有限元理论基础及Abaqus内部实现方式研究系列13：显式和隐式的区别

Theoretical Foundation of Finite Element Method and Research on Internal Implementation of Abaqus Series 13: The Difference Between Explicit and Implicit

SnowWave02 关注 Focus

更新于2021年3月3日 11:30 Updated on March 3, 2021, 11:30 AM 浏览：4326 Views: 4326 评论：5 Comments: 5 收藏：19 Favorited: 19

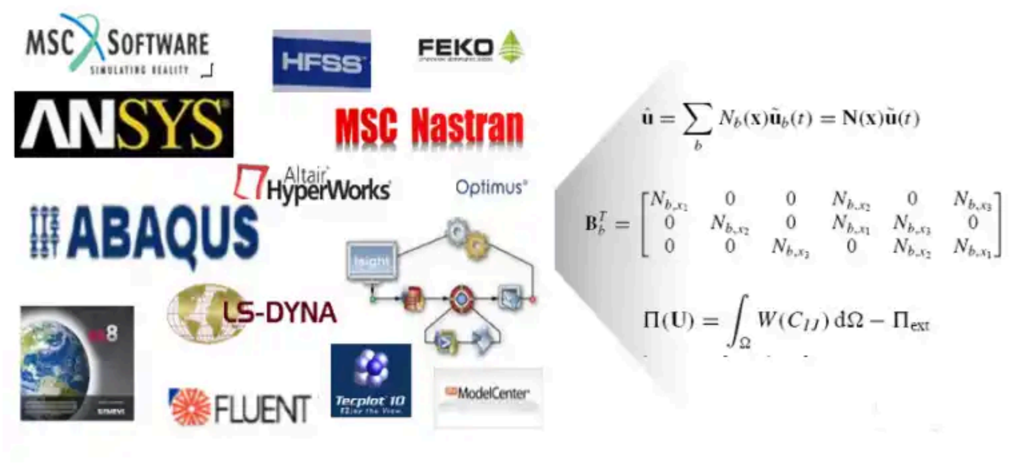
（原创，转载请注明出处） (Original, please indicate the source for reproduction)

==概述== ==Overview==

有限元理论基础及Abaqus内部实现方式研究系列13：显式和隐式的区别的图1

本系列文章研究成熟的有限元理论基础及在商用有限元软件的实现方式。有限元的理论发展了几十年已经相当成熟，商用有限元软件同样也是采用这些成熟的有限元理论，只是在实际应用过程中，商用CAE软件在传统的理论基础上会做相应的修正以解决工程中遇到的不同问题，且各家软件的修正方法都不一样，每个主流商用软件手册中都会注明各个单元的理论采用了哪种理论公式，但都只是提一下用什么方法修正，很多没有具体的实现公式。商用软件对外就是一个黑盒子，除了开发人员，使用人员只能在黑盒子外猜测内部实现方式。

This series of articles studies the mature finite element theoretical foundation and its implementation methods in commercial finite element software. The development of finite element theory has matured over decades, and commercial finite element software also adopts these mature finite element theories. However, in the actual application process, commercial CAE software will make corresponding corrections on the basis of traditional theories to solve different problems encountered in engineering, and the correction methods of each software are different. Each mainstream commercial software manual specifies which theoretical formula each element uses, but only mentions the correction method, and many do not provide specific implementation formulas. Commercial software is essentially a black box, and users can only guess its internal implementation methods from outside, except for developers.


MSC Software
HFSS
FEKO
ANSYS
MSC Nastran
Altair HyperWorks
Optimus
ABAQUS
LS-DYNA
FLUENT
Tecplot 10
ModelCenter
$\hat{\mathbf{u}} = \sum_b N_b(\mathbf{x})\tilde{\mathbf{u}}_b(t) = \mathbf{N}(\mathbf{x})\tilde{\mathbf{u}}(t)$
$\Pi(\mathbf{U}) = \int_\Omega W(C_{IJ})\,d\Omega - \Pi_{ext}$

一方面我们查阅各个主流商用软件的理论手册并通过进行大量的资料查阅猜测内部修正方法，另一方面我们自己编程实现结构有限元求解器，通过自研求解器和商软的结果比较来验证我们的猜测，如同管中窥豹一般来研究的修正方法，从而猜测商用有限元软件的内部计算方法。我们关注CAE中的结构有限元，所以主要选择了商用结构有限元软件中文档相对较完备的Abaqus来研究内部实现方式，同时对某些问题也会涉及其它的Nastran/Ansys等商软。为了理解方便有很多问题在数学上其实并不严谨，同时由于水平有限可能有许多的理论错误，欢迎交流讨论，也期待有更多的合作机会。

On one hand, we consult the theoretical manuals of various mainstream commercial software and guess the internal correction methods through extensive literature review. On the other hand, we program our own structural finite element solver and verify our guesses by comparing the results with those of commercial software. We study the correction methods like a glimpse through a tube, thus guessing the internal calculation methods of commercial finite element software. Since we focus on structural finite elements in CAE, we mainly choose Abaqus, which has relatively complete documentation among commercial structural finite element software, to study the internal implementation methods, and we will also involve other commercial software such as Nastran/Ansys for some issues. Many problems are not mathematically rigorous for the sake of understanding convenience, and due to our limited level, there may be many theoretical errors. We welcome discussions and look forward to more cooperation opportunities.

iSolver介绍： iSolver Introduction:

http://www.jishulink.com/college/video/c12884

==第13篇：显式和隐式的区别== ==Article 13: The Difference Between Explicit and Implicit==

CAE求解方法一般有两种，分别为显式（Explicit）和隐式（Implicit）。显式求解算法基于动力学方程，当前时刻的位移只与前一时刻的速度和位移相关，求解过程中无需迭代；而隐式求解基于虚功原理，一般需要进行迭代计算。

There are generally two types of CAE solution methods, namely explicit (Explicit) and implicit (Implicit). Explicit solution algorithms are based on dynamic equations, where the displacement at the current time only depends on the velocity and displacement at the previous time, and no iteration is required during the solution process; while implicit solutions are based on the principle of virtual work, and generally require iterative calculations.

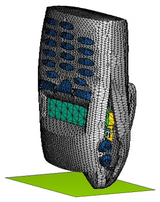
在Abaqus中，显式求解和隐式求解一般都会采用增量求解，即将分析步分割为若干个增量步，在当前增量步达到平衡时计算下一个增量步。

In Abaqus, both explicit and implicit solutions generally use incremental solution methods, which involve dividing the analysis step into several incremental steps and calculating the next incremental step when the balance is reached in the current incremental step.

1. 显式(Explicit) 1. Explicit

在显式求解过程中，每个增量步内不需要进行迭代求解，且无需形成切线刚度矩阵，故每个增量步内计算量相对于隐式求解方法消耗较小，一般与单元规模成正比。但增量步长也不能过大，一般不超过模型最小自由振荡周期的1/10，否则容易导致计算结果发散。

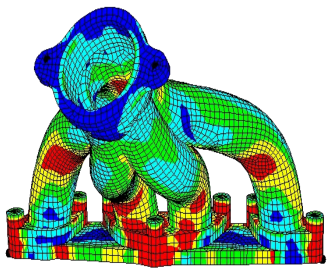
In explicit solution processes, iterative solving is not required within each increment step, and there is no need to form the tangent stiffness matrix. Therefore, the computational effort within each increment step is relatively smaller compared to the implicit solution method, and it generally varies proportionally with the element size. However, the increment step length should not be too large, generally not exceeding 1/10 of the minimum natural oscillation period of the model, otherwise it may easily lead to divergence of the calculation results.



2. 隐式(Implicit) 2. Implicit

在隐式求解过程中，每个增量步都需要进行平衡迭代，需要形成切线刚度矩阵，计算量相对较大，一般与单元规模和迭代收敛速度相关。隐式求解的收敛速度和稳定性根据选择迭代方法的不同而不同。因此，需要针对模型特性选择合适的增量步长，保证计算结果的收敛。

In implicit solution processes, balance iteration is required within each increment step, and the tangent stiffness matrix needs to be formed. The computational effort is relatively larger, generally related to the element size and the convergence speed of iteration. The convergence speed and stability of implicit solution vary with the choice of iterative method. Therefore, it is necessary to select an appropriate increment step length according to the model characteristics to ensure the convergence of the calculation results.



有限元理论基础及Abaqus内部实现方式研究系列13：显式和隐式的区别的图8

综上，无论是显式求解还是隐式求解，都需要根据模型和求解问题合理设置分析步的增量步长和求解方法，保证分析的精度和质量。虽然这两种求解方法已经是有限元的基本动力学求解方法，但由于有限元本身的复杂性，往往很多人都难以理解两者的区别和显式为何发射，本文将以一个简单的算例配合代码实现来直观的解释一下隐式和显式

的区别。

In summary, whether explicit or implicit solving, it is necessary to reasonably set the increment step size and solving method of the analysis step according to the model and the solving problem to ensure the accuracy and quality of the analysis. Although both of these solving methods are basic dynamic solving methods of finite element analysis, due to the complexity of finite element analysis itself, many people often find it difficult to understand the differences between the two and why explicit solving is used. This article will use a simple example and code implementation to intuitively explain the differences between implicit and explicit solving.

有限元理论基础及Abaqus内部实现方式研究系列13：显式和隐式的区别的图9

1.1 前向欧拉（Forward Euler）和后向欧拉（Backward Euler）

1.1 Forward Euler and Backward Euler

前向欧拉和后向欧拉分别是显式和隐式的一个典型计算方法，本文将引用这两个方法来尽量直观地展现显式求解和隐式求解的区别。

Forward Euler and Backward Euler are typical computational methods for explicit and implicit solving, respectively. This article will refer to these methods to try to intuitively demonstrate the differences between explicit and implicit solving.

前向欧拉： Forward Euler:

fn+1 (x) = fn (x) + h*f'n (x)

fn+1(x) = fn(x) + h*f'n(x)

后向欧拉： Backward Euler:

fn+1 (x) = fn (x) + h*f'n+1 (x)

fn+1(x) = fn(x) + h*f'n+1(x)

有限元理论基础及Abaqus内部实现方式研究系列13：显式和隐式的区别的图10

1.2 算例 Example 1.2

现在以一个微分方程算例为例： Take an example of a differential equation:

y'(x) = -y+x+1 y'(x) = -y + x + 1

且有y(0) = 1。 And there is y(0) = 1.

1、前向欧拉法 1. Forward Euler method

使用前向欧拉法，可得： Using the forward Euler method, we have:

yn+1 = yn + h*(-yn +xn +1)

yn+1 = yn + h*(-yn + xn + 1)

显然，这个式子不需要做任何额外运算就能从yn出yn+1，因此每步迭代计算量小。但h在一个最大限制，具体见下面的推导过程：

Clearly, this formula does not require any additional operations to obtain yn+1 from yn, thus the computational effort per iteration is small. However, h has a maximum limit, which is detailed in the following derivation process:

yn+1 = (1-h)yn + h*(xn +1)

yn+1 = (1-h)yn + h*(xn + 1)

可得 Obtainable

yn+1 = (1-h)2 yn-1 + h*(xn +1) + h*(xn-1 +1)

yn+1 = (1-h)^2 yn-1 + h*(xn +1) + h*(xn-1 +1)

则得 Then we have

yn+1 = (1-h)n +O(s2)

yn+1 = (1-h)^n + O(h^2)

当h<2时，y收敛，但h>=2时，y将发散。 When h < 2, y converges, but when h >= 2, y will diverge.

2、后向欧拉法 2. Backward Euler Method

yn+1 = yn + h*(-yn+1 +xn+1 +1)

yn+1 = yn + h*(-yn+1 + xn+1 + 1)

这个式子不能直接得出yn+1, 必须做进一步计算得到

This equation cannot be directly used to obtain yn+1; further calculation is required.

yn+1 = (yn + h*(xn+1 +1)) / (1+h)

yn+1 = (yn + h*(xn+1 + 1)) / (1 + h)

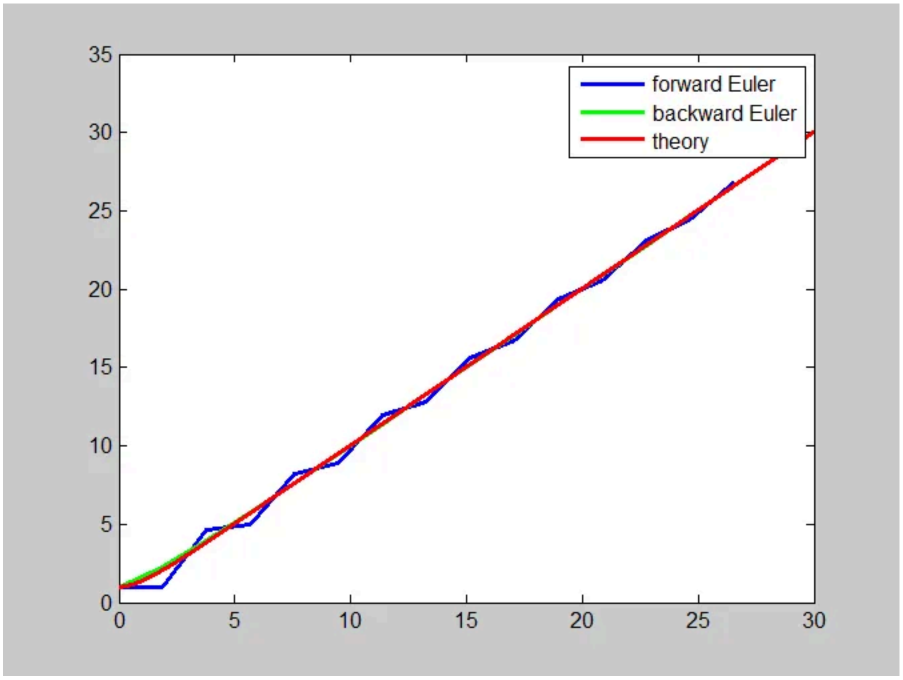
当h>0时，y无条件收敛。 When h > 0, y converges unconditionally.

3、结果比较 3. Result Comparison

假设n=30 Assume n = 30

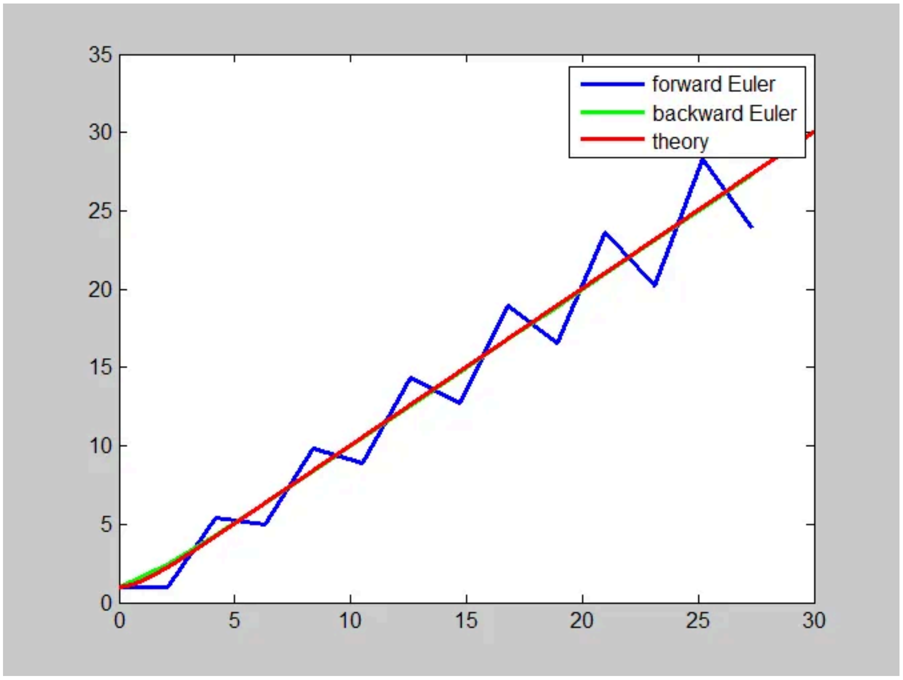
（1）当h = 1.9时，计算结果如下图所示,显式和隐式都和理论接近：

(1) When h = 1.9, the calculation results are shown in the figure below; both explicit and implicit methods are close to the theory:



（2）当h = 2.1时，计算结果如下图所示，显式求解发散了：

(2) When h = 2.1, the calculation results are shown in the figure below, and the explicit solution diverges:



==总结== ==Summary==

本文简单介绍了显式和隐式的区别，本质在于采用不同的物理学平衡方程，因此在不同的物理学问题也有不同的表现。文中给出一个数学算例，并分别利用前向欧拉和后向欧拉求解，以求直观表现显式和隐式在求解过程中的差

异，以及增量步长对求解结果的影响。

This article briefly introduces the difference between explicit and implicit methods, which essentially lie in the use of different physics equilibrium equations, thus showing different behaviors in different physics problems. A mathematical example is given, and the forward Euler and backward Euler methods are used to solve it respectively, in order to intuitively demonstrate the differences between explicit and implicit methods in the solution process, as well as the impact of the incremental step size on the solution results.

如果有任何其它疑问或者项目合作意向，也欢迎联系我们：

If you have any other questions or intentions for project cooperation, feel free to contact us:

snowwave02 From www.jishulink.com

email: snowwave02@qq.com

以往的系列文章： Previous series articles:

有限元理论基础及Abaqus内部实现方式研究系列13：显式和隐式的区别的图15

第一篇：**S4壳单元刚度矩阵研究。**介绍Abaqus的S4刚度矩阵在普通厚壳理论上的修正。

First article: Research on the Stiffness Matrix of S4 Shell Element. Introduces the correction of Abaqus' S4 stiffness matrix in the theory of ordinary thick shell.

http://www.jishulink.com/content/post/338859

第二篇：**S4壳单元质量矩阵研究。**介绍Abaqus的S4和Nastran的Quad4单元的质量矩阵。

Second article: Research on the Mass Matrix of S4 Shell Element. Introduces the mass matrices of Abaqus' S4 and Nastran's Quad4 elements.

http://www.jishulink.com/content/post/343905

第三篇：**S4壳单元的剪切自锁和沙漏控制。**介绍Abaqus的S4单元如何来消除剪切自锁以及S4R如何来抑制沙漏的。

Third article: Shear locking and hourglass control of S4 shell elements. Introduces how Abaqus S4 elements eliminate shear locking and how S4R suppresses hourglassing.

http://www.jishulink.com/content/post/350865

第四篇：**非线性问题的求解。**介绍Abaqus在非线性分析中采用的数值计算的求解方法。

Fourth article: Solution of nonlinear problems. This article introduces the numerical computation methods adopted by Abaqus in nonlinear analysis.

http://www.jishulink.com/content/post/360565

第五篇：**单元正确性验证。**介绍有限元单元正确性的验证方法，通过多个实例比较自研结构求解器程序iSolver与Abaqus的分析结果，从而说明整个正确性验证的过程和iSolver结果的正确性。

Fifth article: Element correctness verification. Introduces the verification methods for finite element element correctness, compares the analysis results of the self-developed structural solver program iSolver with Abaqus through multiple examples, thereby illustrating the entire correctness verification process and the correctness of the iSolver results.

https://www.jishulink.com/content/post/373743

第六篇：**General梁单元的刚度矩阵。**介绍梁单元的基础理论和Abaqus中General梁单元的刚度矩阵的修正方式，采用这些修正方式可以得到和Abaqus梁单元完全一致的刚度矩阵。

Sixth article: Stiffness matrix of General beam element. Introduces the basic theory of beam elements and the correction methods of the General beam element stiffness matrix in Abaqus. By using these correction methods, it is possible to obtain a stiffness matrix that is completely consistent with the Abaqus beam element.

https://www.jishulink.com/content/post/403932

第七篇：**C3D8六面体单元的刚度矩阵。**介绍六面体单元的基础理论和Abaqus中C3D8R六面体单元的刚度矩阵的修正方式，采用这些修正方式可以得到和Abaqus六面体单元完全一致的刚度矩阵。

Seventh article: Stiffness matrix of C3D8 hexahedral element. Introduces the basic theory of hexahedral elements and the correction methods of the C3D8R hexahedral element stiffness matrix in Abaqus. By using these correction methods, it is possible to obtain a stiffness matrix that is completely consistent with the Abaqus hexahedral element.

https://www.jishulink.com/content/post/430177

第八篇：**UMAT用户子程序开发步骤。**介绍基于Fortran和Matlab两种方式的Abaqus的UMAT的开发步骤，对比发现开发步骤基本相同，同时采用Matlab更加高效和灵活。

Eighth article: Steps for UMAT user subroutine development. Introduces the development steps of Abaqus UMAT based on both Fortran and Matlab, and finds that the development steps are basically the same. At the same time, Matlab is found to be more efficient and flexible.

https://www.jishulink.com/content/post/432848

第九篇：**编写线性UMAT Step By Step。**介绍了线性UMAT的接口功能和关键接口变量的含义，并通过简单立方体静力分析的算例详细说明了基于Matlab线性UMAT的开发步骤。

Chapter 9: Writing Linear UMAT Step by Step. Introduces the interface functions of linear UMAT and the meanings of key interface variables, and illustrates the development steps of linear UMAT based on Matlab through a simple cube static analysis example.

http://www.jishulink.com/content/post/440874

第十篇：**耦合约束（Coupling constraints）的研究。**介绍了耦合约束的定义和用途，具体阐述了Abaqus中运动耦合约束和分布耦合约束的原理。

Chapter 10: Research on Coupling Constraints. Introduces the definition and application of coupling constraints, and specifically elaborates on the principles of motion coupling constraints and distributed coupling constraints in Abaqus.

http://www.jishulink.com/content/post/531029

第十一篇：**自主CAE开发实战经验第一阶段总结。**结合自研有限元求解器iSolver第一阶段开发的实战经验，从整体角度上介绍自主CAE的开发难度、时间预估、框架设计、编程语言选择、测试、未来发展方向等。

Chapter 11: Summary of the First Phase of Independent CAE Development Practice. Based on the practical experience of the first phase development of the independently developed finite element solver iSolver, this article introduces the difficulties, time estimation, framework design, programming language selection, testing, and future development directions of independent CAE development from an overall perspective.

http://www.jishulink.com/content/post/532475

第十二篇：**几何梁单元的刚度矩阵。**研究了Abaqus中几何梁的B31单元的刚度矩阵的求解方式，以L梁为例，介绍General梁用到的面积、惯性矩、扭转常数等参数在几何梁中是如何通过几何形状求得的，根据这些参数，可以得到和Abaqus完全一致的刚度矩阵，从而对只有几何梁组成的任意模型一般都能得到Abaqus完全一致的分析结果，并用一个简单的算例验证了该想法。

Twelfth article: Stiffness Matrix of Geometric Beam Element. This article studies the method of solving the stiffness matrix of the B31 element of geometric beam in Abaqus, taking the L beam as an example, and introduces how the parameters such as area, moment of inertia, and torsion constant used in General beam are obtained through geometric shape in geometric beam. Based on these parameters, a stiffness matrix consistent with Abaqus can be obtained, so that for any model composed only of geometric beams, Abaqus can generally obtain consistent analysis results. This idea is verified by a simple example.

http://www.jishulink.com/content/post/534362