QuakeCoRE
NZ Centre for Earthquake Resilience

QuakeCoRE OpenSees Training Workshop 2016

Adding a New Material or Element to OpenSees

UC University of Canterbury
The University of Auckland New Zealand
Resilient Organisations
GNS Science
Massey University
Te Whare Wānanga o te Upoko o te Ika a Maui Victoria University of Wellington
The University of Waikato Te Whare Wānanga o Waikato
BRANZ

Adding New Classes to OpenSees

QuakeCoRE
NZ Centre for Earthquake Resilience

One of the best aspects of the open source nature of OpenSees is that anyone can download the source code and make changes and/or add new classes.

These new or modified classes (e.g. materials, elements, integrators) can be used locally with your own version of the OpenSees code, and with sufficient testing these new classes can be added to the main source code so any OpenSees user can access the functionality that you have added.

There are two main ways to add a new class:

- Create a library (e.g. myNewMat.dll on Windows, or myNewMat.so on Unix-based system) that can be found by the OpenSees interpreter
- Adding the new class to the VisualStudio project (for Windows) or to the Makefiles (for Unix-based) to be compiled and linked with the remainder of the classes included in the OpenSees framework.

Adding New Classes to OpenSees

Detailed instructions for how to implement a new material using the library option (.dll or .so file that can be found by OpenSees) are available on the OpenSees wiki site at :

http://opensees.berkeley.edu/wiki/index.php/Adding_your_own_Code

Today we will look at things in terms of the other option, adding the new class to the VisualStudio project and Makefiles to be compiled/linked with the rest of the OpenSees classes

The primary goal of this module will be to understand the essential steps needed to implement a new class in OpenSees. We will attempt to focus on the important aspects of this process and not get lost in C++ details.

- The intention is to convey that the general process is really quite easy

Adding New Classes to OpenSees

QuakeCoRE
NZ Centre for Earthquake Resilience

Detailed instructions for how to implement a new material using the library option (.dll or .so file that can be found by OpenSees) are available on the OpenSees wiki site at:

http://opensees.berkeley.edu/wiki/index.php/Adding_your_own_Code

Today we will look at things in terms of the other option, adding the new class to the VisualStudio project and Makefiles to be compiled/linked with the rest of the OpenSees classes

The primary goal of this module will be to understand the essential steps needed to implement a new class in OpenSees. We will attempt to focus on the important aspects of this process and not get lost in C++ details.

- The intention is to convey that the general process is really quite easy

New code for OpenSees can be written in C, C++, or FORTRAN. The examples we examine will be in C++, but the essential process is the same regardless.

Adding New Classes to OpenSees

QuakeCoRE
NZ Centre for Earthquake Resilience

The first step is to obtain the OpenSees source code. A local copy can be checked out using subversion (svn). Anyone can checkout the code, only a few people can commit changes directly.

- svn is a version control tool. On Linux or Mac OS X, there is a command line svn client. On Windows, TortoiseSVN (https://tortoisesvn.net/) is a great subversion tool.

Once you have subversion up and running, the source code can be checked out. This can be accomplished by typing (or copy/pasting) into the terminal:

```
svn co svn://peera.berkeley.edu/usr/local/svn/OpenSees/trunk OpenSees
```

Or if using TortoiseSVN, you can right click in the desired directory and select Checkout from the contextual menu that appears, then enter the address above into the appropriate location.

Adding New Classes to OpenSees

QuakeCoRE
NZ Centre for Earthquake Resilience

The directory structure that you checkout will look something like this. To compile the code on your local machine, you will need to follow the instructions given at http://opensees.berkeley.edu/OpenSees/developer/builds.php.

This is a very important step, but it's not the focus of this module, so we will now assume in subsequent discussion that we have a working build of OpenSees

```
enci-crm122:OpenSees crm122$ ll
total 32
-rw-r--r--   1 crm122  admin  2108 18 May 14:07 COPYRIGHT
drwxr-xr-x  15 crm122  admin   510 18 May 14:06 DEVELOPER
drwxr-xr-x  22 crm122  admin   748 18 May 14:07 EXAMPLES
drwxr-xr-x  34 crm122  admin  1156  3 Jun 12:13 MAKES
-rw-r--r--   1 crm122  admin  2740 18 May 14:07 Makefile
lrwxr-xr-x   1 crm122  admin    29 20 May 08:39 Makefile.def -> MAKES/Makefile.def.MacOS10.11
drwxr-xr-x  19 crm122  admin   646 18 May 14:06 OTHER
-rw-r--r--   1 crm122  admin  2775 18 May 14:07 README
drwxr-xr-x   4 crm122  admin   136 18 May 14:06 SCRIPTS
drwxr-xr-x  41 crm122  admin  1394 18 May 14:07 SRC
drwxr-xr-x  11 crm122  admin   374 18 May 14:06 Win32
drwxr-xr-x   7 crm122  admin   238 18 May 14:07 Win64
drwxr-xr-x   4 crm122  admin   136 18 May 14:05 Workshops
drwxr-xr-x   5 crm122  admin   170  3 Jun 14:34 bin
drwxr-xr-x  14 crm122  admin   476  3 Jun 14:34 lib
enci-crm122:OpenSees crm122$
```

Adding New Classes to OpenSees

QuakeCoRE
NZ Centre for Earthquake Resilience

```
[enci-crm122:OpenSees crm122$ cd SRC
[enci-crm122:SRC crm122$ ll
total 272
-rw-r--r--   1 crm122  admin   2489 18 May 14:07 G3Globals.h
-rw-r--r--   1 crm122  admin  53636 18 May 14:07 Makefile
-rw-r--r--   1 crm122  admin   9435 18 May 14:07 Makefile.incl
-rw-r--r--   1 crm122  admin   3002 18 May 14:06 OPS_Globals.h
drwxr-xr-x  11 crm122  admin    374 18 May 14:06 actor
drwxr-xr-x  12 crm122  admin    408 18 May 14:07 analysis
drwxr-xr-x  12 crm122  admin    408 28 May 11:24 api
-rw-r--r--   1 crm122  admin   2008 18 May 14:07 bool.h
-rw-r--r--   1 crm122  admin  39516 18 May 14:07 classTags.h
drwxr-xr-x  44 crm122  admin   1496 28 May 11:21 convergenceTest
drwxr-xr-x  29 crm122  admin    986 28 May 11:24 coordTransformation
drwxr-xr-x  26 crm122  admin    884 28 May 11:24 damage
drwxr-xr-x  28 crm122  admin    952 28 May 11:19 database
drwxr-xr-x  42 crm122  admin   1428 18 May 14:06 doc
drwxr-xr-x  15 crm122  admin    510 18 May 14:07 domain
drwxr-xr-x  53 crm122  admin   1802 28 May 11:22 element
drwxr-xr-x   7 crm122  admin    238 18 May 14:06 graph
drwxr-xr-x  55 crm122  admin   1870 28 May 11:22 handler
drwxr-xr-x  21 crm122  admin    714 24 May 15:02 interpreter
drwxr-xr-x   8 crm122  admin    272 18 May 14:07 java
drwxr-xr-x   5 crm122  admin    170 28 May 11:22 machine
drwxr-xr-x  12 crm122  admin    408 28 May 11:19 material
drwxr-xr-x  19 crm122  admin    646 28 May 11:19 matrix
drwxr-xr-x  18 crm122  admin    612 28 May 11:19 modelbuilder
drwxr-xr-x   6 crm122  admin    204 18 May 14:06 optimization
-rw-r--r--   1 crm122  admin   5675 18 May 14:07 readme
drwxr-xr-x  61 crm122  admin   2074 28 May 11:21 recorder
drwxr-xr-x   7 crm122  admin    238 18 May 14:07 reliability
drwxr-xr-x   3 crm122  admin    102 18 May 14:07 remote
-rw-r--r--   1 crm122  admin   1885 18 May 14:06 remote.h
drwxr-xr-x  51 crm122  admin   1734 28 May 11:21 renderer
drwxr-xr-x   3 crm122  admin    102 18 May 14:06 scripts
drwxr-xr-x   4 crm122  admin    136 18 May 14:07 string
drwxr-xr-x  13 crm122  admin    442 18 May 14:07 system_of_eqn
drwxr-xr-x   8 crm122  admin    272 28 May 11:19 tagged
drwxr-xr-x  30 crm122  admin   1020 28 May 11:21 tcl
-rw-r--r--   1 crm122  admin    135 18 May 14:07 test.h
drwxr-xr-x   4 crm122  admin    136 18 May 14:07 unittest
drwxr-xr-x  24 crm122  admin    816 28 May 11:21 utility
```

The classes for OpenSees are located in the SRC directory.

We will put our new class into the appropriate directory out of the options shown here.

Some of these have further subdirectories to better divide the classes. For example, the material directory contains separate subdirectories for nDMaterials and uniaxialMaterials.

Adding New Classes to OpenSees

QuakeCoRE
NZ Centre for Earthquake Resilience

The first example we will examine is an nDMaterial. Our new nDMaterial subclass will require a header file (ourNewmaterial.h) that contains definitions for the new material and an implementation file (ourNewmaterial.cpp) that contains the material implementation.

- The process for a uniaxialMaterial is identical, just swap out all instances of nDMaterial with unixialMaterial and everything else should be the same

When starting to write your own new material/element/whatever, don't start from a blank text file. Use existing classes that work as an example of what to do in your own code.

We will look at the header file first.

Adding New Material to OpenSees

QuakeCoRE
NZ Centre for Earthquake Resilience

The first example we will examine is an nDMaterial. Our new nDMaterial subclass will require a header file (ourNewmaterial.h) that contains definitions for the new material and an implementation file (ourNewmaterial.cpp) that contains the material implementation.

- The process for a uniaxialMaterial is identical, just swap out all instances of nDMaterial with unixialMaterial and everything else should be the same

When starting to write your own new material/element/whatever, don't start from a blank text file. Use existing classes that work as an example of what to do in your own code.

We will look at the header file first. This file contains the class definition for the example nDMaterial (BoundingCamClay in this case). Here we define all of the member variables and functions for our class.

Adding New Material to OpenSees

QuakeCoRE
NZ Centre for Earthquake Resilience

BoundingCamClay.h (~/OpenSees/SRC/material/nD/UWmaterials) - VIM1

```
/* ****************************************************************** **
**    OpenSees - Open System for Earthquake Engineering Simulation    **
**          Pacific Earthquake Engineering Research Center            **
**                                                                    **
**                                                                    **
** (C) Copyright 1999, The Regents of the University of California    **
** All Rights Reserved.                                               **
**                                                                    **
** Commercial use of this program without express permission of the   **
** University of California, Berkeley, is strictly prohibited.  See   **
** file 'COPYRIGHT'  in main directory for information on usage and   **
** redistribution,  and for a DISCLAIMER OF ALL WARRANTIES.           **
**                                                                    **
** Developed by:                                                      **
**   Frank McKenna (fmckenna@ce.berkeley.edu)                         **
**   Gregory L. Fenves (fenves@ce.berkeley.edu)                       **
**   Filip C. Filippou (filippou@ce.berkeley.edu)                     **
**                                                                    **
** ****************************************************************** */

#ifndef BoundingCamClay_h
#define BoundingCamClay_h

// Written: Kathryn Petek
//          December 2004
// Modified: Chris McGann
//           January 2011

// Description: This file contains the class definition for BoundingCamClay.


#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include <NDMaterial.h>
#include <Matrix.h>
#include <Vector.h>

class BoundingCamClay : public NDMaterial
{
  public:

    // full constructor
    BoundingCamClay(int tag, int classTag, double massDen, double C, double bulk, double OCR,
                    double mu_o, double Alpha, double lambda, double h, double m);
```

At the top of the header file we list the includes that we may need (if you're not sure what to include, look at similar classes and see what they are using)

We also define the BoundingCamClay class as a subclass of the NDMaterial class

Adding New Material to OpenSees

QuakeCoRE
NZ Centre for Earthquake Resilience

BoundingCamClay.h (~/OpenSees/SRC/material/nD/UWmaterials) - VIM1

```
class BoundingCamClay : public NDMaterial
{
  public:

    // full constructor
    BoundingCamClay(int tag, int classTag, double massDen, double C, double bulk, double OCR,
                    double mu_o, double Alpha, double lambda, double h, double m);
    // null constructor
    BoundingCamClay();

    // destructor
    ~BoundingCamClay();

    NDMaterial *getCopy(const char *type);

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

    NDMaterial *getCopy(void);
    const char *getType(void) const;
    int getOrder(void) const;

    Response *setResponse (const char **argv, int argc, OPS_Stream &output);
    int getResponse (int responseID, Information &matInformation);

    int sendSelf(int commitTag, Channel &theChannel);
    int recvSelf(int commitTag, Channel &theChannel, FEM_ObjectBroker &theBroker);

    void Print(OPS_Stream &s, int flag =0);

    int setParameter(const char **argv, int argc, Parameter &param);
    int updateParameter(int responseID, Information &eleInformation);

    // send mass density to element in dynamic analysis
    double getRho(void) {return massDen;};

  protected:

    // input material parameters
    double iC;              // ellipsoildal axis ratio
    double mBulk;           // initial bulk modulus
    double iOCR;            // overconsolidation ratio
    double ikappa;          // elastic compressibility index
    double imu_o;           // elastic shear modulus
    double ialpha;          // pressure-dependent parameter
```

We also define the member variables and functions for our class in the header file.

Some of these functions are public functions that are used by all classes. We need to redefine each one of these public functions unless we just want to use the base class functionality.

Some variables and functions are protected, meaning they can only be used by the current class.

Adding New Material to OpenSees

QuakeCoRE
NZ Centre for Earthquake Resilience

BoundingCamClay3D.h (~/OpenSees/SRC/material/nD/UWmaterials) - VIM3

```
#include <BoundingCamClay.h>

class BoundingCamClay3D : public BoundingCamClay {

//------------------Declarations-------------------------------

  public :

  //null constructor
  BoundingCamClay3D( ) ;

  //full constructor
  BoundingCamClay3D(int tag, double mDen, double c, double bulk, double OCR, double mu_o,
                            double alpha, double lambda, double h, double m);

  //destructor
  ~BoundingCamClay3D( ) ;

  NDMaterial* getCopy( ) ;
  const char* getType( ) const ;
  int getOrder( ) const ;

  int setTrialStrain(const Vector &strain_from_element);

  // Unused trialStrain functions
  int setTrialStrain(const Vector &v, const Vector &r);

  //send back the strain
  const Vector& getStrain( ) ;

  //send back the stress
  const Vector& getStress( ) ;

  //send back the tangent
  const Matrix& getTangent( ) ;
  const Matrix& getInitialTangent( ) ;

  private :


} ; //end of BoundingCamClay3D declarations
```

As BoundingCamClay is an nDMaterial, we have to define how it works in 3D as well as in 2D configurations such as plane strain.

This is accomplished by defining subclasses called BoundingCamClay3D and BoundingCamClayPlaneStrain

Adding New Material to OpenSees

QuakeCoRE
NZ Centre for Earthquake Resilience

BoundingCamClay3D.h (~/OpenSees/SRC/material/nD/UWmaterials) - VIM3

```
#include <BoundingCamClay.h>

class BoundingCamClay3D : public BoundingCamClay {

//------------------Declarations--------------------------------

  public :

  //null constructor
  BoundingCamClay3D( ) ;

  //full constructor
  BoundingCamClay3D(int tag, double mDen, double c, double bulk, double OCR, double mu_o,
                            double alpha, double lambda, double h, double m);

  //destructor
  ~BoundingCamClay3D( ) ;

  NDMaterial* getCopy( ) ;
  const char* getType( ) const ;
  int getOrder( ) const ;

  int setTrialStrain(const Vector &strain_from_element);

  // Unused trialStrain functions
  int setTrialStrain(const Vector &v, const Vector &r);

  //send back the strain
  const Vector& getStrain( ) ;

  //send back the stress
  const Vector& getStress( ) ;

  //send back the tangent
  const Matrix& getTangent( ) ;
  const Matrix& getInitialTangent( ) ;

  private :


} ; //end of BoundingCamClay3D declarations
```

The most important function in the material!!!!

The element sends a trial strain to the nDMaterial, then some algorithm (your constitutive model) determines the corresponding stress and tangent which are then queried by the element using the getStress and getTangent methods

Adding New Material to OpenSees

QuakeCoRE
NZ Centre for Earthquake Resilience

BoundingCamClay3D.cpp (~/OpenSees/SRC/material/nD/UWmaterials) - VIM2

```
//send back order of strain in vector form
int BoundingCamClay3D::getOrder() const
{
    return 6;
}

//get the strain and integrate plasticity equations
int BoundingCamClay3D::setTrialStrain( const Vector &strain_from_element)
{
    mEpsilon = strain_from_element;

    this->plastic_integrator();

    return 0 ;
}

//unused trial strain functions
int BoundingCamClay3D::setTrialStrain(const Vector &v, const Vector &r)
{
    opserr << "YOU SHOULD NOT SEE THIS: BoundingCamClay::setTrialStrain (const Vector &v, const Vector &r)" << endln;
    return this->setTrialStrain (v);
}

//send back the strain
const Vector& BoundingCamClay3D::getStrain()
{
    return mEpsilon;
}

//send back the stress
const Vector& BoundingCamClay3D::getStress()
{
    return mSigma;
}

//send back the tangent
const Matrix& BoundingCamClay3D::getTangent()
{
    return mCep;
}

//send back the tangent
const Matrix& BoundingCamClay3D::getInitialTangent()
{
    return mCep;
```

This is the setTrialStrain function in the BoundingCamClay3D.cpp file.

We set the member variable mEpsilon to be equal to the trial strain from the element, then call the member function plastic_integrator() which is defined in the main BoundingCamClay.cpp file.

Adding New Material to OpenSees

QuakeCoRE
NZ Centre for Earthquake Resilience

BoundingCamClay3D.cpp (~/OpenSees/SRC/material/nD/UWmaterials) - VIM2

```
//send back order of strain in vector form
int BoundingCamClay3D::getOrder() const
{
    return 6;
}

//get the strain and integrate plasticity equations
int BoundingCamClay3D::setTrialStrain( const Vector &strain_from_element)
{
    mEpsilon = strain_from_element;

    this->plastic_integrator();

    return 0 ;
}

//unused trial strain functions
int BoundingCamClay3D::setTrialStrain(const Vector &v, const Vector &r)
{
    opserr << "YOU SHOULD NOT SEE THIS: BoundingCamClay::setTrialStrain (const Vector &v, const Vector &r)" << endln;
    return this->setTrialStrain (v);
}

//send back the strain
const Vector& BoundingCamClay3D::getStrain()
{
    return mEpsilon;
}

//send back the stress
const Vector& BoundingCamClay3D::getStress()
{
    return mSigma;
}

//send back the tangent
const Matrix& BoundingCamClay3D::getTangent()
{
    return mCep;
}

//send back the tangent
const Matrix& BoundingCamClay3D::getInitialTangent()
{
    return mCep;
```

This is where we define what the BoundingCamClay3D class sends back to an element when queried for a stress, a tangent, or an initial tangent.

Adding New Material to OpenSees

QuakeCoRE
NZ Centre for Earthquake Resilience

BoundingCamClayPlaneStrain.cpp + (~/OpenSees/SRC/material/nD/UWmaterials) - VIM4

```
//send back order of strain in vector form
int BoundingCamClayPlaneStrain::getOrder() const
{
    return 3;
}

//get the strain and integrate plasticity equations
int BoundingCamClayPlaneStrain::setTrialStrain(const Vector &strain_from_element)
{
    mEpsilon.Zero();
    mEpsilon(0) = strain_from_element(0);
    mEpsilon(1) = strain_from_element(1);
    mEpsilon(3) = strain_from_element(2);

    this->plastic_integrator();

    return 0;
}

//unused trial strain function
int BoundingCamClayPlaneStrain::setTrialStrain(const Vector &v, const Vector &r)
{
    return this->setTrialStrain (v);
}

//send back the strain
const Vector& BoundingCamClayPlaneStrain::getStrain()
{
    strain(0) = mEpsilon(0);
    strain(1) = mEpsilon(1);
    strain(2) = mEpsilon(3);

    return strain;
}

//send back the stress
const Vector& BoundingCamClayPlaneStrain::getStress()
{
    stress(0) = mSigma(0);
    stress(1) = mSigma(1);
    stress(2) = mSigma(3);

    return stress;
}
```

For comparison, these are the corresponding functions for the BoundingCamClayPlaneStrain subclass.

Note that the setTrialStrain function calls the same member function to run the constitutive model algorithm, but now the strain vector coming from the element only has 3 components so the rest are set to zero.

Adding New Material to OpenSees

QuakeCoRE
NZ Centre for Earthquake Resilience

BoundingCamClay.

```
static int numBoundingCamClayMaterials = 0;

OPS_Export void *
OPS_NewBoundingCamClayMaterial(void)
{
    if (numBoundingCamClayMaterials == 0) {
      numBoundingCamClayMaterials++;
      opserr << "BoundingCamClay nDmaterial - Written: C.McGann, K.Petek, P.Arduino, U.Washington\n";
    }

    NDMaterial *theMaterial = 0;

    int numArgs = OPS_GetNumRemainingInputArgs();

    if (numArgs < 10) {
      opserr << "Want: nDMaterial BoundingCamClay tag? massDensity? C? bulk? OCR? mu_o? alpha? lambda? h? m?" << endln;
      return 0;
    }

    int tag;
    double dData[9];

    int numData = 1;
    if (OPS_GetInt(&numData, &tag) != 0) {
      opserr << "WARNING invalid nDMaterial BoundingCamClay material tag" << endln;
      return 0;
    }
    numData = 9;
    if (OPS_GetDouble(&numData, dData) != 0) {
      opserr << "WARNING invalid material data for nDMaterial BoundingCamClay material with tag: " << tag << endln;
      return 0;
    }

    theMaterial = new BoundingCamClay(tag, 0, dData[0], dData[1], dData[2], dData[3], dData[4], dData[5],
                                      dData[6], dData[7], dData[8]);

    if (theMaterial == 0) {
      opserr << "WARNING ran out of memory for nDMaterial BoundingCamClay material with tag: " << tag << endln;
    }

    return theMaterial;
}
```

This function is the interface between the interpreter and the BoundingCamClay class. It creates the material from the info provided.

Adding New Material to OpenSees

QuakeCoRE
NZ Centre for Earthquake Resilience

BoundingCamClay.cpp (~/OpenSees/SRC/material/nD/UWmaterials) - VIM

```
    return theMaterial;
}

// full constructor
BoundingCamClay::BoundingCamClay(int tag, int classTag, double mDen, double C, double bulk, double OCR, double mu_o,
                                 double Alpha, double lambda, double h, double m)
  : NDMaterial(tag,ND_TAG_BoundingCamClay),
    mEpsilon(6),
    mEpsilon_P(6),
    mEpsilon_n_P(6),
    mSigma(6),
    mSigma_n(6),
    mSIGMAo(6),
    mSIGMAo_n(6),
    mM(6,6),
    mCe(6,6),
    mCep(6,6),
    mI1(6),
    mIIco(6,6),
    mIIcon(6,6),
    mIImix(6,6),
    mIIvol(6,6),
    mIIdevCon(6,6),
    mIIdevMix(6,6),
    mState(7)
{
    massDen = mDen;
    iC = C;
    mBulk = bulk;
    iOCR = OCR;
    imu_o = mu_o;
    ialpha = Alpha;
    ilambda = lambda;
    ih = h;
    im = m;

    this->initialize();
}


// null constructor
BoundingCamClay ::BoundingCamClay()
    : NDMaterial(),
```

The full constructor is the function called when the material is first created. This function should initialize variables with the values of the input parameters entered into the interpreter as well as all variables are needed for the material algorithm.

Adding New Material to OpenSees

QuakeCoRE
NZ Centre for Earthquake Resilience

BoundingCamClay.cpp (~/OpenSees/SRC/material/nD/UWmaterials) - VIM

```
}


// null constructor
BoundingCamClay ::BoundingCamClay()
    : NDMaterial(),
    mEpsilon(6),
    mEpsilon_P(6),
    mEpsilon_n_P(6),
    mSigma(6),
    mSigma_n(6),
    mSIGMAo(6),
    mSIGMAo_n(6),
    mM(6,6),
    mCe(6,6),
    mCep(6,6),
    mI1(6),
    mIIco(6,6),
    mIIcon(6,6),
    mIImix(6,6),
    mIIvol(6,6),
    mIIdevCon(6,6),
    mIIdevMix(6,6),
    mState(7)
{
    massDen = 0.0;
    iC = 1.0;
    mBulk = 1.0;
    iOCR = 1.0;
    imu_o = 0.0;
    ialpha = 0.0;
    ilambda = 1.0;
    ih = 0.0;
    im = 1.0;

    this->initialize();
}

// destructor
BoundingCamClay::~BoundingCamClay()
{
}
```

The null constructor is the function called when a new material instance is created by either a database call or in parallel processing. This function should initialize any variables that are needed for the material algorithm and give some sort of value to the variables that will contain the input parameters.

The destructor provides any special instructions needed when the material object is removed.

Adding New Material to OpenSees

QuakeCoRE
NZ Centre for Earthquake Resilience

BoundingCamClay.cpp (~/OpenSees/SRC/material/nD/UWmaterials) - VIM

```
NDMaterial*
BoundingCamClay::getCopy(const char *type)
{
    if (strcmp(type,"PlanStrain2D") == 0 || strcmp(type,"PlaneStrain") == 0) {
        BoundingCamClayPlaneStrain *clone;
        clone = new BoundingCamClayPlaneStrain(this->getTag(), massDen, iC, mBulk, iOCR, imu_o, ialpha, ilambda, ih, im);
        return clone;
    } else if (strcmp(type,"ThreeDimensional")==0 || strcmp(type,"3D") ==0) {
        BoundingCamClay3D *clone;
        clone = new BoundingCamClay3D(this->getTag(), massDen, iC, mBulk, iOCR, imu_o, ialpha, ilambda, ih, im);
        return clone;
    } else {
        opserr << "BoundingCamClay::getCopy failed to get copy: " << type << endln;
        return 0;
    }
}

int
BoundingCamClay::commitState(void)
{
    // update state variables for next step
    mEpsilon_n_P  = mEpsilon_P;
    mSigma_n      = mSigma;
    mSIGMAo_n     = mSIGMAo;

    mr_n = mr;
    mR_n = mR;
    mKappa_n = mKappa;

    return 0;
}

int BoundingCamClay::revertToLastCommit (void)
{
    return 0;
}

int BoundingCamClay::revertToStart(void)
{
    // added for InitialStateAnalysis
    if (ops_InitialStateAnalysis) {
        // do nothing, keep state variables from last step
    } else {
```

The commitState function is called when the model has achieved global convergence in a given load step.

Any history variables needed for the constitutive algorithm should be updated here.

Adding New Material to OpenSees

QuakeCoRE
NZ Centre for Earthquake Resilience

BoundingCamClay.cpp (~/OpenSees/SRC/material/nD/UWmaterials) - VIM

```
Response*
BoundingCamClay::setResponse (const char **argv, int argc, OPS_Stream &output)
{
    if (strcmp(argv[0],"stress") == 0 || strcmp(argv[0],"stresses") == 0)
        return new MaterialResponse(this, 1, this->getStress());
    else if (strcmp(argv[0],"strain") == 0 || strcmp(argv[0],"strains") == 0)
        return new MaterialResponse(this, 2, this->getStrain());
    else if (strcmp(argv[0], "state") == 0)
        return new MaterialResponse(this, 3, this->GetState());
    else if (strcmp(argv[0], "center") == 0)
        return new MaterialResponse(this, 4, this->GetCenter());
    else
        return 0;
}

int
BoundingCamClay::getResponse(int responseID, Information &matInfo)
{
    switch (responseID) {
        case -1:
            return -1;
        case 1:
            if (matInfo.theVector != 0)
                *(matInfo.theVector) = getStress();
            return 0;
        case 2:
            if (matInfo.theVector != 0)
                *(matInfo.theVector) = getStrain();
            return 0;
        case 3:
            if (matInfo.theVector != 0)
                *(matInfo.theVector) = GetState();
            return 0;
        case 4:
            if (matInfo.theVector != 0)
                *(matInfo.theVector) = GetCenter();
            return 0;
        default:
            return -1;
    }
}
```

The setResponse and getResponse functions define the quantities that can be called by an element recorder as well as the information that is returned by any such call to this material.

Adding New Material to OpenSees

QuakeCoRE
NZ Centre for Earthquake Resilience

BoundingCamClay.cpp (~/OpenSees/SRC

```
int
BoundingCamClay::sendSelf(int commitTag, Channel &theChannel)
{
  // we place all the data needed to define material and it's state
  // int a vector object
  static Vector data(8);
  int cnt = 0;
  data(cnt++) = this->getTag();
  data(cnt++) = iC;
  data(cnt++) = iOCR;
  data(cnt++) = ikappa;
  data(cnt++) = imu_o;
  data(cnt++) = ialpha;
  data(cnt++) = ilambda;
  data(cnt++) = ih;
  data(cnt++) = im;
  data(cnt++) = iepsE_vo;

  // send the vector object to the channel
  if (theChannel.sendVector(this->getDbTag(), commitTag, data) < 0) {
    opserr << "BoundingCamClay::sendSelf - failed to send vector to channel\n";
    return -1;
  }


  return 0;

}

int
BoundingCamClay::recvSelf(int commitTag, Channel &theChannel,
                          FEM_ObjectBroker &theBroker)
{
  // recv the vector object from the channel which defines material param and state
  static Vector data(7);
  if (theChannel.recvVector(this->getDbTag(), commitTag, data) < 0) {
    opserr << "BoundingCamClay::recvSelf - failed to recv vector from channel\n";
    return -1;
  }

  // set the material parameters and state variables
  int cnt = 0;
  this->setTag(data(cnt++));
```

The sendSelf and recvSelf functions define the what is sent and received, respectively, any time a database is set or called or any time a new process creates a copy of this material in parallel processing. These should contain all the data needed to define the material and its state

Adding New Material to OpenSees

QuakeCoRE
NZ Centre for Earthquake Resilience

BoundingCamClay.cpp (~/OpenSees/SRC/material/nD/UWmaterials) - VIM

```
void BoundingCamClay::Print(OPS_Stream &s, int flag )
{
  s << "BoundingCamClay" << endln;
}

int
BoundingCamClay::setParameter(const char **argv, int argc, Parameter &param)
{
    if (strcmp(argv[0],"materialState") == 0) {
        // switch elastic/plastic state
        return param.addObject(5,this);
    } else {
        // invalid parameter type
        opserr << "WARNING: invalid parameter command for BoundingCamClay nDMaterial with tag: " << this->getTag() << endln;
        return -1;
    }

    return -1;
}

int
BoundingCamClay::updateParameter(int responseID, Information &info)
{
    if (responseID == 5) {
        // materialState called - update mElasticFlag
        mElastFlag = (int)info.theDouble;
    }

    return 0;
}
```

The Print function should provide any information about the material to be displayed whenever Print is called by another class.

The setParameter and updateParameter functions define the material's functionality when something like setParameter or updateMaterialStage is called in the interpreter (common functionality is to switch from elastic to plastic response

Adding New Material to OpenSees

QuakeCoRE
NZ Centre for Earthquake Resilience

Once we have defined the functions we just noted, as well as the all important constitutive algorithm function that takes the trial strain from the element and produces a trial stress and tangent to send back, we need to modify a few files to ensure that our new material is included in the compiling and linking steps.

Makefile (~/OpenSees/SRC/material/nD/UWmaterials) -...

```
include ../../../../Makefile.def

OBJS       = BoundingCamClay.o \
             BoundingCamClay3D.o \
             BoundingCamClayPlaneStrain.o \
             ContactMaterial2D.o \
             ContactMaterial3D.o \
          DruckerPrager.o \
             DruckerPrager3D.o \
          DruckerPragerPlaneStrain.o \
          InitialStateAnalysisWrapper.o\
          ManzariDafalias.o\
          ManzariDafalias3D.o\
          ManzariDafaliasPlaneStrain.o\
          ManzariDafaliasRO.o\
          ManzariDafalias3DRO.o\
          ManzariDafaliasPlaneStrainRO.o

all:    $(OBJS)

# Miscellaneous

tidy:
    @$(RM) $(RMFLAGS) Makefile.bak *~ #*# core

clean: tidy
    @$(RM) $(RMFLAGS) $(OBJS) *.o

spotless: clean

wipe: spotless

# DO NOT DELETE THIS LINE -- make depend depends on it.
```

For Linux or Mac builds of OpenSees, we need to add a few lines to the applicable Makefile(s) such that object (.o) files will be created when we compile the code.

The applicable Makefile is the one that resides in the same directory as our material files. If we created a new directory to house our new files, we'll also need to add a line to the ../Makefile to instruct make to look in this new directory

On Windows, the new material needs to be added to the VisualStudio project so it will be compiled when the project is built.

Adding New Material to OpenSees

QuakeCoRE
NZ Centre for Earthquake Resilience

TclModelBuilderNDMater...es/SRC/material/nD) - VIM

```
#include <string.h>

extern NDMaterial *
Tcl_addWrapperNDMaterial(matObj *, ClientData, Tcl_Interp *,  int, TCL_Char **, TclModelBuilder *);

extern  void *OPS_NewReinforcedConcretePlaneStressMaterial(void);
extern  void *OPS_NewFAReinforcedConcretePlaneStressMaterial(void);
extern  void *OPS_NewFAFourSteelRCPlaneStressMaterial(void);
extern  void *OPS_NewRAFourSteelRCPlaneStressMaterial(void);
extern  void *OPS_NewPrestressedConcretePlaneStressMaterial(void);
extern  void *OPS_NewFAPrestressedConcretePlaneStressMaterial(void);
extern  void *OPS_NewFAFourSteelPCPlaneStressMaterial(void);
extern  void *OPS_NewRAFourSteelPCPlaneStressMaterial(void);
extern  void *OPS_NewMaterialCMM(void);

extern  void *OPS_NewElasticIsotropicMaterial(void);
extern  void *OPS_NewElasticOrthotropicMaterial(void);
extern  void *OPS_NewBoundingCamClayMaterial(void);
extern  void *OPS_NewContactMaterial3DMaterial(void);
extern  void *OPS_NewInitialStateAnalysisWrapperMaterial(void);
extern  void *OPS_NewManzariDafaliasMaterial(void);
extern  void *OPS_NewManzariDafaliasMaterialRO(void);
extern  void *OPS_CycLiqCPMaterial(void);
extern  void *OPS_CycLiqCPSPMaterial(void);
extern  void *OPS_NewInitStressNDMaterial(void);
extern  void *OPS_NewStressDensityMaterial(void);
extern  void *OPS_NewJ2BeamFiber2dMaterial(void);
extern  void *OPS_NewJ2PlateFibreMaterial(void);

extern void *OPS_NewLinearCap(void);
extern void *OPS_NewAcousticMedium(void);

extern  void *OPS_NewFSAMMaterial(void); // K Kolozva

#ifdef _HAVE_Damage2P
extern void *OPS_Damage2p(void);
```

Next, we need to add a few lines to the TclModelBuilderNDMaterialCommand.cpp file located in the nDmaterial directory. These commands have to do with the interpreters.

The OPS_NewBoundingCamClayMaterial name should match what we used in the corresponding part of our implementation

Adding New Material to OpenSees

QuakeCoRE
NZ Centre for Earthquake Resilience

TclModelBuilderNDMater...es/SRC/material/nD) - VIM

```
    void *theMat = OPS_CycLiqCPSPMaterial();
    if (theMat != 0)
      theMaterial = (NDMaterial *)theMat;
    else
      return TCL_ERROR;
  }

  else if ((strcmp(argv[1],"BoundingCamClay") == 0)){

    void *theMat = OPS_NewBoundingCamClayMaterial();
    if (theMat != 0)
  theMaterial = (NDMaterial *)theMat;
    else
  return TCL_ERROR;
  }

  else if ((strcmp(argv[1],"ManzariDafalias") == 0)){

    void *theMat = OPS_NewManzariDafaliasMaterial();
    if (theMat != 0)
  theMaterial = (NDMaterial *)theMat;
    else
  return TCL_ERROR;
  }

  else if ((strcmp(argv[1],"ManzariDafaliasRO") ==

    void *theMat = OPS_NewManzariDafaliasMaterialRO
    if (theMat != 0)
  theMaterial = (NDMaterial *)theMat;
    else
  return TCL_ERROR;
  }

  else if ((strcmp(argv[1],"ContactMaterial2D") ==

    void *theMat = OPS_NewContactMaterial2DMaterial
    if (theMat != 0)
```

Next, we need to add a few lines to the TclModelBuilderNDMaterialCommand.cpp file located in the nDmaterial directory. These commands have to do with the interpreters.

The OPS_NewBoundingCamClayMaterial name should match what we used in the corresponding part of our implementation

Adding New Material to OpenSees

QuakeCoRE
NZ Centre for Earthquake Resilience

classTags.h (~/OpenSees/SRC) - VIM

```
// Contact Material - P.Arduino
#define ND_TAG_ContactMaterial2D                14001
#define ND_TAG_ContactMaterial3D                14002
// Drucker-Prager - P.Arduino
#define ND_TAG_DruckerPrager                    14003
#define ND_TAG_DruckerPragerThreeDimensional    14004
#define ND_TAG_DruckerPragerTensionCutoff       14005
#define ND_TAG_DruckerPrager3D                  14006
#define ND_TAG_DruckerPragerPlaneStrain         14007
// CamClay with Bounding Surface - C.McGann
#define ND_TAG_BoundingCamClay                  14008
#define ND_TAG_BoundingCamClay3D                14009
#define ND_TAG_BoundingCamClayPlaneStrain       14010
// Initial state analysis material wrapper - C.McGann
#define ND_TAG_InitialStateAnalysisWrapper      14011
// Manzari Dafalias material - P. Arduino
#define ND_TAG_ManzariDafalias                  14012
#define ND_TAG_ManzariDafalias3D                14013
#define ND_TAG_ManzariDafaliasPlaneStrain       14014
// Manzari Dafalias material - A. Ghofrani
#define ND_TAG_ManzariDafaliasRO                14015
#define ND_TAG_ManzariDafalias3DRO              14016
#define ND_TAG_ManzariDafaliasPlaneStrainRO     14017
// Stress-Dilatancy material - C.McGann
#define ND_TAG_StressDensityModel               14018
#define ND_TAG_StressDensityModel2D             14019
#define ND_TAG_StressDensityModel3D             14020

// MultiaxialCyclicPlasticity, add by Gang Wang
#define ND_TAG_MultiaxialCyclicPlasticity
#define ND_TAG_MultiaxialCyclicPlasticity3D
#define ND_TAG_MultiaxialCyclicPlasticityAxiSymm
#define ND_TAG_MultiaxialCyclicPlasticityPlaneStrain

#define ND_TAG_ConcreteMcftNonLinear5 7601
#define ND_TAG_ConcreteMcftNonLinear7 7602
```

Next, we need to create a unique set of tags for our new class (and any subclasses) and add these to the classTags.h file located in the SRC directory.

It doesn't matter what the tag is as long as it is unique.

Adding New Material to OpenSees

QuakeCoRE
NZ Centre for Earthquake Resilience

Makefile (~/OpenSees/SRC) - VIM

```
    $(FE)/material/nD/CycLiqCPSP3D.o \
    $(FE)/material/nD/CycLiqCPSPPlaneStrain.o \
    $(FE)/material/nD/UWmaterials/DruckerPrager.o \
    $(FE)/material/nD/UWmaterials/DruckerPrager3D.o \
    $(FE)/material/nD/UWmaterials/DruckerPragerPlaneStrain.o \
    $(FE)/material/nD/UWmaterials/ContactMaterial3D.o \
    $(FE)/material/nD/UWmaterials/BoundingCamClay.o \
    $(FE)/material/nD/UWmaterials/BoundingCamClay3D.o \
    $(FE)/material/nD/UWmaterials/BoundingCamClayPlaneStrain.o \
    $(FE)/material/nD/UWmaterials/ManzariDafalias3D.o \
    $(FE)/material/nD/UWmaterials/ManzariDafaliasPlaneStrain.o \
    $(FE)/material/nD/UWmaterials/ManzariDafaliasRO.o \
    $(FE)/material/nD/UWmaterials/ManzariDafalias3DRO.o \
    $(FE)/material/nD/UWmaterials/ManzariDafaliasPlaneStrainRO.o \
    $(FE)/material/nD/UWmaterials/InitialStateAnalysisWrapper.o \
    $(FE)/material/nD/LinearCap.o \
    $(FE)/material/nD/AcousticMedium.o \
    $(FE)/material/nD/FeapMaterial.o \
    $(FE)/material/nD/feap/FeapMaterial01.o \
    $(FE)/material/nD/feap/matl01.o \
    $(FE)/material/nD/feap/FeapMaterial02.o \
    $(FE)/material/nD/feap/matl02.o \
    $(FE)/material/nD/feap/FeapMaterial03.o \
    $(FE)/material/nD/feap/matl03.o \
    $(FE)/material/nD/feap/feapCommon.o \
    $(FE)/material/nD/PressureDependentElastic3D.o \
    $(FE)/material/nD/ElasticIsotropicMaterial.o \
    $(FE)/material/nD/ElasticIsotropicThreeDimensional.o \
    $(FE)/material/nD/ElasticIsotropicPlaneStress2D.o \
    $(FE)/material/nD/ElasticIsotropicPlaneStrain2D.o \
    $(FE)/material/nD/ElasticIsotropicAxiSymm.o \
    $(FE)/material/nD/ElasticIsotropicPlateFiber.o \
    $(FE)/material/nD/ElasticIsotropicBeamFiber.o \
    $(FE)/material/nD/ElasticIsotropicBeamFiber2d.o \
    $(FE)/material/nD/ElasticOrthotropicMaterial.o \
    $(FE)/material/nD/ElasticOrthotropicThreeDimensional.o \
```

For Linux or Mac builds, we also need to add our new class and subclasses to the MATERIAL_LIBS definition in the SRC/Makefile

Adding New Material to OpenSees

QuakeCoRE
NZ Centre for Earthquake Resilience

FEM_ObjectBrokerAllClasses.cpp (~/OpenSees/SRC/actor/objectBroker) - VIM1

```
#include <PlaneStressUserMaterial.h>
//end Yuli Huang & Xinzheng Lu
#include <FeapMaterial03.h>
#include <CycLiqCP3D.h>
#include <CycLiqCPPlaneStrain.h>
#include <CycLiqCPSP3D.h>
#include <CycLiqCPSPPlaneStrain.h>


#include <FluidSolidPorousMaterial.h>
#include <PressureDependMultiYield.h>
#include <PressureDependMultiYield02.h>
#include <PressureIndependMultiYield.h>

#include <ContactMaterial2D.h>
#include <ContactMaterial3D.h>
#include <DruckerPrager3D.h>
#include <BoundingCamClay.h>
#include <BoundingCamClay3D.h>
#include <BoundingCamClayPlaneStrain.h>
#include <ManzariDafalias3D.h>
#include <ManzariDafaliasPlaneSt
#include <ManzariDafaliasRO.h>
#include <ManzariDafalias3DRO.h>
#include <ManzariDafaliasPlaneSt
#include <InitialStateAnalysisWr
#include <StressDensityModel.h>
#include <StressDensityModel2D.h
#include <StressDensityModel3D.h

// Fibers
#include <UniaxialFiber2d.h>
#include <UniaxialFiber3d.h>

// friction models
#include <Coulomb.h>
```

Finally, we need to add our new classes as includes in SRC/actor/objectBroker/FEM_ObjectBrokerAllClasses.cpp to allow our new material to be used in parallel processing or by the database commands.

Once these steps have been completed, we should now be able to compile or build our local OpenSees and test out our new material implementation.

QuakeCoRE
NZ Centre for Earthquake Resilience

Thank you!

www.quakecore.nz

UC University of Canterbury Te Whare Wānanga o Waitaha Christchurch New Zealand
The University of Auckland Te Whare Wānanga o Tāmaki Makaurau New Zealand
Resilient Organisations
GNS Science Te Pū Ao
Massey University
Te Whare Wānanga o te Upoko o te Ika a Maui Victoria University of Wellington
The University of Waikato Te Whare Wānanga o Waikato
BRANZ