PEER

NEESit

Introducing a New Uniaxial Material into OpenSees

Frank McKenna

OpenSees Developers Workshop
Berkeley, CA
August 15, 2006

Sponsored by the National Science Foundation
through the Pacific Earthquake Engineering Research Center
and through NEESit

UniaxialMaterial Functionality

- Represent *all* one-dimensional material models
  - Stress-strain (σ–ε)
  - Force-deformation (s-e)
- Where are these objects used in OpenSees?
  - Truss elements (σ–ε)
  - Zero length elements (s-e)
  - Beam section fibers (σ–ε), e.g., steel, concrete
  - Resultant beam sections (s-e), e.g., M-κ
- Calling object (element or other material object) determines appropriate meaning, σ–ε or s-e

UniaxialMaterial

- Base class, UniaxialMaterial, provides its subclasses with a common interface
- The base class defines no data, only provides *virtual* methods, some with default behavior
- Subclasses provide specific implementations of the UniaxialMaterial interface

UniaxialMaterial Interface

```
class UniaxialMaterial : public Material
{
  public:
    UniaxialMaterial(int tag, int classTag);
    virtual ~UniaxialMaterial();

    virtual int setTrialStrain(double strain, double strainRate = 0.0) = 0;
    virtual double getStrain(void) = 0;
    virtual double getStrainRate(void);
    virtual double getStress(void) = 0;
    virtual double getTangent(void) = 0;
    virtual double getInitialtangent(void) =0;
    virtual double getDampTangent(void);

    virtual int commitState(void) = 0;
    virtual int revertToLastCommit(void) = 0;
    virtual int revertToStart(void) = 0;

    virtual UniaxialMaterial *getCopy(void) = 0;

    virtual Response *setResponse(const char **argv, int argc, Information &info);
    virtual int getResponse(int responseID, Information &info);
    virtual void Print(OPS_Stream &s, int flag =0);

  protected:
  private:
};
```

Must be overridden by subclass, “pure virtual”

Can be overridden by subclass

Constructor

```
UniaxialMaterial::UniaxialMaterial(int tag, int classTag):
Material(tag, classTag)
{
   // Does nothing
}
```

- This constructor calls its base class constructor (Material), which in turn calls the TaggedObject and MovableObject constructors
- Subclass constructors will take material parameters (floating point values or other data types) as arguments

Destructor

```
UniaxialMaterial::~UniaxialMaterial()
{
   // Does nothing
}
```

- This destructor does nothing
- Subclass destructors will deallocate any dynamically allocated memory

setTrialStrain

```
virtual int setTrialStrain(double strain,
                           double strainRate = 0.0);
```

- Must be implemented by subclasses
- Takes a strain and strain rate as arguments
- The default value passed for the strain rate is 0.0
- To store the current strain and strain rate (if used)

getStrain/getStrainRate

```
virtual double getStrain(void) = 0;
```

ε

- Must be implemented by subclasses
- To return the current value of strain

```
double
UniaxialMaterial::getStrainRate(void)
{
    return 0.0;
}
```

$\dot{\varepsilon}$

- This is the default implementation for strain rate

getStress

```
virtual double getStress(void) = 0;
```

$$\sigma = \sigma(\varepsilon, \dot{\varepsilon})$$

- Must be implemented by subclasses
- To return the current value of material stress including both strain and strain rate (if any) effects

getTangent/getDampTangent

```
virtual double getTangent(void) = 0;
```

$$\mathrm{D_t} = \frac{\partial\sigma}{\partial\varepsilon}$$

- Must be implemented by subclasses
- To return the current material tangent

```
double
UniaxialMaterial::getDampTangent(void)
{
   return 0.0;
}
```

$$\eta = \frac{\partial\sigma}{\partial\varepsilon \text{Ý}}$$

- This is the default implementation for the damping tangent

commitState

```
virtual int commitState(void) = 0;
```

- Must be implemented by subclasses
- UniaxialMaterial objects must manage their own history variables
- This method is invoked upon convergence during a time step
- To update the history variables for path dependence (if any) from time t_n to t_{n+1}
- To return 0 if successful and a negative number otherwise

revertToLastCommit/revertToStart

```
virtual int revertToLastCommit(void) = 0;
```

- To revert to last commit, i.e., at time t_n
- For most materials, nothing needs to be done here
- To return 0 if successful and a negative number otherwise

```
virtual int revertToStart(void) = 0;
```

- To revert to initial state
- To return 0 if successful and a negative number otherwise

getCopy

```
virtual UniaxialMaterial *getCopy(void) = 0;
```

- Must be implemented by subclasses
- To return a pointer to a new object which is a copy of this UniaxialMaterial object
- Invoked by a calling object, which is responsible for deallocating the dynamically allocated memory

sendSelf/recvSelf

```
virtual int sendSelf(int cTag, Channel &theChannel) = 0;

virtual int recvSelf(int cTag, Channel &theChannel,
                     FEM_ObjectBroker &theBroker) = 0;
```

- Two methods inherited from MovableObject for parallel processing and database programming
- To return 0 if successfully moved, a negative number otherwise
- Just make them return –1, then we’ll sort it out for you later!

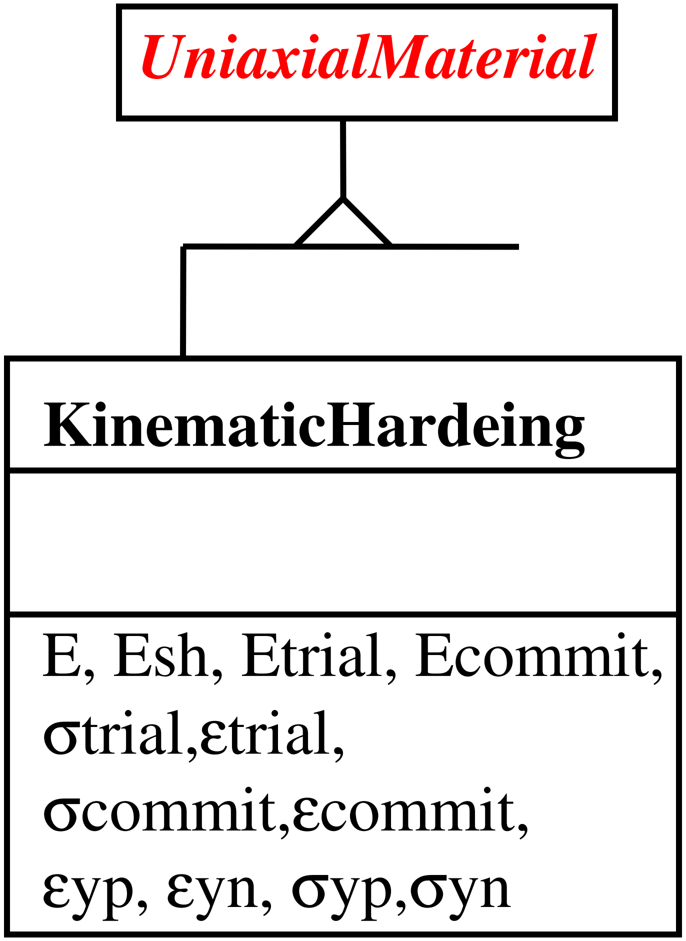
Print

```
  virtual void Print(ostream &s, int flag = 0) = 0;
```

- A utility method inherited from TaggedObject
- To print relevant information about the material, i.e., class name, tag, and material parameters to the stream object `s` passed as an argument
- For example:

```
s << “ElasticMaterial, tag: “ << this->getTag() << endl;
s << “E: “ << E << endln;
```

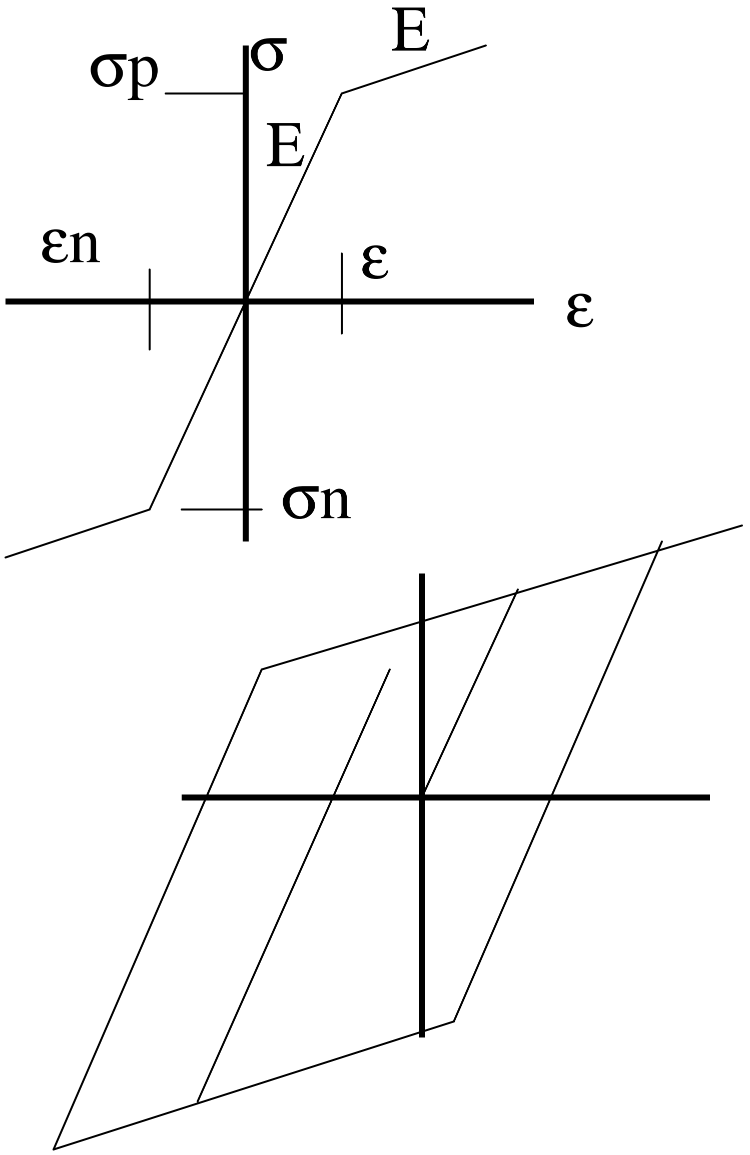
New KinematicHardening Material



UniaxialMaterial

KinematicHardeing

E, Esh, Etrial, Ecommit,
σtrial,εtrial,
σcommit,εcommit,
εyp, εyn, σyp,σyn



Implementation Exercise

Now it’s time to see how it really works …

- Save SRC/material/uniaxial/NewUniaxialMaterial.h/.cpp as KinematicHardening.h/.cpp and add these new files to the VC++ workspace (or Makefile).
- Check they compile & link.
- Make the changes outlined in following slides. (hint suggest compile & link after each method)
- Save SRC/material/uniaxial/TclNewMaterial.cpp as TclHardeningMaterial.cpp and make changes.
- Test it using the OpenSees/Tk UniaxialMaterial tester to see if your implementation works!!!

Interface

```
class KinematicHardeining : public UniaxialMaterial {
      public:
         KinematicHardeing(int tag, double E, double Ekh, double sigY);
         KinematicHardening();
        ~KinematicHardening();
         int setTrialStrain(double strain, double strainRate=0.0);
         double getStrain(void);
         double getStress(void);
        double getTangent(void);
         double getInitialTangent(void);

         int commitState(void);
         int revertToLastCommit(void);
         int revertToStart(void);

        UniaxialMaterial *getCopy(void);
        void Print(OPS_Stream &s, int flag = 0);
  private:
         double E, Ekh;
         double trialStrain, trialStress, trialTangent;
         double commitStrain, commitStress, commitTangent;
         double eyp, eyn, Syp, Syn;
};
```

Implementation

```
KinematicHardeining ::KinematicHardening(int tag, double e, double ekh, double sY)
:UniaxialMaterial(tag, MAT_TAG_KinematicHardening),
 E(e), Ekh(ekh), trialStrain(0.0),trialStress(0.0),trialTangent(e),
 commitStreain(0.0),commitStress(0.0),commitTangent(e)
{
   eyp = sY/e;
   eyn = -eyp;
   Syp = sY;
   Syn =-Sy;
}
KinematicHardeining ::KinematicHardening()
:UniaxialMaterial(tag, MAT_TAG_KinematicHardening)
 E(e), Ekh(ekh), trialStrain(0.0),trialStress(0.0),trialTangent(e),
 commitStreain(0.0),commitStress(0.0),commitTangent(e),eyp(0),eyn(0),Syp(0),Syn(0)
{
 }
KinematicHardening::~KinematicHardening
{
   // does nothing .. No memory to clean up
}
UniaxialMaterial *KinematicHardening::getCopy(void)
{
      KinematicMaterial *theCopy = new KinematicMaterial(this->getTag(), E, Ekh, Syp);
      return theCopy;
};
```

Implementation

```
int KinematicHardeining ::setTrialStrain(double strain, double strainRate)
{
     trialStrain = strain;
     if (trialStrain > eyp) {
       trialTangent = Ekh;
       trialStress = (trialStrain-eyp)*Ekh + Syp;
   } else if (trialStrain < eyn) {
       trialTangent = Ekh;
       trialStress = (trialStrain-eyn)*Ekh + Syn;
   } else {
        trialTangent = E;
        trialStress = (trialStrain - eyn)*E +Syn;
   }
    return 0;
}
double KinematicHardening::getStress(void)
{
     return trialStress;
}
double KinematicHardening::getTangent(void)
{
      return trialTangent;
}
```

Implementation

```
int KinematicHardeining ::commitState(void)
{
    if (trialStrain > eyp || trialStrain < eyn) {
      double diff ;
      if (trialStrain > eyp)
         diff =  trialStrain-eyp;
     else
         diff = trialStrain-eyn;
      eyp += diff;
      eyn += diff;
      Syp += diff*Ekh;
      Syn += diff*Ekh;
    }
    commitStrain = trialStrain;
    commitStress = trialStress;
    commitTangent = trialTangent;
    return 0;
 }
int KinematicHardening::revertToLastCommit(void)
{
     trialStrain = commitSTrain;
     trialStress = commitStress;
     trialTangent = commitTangent;
}
```

TclModelBuilderMaterialCommand.cpp additions:

```
int TclCommand_KinematicHardening(ClientData clientData, Tcl_Interp
      *interp, int argc, TCL_Char **argv, TclModelBuilder *tB);
```

```
else if strcmp(argv[1],”KinematicHardening” == 0) {
   return  TclCommand_KinematicHardening(clientData, interp, argc, argv,
      theTclBuilder);
}
```

TclKinematicHardening.cpp changes:

```
int TclCommand_KinematicHardening(ClientData clientData, Tcl_Interp
      *interp, int argc, TCL_Char **argv, TclModelBuilder *tB)
{
    int tag;
   double E, Ekh, Sy;
   UniaxialMaterial *theUniaxialMaterial = 0;
   Tcl_GetInt(interp, argv[2], &tag);
    Tcl_GetDouble(interp, argv[3], &E);
    Tcl_GetDouble(interp, argv[4], &Ekh);
    Tcl_GetDouble(interp, argv[5], &Sy);
    theUniaxialMaterial = new KinematicHardening(tag, E, Ekh, Sy);
    return theTclBuilder->addUniaxialMaterial(*theMaterial);
```