非線形有限要素解析におけるオプジェクト指向解法

TANAKA, Shigenori / 前田, 重行 / TAKEDA, Hiroshi / 田中, 成紀 / MAEDA, Shigeyuki / 武田, 洋

(出版者 / Publisher)
法政大学計算センター
(雑誌名 / Journal or Publication Title)
法政大学計算センター研究報告 / Bulletin of Computer Center, Hosei University
(巻 / Volume)
7
(開始ページ / Start Page)
7
(終了ページ / End Page)
13
(発行年 / Year)
1994-03-31
(URL)
https://doi.org/10.15002/00024684

非線形有限要素解析におけるオブジェクト指向解法

田中 成紀、前田 重行
法政大学工学部土木工学科*
武田 洋
法政大学計算センター*

1．はじめに

この論文は従来の有限要素解析ソフトの問題点と、オブジェクト指向プログラミングによって提供されるポテンシャル解法を記述し、オブジェクト指向プログラミングとアプリケーションへの拡張の基礎概念を紹介する。非線形有限要素の基礎は、新しい観点のオブジェクト指向数値解析プログラムを用いて説明する。

さらに、ベクトルやマトリックスクラス等の幾つかのクラスは、基本的な代数マトリックスを計算するために用いる。それらのベクトルやマトリックスクラスは演算子のオーバーロードと多重定義の表記法を用いることによって、明確な構造化プログラムを形成する。

オブジェクト指向プログラミングは非線形有限要素解析プログラムを改良するために用いる事が出来る。有限要素解析プログラムは複雑で間違い易い典型的な例として知られている。従来のソフトの複雑な操作とデータ構造は、基本的な意図を不明瞭にする傾向がある。そのため後の開発と管理をとても困難なものにする。この論文では非線形有限要素解析に対して、オブジェクト指向プログラミングを用いることによる、この手法の有効性及び数値解析における効率性の検討をすることが目的である。

1．1　既存の有限要素解析プログラムにおける問題点

有限要素解析は複雑な工学問題に対する優れた解析手法である。しかし優れた研究者でさえも、この手法を用いるためには技術的な専門知識を必要とする。さらに従来の解析プログラムは、作業目的毎のプログラムの改良が簡単ではないため、開発者はそれらの有限要素ライブラリーを再構築しなければならない。

1．2　ソフト開発者のかかえる主な問題点

実際の工学的問題における新たな問題に対するソフトの要求により、ソフト開発者は既存のプログラムをほとんど改良しなければならない。そして開発者はその管理に多くの努力を必要とする。換言すればソフト開発者がプログラムの再構築において既存のプログラムのどこを利用するかを決定することであり、またそれら新しいソフトの開発に使うツールを決定することである。

＊　〒184　東京都小金井市梶野町３－７－２

1．3　オブジェクト指向プログラミングの利点

最近、数値解析プログラミングにオブジェクト指向法を用いる傾向がある。この傾向はソフト開発者の関心を反映した一般的な技術動向の一つである。従来の手続き型プログラム方法に比較して、オブジェクト指向解法を用いると開発時間とその結果もたらされるプログラムの大きさはかなり減少する。さらにソフト開発者は手続き型言語プログラムにオブジェクト指向概念を取り入れた複合した開発環境を使用する事が出来、既存のプログラムを維持したままオブジェクト指向環境を取り入れる事が出来る。

2．オブジェクト指向方法

C, Pascal や Fortran の様な手続き型言語は、受動的なデータと要素が占めるメモリーを考え、データは関数と手続きによって処理する。データと算法は異なった構造をしているので、それぞれ異なった方法で宣言や定義を行う。手続き型言語との違いは、オブジェクトはデータとメソードと呼ばれる関数を含んでいる事である。オブジェクトはそのメソード自身がそのデータのみを操作する。そして外部関数はこのオブジェクトのデータを変化させることは出来ない。外部関数と他のオブジェクトはメッセージを送ることによってだけ、ある一つのオブジェクトと連結される。一般にメッセージはメソードの呼び出しに使用される。そのためオブジェクトがメッセージ受けると、その内容の手続きの一つを実行する。またデータのカプセル化により好ましくない二次的効果を回避する事が出来る。オブジェクトの構成は図１に示す。

プログラムは相互にメッセージを送りあうことが出来るオブジェクト間の幾つかの伝達を含んでいる。同じメッセージが、このメッセージに対してそれ自身異なった手続きを持つ異なったオブジェクトに送ることができる。オブジェクトの性質によればそれらは同じメッセージに対して異なった挙動を示す、この挙動は多重定義と呼ばれている。‘＋’や‘＝’の様な演算子もまた使用できるメッセージである。それらは新しい方法で全てのオブジェクトに宣言する事が出来る。そして数に対する演算子の本来の意味は特定のオブジェクトのための新しい意味によってオーバーロードされる。このように一つのオブジェクトの一つの演算子は異なった意味を含んで持つことが可能である。

同質のオブジェクトは一つのクラスの中に含まれる。一般的にクラスの宣言はＣの構造体やパスカルのレコード等の型定義ような、様々なデータタイプの構造と多くのメソードを含んでいる。宣言もまたクラスのオブジェクトを初期化したり消去したりするコンストラクターとデストラクターを含んでいる。

このとき初期化されたオブジェクトをインスタンスと呼ぶ。それらのオブジェクトはクラスのデータセットを含み、クラスのメソードの中に記述されている挙動をする。クラスはツリー構造と同様な形式の中に組織化されたクラス階級組織の中に配列される。データと上位クラスのメソードは下位クラスに継承される。クラスの構成は図2に示す。

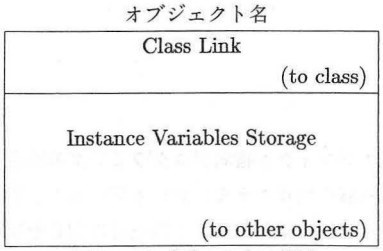

図1　オブジェクトの構成

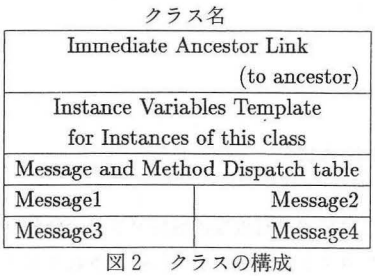

図2　クラスの構成

3．非線形有限要素解析

非線形有限要素解析の定式化、組立や適用に対する従来の方法はオブジェクト指向環境に簡単に移行することが出来る。この節では非線形有限要素法の理論上の基礎と解析問題の構成要素を表すためのクラスの設計を記述する。

3．1　幾何学的非線形問題

変位（ひずみ）の大小に関わらず、内部及び外部の'力'の釣り合い条件は満たされていなければならない。そして、もし変位が有限個の（節点）パラメータ$\mathbf{u}$によって一般の方法で既定されるなら、仮想仕事の原理を用いて必要な釣り合い方程式を得ることが出来る。しかしながら、ここではお互いに関連する'応力''ひずみ'の異なった定義を用いなければならない。これは一般に次式の様に書くことが出来る。

$$\mathbf{R}(\mathbf{u}) = \int_V \overline{\mathbf{B}}^T \boldsymbol{\sigma} dV - \mathbf{f} = \mathbf{0} \tag{1}$$

$\mathbf{R}$は外部と内部の一般的な力の総和を表し、そして$\overline{\mathbf{B}}$はひずみの定義から次式のように定義される。

$$d\boldsymbol{\varepsilon} = \overline{\mathbf{B}} d\mathbf{u} \tag{2}$$

もしひずみが大きければ、バーはひずみが非線形関係で変位に依存するために加えられる。そしてマトリックス$\overline{\mathbf{B}}$は$\mathbf{u}$の関数となる。次式の様に表すと便利である。

$$\overline{\mathbf{B}} = \mathbf{B}_O + \mathbf{B}_L(\mathbf{u}) \tag{3}$$

$\mathbf{B}_O$は線形微小ひずみ解析におけるマトリックスと同じである、そして$\mathbf{B}_L$は変位の関数である。一般に$\mathbf{B}_L$はこの様な変位の関数であることがわかる。もしひずみが微小ならば、一般に弾性関係は次式のように書くことが出来る。

$$\boldsymbol{\sigma} = \mathbf{D}(\boldsymbol{\varepsilon} - \boldsymbol{\varepsilon}_O) + \boldsymbol{\sigma}_O \tag{4}$$

$\mathbf{D}$は普通の弾性マトリックスである。

もしニュートン・ラプソン法を用いるならば、ここで説明しているような$d\mathbf{u}$と$d\mathbf{R}$間の関係を必要とする。この様に$d\mathbf{u}$関して公式(1)の変分をとれば次式を得る事が出来る。

$$d\mathbf{R} = \int_V d\overline{\mathbf{B}}^T \boldsymbol{\sigma} dV + \int_V \overline{\mathbf{B}}^T d\boldsymbol{\sigma} dV = \mathbf{K}_T d\mathbf{u} \tag{5}$$

公式(4)と(2)を用いると次式のようになる。

$$d\boldsymbol{\sigma} = \mathbf{D} d\boldsymbol{\varepsilon} = \mathbf{D}\overline{\mathbf{B}} d\mathbf{u} \tag{6}$$

もし公式(3)が成り立つなら次式となる。

$$d\overline{\mathbf{B}} = d\mathbf{B}_L \tag{7}$$

それ故に次式を得る。

$$d\mathbf{R} = \int_V d\mathbf{B}_L^T \boldsymbol{\sigma} dV + \overline{\mathbf{K}} d\mathbf{u} \tag{8}$$

$\overline{\mathbf{K}}$は次式に表す。

$$\overline{\mathbf{K}} = \int_V \overline{\mathbf{B}}^T \mathbf{D}\overline{\mathbf{B}} dV = \mathbf{K}_O + \mathbf{K}_L \tag{9}$$

$\mathbf{K}_O$は一般の微小変位剛性マトリックスを示す、即ち次式となる。

$$\mathbf{K}_O = \int_V \mathbf{B}_O^T \mathbf{D}\mathbf{B}_O dV \tag{10}$$

マトリックス$\mathbf{K}_L$は大変位によるもので、それは次式によって与える。

$$\mathbf{K}_L = \int_V (\mathbf{B}_O^T \mathbf{D}\mathbf{B}_L + \mathbf{B}_L^T \mathbf{D}\mathbf{B}_L + \mathbf{B}_L^T \mathbf{D}\mathbf{B}_O) dV \tag{11}$$

$\mathbf{K}_L$は初期変位マトリックス、大変位マトリックス等としてよく知られている、そして$\mathbf{u}$の中に線形と2次のある条件だけを含む。公式(8)の第1項は一般に次式のように書かれる。

$$\int_V d\mathbf{B}_L^T \boldsymbol{\sigma} dV = \mathbf{K}_\sigma d\mathbf{u} \tag{12}$$

$\mathbf{K}_\sigma$は応力レベルに関する対称マトリックスである。このマトリックスは初期応力マトリックスや幾何剛性マトリックスとして一般によく知られている。以上の結果から次式を導く。

$$d\mathbf{R} = (\mathbf{K}_O + \mathbf{K}_\sigma + \mathbf{K}_L) d\mathbf{u} = \mathbf{K}_T d\mathbf{u} \tag{13}$$

この様に$\mathbf{K}_T$は全体の接線剛性マトリックスである。

この前述の全式はオブジェクトとして考えられるクラスを含んでいる。力、変位、ひずみと応力はクラスを用いて定義する。この理論での重要な概念は空間内での参考点、即ち節点の考え方である。力、変位、境界条件、そして幾何学的領域でさえ節点に依存する。有限要素の記述を始めるためのクラスは節点のクラスである。節点の固定は自由度と関係づけられた変位を固定することで表す。

［節点クラス］ 節点クラスにあるメソードはデータファイルへの読み書きの手続きを行い、さらに、番号付けされた節点への自由度の割り当てや、計算された変位ベクトルの節点に対する割り当てを行う。節点クラスの構成は図3に示す。

［変位境界条件クラス］ 変位境界条件クラスは節点の境界条件を表すために用いる。変位境界条件は固定状態または幾つかの拘束状態によって表す。このクラスのオブジェクトは節点がどのように拘束されているか表す。自由度を固定するために拘束されている節点にメッセージを送る一つのメソードはこのクラスに置く。変位境界条件クラスは図4に示す。

［力学的境界条件クラス］ 力学的境界条件は適用された荷重ベクトルと載荷された節点を参照することによって表す。このメソードの基本的な操作は節点にメッセージを送ることやその自由度を調べること、そして与えられた荷重を適切な位置に加えるために荷重ベクトルクラスにメッセージを送ること含む。力学的境界条件クラスは図5に示す。

節点

(to object)	
num　dof x　u	
read	AssignDOF
write	AssignDisp
writeDisp	

図3　節点クラス

変位境界条件

(to object)	
num fixity nodeNum	
read	ReduceDOF
write	

図4　変位境界条件クラス

力学的境界条件

(to object)	
num f nodeNum	
read	AddToLoad Vector
write	

図5　力学的境界条件クラス

3．2　非線形問題

3．2．1　非線形弾性

もし非線形応力－ひずみ関係を用いる場合は、$\mathbf{D}=\mathbf{D}(\boldsymbol{\sigma})$は式(14)にあるように増分弾性マトリックスとなる。

$$\mathbf{D}_T = \frac{\partial \boldsymbol{\sigma}}{\partial \boldsymbol{\varepsilon}} \tag{14}$$

$\mathbf{D}_T$は接線弾性マトリックスとして一般によく知られている。

3．2．2　塑性

固体の塑性的な挙動は非一義的な応力－ひずみ関係による非線形弾性の挙動とは対称的な特徴がある。事実、塑性の一つの定義は荷重除去時に非可逆のひずみが存在する事である。

降伏曲面　応力σが一般的な降伏条件を満足したときのみ降伏が生じるということは実験事実として広く一般に仮定されている。

$$F(\boldsymbol{\sigma},\kappa)=0 \tag{15}$$

流れの法則　Von Misesは初めて塑性ひずみ増分の限界を定める基本的な挙動は、降伏曲面と関係があると言うことを提案した。

$$d\boldsymbol{\varepsilon}^p = d\lambda \frac{\partial F}{\partial \boldsymbol{\sigma}} \tag{16}$$

dλはまだ決定されていない比例定数を表す。この法則はn次元の応力－ひずみ空間における降伏局面に対する塑性ひずみ増分ベクトルの垂直性原理として一般によく知られている。

$$Q = Q(\boldsymbol{\sigma},\kappa) \tag{17}$$

法則による制約は、式(16)と同様な塑性ひずみ増分を定義するような塑性ポテンシャルによって緩和する事が出来る。

$$d\boldsymbol{\varepsilon}^p = d\lambda \frac{\partial Q}{\partial \boldsymbol{\sigma}} \tag{18}$$

$Q=F$の特別の場合は結合塑性法則として一般によく知られている。この関係が満足されないとき、法則は非結合と呼ぶ。

増分応力－ひずみ関係　応力の微小な増分の間に生じるひずみの変化は弾性と塑性の部分に分解できると仮定すると次式を得る。

$$d\boldsymbol{\varepsilon} = d\boldsymbol{\varepsilon}^e + d\boldsymbol{\varepsilon}^p \tag{19}$$

塑性関係式(18)を用いることにより式(19)を次式の様に書くことができる。

$$d\boldsymbol{\varepsilon} = \mathbf{D}^{-1} d\boldsymbol{\sigma} + \frac{\partial Q}{\partial \boldsymbol{\sigma}} d\lambda \tag{20}$$

塑性降伏が生じるとき、その応力は式(15)によって与えられる降伏曲面にある。この式を微分すると次の式を得る。

$$\left\{\frac{\partial F}{\partial \boldsymbol{\sigma}}\right\}^T d\boldsymbol{\sigma} - A d\lambda = 0 \tag{21}$$

未決定の定数dλはここで消去することが出来（理想塑性材料において0であるかもしれないAで割らないように）次式(22)は課せられたひずみ増分に関する応力変化を決定する。

$$d\boldsymbol{\sigma} = \mathbf{D}^*_{ep} d\boldsymbol{\varepsilon} \tag{22}$$

$$\mathbf{D}^*_{ep} = \mathbf{D} - \mathbf{D}\left\{\frac{\partial Q}{\partial \boldsymbol{\sigma}}\right\}\left\{\frac{\partial F}{\partial \boldsymbol{\sigma}}\right\}^T \mathbf{D}\left[A + \left\{\frac{\partial F}{\partial \boldsymbol{\sigma}}\right\}^T \mathbf{D}\left\{\frac{\partial Q}{\partial \boldsymbol{\sigma}}\right\}\right]^{-1} \tag{23}$$

弾性マトリックス $\mathbf{D}^*_{ep}$ は増分解析において弾性マトリックス $\mathbf{D}_T$ に変わるものである。

［材料クラス］ 弾塑性マトリックス$\mathbf{D}^*_{ep}$はその後の計算に役に立つ材料挙動の記述を与える。材料特性はヤング率やポアソン比の様な基本的な特性でもって記述する。それらの材料特性を示し構成マトリックスに関する計算をするためのクラスは図6に示す。

材料

(to object)	
num υ $\mathbf{D}^*_{ep}$	
$\mathbf{E}$ t	
read	Init
write	

図6　材料クラス

形状関数

(to object)	
N **Nr** **Ns**	
determinant	
invJL	LocalToGlobal
Nrs	$\overline{\mathbf{B}}$-Matrix
writeDisp	

図7　形状関数クラス

Gauss

(to object)
weights
total
points
Init

図8　Gauss クラス

3.2.3　アイソパラメトリック概念

この名前は要素の形状と要素変位が同じ形状関数を用いて変換することを意味する。数値積分有限要素の計算の利点はいくつかの要素タイプに対して形状関数と導関数を定式化する事によって示す事が出来る。一般的問題に対する数値積分公式はGauss Legendre の求積法に基づいて与える。この概念は一般的な要素クラスの設計の中に利用される。

[形状関数クラス]　このクラスは形状関数($\mathbf{N}$)、導関数（ Nr 、と Ns ）、ヤコビアンの行列式を参照する。ひずみ－変位マトリックス($\overline{\mathbf{B}}$)の計算を導くメソードは線形演算子を持った逆Jacobian マトリックスの結果を計算する手続きと、Nrs マトリックスに導関数ベクトルを蓄える手続きを含む。このクラスは図7に示す。

[ガウスクラス]　このクラスは全ての積分点を参照し、同様にベクトルの形成に実際の点や重みを参照する。ガウスベクトルに蓄えられたデータを初期化する事だけに用いられるのはメソードだけである。ガウスデータは同じ種類の積分を用いた全ての要素に同じであるので、それらベクトルは要素クラスによって共有される。このクラスは図8に示す。

[要素クラス]　このクラスはガウス、材料、形状関数、節点、節点座標、接線剛性マトリックス($\mathbf{K}_T$)と応力($\mathbf{S}$)を参照する。バンド幅の計算、要素剛性を加えるための指標の組立、節点座標と変位の呼び出し等の一般的なメソードは要素の定式化とは関係なく実行出来る。剛性と応力の計算は形状関数とガウスのクラスからの幾つかのサブメソードを用いて実行する事が出来る。

要素

(to Object)	
num gauss $\mathbf{K}_T$	
material shapeFcns $\mathbf{S}$	
nodes numNodes	
read	Displacements
write	Stiffness
Bandwidth	Stress
Indices	
Coords	

図9　要素クラス

オブジェクトリスト

(to Object)	
theList	listSize
Init	Free
Add	Remove
Find	DoToAll
DestroyList	

図10　オブジェクトリストクラス

節点リスト

(to ObjectList)	
groupTitle	
read	AssignDOF
write	AssignDisp
writeDisp	

図11　節点リストクラス

材料リスト

(to ObjectList)	
groupTitle flag	
read	write

図12　材料リストクラス

変位境界条件リスト

(to ObjectList)	
groupTitle	
read	write
ReduceDOF	

図13　変位境界条件リストクラス

力学的境界条件リスト

(to ObjectList)	
groupTitle	
read	write
AddToLoadVector	

図14　力学的境界条件リストクラス

3.3　組立

節点、材料、要素の定義及び有限要素定式化の基本的な構成を含むクラスライブラリーを得ると、次にそれらを有益なプログラムに組み上げることが必要となる。効率よいオブジェクト指向プログラミングはオブジェクトの適切な管理と相互の関係が重要である。

3.3.1　オブジェクトリスト

線形リストを用いる事によるオブジェクトの組織化に関する標準的な方法を採用する。線形リスト管理はオブジェクトのリストを扱う一般的なクラスを作ることによって行う。このクラスは'オブジェクトリスト'と呼ぶ。オブジェクトリストクラスの例は節点インスタンスの組織化を用いる。このリストは節点を扱う幾つかの手続きを持ったオブジェクトリストである。これは全てのリストの一般的な方法を含む。材料インスタンスは材料リストと呼ばれているオブジェクトリストを用いて組織する。要素グループクラスは上位レベルの制御を行う要素グループリストクラスからのメッセージに対して中間的に作用する。

3.3.2　クラス階級組織

高度に洗練された有限要素解析プログラムは答えを計算するために複雑なデータと制御構造を用いる。オブジェクト指向プログラミングは抽象的なレベルに作用するクラス階級組織を作ることによって下位レベルのプログラム記述を簡単にする。基本のクラスは実際の数に作用する、ところが制御と組織的な関数を実行するクラスは形成される抽象的なデータ形式を処理する。

3.3.3　外部手続き

オブジェクト指向言語の最も重要な特徴の一つは現存するソフトを利用する能力である。ベクトルとマトリックスクラスを含む幾つかのクラスは基本的な代数マトリックスを操作するために導入される。

ベクトル

(to Object)	
n x	
read	Add
write	Scale
alloc	Dot
dealloc	Free

図15　ベクトルクラス

マトリックス

(to Object)	
m　n　x	
read	Add
write	Scale
alloc	Transpose
dealloc	Multiply
Free	MultVector

図16　マトリックスクラス

バンドマトリックス

(to Matrix)	
AddMatrix	Solve

図17　バンドマトリックスクラス

C＋＋におけるクラスマトリックスとベクトルのヘッダーファイルの例を示す。

[クラスマトリックス]

```
//-------------------------------------------------------------------
// File Matrix.hpp
//-------------------------------------------------------------------
#ifndef MATRIX_HPP                    //prevents multiple includes
#define MATRIX_HPP
#include <iostream.h>                 //headerfile for in-outputstream
#include <stdlib.h>
//-------------------------------------------------------------------
class Matrix{
 friend class Vector;
 private:
   double** m;                        //base pointer
   int r,c;                           //number of elements
 public:
 // constructor and destructor
   Matrix();                          //creates a matrix with 3 rows and 3 columuns
   Matrix(int rows,int columns);      //creates a matrix with m rows and n columuns
   Matrix(int rows,int columns,double initval);//initialization of a matrix with the value initval
   Matrix(Matrix& x);                 //initialization by a matrix x
   ~Matrix();                         //destructor
   int upper_row(){return(r-1);}      //upper bound of rows (coded inline)
   int upper_col(){return(c-1);}      //upper bound of columns (coded inline)
   double& operator()(int row,int col); //range checked element index runs from 1 to number of row/col
   Matrix& operator=(Matrix& x);      //addition operator
   Matrix& operator=(double  d);      //addition operator
   friend Matrix  operator+(Matrix&,Matrix&);//adition operator
   friend Matrix  operator-(Matrix&,Matrix&);
   friend Matrix  operator*(Matrix&,Matrix&);
   friend Vector  operator*(Matrix&,Vector );
   friend Vector  operator*(Vector&,Matrix );
   friend Matrix  operator*(Matrix&,double );
   friend Matrix  operator*(double ,Matrix&);
   friend Matrix  trans(Matrix&);     //transpose matrix
   friend Matrix  inv  (Matrix&);     //inverse matrix
   friend double  det  (Matrix&);     //determinat
   friend ostream& operator<<(ostream&,Matrix&);
   void print(char*msg="");
};
#endif
// end of file matrix.h
```

[クラスベクトル]

```
//-----------------------------------------------------------------
// File Vector.hpp
//-----------------------------------------------------------------
#ifndef VECTOR_HPP                      //prevents multiple includes
#define VECTOR_HPP
#include <iostream.h>                   //headerfile for in-outputstream
#include <stdlib.h>
//-----------------------------------------------------------------
class Vector{
friend class Matrix;
friend class Node;
friend class Nodetree;
private:
  double** v;                           //base pointer
  int r;                                //number of elements
public:
// constructor and destructor
  Vector();                             //creates a Vector with 3 rows
  Vector(int rows,int columns);         //creates a Vector with m rows
  Vector(int rows,int columns,double initval);//initialization of a Vector with the value initval
  Vector(Vector& x);                    //initialization by a Vector x
  ~Vector();                            //destructor
  int upper_row(){return(r-1);}         //upper bound of rows (coded inline)
  double& operator()(int row);          //range checked element index runs from 1 to number of row
  int getrow();                         //get the number of rows
  double& vget(int row);                //get the pointer of member v[i]
  double& vput(int row,double vp);      put *v on the pointer of member v[i]
  Vector& operator=(Vector& x);         //assignment operator
  Vector& operator=(double  d);         //assignment operator
  friend Vector  operator+(Vector&,Vector&);//adition operator
  friend Vector  operator-(Vector&,Vector&);
  friend double  operator*(Vector&,Vector&);
  friend Vector  operator*(Vector&,double );
  friend Vector  operator*(double ,Vector&);
```

4．終わりに

一般的な結論はオブジェクト指向プログラミングは非線形有限要素解析プログラムを改善するのに有益である。オブジェクト指向プログラムのプログラム化はより少ない時間で行え、より小さいプログラムになり、同等の手続き言語のそれよりもデータと手続きのよりよい管理を提供する事が出来る。

参考文献

[1] S.Maeda, S.Tanaka and H.Takeda,An object oriented-approach for non-liniar finite ele-ment anarisys *Non-liniar anarisys and design for shell and spatial structures.*, 105-112 (1993) .

[2] S.P. Scholz, Elements of an object-oriented fem++ program in c++. *Computer and Structure.*, **Vol.43,No. 3,** 517-529 (1992) .

[3] B.W.R. Forde, R.O. Foschi and SF. Stiemer, Object-oriented finite element analysis. *Computer and Structure.*, **Vol.34,No.3,** 355-374 (1990) .

[4] T. Zimmermann, Y. Dubois-Pelerin and P. Bomme, Object-oriented finite element pro-gramming: I.Govern principles. *Comp. Methods Appl. Mech. Eng.*, **98,** 291-303 (1992)

[5] T. Zimmermann, Y. Dubois-Pelerin and P. Bomme, Object-oriented finite element pro-gramming: II.A prototype program in Smalltalk. *Comp. Methods Appl. Mech. Eng.*, **98,** 361-397 (1992) .

[6] O.C. Zienkiewicz and R.L.Taylor, *The finite element method,4th ed.*, **Vol.2,** McGraw-Hill, (1991) .

キーワード

オブジェクト指向解法、非線形有限要素解析、連続体力学

Summary

AN OBJECT-ORIENTED APPROACH FOR NONLINEAR FINITE ELEMENT ANALYSIS

Shigenori Tanaka, Shigeyuki Maeda
Department of Civil Engineering, Hosei University
Hiroshi Takeda
Computer Center, Hosei University

This paper describes the problems with conventional finite element analysis software and the potential solutions offered by object-oriented programs. It introduces the basic concepts of object-oriented programming and of expandable applications. Nonlinear finite element fundamentals are explained using a new perspective leading to the implementation of an object-oriented numerical analysis program.

In addition, a few classes were introduced to handle the basic matrix algebra, including the Vector and Matrix classes. The classes Vector and Matrix provide a symbolic notation by way of operator overloading and polymorphic methods, which leads to clearly structured programs.

Key Words

object-oriented approach, nonlinear finite element analysis, continuum mechanics

3-7-2, Kajino-cho, Koganei-shi, Tokyo, 184, Japan.