International Journal of Computer Applications (0975 – 8887)
Volume 40– No.14, February 2012

Numerical Modelling of Concrete Tensile Strength Test by Wrapping Scripting Language with Compiled Library

Md. Golam Rashed
Department of Civil Engineering, Ahsanullah University of Science and Technology (AUST), Dhaka-1208, Bangladesh

Raquib Ahsan, Phd
Department of Civil Engineering, Bangladesh University of Engineering and Technology (BUET), Dhaka-1000, Bangladesh

Sharmin Reza Chowdhury, Phd
Department of Civil Engineering, Ahsanullah University of Science and Technology (AUST), Dhaka-1208, Bangladesh

ABSTRACT
The importance of engineering simulation is increasing day by day with the increase of computing power. The finite element analysis method is one of the widely used approaches for this purpose. To achieve optimum simulation, there is no alternative to take complete control over the code which proprietary commercial codes fail to offer. This paper focuses on the review of the development of a finite element analysis framework using freely available python libraries and wrapping legacy C/C++ or Fortran libraries around python; and its verification as a viable finite element solution with an example of concrete tensile strength test simulation.

Keywords
Finite Element Analysis, Numerical Modeling, Engineering Simulation, Scientific Computing, Sparse Matrix, Python.

1. INTRODUCTION
The finite element analysis is a computer based numerical technique which gives near accurate result in modeling complex engineering phenomena with very small error margin [1]. Major applications for FEA include static, dynamic and thermal characterizations of mechanical phenomena occuring in nature and in real life [2] [3]. FEA is also being used as a tool to model biological phenomena occuring in human body [4] and to optimize smart characteristics of advanced materials & devices [5]. Advances in computer hardware have made FEA easier and very efficient for solving complex engineering problems on desktop computers. To achieve optimum simulation, there is no alternative to take complete control over the code which proprietary commercial codes fail to offer. Moreover, To avoid input errors and increase the simulation reliability, the user must have the access to the source code level. This paper discusess the present numerical modeling practice and suggests an optimum numerical modeling framework from both the performance and accessibility point of view.

2. PRACTICES IN ENGINEERING SIMULATION
There are three possible approaches by which finite element analysis is performed for an engineering phenomenon.
Firstly, one can use a propritery finite element analysis software package, such as ABAQUS, ANSYS or even general purpose prototyping framework MATLAB. The main advantage of these FEA software packages is that they come equiped with both pre- and post-prossesor. The operation of a specific software is usually detailed in the documentation accompanying the software, and vendors of the more expensive codes will often offer workshops or training sessions as well to help users learn the intricacies of code operation. One problem users may have even after this training is that the code tends to be a "black box" whose inner workings are not understood by the end user. Thus, although ABAQUS, ANSYS are good for finite element analysis packages, they are not suitable for full-blown research projects, where complete control is the top most priority. Furthermore, they have fixed keywords, are not open source, very expensive and not widely accepted in academia.
Secondly, one could develop their own FEA program, in a high-level scripting language. However, one should realize that, there is a concern for speed if the whole program is coded in a scripting language alone. FEA involves solution of large matrices which requires high computing power as well as efficient algorithm. In many cases, pre-written library routines offer solutions to numerical problems which are quite hard to improve upon [6].
Thirdly, coding can be done in a high-level scripting language, but invoking pre-written, pre-compiled routines in commonly available subroutine libraries, such as UMFPACK, LAPACK and BLAS, to perform all of the real numerical work. This approach is most beneficial for ease in coding and performance points of view. This approach is used by the majority of researchers and scientists in their modern FEA simulation programs. In developing efficient FEA framework, using FORTRAN or C/C++ is less preferable as the code becomes less expressive, making it difficult to maintain over the years of reaserch where different people work on a single code at different times.
So, it is evident that the use of open source FEA software is beneficial for both the user and researcher, and it will further enhance the simulation reliability if it is a self coded one.

3. DEVELOPMENT OF OPEN SOURCE FINITE ELEMENT ANALYSIS FRAMEWORK
Optimum simulation depends on complete control over the finite element analysis framework but commercial products fail to offer such control as they are proprietary, closed source. So a user is unable to see what is going on behind the scene during the simulation. Developing a finite element analysis framework, or at least using an open source finite element analysis software will help the end user by achieving complete control over the simulation process. Open source software is community driven, so large number of users can contribute to the continuing development of the software. The

uses of freely available modules ensure least cost in both usability and maintenance.

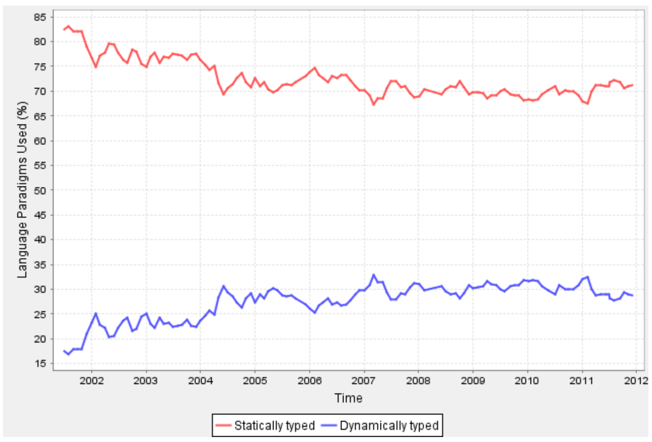

Figure 1: Uptrend of Dynamically typed language [7]

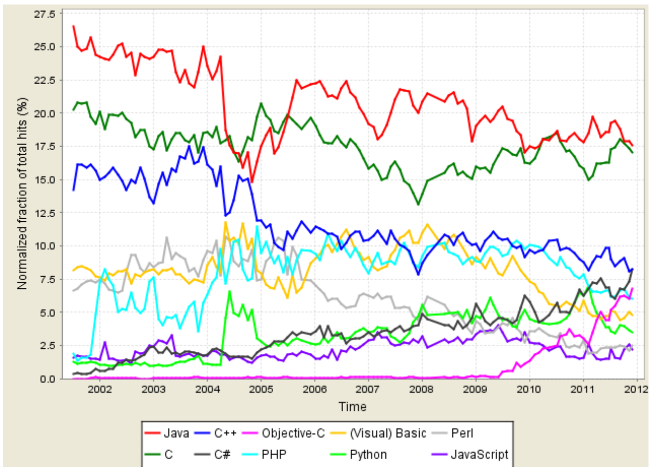

Figure 2: Increasing popularity of Python among the top 10 most popular languages [7]

Over the years dynamic languages have steadily gained acceptance over static languages (Figure 1). The popularity of python as a general purpose programming language as well as a scientific computing tool is on the rise (Figure 2). According to Millman python is the best choice for the scripting environment in developing high performance scientific code [8]. Python is used as the glue language because it is highly readable and easy to understand where end user can use this as the interface to the complex compiled code. Python offers the benefits of object-oriented and generic programming, together with a syntax that is simpler and clearer than static languages, providing high code re-use [9]. It has all major scientific libraries available either as standard library or as third party open source library. Libraries like NumPy, SciPy, Matplotlib and Mayavi allow Python to be used effectively in scientific computing [10] [11]. The advantages of python are tabulated against traditional FORTRAN and C/C++ in Table 1.

To overcome all the difficulties associated with commercial FEA packages and to take advantage of python, an open source FEA framework named "simple finite element in python" or "Sfepy" has been developed. Sfepy is finite element analysis software written almost entirely in Python, with exception of the most time demanding routines, those are written in C and wrapped by Cython or written directly in Cython. Sfepy is a software for solving systems of coupled partial differential equations by the finite element method in 2D and 3D. Solutions of linear, nonlinear problems are possible in Sfepy [12].

The finite element method is comprised of three major phases: (1) Pre-processing, in which the analyst develops a finite element mesh to divide the subject geometry into sub domains for mathematical analysis (Figure 3), and applies material properties and boundary conditions. Gmsh provides a reliable pre-processing solution and is the pre-processor of the developed FEA framework. The simplest way of using Sfepy is to mesh the geometry using Gmsh and then solve a system of PDE's describing the physical problem in python programming language defined in a problem description file.

Table 1: Comparison of C/C++, FORTRAN and Python [13] [14] [15] [8]

Language	**C/C++**	**Fortran**	**Python**
Intended use	Application, System	Application, Numerical Computing.	Application, General, Web, Scripting.
Paradigm	Imperative, procedural, object-oriented (C++).	Generic, imperative, object-oriented, procedural.	Aspect-oriented, functional, imperative, object-oriented, reflective.
Type strength	strong	strong	strong
Type safety	unsafe	safe	safe
Expression of types	explicit	explicit	implicit
Type checking	static	static	dynamic
Failsafe I/O	No	No	Yes
Statements ratio	1 / 2.5	2	6
Lines ratio*	1	0.8	6.5

* The ratio of line count tests won by each language to the number won by C when using the Compare to feature at Shootout.alioth.debian.org. C gcc was used for C, C++ g++ was used for C++, and FORTRAN G95 was used for FORTRAN.

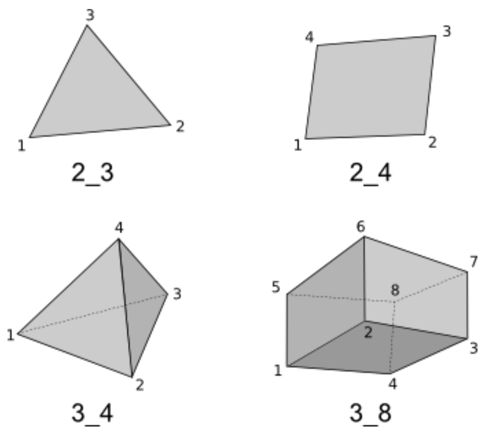

Figure 3: Supported geometry elements [12]

(2) Solution, during which the program assembles the governing equations from the model into matrix form and solves numerically for the primary quantities. The assembly process depends on the type of analysis such as static or dynamic, on the model's element types and properties, material properties and boundary conditions. The dataset prepared by the pre-processor is used as input to the finite element code itself, which constructs and solves a system of linear or nonlinear algebraic equations in the form $Kd = r$, where K is the system stiffness matrix, d is the nodal degree of freedom displacement vector, and r is the applied nodal load vector. The strain-displacement relation may be introduced into the stress-strain relation to express stress in terms of displacement. Under the assumption of compatibility, the differential equations of equilibrium along with the boundary conditions then determine a unique displacement field solution, which in turn determines the strain and stress fields. The chances of directly solving these equations are less, hence the need of approximate numerical techniques rises. Various Solution methods exists for finite element matrix equations. In the case of the linear static $Kd = r$, during solution inversion of K is numerically unstable and computationally expensive. A better alternative is Cholesky factorization, a form of Gauss elimination, and a minor variation on the "*LDU*" factorization scheme. The K matrix may be efficiently factored into LDU, where L is lower triangular, D is diagonal, and U is upper triangular, resulting in $LDUd = r$. Since L and D are easily inverted, and U is upper triangular, d may be determined by back-substitution. In wavefront method, the equations are assembled and reduced at the same time [16]. Sparse matrix technique is the best modern solution method. As the stiffness matrix has a large number of zero entries, solution time and storage can be reduced by a factor of 10 or more by using optimized algorithm [6]. In developing Sfepy, solver SuperLU integrated in scipy library is employed.

Table 2: Comparison of popular sparse direct solver [6]

Compiled library	**Language**	**Free to academics**	**Factorization algorithm**
BCSLIB-EXT	F77	No	Multifrontal
MA57	F77/F90	No	Multifrontal
MUMPS	F90	Yes	Multifrontal
Oblio	C++	Yes	Left-looking, right-looking, multifrontal
PARDISO	F77 & C	Yes	Left-right looking
SPOOLES	C	Yes	Left-looking
SPRSBLKLLT	F77	Yes	Left-looking
SuperLU	C	Yes	Left-looking, Unsymmetric
TAUCS	C	Yes	Left-looking, Multifrontal
UMFPACK	C	Yes	Unsymmetric multifrontal
WSMP	F90 & C	Yes	Multifrontal

And (3) Post-processing, in which the analyst checks the validity of the solution, examines the values of primary quantities such as displacements and stresses, and derives & examines additional quantities such as specialized stresses and error indicators. In developing Sfepy, matplotlib for 2D and Mayavi for 3D visualization of solution results is employed.

4. VERIFICATION

The tensile strength test of concrete which is also known as split cylinder test, shown in Figure 4(a), is simulated with the developed finite element analysis framework.

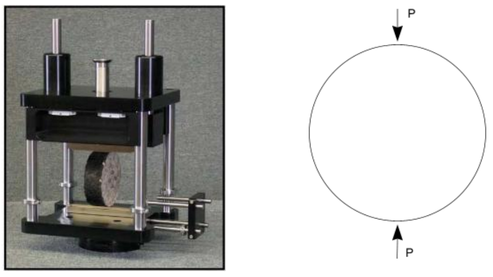

Figure 4: (a) Indirect Tesnsile Strength test setup, (b) Simplified 2D diametrically point loaded disk.

In this test a cylindrical specimen is loaded across its diameter to failure. To model this problem using finite elements, the indirect tensile test can be simplified to represent a diametrically point loaded disk as shown in Figure 4(b). At the centre of the cylindrical specimen of diameter (D) and thickness (t), Tensile stress (σ_t) is developed horizontally and Compressive stress (σ_t) is developed vertically for a point load P. The compressive stress is 3 times the tensile stress and the analytical formulation for these are, respectively:

$$\sigma_t = \frac{2P}{\pi t D} \quad (1)$$

$$\sigma_c = \frac{6P}{\pi t D} \quad (2)$$

The specimen has a diameter of 150 mm. Assuming plane strain conditions, the test is modeled using 2D triangle element and only one quarter of the 2D model is used as the model geometry is symmetrical about x- and y-axes (Figure 5). The material is assumed to be linear elastic and isotropic with a Young’s modulus of 2,000 MPa and a Poisson’s ratio of 0.4, which are inputed in the linear elastic isotropic equation as Lamé’s parameter. The vertical displacement of the nodes in the bottom and the horizontal displacement of the nodes in the left are set to zero to ensure symmetry.

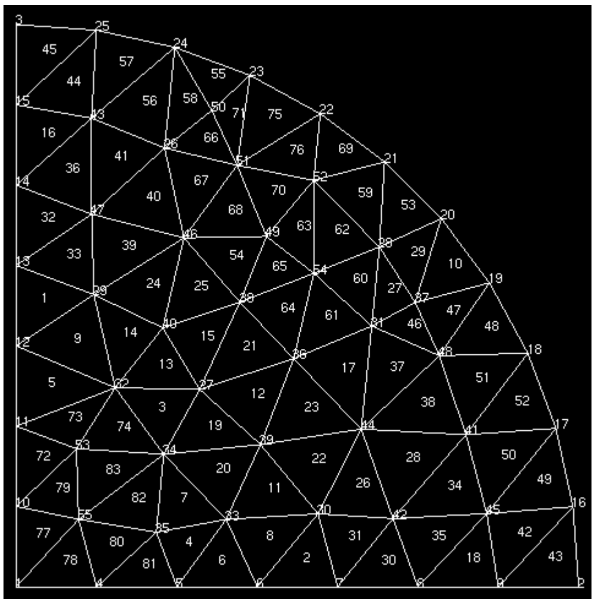
Figure 5: Meshed geometry

The resultant matrix from the governing equation is unsymmetric sparse in nature so optimized Umfpack library is used instead of SuperLU in the modeling of the test which provided better performance (Table 2).
The stresses in the model had been calculated but these were averaged from those calculated at Gauss quadrature points within the elements. Stresses at bottom-left node is calculated as it is the centre of the concrete disk. The analytic solution is independent of mesh refinement influence, so it will always be same for as long as the whole geometry remains intact. On the other hand, the finite elemnt analysis solution is influenced by mesh refinement. The finer the mesh is, the more accurate the result will be. The FEA result for both a coarse and fine mesh is shown in Table 3.
It is established that the developed FEA framework and the modeling technique outlined in this paper works correctly as the FEA approximate solution of the modeled concrete tensile strength test is very close to the analytical solution and moves towards convergence with mesh refinement. The implementation of optimized Umfpack library to solve large unsymmetric sparse matrix has proven to be efficient.

Table 3: Analysis summary with coarse and fine mesh

Mesh type	Applied load (N)	Analytic horizontal tensile stress (MPa/mm)	FEA horizontal tensile stress (MPa/mm)	% error w.r.t Analytic solution	Analytic vertical compressive stress (MPa/mm)	FEA vertical compressive stress (MPa/mm)	% error w.r.t Analytic solution
Coarse Mesh - 83 Elements	2000	8.48826	7.57220	+10.79	25.4648	25.8660	-1.57
Fine Mesh - 5568 Elements	2000		8.50042	-0.14		25.4300	+0.13

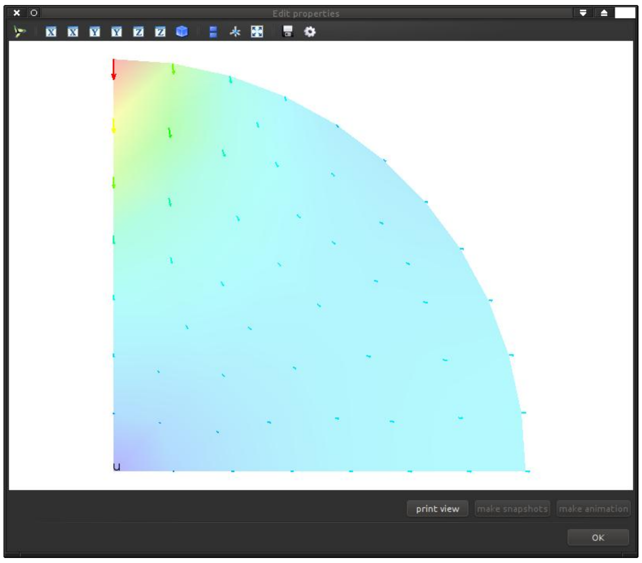

Figure 6: Visualization of the solution

5. CONCLUSION

Taking the advantages of full control over the Engineering simulation using custom coded finite element analysis software along with its structure and working principle is focused in this paper. As the source code is freely available to use and modify, many user can contribute their individual knowledge regarding various aspects of numerical modeling.

This paper also focuses on the necessity and ease of developing engineering simulation framework. The evaluation of the developed framework is presented with comparison to an analytical solution. It has been observed that development and use of open source engineering simulation software offers four main benefits:
1. Anyone can contribute to the continuing development of the software, as it is open source.
2. Optimum simulation is obtained as the user can see what is running behind the scene.
3. The use of legacy mathematical libraries written in static language offers great code reuse and saves time; also it is easier to change the underlying legacy library.
4. Use of freely available libraries provides maximum economy.

6. REFERENCES

[1] Susmita Sinha, Sunipa Roy, C. K. Sarkar. "Design & Electro-Thermal Analysis of Microheater for Low Temperature MEMS based Gas Sensor." International Symposium on Devices MEMS, Intelligent Systems & Communication. Sikkim, India: International Journal of Computer Applications, 2011. 26-31.

[2] J. N. Sharma, Dinkar Sharma, Sheo Kumar. "Analysis of Stresses and Strains in a Rotating Homogeneous Thermoelastic Circular Disk by using Finite Element Method." International Journal of Computer Applications, 2011: 10-14.

[3] Saleh Alsubari, Hassan Chaffoui. "Homogenization of a Composite Periodic Structure in the Case of Composite Plate ." International Journal of Computer Applications, 2011: 28-33.

[4] Mamta Agrawal, Neeru Adlakha, K.R. Pardasani. "Modelling and Simulation of Thermal Effect of Metastasis of Tumors in Human Limbs." International Symposium on Devices MEMS, Intelligent Systems & Communication. Sikkim, India : International Journal of Computer Applications, 2011. 24-30.

[5] S.Maflin Shaby, A.Vimala Juliet. "Improving the Sensitivity of MEMS Piezoresistive Pressure Sensor using Polysilicon Double Nanowire." International Journal of Computer Applications , 2011: 1-4.

[6] Nicholas I. M. Gould, Yifan Hu, Jennifer A. Scott. A numerical evaluation of sparse direct solvers for the solution of large sparse, symmetric linear systems of equations. Oxfordshire, UK: Council for the Central Laboratory of the Research Councils, 2005.

[7] TIOBE Programming Community Index. 12 30, 2011. http://www.tiobe.com/index.php/tiobe_index (accessed 12 30, 2011).

[8] Millman, K. "Python for Scientists and Engineers." Computing in Science & Engineering 13, no. 2 (2011): 9 - 12.

[9] Rossum, Guido van. Glue It All Together With Python. Monterey, California: Object Services and Consulting, Inc., 1998.

[10] Pérez, F. "Python: An Ecosystem for Scientific Computing." Computing in Science & Engineering 13, no. 2 (2011): 13 - 21.

[11] van der Walt, S. "The NumPy Array: A Structure for Efficient Numerical Computation." Computing in Science & Engineering 13, no. 2 (2011): 22 - 30.

[12] Cimrman, Robert. SfePy: Simple Finite Elements In Python. 12 30, 2011. http://sfepy.org/ (accessed 12 30, 2011).

[13] Ritchie, Dennis M. "The Development of the C Language." The second ACM SIGPLAN History of Programming Languages Conference. New York: ACM, 1993. 201–208.

[14] Stroustrup, Bjarne. The C++ Programming Language. Addison-Wesley, 2000.

[15] Akin, Ed. Object Oriented Programming via Fortran 90/95. Cambridge University Press, 2003.

[16] Roensch, Steven J. The Finite Element Method: A Four-Article Series. ASME newsletter, 1996.