the form:
\[ R(U, \lambda) = \lambda P^* - FR(U) \]

- **Load Control**
  \[ \lambda_n = \lambda_{n-1} + \Delta\lambda \]
  *does not require a reference load, i.e. loads in load patterns with Linear series and all other loads constant.*

- **Displacement Control**
  \[ U_{jn} = U_{j\ n-1} + \Delta U_j \]

- **Arc Length**
  \[ \Delta U_n^\Delta U_n + \alpha^2 \Delta\lambda_n = \Delta s^2 \]

- **Minimum Unbalance Displacement Norm**
  \[ \frac{d}{d\Delta\lambda} (\Delta U_n^\Delta U_n) = 0 \]
algorithm command:
- to specify the steps taken to solve the nonlinear equation

• Linear Algorithm

```cpp
theIntegrator->formUnbalance();
theIntegrator->formTangent();
theSOE->solve();
theIntegrator->update(theSOE->getX());
```

algorithm Linear

• Newton-Raphson Algorithm

```cpp
def main():
    theIntegrator->formUnbalance();
    do {
        theIntegrator->formTangent();
        theSOE->solve();
        theIntegrator->update(theSOE->getX());
    } while (theTest->test() == fail)
```

algorithm Newton

• Modified Newton Algorithm

algorithm ModifiedNewton <-initial>

• Accelerated Modified Newton Algorithm

algorithm KrylovNewton <-initial>
**constraints command:**
- to specify how the constraints are enforced

\[
U_c = C_{rc} U_r \\
C U = 0 \\
T U_r = [U_r \ U_c]^\wedge
\]

\[
[C_r \ C_c]^\wedge [U_r \ U_c] = 0
\]

- **Transformation Handler**

\[
K^* U_r = R^* \\
K^* = T^KT \\
R^* = T^R
\]

in OpenSees currently don’t allow retained node in one constraint to be a constrained node in another constraint

- **Lagrange Handler**

\[
\begin{bmatrix} K & C^\wedge \end{bmatrix} \begin{bmatrix} U \\ \lambda \end{bmatrix} = \begin{bmatrix} R \end{bmatrix}
\]

- **Penalty Handler**

\[
[K + C^\wedge \alpha C] U = [R + C^\wedge \alpha Q]
\]
system command:
- to specify how matrix equation $KU = R$ is stored and solved

• Profile Symmetric Positive Definite (SPD)
  
  system ProfileSPD

• Banded Symmetric Positive Definite
  
  system BandSPD

• Sparse Symmetric Positive Definite
  
  system SparseSPD

• Banded General
  
  system BandGeneral

• Sparse Symmetric
  
  system SparseGeneral

  system Umfpack
**numberer command:**
- to specify how the degrees of freedom are numbered

- **Plain Numberer**
  nodes are assigned dof arbitrarily
  - `numberer Plain`

- **Plain Numberer**
  nodes are assigned dof using the Reverse Cuthill-McKee algorithm
  - `numberer RCM`
test command:
- to specify when convergence has been achieved
  all look at system: $KU = R$

- **Norm Unbalance**
  $$\sqrt{R^{R}} < tol$$
  test NormUnbalance tol? numIter? <flag?>

- **Norm Displacement Increment**
  $$\sqrt{U^{U}} < tol$$
  test NormDispIncr tol? numIter? <flag?>

- **Norm Energy Increment**
  $$\frac{1}{2} (U^{R}) < tol$$
  test NormEnergyIncr tol? numIter? <flag?>

- **Relative Tests**
  test RelativeNormUnbalance tol? numIter? <flag?>
  test RelativeNormDispIncr tol? numIter? <flag?>
  test RelativeNormEnergyIncr tol? numIter? <flag?>
analyze command:
- to perform the static/transient analysis

•Static Analysis

```cpp
for (int i=0; i<numIncr; i++) {
    theIntegrator->newStep();
    theAlgorithm->solveCurrentStep();
    theModel->commit();
}
```

•Transient Analysis

```cpp
for (int i=0; i<numIncr; i++) {
    theIntegrator->newStep(dt);
    theAlgorithm->solveCurrentStep();
    theModel->commit();
}
```
Example Analysis:

• Static Nonlinear Analysis with LoadControl
  
  constraints transformation
  numberer RCM
  system BandGeneral
  test NormDispIncr 1.0e-6 6 2
  algorithm Newton
  integrator LoadControl 0.1
  analysis Static
  analyze 10

• Transient Nonlinear Analysis with Newmark
  
  constraints transformation
  numberer RCM
  system BandGeneral
  test NormDispIncr 1.0e-6 6 2
  algorithm Newton
  integrator Newmark 0.5 0.25
  analysis Transient
  analyze 2000 0.01