Iterative Solvers in the Finite Element Solution of Transient Heat Conduction

Mile R. Vujičić
PhD student

Steve G.R. Brown
Senior Lecturer

Materials Research Centre
School of Engineering
University of Wales Swansea
Swansea SA2 8PP,
Wales, UK

In this paper three-dimensional transient heat conduction in homogeneous materials using different iterative solvers based on the conjugate gradient method are studied. The methods used are preconditioned conjugate gradient, least square conjugate gradient, conjugate gradient squared, preconditioned conjugate squared, bi-conjugate gradient, preconditioned bi-conjugate gradient, bi-conjugate gradient stabilized and preconditioned bi-conjugate stabilized method as well as a Gaussian elimination method with incomplete Cholesky factorisation as a comparison. Time dependence is solved using both the finite difference and the finite element scheme. The finite difference scheme includes backward and forward Euler method and θ–method (Crank-Nicholson). The finite element scheme includes the weighted residual method and the least squares method. For the analysis, in the aim to analyse dependence between numerical scheme and obtained temperatures, we use a brick that is divided into different number 8-noded or 20-noded hexahedral elements with Dirichlet boundary conditions.

Keywords: *transient heat conduction, time stepping, iterative solvers, residual, inner iterations*

1. INTRODUCTION

The analysis of heat conduction (steady and transient) is well established, and there are several different approaches or techniques that can be used (finite difference, boundary element, finite element and finite volume method).

The finite element method has been developed in parallel with the introduction of powerful and fast computers during the last forty years. It started in the field of structural analysis for stress calculations mainly between 1940 and 1960. Later, the concept based on dividing a structure into small substructures of various shapes and re-assembled after each substructure had been analyzed become a very useful tool for heat transfer calculations, solving mechanic fluids problems, electro magnetic calculation etc.

Discretization of the relevant partial differential equation gives a set of linear or nonlinear algebraic equations.

$$\mathbf{Ax} = \mathbf{b} \qquad (1.1)$$

Generally speaking there are several families of methods than can be used for solving systems of algebraic equations, such as direct (Gauss elimination), multigrid and iterative methods.

As problem sizes grow, the storage requirement becomes a burden, even on a modern computer. For this reason, iterative methods, which require less memory, have been developed as alternative solution strategies.

2. TRANSIENT HEAT CONDUCTION AND FINITE ELEMENT METHOD

The principle of conservation of heat energy over a fixed volume, V, of a medium that is bounded by a closed surface S, can be used to derive the equation that governs heat conduction. Usually, heat transfer is treated in a domain Ω bounded by a closed surface Γ. The classical statement of this can be written as:

$$\frac{\partial T}{\partial t} = \frac{1}{\rho c_p}\left[\operatorname{div}(k \operatorname{grad} T) + Q\right] \quad \text{in } \Omega \qquad (2.1)$$

$$T = f(x) \quad \text{for } x \text{ on } \Gamma_1 \qquad (2.2)$$

$$q(x) = -(k \operatorname{grad} T)\cdot n = -k\frac{\partial T}{\partial n} \quad \text{for } x \text{ on } \Gamma_2 \qquad (2.3)$$

where k is thermal conductivity and Q is a source or sink of heat energy. This equation is well known as Fourier's law of conduction.

Applying the finite element technique in which the Galerkin method is used for weighting functions we obtain following matrix equation system;

$$\mathbf{M}\dot{\mathbf{T}} + \mathbf{K}\,\mathbf{T} = \mathbf{f} \ , \qquad (2.4)$$

Received: September 2004, accepted: November 2004
Correspondence to: Mile Vujičić,
Materials Research Centre, School of Engineering,
University of Wales Swansea, Swansea
Singleton Park SA2 8PP, Wales, UK
E-mail:293097@swansea.ac.uk

where $\mathbf{M}$ is the capacitance (mass) matrix, $\mathbf{K}$ is the conductance (coefficient stiffness) matrix and $\dot{\mathbf{T}}$ is the temperature differentiated with respect to time.

2.1. Finite difference time stepping

There are several finite difference time stepping schemes, here the explicit (forward Euler) method , the implicit (backward Euler) method and Crank-Nicolson form of the θ -method are used.

Explicit (forward Euler) method

This is a first-order method and can be derived from a Taylor's series expression. After applying this method on equation (2.4) we obtain,

$$\mathbf{M}T^{n+1} = (\mathbf{M} - \Delta t\,\mathbf{K})T^n + \Delta t\,\mathbf{f}^n \quad . \tag{2.5}$$

Implicit (backward Euler) method

The implicit method requires much more storage than the explicit method but it provides the potential advantage of using a large time step, which may result in a more efficient procedure. The matrix equation system (2.4) for this method becomes:

$$(\mathbf{M} + \Delta t\,\mathbf{K})T^{n+1} = \mathbf{M}T^n + \Delta t\,\mathbf{f}^n \tag{2.6}$$

θ (Crank-Nicolson) method

The Crank-Nicolson method is an implicit method with second-order accuracy and it is a special case of the θ method. The matrix equation system (2.4) for this method becomes:

$$(\mathbf{M} + \theta\Delta t\,\mathbf{K})T^{n+1} = (\mathbf{M} - (1-\theta)\Delta t\,\mathbf{K})T^n + \Delta t\,\mathbf{f}^n \tag{2.7}$$

A different value of θ defines different methods. For $\theta = 0$, equation (2.7) is identical to equation (2.5). When $\theta = 1$ equation (2.7) is identical to equation (2.6). The Crank-Nicolson method is defined with $\theta = 0.5$.

2.2 Finite element time stepping

Finite element time stepping uses the same technique for time discretization as well as being used for the discretization of a domain [6]. In the text written below two different techniques are described:

a) the weighted residual method; and
b) the least squared method.

Weighted residual method

After time discretization applied on equation (2.4) and using the weighted residual method we will have:

$$\left(\frac{\mathbf{M}}{\Delta t} + \mathbf{K}\theta\right)T^{n+1} = \left[\frac{\mathbf{M}}{\Delta t} - \mathbf{K}(1-\theta)\right]T^n + \hat{\mathbf{f}} \quad , \tag{2.8}$$

where are

$$\theta = \frac{\int_0^1 W_j\,\xi\,\mathrm{d}\xi}{\int_0^1 W_j\,\mathrm{d}\xi} \quad \text{and} \quad \hat{\mathbf{f}} = \frac{\int_0^1 W_j\,\mathbf{f}\,\mathrm{d}\xi}{\int_0^1 W_j\,\mathrm{d}\xi} \; .$$

The difference between the finite element and finite difference approach is that θ in the finite difference algorithm is completely between 0 and 1, while in the finite element algorithm it depends on the weighting function W_j. This means that all methods mentioned earlier are just special cases of the weighted residual method.

Least squared method

The main idea of this method is that error with respect to T^{n+1} can be minimized to produce a least squares algorithms.

$$\left[\frac{\mathbf{M}^T\mathbf{M}}{\Delta t} + \frac{\left(\mathbf{K}^T\mathbf{M} + \mathbf{M}^T\mathbf{K}\right)}{2} + \mathbf{K}^T\mathbf{K}\frac{\Delta t}{3}\right]T^{n+1} =$$
$$\left[\frac{\mathbf{M}^T\mathbf{M}}{\Delta t} + \frac{\left(\mathbf{K}^T\mathbf{M} - \mathbf{M}^T\mathbf{K}\right)}{2} - \mathbf{K}^T\mathbf{K}\frac{\Delta t}{6}\right]T^n - \mathbf{K}^T\int_0^1 f\xi\frac{\mathrm{d}\xi}{\Delta t} \tag{2.9}$$

3. ITERATIVE SOLVERS

3.1. Conjugate gradient method (CG)

This is an old and well-known non-stationary method discovered independently by Hestenes and Stiefel in the early 1950's [1]. Also, it is the most popular iterative method for solving large, symmetric and positive definite systems of linear equations. In fact, this method can be used for solving these types of equations only. Historically speaking, the conjugate gradient method came after the method of steepest descent and the method of conjugate directions. De facto, the CG method is the conjugate direction method where the search directions are constructed by conjugation of the residuals. The conjugate gradient strategy is presented in the following pseudo code.

```
1. initial guess $\mathbf{x}^{(0)}$ ( for example $\mathbf{x}^{(0)} = 0$ )
2. compute $\mathbf{r}^{(0)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(0)}$
3. $\mathbf{p}^{(0)} = \mathbf{r}^{(0)}$
do for $i$=1, 2, …, iterations limit
    $\mathbf{q}^{(i)} = \mathbf{A}\mathbf{p}^{(i-1)}$ ,
    $\alpha^{(i)} = \frac{\left(\mathbf{r}^{(i-1)},\mathbf{r}^{(i-1)}\right)}{\left(\mathbf{p}^{(i-1)},\mathbf{q}^{(i)}\right)}$ ,
    $\mathbf{x}^{(i)} = \mathbf{x}^{(i-1)} + \alpha^{(i)}\mathbf{p}^{(i-1)}$ ,
    $\mathbf{r}^{(i)} = \mathbf{r}^{(i-1)} - \alpha^{(i)}\mathbf{q}^{(i)}$ ,
    $\beta^{(i)} = \frac{\left(\mathbf{r}^{(i)},\mathbf{r}^{(i)}\right)}{\left(\mathbf{r}^{(i-1)},\mathbf{r}^{(i-1)}\right)}$ ,
    $\mathbf{p}^{(i)} = \mathbf{r}^{(i)} + \beta^{(i)}\mathbf{p}^{(i-1)}$ ,
check convergence, continue if necessary
end
```

3.2. Preconditioned Conjugate gradient method (PCG)

Increasing of the number of elements obtained after a discretization causes the standard CG method to converge more slowly. This can be overcome by replacing the problem whose solution we seek by another one that has the same solution but which converges much faster. There are several different techniques for this type of 'preconditioning' three of which are presented here namely, splitting, left preconditioning and right preconditioning. The splitting technique includes Jacobi, Gauss-Seidel, and Successive Over Relaxation etc. The pseudo code for the PCG method, incorporating the Jacobi method for preconditioning, is presented below.

```
1. initial guess $\mathbf{x}^{(0)}$ ( for example $\mathbf{x}^{(0)} = 0$ )
2. compute $\mathbf{r}^{(0)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(0)}$
3. $\mathbf{p}^{(0)} = \mathbf{d}^{(0)} = \mathbf{M}^{-1}\mathbf{r}^{(0)}$
do for $i$=1, 2, …, iterations limit
    $\mathbf{q}^{(i)} = \mathbf{A}\,\mathbf{p}^{(i-1)}$,
    $\alpha^{(i)} = \frac{\left(\mathbf{d}^{(i-1)},\mathbf{r}^{(i-1)}\right)}{\left(\mathbf{p}^{(i-1)},\mathbf{q}^{(i)}\right)}$,
    $\mathbf{x}^{(i)} = \mathbf{x}^{(i-1)} + \alpha^{(i)}\,\mathbf{p}^{(i-1)}$,
    $\mathbf{r}^{(i)} = \mathbf{r}^{(i-1)} - \alpha^{(i)}\mathbf{q}^{(i)}$,
    $\mathbf{d}^{(i)} = \mathbf{M}^{-1}\mathbf{r}^{(i)}$,
    $\beta^{(i)} = \frac{\left(\mathbf{q}^{(i)},\mathbf{r}^{(i)}\right)}{\left(\mathbf{q}^{(i-1)},\mathbf{r}^{(i-1)}\right)}$,
    $\mathbf{p}^{(i)} = \mathbf{d}^{(i)} + \beta^{(i)}\mathbf{p}^{(i-1)}$,
check convergence, continue if necessary
end
```

The matrix $\mathbf{M}$ is the preconditioning matrix.

3.3. Least squared conjugate gradient method (LSCG)

In the case that matrix $\mathbf{A}$ is non symmetric we cannot apply the CG method. Multiplying both sides of the equation (1.1) by $\mathbf{A}^T$ yields:

$$\mathbf{A}^T\mathbf{r} = \mathbf{A}^T - \mathbf{A}^T\mathbf{A}\,\mathbf{x} \tag{3.1}$$

The new matrix $\mathbf{A}^T\mathbf{A}$ is always symmetric. The pseudo code for LSCG is:

```
1. initial guess $\mathbf{x}^{(0)}$ ( for example $\mathbf{x}^{(0)} = 0$ )
2. compute $\mathbf{r}^{(0)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(0)}$
3. $\mathbf{p}^{(0)} = \mathbf{s}^{(0)} = \mathbf{A}^T\mathbf{r}^{(0)}$
do for $i$=1, 2, …, iterations limit
    $\mathbf{q}^{(i)} = \mathbf{A}\,\mathbf{p}^{(i-1)}$,
    $\alpha^{(i)} = \frac{\left(\mathbf{s}^{(i-1)},\mathbf{s}^{(i-1)}\right)}{\left(\mathbf{q}^{(i)},\mathbf{q}^{(i)}\right)}$,
    $\mathbf{x}^{(i)} = \mathbf{x}^{(i-1)} + \alpha^{(i)}\mathbf{p}^{(i-1)}$,
    $\mathbf{r}^{(i)} = \mathbf{r}^{(i-1)} - \alpha^{(i)}\mathbf{q}^{(i)}$,
    $\mathbf{s}^{(i)} = \mathbf{A}^T\mathbf{r}^{(i)}$,
    $\beta^{(i)} = \frac{\left(\mathbf{s}^{(i)},\mathbf{s}^{(i)}\right)}{\left(\mathbf{s}^{(i-1)},\mathbf{s}^{(i-1)}\right)}$,
    $\mathbf{p}^{(i)} = \mathbf{s}^{(i)} + \beta^{(i)}\mathbf{p}^{(i-1)}$,
check convergence, continue if necessary
end
```

3.4. Bi-conjugate gradient method (BCG)

Instead of solving the system of equations (1.1), in this method we are going to solve the following system of equations,:

$$\begin{aligned}\mathbf{A}\,\mathbf{x} &= \mathbf{b}\,,\\ \mathbf{A}^T\tilde{\mathbf{x}} &= \mathbf{b}\,,\end{aligned} \tag{3.2}$$

where two sequences of residuals are updated,:

$$\begin{aligned}\mathbf{r}^{(i)} &= \mathbf{r}^{(i-1)} - \alpha^{(i)}\mathbf{A}\,\mathbf{p}^{(i)}\\ \tilde{\mathbf{r}}^{(i)} &= \tilde{\mathbf{r}}^{(i-1)} - \alpha^{(i)}\mathbf{A}^T\tilde{\mathbf{p}}^{(i)}\end{aligned} \tag{3.3}$$

and two sequences of search directions:

$$\begin{aligned}\mathbf{p}^{(i)} &= \mathbf{r}^{(i-1)} + \beta^{(i-1)}\mathbf{p}^{(i-1)}\\ \tilde{\mathbf{p}}^{(i)} &= \tilde{\mathbf{r}}^{(i-1)} + \beta^{(i-1)}\tilde{\mathbf{p}}^{(i-1)}\end{aligned} \tag{3.4}$$

The pseudo code is given in the following text:

```
1. initial guess $\mathbf{x}^{(0)}$ ( for example $\mathbf{x}^{(0)} = 0$ )
2. compute $\mathbf{r}^{(0)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(0)}$
3. choose $\tilde{\mathbf{r}}^{(0)}$ (for example $\tilde{\mathbf{r}}^{(0)} = \mathbf{r}^{(0)}$)
4. $\rho^{(0)} = 1$.
do for $i$=1, 2, …, iterations limit
    $\rho^{(i)} = \left(\tilde{\mathbf{r}}^{(i-1)},\mathbf{r}^{(i-1)}\right)$,
    $\beta^{(i)} = \frac{\rho^{(i)}}{\rho^{(i-1)}}$,
    $\rho^{(i-1)} = \rho^{(i)}$,
    $\mathbf{p}^{(i)} = \mathbf{r}^{(i)} + \beta^{(i)}\mathbf{p}^{(i-1)}$,
    $\tilde{\mathbf{p}}^{(i)} = \tilde{\mathbf{r}}^{(i)} + \beta^{(i)}\tilde{\mathbf{p}}^{(i-1)}$,
    $\sigma^{(i)} = \left(\tilde{\mathbf{p}}^{(i)},\mathbf{A}\,\mathbf{p}^{(i)}\right)$,
    $\alpha^{(i)} = \frac{\rho^{(i)}}{\sigma^{(i)}}$,
    $\mathbf{r}^{(i)} = \mathbf{r}^{(i-1)} - \alpha^{(i)}\mathbf{A}\,\mathbf{p}^{(i)}$,
    $\tilde{\mathbf{r}}^{(i)} = \tilde{\mathbf{r}}^{(i-1)} - \alpha^{(i)}\mathbf{A}^T\tilde{\mathbf{p}}^{(i)}$,
    $\mathbf{x}^{(i)} = \mathbf{x}^{(i-1)} + \alpha^{(i)}\mathbf{p}^{(i)}$,
check convergence, continue if necessary
end
```

3.5. Preconditioned Bi-conjugate gradient method (PBCG)

The difference between this method and the BCG method is two more steps that involve preconditioning.

```
1. initial guess $\mathbf{x}^{(0)}$ ( for example $\mathbf{x}^{(0)} = 0$ )
2. compute $\mathbf{r}^{(0)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(0)}$
3. choose $\tilde{\mathbf{r}}^{(0)}$ (for example $\tilde{\mathbf{r}}^{(0)} = \mathbf{r}^{(0)}$)
4. $\rho^{(0)} = 1$.
do for $i$=1, 2, …, iterations limit
solve $\mathbf{M}\mathbf{z}^{(i-1)} = \mathbf{r}^{(i-1)}$
solve $\mathbf{M}^T\tilde{\mathbf{z}}^{(i-1)} = \tilde{\mathbf{r}}^{(i-1)}$
    $\rho^{(i)} = \left(\mathbf{z}^{(i-1)}, \tilde{\mathbf{r}}^{(i-1)}\right)$,
    $\beta^{(i)} = \frac{\rho^{(i)}}{\rho^{(i-1)}}$,
    $\rho^{(i-1)} = \rho^{(i)}$,
    $\mathbf{p}^{(i)} = \mathbf{z}^{(i)} + \beta^{(i)}\mathbf{p}^{(i-1)}$,
    $\tilde{\mathbf{p}}^{(i)} = \tilde{\mathbf{z}}^{(i)} + \beta^{(i)}\tilde{\mathbf{p}}^{(i-1)}$,
    $\sigma^{(i)} = \left(\tilde{\mathbf{p}}^{(i)}, \mathbf{A}\tilde{\mathbf{p}}^{(i)}\right)$,
    $\alpha^{(i)} = \frac{\rho^{(i)}}{\sigma^{(i)}}$,
    $\mathbf{r}^{(i)} = \mathbf{r}^{(i-1)} - \alpha^{(i)}\mathbf{A}\mathbf{p}^{(i)}$,
    $\tilde{\mathbf{r}}^{(i)} = \tilde{\mathbf{r}}^{(i-1)} - \alpha^{(i)}\mathbf{A}^T\tilde{\mathbf{p}}^{(i)}$,
    $\mathbf{x}^{(i)} = \mathbf{x}^{(i-1)} + \alpha^{(i)}\mathbf{p}^{(i)}$,
check convergence, continue if necessary
end
```

3.6. Conjugate gradient squared method (CGS)

Developed by Sonneveld [3] this method has been derived from BCG method, where using the transpose of $\mathbf{A}$ is avoided and thereby faster convergence for roughly the same computation cost is obtained.
The pseudo code is given in the following text:

```
1. initial guess $\mathbf{x}^{(0)}$ ( for example $\mathbf{x}^{(0)} = 0$ )
2. compute $\mathbf{r}^{(0)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(0)}$
3. choose $\tilde{\mathbf{r}}^{(0)}$ (for example $\tilde{\mathbf{r}}^{(0)} = \mathbf{r}^{(0)}$)
4. $\mathbf{q}^{(0)} = \mathbf{p}^{(0)} = 0$.
5. $\rho^{(0)} = 1$.
do for $i$=1, 2, …,i terations limit
    $\rho^{(i)} = \left(\tilde{\mathbf{r}}^{(i-1)}, \mathbf{r}^{(i-1)}\right)$,
    $\beta^{(i)} = \frac{\rho^{(i)}}{\rho^{(i-1)}}$,
    $\rho^{(i-1)} = \rho^{(i)}$,
    $\mathbf{z}^{(i)} = \mathbf{r}^{(i-1)} + \beta^{(i)}\mathbf{q}^{(i-1)}$,
    $\mathbf{p}^{(i)} = \mathbf{z}^{(i)} + \beta^{(i)}\left(\mathbf{q}^{(i-1)} + \beta^{(i)}\mathbf{p}^{(i-1)}\right)$,
    $\mathbf{v}^{(i)} = \mathbf{A}\mathbf{p}^{(i)}$,
    $\sigma^{(i)} = \left(\tilde{\mathbf{r}}^{(i)}, \mathbf{v}^{(i)}\right)$,
    $\alpha^{(i)} = \frac{\rho^{(i)}}{\sigma^{(i)}}$,
    $\mathbf{q}^{(i)} = \mathbf{z}^{(i)} - \alpha^{(i)}\mathbf{A}\mathbf{v}^{(i)}$,
    $\mathbf{r}^{(i)} = \mathbf{r}^{(i-1)} - \alpha^{(i)}\mathbf{A}\left(\mathbf{z}^{(i)} + \mathbf{q}^{(i)}\right)$,
    $\mathbf{x}^{(i)} = \mathbf{x}^{(i-1)} + \alpha^{(i)}\left(\mathbf{z}^{(i)} + \mathbf{q}^{(i)}\right)$,
check convergence, continue if necessary
end
```

3.7. Preconditioned conjugate gradient squared method (PCGS)

The pseudo code is given in the text that follows without any theoretical explanation.

```
1. initial guess $\mathbf{x}^{(0)}$ (for example $\mathbf{x}^{(0)} = 0$)
2. compute $\mathbf{r}^{(0)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(0)}$
3. choose $\tilde{\mathbf{r}}^{(0)}$ (for example $\tilde{\mathbf{r}}^{(0)} = \mathbf{r}^{(0)}$)
4. $\mathbf{q}^{(0)} = \mathbf{p}^{(0)} = 0$.
5. $\rho^{(0)} = 1$.
do for $i$=1, 2, …, iterations limit
    $\rho^{(i)} = \left(\tilde{\mathbf{r}}^{(i-1)}, \mathbf{r}^{(i-1)}\right)$,
    $\beta^{(i)} = \frac{\rho^{(i)}}{\rho^{(i-1)}}$,
    $\rho^{(i-1)} = \rho^{(i)}$,
    $\mathbf{z}^{(i)} = \mathbf{r}^{(i-1)} + \beta^{(i)}\mathbf{q}^{(i-1)}$,
    $\mathbf{p}^{(i)} = \mathbf{z}^{(i)} + \beta^{(i)}\left(\mathbf{q}^{(i-1)} + \beta^{(i)}\mathbf{p}^{(i-1)}\right)$,
solve $\mathbf{M}\hat{\mathbf{p}}^{(i)} = \mathbf{p}^{(i)}$
    $\hat{\mathbf{v}}^{(i)} = \mathbf{A}\hat{\mathbf{p}}^{(i)}$,
    $\sigma^{(i)} = \left(\tilde{\mathbf{r}}^{(i)}, \hat{\mathbf{v}}^{(i)}\right)$,
    $\alpha^{(i)} = \frac{\rho^{(i)}}{\sigma^{(i)}}$,
    $\mathbf{q}^{(i)} = \mathbf{z}^{(i)} - \alpha^{(i)}\mathbf{A}\hat{\mathbf{v}}^{(i)}$,
solve $\mathbf{M}\hat{\mathbf{z}}^{(i)} = \mathbf{z}^{(i)} + \mathbf{q}^{(i)}$
    $\mathbf{x}^{(i)} = \mathbf{x}^{(i-1)} + \alpha^{(i)}\hat{\mathbf{z}}^{(i)}$,
    $\hat{\mathbf{q}}^{(i)} = \mathbf{A}\hat{\mathbf{z}}^{(i)}$,
    $\mathbf{r}^{(i)} = \mathbf{r}^{(i-1)} - \alpha^{(i)}\hat{\mathbf{q}}^{(i)}$,
check convergence, continue if necessary
end
```

3.8. Bi-conjugate gradient stabilized method (BCGSTAB)

This method was developed by Van der Vorst [4] and it presents a modification of the CGS method.
The pseudo code is given in the following text:

```
1. initial guess $\mathbf{x}^{(0)}$ ( for example $\mathbf{x}^{(0)} = 0$ )
2. compute $\mathbf{r}^{(0)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(0)}$
3. $\mathbf{p}^{(0)} = \mathbf{r}^{(0)}$
4. $\mathbf{q}^{(0)} = \mathbf{v}^{(0)} = 0.$
5. $\hat{\omega}^{(0)} = \beta^{(0)} = \alpha^{(0)} = 1.$
do for $i$=1, 2, …, iterations limit
    $\hat{\beta}^{(i)} = \left(\mathbf{p}^{(i-1)}, \mathbf{r}^{(i-1)}\right),$
    $\omega^{(i)} = \frac{\hat{\beta}^{(i)}}{\beta^{(i-1)}} \times \frac{\hat{\omega}^{(i-1)}}{\alpha^{(i-1)}},$
    $\beta^{(i-1)} = \hat{\beta}^{(i)},$
    $\mathbf{q}^{(i)} = \mathbf{r}^{(i-1)} + \hat{\omega}^{(i)}\left(\mathbf{q}^{(i-1)} - \alpha^{(i-1)}\mathbf{v}^{(i-1)}\right),$
    $\mathbf{v}^{(i)} = \mathbf{A}\,\mathbf{q}^{(i)},$
    $\hat{\omega}^{(i)} = \frac{\hat{\beta}^{(i)}}{\left(\mathbf{p}^{(i)}, \mathbf{v}^{(i)}\right)},$
    $\mathbf{s}^{(i)} = \mathbf{r}^{(i-1)} - \hat{\omega}^{(i)}\mathbf{v}^{(i)},$
    $\mathbf{t}^{(i)} = \mathbf{A}\,\mathbf{s}^{(i)},$
    $\alpha^{(i)} = \frac{\left(\mathbf{t}^{(i)}, \mathbf{s}^{(i)}\right)}{\left(\mathbf{t}^{(i)}, \mathbf{t}^{(i)}\right)},$
    $\mathbf{x}^{(i)} = \mathbf{x}^{(i-1)} + \hat{\omega}^{(i)}\mathbf{q}^{(i)} + \alpha^{(i)}\mathbf{s}^{(i)},$
    $\mathbf{r}^{(i)} = \mathbf{s}^{(i)} - \alpha^{(i)}\mathbf{t}^{(i)},$
check convergence, continue if necessary
end
```

3.9. Preconditioned bi-conjugate gradient stabilized method (PBCGSTAB)

```
1. initial guess $\mathbf{x}^{(0)}$ ( for example $\mathbf{x}^{(0)} = 0$ )
2. compute $\mathbf{r}^{(0)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(0)}$
3. $\mathbf{p}^{(0)} = \mathbf{r}^{(0)}$
4. $\mathbf{q}^{(0)} = \mathbf{v}^{(0)} = 0.$
5. $\hat{\omega}^{(0)} = \beta^{(0)} = \alpha^{(0)} = 1.$
do for $i$=1, 2, …, iterations limit
    $\hat{\beta}^{(i)} = \left(\mathbf{p}^{(i-1)}, \mathbf{r}^{(i-1)}\right),$
    $\omega^{(i)} = \frac{\hat{\beta}^{(i)}}{\beta^{(i-1)}} \times \frac{\hat{\omega}^{(i-1)}}{\alpha^{(i-1)}},$
    $\beta^{(i-1)} = \hat{\beta}^{(i)},$
    $\mathbf{q}^{(i)} = \mathbf{r}^{(i-1)} + \hat{\omega}^{(i)}\left(\mathbf{q}^{(i-1)} - \alpha^{(i-1)}\mathbf{v}^{(i-1)}\right),$
  solve $\mathbf{M}\hat{\mathbf{q}}^{(i)} = \mathbf{q}^{(i)}$
    $\mathbf{v}^{(i)} = \mathbf{A}\hat{\mathbf{q}}^{(i)},$
    $\hat{\omega}^{(i)} = \frac{\hat{\beta}^{(i)}}{\left(\mathbf{p}^{(i)}, \mathbf{v}^{(i)}\right)},$
    $\mathbf{s}^{(i)} = \mathbf{r}^{(i-1)} - \hat{\omega}^{(i)}\mathbf{v}^{(i)},$
  solve $\mathbf{M}\hat{\mathbf{s}}^{(i)} = \mathbf{s}^{(i)}$
    $\mathbf{t}^{(i)} = \mathbf{A}\hat{\mathbf{s}}^{(i)},$
    $\alpha^{(i)} = \frac{\left(\mathbf{t}^{(i)}, \mathbf{s}^{(i)}\right)}{\left(\mathbf{t}^{(i)}, \mathbf{t}^{(i)}\right)},$
    $\mathbf{x}^{(i)} = \mathbf{x}^{(i-1)} + \hat{\omega}^{(i)}\mathbf{q}^{(i)} + \alpha^{(i)}\mathbf{s}^{(i)},$
    $\mathbf{r}^{(i)} = \mathbf{s}^{(i)} - \alpha^{(i)}\mathbf{t}^{(i)},$
check convergence, continue if necessary
end
```

4. NUMERICAL RESULTS AND DISCUSSION

The analysis, as mentioned earlier, considers the brick (Fig. 1) that is divided into 1000, 8000, 27000, 64000 and 125000 hexahedral elements. The back face of the brick is kept at $0\,°\mathrm{C}$ while the initial temperature was $100\,°\mathrm{C}$.

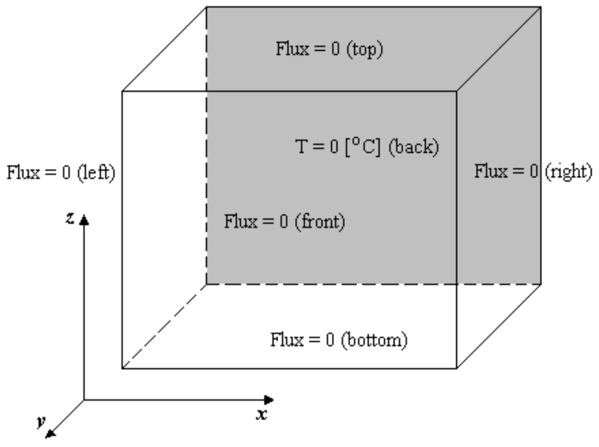

Figure 1. Geometry with boundary conditions

The analysis considers a several comparisons, in the following order: type of finite element, time stepping and iterative solvers. Also, the analysis considers temperature dependence on different meshes for the chosen time stepping (Crank-Nicolson).

Fig. 2 presents transient temperature through the brick for 8 noded and 20 noded hexahedron elements. From the Fig. 2 we can see that 8-noded elements give better results than 20-noded elements.

Fig. 3 presents results using different methods for time stepping.

To make our comparison between different time stepping schemes clearer temperature changes occurring at node 1 are shown in Fig. 4.

At the same time few different meshes were studied for the chosen time stepping (Crank-Nicolson). Obtained temperatures at node 1 are presented in Fig. 5.

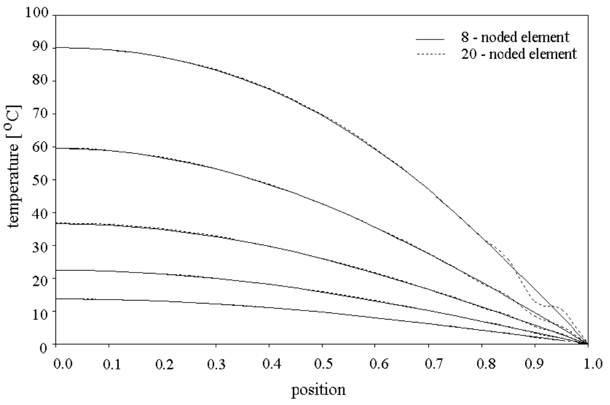

Figure 2. Transient temperature for 8 and 20 node elements

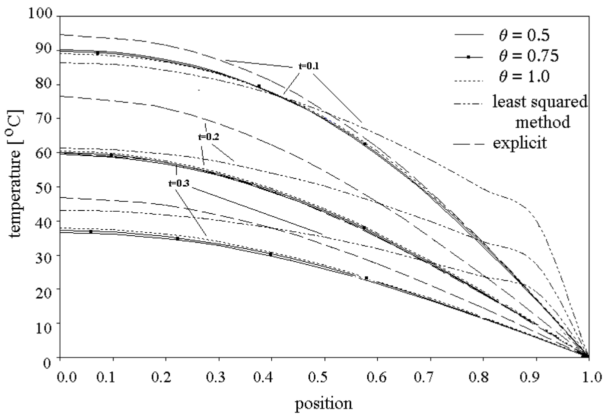

Figure 3. Transient temperatures for different time stepping

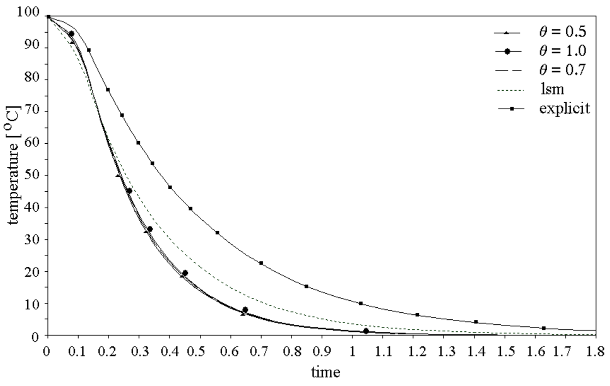

Figure 4. Transient temperatures at node 1

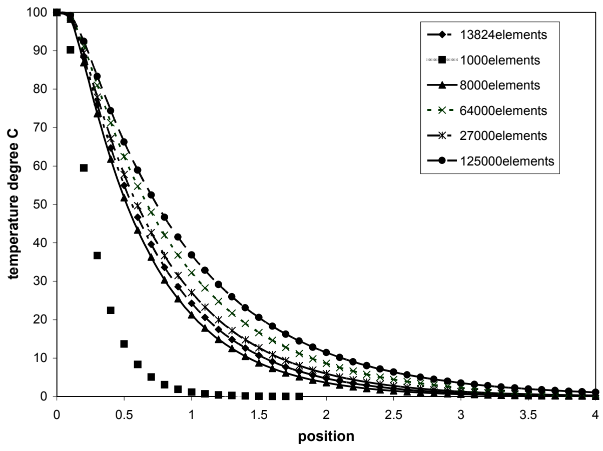

Figure 5. Transient temperatures at node 1

In the following figures (Fig. 6, Fig. 7 and Fig. 8) the decreasing trend of the absolute values of the residuals (a characteristic of iterative solvers) is shown.

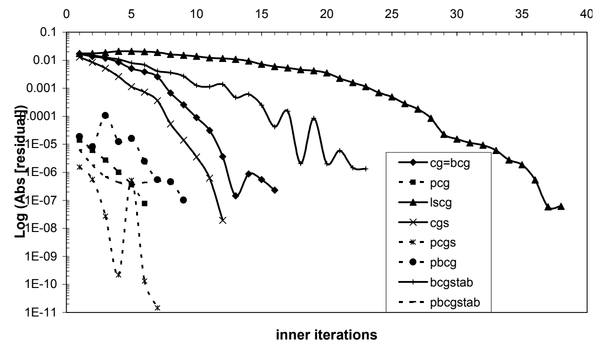

Figure 6. Residuals at time t=0.1 for node 1

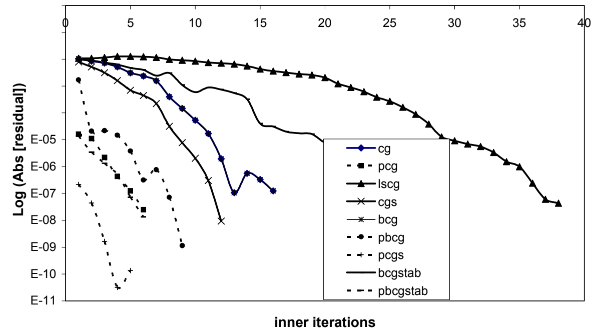

Figure 7. Residuals at time t=0.5 for node 1

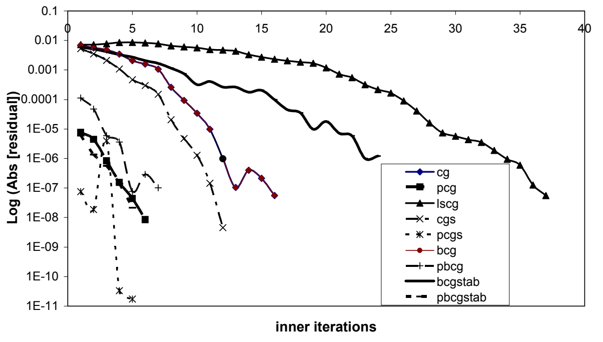

Figure 8. Residuals at time t=1.0 for node 1

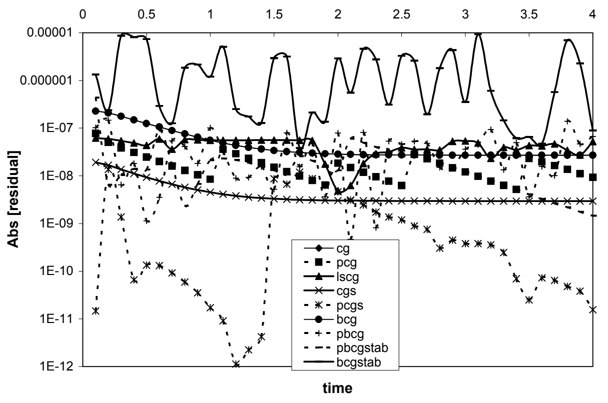

Figure 9. Residuals of the last iterations in time

Finally, in the Fig. 9 we gave the absolute value of the residuals at last iteration during time.

From the diagrams it can be seen that the least squared conjugate gradient method (LSCG) has a monotonically decreasing residual where convergence is the slowest of all the methods presented in this paper.

The number of inner iterations is a very useful parameter for our comparison. According to the number of inner iterations, as seen in Fig. 10, it can be seen that the LSCG method requires the highest number of inner iterations while the preconditioned conjugate gradient method requires the least.

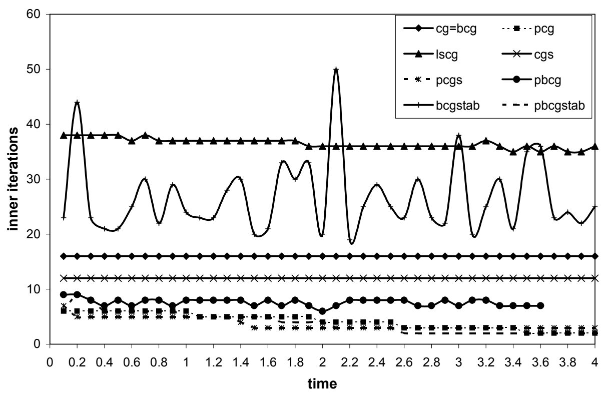

Figure 10. Number of inner iterations in time

Sometimes, taking the number of inner iterations as the main assessment criteria can lead to incorrect conclusions. For example, computational time (CPU) required to solve the problem using BCG method is twice that spent using the CG method although there are the same number of inner iterations for both. So, computational time becomes a very important parameter in any analysis of iterative solvers. In the table presented below CPU times for the solvers are shown. All these results are for the numerical scheme that includes 1000 elements with 10x10x10 divisions.

Table 1. CPU time for chosen solvers

solvers	CPU[s]
CG	11.4064
PCG	3.264694
LSCG	55.61998
CGS	15.87282
PCGS	6.409216
BCG	26.48809
PBCG	14.30056
BCGSTAB	34.90018
PBCGSTAB	6.970022
Gauss	6.569446

Table 2. CPU time for different mesh

mesh	PCG	Gauss
20x20x20	62.44	177.92
30x30x30	303.81	1998.08
40x40x40	927.02	No point to run
50x50x50	2159.42	No point to run

The last method is method based on Gauss elimination method with preconditioning. For preconditioning is used incomplete Cholesky factorization. An increase of number of elements will effect that CPU time will increase rapidly. In the following table we present CPU time for the PCG and Gauss method.

An analysis of the selected solvers shows that the most time-consuming operation is MATMUL. During our calculation this operation is used at least two times. The results presented in the previous table are obtained using MATMUL operation as matrix-vector product. Instead of using matrix-vector product we can use matrix-matrix product and accelerate our calculation by almost three times. For the example, we have that CPU time for the PCG method is 1.09157 [s] and for BCG method is 9.253305 [s].

5. CONCLUSION

Iterative solvers have great advantages over direct solvers [5] when one has to solve industrial problems with possibly hundreds of thousands elements or more. One of the parameters such as CPU time where we needed 2159.42[s] to solve our problem applying slower version of PCG method (faster one will give CPU=712[s]) while there wasn't any point to apply the Gauss method, was shown.

Also, when solving symmetric positive definite matrix systems, the PCG method appears to be the fastest. Unfortunately, not all industrially relevant partial differential equations are Laplace or Poisson equations. In the case that we have to deal with non symmetric matrix system (the finite element discretization of Navier-Stokes equations) other conjugate gradient methods such as BCG, BCGSTAB, CGS and LSCG appear very useful tool for solving these equations.

REFERENCES

[1] Hestenes M., Stiefel E.: Methods of conjugate gradients for solving linear systems of equations, J. Res. Nat. Bur. Stan., Vol.49, pp. 409-435, 1952.

[2] Fletcher R.: *Conjugate gradient methods for indefinite systems*, Lecture Notes in Mathematics, Springer, Berlin, pp. 73-89, 1976.

[3] Sonneveld P.: CGS, a fast Lancosz-type solver for non-symmetric linear systems, SIAM J. Sci. Stat. Comput., Vol. 10, pp. 36-52, 1984.

[4] Van der Vorst H., BCGSTAB: a fast and smoothly converging variant of BCG for the solution of non-symmetric linear systems, SIAM J.Sci. Stat. Comp., Vol. 18, pp. 631-634, 1992.

[5] Irons B.: A frontal solution program for finite element analysis, Int. J. Num. Meth. Eng., Vol. 10, 1970.

[6] Lewis R. W., Morgan K., Thomas H. R., Seetharamu K. N.: *The Finite Element Method in Heat Transfer Analysis*, John Willey and Sons, Chichester, 1996.

[7] Smith I. M., Griffiths D. V.: *Programming-The Finite Element Method*, John Willey and Sons, Chichester, 1998.

[8] Golub G., Van Loan C.: *Matrix Computations*, John Hopkins University Press, 1996.

[9] Saad Y., Iterative Methods for Spars Linear Systems, SIAM 2003 (PWS 1996), 2000 .

[10] Lavery N.P., Taylor C.: Iterative and Multigrid Methods in the Finite Element Solution of Incompressible and Turbulent Fluid Flow, International Journal for Numerical Methods in Fluids, John Willey and Sons, Vol. 30, No. 6, pp 609-634, 1999.