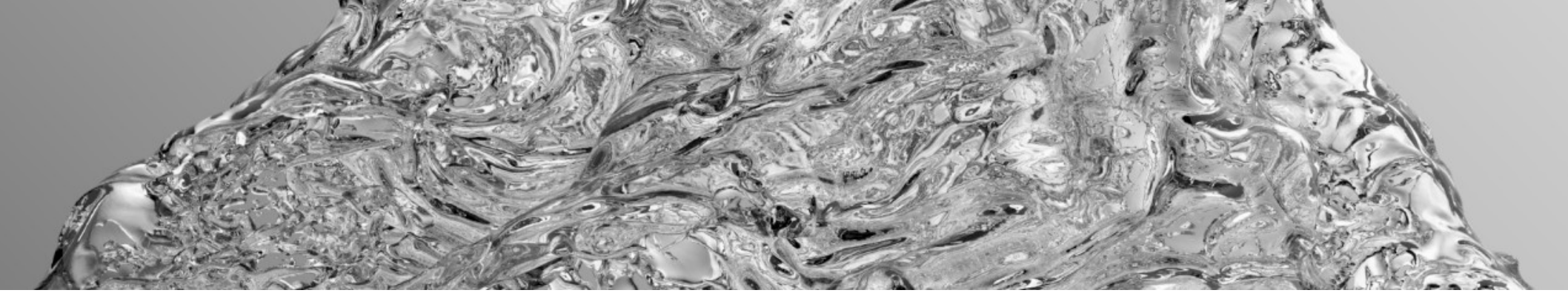


EEN-E2001 Computational Fluid Dynamics

Lecture 6: Matrices, Ax=b and final assignment

Prof. Ville Vuorinen

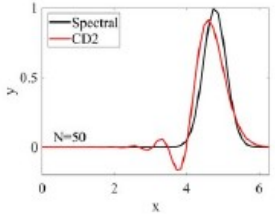
February 27th 2023

Aalto University, School of Engineering

Lecture 1: Linear PDEs and finite difference method

$$\frac{\partial T}{\partial t}+\nabla\cdot T\boldsymbol{u}=\alpha\nabla^2 T$$

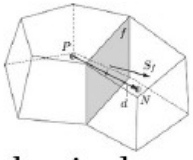
$$\frac{\partial T}{\partial x}\approx\frac{T_{i+1}-T_{i-1}}{2\Delta x}$$



Lecture 2: Gauss' theorem and finite volume method

$$\int_\Omega \nabla\cdot(T\boldsymbol{u})\,d\Omega=\int_{\partial\Omega}(T\boldsymbol{u})\cdot\boldsymbol{n}\,dS$$

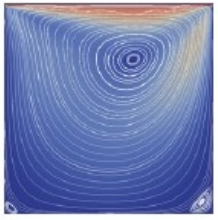
$$\int_{\partial\Omega}\boldsymbol{u}\cdot\boldsymbol{n}\,dS\approx\Sigma_f\boldsymbol{u}_f\cdot\boldsymbol{n}_f\,dS_f$$



Lecture 3: Navier-Stokes equation and pressure

$$\frac{\partial\boldsymbol{u}}{\partial t}+\nabla\cdot\boldsymbol{uu}=-\nabla p+\nu\nabla^2\boldsymbol{u}$$

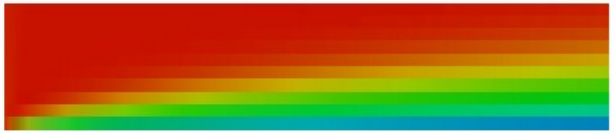
$$-\nabla^2 p=\nabla\cdot\nabla\cdot\boldsymbol{uu}$$

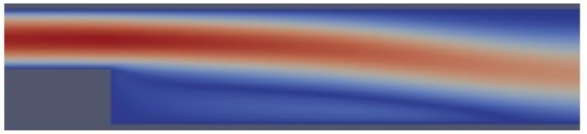


Lecture 4: OpenFOAM code and structure

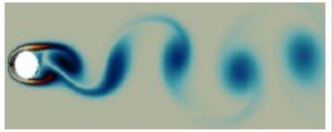
```
fvVectorMatrix UEqn
(
      fvm::ddt(U)
    + fvm::div(phi, U)
    - fvm::laplacian(nu, U)
);
```

Lecture 5: Simulating fluid physical phenomena: part A





Lecture 6: Simulating fluid physical phenomena: part B



Intended learning objectives of the lecture

After the lecture the student:

- Understands how to get started and solve the final assignment.

- Can explain what is a discretization matrix and how to solve 1d Poisson equation with matrices.

CFD simulation and PDE solution includes at least the following aspects covered on the course

1) **Physics** identification.

2) **Mathematical equations and physics interpretation.** Boundary/initial conditions.

3) **Objectives, feasibility, and time-constraints**.

4) **Numerical method and modeling assumptions**.

5) **Geometry and mesh generation**.

6) **Computing** i.e. running simulation.

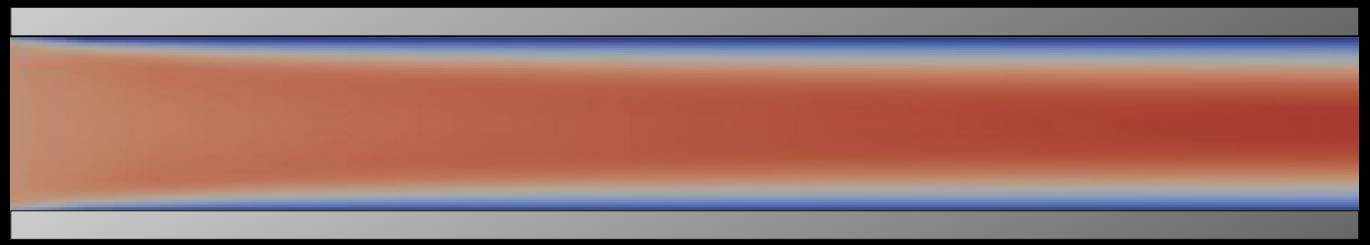
7) **Visualization and post-processing**.

8) **Validation and verification, reference data**. Reporting, analysis and discussion of the results. Are the results sane?

Connection between viscous and thermal boundary layers (TBL relevance: final assignment)

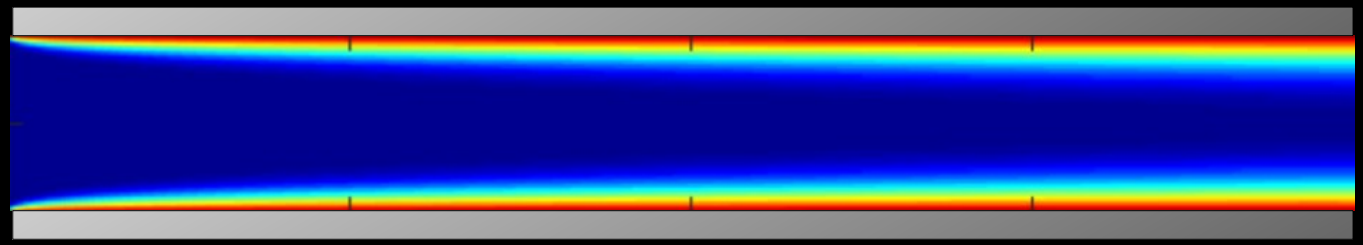
Velocity field in a laminar channel flow and viscous BL:

Key physics: viscous diffusion (viscosity) builds up the BL structure



Temperature field and thermal BL:

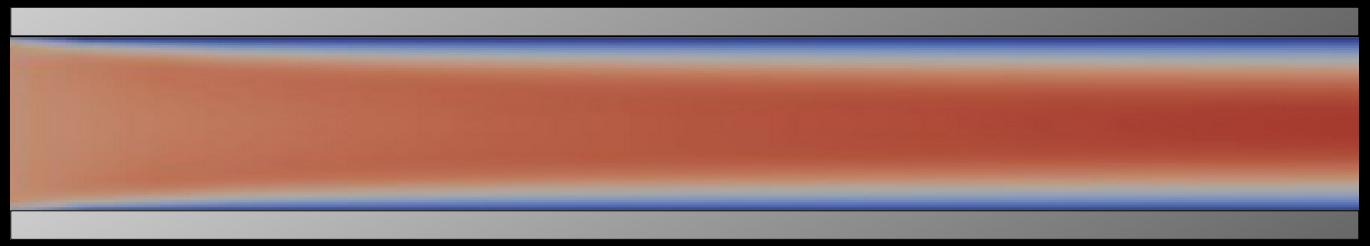
Key physics: thermal diffusion builds up the BL structure



Connection between viscous and thermal boundary layers (TBL relevance: final assignment)

Velocity field in a laminar channel flow and viscous BL:

Key physics: viscous diffusion (viscosity) builds up the BL structure



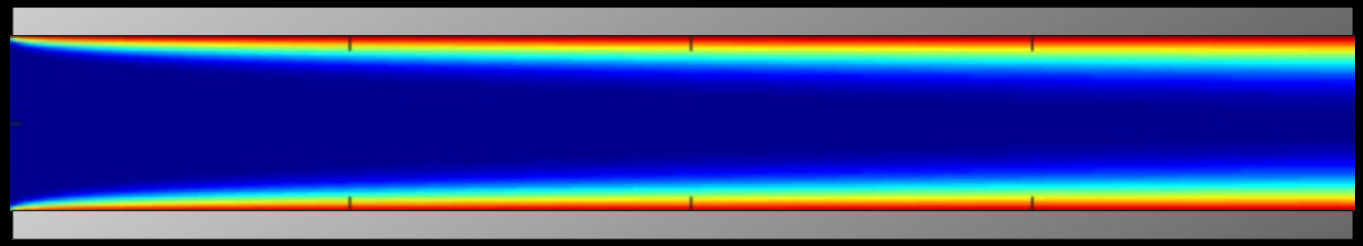
$$\frac{\partial u_i}{\partial t} + \vec{u} \cdot \nabla u_i = -\nabla p + \nu \nabla^2 u_i \quad \rightarrow \quad \frac{\partial^2 u_1}{\partial y^2} = \frac{\partial p}{\partial x}$$

Assumption: in steady state, the fully developed flow becomes 1d $\rightarrow \mathbf{u}=(u_1(y), 0)$

$$\frac{\partial T}{\partial t} + \vec{u} \cdot \nabla T = \alpha \nabla^2 T \quad \rightarrow \quad \alpha \frac{\partial^2 T}{\partial x^2} + \alpha \frac{\partial^2 T}{\partial y^2} = u_1 \frac{\partial T}{\partial x}$$

Temperature field and thermal BL:

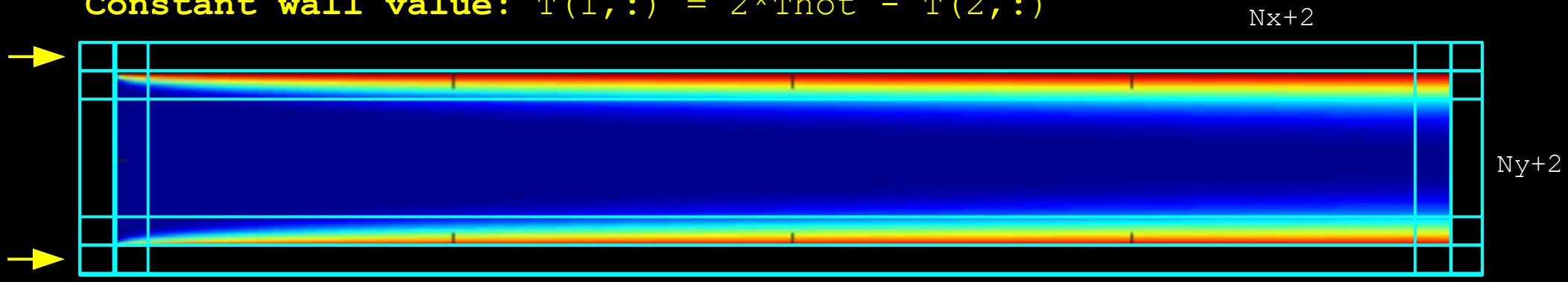
Key physics: thermal diffusion builds up the BL structure



In the final assignment, temperature field is solved in a 2d channel assuming velocity = constant (slip walls).

Below: the idea of giving BCs and using ghost cells shown.

Constant wall value: `T(1,:) = 2*Thot - T(2,:)`



Constant wall value: `T(Ny+2,:) = 2*Thot - T(Ny+1,:)`

Constant inlet value:
`T(:,1) = 2*Tcold - T(:,2)`

Zero-gradient outlet:
`T(:,Nx+2) = T(:,Nx+1)`

→ Temperature is stored into matrix of size (N_y+2) x (N_x+2)

→ Initialization of such a matrix in Matlab: `T = zeros(Ny+2,Nx+2)`

→ The boundary conditions are stored in 1st and last columns as well as 1st and last rows. Those "extra rows/cols" are called often "ghost cells".

→ To update temperature at cell centers: `T = T + dT;`

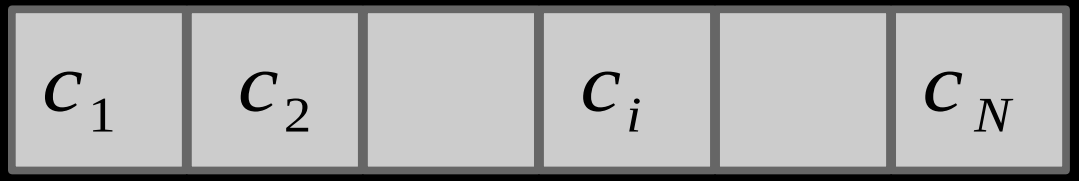
→ **Important:** wall is not at cell center but dy/2 from cell center.

Numerical computation of partial derivatives of a function using matrices

Consider CD2 method for estimation of partial derivatives

$$\frac{\partial c}{\partial x} \approx \frac{c_{i+1}^n - c_{i-1}^n}{2\,\Delta x}$$

$$\frac{\partial^2 c}{\partial x^2} \approx \frac{c_{i+1}^n - 2\,c_i^n + c_{i-1}^n}{\Delta x^2}$$



Where is a vector ?

Where is a matrix ?

Computers understand numeric data commonly as scalars, vectors, or matrices

N by N differentiation matrix: 1st derivative

$$D_x = \frac{1}{2\Delta x}\begin{pmatrix} 0 & 1 & 0 & \cdots & -1 \\ -1 & 0 & 1 & 0 & \cdots \\ 0 & -1 & 0 & 1 & \cdots \\ \vdots & & & & \vdots \\ 1 & 0 & \cdots & -1 & 0 \end{pmatrix}$$

N by N differentiation matrix: 2nd derivative

$$D_{xx} = \frac{1}{\Delta x^2}\begin{pmatrix} -2 & 1 & 0 & \cdots & 1 \\ 1 & -2 & 1 & 0 & \cdots \\ 0 & 1 & -2 & 1 & \cdots \\ \vdots & & & & \vdots \\ 1 & 0 & \cdots & 1 & -2 \end{pmatrix}$$

Computers understand numeric data commonly as scalars, vectors, or matrices

N by N differentiation matrix: 1st derivative

$$D_x = \frac{1}{2\Delta x}\begin{pmatrix} 0 & 1 & 0 & \cdots & -1 \\ -1 & 0 & 1 & 0 & \cdots \\ 0 & -1 & 0 & 1 & \cdots \\ \vdots & & & & \vdots \\ 1 & 0 & \cdots & -1 & 0 \end{pmatrix}$$

N by N differentiation matrix: 2nd derivative

$$D_{xx} = \frac{1}{\Delta x^2}\begin{pmatrix} -2 & 1 & 0 & \cdots & 1 \\ 1 & -2 & 1 & 0 & \cdots \\ 0 & 1 & -2 & 1 & \cdots \\ \vdots & & & & \vdots \\ 1 & 0 & \cdots & 1 & -2 \end{pmatrix}$$

Note: one typically has to carefully think about the boundary conditions. Here, we have assumed periodic bc's so that "what flows out from left/right end returns from the right/left end".

We can now express the “derivative vectors” of function c (stored in another vector) as follows

Estimate 1st derivative

$$\boldsymbol{c}_x = \boldsymbol{D}_x \boldsymbol{c}$$

Estimate 2nd derivative

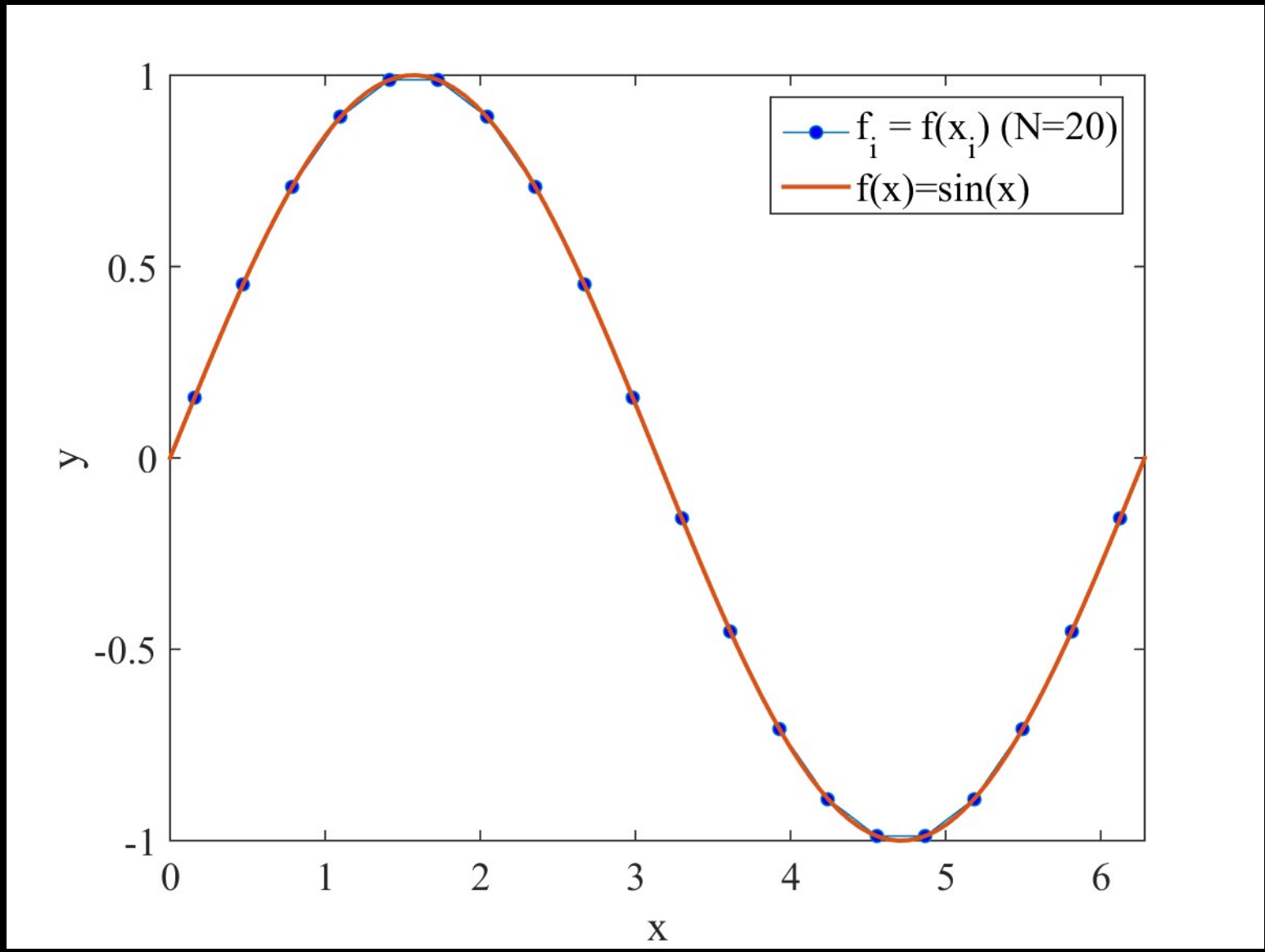
$$\boldsymbol{c}_{xx} = \boldsymbol{D}_{xx} \boldsymbol{c}$$

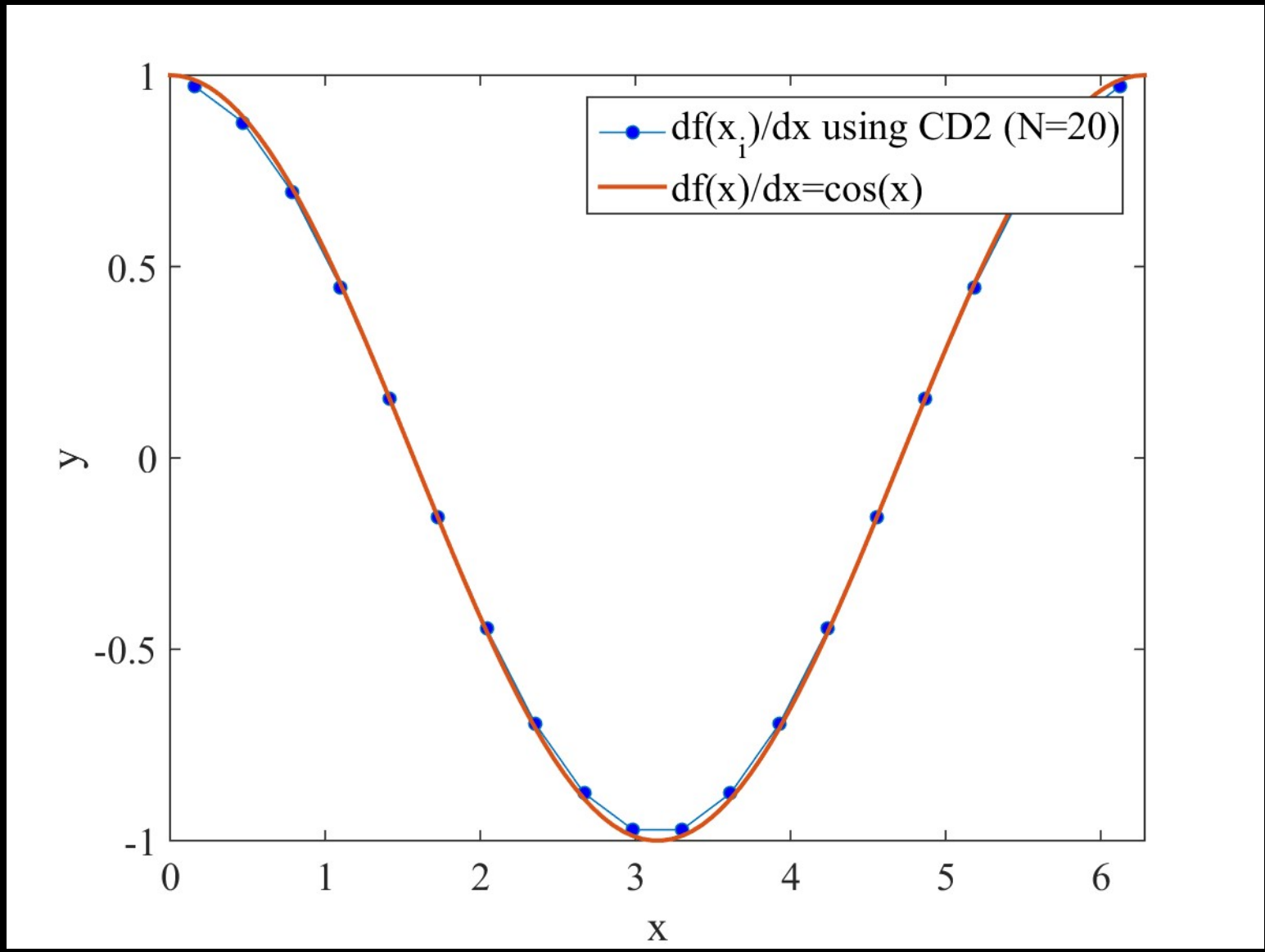
where, assuming periodic boundary conditions, the following vectors are observed:

$$\boldsymbol{c} = \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_N \end{pmatrix} \qquad \frac{\partial \boldsymbol{c}}{\partial x} = \frac{1}{2\Delta x} \begin{pmatrix} c_2 - c_N \\ c_3 - c_1 \\ \vdots \\ c_1 - c_{N-1} \end{pmatrix} \qquad \frac{\partial^2 \boldsymbol{c}}{\partial x^2} = \frac{1}{\Delta x^2} \begin{pmatrix} c_2 - 2c_1 + c_N \\ c_3 - 2c_2 + c_1 \\ \vdots \\ c_1 - 2c_N + c_{N-1} \end{pmatrix}$$

Practical implementation of CD2 finite difference scheme in Matlab

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Copying the text below to a Matlab file PlotDeriv.m
% enables to computing derivative of a f(x)=sin(x) using CD2 scheme
% using differentiation matrix
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear *
N=20; L=2*pi; dx = L/N;  % N points, domain length L
x = linspace(dx/2,L-dx/2, N);  % coordinate points, linearly spaced dx/2,3dx/2, … , L-dx/2
f=sin(x);  % test function f(x)=sin(x)

Dx = sparse(N,N);  % differentiation matrix is a sparse matrix = most elements are zeros

for(i=2:(N-1))
Dx(i,i+1)=1; Dx(i,i-1)=-1;  % fill the matrix but not boundary yet
end

Dx(1,N)=-1; Dx(1,2)=1;  % left boundary periodic
Dx(N,N-1)=-1; Dx(N,1)=1;  % right boundary periodic

Dx = Dx/(2*dx);  % divide by 2*dx

DfdxCD2 = Dx*(f');  % calculate dfdx by CD2: note f' transposes f, rewrite the ' in the editor
                    % to ensure Matlab compatibility
figure(1), clf; plot(x, DfdxCD2,'ko-')  % plot picture
```


1
0.5
0
-0.5
-1
y
0
1
2
3
4
5
6
x
f_i = f(x_i) (N=20)
f(x)=sin(x)


df(x_i)/dx using CD2 (N=20)
df(x)/dx=cos(x)
1
0.5
0
-0.5
-1
y
0
1
2
3
4
5
6
x

Differentiation matrices and practical implementation of finite difference method (CD2+Euler) for convection-diffusion eqn. Final assignment is a straightforward implementation of this but just need the matrices in y-direction as well.

Continuous equation:

$$\frac{\partial c}{\partial t} + u\frac{\partial c}{\partial x} = \alpha\frac{\partial^2 c}{\partial x^2}$$

Discretized equation by finite difference method **(see: Lectures 1-2)**

$$c_i^{n+1} = c_i^n - \Delta t\, u\frac{c_{i+1}^n - c_{i-1}^n}{2\Delta x} + \Delta t\,\alpha\frac{c_{i+1}^n - 2c_i^n + c_{i-1}^n}{\Delta x^2}$$

And, this transformed to a vector-matrix form where the vectors are N by 1 and the matrices are N by N sparse matrices:

$$\boldsymbol{c}_{n+1} = \boldsymbol{c}_n - \Delta t\, u\, \boldsymbol{D}_x \boldsymbol{c}_n + \Delta t\,\alpha\, \boldsymbol{D}_{xx}\, \boldsymbol{c}_n$$

It is straightforward to write this matrix-vector equation in Matlab and a for-loop in time, after the differentiation matrices have been defined. See a few slides back.

Solution of linear systems of equations

$$\mathbf{Ax} = \mathbf{b}$$

It is a common situation in CFD where we can not directly ("explicitly") find the solution on the next timestep or when the solution is steady state

Consider 1d heat equation with source term (spatial heating/cooling) on RHS using periodic boundary conditions. This is essentially the same problem that we need to solve for pressure in incompressible flows, the Poisson equation.

$$f(x) = \frac{\partial^2 T}{\partial x^2}$$

The problem can be transformed into the following matrix equation:

$$\boldsymbol{f} = \boldsymbol{D}_{xx}\boldsymbol{T}$$

$$\boldsymbol{f} = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_N \end{pmatrix} \qquad \boldsymbol{D}_{xx} = \frac{1}{\Delta x^2}\begin{pmatrix} -2 & 1 & 0 & \cdots & 1 \\ 1 & -2 & 1 & 0 & \cdots \\ 0 & 1 & -2 & 1 & \cdots \\ \vdots & & & & \vdots \\ 1 & 0 & \cdots & 1 & -2 \end{pmatrix} \qquad \boldsymbol{T} = \begin{pmatrix} T_1 \\ T_2 \\ T_3 \\ \vdots \\ T_N \end{pmatrix}$$

Observations 1/2

1) The solution is unique upto a constant because if T is a solution then also $T+const.$ is a solution

$$f(x)=\frac{\partial^2(T+const.)}{\partial x^2}=\frac{\partial^2 T}{\partial x^2}$$

2) This means that we need to fix the solution in one point. We could for instance require that $T_1 = f_1$ which would allow modifying the matrix in way that allows solution of the problem.

$$D_{xx}^{mod}=\frac{1}{\Delta x^2}\begin{pmatrix} \Delta x^2 & 0 & 0 & \cdots & 0 \\ 1 & -2 & 1 & 0 & \cdots \\ 0 & 1 & -2 & 1 & \cdots \\ \vdots & & & & \vdots \\ 1 & 0 & \cdots & 1 & -2 \end{pmatrix} \qquad T=\begin{pmatrix} T_1 \\ T_2 \\ T_3 \\ \vdots \\ T_N \end{pmatrix}$$

Observations 2/2

3) Compare the two matrices

$$D_{xx} = \frac{1}{\Delta x^2}\begin{pmatrix} -2 & 1 & 0 & \cdots & 1 \\ 1 & -2 & 1 & 0 & \cdots \\ 0 & 1 & -2 & 1 & \cdots \\ \vdots & & & & \vdots \\ 1 & 0 & \cdots & 1 & -2 \end{pmatrix} \qquad D_{xx}^{mod} = \frac{1}{\Delta x^2}\begin{pmatrix} \Delta x^2 & 0 & 0 & \cdots & 0 \\ 1 & -2 & 1 & 0 & \cdots \\ 0 & 1 & -2 & 1 & \cdots \\ \vdots & & & & \vdots \\ 1 & 0 & \cdots & 1 & -2 \end{pmatrix}$$

→ We see that in the original matrix the row sums are 0 i.e. -2+1+1=0 for each row. Such aspect is typically a sign that Ax=b does not pose a unique solution.

→ We see that in the modified matrix the row sums are 0 except for row number 1 for which the sum equals to 1. This is a typical positive indication that a matrix equation Ax=b will have a solution.

Typical ways to solve matrix equation Ax=b

1) **Direct inversion** of the matrix → done rather seldom in practice in CFD

$$x = A^{-1} b$$

2) **LU-decomposition** of the matrix into lower diagonal matrix L and upper diagonal matrix U → done every now and then in CFD

$$A = LU \qquad LUx = b$$

The system can be solved in two parts which are both easy to solve by back-substitution (also called Thomas algorithm) in two steps

$$Ly = b \rightarrow y = \ldots \qquad Ux = y \rightarrow x = \ldots$$

3) **Gauss-Seidel method**: iterate $L_A x_{k+1} = b - U_A x_k, \; where \; A = L_A + U_A$

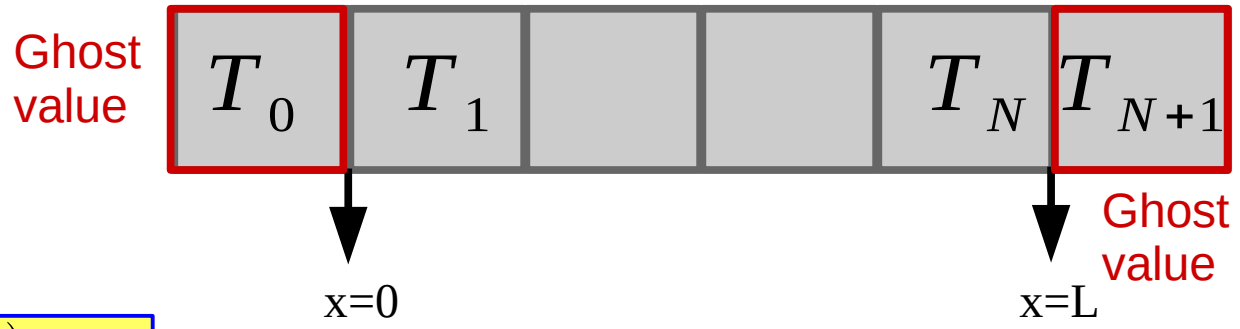
4) **Pre-conditioned conjugate gradient methods** → common iterative method in practice

5) **Multigrid methods** → common iterative method in practice

Implementation of boundary conditions outlined

Let us again consider the heat equation in 1d

$$0 = \frac{\partial^2 T}{\partial x^2}$$



Boundary conditions:

$$T(x=0) = T_A \text{ and } T(x=L) = T_B$$

The BC's can be implemented in various Ways. Here one of them is discussed:

$$D_{xx} = \frac{1}{\Delta x^2}\begin{pmatrix} -2 & 1 & 0 & \cdots & 1 \\ 1 & -2 & 1 & 0 & \cdots \\ 0 & 1 & -2 & 1 & \cdots \\ \vdots & & & & \vdots \\ 1 & 0 & \cdots & 1 & -2 \end{pmatrix}$$

Assumption 1) the BC is indeed exactly at $x=0$ and $x=L$ i.e. it is not a solution point

Assumption 2) The differentiation matrix is modified in the corners

To enforce certain value exactly at $x=0$ we can enforce:

$$0.5(T_0 + T_1) = T_A \rightarrow T_0 = 2T_A - T_1$$

→ This leads to modification of matrices in the corners so that the desired value for T_0 is gained

Cont.

NOTE: Equally well, we could have implemented the zero-gradient boundary condition type by requiring that the solution is constant across the border (i.e. gradient must be zero) by $T_o = T_1$