Object oriented finite element probabilistic stress analysis

Fernando César Meira Menandro
fmenandro@uol.com.br
Departamento de Engenharia Mecânica
Universidade Federal do Espírito Santo
Av. Fernando Ferrari, 514, 29060-050, Vitória - ES

November 13, 2006

Abstract

An object oriented finite element code was developed to take advantage of the clearly object oriented nature of finite elements, specially with respect to property inheritance and function overloading. Furthermore, the operator overloading characteristic of the C++ language allowed for the inclusion in the software, without any additional programming effort, of the probability density functions of any variables one might wish to analyze. By no additional programming effort it is meant that the actual finite element code is exactly the same as a deterministic code (obviously, programming of the overloaded operators and data type was necessary). The resulting software expands the realm of application of the finite element method for true probabilistic stress analysis, either for actual experimental verification of for probabilistic design with respect to uncertainties associated to physical properties, boundary conditions, and geometry.

1 Introduction.

The use of the finite element method for structural analysis is already well consolidated in the engineering community. The advances in materials and computational resources, though, have substantially altered basic premises which guided the development of this method. It is no more necessary to seek computationally fast and economic solutions, since processing speed and memory availability are no longer limiting factors. This new paradigm calls for ever more realistic approximate models, which means, for less approximate and more perfect models of reality. These new models require more care in the analysis, in details that were until recently completely neglected, either due to the imprecision of the models themselves or the cumbersome nature of the implementation of such details.

Stemming from this premise, this work shows the development of a software that takes into account the probability density functions of the involved variables as an integral part of the analysis. Although there exists a Stochastic Finite Element Method, the implementation shown here is far simpler, as it takes advantage of the operator overloading capabilities of the C++ programming language.

2 Theoretical Background.

In this section a brief introduction on probability theory is presented, to illustrate the developed procedures. Short accounts of object oriented programming as well as of the finite element method are also presented.

2.1 Probability.

The initial part of our theoretical background recalls some concepts and definitions on probability theory, and can be found in any introductory text on the subject [1].

Definition 2.1 *Let ε be an experiment and S a sampling space associated to the experiment. A function X, that associates to each element $s \in S$ a real number, $X(s)$, is called a random variable.*

Definition 2.2 *Let X be a continuous random variable. The* probability density function *f, written in short form as pdf, is a function f that satisfy the following conditions:*

(a) $f(x) \geq 0$ *for all* $x \in R_X$,

(b) $$\int_{R_X} f(x)dx = 1. \tag{1}$$

Furthermore, one can define for any $c < d$ *(in* R_X*),*

$$P(c < X < d) = \int_c^d f(x)dx = 1. \tag{2}$$

The definition of the accumulated distribution function will be useful and is presented next.

Definition 2.3 *Let* X *be a random variable, discrete or continuous. Function* F *is defined as the* accumulated distribution function *of the random variable* X*, (short form: df) as* $F(x) = P(X \leq x)$*.*

To this definition follow some theorems which characterize the accumulated distribution function as a non-decreasing function and define its relation with the probability density function:

$$F(x) = \int_{-\infty}^{x} f(s)ds; \tag{3}$$

or

$$f(x) = \frac{d}{dx}F(x). \tag{4}$$

Besides these definitions, it can be shown that for two independent continuous random variables X and Y, with pdfs g and h, respectively, the pdf s of the variable $Z = X + Y$ is given by [1, p. 275]:

$$s(z) = \int_{-\infty}^{+\infty} g(w)h(z - w)dw. \tag{5}$$

It can also be shown that for two independent continuous random variables X e Y, with pdfs g and h, respectively, the pdf p of the variable $V = XY$ is given by [1, p. 115]:

$$p(v) = \int_{-\infty}^{+\infty} g(u)h\left(\frac{v}{u}\right)\left|\frac{1}{u}\right| du, \tag{6}$$

and that for two independent continuous random variables X e Y, with pdfs g and h, respectively, the pdf q of the variable $U = X/Y$ is given by [1, p. 117]:

$$q(u) = \int_{-\infty}^{+\infty} g(vu)h(v)\,|v|\,dv. \tag{7}$$

It can be also shown that for a random variable X with pdf f, the pdf g of a random variable $Y = H(X)$, monotonic function (increasing or decreasing) of X is given by [1, p. 92]:

$$g(y) = f(x)\left|\frac{dx}{dy}\right|. \tag{8}$$

If the accumulated distribution function $y = H(x)$ is increasing,

$$G(y) = F(x), \tag{9}$$

and for $y = H(x)$ decreasing,

$$G(y) = 1 - F(x). \tag{10}$$

2.2 Programming Language.

The choice of the programming language depends of the verification of some fundamental requisites, such as: execution efficiency, low level resources, object oriented capabilities, and ease of graphics programming [2].

Among the available programming languages, the one that best suits the proposed requisites is C++ [3][4]. Since the proposed system should be able to function either under Windows^{tm} or Linux^{tm}, with a graphics interface, a visual development environment would be a better option, and the KDevelop environment was chosen.

There are four basic properties which characterize object oriented programming [5]: Abstraction, encapsulation, inheritance and polymorphism. Abstraction is the capacity of representing abstract data types, or abstract concepts, with a user created class of objects. Encapsulation is the capacity of encompassing intrinsic properties within the class programming, without leaving to the final programmer (or the user) the need for specific knowledge of the abstract data type. Inheritance is the capacity to write general coding

that can apply to all objects instantiated with that class, or with classes derived from the class that contain that code. Polymorphism, on the other hand, allow classes derived from the same class to have different behavior and carry different information. The finite element method is quite suitable to this programming paradigm. Since all elements are derived from the same general principles, some routines concerned with post-processing, graphical user interface, or retrieval of element properties, can clearly be written in a general way such as to allow inheritance of these methods. All elements have nodes, connectivity, material properties, real constants associated, but each might have a different number of such properties and thus, different behavior concerning these properties. Some element characteristics, such as their internal displacement interpolation functions, shape functions, and material behavior type, can be encapsulated, leaving for the user only the trouble of using the element in a piece of code. Finally, different elements have different behavior, and this can be accomplished by overloading existing methods on the parent class, such that the new class has its own behavior.

2.2.1 Operator Overloading

The operator overloading technique, present in the C++ programming language, allows the definition of new algebraic operators for new data types. With this technique, and the definitions on the previous section, it was possible to define a set of operators for a new abstract data type created to represent a random variable.

Preliminary studies pointed towards the storage of points related to the accumulated distribution function, since it is a monotonically increasing function with values always between zero and one. The adoption of this function as the storage data for the random variables would limit the amount of memory storage, but it would also increase the amount of computation needed for each operation to be performed. The probability density function (pdf) was thus chosen as the storage function. The operations will be performed on these functions for each variable, and each variable is stored as a set of properties defining the function.

Programming the random variable class required knowledge on data structures and advanced numerical methods, as well as object oriented programming. Finite element programing for this code was, on the other hand, quite simple, since the operator overloading took care of the difficulties generated by the random variables[8].

2.3 Finite Element Method.

The formulation of the finite element method can be simplified by taking the minimizing coefficients of the functional [6][7]

$$\Pi = \frac{1}{2}\int_V \varepsilon^T C \varepsilon dV - \int_V U^T f^B dV - \int_S U^{S^T} f^S dS, \tag{11}$$

where ε is the strain, U is the associated displacement in each point of the element, f^B is the body force and f^S is the surface force.

Different will involve integration in different domains (dimensionally speaking) and might involve different interpolation functions.

By adopting a set of displacement interpolating functions one can generate a system of equations which will represent the physical behavior in the element. For irregular domains it becomes necessary to also map the deformed element in an undeformed configuration, which creates the need for position interpolation functions, also called shape functions. For the present paper, only the interpolation functions for the two-dimensional linear quadrilateral isoparametric element will be shown. They are (for r and s, the parametric coordinates in undeformed space):

$$\left.\begin{array}{l} N_1 = \frac{1}{4}(1+r)(1+s), \\ N_2 = \frac{1}{4}(1-r)(1+s), \\ N_3 = \frac{1}{4}(1-r)(1-s), \\ N_4 = \frac{1}{4}(1+r)(1-s). \end{array}\right\} \tag{12}$$

The previous equations, as stated, interpolate both displacements and shape as follows:

$$\left.\begin{array}{l} x = \sum_{i=1}^{q} N_i x_i, \\ y = \sum_{i=1}^{q} N_i y_i, \end{array}\right\} \tag{13}$$

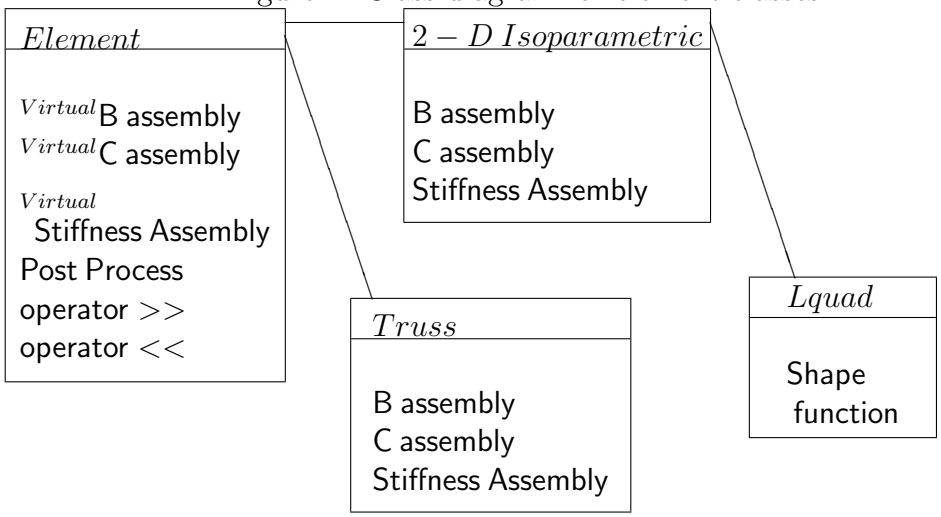
$$\left.\begin{array}{l} u = \sum_{i=1}^{q} N_i u_i, \\ v = \sum_{i=1}^{q} N_i v_i, \end{array}\right\} \tag{14}$$

where x_i, y_i, u_i, and v_i are, respectively, the coordinates and displacements of node i of the element, which has q nodes.

For the truss element, the stiffness matrix can be obtained both for the same approach shown above and from strength of materials considerations.

For the 2-D isoparametric element, the programing language showed its full potential. The actual routines written were the constructor, `B assembly`,

Figure 1: Class diagram for element classes



`C assembly`, and `Stiffness assembly` (using $\mathtt{B}^T\mathtt{CB}$), all for the 2-D Isoparametric element class, overloading the element routines. For the actual Linear quadrilateral element it was only necessary to overload the shape functions (displacement interpolation functions, since we are dealing with isoparametric elements). An example of the element class tree is as follows:

The programmed shape function is as follows:

```
void elemento2D4N::monta_n(int pg)
{
#ifdef ALEATORIO
   aleatorio
#else
   double
#endif
          r,s,J[2][2],invJ[2][2],um=1.0,quatro=4.0;
   double xpg[ptg],wpg[ptg];
   int p=ptg;
   for (int i=0;i<2;i++)
      for (int n=0;n<4;n++)
         dn[2*n+i]=dN[2*n+i]=0.0;
   pontos_de_gauss(p,xpg,wpg);
   r=xpg[pg%p];
   s=xpg[pg/p];
   peso=wpg[pg%p];
   peso*=wpg[pg/p];
   N[0]=(um+r)*(um+s)/quatro;
   N[1]=(um-r)*(um+s)/quatro;
   N[2]=(um-r)*(um-s)/quatro;
   N[3]=(um+r)*(um-s)/quatro;
   dn[0]=(um+s)/quatro;
   dn[1]=(um+r)/quatro;
   dn[2]= -(um+s)/quatro;
   dn[3]=(um-r)/quatro;
   dn[4]= -(um-s)/quatro;
   dn[5]= -(um-r)/quatro;
   dn[6]=(um-s)/quatro;
   dn[7]= -(um+r)/quatro;
```

```
   J[0][0]=J[0][1]=J[1][0]=J[1][1]=0.0;
   for (int i=0;i<2;i++)
      for (int j=0;j<2;j++)
         for (int n=0;n<4;n++)
            J[i][j]+=dn[2*n+i]*this->pno[n]->qx(j);
   detJ=J[0][0]*J[1][1]-J[1][0]*J[0][1];
   invJ[0][0]=J[1][1]/detJ;
   invJ[1][1]=J[0][0]/detJ;
   invJ[0][1]=-J[0][1]/detJ;
   invJ[1][0]=-J[1][0]/detJ;
   for (int i=0;i<2;i++)
      for (int j=0;j<2;j++)
         for (int n=0;n<4;n++)
            dN[2*n+i]+=invJ[i][j]*dn[2*n+j];
   peso*=detJ;
}
```

3 Results

In this section a couple sample problems are shown, to illustrate the potential of the present technique.

3.1 Truss

The truss example shown below was created to test the obtained results. The results were compared to the reference [9].

3.1.1 Deterministic Values

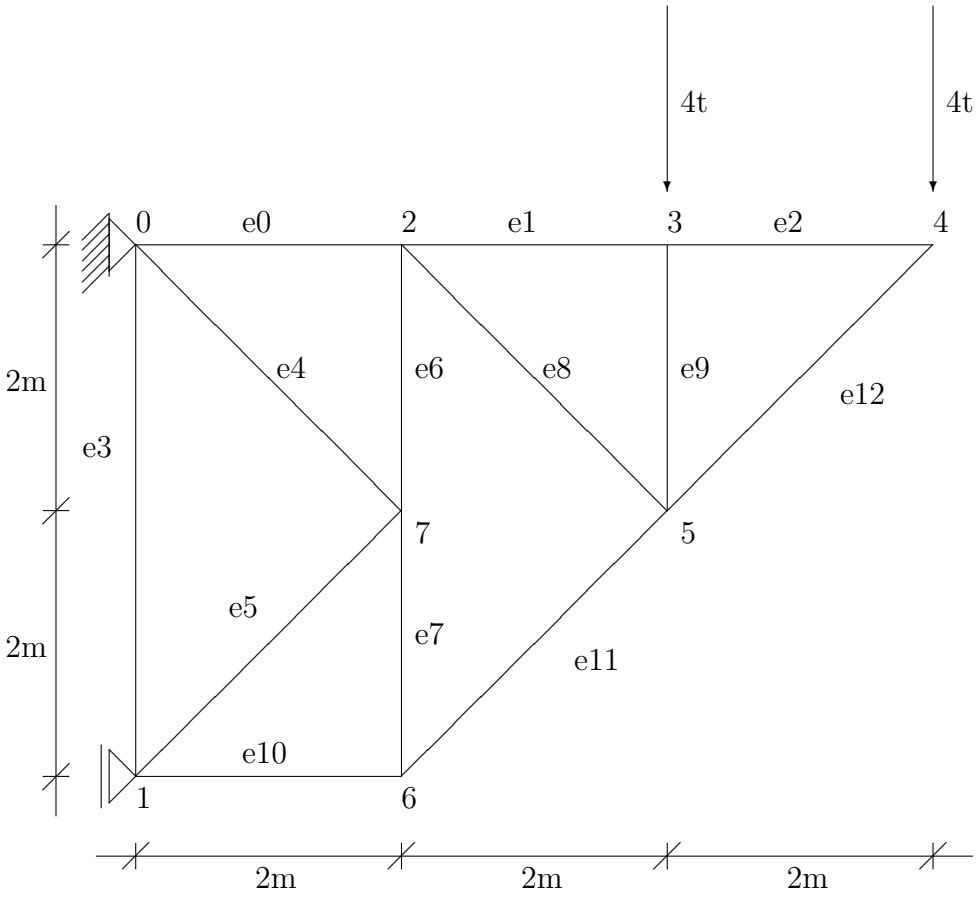
In this case, all data values furnished were deterministic. The results, in this case, as expected, all present zero deviation from the mean.

It can be observed that the tension (or compression) force, described in the output data as calculated stress (tensão calculada) are correct. It is important to point out that the used printing format prints only the mean and standard deviation, but the program keeps in storage the description of the complete probability density function associated to each variable, and it is possible to the user to recover this information.

```
Elemento 0:  Tensao calculada = 6  0
Elemento 1:  Tensao calculada = 4  0
Elemento 2:  Tensao calculada = 4  0
Elemento 3:  Tensao calculada = 4  0
Elemento 4:  Tensao calculada = 5.65685  0
Elemento 5:  Tensao calculada = -5.65685  -0
Elemento 6:  Tensao calculada = -2  -0
Elemento 7:  Tensao calculada = 6  0
Elemento 8:  Tensao calculada = 2.82843  0
Elemento 9:  Tensao calculada = -4  -0
Elemento 10: Tensao calculada = -6  -0
Elemento 11: Tensao calculada = -8.48528  -0
Elemento 12: Tensao calculada = -5.65685  -0

Vetor de deslocamentos resultantes =
```

Figure 2: Truss problem



```
u[0]=0  0
u[1]=0  0
u[2]=0  0
u[3]=-0.0016  0
u[4]=0.0012  0
u[5]=-0.00346274  0
u[6]=0.002  0
u[7]=-0.00812548  0
u[8]=0.0028  0
u[9]=-0.0139196  -0
u[10]=-0.00153137  0
u[11]=-0.00732548  -0
u[12]=-0.0012  -0
u[13]=-0.00426274  -0
u[14]=-0.0008  0
u[15]=-0.00306274  0
```

3.1.2 Variation of the loading.

In this case, the values of all furnished data were deterministic, except those for the applied forces. These were described by a constant probability distribution function (pdf), with values between -4,04 and -3,96 (plus or minus 1%). This function has the calculated standard deviation of 0,023094. The results present mean and standard deviation for all calculated values, showing how this small uncertainty about entry data can alter our analysis results. It is interesting to notice that bar 7 presents the highest relative standard deviation (about 1,36% of the axial force value). This result would not be of easy determination using conventional finite element techniques.

```
Elemento 0:  Tensao calculada = 6  0
Elemento 1:  Tensao calculada = 4  0
Elemento 2:  Tensao calculada = 4  0.0086785
Elemento 3:  Tensao calculada = 4  0
Elemento 4:  Tensao calculada = 5.65685  -0.0221034
Elemento 5:  Tensao calculada = -5.65685  0.0221034
Elemento 6:  Tensao calculada = -2  -0
Elemento 7:  Tensao calculada = 6  0.0818713
Elemento 8:  Tensao calculada = 2.83315  0.0192891
Elemento 9:  Tensao calculada = -3.99294  0.0370206
Elemento 10: Tensao calculada = -6  -0
Elemento 11: Tensao calculada = -8.48365  0.0389415
Elemento 12: Tensao calculada = -5.65685  -0

Vetor de deslocamentos resultantes =

u[0]=0  0
u[1]=0  0
u[2]=0  0
u[3]=-0.0016  0
u[4]=0.0012  0
u[5]=-0.00346273  2.15046e-06
u[6]=0.002  0
u[7]=-0.00812548  6.40233e-06
u[8]=0.0028  1.7357e-06
u[9]=-0.0139197  -2.59929e-05
u[10]=-0.00153133  7.84972e-06
u[11]=-0.00732548  -4.42069e-06
u[12]=-0.0012  -0
u[13]=-0.00426274  -1.37984e-05
u[14]=-0.0008  0
u[15]=-0.00306274  8.84137e-06
```

3.2 Two-dimensional isoparametric element.

The problem consisted of a single element, with either a probabilistic load (plus of minus one percent) or a probabilistic size (the width varying plus or minus half percent). The results, for the three options, deterministic, varying load or varying width, are shown next:

Deterministic	Varying Load	Varying Width
0	0 ± 0	0 ± 0
0	0 ± 0	0 ± 0
20	20.0005 ± 0.147214	20.9039 ± 1.88914
5.77645e-16	0.00143169 ± 0.132165	1.35551 ± 1.97109
20	19.9979 ± 0.145676	19.0214 ± 1.16508
-5	-4.99829 ± 0.159795	-2.93172 ± 1.81429
0	0 ± 0	0 ± 0
-5	-4.99982 ± 0.0759411	-5.27838 ± 0.862019

The above results might lead to incorrect conclusions, such as the mean plus deviation not covering the deterministic solution. As in the fifth displacement, for varying width, -2.93172 minus 1.81429 does not add to -5.

The results, however, show interesting behavior that can best be seen on the interval of possible resultant solutions:

Deterministic	Varying Load	Varying Width
0	[0,0]	[0,0]
0	[0,0]	[0,0]
20	[[19.4391,20.5609]	[10.5513,26.9434]
5.77645e-16	[-0.578156,0.578156]	[-15.9084,7.46965]
20	[19.5424,20.4576]	[15.2713,28.466]
-5	[-5.49019,-4.50981]	[-18.9973,1.39452]
0	[0,0]	[0,0]
-5	[-5.247,-4.753]	[-11.9851,-1.61036]

It can be observed that the probabilistic nature of loading does not affect seriously the results, or at least not as seriously as uncertainties about the geometric properties of the specimen.

4 Final remarks.

The developed software presents promising results for structural stress-strain analysis. This software allows the user to know how small assembly imperfections, or small differences in the prescribed loading or material properties, might change the internal stresses in the structure. More precisely, this program allows the user to know the probability density functions of the variables involved as a function of the probability density functions of the project variables.

The proposed methodology of finite element object oriented programming has proved its usefulness through the ease of programming and code maintenance for the generated code. The addition of new elements is quite simple and requires almost no additional programming.

The analysis of a very simple example shows a dependency on geometric variables that is more important than loading uncertainties. Further work will cover this aspect of the probabilistic nature of the stress-strain problem.

References

[1] P. L. Meyer, Probabilidade: aplicações à estatística. Livros Técnicos e Científicos, Rio de Janeiro, 1981.

[2] F. C. M. Menandro and J. C. F. de Oliveira, Object Oriented Finite Element Implementation, *XXVII Congresso Brasileiro de Engenharia Mecânica*, Associação Brasileira de Ciências Mecânicas - ABCM,(2003).

[3] H. M. Deitel and P. J. Deitel, C++ como programar. 5. ed. Porto Alegre: Bookman, 2006.

[4] B. Stroustrup, A Linguagem de Programao C++, Ed. Bookman, 2005.

[5] G. Booch, Object oriented design with applications. Redwood City: Benjamin/Cummings, c1991. 580p.

[6] J. N. Reddy, An Introduction to the Finite Element Method, 2005

[7] K.-J. Bathe, Finite Element Procedures, Prentice-Hall, 1995.

[8] A. Haldar and S. Mahadevan, Reliability Assessment Using Stochastic Finite Element Analysis, Wiley, 2000.

[9] J. C. A. Sussekind, Curso de Análise Estrutural, Vol. 1 a 3. Editora Globo, Porto Alegre - Rio de Janeiro, 1983.