Solving Diffusion Equation

Saeed Taghavi[1], Sarah Yousefizadeh[2]
[1] s.taghavi@iasbs.ac.ir
[2] yousefi@iasbs.ac.ir

July 17, 2019

Abstract

We consider computational issues arising from the basic conjugate gradient algorithm as well as surveying typical application problems that require an iterative solution of large martix-formulated problems.

Contents

Introduction

An excellent review of iterative solvers and some of the general computational issues for their efficient implementation is given in [1].
Consider application problems that can be formulated in terms of the matrix equation $A\vec{x} = \vec{b}$. The structure of matrix A is highly dependent on the particular type of application and some applications such as computational electromagnetics give rise to a matrix that is effectively dense [2] and can be solved using direct methods [3] such as Guassian elimination, whereas others such as computational fluid dynamics [4] generate a matrix that is sparse, having most of its elements identically zero. Conjugate gradient (CG) and other iterative methods are prefered over simple Guassian elimination when A is very large and sparse, and where storage space for the full matrix would either be impractical or too slow to access through the secondary memory system. A large number of computational number of computationally

expensive scientific and engineering applications, e.g. structural analysis, fluid dynamics, aerodynamics, lattice gauge simulation, and cricuit simulation, are based on the solution of large sparse systems of linear equations. Iterative methods are employed in many of these applications. While the CG method itself is no longer considered state-of-the art in terms of its numerical stability and convergence prperties, its computaional structure is similar to that of methods such as Bi-Conjugate Gradient (BiCG). CG codes have been used in a number of benchmark suites [1] as PARKBENCH [5] and NAS [6].

In this report we focus on solving a diffution problem using CG algorithm. We start by explaning the CG method for a linear system, then we explaine our problem and discretizing it; finally we use CG method to solve our problem.

1 Relevant Definitions and Notation

Partial Differential Equations (PDEs) is an equation in which the solution is a function which has two or more independent variables. Linear PDEs are PDEs in which the unknown function and its derivatives all have power 1. The CG method is utilized to estimate solutions to linear PDEs.
A solution to a PDE can be found by linearizing into a system of linear equations:

$$\begin{aligned}
a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n &= b_1 \\
a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n &= b_2 \\
\vdots \quad \vdots \quad \ddots \quad \vdots &= \vdots \\
a_{n1}x_1 + a_{n2}x_2 + ... + a_{nn}x_n &= b_n
\end{aligned}$$

This system of equations will be written in matrix form $A\vec{x} = \vec{b}$, where

$$A = \begin{bmatrix} a_{11} & a_{12} & ... & a_{1n} \\ a_{21} & a_{22} & ... & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & ... & a_{nn} \end{bmatrix} \qquad x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \qquad b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \tag{1}$$

The matrix A is referred to as the Coefficient Matrix, and x the Solution Matrix.
A matrix is defined to besparseif a large proportion of its entries are zero.
A $n \times n$ matrix A is defined to be symmetric if $A = A^T$.
A $n \times n$ matrix A is Positive Definite if $x^T Ax > 0$ for every n-dimensional column matrix $x \neq 0$, where x^T is the transpose of vector x.
Typically systems of equations arising from linearizing PDEs yield coefficient matrices which are sparse. However, when discussing large sparse matrix solutions, it is not sufficient to know that the coefficient matrix is sparse. Systems arising from linear PDEs have a coefficient matrix which have non-zero diagonals and all other entries zero. When programming the CG method, a programmer can utilize the sparse characteristic and only program the non-zero diagonals to reduce storage and improve efficiency. In order for the CG method to yield a solution the coefficient matrix needs to be symmetric and positive definite.

2 Linearization of PDEs

There are several established methods to linearize PDEs. The method that will be discussed is finite differences (FD). The solution of PDEs by means of FD is based on approximating derivatives of continuous functions, i.e. the actual partial differential equation, by discretized versions of the derivatives based on discrete points of the functions of interest. For the following example for linearizing the one-dimensional heat equation, the Forward Difference Method is utilized. Note that this process will work for all linear

[1]In computing, a benchmark is the act of running a computer program, a set of programs, or other operations, in order to assess the relative performance of an object, normally by running a number of standard tests and trials against it. For more information read the appendix.

PDEs.
Finite difference approximations to PDEs can be derived through the use of Taylor series expansions. Suppose we have a function $f(x)$, which is continuous and differentiable over the range of interest. Lets also assume we know the value $f(x_i)$ and all the derivatives at $x = x_i$. The forward Taylor-series expansion for $f(x_i + \Delta x)$, away from the point x_i by a small amount h (sometimes here also denoted by Δx), gives:

$$f(x_i + h) = f(x_i) + \frac{\partial f(x_i)}{\partial x} h + ... + \frac{\partial^n f(x_i)}{\partial x^n} \frac{h^n}{n!} \tag{2}$$

We can express the first derivative of f by rearranging equation 2:

$$\frac{\partial f(x_i)}{\partial x} = \frac{f(x_i + h) - f(x_i)}{h} - ... - \frac{\partial^n f(x_i)}{\partial x^n} \frac{h^n}{hn!} \tag{3}$$

If we now only compute the first term of this equation as an approximation, we can writea discretized version:

$$\frac{\partial f(x_i)}{\partial x} = \frac{f_{i+1} - f_i}{h} \tag{4}$$

it is called the *forward FD derivative*, where functions $f_i = f(x_i)$ are evaluated at discretely spaced x_i with $x_{i+1} = x_i + h$, wherethe node spacing, or *resolution*, h (or Δx) is assumed constant.

We can also expand the Taylor series backward.

$$f(x_i - h) = f(x_i) - \frac{\partial f(x_i)}{\partial x} h + ... + \frac{\partial^n f(x_i)}{\partial x^n} \frac{-1^n h^n}{n!} \tag{5}$$

In this case, the first, backward difference can be obtained by

$$\frac{\partial f(x_i)}{\partial x} = \frac{f_i - f_{i-1}}{h} \tag{6}$$

By adding equations 4 and 6 an approximation of the second derivative is obtained

$$f_i'' = \frac{f_{i+1} - 2f_i + f_{i-1}}{h^2} \tag{7}$$

3 One-dimensional heat equation

Now, consider the one-dimensional heat equation:

$$\frac{\partial \phi(x,t)}{\partial t} = D \frac{\partial^2 \phi(x,t)}{\partial^2 x} \tag{8}$$

We are interested in the evolution of $\phi(x,t)$ in time which satisfies equation 8. The first step in the finite differences method is to construct a grid with points on which we are interested in solving the equation (this is called discretization, see figure 1).

The next step is to replace the continuous derivatives of equation 8 with their finite difference approximations. The derivative of ϕ versus time ($\frac{\partial \phi}{\partial t}$) can be approximated with a forward finite difference approximation as

$$\frac{\partial \phi}{\partial t} \approx \frac{\phi_i^{n+1} - \phi_i^n}{t^{n+1} - t^n} = \frac{\phi_i^{n+1} - \phi_i^n}{\Delta t} \tag{9}$$

Here, n represents the time step; and the subscript i refers to the location. Both n and i are integers; n varies from 1 to n_t (total number of time steps) and i varies from 1 to n_x (total number of grid points in x-direction).The spatial derivative of equation 8 is replaced by a central finite difference approximation, i. e.

$$\frac{\partial^2 \phi}{\partial x^2} = \frac{\partial}{\partial x} \frac{\partial \phi}{\partial x} \approx \frac{\frac{\phi_{i+1}^n - \phi_i^n}{\Delta x} - \frac{\phi_i^n - \phi_{i-1}^n}{\Delta x}}{\Delta x} = \frac{\phi_{i+1}^n - 2\phi_i^n + \phi_{i-1}^n}{(\Delta x)^2} \tag{10}$$

Substituting equations 9 and 10 into equation 8 gives:

$$\frac{\phi_i^{n+1} - \phi_i^n}{\Delta t} = D \frac{\phi_{i+1}^n - 2\phi_i^n + \phi_{i-1}^n}{(\Delta x)^2} \tag{11}$$

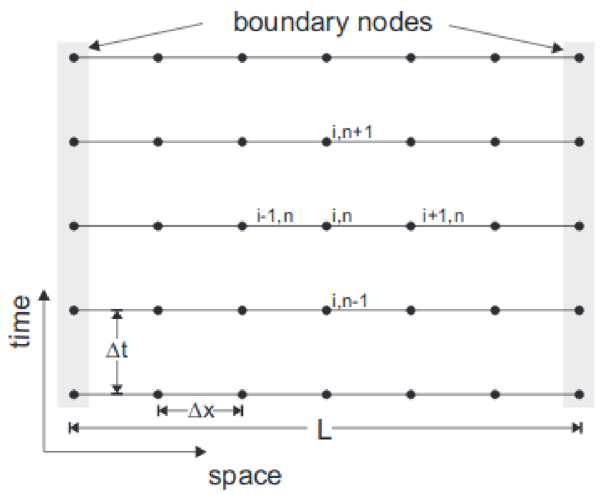


Figure 1: Finite difference discretization of the 1D heat problem

Rearrenging the equation 11 gives an evolutionary equation for ϕ, this method is called *explicit scheme*. An alternative approach is an *implicit finite difference scheme*, where the spatial derivatives $\frac{\partial^2 \phi}{\partial x^2}$ are evaluated at the new time step. The simplest implicit discretization of equation 8 is

$$\frac{\phi_i^{n+1} - \phi_i^n}{\Delta t} = D \frac{\phi_{i+1}^{n+1} - 2\phi_i^{n+1} + \phi_{i-1}^{n+1}}{(\Delta x)^2} \tag{12}$$

A *fully implicit* scheme where the time derivative is taken backward can be rearranged so that unknown terms are on the left and known terms are on the right

$$-s\phi_{i+1}^{n+1} + (1+2s)\phi_i^{n+1} - s\phi_{i-1}^{n+1} = \phi_i^n \tag{13}$$

where

$$s = \frac{D\Delta t}{(\Delta x)^2} \tag{14}$$

Note that in this case we no longer have an explicit relationship for ϕ_i^{n+1}, ϕ_{i-1}^{n+1} and ϕ_{i+1}^{n+1}. Instead, we have to solve a linear system of equations, which is discussed further below.
We solve the heat equation (8) on the domain $0 \leq x \leq 1$ subject to the following boundary conditions for fixed temperature

$$\phi(x=0) = \phi_{left} \tag{15}$$
$$\phi(x=1) = \phi_{right} \tag{16}$$

Starting with fixed temperature BCs (15), the boundary condition on the left boundary gives

$$\phi_1 = \phi_{left}, \tag{17}$$

and the one on the right

$$\phi_{nx} = \phi_{right}. \tag{18}$$

Equation 13, 17 and 18 can be written in matrix form as

$$A\vec{x} = \vec{b}. \tag{19}$$

Acording to equation 13, for a five-node grid, for example, the linear system becomes

$$\begin{aligned}
&grid\ point\ \#1: && \phi_1^{n+1} = \phi_1^n && = \phi_{left}^n \\
&grid\ point\ \#2: && -s\phi_1^{n+1} + (1+2s)\phi_2^{n+1} - s\phi_3^{n+1} && = \phi_2^n \\
&grid\ point\ \#3: && -s\phi_2^{n+1} + (1+2s)\phi_3^{n+1} - s\phi_4^{n+1} && = \phi_3^n \\
&grid\ point\ \#4: && -s\phi_3^{n+1} + (1+2s)\phi_4^{n+1} - s\phi_5^{n+1} && = \phi_4^n \\
&grid\ point\ \#5: && \phi_5^{n+1} = \phi_5^n && = \phi_{right}^n
\end{aligned}$$

thus the coefficient matrix A is

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -s & (1+2s) & -s & 0 & 0 \\ 0 & -s & (1+2s) & -s & 0 \\ 0 & 0 & -s & (1+2s) & -s \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{20}$$

and vectors $\vec{x}$ and $\vec{b}$ are

$$x = \begin{bmatrix} \phi_1^{n+1} \\ \phi_2^{n+1} \\ \phi_3^{n+1} \\ \phi_4^{n+1} \\ \phi_5^{n+1} \end{bmatrix} \qquad b = \begin{bmatrix} \phi_{left} \\ \phi_2^n \\ \phi_3^n \\ \phi_4^n \\ \phi_{right} \end{bmatrix} \tag{21}$$

Note that matrix A will have a unity entry on the diagonal and zero else for each node where Dirichlet (fixed temperature) boundary conditions apply.
In case of the Dirichlet boundary condition, it is better if we get rid of the boundary points (ϕ_{left} and ϕ_{right}) in the solution vector, that is we only take the evolving points. In this case, we have to take the effects of boundaries in a boundary condition vector on the right hand side of the equation 19, and we can rewrite the equation as:

$$A'\vec{x'} = \vec{b'} + \vec{b.c.} \tag{22}$$

where, for a five-node grid the coefficient matrix is:

$$A' = \begin{bmatrix} (1+2s) & -s & 0 \\ -s & (1+2s) & -s \\ 0 & -s & (1+2s) \end{bmatrix} \tag{23}$$

and vectors $\vec{x}$, $\vec{b}$, and $\vec{b.c.}$ are:

$$\vec{x'} = \begin{bmatrix} \phi_2^{n+1} \\ \phi_3^{n+1} \\ \phi_4^{n+1} \end{bmatrix} \qquad \vec{b'} = \begin{bmatrix} \phi_2^n \\ \phi_3^n \\ \phi_4^n \end{bmatrix} \qquad \vec{b.c.} = s \begin{bmatrix} \phi_{left} \\ 0 \\ \phi_{right} \end{bmatrix} \tag{24}$$

Linear system 22 is like linear system 19, but only for the evolving points (we have ignored the boundaries). Matrix A also has an overall peculiar form because most entries off the diagonal are zero. This sparseness can be exploited by specialized linear algebra routines, both in terms of storage and speed. By avoiding computations involving zero entries of the matrix, much larger problems can be handled than would be possible if we were to store the full matrix. In particular, the fully implicit FD scheme leads to a tridiagonal system of linear equations that can be solved with various well known methods. Thus solving a PDE is reduced to solve a system of linear equations. There are many methods of numerical solution of linear systems.

4 Numerical Solution of Linear Systems

The problem of solving the equation $A\vec{x} = \vec{b}$ is to find the inverse of the coefficient matrix (A^{-1}). Multiplying A^{-1} to the both sides of the equation leads to $A^{-1}A\vec{x} = A^{-1}\vec{b}$, with $A^{-1}A = \mathbf{1}$, it becomes $\vec{x} = A^{-1}\vec{b}$. So, it is clear that, all we have to do is to calculate the inverse of the coefficient matrix (A^{-1}). numerical methods for solving linear systems of equations can generally be divided into two classes [7] :

- **Direct methods:** In the absence of rundoff error (assuming infinite precision!) such methods would yeild the exact solution, such as Gaussian Elimination, LU Decomposition, and etc.;
- **Iterative methods:** These methods are consist of approximating solutions; start with a first approximation $x^{(0)}$ and compute iteratively a sequence of (hopefully increasingly better) approximations $x^{(i)}$, without ever reaching x. GaussSeidel Method, Conjugate Gradient Method, and etc. are some well known iterative methods.

We will explaine more about LU Decomposition, and Conjugate Gradient Method in the next sections.

4.1 LU Decomposition

The LU decomposition method consists in factorizing A into a product of two triangular matrices $A = LU$, where L is lower triangular and U is upper triangular. This decomposition allows us to reduce the solution of the system $Ax = b$ to solving two triangular systems $Ly = b$ and $Ux = y$. Because L and U are triangular matrices finding the inverse matrix for them is much more easier than finding the inverse matrix for a general matrix, so the LU factorization provides a way of computing A^{-1}. Here is two-step procedure to find the inverse of a matrix A:

1. Find the LU decomposition $A = LU$;
2. Find the inverse of $A^{-1} = U^{-1}L^{-1}$ by inverting the matrices U and L.

There are two subroutines in the Linear Algebra PACKage (LAPACK) which do the same job:

- the **SGETRF** subroutine [2] computes an LU factorization of a general M-by-N matrix A using partial pivoting with row interchanges.
- the **SGETRI** subroutine computes the inverse of a matrix using the LU factorization computed by SGETRF.

Our algorithm for solving the one-dimensional heat equation using LU fectorization is summarised as:

1. assembling system in a desired boundary conditions and building the coeficient matrix(A);
2. calculating the inverse of the coeficient matrix(A^{-1}) using LU decomposition;
3. setting the initial conditions;
4. loop over time:
   (a) $\phi^n = A^{-1}\phi^{n-1}$;

4.2 Conjugate Gradient Algorithm

The classic Conjugate Gradient non-stationary iterative algorithm as defined in [8] and references therein can be applied to solve symmetric positive-definite matrix equations. They are preferred over simple Gaussian algorithms because of their faster convergence rate if A is very large and sparse.
consider the prototype problem $A\vec{x} = \vec{b}$ to be solved for $\vec{x}$ which can be expressed in the form of iterative equations for the solution $\vec{x}$ and residual (gradient) $\vec{r}$ [9].

$$\vec{x^k} = \vec{x^{k-1}} + \alpha^k \vec{p^k} \tag{25}$$

$$\vec{r^k} = \vec{r^{k-1}} - \alpha^k \vec{q^k} \tag{26}$$

where the new value $vecx$ is a function of its old value, the scaler step size α and the search direction vector $\vec{p}$ at the k'th iteration and $\vec{q^k} = A\vec{p^k}$.
The values of x are guaranteed to converge in, at most, n iterations, where n is the order of the system, unless the problem is ill-conditioned in which case roundoff errors often prevent the algorithm from

[2]check `www.netlib.org` for more information on these subroutines

furnishing a suffciently precise solution at the nth step. In well-conditioned problems, the number of iterations necessary for satisfactory convergence of the conjugate gradient method can be much less than the order of the system. Therefore, the iterative procedure is continued until the residual $\vec{r^k} = \vec{b}^k - A\vec{x}^k$ meets some stopping criterion, typically of the form: $\|\vec{r^k}\| \leq$ tol. (tol.is a tolerance level). The CG algorithm uses:

$$\alpha = (\vec{r^k}.\vec{r^k})/(\vec{p^k}.A\vec{p^k}) \tag{27}$$

with the search directions chosen using:

$$\vec{p^k} = \vec{r}^{k-1} + \beta^{k-1}\vec{p}^{k-1} \tag{28}$$

with

$$\beta^{k-1} = (\vec{r}^{k-1}.\vec{r}^{k-1})/(\vec{p}^{k-2}.A\vec{p}^{k-2}) \tag{29}$$

which ensures that the search directions form an A-orthogonal system. The non-preconditioned CG algorithm for the initial "guessed" solution vector $\vec{x}^0 = 0$ is summarised as:

```
$\vec{p} = \vec{r} = \vec{b};\ \vec{x} = 0;\ \vec{q} = A\vec{p}$
$\rho = \vec{r}.\vec{r};\ \alpha = \rho/(\vec{p}.\vec{q})$
$\vec{x} = \vec{x} + \alpha\vec{p};\ \vec{r} = \vec{r} - \alpha\vec{q}$
DO $k = 2, Niter$
$\rightarrow \rho_0 = \rho;\ \rho = \vec{r}.\vec{r};\ \beta = \rho_0/\rho$
$\rightarrow \vec{p} = \vec{r} + \beta\vec{p};\ \vec{q} = A\vec{p}$
$\rightarrow \alpha = \rho/(\vec{p}.\vec{q})$
$\rightarrow \vec{x} = \vec{x} + \alpha\vec{p};\ \vec{r} = \vec{r} - \alpha\vec{q}$
$\rightarrow$ IF (stop criterion) EXIT
END DO
```

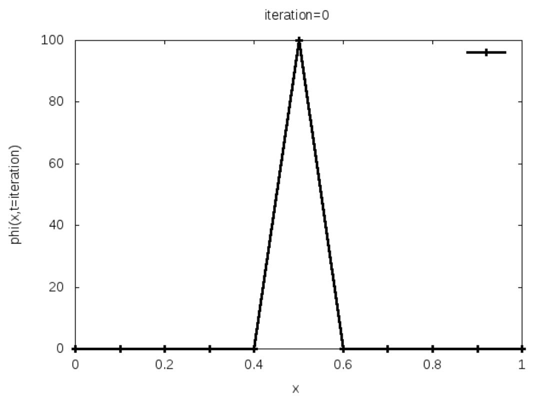
The Conjugate Gradient method involves one matrix-vector product, three vector updates, and two inner products per iteration. Implementation of this algorithm requires storage for four vectors: $\vec{x}$, $\vec{r}$, $\vec{r}$ and $\vec{q}$ as well as the matrix A and working scalars α and β.
Note that there are other CG Algorithms too, such as Bi-Conjugate Gradient (BiCG) method which can be applied to non-symmetric matrices.
Our algorithm for solving the one-dimensional heat equation using CG is summarised as:

1. Assembling system in a desired boundary conditions and building the coeficient matrix(A);
2. setting the initial conditions;
3. loop over time:
    (a) Using CG to fing the ϕ for the next timestep;

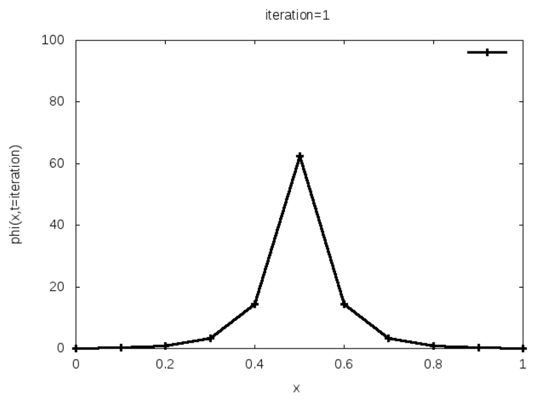
Also there are maximum tolrrance and maximum iteration for the CG algorithm to be sure whenever each one satisfied the CG loop will break.

Algorithm 1: How to write algorithms

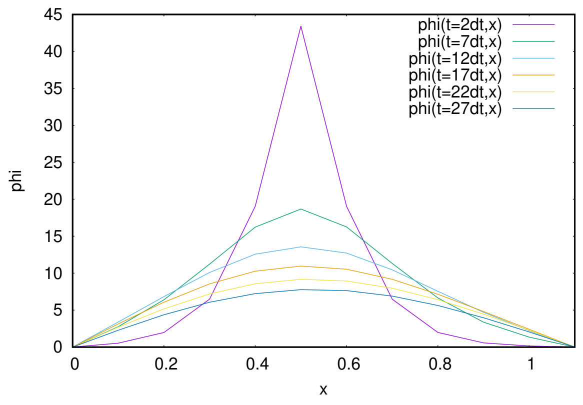
```
   Data: this text
   Result: how to write algorithm with LaTeX2e
1  initialization;
2  while not at end of this document do
3      read current;
4      if understand then
5          go to next section;
6          current section becomes this one;
7      else
8          go back to the beginning of current section;
9      end
10 end
```



(a) initial condition for one-dimensional heat equation



(b) the heat distribution after one timestep.



(c) the heat distribution after several timesteps.

Figure 2: solving one-dimensional heat equation using LU factorization.

5 Results for one-dimensional heat equation

We have tried both LU decomposition and CG methods for solving one-dimentional heat equation, to compare their efficiency.

The parameters set to $D = 1.0$, $dt = 2.0^{-8.0}$, $x = [0, 1.1]$, $n_x = 10$, and for initial condition we have $phi(x - 0.5) = 100.0$ (figure 2a). Solving ϕ for the next timesteps leads to figures 2b and 2c.

We can study several initial conditions to get a deeper insight into the different diffusion problems(figures 4a and 4b).

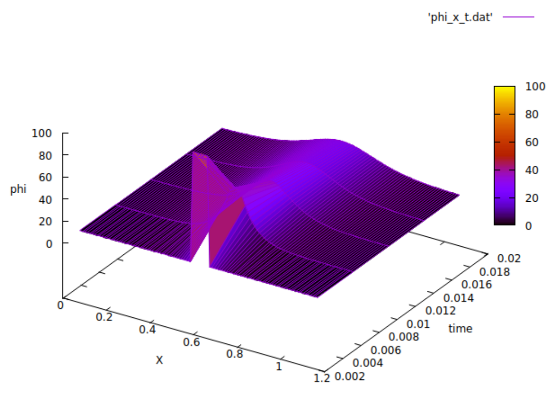
todo: compare the run time of two methods.

In the next section we will explaine all the issues and considerations for the two-dimentional heat equation.

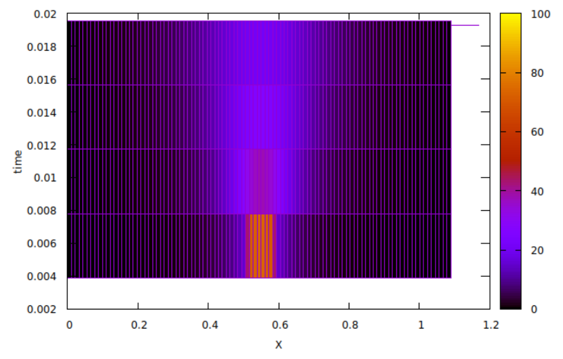
6 Two-dimensional diffusion equation

Now we consider two-dimentional diffusion equation; if the diffusion coefficient (D) is costant, then the two-dimensional diffusion equation reduces to the following linear equation:

$$\frac{\partial \phi(\vec{r}, t)}{\partial t} = D\nabla^2 \phi(\vec{r}, t) \tag{30}$$

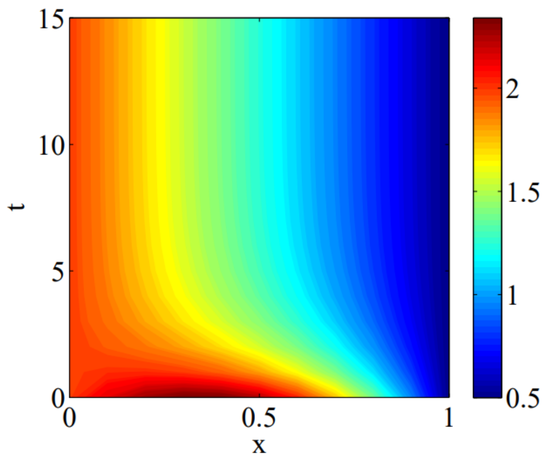


(a) the heat distribution over time

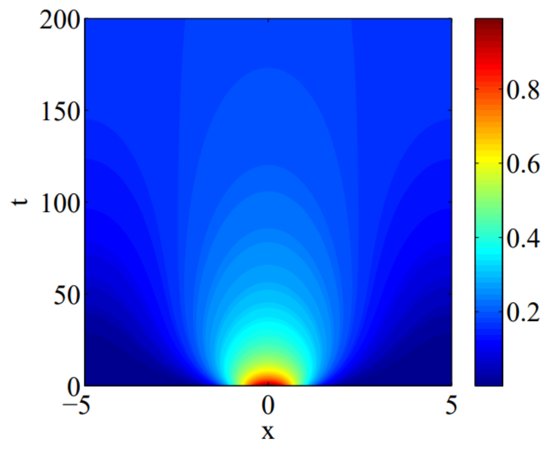


(b) the heat distribution map over time.

Figure 3: solving one-dimensional heat equation using CG method.



(a) Contour plot of the heat distribution, $\phi(0,t) = T_l$, $\phi(L,t) = T_r$.



(b) Contour plot of the diffusion of the initial Gauss pulse.

Figure 4: solving one-dimensional heat equation using CG method.

and if we assume that $\vec{r} = (x, z)$ it becomes to

$$\frac{\partial \phi(x,z,t)}{\partial t} = D\left(\frac{\partial^2 \phi(x,z,t)}{\partial x^2} + \frac{\partial^2 \phi(x,z,t)}{\partial z^2}\right). \tag{31}$$

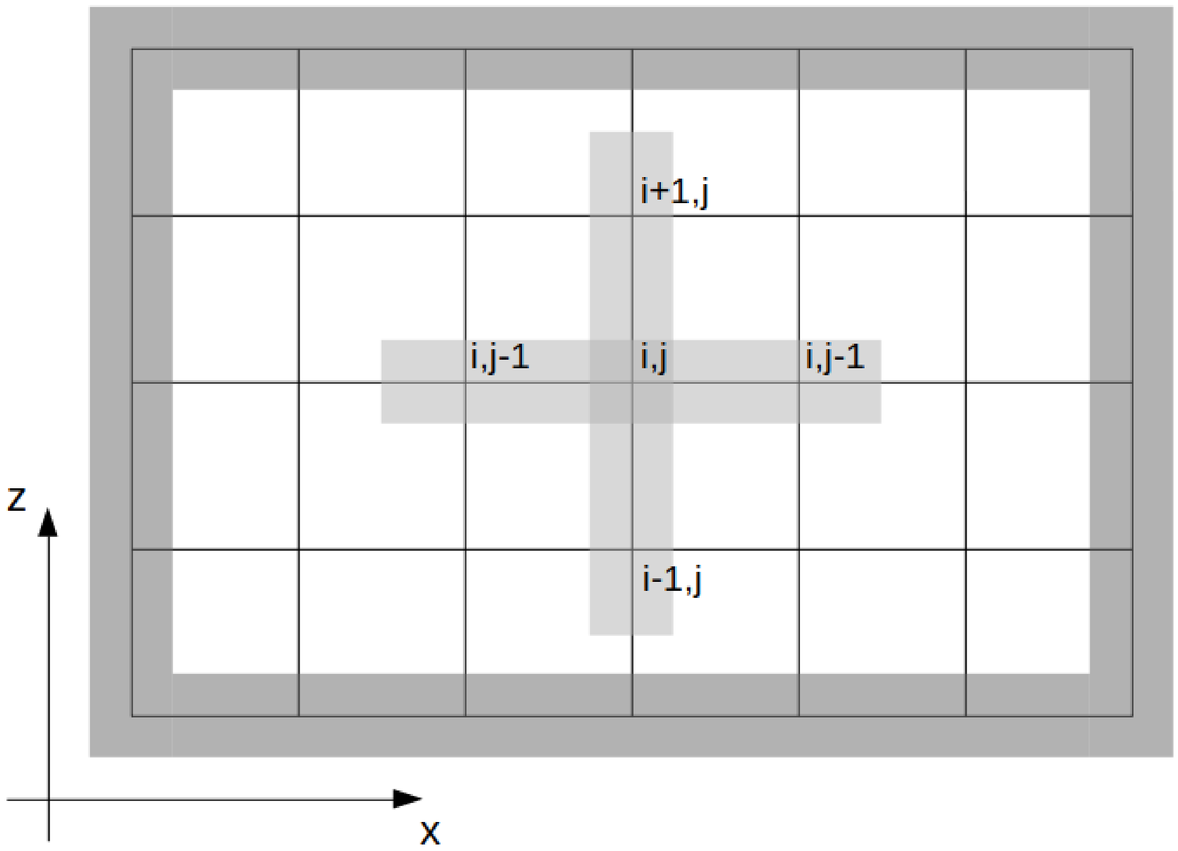


Figure 5: Finite difference discretization of the 2D heat problem

According to the previous explanation on Taylor-series expansion and Linearization of PDEs, if we employ a fully implicit, unconditionally stable discretization scheme as we have done for the 1D equation , equation 55 can be written as

$$\frac{\phi_{i,j}^{n+1} - \phi_{i,j}^{n}}{\Delta t} = D(\frac{\phi_{i,j+1}^{n+1} - 2\phi_{i,j}^{n+1} + \phi_{i,j-1}^{n+1}}{(\Delta x)^2} + \frac{\phi_{i+1,j}^{n+1} - 2\phi_{i,j}^{n+1} + \phi_{i-1,j}^{n+1}}{(\Delta z)^2})$$

Rearranging terms with $n+1$ on one side and terms with n on the other side gives

$$\phi_{i,j}^{n} = - s_z \phi_{i+1,j}^{n+1} - s_x \phi_{i,j+1}^{n+1} + (1 + 2s_z + 2s_x)\phi_{i,j}^{n+1} - s_z \phi_{i-1,j}^{n+1} - s_x \phi_{i,j-1}^{n+1}$$

As in the 1D case, we have to write these equations in a matrix A and a vector $\vec{b}$. From a practical point of view, this is a bit more complicated than in the 1D case, since we have to deal with book-keeping issues, i.e. the mapping of $\phi_{i,j}$ to the entries of a vector Φ_k.
If a 2D scalar field is to be solved for with an equivalent vector Φ, the nodes have to be numbered continuously, for example as in Figure 8. The derivative versus x-direction is then e.g.

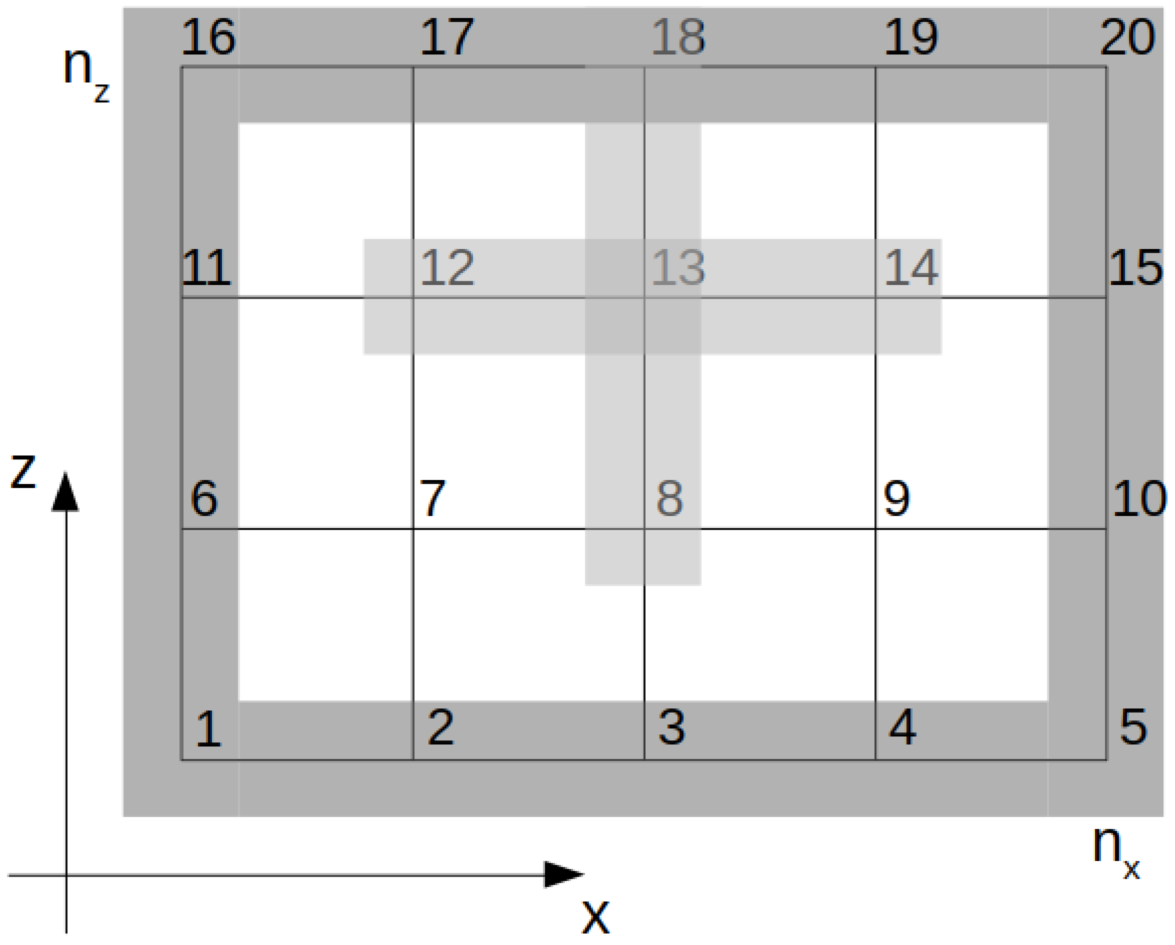


Figure 6: Numbering scheme for a 2D grid with $n_x = 5$ and $n_z = 4$.

$$\frac{\partial^2 \phi}{\partial x^2}|_{i=3,j=3} = \frac{1}{(\Delta x)^2}(\Phi_{14} - 2\Phi_{13} + \Phi_{12}) \tag{32}$$

and the derivative versus z-direction is given by

$$\frac{\partial^2 \phi}{\partial z^2}|_{i=3,j=3} = \frac{1}{(\Delta z)^2}(\Phi_{18} - 2\Phi_{13} + \Phi_{8}) \tag{33}$$

If n_x are the number of grid points in x-direction and n_z the number of points in z-direction, we can write equations 58 and 59 in a more general way as:

$$\frac{\partial^2 \phi}{\partial x^2}|_{i,j} = \frac{1}{(\Delta x)^2}(\phi_{(i-1)n_x+j+1} - 2\phi_{(i-1)n_x+j} + \phi_{(i-1)n_x+j-1}) \tag{34}$$

$$\frac{\partial^2 \phi}{\partial z^2}|_{i,j} = \frac{1}{(\Delta z)^2}(\phi_{in_x+j} - 2\phi_{(i-1)n_x+j} + \phi_{(i-2)n_x+j}) \tag{35}$$

As an example, for a 4×5 grid there are 20 grid points but 14 of them are boundary points (located at left, right, top, and bottom) which do not have evolution. There are only 6 middle points that have evolution in time. Thus, the coefficint matrix A is 20×20, and on 14 rows we have just one non-zero element (corresponding to the boundary condition).

According to equation 57, for a 4×5 grid, for example, at the boundary points the linear system becomes

$$\begin{aligned}
bottom: &\quad \phi_{1,1}^{n+1} = \phi_{1,2}^{n+1} = \phi_{1,3}^{n+1} = \phi_{1,4}^{n+1} = \phi_{1,5}^{n+1} = \phi_{bottom} && (36)\\
top: &\quad \phi_{4,1}^{n+1} = \phi_{4,2}^{n+1} = \phi_{4,3}^{n+1} = \phi_{4,4}^{n+1} = \phi_{4,5}^{n+1} = \phi_{top} && (37)\\
left: &\quad \phi_{2,1}^{n+1} = \phi_{3,1}^{n+1} = \phi_{left} && (38)\\
right: &\quad \phi_{2,5}^{n+1} = \phi_{3,5}^{n+1} = \phi_{right} && (39)\\
& && (40)
\end{aligned}$$

$$\begin{aligned}
grid\ point\ 2,2: &-s_z\phi_{3,2}^{n+1} - s_x\phi_{2,3}^{n+1} + (1+s_z+s_x)\phi_{2,2}^{n+1} - s_z\phi_{1,2}^{n+1} - s_x\phi_{2,1}^{n+1} = \phi_{2,2}^{n+1} && (41)\\
grid\ point\ 2,3: &-s_z\phi_{3,3}^{n+1} - s_x\phi_{2,4}^{n+1} + (1+s_z+s_x)\phi_{2,3}^{n+1} - s_z\phi_{1,3}^{n+1} - s_x\phi_{2,2}^{n+1} = \phi_{2,3}^{n+1} && (42)\\
grid\ point\ 2,4: &-s_z\phi_{3,5}^{n+1} - s_x\phi_{2,5}^{n+1} + (1+s_z+s_x)\phi_{2,4}^{n+1} - s_z\phi_{1,4}^{n+1} - s_x\phi_{2,3}^{n+1} = \phi_{2,4}^{n+1} && (43)\\
grid\ point\ 3,2: &-s_z\phi_{4,2}^{n+1} - s_x\phi_{3,3}^{n+1} + (1+s_z+s_x)\phi_{3,2}^{n+1} - s_z\phi_{2,2}^{n+1} - s_x\phi_{3,1}^{n+1} = \phi_{3,2}^{n+1} && (44)\\
grid\ point\ 3,3: &-s_z\phi_{4,3}^{n+1} - s_x\phi_{3,4}^{n+1} + (1+s_z+s_x)\phi_{3,3}^{n+1} - s_z\phi_{2,3}^{n+1} - s_x\phi_{3,2}^{n+1} = \phi_{3,3}^{n+1} && (45)\\
grid\ point\ 3,4: &-s_z\phi_{4,4}^{n+1} - s_x\phi_{3,5}^{n+1} + (1+s_z+s_x)\phi_{3,4}^{n+1} - s_z\phi_{2,4}^{n+1} - s_x\phi_{3,3}^{n+1} = \phi_{3,4}^{n+1} && (46)\\
& && (47)
\end{aligned}$$

and the linear system for the 6 other points located between the boundaries becomes 73.

As we explained previously the book-keeping issues, i.e. the mapping of $\phi_{i,j}$ to the vector Φ_k , vectors $\vec{x}$ and $\vec{b}$ become equation 74.

$$x = \begin{bmatrix} \phi_{1,1}^{n+1}\\ \phi_{1,2}^{n+1}\\ \phi_{1,3}^{n+1}\\ \phi_{1,4}^{n+1}\\ \phi_{1,5}^{n+1}\\ \phi_{2,1}^{n+1}\\ \phi_{2,2}^{n+1}\\ \phi_{2,3}^{n+1}\\ \phi_{2,4}^{n+1}\\ \phi_{2,5}^{n+1}\\ \phi_{3,1}^{n+1}\\ \phi_{3,2}^{n+1}\\ \phi_{3,3}^{n+1}\\ \phi_{3,4}^{n+1}\\ \phi_{3,5}^{n+1}\\ \phi_{4,1}^{n+1}\\ \phi_{4,2}^{n+1}\\ \phi_{4,3}^{n+1}\\ \phi_{4,4}^{n+1}\\ \phi_{4,5}^{n+1} \end{bmatrix} = \begin{bmatrix} \Phi_1^{n+1}\\ \Phi_2^{n+1}\\ \Phi_3^{n+1}\\ \Phi_4^{n+1}\\ \Phi_5^{n+1}\\ \Phi_6^{n+1}\\ \Phi_7^{n+1}\\ \Phi_8^{n+1}\\ \Phi_9^{n+1}\\ \Phi_{10}^{n+1}\\ \Phi_{11}^{n+1}\\ \Phi_{12}^{n+1}\\ \Phi_{13}^{n+1}\\ \Phi_{14}^{n+1}\\ \Phi_{15}^{n+1}\\ \Phi_{16}^{n+1}\\ \Phi_{17}^{n+1}\\ \Phi_{18}^{n+1}\\ \Phi_{19}^{n+1}\\ \Phi_{20}^{n+1} \end{bmatrix}, \qquad b = \begin{bmatrix} \phi_{1,1}^{n}\\ \phi_{1,2}^{n}\\ \phi_{1,3}^{n}\\ \phi_{1,4}^{n}\\ \phi_{1,5}^{n}\\ \phi_{2,1}^{n}\\ \phi_{2,2}^{n}\\ \phi_{2,3}^{n}\\ \phi_{2,4}^{n}\\ \phi_{2,5}^{n}\\ \phi_{3,1}^{n}\\ \phi_{3,2}^{n}\\ \phi_{3,3}^{n}\\ \phi_{3,4}^{n}\\ \phi_{3,5}^{n}\\ \phi_{4,1}^{n}\\ \phi_{4,2}^{n}\\ \phi_{4,3}^{n}\\ \phi_{4,4}^{n}\\ \phi_{4,5}^{n} \end{bmatrix} = \begin{bmatrix} \phi_{bottom}\\ \phi_{bottom}\\ \phi_{bottom}\\ \phi_{bottom}\\ \phi_{bottom}\\ \phi_{left}\\ \Phi_7^n\\ \Phi_8^n\\ \Phi_9^n\\ \phi_{right}\\ \phi_{left}\\ \Phi_{12}^n\\ \Phi_{13}^n\\ \Phi_{14}^n\\ \phi_{right}\\ \phi_{top}\\ \phi_{top}\\ \phi_{top}\\ \phi_{top}\\ \phi_{top} \end{bmatrix} \tag{48}$$

And the coefficient matrix, A, is 75.
Now two-dimentional heat equation (54) on a 5×4 grid reduced to a linear system $A\vec{x} = \vec{b}$, with $\vec{x}$, $\vec{b}$, and A are given by equations 74, 75.
This is the way to find the coefficient matrix, and notice that for different boundary conditions we should modify some elements in the coefficient matrix. You can easily understand the pattern in the coefficient matrix, it will help you to build it in your code.
Now that we know how to build the coefficient matrix for two-dimentional problem, it is the time to check what we have orgnized so far.

The coefficient matrix, A, for a 5×4 grid is

$$A = \left[\begin{array}{ccccc|ccccc|ccccc|ccccc}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -s_z & 0 & 0 & 0 & -s_x & (1+2s_z+2s_x) & -s_x & 0 & 0 & 0 & -s_z & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -s_z & 0 & 0 & 0 & -s_x & 1+2s_z+2s_x & -s_x & 0 & 0 & 0 & -s_z & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -s_z & 0 & 0 & 0 & -s_x & 1+2s_z+2s_x & -s_x & 0 & 0 & 0 & -s_z & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -s_z & 0 & 0 & 0 & -s_x & 1+2s_z+2s_x & -s_x & 0 & 0 & 0 & -s_z & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -s_z & 0 & 0 & 0 & -s_x & 1+2s_z+2s_x & -s_x & 0 & 0 & 0 & -s_z & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -s_z & 0 & 0 & 0 & -s_x & 1+2s_z+2s_x & -s_x & 0 & 0 & 0 & -s_z & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{array}\right] \tag{49}$$

And just for comparing, the coefficient matrix, A, for a 4×5 grid is

$$A = \left[\begin{array}{cccc|cccc|cccc|cccc|cccc}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -s_z & 0 & 0 & -s_x & 1+2s_z+2s_x & -s_x & 0 & 0 & -s_z & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -s_z & 0 & 0 & -s_x & 1+2s_z+2s_x & -s_x & 0 & 0 & -s_z & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -s_z & 0 & 0 & -s_x & 1+2s_z+2s_x & -s_x & 0 & 0 & -s_z & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -s_z & 0 & 0 & -s_x & 1+2s_z+2s_x & -s_x & 0 & 0 & -s_z & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -s_z & 0 & 0 & -s_x & 1+2s_z+2s_x & -s_x & 0 & 0 & -s_z & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -s_z & 0 & 0 & -s_x & 1+2s_z+2s_x & -s_x & 0 & 0 & -s_z & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{array}\right] \tag{50}$$

Like what we have done for the one-dimentional part, here we can ignore the boundary points in the solution vector (all ϕ_{top}, ϕ_{bottom}, ϕ_{left}, and $\phi right$) and just solve the equation for the evolving middle ponits. Like befor we consider the boundary conditions in a boundary condition vector ($\vec{b.c.}$). For two dimensional case, we can rewrite the linear system as:

$$A'\vec{x'} = \vec{b} + \vec{b.c.} \tag{51}$$

where vectors $\vec{x'}$, $\vec{b'}$, and $\vec{b.c.}$ are:

$$\vec{x'} = \begin{bmatrix} \Phi_7^{n+1} \\ \Phi_8^{n+1} \\ \Phi_9^{n+1} \\ \Phi_{12}^{n+1} \\ \Phi_{13}^{n+1} \\ \Phi_{14}^{n+1} \end{bmatrix} \quad \vec{b'} = \begin{bmatrix} \Phi_7^{n} \\ \Phi_8^{n} \\ \Phi_9^{n} \\ \Phi_{12}^{n} \\ \Phi_{13}^{n} \\ \Phi_{14}^{n} \end{bmatrix} \quad \vec{b.c.} = \begin{bmatrix} +s_z\phi_{bottom} + s_x\phi_{left} \\ +s_z\phi_{bottom} \\ +s_z\phi_{bottom} + s_x\phi_{right} \\ +s_z\phi_{top} + s_x\phi_{left} \\ +s_z\phi_{top} \\ +s_z\phi_{top} + s_x\phi_{right} \end{bmatrix} \tag{52}$$

and coefficient matrix A' is 79

$$A' = \begin{bmatrix} (1+2s_z+2s_x) & -s_x & 0 & -s_z & 0 & 0 \\ -s_x & (1+2s_z+2s_x) & -s_x & 0 & -s_z & 0 \\ 0 & -s_x & (1+2s_z+2s_x) & -s_x & 0 & -s_z \\ -s_z & 0 & -s_x & (1+2s_z+2s_x) & -s_x & 0 \\ 0 & -s_z & 0 & -s_x & (1+2s_z+2s_x) & -s_x \\ 0 & 0 & -s_z & 0 & -s_x & (1+2s_z+2s_x) \end{bmatrix} \tag{53}$$

7 Two-dimensional diffusion equation

Now we consider two-dimentional diffusion equation; if the diffusion coefficient (D) is costant, then the two-dimensional diffusion equation reduces to the following linear equation:

$$\frac{\partial \phi(\vec{r},t)}{\partial t} = D\nabla^2\phi(\vec{r},t) \tag{54}$$

and if we assume that $\vec{r} = (x, z)$ it becomes to

$$\frac{\partial \phi(x,z,t)}{\partial t} = D\left(\frac{\partial^2\phi(x,z,t)}{\partial x^2} + \frac{\partial^2\phi(x,z,t)}{\partial z^2}\right). \tag{55}$$

According to the previous explanation on Taylor-series expansion and Linearization of PDEs, if we employ a fully implicit, unconditionally stable discretization scheme as we have done for the 1D equation , equation 55 can be written as

$$\frac{\phi_{i,j}^{n+1} - \phi_{i,j}^{n}}{\Delta t} = D\left(\frac{\phi_{i,j+1}^{n+1} - 2\phi_{i,j}^{n+1} + \phi_{i,j-1}^{n+1}}{(\Delta x)^2} + \frac{\phi_{i+1,j}^{n+1} - 2\phi_{i,j}^{n+1} + \phi_{i-1,j}^{n+1}}{(\Delta z)^2}\right) \tag{56}$$

Rearranging terms with $n+1$ on one side and terms with n on the other side gives

$$\phi_{i,j}^{n} = -s_z\phi_{i+1,j}^{n+1} - s_x\phi_{i,j+1}^{n+1} + (1+2s_z+2s_x)\phi_{i,j}^{n+1} - s_z\phi_{i-1,j}^{n+1} - s_x\phi_{i,j-1}^{n+1} \tag{57}$$

As in the 1D case, we have to write these equations in a matrix A and a vector $\vec{b}$. From a practical point of view, this is a bit more complicated than in the 1D case, since we have to deal with book-keeping issues, i.e. the mapping of $\phi_{i,j}$ to the entries of a vector Φ_k.
If a 2D scalar field is to be solved for with an equivalent vector Φ, the nodes have to be numbered continuously, for example as in Figure 8. The derivative versus x-direction is then e.g.

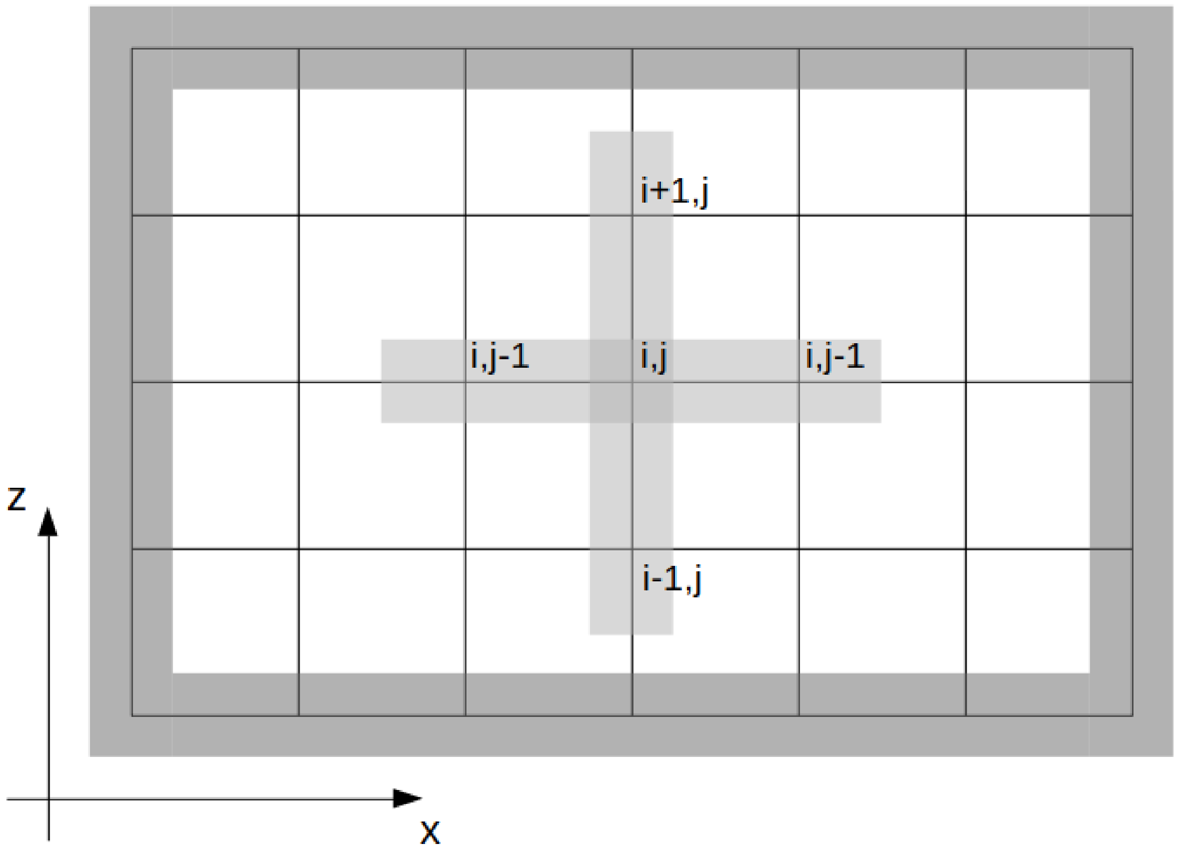


Figure 7: Finite difference discretization of the 2D heat problem

$$\frac{\partial^2 \phi}{\partial x^2}|_{i=3,j=3} = \frac{1}{(\Delta x)^2}(\Phi_{14} - 2\Phi_{13} + \Phi_{12}) \tag{58}$$

and the derivative versus z-direction is given by

$$\frac{\partial^2 \phi}{\partial z^2}|_{i=3,j=3} = \frac{1}{(\Delta z)^2}(\Phi_{18} - 2\Phi_{13} + \Phi_{8}) \tag{59}$$

If n_x are the number of grid points in x-direction and n_z the number of points in z-direction, we can write equations 58 and 59 in a more general way as:

$$\frac{\partial^2 \phi}{\partial x^2}|_{i,j} = \frac{1}{(\Delta x)^2}(\phi_{(i-1)n_x+j+1} - 2\phi_{(i-1)n_x+j} + \phi_{(i-1)n_x+j-1}) \tag{60}$$

$$\frac{\partial^2 \phi}{\partial z^2}|_{i,j} = \frac{1}{(\Delta z)^2}(\phi_{in_x+j} - 2\phi_{(i-1)n_x+j} + \phi_{(i-2)n_x+j}) \tag{61}$$

As an example, for a 4×5 grid there are 20 grid points but 14 of them are boundary points (located at left, right, top, and bottom) which do not have evolution. There are only 6 middle points that have evolution in time. Thus, the coefficint matrix A is 20×20, and on 14 rows we have just one non-zero element (corresponding to the boundary condition).

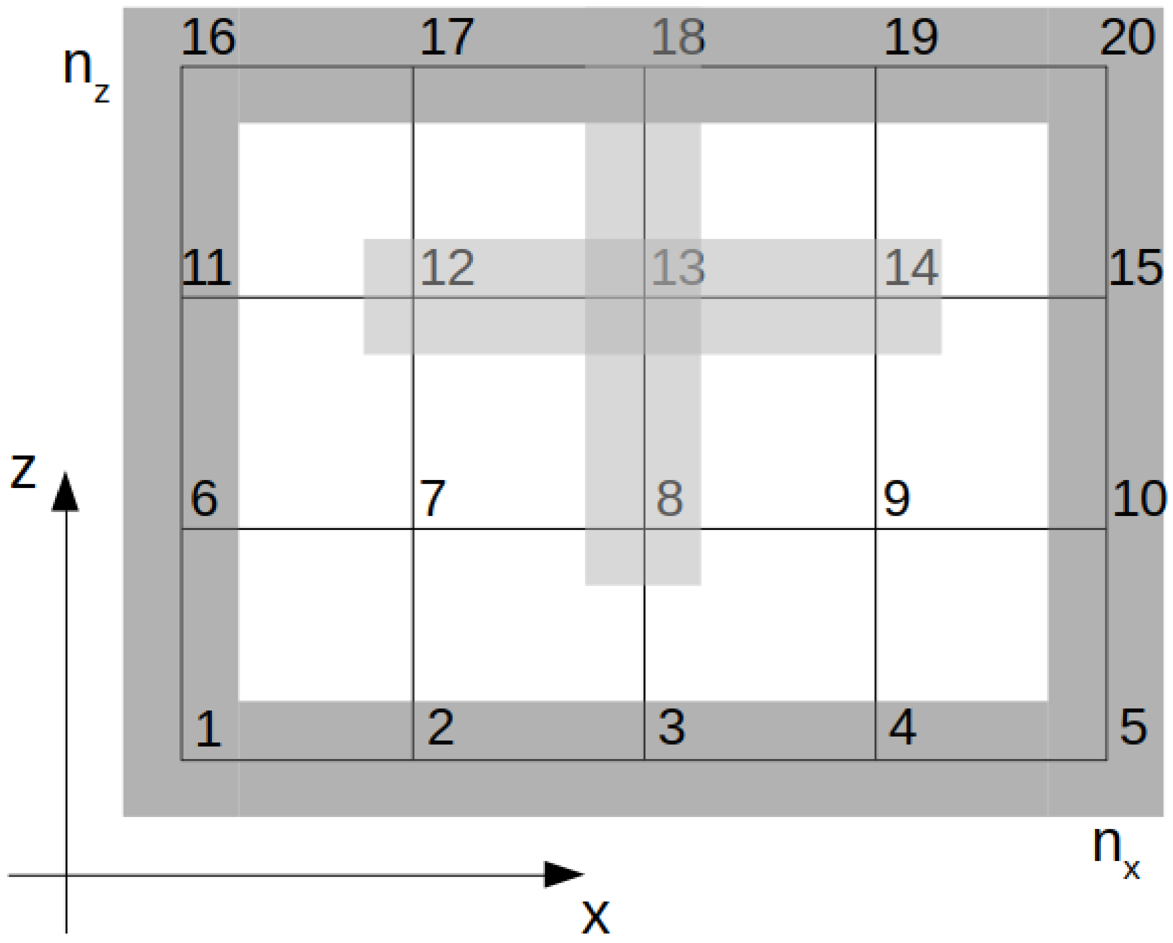


Figure 8: Numbering scheme for a 2D grid with $n_x = 5$ and $n_z = 4$.

According to equation 57, for a 4×5 grid, for example, at the boundary points the linear system becomes

$$\begin{aligned}
bottom: &\quad \phi_{1,1}^{n+1} = \phi_{1,2}^{n+1} = \phi_{1,3}^{n+1} = \phi_{1,4}^{n+1} = \phi_{1,5}^{n+1} = \phi_{bottom} && (62)\\
top: &\quad \phi_{4,1}^{n+1} = \phi_{4,2}^{n+1} = \phi_{4,3}^{n+1} = \phi_{4,4}^{n+1} = \phi_{4,5}^{n+1} = \phi_{top} && (63)\\
left: &\quad \phi_{2,1}^{n+1} = \phi_{3,1}^{n+1} = \phi_{left} && (64)\\
right: &\quad \phi_{2,5}^{n+1} = \phi_{3,5}^{n+1} = \phi_{right} && (65)\\
& && (66)
\end{aligned}$$

$$\begin{aligned}
grid\ point\ 2,2: -s_z\phi_{3,2}^{n+1} - s_x\phi_{2,3}^{n+1} + (1+s_z+s_x)\phi_{2,2}^{n+1} - s_z\phi_{1,2}^{n+1} - s_x\phi_{2,1}^{n+1} &= \phi_{2,2}^{n+1} && (67)\\
grid\ point\ 2,3: -s_z\phi_{3,3}^{n+1} - s_x\phi_{2,4}^{n+1} + (1+s_z+s_x)\phi_{2,3}^{n+1} - s_z\phi_{1,3}^{n+1} - s_x\phi_{2,2}^{n+1} &= \phi_{2,3}^{n+1} && (68)\\
grid\ point\ 2,4: -s_z\phi_{3,5}^{n+1} - s_x\phi_{2,5}^{n+1} + (1+s_z+s_x)\phi_{2,4}^{n+1} - s_z\phi_{1,4}^{n+1} - s_x\phi_{2,3}^{n+1} &= \phi_{2,4}^{n+1} && (69)\\
grid\ point\ 3,2: -s_z\phi_{4,2}^{n+1} - s_x\phi_{3,3}^{n+1} + (1+s_z+s_x)\phi_{3,2}^{n+1} - s_z\phi_{2,2}^{n+1} - s_x\phi_{3,1}^{n+1} &= \phi_{3,2}^{n+1} && (70)\\
grid\ point\ 3,3: -s_z\phi_{4,3}^{n+1} - s_x\phi_{3,4}^{n+1} + (1+s_z+s_x)\phi_{3,3}^{n+1} - s_z\phi_{2,3}^{n+1} - s_x\phi_{3,2}^{n+1} &= \phi_{3,3}^{n+1} && (71)\\
grid\ point\ 3,4: -s_z\phi_{4,4}^{n+1} - s_x\phi_{3,5}^{n+1} + (1+s_z+s_x)\phi_{3,4}^{n+1} - s_z\phi_{2,4}^{n+1} - s_x\phi_{3,3}^{n+1} &= \phi_{3,4}^{n+1} && (72)\\
& && (73)
\end{aligned}$$

and the linear system for the 6 other points located between the boundaries becomes 73.

As we explained previously the book-keeping issues, i.e. the mapping of $\phi_{i,j}$ to the vector Φ_k , vectors $\vec{x}$ and $\vec{b}$ become equation 74.

$$x = \begin{bmatrix} \phi^{n+1}_{1,1} \\ \phi^{n+1}_{1,2} \\ \phi^{n+1}_{1,3} \\ \phi^{n+1}_{1,4} \\ \phi^{n+1}_{1,5} \\ \phi^{n+1}_{2,1} \\ \phi^{n+1}_{2,2} \\ \phi^{n+1}_{2,3} \\ \phi^{n+1}_{2,4} \\ \phi^{n+1}_{2,5} \\ \phi^{n+1}_{3,1} \\ \phi^{n+1}_{3,2} \\ \phi^{n+1}_{3,3} \\ \phi^{n+1}_{3,4} \\ \phi^{n+1}_{3,5} \\ \phi^{n+1}_{4,1} \\ \phi^{n+1}_{4,2} \\ \phi^{n+1}_{4,3} \\ \phi^{n+1}_{4,4} \\ \phi^{n+1}_{4,5} \end{bmatrix} = \begin{bmatrix} \Phi^{n+1}_{1} \\ \Phi^{n+1}_{2} \\ \Phi^{n+1}_{3} \\ \Phi^{n+1}_{4} \\ \Phi^{n+1}_{5} \\ \Phi^{n+1}_{6} \\ \Phi^{n+1}_{7} \\ \Phi^{n+1}_{8} \\ \Phi^{n+1}_{9} \\ \Phi^{n+1}_{10} \\ \Phi^{n+1}_{11} \\ \Phi^{n+1}_{12} \\ \Phi^{n+1}_{13} \\ \Phi^{n+1}_{14} \\ \Phi^{n+1}_{15} \\ \Phi^{n+1}_{16} \\ \Phi^{n+1}_{17} \\ \Phi^{n+1}_{18} \\ \Phi^{n+1}_{19} \\ \Phi^{n+1}_{20} \end{bmatrix}, \qquad b = \begin{bmatrix} \phi^{n}_{1,1} \\ \phi^{n}_{1,2} \\ \phi^{n}_{1,3} \\ \phi^{n}_{1,4} \\ \phi^{n}_{1,5} \\ \phi^{n}_{2,1} \\ \phi^{n}_{2,2} \\ \phi^{n}_{2,3} \\ \phi^{n}_{2,4} \\ \phi^{n}_{2,5} \\ \phi^{n}_{3,1} \\ \phi^{n}_{3,2} \\ \phi^{n}_{3,3} \\ \phi^{n}_{3,4} \\ \phi^{n}_{3,5} \\ \phi^{n}_{4,1} \\ \phi^{n}_{4,2} \\ \phi^{n}_{4,3} \\ \phi^{n}_{4,4} \\ \phi^{n}_{4,5} \end{bmatrix} = \begin{bmatrix} \phi_{bottom} \\ \phi_{bottom} \\ \phi_{bottom} \\ \phi_{bottom} \\ \phi_{bottom} \\ \phi_{left} \\ \Phi^{n}_{7} \\ \Phi^{n}_{8} \\ \Phi^{n}_{9} \\ \phi_{right} \\ \phi_{left} \\ \Phi^{n}_{12} \\ \Phi^{n}_{13} \\ \Phi^{n}_{14} \\ \phi_{right} \\ \phi_{top} \\ \phi_{top} \\ \phi_{top} \\ \phi_{top} \\ \phi_{top} \end{bmatrix} \tag{74}$$

And the coefficient matrix, A, is 75.
Now two-dimentional heat equation (54) on a 5×4 grid reduced to a linear system $A\vec{x} = \vec{b}$, with $\vec{x}$, $\vec{b}$, and A are given by equations 74, 75.
This is the way to find the coefficient matrix, and notice that for different boundary conditions we should modify some elements in the coefficient matrix. You can easily understand the pattern in the coefficient matrix, it will help you to build it in your code.
Now that we know how to build the coefficient matrix for two-dimentional problem, it is the time to check what we have orgnized so far.

The coefficient matrix, A, for a 5×4 grid is

$$A = \left[\begin{array}{ccccc|ccccc|ccccc|ccccc}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -s_z & 0 & 0 & 0 & -s_x & (1+2s_z+2s_x) & -s_x & 0 & 0 & 0 & -s_z & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -s_z & 0 & 0 & 0 & -s_x & 1+2s_z+2s_x & -s_x & 0 & 0 & 0 & -s_z & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -s_z & 0 & 0 & 0 & -s_x & 1+2s_z+2s_x & -s_x & 0 & 0 & 0 & -s_z & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -s_z & 0 & 0 & 0 & -s_x & 1+2s_z+2s_x & -s_x & 0 & 0 & 0 & -s_z & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -s_z & 0 & 0 & 0 & -s_x & 1+2s_z+2s_x & -s_x & 0 & 0 & 0 & -s_z & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -s_z & 0 & 0 & 0 & -s_x & 1+2s_z+2s_x & -s_x & 0 & 0 & 0 & -s_z & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{array}\right] \tag{75}$$

The coefficient matrix, A, for a 4×5 grid is

$$A = \left[\begin{array}{cccc|cccc|cccc|cccc|cccc}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -s_z & 0 & 0 & -s_x & 1+2s_z+2s_x & -s_x & 0 & 0 & -s_z & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -s_z & 0 & 0 & -s_x & 1+2s_z+2s_x & -s_x & 0 & 0 & -s_z & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -s_z & 0 & 0 & -s_x & 1+2s_z+2s_x & -s_x & 0 & 0 & -s_z & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -s_z & 0 & 0 & -s_x & 1+2s_z+2s_x & -s_x & 0 & 0 & -s_z & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -s_z & 0 & 0 & -s_x & 1+2s_z+2s_x & -s_x & 0 & 0 & -s_z & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -s_z & 0 & 0 & -s_x & 1+2s_z+2s_x & -s_x & 0 & 0 & -s_z & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{array}\right] \tag{76}$$

Like what we have done for the one-dimentional part, here we can ignore the boundary points in the solution vector (all ϕ_{top}, ϕ_{bottom}, ϕ_{left}, and $\phi right$) and just solve the equation for the evolving middle ponits. Like befor we consider the boundary conditions in a boundary condition vector ($\vec{b.c.}$). For two dimensional case, we can rewrite the linear system as:

$$A'\vec{x'} = \vec{b} + \vec{b.c.} \tag{77}$$

where vectors $\vec{x'}$, $\vec{b'}$, and $\vec{b.c.}$ are:

$$\vec{x'} = \begin{bmatrix} \Phi_7^{n+1} \\ \Phi_8^{n+1} \\ \Phi_9^{n+1} \\ \Phi_{12}^{n+1} \\ \Phi_{13}^{n+1} \\ \Phi_{14}^{n+1} \end{bmatrix} \quad \vec{b'} = \begin{bmatrix} \Phi_7^{n} \\ \Phi_8^{n} \\ \Phi_9^{n} \\ \Phi_{12}^{n} \\ \Phi_{13}^{n} \\ \Phi_{14}^{n} \end{bmatrix} \quad \vec{b.c.} = \begin{bmatrix} +s_z\phi_{bottom} + s_x\phi_{left} \\ +s_z\phi_{bottom} \\ +s_z\phi_{bottom} + s_x\phi_{right} \\ +s_z\phi_{top} + s_x\phi_{left} \\ +s_z\phi_{top} \\ +s_z\phi_{top} + s_x\phi_{right} \end{bmatrix} \tag{78}$$

and coefficient matrix A' is 79

$$A' = \begin{bmatrix} (1+2s_z+2s_x) & -s_x & 0 & -s_z & 0 & 0 \\ -s_x & (1+2s_z+2s_x) & -s_x & 0 & -s_z & 0 \\ 0 & -s_x & (1+2s_z+2s_x) & -s_x & 0 & -s_z \\ -s_z & 0 & -s_x & (1+2s_z+2s_x) & -s_x & 0 \\ 0 & -s_z & 0 & -s_x & (1+2s_z+2s_x) & -s_x \\ 0 & 0 & -s_z & 0 & -s_x & (1+2s_z+2s_x) \end{bmatrix} \tag{79}$$

8 Results for two-dimensional heat equation

8.1 The case of constant diffusion coefficient

In the previous sections we have found that the CG algorithm are much more efficient than the direct methods such as LU decomposition, so all the simulation for the two-dimentional equation have been done using CG method.

After 10 steps we will have figure 10b. There is also an animation accessible at: `http://www.github.com/`.

In this case (constant diffusion coefficient), it is clear that the diffusion process is symmetric.

8.2 The case of random diffusion coefficient

The initial conditions of this case is the same as the initial condition for the constant diffusion coefficient (figure 9).

In contrast with the case of the constant diffusion coefficient, because of those random points which will not diffuse ($D(\vec{r}^*) = 0$), here the distribution of ϕ is not symmetric.
insert the plots for simple 2d and then the assignment here

Appendix

As computer architecture advanced, it became more difficult to compare the performance of various computer systems simply by looking at their specifications. Therefore, tests were developed that allowed comparison of different architectures. For example, Pentium 4 processors generally operated at a higher

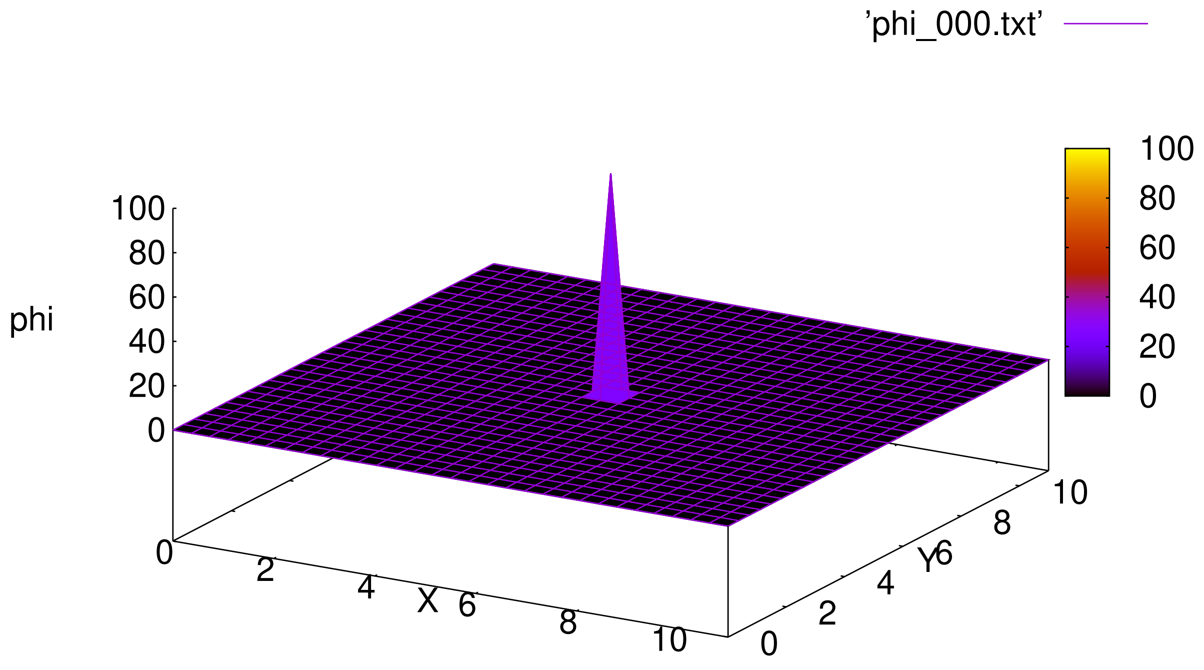


Figure 9: initial condition for the 2D heat problem

clock frequency than Athlon XP or PowerPC processors, which did not necessarily translate to more computational power; a processor with a slower clock frequency might perform as well as or even better than a processor operating at a higher frequency [3].
Benchmarks are designed to mimic a particular type of workload on a component or system. Benchmarking is usually associated with assessing performance characteristics of computer hardware, for example, the floating point operation performance of a CPU.

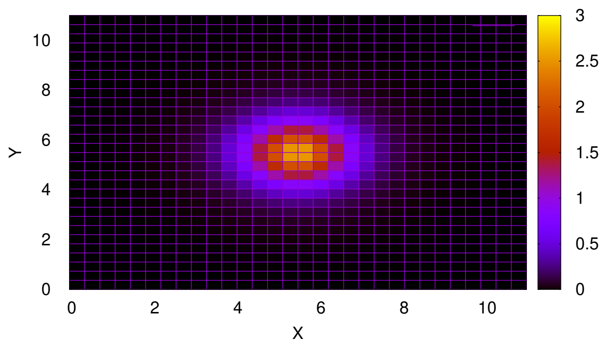
PARKBENCH

The PARKBENCH (PARallel Kernels and BENCHmarks) committee, originally called the Parallel Benchmark Working Group (PBWG) was founded at Supercomputing '92 in Minneapolis, when a group of about 50 people interested in computer benchmarking met under the joint initiative of Tony Hey and Jack Dongarra, and the chairmanship of Roger Hockney. The objectives of the PARKBENCH group are;

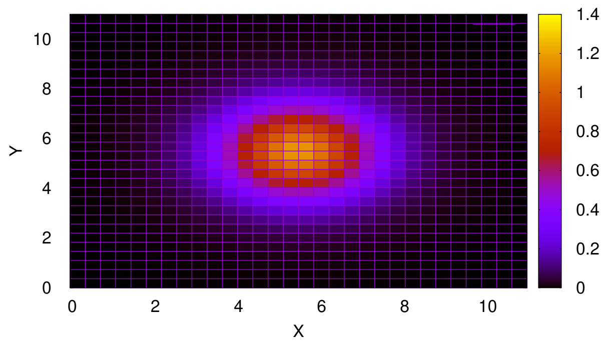
- To establish a comprehensive set of parallel benchmarks that is generally accepted by both users and vendors of parallel systems.
- To provide a focus for parallel benchmark activities and avoid unnecessary duplication of effort and proliferation of benchmarks.
- To set standards for benchmarking methodology and result-reporting together with a control database/repository for both the benchmarks and the results.
- To make the benchmarks and results freely available in the public domain.

Further information on PARKBENCH may be obtained at:
`http://www.netlib.org/parkbench` .

[3]The megahertz myth, or less commonly the gigahertz myth, refers to the misconception of only using clock rate (for example measured in megahertz or gigahertz) to compare the performance of different microprocessors.

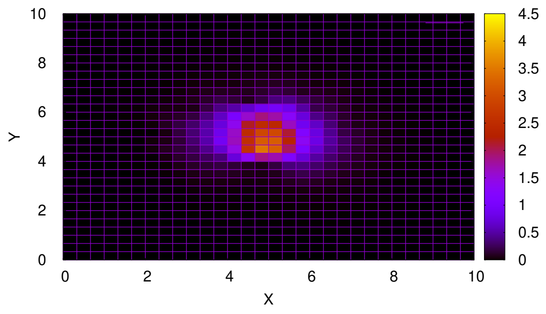


(a) the heat distribution map after 5 timesteps for the 2D heat problem

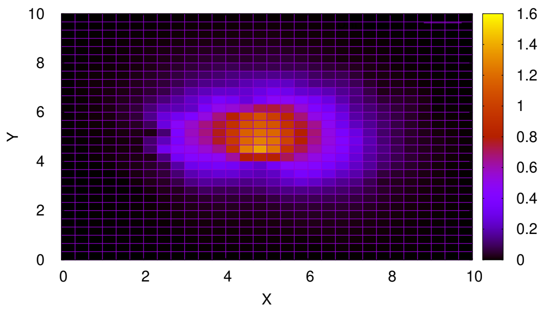


(b) the heat distribution map after 10 timesteps for the 2D heat problem

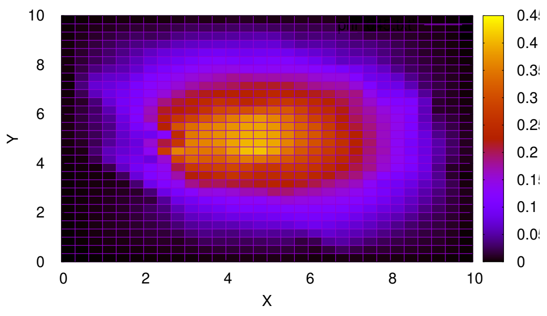
Figure 10: solving two-dimensional heat equation using CG method.



(a) the heat distribution map after 5 timesteps for the 2D heat problem



(b) the heat distribution map after 15 timesteps for the 2D heat problem



(c) the heat distribution map after 45 timesteps for the 2D heat problem

Figure 11: solving two-dimensional heat equation using CG method.

NAS Parallel Benchmarks

NAS Parallel Benchmarks (NPB) are a set of benchmarks targeting performance evaluation of highly parallel supercomputers. The NAS Parallel Benchmarks (NPB) are a small set of programs designed to help evaluate the performance of parallel supercomputers. They are developed and maintained by the NASA Advanced Supercomputing (NAS) Division (formerly the NASA Numerical Aerodynamic Simulation Program) based at the NASA Ames Research Center. The benchmarks are derived from computational fluid dynamics (CFD) applications and consist of five kernels and three pseudo-applications. The benchmark suite has been extended to include new benchmarks for unstructured adaptive meshes, parallel I/O, multi-zone applications, and computational grids.
The original eight benchmarks specified in NPB 1 mimic the computation and data movement in CFD applications:

- IS - Integer Sort, random memory access
- EP - Embarrassingly Parallel
- CG - Conjugate Gradient, irregular memory access and communication
- MG - Multi-Grid on a sequence of meshes, long- and short-distance communication, memory intensive
- FT - discrete 3D fast Fourier Transform, all-to-all communication
- BT - Block Tri-diagonal solver
- SP - Scalar Penta-diagonal solver
- LU - Lower-Upper Gauss-Seidel solver

and there are several other benchmarks for unstructured computation, parallel I/O, and data movement. As of NPB 3.3, eleven benchmarks are defined. Further information on NAS may be obtained at:
`www.nas.nasa.gov/publications/npb.html` .

References

[1] Richard Barrett, Michael W Berry, Tony F Chan, James Demmel, June Donato, Jack Dongarra, Victor Eijkhout, Roldan Pozo, Charles Romine, and Henk Van der Vorst. *Templates for the solution of linear systems: building blocks for iterative methods*, volume 43. Siam, 1994.

[2] Gang Cheng, Kenneth A Hawick, Gerald Mortensen, and Geoffrey C Fox. Distributed computational electromagnetics systems. In *PPSC*, pages 231–236, 1995.

[3] Iain S Duff, Albert Maurice Erisman, and John Ker Reid. *Direct methods for sparse matrices.* Oxford University Press, 2017.

[4] E Bogusz, G Fox, T Haupt, K Hawick, and S Ranka. Preliminary evaluation of high-performance fortran as a language for computational fluid dynamics. In *Fluid Dynamics Conference*, page 2262, 1994.

[5] R Hockney. Parkbench report: Public international benchmarks for parallel computers. *Scientific Programming*, 3(2):101–146, 1994.

[6] David Bailey, Tim Harris, William Saphir, Rob Van Der Wijngaart, Alex Woo, and Maurice Yarrow. The nas parallel benchmarks 2.0. Technical report, Technical Report NAS-95-020, NASA Ames Research Center, 1995.

[7] Grégoire Allaire and Sidi Mahmoud Kaber. *Numerical linear algebra*, volume 55. Springer, 2008.

[8] Jack J Dongarra, Iain S Duff, Danny C Sorensen, and Henk A Van der Vorst. *Solving linear systems on vector and shared memory computers.* Siam Philadelphia., 1991.

[9] KA Hawick, Kivanc Dincer, Guy Robinson, and GC Fox. Conjugate gradient algorithms in fortran 90 and high performance fortran. Technical report, NPAC Technical Report SCCS-691, 1995.