OpenSees: Analysis

Frank McKenna

NEES Hybrid Simulation Workshop
Berkeley, CA
April 24-25, 2006
Main Abstractions in OpenSees

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

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

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

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

In this presentation we focus on the ANALYSIS
Analysis Classes:

- to update the state of the Domain

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();
}
Algorithm Classes:
- to specify the steps taken to solve the nonlinear equation

```
theIntegrator->formUnbalance();
theIntegrator->formTangent();
theSOE->solve();
theIntegrator->update(theSOE->getX());
theIntegrator->formUnbalance();
```
Integrator Classes:

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

- Static Integrators for Use in Static Analysis

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

  - Load Control \( \lambda_n = \lambda_{n-1} + \Delta \lambda \)
  - Displacement Control \( U_j^n = U_j^{n-1} + \Delta U_j \)
  - Arc Length \( \Delta U_n^\Delta U_n + \alpha^2 \Delta \lambda_n = \Delta s^2 \)
\[ \lambda = \text{theModel->getTime();} \]
\[ \lambda += \delta \lambda. \]
\[ \text{theModel->applyLoad(\lambda);} \]

\[ U = \text{Ut;} \]
\[ V = a1*Vt + a2 * At \]
\[ A = a3*Vt + a4 * At \]
\[ \text{time} = \text{theModel->getTime();} \]
\[ \text{time} += \text{dT;} \]
\[ \text{theModel->setResponse(U,V,A);} \]
\[ \text{theModel->update(time);} \]

\[ U += \text{dU} \]
\[ V += c1 * \text{dU} \]
\[ A += c2 * \text{dU} \]
\[ \text{theModel->setResponse(U,V,A);} \]
\[ U_c = C_{rc} U_r \]
\[ T U_r = [U_r \ U_c]^\wedge \]

**ConstraintHandler**

- to specify how the constraints are enforced

\[ C U = 0 \]

\[ T U_r = [U_r \ U_c]^\wedge \]

**Transformation**
- handle()

**Lagrange**
- handle()

**Penalty**
- handle()

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

\[ \begin{bmatrix} K & C^\wedge \\ C & 0 \end{bmatrix} \begin{bmatrix} U \\ \lambda \end{bmatrix} = \begin{bmatrix} R \\ Q \end{bmatrix} \]

\[ [K + C^\wedge \alpha C] U = [R + C^\wedge \alpha Q] \]
SystemofEqn and Solver Classes:

- the SystemOfEqn classes store the matrix equations
- The Solver classes work on the SystemOfEqn classes to solve the eqn.

**SystemOfEqn**

**Solver**

**LinearSOE**
- addA()
- addB()
- zeroA
- zeroB
- solve()
- getX()
- getB()

**EigenSOE**
- addA()
- addM()
- zeroA
- zeroM
- solve()
- getEigenvalue()
- getEigenVector()

**LinearSolver**
- solve()

**EigenSolver**
- solve()
**Numberer command:**
- to specify how the degrees of freedom are numbered

```
for each DOF_Group
  if not constrained
    assign next dof numbers
  else
    for those dof not constrained
      assign next dof numbers
  for each DOF_Group
    if constrained in MP_Constraint
      for each dof constrained in MP find retained dof number and assign.
```
ConvergenceTest Classes:
- to determine if convergence has been achieved

```cpp
const Vector &x = theSOE->getX();
double norm = b.norm();
if (norm < tol)
  return 0;
if (count < maxCount) {
  count++;
  return -1;
} else
  return -2;
```

```cpp
const Vector &b = theSOE->getB();
double norm = b.norm();
if (norm < tol)
  return 0;
if (count < maxCount) {
  count++;
  return -1;
} else
  return -2;
```

```cpp
const Vector &b = theSOE->getB();
const Vector &x = theSOE->getX();
double norm = b^x;
if (norm < tol)
  return 0;
if (count < maxCount) {
  count++;
  return -1;
} else
  return -2;
```
Analysis

- **Constraints**: type? args...
- **Numberer**: type? args...
- **Algorithm**: type? args...
- **Integrator**: type? args...
- **System**: type? args...
- **Analysis**: type? args...
- **Analyze**: args ...

*OpenSees*
analysis command:

• Static Analysis
  - Static Analysis
• Transient Analysis
  - Transient Analysis
  - both incremental solution strategies

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

TransientAnalysis
  analyze()
  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 HHT \alpha
  ```

- CentralDifference
  
  ```
  integrator CentralDifference
  ```

- Alpha Operator Splitting Method
  
  ```
  Integrator AlphaOS \alpha \beta \gamma
  ```
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**
  \[ U_j n = U_j n-1 + \Delta U_j \]

- **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());
```

• Newton-Raphson Algorithm

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

• Modified Newton Algorithm

```cpp
algorithm ModifiedNewton <-initial>
```

• Accelerated Modified Newton Algorithm

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

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

• Transformation Handler

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

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

• Lagrange Handler

\[
\begin{bmatrix}
K & C^\wedge \\
C & 0
\end{bmatrix}
\begin{bmatrix}
U \\
\lambda
\end{bmatrix} =
\begin{bmatrix}
R \\
Q
\end{bmatrix}
\]

• Penalty Handler

\[
[ K + C^\wedge \alpha C] \quad U = [ R + C^\wedge \alpha Q]
\]
**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

- Plain Numberer
  - nodes are assigned dof using the Reverse Cuthill-McKee algorithm

```plaintext
numberer Plain
```

```plaintext
numberer RCM
```
**test command:**
- to specify when convergence has been achieved

all look at system: \( \mathbf{KU} = \mathbf{R} \)

- **Norm Unbalance**
  \[ \sqrt{\mathbf{R}^\mathbf{R}} < \text{tol} \]
  \[ \text{test NormUnbalance \ tol? numIter? <flag?>} \]

- **Norm Displacement Increment**
  \[ \sqrt{\mathbf{U}^\mathbf{U}} < \text{tol} \]
  \[ \text{test NormDispIncr \ tol? numIter? <flag?>} \]

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

- **Relative Tests**

  \[ \text{test RelativeNormUnbalance \ tol? numIter? <flag?>} \]
  \[ \text{test RelativeNormDispIncr \ tol? numIter? <flag?>} \]
  \[ \text{test RelativeNormEnergyIncr \ tol? numIter? <flag?>} \]
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: remember these!

• 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
Commands that Return Values

• analyze command
  The analyze command returns 0 if successful. It returns a negative number if not
  
  \[ \text{set ok [analyze numIter } \lt \Delta T> ] \]

• getTime command
  The getTime command returns pseudo time in Domain.
  
  \[ \text{set currentTime [getTime]} \]

• nodeDisp command
  The nodeDisp command returns a nodal displacement.
  
  \[ \text{set disp [nodeDisp node dof]} \]
Example Usage – Displacement Control

```plaintext
set maxU 15.0; set dU 0.1
constraints transformation
numberer RCM
system BandGeneral
test NormDispIncr 1.0e-6 6 2
algorithm Newton
integrator DispControl 3 1 $dU
analysis Static
set ok 0
set currentDisp 0.0
while {ok == 0 && currentDisp < maxU} {
    set ok [analyze 1]
    if {ok != 0} {
        test NormDispIncr 1.0e-6 1000 1
        algorithm ModifiedNewton –initial
        set ok [analyze 1]
        test NormDispIncr 1.0e-6 6 2
        algorithm Newton
    }
    set currentDisp [nodeDisp 3 1]
}
```
Example Usage – Transient Analysis

```plaintext
set tFinal 15.0; set dT 0.01;
constraints Transformation
numberer RCM
system BandGeneral
test NormDispIncr 1.0e-6 6 2
algorithm Newton
integrator Newmark 0.5 0.25
analysis Transient
set ok 0
set currentTime 0.0
while {$ok == 0 && $currentTime < $tFinal} {
    set ok [analyze 1 $dT]
    if {$ok != 0} {
        test NormDispIncr 1.0e-6 1000 1
        algorithm ModifiedNewton –initial
        set ok [analyze 1 $dT]
        test NormDispIncr 1.0e-6 6 2
        algorithm Newton
    }
    set currentTime [getTime]
}
```