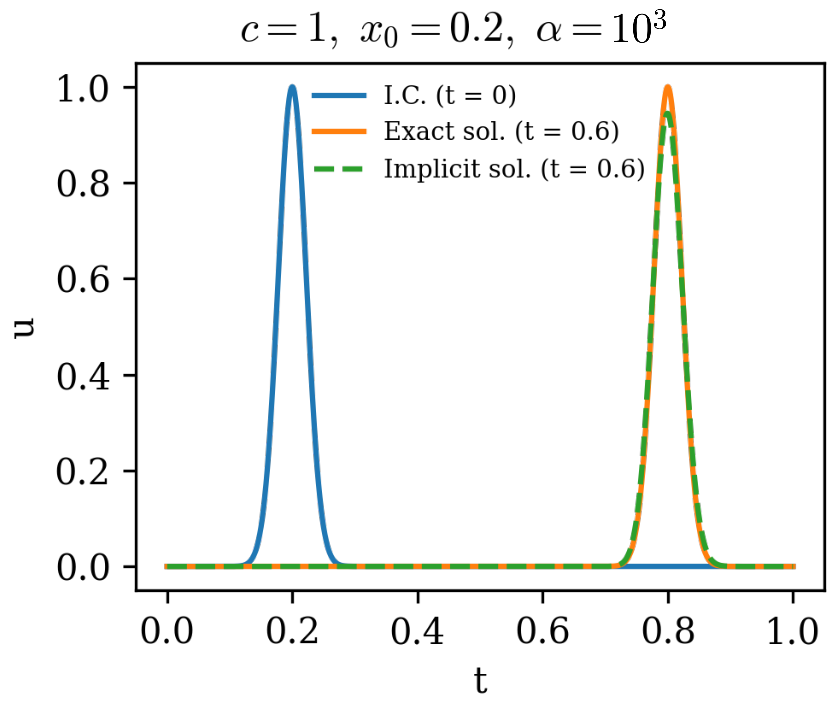
Solving PDE

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0, \quad u(x,\ t=0) = u_0 e^{-\alpha(x-x_0)^2}$$

Implicit discretization
Forward in time, central in space (FTCS)

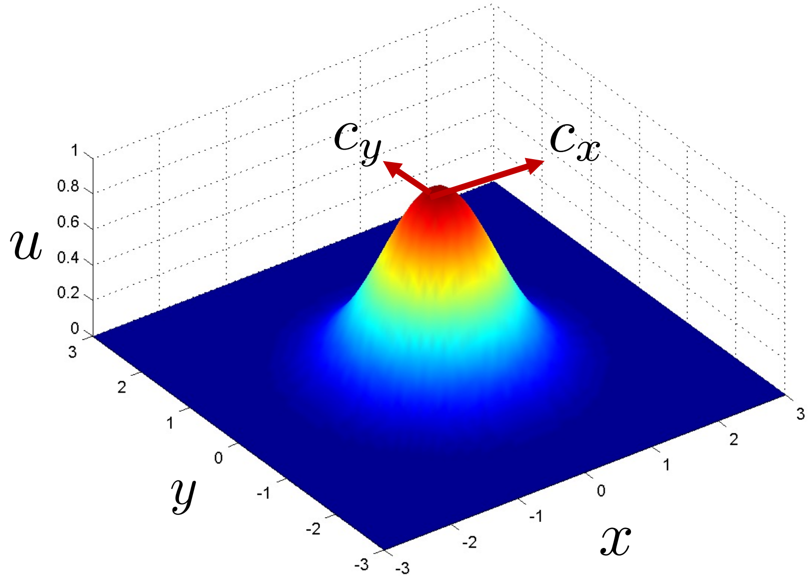
$$-\frac{N_c}{2}\, u_{i-1}^{n+1} + u_i^{n+1} + \frac{N_c}{2}\, u_{i+1}^{n+1} = u_i^n, \quad N_c = \frac{c\Delta t}{\Delta x}$$



Let's extend this simple equation to higher dimensions and add more terms to look like **Navier-Stokes** equation.

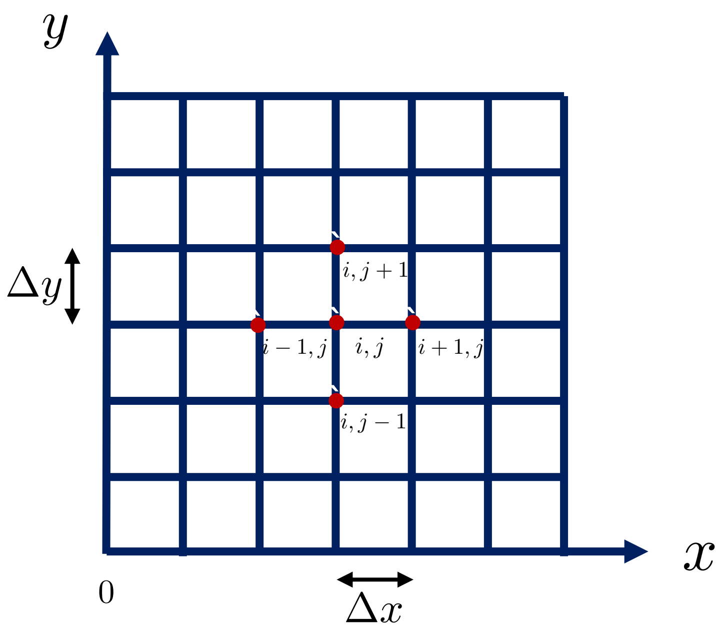
Solving PDE

$$\frac{\partial u}{\partial t} + c_x \frac{\partial u}{\partial x} + c_y \frac{\partial u}{\partial y} = 0, \quad u(x,\ t=0) = u_0 e^{-\alpha[(x-x_0)^2+(y-y_0)^2]}$$



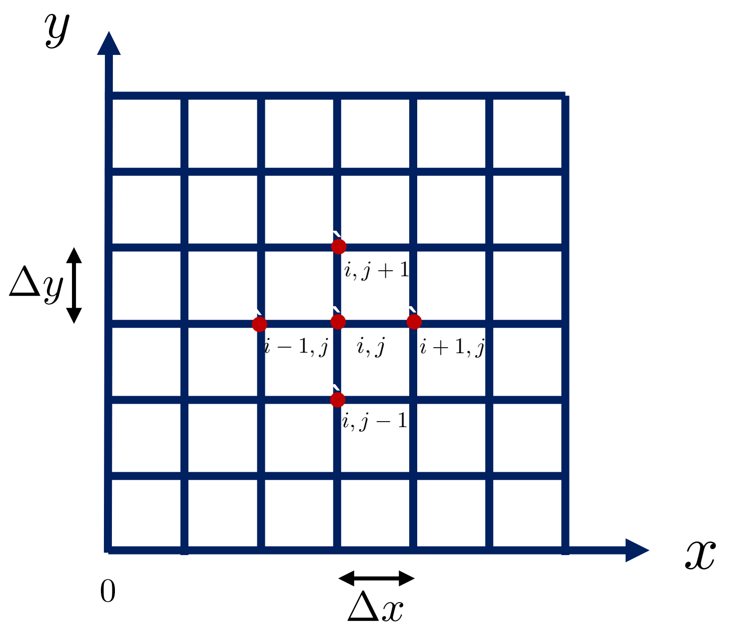
Implicit FTCS:

$$\frac{u_{ij}^{n+1} - u_{ij}^{n}}{\Delta t} + c_x \left[\frac{u_{i+1,j}^{n+1} - u_{i-1,j}^{n+1}}{2\Delta x} \right] + c_y \left[\frac{u_{i,j+1}^{n+1} - u_{i,j-1}^{n+1}}{2\Delta y} \right] = 0$$



Solving PDE

$$\frac{\partial u}{\partial t} + c_x \frac{\partial u}{\partial x} + c_y \frac{\partial u}{\partial y} = 0, \quad u(x,\ t=0) = u_0 e^{-\alpha[(x-x_0)^2+(y-y_0)^2]}$$



Implicit FTCS:

$$\frac{u_{ij}^{n+1} - u_{ij}^{n}}{\Delta t} + c_x \left[\frac{u_{i+1,j}^{n+1} - u_{i-1,j}^{n+1}}{2\Delta x} \right] + c_y \left[\frac{u_{i,j+1}^{n+1} - u_{i,j-1}^{n+1}}{2\Delta y} \right] = 0$$

$$\left[\begin{array}{ccccccccccccccccc}
 & & & & & & & \vdots & & & & & & & & & \\
\cdots & 0 & -\frac{c_y\Delta t}{2\Delta x} & 0 & \cdots & 0 & -\frac{c_x\Delta t}{2\Delta x} & 1 & \frac{c_x\Delta t}{2\Delta x} & 0 & \cdots & 0 & \frac{c_y\Delta t}{2\Delta x} & 0 & \cdots & & \\
 & \cdots & 0 & -\frac{c_y\Delta t}{2\Delta y} & 0 & \cdots & 0 & -\frac{c_x\Delta t}{2\Delta x} & 1 & \frac{c_x\Delta t}{2\Delta x} & 0 & \cdots & 0 & \frac{c_y\Delta t}{2\Delta y} & 0 & \cdots & \\
 & & \cdots & 0 & -\frac{c_y\Delta t}{2\Delta y} & 0 & \cdots & 0 & -\frac{c_x\Delta t}{2\Delta x} & 1 & \frac{c_x\Delta t}{2\Delta x} & 0 & \cdots & 0 & \frac{c_y\Delta t}{2\Delta y} & 0 & \cdots \\
 & & & & & & & \vdots & & & & & & & & &
\end{array}\right]
\begin{bmatrix} \vdots \\ u_{i,j-1} \\ \vdots \\ u_{i-1,j} \\ u_{i,j} \\ u_{i+1,j} \\ \vdots \\ u_{i,j+1} \\ \vdots \end{bmatrix}
=
\begin{bmatrix} \vdots \\ b_{i,j-1} \\ \vdots \\ b_{i-1,j} \\ b_{i,j} \\ b_{i+1,j} \\ \vdots \\ b_{i,j+1} \\ \vdots \end{bmatrix}$$

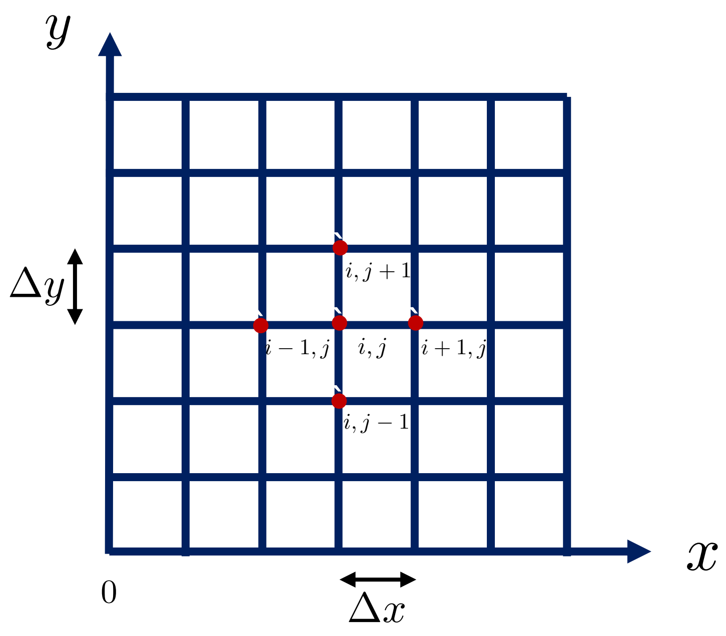
Solve a penta-diagonal matrix equation

Solving PDE

$$\frac{\partial u}{\partial t} + (\boldsymbol{c} \cdot \nabla)\, u = 0, \quad u(\boldsymbol{x},\ t=0) = u_0 e^{-\alpha \|\boldsymbol{x}-\boldsymbol{x}_0\|^2}$$

Replace: $u \to \boldsymbol{u}$, $\boldsymbol{c} \to \boldsymbol{u}$

$$\frac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{u} \cdot \nabla)\, \boldsymbol{u} = 0$$



$$\frac{\boldsymbol{u}_{ij}^{n+1} - \boldsymbol{u}_{ij}^{n}}{\Delta t} + u_x \left[\frac{\boldsymbol{u}_{i+1,j}^{n+1} - \boldsymbol{u}_{i-1,j}^{n+1}}{2\Delta x}\right] + u_y \left[\frac{\boldsymbol{u}_{i,j+1}^{n+1} - \boldsymbol{u}_{i,j-1}^{n+1}}{2\Delta y}\right] = 0$$

$$\frac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{u} \cdot \nabla)\, \boldsymbol{u} = \nu \nabla^2 \boldsymbol{u}$$

$$\begin{aligned} \nu \nabla^2 \boldsymbol{u} &= \nu \left[\frac{\partial^2 \boldsymbol{u}}{\partial x^2} + \frac{\partial^2 \boldsymbol{u}}{\partial y^2}\right] \\ &= \nu \left[\frac{\boldsymbol{u}_{i-1,j}^{n+1} - 2\boldsymbol{u}_{i,j}^{n+1} + \boldsymbol{u}_{i+1,j}^{n+1}}{\Delta x^2} + \frac{\boldsymbol{u}_{i,j-1}^{n+1} - 2\boldsymbol{u}_{i,j}^{n+1} + \boldsymbol{u}_{i,j+1}^{n+1}}{\Delta y^2}\right] \end{aligned}$$

Incompressible Navier-Stokes solution

$$\nabla \cdot \boldsymbol{u} = 0$$

$$\frac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{u} \cdot \nabla)\boldsymbol{u} = -\nabla \left(\frac{p}{\rho} \right) + \frac{1}{Re} \nabla^2 \boldsymbol{u}$$

Non-linear term, can't have implicit discretization without linearization.

First, let's consider steady solution for simplicity.

Steady, incompressible Navier-Stokes solution

$$\nabla \cdot \boldsymbol{u} \;=\; 0$$

$$(\boldsymbol{u} \cdot \nabla)\boldsymbol{u} - \frac{1}{Re}\nabla^2 u \;= -\nabla P$$

Momentum equation can be discretized as:

$$(\boldsymbol{u}^n \cdot \nabla)\boldsymbol{u}^{n+1} - \frac{1}{Re}\nabla^2 u^{n+1} \;= -\nabla P^n$$

$$\mathcal{M}\boldsymbol{u}^{n+1} \;=\; -\nabla P$$

$\mathcal{M}$ obtained from discretizing the terms in the LHS

$$\mathcal{M}\boldsymbol{u}^{n+1} \;= \mathcal{A}\boldsymbol{u}^{n+1} \;-\; \mathcal{H}$$

$\mathcal{A}$: Diagonal part of $\mathcal{M}$

$\mathcal{H}$: off-diagonal part of $\mathcal{M}u^n$

Diagonal matrices are easily invertible.

Steady, incompressible Navier-Stokes solution

Momentum equation: $$\mathcal{M}\boldsymbol{u}^{n+1} = -\nabla P$$

Momentum equation (semi-implicit form): $$\mathcal{A}\boldsymbol{u}^{n+1} - \mathcal{H} = -\nabla P$$

Corrected velocity at next step: $$\boldsymbol{u}^{n+1} = \mathcal{A}^{-1}\mathcal{H} - \mathcal{A}^{-1}\nabla P$$

Continuity equation: $\nabla \cdot \boldsymbol{u} = 0$ $$\nabla \cdot [\mathcal{A}^{-1}\mathcal{H} - \mathcal{A}^{-1}\nabla P] = 0$$

Pressure – Poisson equation: $$\nabla \cdot (\mathcal{A}^{-1}\nabla P) = \nabla \cdot (\mathcal{A}^{-1}\mathcal{H})$$

SIMPLE algorithm

Semi-Implicit Method for Pressure Linked Equations

1. Solve momentum equation.

$$\mathcal{M}\boldsymbol{u}^{n+1} = -\nabla P$$

2. Since velocity field doesn't satisfy continuity equation, solve Poisson equation for pressure

$$\nabla \cdot (\mathcal{A}^{-1} \nabla P) = \nabla \cdot (\mathcal{A}^{-1}\mathcal{H})$$

3. Correct velocity field.

$$\boldsymbol{u}^{n+1} = \mathcal{A}^{-1}\mathcal{H} - \mathcal{A}^{-1} \nabla P$$

4. Since, corrected velocity field doesn't satisfy momentum equation, go back to step 1 to solve momentum equation until convergence.

```
#include "fvCFD.H"
#include "singlePhaseTransportModel.H"
#include "turbulentTransportModel.H"
#include "simpleControl.H"
#include "fvOptions.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

int main(int argc, char *argv[])
{
    #include "postProcess.H"

    #include "setRootCaseLists.H"
    #include "createTime.H"
    #include "createMesh.H"
    #include "createControl.H"
    #include "createFields.H"
    #include "initContinuityErrs.H"

    turbulence->validate();

    // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

    Info<< "\nStarting time loop\n" << endl;

    while (simple.loop(runTime))
    {
        Info<< "Time = " << runTime.timeName() << nl << endl;

        // --- Pressure-velocity SIMPLE corrector
        {
            #include "UEqn.H"
            #include "pEqn.H"
        }

        laminarTransport.correct();
        turbulence->correct();

        runTime.write();

        Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
            << "  ClockTime = " << runTime.elapsedClockTime() << " s"
            << nl << endl;
    }

    Info<< "End\n" << endl;

    return 0;
}
```

```
    // Momentum predictor

    MRF.correctBoundaryVelocity(U);

    tmp<fvVectorMatrix> tUEqn
    (
        fvm::div(phi, U)
      + MRF.DDt(U)
      + turbulence->divDevReff(U)
     ==
        fvOptions(U)
    );
    fvVectorMatrix& UEqn = tUEqn.ref();

    UEqn.relax();

    fvOptions.constrain(UEqn);

    if (simple.momentumPredictor())
    {
        solve(UEqn == -fvc::grad(p));

        fvOptions.correct(U);
    }
```

```
{
    volScalarField rAU(1.0/UEqn.A());
    volVectorField HbyA(constrainHbyA(rAU*UEqn.H(), U, p));
    surfaceScalarField phiHbyA("phiHbyA", fvc::flux(HbyA));
    MRF.makeRelative(phiHbyA);
    adjustPhi(phiHbyA, U, p);

    tmp<volScalarField> rAtU(rAU);

    if (simple.consistent())
    {
        rAtU = 1.0/(1.0/rAU - UEqn.H1());
        phiHbyA +=
            fvc::interpolate(rAtU() - rAU)*fvc::snGrad(p)*mesh.magSf();
        HbyA -= (rAU - rAtU())*fvc::grad(p);
    }

    tUEqn.clear();

    // Update the pressure BCs to ensure flux consistency
    constrainPressure(p, U, phiHbyA, rAtU(), MRF);

    // Non-orthogonal pressure corrector loop
    while (simple.correctNonOrthogonal())
    {
        fvScalarMatrix pEqn
        (
            fvm::laplacian(rAtU(), p) == fvc::div(phiHbyA)
        );

        pEqn.setReference(pRefCell, pRefValue);

        pEqn.solve();

        if (simple.finalNonOrthogonalIter())
        {
            phi = phiHbyA - pEqn.flux();
        }
    }

    #include "continuityErrs.H"

    // Explicitly relax pressure for momentum corrector
    p.relax();

    // Momentum corrector
    U = HbyA - rAtU()*fvc::grad(p);
    U.correctBoundaryConditions();
    fvOptions.correct(U);
}
```