ELSEVIER

Advances in Engineering Software 32 (2001) 927–935

ADVANCES IN ENGINEERING SOFTWARE

www.elsevier.com/locate/advengsoft

A C++ object-oriented programming strategy for the implementation of the finite element sensitivity analysis for a non-linear structural material model

Lluís Gil, Gabriel Bugeda*

Department of Strength of Materials and Structures in Engineering, Technical University of Catalonia UPC, Campus Nord UPC, Mòdul C1, C/ Gran Capità s/n, 08034 Barcelona, Spain

Received 6 September 2000; accepted 26 June 2001

Abstract

The Finite Element Method (FEM) has become the most popular numerical method for solving a wide variety of complex engineering problems. However, from the programming point of view, when a FEM program has a lot of computational capabilities it is very difficult to maintain and enlarge the program codes. Recently the object-oriented programming paradigm has become a powerful method for overcoming such difficulties. This paper contains the description of an open environment for the FEM, written in C++, with explanations about how the sensitivity analysis and the non-linear material behaviour (damage models) have been taken into account. © 2001 Elsevier Science Ltd. All rights reserved.

Keywords: Sensitivity analysis; Non-linear analysis; Object-oriented programming

1. Introduction

The continuous increase of the capabilities of hardware and software tools allows to solve very complex structural problems. Nowadays, the use of new materials, new industrial processes and coupling between different physical phenomena can be considered altogether. This has produced a progressive substitution of the traditional experimental methods by the less expensive numerical analysis.

Since the very beginning, the task of programming the Finite Element Method FEM has been increasing in difficulty over the years. The more that engineers want to compute the more difficult it becomes to implement. The reasons rise in the difficulty of the mathematical boundary problems and in the language and programming strategies.

In this paper the authors present an overview of the FEM programming drawbacks and comment their experience with different programming strategies. Finally, an object-oriented approach for the implementation of non-linear material behaviour and its sensitivity analysis is described.

* Corresponding author.
E-mail address: bugeda@cimne.upc.es (G. Bugeda).

2. FEM from object-oriented programming

The problem of programming FEM with object-oriented techniques has frequently been treated in literature. Forde [1] made the first and notable approach to the problem statement and solution, and it was also followed by contribution [2]. Zimmermann and Dubois-Pèlerin presented the main advances during the last decade with a very active team in this field (see Refs. [3–8]). Finally, some references must also be made to Kong [9] and Mackie [10]. Most of the mentioned authors offered their personal point of view with regards to object design in FEM for problems, which range from linear elastic to non-linear and dynamics. All the contributions have been very valuable and rich, and the scope goes from philosophical discussions linking FEM and objects to the design of a FEM shell as a natural language for problem-solving in engineering. Moreover, the natural approach to the problem depends on personal views, so that the final implementation is not an easy task. Notice that, on one hand, FEM behaves as a procedural method with a clear flow but, on the other, the method contains abstract data like nodes, material properties, etc. In spite of the advantages of object-orientation, this approach emphasises data abstraction and the programmer tends to forget that FEM is a procedure, hence FEM should be programmed balancing both concepts.

PII: S0965-9978(01)00039-4

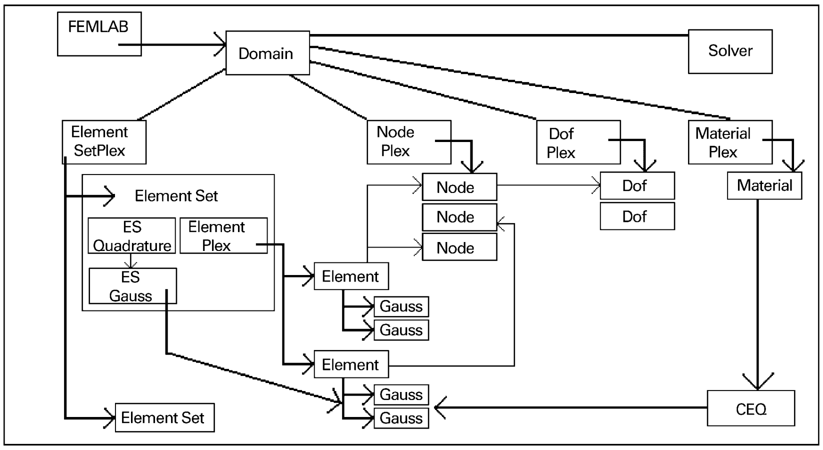

Fig. 1. Relationship between the different FEM classes.

3. The OO methodology in FemLab

FemLab is an object-oriented finite element framework. FemLab was developed [11] in the International Centre for Numerical Methods CIMNE and it becomes a good platform for further developments. FemLab provides a basic general structure through a hierarchy of classes that allows the programmer to build a FEM code adapted to each necessity. The programmer selects the type and the part of the code he needs depending on his mathematical model. He/she chooses the appropriate classes and modifies them conveniently. The encapsulation property provides the security that the new code will not interfere with the rest of existing classes. In addition, the programmer can create completely new classes or can derive some others using the inheritance properties and, finally, he/she can redefine some functions using the polymorphism property and help to maintain a coherence in the lecturing of the code. Therefore, it can be said that the FemLab code was written with the objective of exploiting as much as possible the possibilities of OO and C++.

The original version of FemLab allowed for the static linear analysis in one, two and three dimensions with a big variety of elements. Inside of FemLab there are complex programming concepts like: the Sets of data, Plexes or linked lists of objects, libraries of vectors and matrices, a symbolic language for the matrix operations and, specially, the hierarchy of classes that represents the knowledge of the FEM.

The FemLab context provides a code with a hierarchy of classes where the main relations are established in Fig. 1.

There are three main classes:

(i) FemLab class that performs the analysis of the problem. See Fig.2.
(ii) Domain class that contains the problem data.
(iii) Solver class that solves a system of equations.

The Domain class contains lists of elements, nodes, degrees of freedom, materials and the solver. Each list contains a different type of object. The use of an object-oriented paradigm makes it possible to create a structure that is logic and consistent with the finite element theory that has been established at the moment of defining the relations among: sets of elements, elements, integration points, constitutive equations and materials, as well as between nodes and elements.

The list of degrees of freedom, DofPlex, contains all the existing degrees of freedom in the finite element mesh. For example, in a plane strain elasticity problem each node has

```
class FemLab // Master class of the Finite Element Laboratory
{
  Domain*  Mesh; // The mesh data

  void     ConstructMesh            ();    // read mesh data
  void     StartNewInterval         ();    // following and advance and correction procedure
  void     AdvanceAndPredict        ();
  void     UpdateDofs               ();
  void     ApplyBoundaryConditions ();
  void     StartNewIteration        ();
  void     AssembleMatrixes         ();
  void     ComputeUnknowns          ();
  int      HasConverged             ();
  void     OutputResults            ();    // the end
};
```

Fig. 2. Class FemLab contains the whole Finite Elements problem.

```
        ArcType arc_type;         // type of arc-length
        void ArcLength(void);     // procedure for computing arc-length
```

Fig. 3. New arc-length variables and functions defined inside Domain class.

two possible movements. The degrees of freedom are related with the unknowns vector of the Solver object.

The NodePlex list contains all the nodes of the finite element mesh, and each node has a pointer to the corresponding degrees of freedom. The boundary conditions are applied to the degrees of freedom through the existing relation with the nodes.

The ElementSetPlex list contains sets of elements, which, in turn, are lists of elements. Each set of elements has a common integration rule. In addition, each element of the mesh contains the jacobian of the isoparametric transformation in order to facilitate the computation of the stiffness matrix and the stresses at the integration points. Due to that, each Set of elements contains different Elements and each Element contains different integration points.

In order to evaluate the stresses at the integration points it is necessary to know the constitutive equation, which is controlled by the CEQ object. This is related with the Material class where the intrinsic properties of the material like the Young modulus, the Poisson ratio, the density, specific heat, etc. are stored.

The definition of a Solver associated to the domain but independent from the data of the structure represents a clear advantage in the Input of the data and in the manipulation of the code. This allows to programme different solvers independently of the data. Likewise, the possibility of using symbolic expressions in the operations between matrices provides a very clear code.

A possible drawback of the use of the object-oriented programming for the FEM is the abuse of the definition of objects. In this case, the intrinsic sequential characteristic of the FEM is diluted inside the objects and their complexity becomes more an inconvenience than a facility. Nevertheless, C++ allows one to develop the sequential programming together with the objects organisation and, consequently, to obtain an equilibrium between objects and procedures.

The protection of data is a fundamental aspect of producing maintainable code, but this advantage can appear to be an inconvenience, leading to an increase in function calls. However, these perceived difficulties can be overcome using the protected keyword. Furthermore, inline.

4. Implementation of non-linear material behaviour

4.1. Arc-length strategy

When the structural material has a non-linear behaviour the equilibrium equations are normally solved in an incremental-iterative way. When the material model behaviour represents softening phenomena the use of arc-length strategies is almost mandatory in order to achieve convergence. In this case, the equilibrium equations of the problem can be written as, Crisfield [12]:

$$\mathbf{K}_s(\mathbf{u})\mathbf{u} = \lambda\mathbf{f} \tag{1}$$

where $\mathbf{K}_s$ is the equilibrium secant matrix, $\mathbf{f}$ is the nodal forces vector, $\mathbf{u}$ is the unknown displacements vector and λ is the arc-length parameter. This equilibrium equation is used together with the arc-length displacement control condition:

$$g(\mathbf{u}(\lambda)) = 0 \tag{2}$$

Notice that, compared with a traditional equilibrium problem, the control condition $g(\mathbf{u})$ applied on the displacements introduces an additional restriction but, on the other side, an additional unknown λ is also present. This balances the total number of equations with the total number of unknowns.

The inclusion of the arc-length condition allows solving the equilibrium problem with the corresponding applied forces, but the amount (level) of the applied forces is controlled by the displacement condition. Different arc-length strategies exist depending on the conditions applied to the displacements. The corresponding organisation of classes is shown in Fig. 5. The arc-length parameter λ modifies the level of the external applied forces and, therefore, the functions that work with this parameter need to have access to the pseudo-time variable that controls the increment of loads. This variable is contained in one of the main objects of the system, Domain. Therefore, this object has to contain all the functions related with the arc-length. This can be achieved by adding the next part of code, see Fig. 3.

Notice that in order to compute the arc-length condition (see Fig. 4) it is necessary to have access to the displacements that are stored in the degrees of freedom, the Dofs object. Taking into account that Domain contains this object, this access is trivial from FemLab. Nevertheless, if this displacements where stored only in the nodal objects, the corresponding access could also be obtained with a very small increase of the computational time.

In Fig. 4 also appear different types of arc-length conditions: spherical, normal plane and displacement control. Everyone of them has a different expression for Eq. (2), see Ref. [12].

4.2. Damage model with softening

The stress–strain relations that control the internal response of the structure material define the constitutive

```
void Domain::ArcLength(void)
{
  switch(arc_type)   // switch ArcType
  { case spherical:       delta=arc_spherical_path();         break;
    case normal_plane:    delta=arc_normal_plane_path();      break;
    case disp_control:    delta=arc_disp_control();           break;
    default:              Error("such arc length not implemented yet"); break;
  }
  // updating the increment
  int idx = 0; Dof* dofP;
  while(dofP=dof.next(idx)) dofP->dx(ci)+=delta*dofP->arc_x();

  // updating load
  idx = 0;
  while(dofP=dof.next(idx))
    { double dload=delta*dofP->arc_l();  dofP->ApplyExternalLoad(dload); }
 return;
}
```

Fig. 4. Computing the arc-length condition in the arc-length method.

behaviour. In a damage-softening model the non-linear response is defined through the use of a damage internal variable d.

$$\boldsymbol{\sigma} = [1 - d]\mathbf{D}\boldsymbol{\epsilon} \tag{3}$$

where $\boldsymbol{\sigma}$ is the stress vector, $\boldsymbol{\epsilon}$ is the strains vector, $\mathbf{D}$ is the elastic constitutive matrix and d is the damage internal variable. The damage model establishes an evolution law of the internal variable in agreement with a threshold stress norm and the energy dissipation (see Lubliner et al. [13]).

The damage parameter evolves in terms of the strain state. In this work the following evolution equation has been used in Oliver et al. [14]:

$$d = 1 - \frac{\tau^*}{\tau} e^{A\left(1 - \frac{\tau}{\tau^*}\right)} \tag{4}$$

where τ is a norm of the stress state, in this case, it is equivalent to the deformation energy, τ^* is the threshold value of this norm (a material property related to the maximum compression stress f_c) above which the material starts to damage and A is a softening parameter which depends on the characteristic length of the elements l_c (a measure of the mesh elements size), fracture energy G_f, maximum traction stress f_t and Young modulus E as follows:

$$A = \frac{2l_c f_t^2}{2G_f E - 1} \tag{5}$$

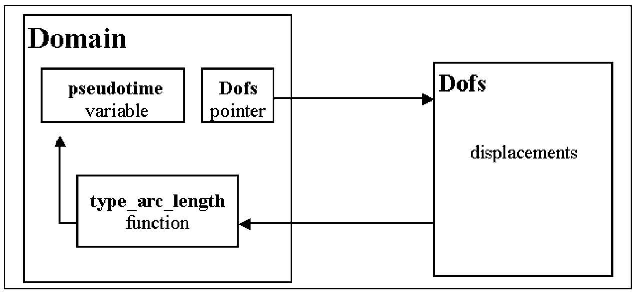

Fig. 5. Relationship between classes to implement an arc-length strategy.

The dependence of A on the mesh size l_c accounts for the proper numerical structural response with respect to the size of the finite elements. It means that, in practice, the constitutive equation at each element depends on its size. In case of shape sensitivity analysis this effect must be taken into account because a change on the structural shape can affect the element sizes and, in consequence, also the stress values.

From the programming implementation point of view, the non-linearity of the material directly affects the material because it is necessary to introduce new variables like the maximum allowable stresses f_t and the fracture energy G_f. In addition, the non-linearity affects the constitutive behaviour of the elements in a double sense: first, in the computation of the stiffness matrix used for the resolution of the system of incremental equations and, second, in the computation of the stresses performed for the control of the residual.

Due to that, a special class for the damage material named DamageM has been defined. This class contains information about the elastic properties and the internal variables that characterises the damage level, the maximum allowable traction (f_t) and compression (f_c) stresses, and the fracture energy G_f (Fig. 6).

Another class, DamageCEQ, has been defined for the constitutive behaviour. This class contains all the internal variables and all the functions of the formulation of the damage constitutive equations including the failure criterion, the internal damage variable d and the A parameter of the model Fig. 7.

This constitutive behaviour is produced at each integration point. Other classes related with the elements have been defined for the evaluation of the stiffness matrix and the residual forces. Briefly, two main classes have been created for the control the damage behaviour. The class of the damage elements Set, DamageElmSet, contains the general integration rule and all the elements. On the other side, the DamageElm class is the finite element of damage type.

The DamageElm class contains the procedures and variables in charge of the computation of the secant stiffness matrix (see Fig. 8) and the pointers to the objects of the

```
double ft, fc;     // max stress tension/compression
double Gfx;        // fracture energy per area
```

Fig. 6. Material variables included in DamageM class.

```
double A_val;                   // store A value
double tau_max;                 // maximum limit stress
double disipation;              // store energy disipated

double evaluate_d(Matrix& strain, Matrix& dstrain);   // evaluate d
double evaluate_A(void);                              // update A
```

Fig. 7. Constitutive equation variables and methods included in DamageCEQ class.

constitutive equation. As a particular point, we want to emphasise that the characteristic length l_c, which is used in the damage formulation, is associated to the element and not to the object that contains the constitutive equation. This organisation seems to be the most adequate, even taking into account that l_c is used for the computation of the d parameter of the formulation and that the element is not using l_c anymore.

The scheme of classes for the implementation of the damage model is shown in Fig. 9.

The implementation of the classes related with the non-linear behaviour of the material has been done through an inheritance of the abstract classes CEQ, Material, Element Set and Element. As a consequence, they are independent of the classes that define the elastic behaviour of the material, as is shown in Fig. 10. This point is worthy of discussion because any non-linear constitutive equation is linear until some threshold value is reached. Therefore, the elasticity is directly related with the non-linearity. As a consequence, a possible alternative would consist in deriving the non-linear classes (for instance, damage or plasticity) from the elastic classes as is shown in Fig. 11. Nevertheless, this possibility has not been used because the damage theory defines its own behaviour scheme in the linear as well as in the non-linear ranges of behaviour. Due to that it seems logical that this class should exist by itself and not as an inheritance of the elasticity. The last alternative (Fig. 12) would be to consider elasticity as an inheritance of the non-linear behaviour. This last scheme would be much more coherent because the inheritance would work as a particular case of a more general class that is the non-linear one. Nevertheless, due to the fact that many different non-linear behaviour models exist, this would produce an unnecessarily complex hierarchy of classes.

5. Implementation of the sensitivity analysis

The sensitivity analysis provides the derivatives of the structural response with respect to the design variables. In order to include this analysis into the FEM code, the internal structure of FemLab needs to be modified at different levels:

- The DSAFemLab class is created as a derivation of the FemLab one. FemLab performs the structural analysis, and DSAFemLab performs, in addition, the sensitivity analysis (see Fig. 13). This class contains the general algorithm for the structural analysis and its sensitivities with respect to the design variables.
- The DSAMaster, DSAddm, DSAadm classes are created. The DSAMaster is a main class that contains the fundamental methods and data. Some of them are declared as virtual and will be specified later in the derived classes DSAddm and DSAadm. In particular, the DSAddm class is in charge of the application of the direct derivation method for the computation of the sensitivity analysis and, due to that, it contains the strategy for the computation of the pseudo-load vector, etc.
- The following elemental classes are derived for the computation of the system of equations that provides

```
MatrixT DamageElm::SecantstiffnessMatrix()
{
   Matrix D = Set()->Mater()->giveConstitutiveMatrix(dtype);
   MatrixT K;
   for(int iGauss=1; iGauss<=NIntPt; iGauss++)
    {
      Gauss* Gp = quad->gauss(iGauss);
      DamageCEQ* Ceq=(DamageCEQ*)Gp->ceq;
      double d=Ceq->get_damage();
      Matrix B  = B_Matrix_At(Gp);
      K += (1.-d) * B.t() * D * B * Gp->dVolume;
    }
  return K;
}
```

Fig. 8. How to compute the secant matrix and return it.

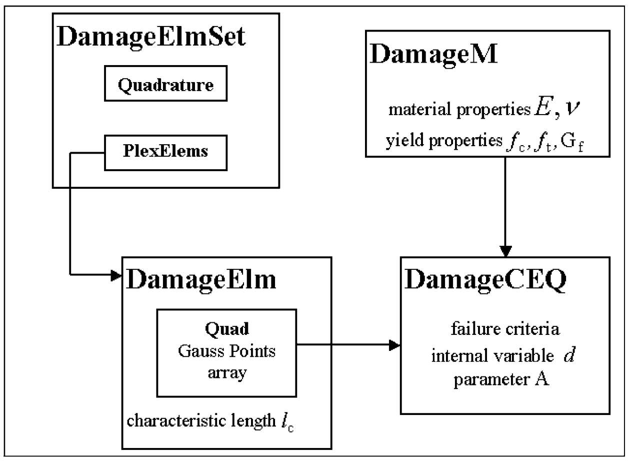

Fig. 9. Classes for the non-linear damage model.

the sensitivities in linear and non-linear regime: SolidDSAElmSet, SolidDSAElm, PlasticDSAElmSet, PlasticDSAElm, DamageDSAElmSet and DamageDSAElm (see Fig. 14 as an example). The SolidDSALoadSet and SolidDSALoad classes are created to deal with the possible modification of loads.

For the linear case, a classic sensitivity analysis based on the direct differentiation methods has been implemented. Therefore, the sensitivities can be obtained from the solution of a system of equations similar to the equilibrium one and using a pseudo-load vector:

$$\mathbf{K}\frac{\mathrm{d}\mathbf{u}}{\mathrm{d}q} = \mathbf{f}^* \tag{6}$$

where $\mathbf{K}$ is the elastic stiffness matrix of the structural system, $\mathrm{d}\mathbf{u}/\mathrm{d}q$ are the derivatives of the nodal displacements with respect to the design variable q and, finally, the pseudo-load vector is computed as:

$$\mathbf{f}^* = \frac{\mathrm{d}\mathbf{f}}{\mathrm{d}q} - \frac{\mathrm{d}\mathbf{K}}{\mathrm{d}q}\mathbf{u} \tag{7}$$

For the non-linear case it has been necessary to differentiate the equilibrium equations, including the damage model, and the arc-length condition:

$$\begin{bmatrix} \mathbf{K}_\mathrm{T} & -\mathbf{f} \\ \mathrm{d}g/\mathrm{d}q & 0 \end{bmatrix} \begin{Bmatrix} \mathrm{d}\mathbf{u}/\mathrm{d}q \\ \mathrm{d}\lambda/\mathrm{d}q \end{Bmatrix} = \begin{Bmatrix} \mathbf{f}^*_\mathrm{new} \\ 0 \end{Bmatrix} \tag{8}$$

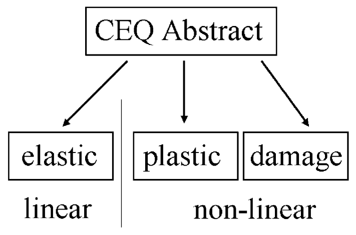

Fig. 10. Choosen one.

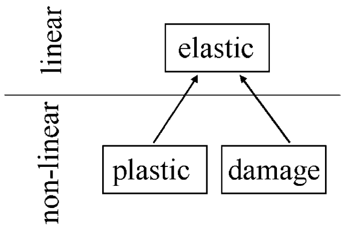

Fig. 11. Alternative 1.

where $\mathbf{K}_\mathrm{T}$ is the tangent stiffness matrix, $\mathbf{f}$ is the nodal forces vector, $\mathbf{f}^*_\mathrm{new}$ is an extended pseudoload vector, and $\mathrm{d}g/\mathrm{d}q$ is the derivative of the arc-length condition with respect to q. The unknowns vector contains the sensitivities of the displacements and the load factor λ.

The detailed description of this formulation is not the objective of this paper, and as a result the reader is addressed to Refs. [15,16].

Finally, few of the existing classes need to be modified in order to store the new information provided by the sensitivities: Dofs, CEQs and the integration rules.

As an example, Fig. 15 shows the hierarchy defined for the linear elements that can be generalised for the rest of elements. Notice that the dark parts are the modified or added ones, and the arrows represents the inheritance between classes. This structure is organised in different isolated parts that are related between them in a logical way.

The main functions of the algorithm have to call the parent objects in order to avoid rewriting the sequential part of the code that performs the finite element analysis. This brings the necessity of defining pure virtual functions in the parent classes that are used by the compiler to announce that a call to the correct function included in a children class must be done during the execution time. In an inevitable way, this implies an increase in the declaration of functions in the child classes that, in fact, do not do anything.

Finally, we want to comment that in the Dofs class the computation of the sensitivity analysis produced an increase of the stored data, because not only the nodal displacements but also their derivatives with respect to the design variables must be stored. These vectorial magnitudes must be stored in a dynamical way reserving only the strictly necessary amount of memory depending on whether the analysis is with or without sensitivities.

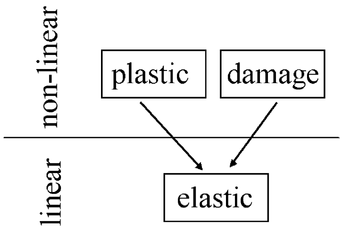

Fig. 12. Alternative 2.

```
class DSAFemLab : public FemLab {
  public:
    int type_of_analysis;         // ddm,adm strategy
    DSAMaster* DSALab;            // object that controls the sensitivity procedure

    void ReadSensitivityData(void);
    void ComputeSensitivity(void);
    void ResetPseudoload(void);

    void     UpdateDofs   ();     // redefine the function through polimorfism
};
```

Fig. 13. Sensitivity analysis main class is an inheritance of FemLab class (see Fig. 1).

6. Assessment example

The quality and the reliability of the formulations, and its implementation, proposed in this paper are assessed here through the resolution of a simple test case.

This test case studies the quality of the formulation for the case of a concrete iso-static beam with a bending behaviour. The selected design variable is the thickness of the beam. This test case shows the application of the formulation to predict the behaviour of a beam that, due to building errors, has a thickness smaller than the designed and analysed one. This test case is analysed with a damage model with-strain softening and the main aim is to predict the modification of the peak load produced by the building error.

The geometry and the applied load of this test case are described in Fig. 16 and 17. Fig. 17 also describes the symmetry approach used for the analysis of the structural problem.

The different data used for the structural analysis is the following:

- A plane stress model with a depth of 50 cm has been assumed.
- Arc-length method controlling the displacements of node 11 (see Fig. 18) with increments of $\Delta l = 0.006$ cm at each step of the solution process.
- The convergence criteria for the solution of the equilibrium problem has been defined in terms of the ratio between the norm of the residual forces and the norm of the external forces. This ratio has been limited to a 1%. The finite element mesh shown in Fig. 18 contains 40 8-noded quadratic elements.
- The design variable is the thickness of the beam.
- The material properties are in Table 1.

Two different analyses have been performed:

(i) The first analysis corresponds to the original structure. For this structure the sensitivities of the displacements with respect to the design variable have been computed at each equilibrium point.
(ii) The second analysis corresponds to a modified (perturbed) structure that has been obtained by applying a reduction of 2.5% to the design variable. This means that the thickness of the modified beam is 5 cm smaller than the original one.

In order to check the quality of the proposed formulation the following curves have been compared:

1. Displacement–load curve corresponding to node 21 of the original structure.
2. Displacement–load curve corresponding to the same node of the modified structure obtained by a direct analysis.
3. Displacement–load curve corresponding to the same node of the modified structure obtained by a first order projection using the results of the original one and its sensitivities.

The comparison of the response curves between the original, modified and projected structural behaviour is shown in Fig. 19. It should be noted that the curves projected and modified superpose quite well. This reveals a good behaviour of the proposed formulation for this test case. In particular, the projected curve allows a very good

```
class DamageDSAElm : public DamageElm {
  public:
    MatrixT  DvdstiffnessMatrix(Matrix& Disp);  // dK/dq
    MatrixT  DvdB_Matrix_At(Gauss* Gp);         // dB/dq

    void build_quad_elm_for_dsa(int var_design);        // derivate quadrature

    void compute_secant_pseudoload(int var_design);     // pseudoload
    void compute_stress_sensitivity(int var_design);    // dG/dq at each Gauss Point
};
```

Fig. 14. Inheritance is a powerful property for performing sensitivity analysis from existing classes.

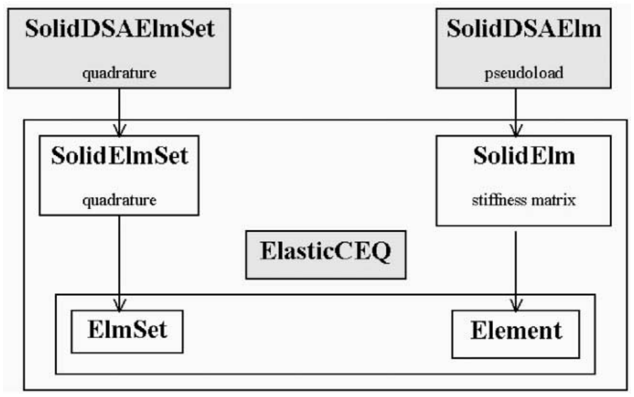

Fig. 15. Modified classes for the sensitivity analysis.

engineering estimation of the ultimate load of the modified structure.

7. Conclusions

The programming techniques used for the implementation of the FEM have always been strongly linked with the available programming languages. Nowadays, the object-oriented programming is an interesting alternative for the production of finite element codes easy to maintain and to expand. In this work, the transition from a linear elastic code to a non-linear material model with arc-length and sensitivity analysis has been presented. The success of this implementation relays in the object-oriented programming.

Generally speaking, most of the authors agree in the correct adaptation of the finite element strategies to the object-oriented coding. Nevertheless, we want to emphasise that due to the implicit philosophy of the FEM, the maximum performance of the object-oriented systems is only achieved if they are combined with some procedural coding parts. In this context, C++ is, nowadays, the best developing language for the FEM.

We have experienced that during the development of object-oriented codes the time dedicated by the programmer to the organisation of the information and its management is much bigger than in the procedural programming case. This is because the object orientation requires a higher level of rigour. There must be taking account that an inconvenient of the object-oriented programming appears when the class organisation is incorrect and an abuse in the number of objects and in the privacy of data is produced. In this sense, the object-oriented programming is not a flexible tool. Nevertheless, the generation of new classes and the expansion of computation capabilities are generally facilitated, particularly thanks to the inheritance.

The results obtained in the solution of the presented example show the good performance not only of the proposed formulation, but also of its implementation using the object-oriented approach. Finally, we conclude that the expansion possibilities of the object-oriented codes are much bigger than in the code produced using procedural languages.

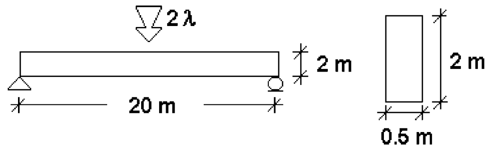

Fig. 16. Geometrical definition of test case.

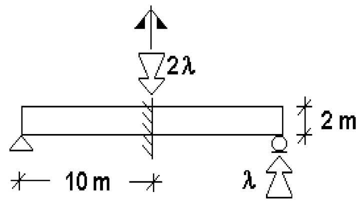

Fig. 17. Applied load and symmetry approach.

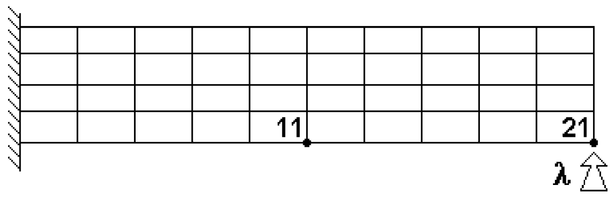

Fig. 18. Finite element mesh for test case.

Table 1
Material properties

Young modulus E (KN/m^2)	2.1×10^6
Poisson ratio	0.2
Maximum compression stress (KN/m^2)	2.0×10^2
Maximum traction stress (KN/m^2)	500.0
Fracture energy (J/m^2)	200.0

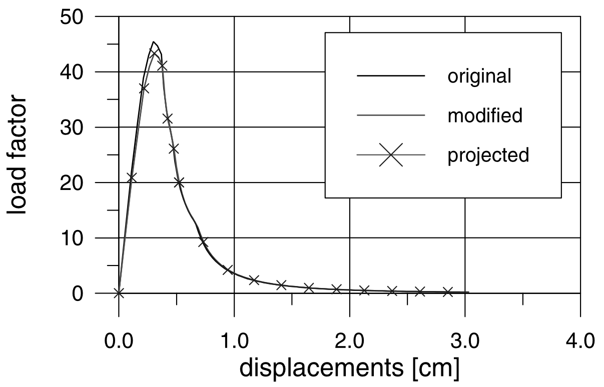

Fig. 19. Superposition of the displacement-load curves corresponding to the original, the modified and the projected structures for test case.

References

[1] Forde BWR, Foschi RO, Stiemer SF. Object-oriented finite element analysis. Comput Struct 1990;34:355–74.

[2] Miller GR. An object-oriented approach to structural analysis and design. Comput Struct 1991;40:75–82.

[3] Zimmermann T, Dubois-Pélerin Y, Bomme P. Object-oriented finite element programming: I Governing principles. Comput Meth Appl Mech Engng 1992;98:291–303.

[4] Zimmermann T, Eyheramendy D. Object-oriented finite element. I Principles of symbolic derivations and automatic programming. Comput Meth Appl Mech Engng 1996;132:259–76.

[5] Dubois-Pélerin Y, Zimmermann T. Object-oriented finite element programming: III An efficient implementation in C++. Comput Meth Appl Mech Engng 1993;108:165–83.

[6] Eyheramendy D, Zimmermann T. Object-oriented finite element. II A symbolic environment for automatic programming. Comput Meth Appl Mech Engng 1996;132:277–304.

[7] Dubois-Pelerin Y, Pegon P. Improving modularity in object-oriented finite element programming. Commun Num Meth Engng 1997;13:193–8.

[8] Dubois-Pelerin Y, Pegon P. Object-oriented programming in nonlinear finite element analysis. Comput Struct 1998;67(4):225–42.

[9] Kong XA, Chen DP. An object-oriented design of FEM programs. Comput Struct 1995;57:157–66.

[10] Mackie RI. Using objects to handle complexity in Finite Element software. Engng Comput 1997;13:99–111.

[11] Galindo M, ‘FemLab v. 1.0’. Technical Report n. IT-114. Ed. Cimne. Barcelona,. 1994.

[12] Crisfield MA. Non-linear finite element analysis of solids and structures. New York: Wiley, 1991.

[13] Lubliner J, Oliver J, Oller S, Oñate E. A plastic damage model for concrete. Int J Solid Struct 1989;25:299–326.

[14] Oliver J, Cervera M, Oller S, Lubliner J. A simple damage model for concrete, including long term effects. In: Proceedings of II International Conference on Computer Aided Analysis and Design of Concrete Structures, vol. 2. Austria: Zell Am See, 1990. p. 945–58.

[15] Bugeda G, Gil L, Oñate E. Structural shape sensitivity analysis for nonlinear material models with strain softening. Struct Optim 1999;17:162–71.

[16] Bugeda G, Gil L. Shape sensitivity analysis for structural problems with non-linear material behaviour. Int J Num Meth Engng 1999;46:1385–404.