Personal Report

Documentation for ConFEM2D program

- A FE software written in Python 2.7 -

Contents

Introduction

There are many different commercial FEA softwares largely used in design of RC structures that work very well in linear elastic calculation. Unlike linear calculation, the nonlinear calculation is not largely applied yet because of its complexities and time consuming. With two years experiences as structural engineer mainly working on the modeling of RC structure, I mainly used ANSYS and ATENA software for the purpose of nonlinear structural analysis. These commercial softwares are very universal and trustworthy but they are expensive, obviously close-source and sometimes user was limited in the choice of nonlinear concrete material. An inconvenient of using these software is that user doesn't really know how the nonlinear calculation works under the hood. These limitations and my intermediate's knowledge of nonlinear FEA solver was leading my intention to contribute into a open-source finite element analysis project, in my opinion, it's the best way to learn and to practice. And the one I chose is the **ConFem**[1] FE library provided by prof. ***Ulrich-Haussler Combe***. The source code is written in Python programming language and in Oriented-Object Programming style *(OOP)*. The *OOP* style makes the source code easier for maintain and for development. The things that convinced me to join in this project is its rich material libraries focusing on reinforcement concrete material and the good book [5] supporting alongside. **ConFEM2D** is a restricted version that mainly focuses on 2D plate and shell element types. I'm working on this open source project to develop/implement new material, and to improve the code performance. The project is now live on :
https://github.com/DuongDoNgoc/ConFEM2D [8]

This document is organized into five sections. The objective of Section 1 is to present the front end of the software. Users will be presented the input data and see the what the software can do as output. As an RC structural engineers, I show only examples related to RC structures as illustration ☺. Section 2 presents how the global system stiffness matrix is built. In finite element model, the system stiffness matrix is largely sparse, so it need a special treatment to take advance of this characteristic in computer's storage. Section 3 presents the efficient methods used for solving a nonlinear problem. A specific method might be good for a specific problem but might be not for others. How to choice the suitable solver will be discussed. Section 4 focuses on nonlinear concrete materials. The linear elastic behavior will be firstly presented as the basic, then three most efficient nonlinear materials for modeling RC structure will be presented with its range of use. Because of the highly nonlinear behavior of concrete, great care and trial and errors are necessary for obtain solution to practical problems. The last Section is for remaining and potential points to improve the software because it is still a working project.

[1]The official website of ConFem software is : *http://concrete-fem.com*

1 Input Data

The software is written as Python 2.7 source code, whereby using parts of *NumPy, SciPy and Matplotlib* packages. User needs to define a finite element model in a *"project-name.in.txt"* file. This input file contains all information for finite element model like nodes, elements, material, boundary conditions, step and the appropriate solver chosen by user.
The data of a particular problem are given with plain ASCII-files with a naming scheme: *<name>.<type>.txt*. The types for the problem data are as follows, see table 1:

Type	Description	User defined input	Module written output
input	Model data	Mandatory	–
opt	optional for written files	optional	–
plt	Scaling factors for plots	optional	–
protocol	Protocol of computation	–	always
elemout	Results for elements integrations points	–	always
nodeout	Results for nodes	–	always
timeout	Results for every time increment for selected nodes	–	if opt exists

Table 1: Type of files

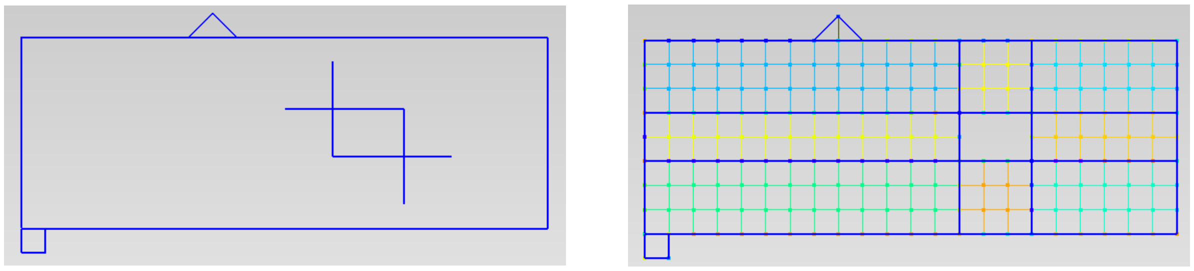
To run the program using source code, edit the script `ConFEM2D_main.py` in the final section:

```
if __name__ == "__main__":
    Name="../path/project-name"                # input data name
    ...
```

As an end user, the heavy works are in modeling and in generating the finite element mesh. Among external software specifying in creating finite element mesh, I choose to use **Gmsh** *(version 4.0)* software because of its simplicity, rapidity and portability. Furthermore, Gmsh4.0 provides a Python API which permits directly interact via python console. A derivative python package called *'pygmsh'* can also create/modify efficiently the FE model in python code. After having a ready-to-use finite element mesh generated from Gmsh, the assignment of element type, material type, boundary conditions, solver and of time increment steps can be easily editable in the input data sample file via the relative predefined keywords.

1.1 Preparing finite element mesh with Gmsh4.0

Gmsh is a three-dimensional finite element grid generator with a build-in CAD engine and post-processor. Its design goal is to provide a fast, light and user-friendly meshing tool with parametric input and advanced visualization capabilities.



(a) Geometry designed in Gmsh4.0 (b) Mesh generated by Gmsh4.0

Figure 1: A finite element model of a deep RC beam with opening

Gmsh is built around four modules: ***geometry, mesh***, solver and post-processing. All geometrical, mesh, solver and post-processing instructions are prescribed either interactively using the graphical user interface (GUI) or in text files using Gmsh's own scripting language. Interactive actions generate language bits in the input files, and vice versa. This makes it possible to automate all treatments, using loops, conditionals and external system calls. [1]
A very know weakness of Gmsh is : " Gmsh is not a multi-bloc mesh generator: all meshes produced by Gmsh are conforming in the sense of finite element meshes." This raises difficulties in the modeling of reinforced concrete structure where concrete and reinforcements are usually tied together by bond elements. Gmsh is pretty good at generate 1D, 2D, 3D finite element meshes, but when mesh blocks intersect or are tied together Gmsh doesn't provide auto-detecting interfaces or merging of coincided nodes function.
By design, Gmsh does not try to incorporate every possible definition of boundary conditions or material properties—this is a job best left to the solver. Instead, Gmsh provides a simple mechanism to tag groups of elements, and it is up to the solver to interpret these tags as boundary conditions, materials, etc. Associating tags with elements in Gmsh is done by defining physical groups (Physical Points, Physical Curves, Physical Surfaces and Physical Volumes).
Once the meshing is done, the output MSH file format give full information of the mesh. The MSH file format (version 4) contains one mandatory section giving information about the file (*$MeshFormat*), followed by several sections defining the physical group names (*$PhysicalName*, optional), the entities (*$Entities*), the partitioned entities (*$PartitionedEntities*, optional), nodes (*$Nodes*), elements (*$Elements*). These mesh information will be then filtered by a small python program called *FilterGmsh.py* which returns necessary information for ConFEM2D's input data such as nodal data, element data, boundary conditions data, load data.

1.2 Key word for Input Data

The input file is organized through *sections*. Every sections starts with a *section keyword* which may be followed by *options* in the same line. Options and keyword are generally separated by commas. The following

Section	Descriptions
∗NODE	Nodal data
∗SOLID SECTION	Definitions for trusses, plates, with reference to material types end element types
∗SHELL SECTION	Definitions of slab and shell structures with reference to material types end element types
∗MATERIAL	Material types and their parameters
∗ELEMENT	Element data
∗STEP	Definitions for the incrementally iterative procedure, boundary conditions, loading

Table 2: Key words for input data

section keyword are currently used, see table 2.

A line with leading '∗∗' may be used as comment line and is ignored. Leading blanks, intermediate blanks and blank lines are generally ignored.

The section keyword ∗STEP has no own data lines, but has own keywords which are currently the following:

∗STATIC, (RIKS) : quasi-static computation

Δ_t, t, s

RIKS(optional) : arc-length method is used for step size of load incrementation

Δ_t = time step for load incrementation (pseudo-time)

t = time target

s = prescribed scalar length of displacement vector in case of arc length method

Default method for solution of systems of equations is the Newton-Raphson method in case that a line with ∗SOLUTION TECHNIQUE is not given.

∗SOLUTION TECHNIQUE, TYPE=MODIFIED-NR/QUASI-NEWTON

Linear problem are treated like nonlinear problems but will generally stop after the first equilibrium iteration.

This ConFem's document provides completed instructions for Keywords :

`http://concrete-fem.com/wp-content/uploads/2017/12/docu/SEC_0500.html`

The following combinations between element types and material types are currently possible, see table 3.

1.3 Optional files

The options file (*.opt.txt) can be used to request specific outputs. There are three kinds of output that can be requested, depending on the type of analysis:

- writeNodes - output nodal displacements and forces over time
- LineSearch - line search iteration method

Material	T1D2	T2D2	T3D2	S1D2	CPE3	CPE4	CPS3	CPS4	SB3	SH4	SH3
ELASTIC	+	+	+	+	+	+	+	+	+	+	+
ELASTICLT	+	+	+	-	+	+	+	+	-	+(*)	+(*)
ISODAMAGE	+	+	+	-	+	+	+	+	-	+(*)	+(*)
MISES	+	+	+	-	+	+	+	+	-	+	+
MISESREMEM	-	-	-	-	+	+	+	+	-	-	-
SPRING1D(bond)	-	-	-	+	-	-	-	-	-	-	-
NLSLAB	-	-	-	-	-	-	-	-	+	-	-

Table 3: Possible combinations between element types and material types

- reinforcementDesign - for use with ConPlaD

1.4 Examples

The following examples presents the capabilities in nonlinear calculation of ConFEM2D software. More examples are planning to be added. The files corresponding to these examples are available in this link [8].

1.4.1 Example 1: Reinforcement design for a deep beam with linear elastic internal force

The system and loading is illustrated in figure 2. Young's modulus and Poisson's ratio are chosen with $E = 31900 MN/m^2, \nu = 0.2$ for a linear elastic calculation. The reinforcement yield strength is assumed with $f_{yk} = 500 MN/m^2$. The width of the deep beam is $b = 0.6m$. Safety factors are not regarded. Plane stress conditions are assumed.

The basic procedure is to firstly perform a linear elastic calculation for internal forces, then followed by a

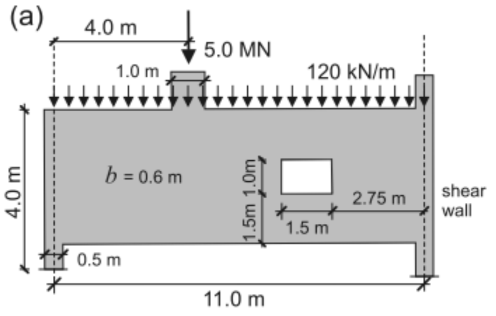


Figure 2: Deep Beam's system and loading

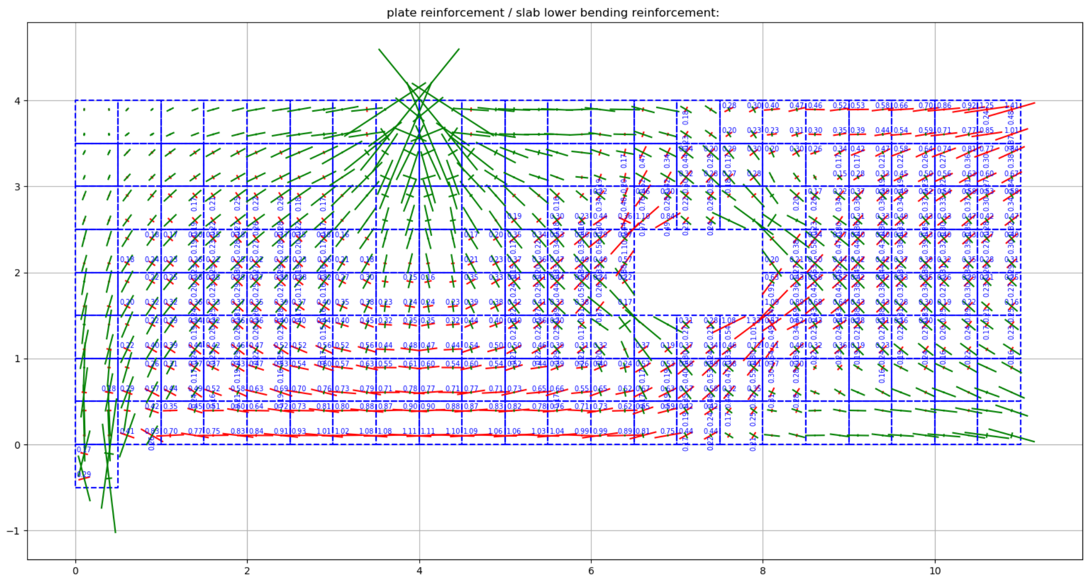


Figure 3: Reinforcement design of deep beam

reinforcement design and concrete proof with methods of limit analysis. A state of stress with components $\sigma_x, \sigma_y, \sigma_{xy}$ is calculated for each integration point of each element and the design procedure is performed for all these points. The results of reinforcement design and principal stress vectors are given in figure 3. The minimum reinforcement ratio is also chosen with $\rho_{x,min} = \rho_{y,min} = \rho_{min} = 0.15\%$. This minimum ratio serves as a base for supplementary reinforcement.

1.4.2 Example 2: Simulation of cracked reinforced deep beam

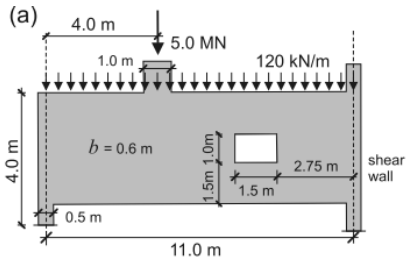


Figure 4: System and loading

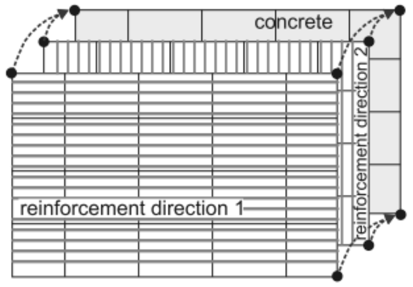
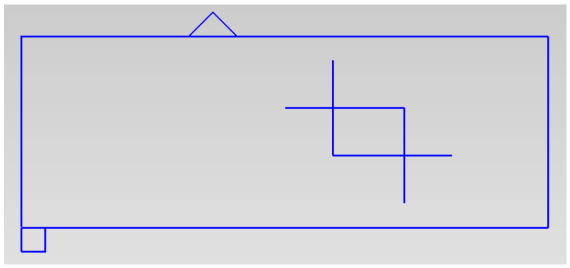


Figure 5: Overlay of elements

We refer to the previous deep beam example given in fig 2. In contrast of a linear elastic calculation of internal force as treated in example 1, the nonlinear stress-strain behavior of concrete and reinforcement will be considered to some extent. The following material properties are assumed:

- The concrete material is chosen as a ElasticLT material with a uniaxial tensile strength limit $f_{ct} = 1.0 MN/m^2$
- Crack energy has a magnitude of roughly $100 Nm/m^2$. A value of $G_f = 50 Nm/m^2$ and a crack band width $b_w = 0.02m$ are chosen. This leads to a critical crack width $w_{cr} = 0.1mm$.
- The reinforcement material model with parameters: Young's modulus $E_s = 200000 MN/m^2$, initial yield limite $f_{yk} = 500 MN/m^2$, ultimate strength $f_t = 550 MN/m^2$, strain at ultimate strength $\epsilon_u = 0.05$, hardening modulus $E_T = 1053 MN/m^2$.
- The idealized reinforcements distributing in x-, y-directions are modeled as overlay sheets of the concrete plate, see fig 5. Furthermore, two sire bars are added to reinforce the opening.
- The loading is applied by displacement control of a node where a concentrated load acts upon. Distributed load are neglected. In contrast to load control, the control of displacements allows to model limit states of a structure, i.e., limited loading with increasing displacements.

The discretization is shown in fig 1. Special care has to be given for the lower left-hand support. A nodal support of the RC deep beam itself is not appropriate as this leads to concentrated hight tensile stresses. A more realistic approach is given with a support by an extra element, whereby this element is assumed as


(a) Geometry designed in Gmsh4.0

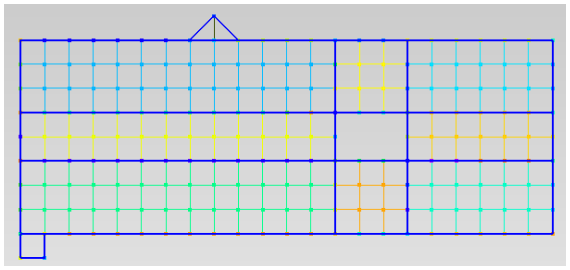
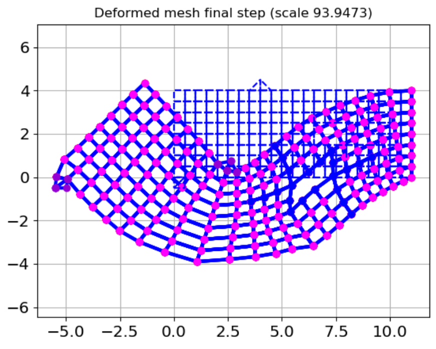

(b) Mesh generated by Gmsh4.0

Figure 6: Finite element model of RC deep beam

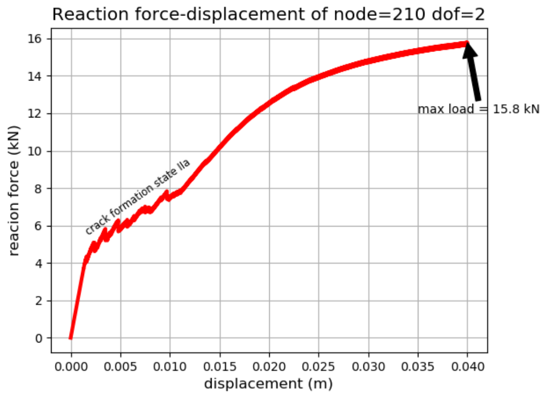
linear elastic with $E_c = 31900MN/m^2$,$\nu = 0.2$ and with a vertical nodal support of its lower two nodes. This support may move freely in the horizontal direction.

The problem is physically nonlinear due to concrete cracking and elasto-plastic reinforcement behavior and requires an incrementally iterative approach. Cracking will start in an early stage of loading. In the case of a detection of a new crack, an equilibrium iteration sequence is performed without applying a load increase. The resulting load-displacement curve is shown in figures 7, 8. Four stages can be observed as is typical for reinforced concrete:

- Uncracked state I
- Crack formation state IIa with elastic reinforcement behavior.
- Stabilized cracking state IIb with elastic reinforcement behavior.
- Stabilized cracking state III with yielding reinforcement.
- Post-pic after the limit load is achieved.

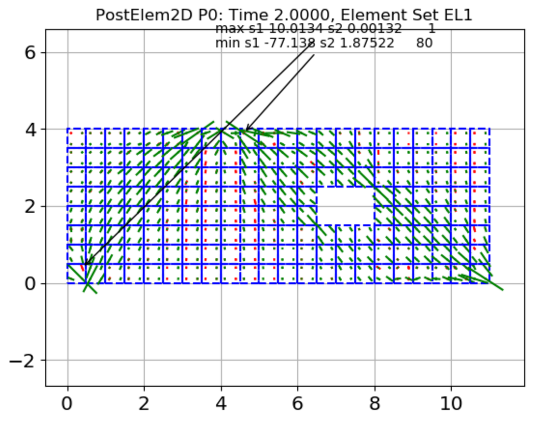


(a) Deformed mesh

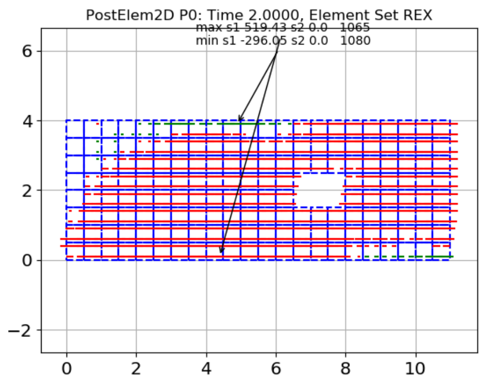


(b) Reaction force - displacement curve

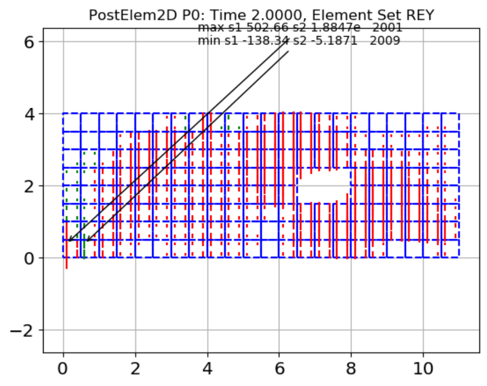
Figure 7: Results of example 2



(a) Principal stress flow of concrete elements



(b) Stress of distributed reinforcement in x-direction



(c) Stress of distributed reinforcement in y-direction

Figure 8: Results of example 2

1.4.3 Example 3: Nonlinear calculation for a simple RC shell slab

Remain to add

1.4.4 Example 4: Elasto-plastic rectangular slab with opening and free edges

The slab's geometry with opening and boundary conditions is shown in figure 9. A single support is given in the lower left corner. The left and the lower edges are not supported. The upper and right edges are simply supported (hinged). The slab thickness is $h = 0.3m$. The material properties are assumed with a Young's modulus $E = 31900MN/m^2$ and Poisson's ratio $\nu = 0.2$, these will be used as the linear elastic calculation to compare with a nonlinear elasto-plastic calculation afterward. A uniform loading is given by $p = 10kN/m^2$. Assumed a pre-calculated reinforcement design somehow gives the following parameters:

- initial bending stiffness $K_{s,x} = K_{s,y} = 5.8MNm^2/m$

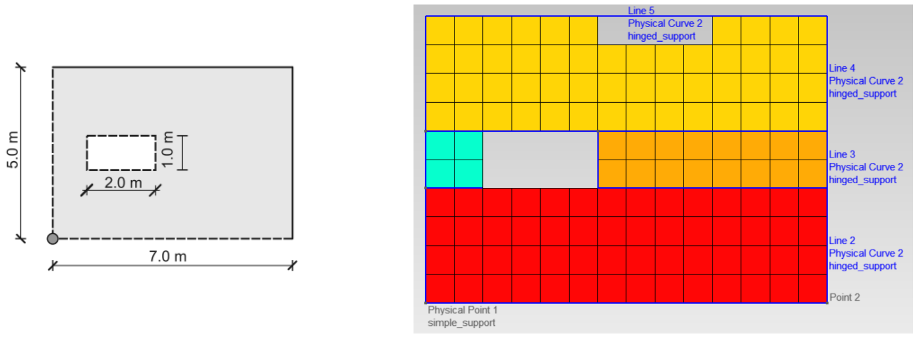
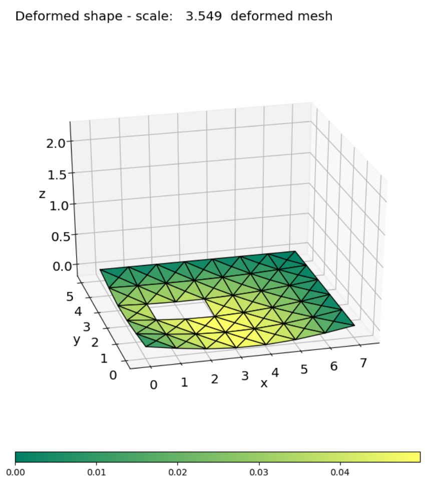


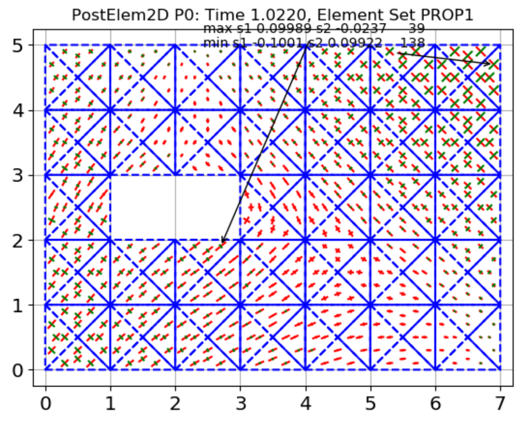
Figure 9: Geometry and Discretization of a slab with opening

- initial yielding moment $m_{y,x} = m_{y,y} = 0.054 MNm/m$
- hardening bending stiffness $K_{T,x} = K_{T,y} = 0.12 MNm^2/m$

An incrementally iterative scheme with Newton-Raphson iteration within each loading increment is used as the solution method. The computed principal moments and deflection for the final loading are shown in figure 10. The maximum displacement increase by a factor of approximately 10 times compared to the linear elastic case. This corresponding to bending stiffness relation which are $72.5 MNm/m_2$ for the linear elastic case and $5.8 MNm/m_2$ for the elasto-plastic case. Slight redistribution of moments can be observed. Moments m_x, m_y slightly exceed (due to hardening) the prescribed yield limit of $m_{y,x} = m_{y,y} = 0.054 MNm/m$ while maximum values are in range of $0.065 MNm/m$ in case of the linear elastic calculations. Using this simplified elasto-plastic 'NLSlab' material gives a potential for moment redistribution compared to the linear elastic case.



(a) Deformed mesh



(b) Flow of principal moments

Figure 10: Results of example 4

2 Building the system stiffness matrix

The final results of this phase are the required structure matrices for the solution of the system equilibrium equations. In a static analysis, the computer program needs to calculate the structure stiffness matrix and the load vectors. In a dynamic analysis, the program must also establish the system mass and damping matrices. In the implementation to be described here, the calculation of the structure matrices is performed as follows:

1. The nodal point and element information are read and /or generated.
2. The element stiffness matrices, mass matrices and equivalent nodal loads are calculated.
3. The structure matrices K, M, C, and R, whichever are applicable, are assembled.

The program organization during the 2nd phase of calculating the element stiffness matrices consists of calling the appropriate element subroutines for each element. During the element matrix calculations, the element coordinates, properties, and load sets, which have been read and stored in the first phase, are needed. After calculation, either an element matrix would be stored on backup storage, because the assemblage into the structure matrices is carried out later.
The assemblage process for obtaining the structure matrix K is symbolically written

$$K = \sum_{i=1}^{end} K^{(i)}$$

where the matrix $K^{(i)}$ is the stiffness matrix of the ith element and the summation goes over all elements in the assemblage. In an analogous manner, the structure mass matrix and load vectors are assembled from the element mass matrices and element load vectors, respectively. In addition to element stiffness, mass and load matrices, concentrated stiffnesses, masses and loads corresponding to specific degrees of freedom can also be added. It should be noted that the element stiffness matrices $K^{(i)}$ are the same order as the structure stiffness matrix K. However, considering the internal structure of the matrices $K^{(i)}$, nonzero elements are in only those rows and columns that correspond to element degrees of freedom. Therefore, in pratice, we only need to store the compacted element stiffness matrix [2]. This is where the concept of sparse matrix come into handy.
The figure below shows the element pattern of a typical stiffness matrix, see figure 11. The matrix is symmetric, banded and highly sparsed.

2.1 Sparse matrix

What is a sparse matrix ? A **sparse matrix** is simply a matrix with a large number of zeros values. In contrast, a matrix where many or most entries are non-zero is said to be dense. Sparse matrices higly arise in FD and FEM methods. There are a variety of sparse matrix formats which are designed to exploit different

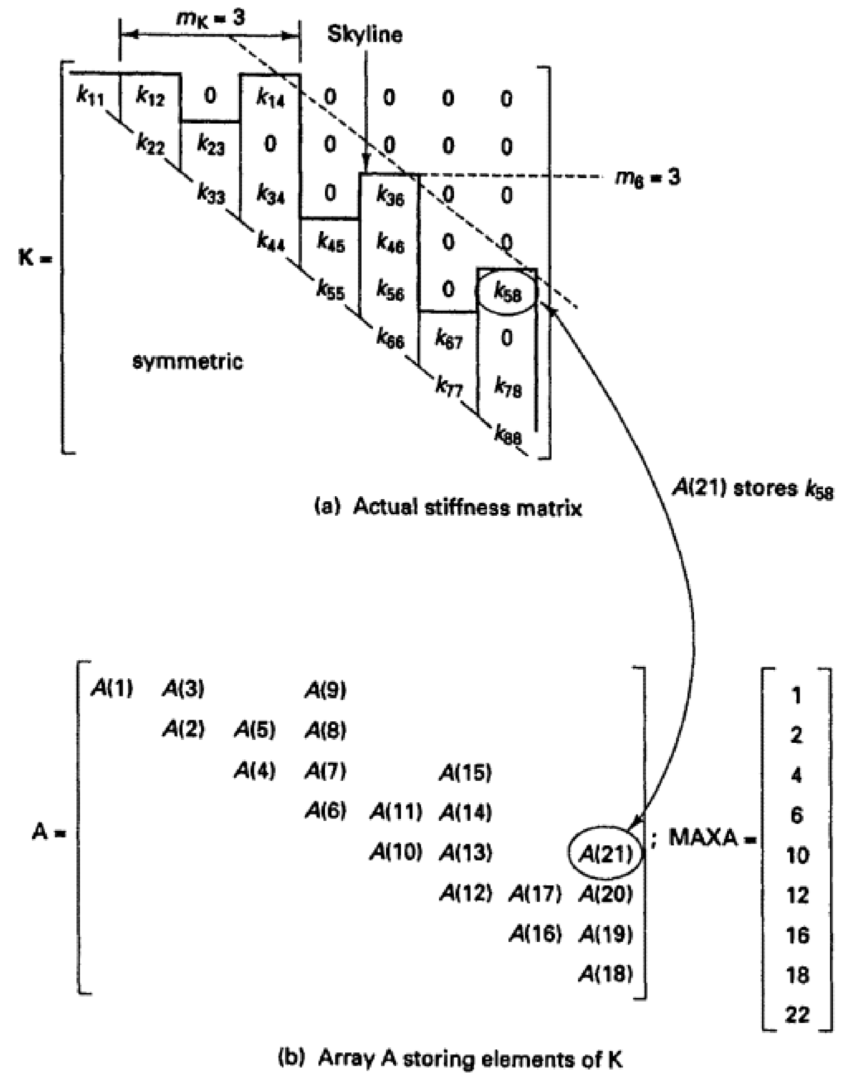


Figure 11: Storage scheme used for a typical system stiffness matrix.

sparsity patterns and different methods for accessing and manipulating matrix entries. The Python's SciPy package provides scipy.sparse module for efficiently store data structures for 2D sparse matrices. There are several avaiable sparse matrix formats such as : `csc_matrix` -Compressed Sparse Column, `csr_matrix` -Compressed Sparse Row, `lil_matrix` -List of Lists, `coo_matrix` -Coordinate,... Each sparse format has certain advantages and disadvantages. For instance, adding new non-zero entries to a `lil_matrix` is fast, however changing the sparsity pattern of a `csr_matrix` requires a significant amount of work. On the other hand, operations such as matrix-vector multiplication and matrix-matrix arithmetic are much faster with `csr_matrix` than `lil_matrix`. A good strategy is to construct matrices using one format and then convert them to another that is better suited for efficient computation.[3]

ConFEM2D program uses `lil_matrix` scheme for building the system stiffness matrix from scratch, then convert it to `csc_matrix` for efficient computation of LU decomposition.

2.2 Reordering sparse matrix by Cuthill-McKee algorithm

To solve sparse linear systems (this problem will be discussed in more details in the next chapter of Solver for nonlinear problem), the direct method is chosen. The direct method can be thought of as a extension of dense matrix factorization, like the LU factorization, to sparse matrices. The goal of such algorithm is to produce factors (e.g. matrices L and U) that are as sparse as possible. The sophisticated Cuthill-McKee algorithm are used to reorder rows and columns of the input matrix so that the **fill-in** of sparse factors is minimized. It is an algorithm to permute a sparse matrix that has a symmetric sparsity pattern into a band matrix form with a small bandwidth, thus making it better suited for use in linear solvers. Largely applied in FEM, the algorithm reduces bandwidth of a matrix by reordering nodes in a mesh in the degree order. This figure below[2] shows the original adjacency graph and the new RCM ordered grapgh as matrices, see figure 12.

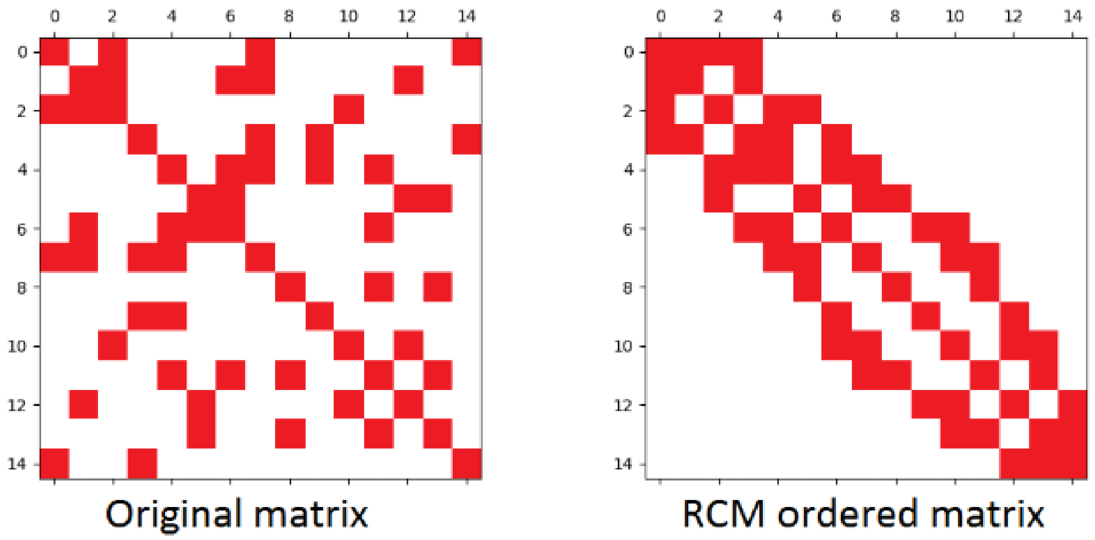


Figure 12: Original and Ordered matrix.

SciPy python's package provide the function *scipy.sparse.csgraph.reverse_cuthill_mckee* which returns the permutation array that orders a sparse CSR or CSC matrix in Reverse-Cuthill McKee ordering. Once the system global stiffness matrix is fully built, we can use this function to find the permutation array of the system stiffness matrix, then proceed the permutation by simply using Numpy fancy indexing. The permutation need also to be applied to load vector and affects the resulting displacement vector. A more efficient way to use the Cuthill-McKee algorithm instead of using it after the system stiffness matrix is fully

[2]For further detail, please visit: http://www.juliafem.org/examples/2017-08-29-reordering-nodes-with-the-RCM-algorithm

built is apply it before building of the system stiffness matrix. That means the Cuthill-McKee algorithm will be applied right after the nodal point and element information are read/generated. It requires to directly write the algorithm in the source code. Thankfully, there are many online resources that help to build it, I would refer to the code snippet for finding the Reverse Cuthill-McKee (RCM) ordering of a symmetric sparse matrix in CSR or CSC format using Cython, which is a part of the QuTiP: Quantum Toolbox in Python package. [4]

3 Solver for nonlinear problem

The nonlinearity is due to nonlinear material behavior. Nonlinear material behavior is not only characteristic for cracked reinforced concrete but also occurs with all other solid materials at the latest with approaching strength. Discretization of nonlinear problems in space and time leads to a form :

$$r(u) = p - f(u)$$

with a *residual* **r**, *internal force* **f** depending on *displacement* **u** and *external loads* **p** which are assumed to be independent of u. The general case is nonlinear dependence of **f** on **u**.

3.1 Newton-Raphson's method

A linear Taylor expansion is used as a basic approach:

$$r(u^{(i)} + \delta u) \approx r(v^{(i)}) + {K_T}^{(i)}.\delta u = 0$$

with a tangential stiffness matrix

$${K_T}^{(i)} = -\left.\frac{\partial r}{\partial u}\right|_{u=u^{(i)}} = \left.\frac{\partial f}{\partial u}\right|_{u=u^{(i)}}$$

leading to the *Newton-Raphson method* :

$$\delta u = \left[{K_T}^{(i)}\right]^{-1}.r_{(u^{(i)})}$$

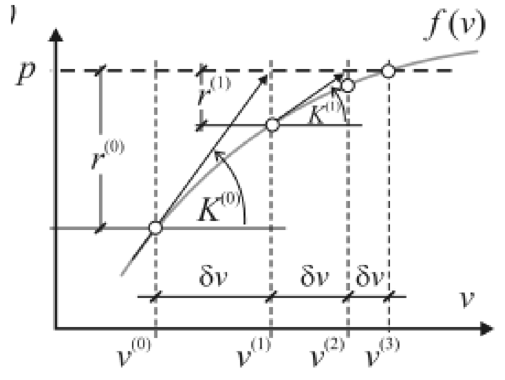
$$u^{i+1} = u^i + \delta u$$

with (hopefully) an improved value $u^{(i+1)}$. Iteration may stop if $\|r(u^{(i+1)})\| \ll 1$ and $\delta u \ll 1$ with a suitable norm $\|.\|$ transforming a vector into a scalar. The method generally has a fast convergence but is relatively costly. The tangential stiffness matrix has to be computed in *every iteration step* (i) and a decomposition in order to solve (LU decomposition instead of inversion) has to be performed on it to determine δu.
Iterative methods like Newton-Raphson are embedded in an incrementally iterative scheme. Thus, loading is given as a history: $p = p(t)$. An appropriate choice is $0 \leq t \leq 1$ for the scaling of the load history (pseudo-)time, which is different from real time in the case of a quasi-static analysis. The following steps are performed in the incrementally iterative scheme, see figure 13:

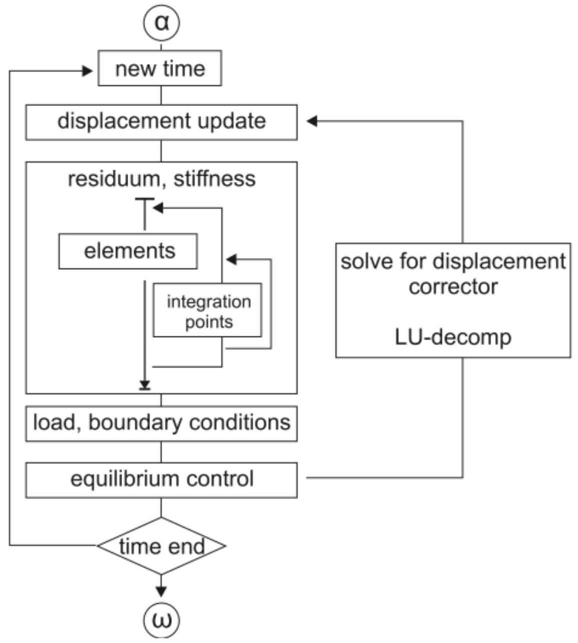
1. Discrete time values t_k are regarded with a time step $\Delta t = t_{k+1} - t_k$ and an initial time $t_0 = 0$. A loading $p_k = p(t_k)$ is prescribed for all time steps. The incremental material law $\dot{\sigma}(t) = C_T.\dot{\epsilon}(t)$ is integrated by a numerical integration of stresses and strains. Denotes $\sigma_k = \sigma(t_k)$, $\epsilon_k = \epsilon(t_k)$.

$$\sigma_{k+1} = \sigma_k + C_{T,k+1}.(\epsilon_{k+1} - \epsilon_k)$$

where $C_{T,k+1}$ indicates dependence of the tangential material stiffness on σ_{k+1} and/or ϵ_{k+1}.



(a) Incremental Newton-Raphson procedure



(b) Flow of displacement based nonlinear calculation

Figure 13: Newton-Raphson method

2. Unknowns are nodal displacement $u_k = u(t_k)$ leading to strain ϵ_k, stresses σ_k and internal nodal forces f_k except the initial state. This state described by u_0, ϵ_0, σ_0, f_0 is assumed to be known and initial equilibrium is given by $r_0 = p_0 - f_0 = 0$.

3. The solution starts with t_1 and u_1 has to be determinated. This is performed with an iteration $u_1^{(0)}, \ldots, u_1^{(i)}$ with, e.g., the Newton-Raphson method using an initial $u_1^{(0)} = u_0$. The iteration involves $\epsilon_1^i, \sigma_1^i, C_{T,1}^i$.

4. A converged u_1 and the corresponding strains ϵ_1 and stresses σ_1 serve as a base for t_2 and so on until a target time is reached.

To calculate the inverse K_T^{-1} of the tangential stiffness matrix K_T, a *LU decomposition* is performed efficiently numerical computation. The matrix K_T is decomposed into

$$K_T = L.U, L, U \in R^{nxn}$$

with a lower triangular matrix L with components $L_{ij} = 0 if j > i$ and $L_{ii} = 1$ and an upper triangular matrix U with components $U_{ij} = 0 if j < i$. Thus, the task of a particular iteration is reformulated as

$$L.\omega = r, U.\delta u = \omega$$

or a sequence of forward and backward substitutions which for given L,U,r is computationally relatively inexpensive due to the triangular structure of L,U. The LU decomposition is performed only once for a iteration sequence. The scalar version to compute the incremental displacement :

$$\delta u = \left[K_T^{(i=0)}\right]^{-1}.r(u^{(i)}$$

However, the convergence of the iterative approach to solve linear equations cannot be guaranteed. The convergence will not be reach for "*rough*" dependence of **f** on **u**. In cases of rough f-u behavior, i.e., the tangential stiffness does not approximately indicate the actual f-u path, the Newton-Raphson method generally will not converge to a solution u with a residual $r \longrightarrow 0$. [5]

3.2 Quasi Newton method- the BFGS's method

A further variation is given with *secant methods* so-called Quasi-Newton method. They derive from an idea of Davidon - to update the matrix K (secant stiffness) in a simple way after each iteration, rather than to recompute it (tangential stiffness) entirely (Newton's method) or leave it unchanged (modified Newton). A popular approach is given with the BFGS method which defines the secant stiffness matrix by :

$$K_S^{(i+1)} = K_S^{(i)} + \frac{\delta r.\delta r^T}{\delta r^T.\delta u} - \frac{K_S^{(i)}.\delta u.\delta u^T.K_S^{(i)}}{\delta u^T.K_S^{(i)}.\delta u}$$

It is known that the BFGS is **most conveniently written in terms of K_S^{-1} rather than K_S**. The inverse is given with the so-called *BFGS normal form* :

$$\left[K_S^{(i+1)}\right]^{-1} = \left(I - \frac{\delta u.\delta r^T}{\delta r^T.\delta u}\right).\left[K_S^{(i)}\right]^{-1}.\left(I - \frac{\delta r.\delta u^T}{\delta r^T.\delta u}\right) + \frac{\delta u.\delta u^T}{\delta r^T.\delta u}$$

with the unit matrix I. This is used with the iteration rule :

$$u^{(i+2)} = u^{(i+1)} + \left[K_S^{(i+1)}\right]^{-1}.r(u^{(i+1)})$$

The computation of sequence of the BFGS normal form may be efficiently implemented on the basics of the LU decomposition of a given $K_S^{(0)}$ - the initial tangential stiffness is appropriate - with a recursion on vector products like $\delta u.\delta r^T$. A description of details can be found in [6].
The BFGS method - with the option to combine it with a so-called *line search* - might converge to a solution **u** with a residual $\mathbf{r} \longrightarrow 0$ in case of rough **f-u** behavior where the Newton-Raphson method fails.

3.3 Line search

The line search option attempts to improve Newton-Raphson solution **u** by scaling the solution vector by a scalar value termed the line search parameter **s**. In some situations, updating the solution $u_k^{(i+1)} = u_k^{(i)} + \delta u$ with full δu leads to solution instabilities. Hence, if the line search option is used, the solution updating is modified to be :

$$u_k^{(i+1)} = u_k^{(i)} + s.\delta u$$

where : s is the line search parameter, $0.2 < s < 1.0$.

s is automatically determined by minimizing the energy of the system, which reduces to finding the zero of the nonlinear equation

$$g_s = \delta u^T.r(u^{(i)} + s.\delta u)$$

where : g_s = gradient of the potential energy with respect to s. An iterative solution scheme is sued to solve this equation. Iterations are continued until either :

1. g_s is less than $0.5g_{(s=0)}$
2. g_s is not changing significantly between iterations.
3. more than six iterations have been performed.

If $g_{(s=0)} < 0$, no iterations are performed and s is set to 1.0. The scaled solution δu is used to update the current DOF values $u^{(i)}$ and the next equilibrium iteration is performed.

4 Material and Element types

The majority information of this chapter is referenced to [5]

4.1 Isotropy and Linearity

From a mechanical point of view, material behavior is primarily focused on strains and stresses. The formal definition of strains and stresses assume a homogeneous area of matter. Regarding the virgin state of solids, their behavior initially can be assumed as linear elastic in nearly all relevant cases. Furthermore, the behavior can be initially assumed as isotropic in many cases,i.e., the reaction of a material is the same in all directions. The following types of elasticity are listed exemplary:

- Uniaxial elasticity

$$\sigma = E\,\epsilon \tag{1}$$

  with uniaxial stress σ, Young's modulus E, and uniaxial strain ϵ.

- Isotropic plane strain is determined with $\epsilon_z = 0$ and

$$\begin{pmatrix}\sigma_x\\ \sigma_y\\ \sigma_{xy}\end{pmatrix} = \frac{E(1-\nu)}{(1+\nu)(1-2\nu)}\begin{bmatrix}1 & \frac{\nu}{1-\nu} & 0\\ \frac{\nu}{1-\nu} & 1 & 0\\ 0 & 0 & \frac{1-2\nu}{2(1-\nu)}\end{bmatrix}\cdot\begin{pmatrix}\epsilon_x\\ \epsilon_y\\ \gamma_{xy}\end{pmatrix} \tag{2}$$

  with stress components $\sigma_x, \sigma_y, \sigma_{xy}$, Young's modulus E, Poisson's ratio ν, and strain components $\epsilon_x, \epsilon_y, \gamma_{xy}$. The lateral stress is given by $\sigma_z = \frac{E\nu}{(1+\nu)(1-2\nu)}(\epsilon_x + \epsilon_y)$.

- Isotropic plane stress

$$\begin{pmatrix}\sigma_x\\ \sigma_y\\ \sigma_{xy}\end{pmatrix} = \frac{E}{1-\nu^2}\begin{bmatrix}1 & \nu & 0\\ \nu & 1 & 0\\ 0 & 0 & \frac{1-\nu}{2}\end{bmatrix}\cdot\begin{pmatrix}\epsilon_x\\ \epsilon_y\\ \gamma_{xy}\end{pmatrix} \tag{3}$$

  ensuring $\sigma_z = 0$ for every combination $\epsilon_x, \epsilon_y, \gamma_{xy}$. The lateral strain is given by $\epsilon_z = -\frac{\nu}{E}(\sigma_x + \sigma_y)$

- Triaxial isotropic linear elastic

$$\begin{pmatrix}\sigma_{11}\\ \sigma_{22}\\ \sigma_{33}\\ \sigma_{23}\\ \sigma_{13}\\ \sigma_{12}\end{pmatrix} = \begin{bmatrix} \frac{E(1-\nu)}{(1+\nu)(1-2\nu)} & \frac{E\nu}{(1+\nu)(1-2\nu)} & \frac{E\nu}{(1+\nu)(1-2\nu)} & 0 & 0 & 0 \\ \frac{E\nu}{(1+\nu)(1-2\nu)} & \frac{E(1-\nu)}{(1+\nu)(1-2\nu)} & \frac{E\nu}{(1+\nu)(1-2\nu)} & 0 & 0 & 0 \\ \frac{E\nu}{(1+\nu)(1-2\nu)} & \frac{E\nu}{(1+\nu)(1-2\nu)} & \frac{E(1-\nu)}{(1+\nu)(1-2\nu)} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{E}{2(1+\nu)} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{E}{2(1+\nu)} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{E}{2(1+\nu)} \end{bmatrix} \cdot \begin{pmatrix}\epsilon_{11}\\ \epsilon_{22}\\ \epsilon_{33}\\ \epsilon_{23}\\ \epsilon_{13}\\ \epsilon_{12}\end{pmatrix} \tag{4}$$

The stress components above are arranged in vector by the *Voight notation.* Where by *strain* is derived from *displacement* with small strain components

$$\epsilon_{ij} = \frac{1}{2}(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}) \qquad i,j = 1,\cdots,3$$

- Plane bending

$$M = EJ\kappa$$

with the moment M, curvature κ, Young's modulus E, and cross-sectional moment of inertia J. This is covered by the concept of generalized stresses with $\sigma = (M)$ and generalized strains $\epsilon = (\kappa)$.

Equations (1),(2),(3) above are special case of

$$\sigma = C\ .\ \epsilon \tag{5}$$

with the constant material stiffness matrix C describing a linear material behavior. Upon approaching material strength, the behavior becomes physically nonlinear. A simple case is given by the uniaxial elastoplastic law

$$\sigma = \begin{cases} \sigma = E(\epsilon - \epsilon_p) & for - \epsilon_p \leq \epsilon \geq \epsilon_p \\ sign(\epsilon).f_y & otherwise \end{cases} \tag{6}$$

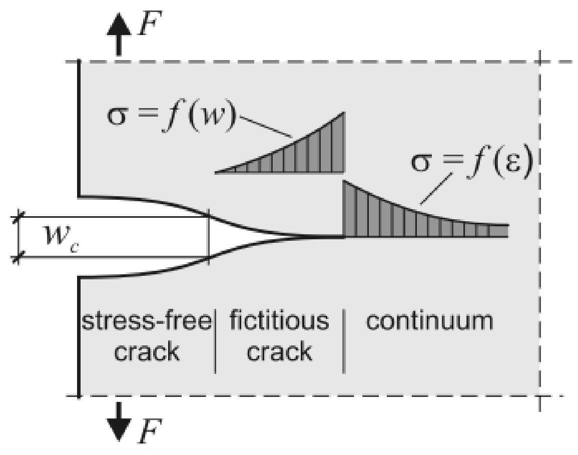
and

$$\dot{\epsilon_p} = \dot{\epsilon} \quad for \quad |\sigma| = f_y \tag{7}$$

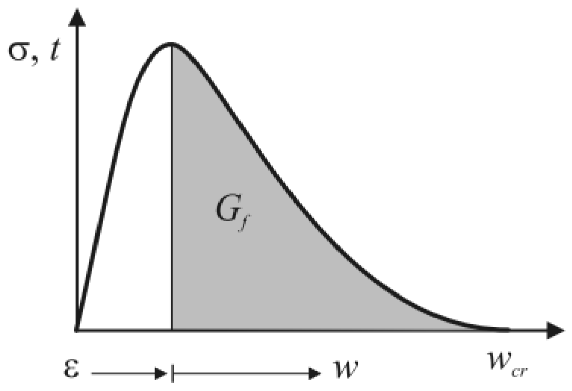
with a yield stress f_y (unsigned) and an internal state variable ϵ_p. An internal state variable captures the preceding load history.
In case of nonlinear material equations at least an incremental form

$$\dot{\sigma} = C_T.\dot{\sigma}$$

should exist with the tangential material stiffness C_T, which is no longer constant anymore but might depend on stress, strain, and internal state variables.



(a) Cohesive crack model



(b) Law of cohesive crack model

Figure 14: Cohesive crack model

4.2 Material ElasticLT

The ElasticLT material is an extension of the Elastic material with limited tensile strength. This material is especially suitable for concrete whose tensile strength is relatively small compared to its compressive strength. Crack formation is a characteristic property of plain and reinforced concrete structures. Cracks are unavoiable due to the relatively low tensile strength of concrete. Crack formation changes the relations of stiffness.

A basic concept of crack modeling is the "*cohesive crack model*". The cohesive crack model assigns surface tractions along fictitious crack boundaries. A cohesive crack law relates surface tractions and fictitious crack widths. Figure 14 illustrates the cohesive crack model.[5]

The ElasticLT material use the "*smeared crack model*" for crack modeling. The smeared stress-strain relation blends or *smears strains of uncracked parts and cracking strains into a unified continuous strain*. Cracking starts in the uniaxial case when the uniaxial stress reaches the uniaxial tensile stress f_{ct}. This is extended to the multiaxial case with the *Rankine criterion*. That means cracking will start when the largest principal stress reaches the uniaxial tensile strength f_{ct}. The crack direction is assumed as normal to the direction of principal stress inducing the crack. Thus, **principal direction and the normal n of the local tangential cracking plane coincide**. This fact is used to determine the local cracking plane in case of multiaxial smeared crack model.

It should be noted that the crack geometry in smeared crack model cannot be precisely determined due to smearing unlike discrete crack, only some area of cracking. Because the smeared crack model leads to a modification of the material law of the uncracked material. Thus, it is applied to integration points within elements, which have a fixed position. But an exact crack position is not determined. The figure 15 illustrates the subroutine to compute stress and tangential stiffness matrix at an integration point from a given strain.

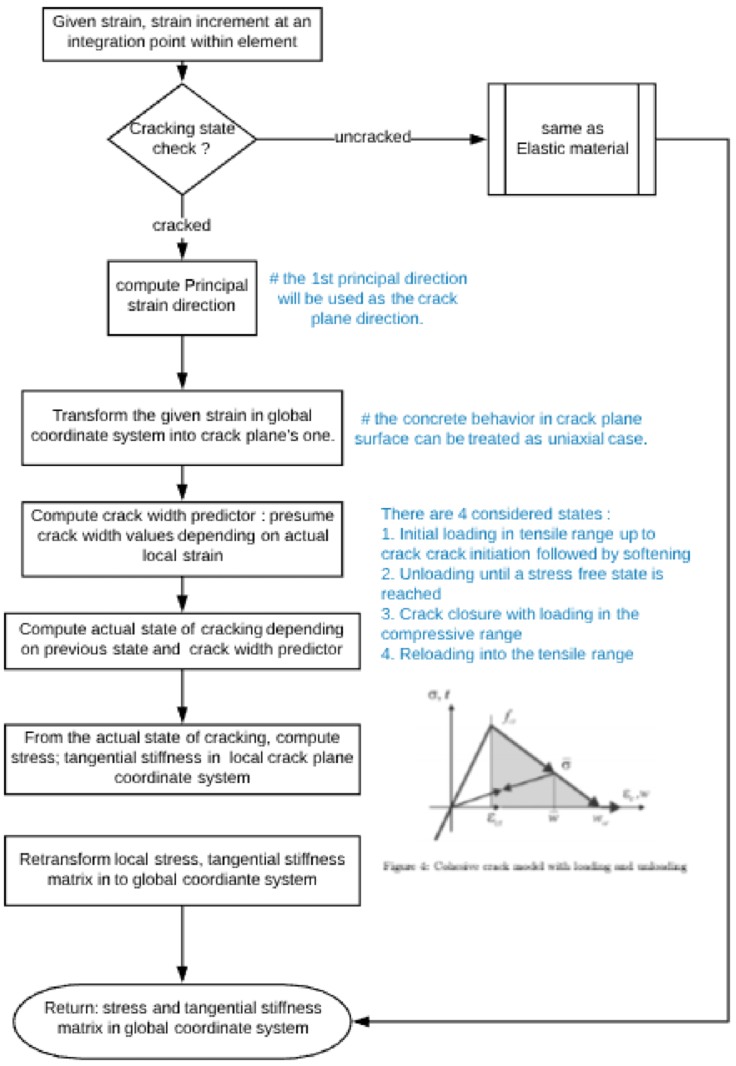

Given strain, strain increment at an integration point within element
Cracking state check ?
uncracked
same as Elastic material
cracked
compute Principal strain direction
# the 1st principal direction will be used as the crack plane direction.
Transform the given strain in global coordinate system into crack plane's one.
# the concrete behavior in crack plane surface can be treated as uniaxial case.
Compute crack width predictor : presume crack width values depending on actual local strain
There are 4 considered states :
1. Initial loading in tensile range up to crack crack initiation followed by softening
2. Unloading until a stress free state is reached
3. Crack closure with loading in the compressive range
4. Reloading into the tensile range
Compute actual state of cracking depending on previous state and crack width predictor
From the actual state of cracking, compute stress; tangential stiffness in local crack plane coordinate system
Retransform local stress, tangential stiffness matrix in to global coordiante system
Return: stress and tangential stiffness matrix in global coordinate system
Figure 4: Cohesive crack model with loading and unloading

Figure 15: Subroutine of ElasticLT material.

4.3 Material IsoDamage

Damage assume a degrading material stiffness without permanent strains upon loading. The base approach for *isotropic damage* is :

$$\sigma = (1 - \mathbf{D})E.\epsilon$$

with the isotropic linear elastic material stiffness E. This is applicable for the triaxial behavior and includes biaxial and uniaxial behavior. D is a state variable, this scalar damage variable by definition has a range $0 \leq D \geq 1$ where by $D = 0$ denotes a fully undamaged material and $D = 1$ a fully damaged material leading to $\sigma = 0$ for every ϵ. The damage variable D needs a law describing its development from 0 to 1. It is

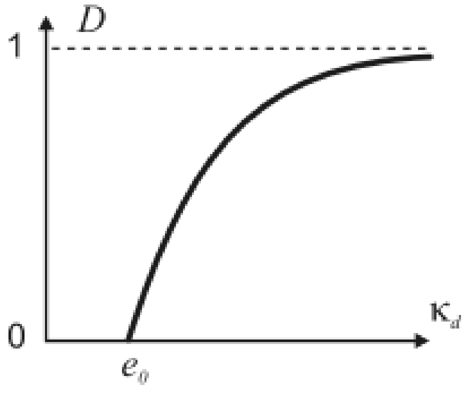


Figure 16: Damage variable D depending on equivalent strain κ_d

coupled to an internal state variable κ_d which comprises the load history. Strain -based damage approach starts with the following relation between damage variable D and the state variable κ_d:

$$D(\kappa_d) = \begin{cases} 0 & \kappa_d \leq e_0 \\ 1 - e^{-\left(\dfrac{\kappa_d - e_0}{e_d}\right)^{g_d}} & \kappa_d \geq e_0 \end{cases} \tag{8}$$

see fig 16, with constant material parameters e_0, e_d, g_d. The internal state variable κ_d is considered as *equivalent strain* for strain-based damage. The equivalent strain is related to the strain ϵ with a *damage function.* ConFEM2D uses the *Hsieh-Ting-Chen damage function* which has a formal similarity of the Hsieh-Ting-Chen strength surface, more details in [7]:

$$F = c_1 J_{2,\epsilon} + \kappa_d \left(c_2 \sqrt{J_{2,\epsilon}} + c_3 \epsilon_1 + c_4 I_{1,\epsilon} \right) - \kappa_d^2$$

The incremental form of the material law for damage is derived as below :

$$\dot{D} = \frac{dD}{d\kappa_d} \dot{\kappa_d} = -\frac{\dfrac{dD}{d\kappa_d}}{\dfrac{\partial F}{\partial \kappa_d}} \frac{\partial F}{\partial \epsilon} . \dot{\epsilon} \tag{9}$$

$$\dot{\sigma} = C_T . \dot{\epsilon} \tag{10}$$

$$C_T = \begin{cases} (1-D)E + \dfrac{\dfrac{dD}{d\kappa_d}}{\dfrac{\partial F}{\partial \kappa_d}} . \sigma_0 . \dfrac{\partial F}{\partial \epsilon} & \textit{loading} \\ (1-D)E & \textit{unloading} \end{cases} \tag{11}$$

The form $\sigma_0 . \dfrac{\partial F}{\partial \epsilon}$ is an outer (dyadic) product $\otimes$ of two vectors. The quantities $\dfrac{\partial F}{\partial \epsilon}, \dfrac{\partial F}{\partial \kappa_d}, \dfrac{dD}{d\kappa_d}$ has to be computed from the forms for F,D.

4.4 Wrapping for Reinforced Concrete Shell SH4 element

The layer model is suitable for continuum-based thin shells as SH4 shell element. It contains either concrete and reinforcement layers. A layer is a plane through the point *r,s,t* with the shell director $V_n(r,s)$ as normal. Every thickness coordinate t has its own layer whereby it shares the normal with all layers of the same shell position *r,s*. Thus, shell reinforcement layers are implemented to subject to integration along the collateral t-direction. The contribution of reinforcement shell layers to internal forces are the same as concrete layers (note that thin reinforcement layers do not contribute to transverse shear forces). It should be also noted that rigid bond between concrete and reinforcement is assumed in this implementation.
To take advance of the pre-built continuum-based thin shell SH4 element, a new wrapper class is defined to wrap concrete and reinforcement layers into a complete element. It simply adds reinforcement layer's integration points to concrete layers. Each layer has its own material behavior, i.e the concrete layer can use Elastic, ElasticLT or IsoDamage material, while the reinforcement layer can use Elastic or MiseUniaxial material.

4.5 Elasto-plastic NLSlab

The NLSlab material is exclusively used for the SB3 slab element. The SB3 slab element is characterized by Kirchhoff slab that ignore the coupling of bending moments and normal forces. The element yields the following output:

- Displacement $u = (w, \phi_y, \phi_x)$
- Generalized strains as curvatures $\epsilon = (\kappa_x, \kappa_y, \kappa_{xy})$
- Generalized stresses as bending moments $\sigma = (m_x, m_y, m_{xy})$

The linear elastic relation between generalized strains and generalized stresses for the Kirchhoff slab:

$$\sigma = C.\epsilon \qquad C = K.\begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \dfrac{1-\nu}{2} \end{bmatrix} \qquad K = \frac{Eh^3}{12(1-v^2)} \tag{12}$$

with Young's modulus E, Poisson's ratio ν and the slab height h.
The NLSlab material use simplified approach to describe nonlinear moment-curvature behavior. An incremental simplified approach of the tangential material stiffness is chosen as :

$$\dot{\sigma} = C_T.\dot{\epsilon} \qquad C_T = \begin{bmatrix} K_{Tx} & 0 & 0 \\ 0 & K_{Ty} & 0 \\ 0 & 0 & K_{Txy} \end{bmatrix} \tag{13}$$

with the bending stiffness K_{Tx}, K_{Ty} derived from uniaxial beam behavior. This corresponds to an ***orthotropic behavior***. The twisting stiffness $C_{T33} = K_{Txy}$ is derived based on the theory of orthotropic slabs leading to

$$K_{Txy} = \frac{1}{2}\sqrt{K_{Tx}K_{Ty}}$$

The slab's cross sections with normals in the x- and y-directions are treated separately to derive K_{Tx}, K_{Ty} and each is derived as for a beam cross section of unit width. An elastoplastic moment-curvature relation is then defined as the figure 17 with $N = 0$. Material behavior in each direction is described by :

- initial bending stiffness K_s
- initial yielding moment m_{yk}
- hardening bending stiffness K_T
- current plastic curvature κ_p as state parameter

in analogy to uniaxial elasto-plastic behavior.

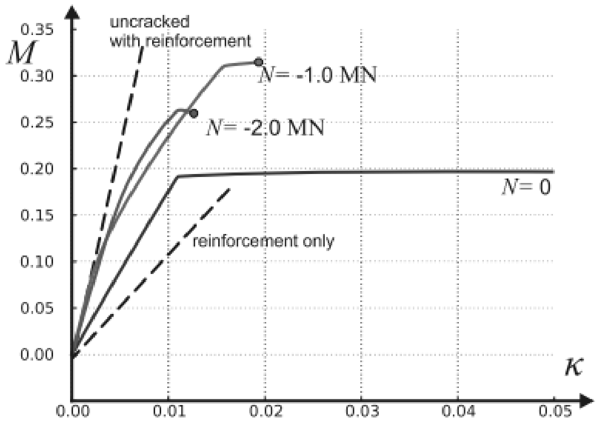


Figure 17: Elastoplastic moment-curvature relation

5 Points to improve

Nothing is perfect. There exists many aspects to get ConFEM2D software better. The following points resume what I think should do to improve :

1. Accelerating calculation time of the solver by using better linear algebra libraries. The sparse solver can better perform by using C++ code-based libraries such as uBLAS, Trilinos library instead of the python standard SciPy library which uses SuperLU/Umfpack library.
2. Building GUI (by Tkinter) for more intuitive interfacing with software.
3. Finding a better mesh generator instead of Gmsh4.0 ?
4. Developpement of SB4 slab quad element.

References

[1] Gmsh4.0 free software. `http://gmsh.info/`

[2] Klaus-Jurgen Bathe. *Finite Element Procedures*. 2nd edition, 2014.

[3] SciPyPackages/Sparse. `http://scipy.github.io/old-wiki/pages/SciPyPackages/Sparse`

[4] Paul D. Nation and Robert J. Johansson. *Reverse Cuthill-McKee function, a part of QuTiP: Quantum Toolbox in Python.* http://science-and-samgaetang.blogspot.com/2014/01/reverse-cuthill-mckee-ordering-in.html

[5] Ulrich Haussler-Combe. *Computational methods for reinforced concrete structures.* Ernst & Sohn, Nov 2014.

[6] H. Matthies and G. Strang. *The solution on nonlinear finite element equations.* International Journal for Numerical Methods in Engineering , 14:1613–1626, 1979.

[7] S. Hsieh, E. Ting, and W. Chen. *A plasticity fracture-model for concrete.* Int. J.Solids Structures,18:181-197, 1982.

[8] GitHub repository of ConFEM2D. `https://github.com/DuongDoNgoc/ConFEM2D`