EMAT30008: Scientific Computing

Part 3: PDE Problems

Martin Homer

Department of Engineering Mathematics
martin.homer@bristol.ac.uk

Outline

- Learn about **finite difference methods**: (one way) to numerically solve (some) partial differential equations problems
- Main focus will be applying them to parabolic PDEs, though the ideas can be extended to hyperbolic and elliptic PDE problems

Rough outline

- PDE recap
- Finite difference method basics
- Explicit vs. implicit methods
- Stability and accuracy
- Implementation issues

- We'll start by solving the heat equation with homogenous boundary conditions, and then generalise to deal with
  - different boundary conditions (Dirichlet vs. Neumann)
  - other parabolic PDEs
  - more space dimensions

PDEs — recap

- General 2nd order semilinear PDE for a function u of two variables x and y has the form

$$a\frac{\partial^2 u}{\partial x^2} + b\frac{\partial^2 u}{\partial x \partial y} + c\frac{\partial^2 u}{\partial y^2} = f\left(u, \frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}, x, y\right)$$

- a, b, c can be functions of x and y
- There are three classes of such PDEs
  - $b^2 - 4ac > 0$: **hyperbolic**, has solutions that travel, discontinuities can be generated and propagate, can be tricky to solve computationally
  - $b^2 - 4ac < 0$: **elliptic**, has stationary smooth solutions, usually not too hard to solve by computation
  - $b^2 - 4ac = 0$: **parabolic**, has solutions that diffuse, may have features of both

PDEs — recap

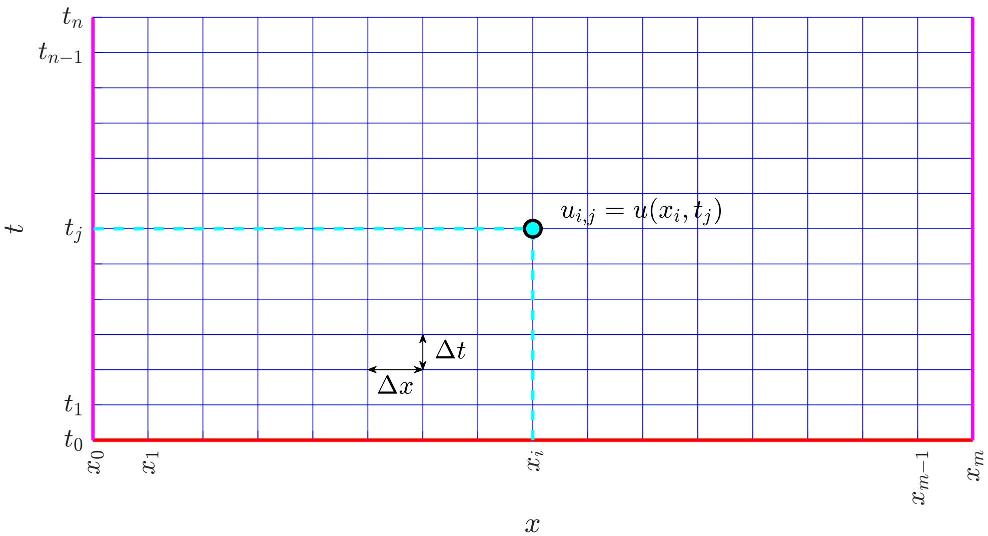
Class	Canonical example	Initial/Boundary conditions
Parabolic	Heat equation $u_t = \kappa u_{xx}$ $u_t = \kappa \nabla^2 u$	*Initial/boundary value problem* x space coord., t time coord. Need u at $t = 0$, plus one condition on each boundary[1] in space (u or $\frac{\partial u}{\partial n}$)
Hyperbolic	Wave equation $u_{tt} = c^2 u_{xx}$ $u_{tt} = c^2 \nabla^2 u$	*Initial/boundary value problem* x space coord., t time coord. Need u and u_t at $t = 0$, plus one condition on each boundary in space (u or $\frac{\partial u}{\partial n}$)
Elliptic	Laplace's equation $u_{xx} + u_{yy} = 0$	*Boundary value problem* x, y space coords.; no time coord. Need one condition everywhere on the (closed) boundary (u or $\frac{\partial u}{\partial n}$)

[1] Dirichlet = prescribed u, Neumann = prescribed $\frac{\partial u}{\partial n}$ on a boundary

Numerical methods for PDEs

- We will focus on **finite difference** methods
  - discretize the domain into a regular grid: $x_i = x_0 + i\Delta x$, $t_j = t_0 + j\Delta t$
  - discretize the solution values at the gridpoints: $u_{i,j} = u(x_i, t_j)$
  - discretize the PDE: use finite differencing to approximate derivatives as combinations of $u_{i,j}$
  - leads to sets of equations for $u_{i,j}$
  - solve to find the $u_{i,j}$ using linear algebra/nonlinear system techniques
- Pros and cons:
  - \+ straightforward, both mathematically and also from a coding perspective
  - − only really works for simple (rectangular) domains
- There are many alternative approaches: e.g., finite element/volume methods, spectral methods, method of lines, ...

Grid



Approximating derivatives

- Approximate first order derivatives with forward, backward, or central difference; e.g.

$$\frac{\partial u}{\partial t}(x_i, t_j) \approx \frac{u(x_i, t_j + \Delta t) - u(x_i, t_j)}{\Delta t} = \frac{u_{i,j+1} - u_{i,j}}{\Delta t}$$

$$\text{or} \quad \frac{\partial u}{\partial t}(x_i, t_j) \approx \frac{u(x_i, t_j) - u(x_i, t_j - \Delta t)}{\Delta t} = \frac{u_{i,j} - u_{i,j-1}}{\Delta t}$$

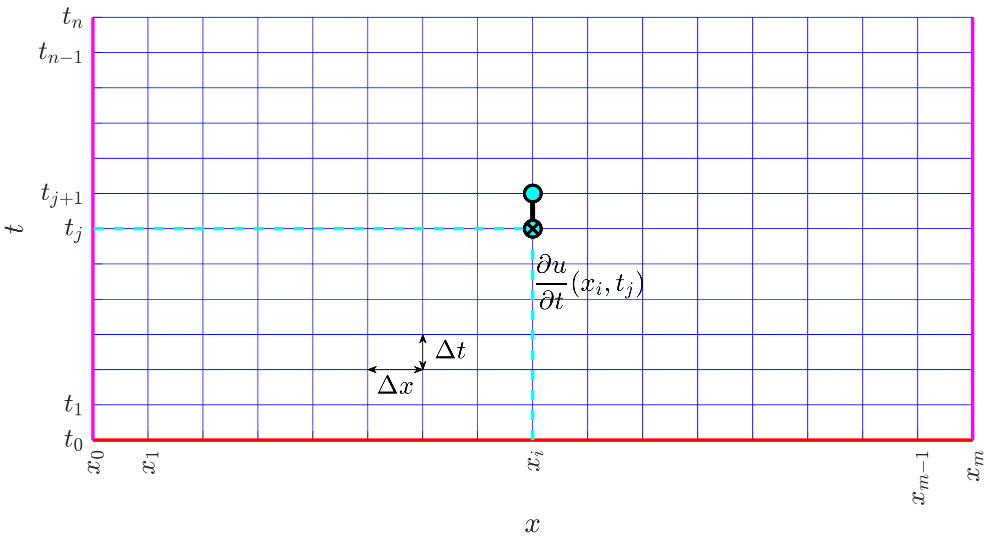
$$\text{or} \quad \frac{\partial u}{\partial t}(x_i, t_j) \approx \frac{u(x_i, t_j + \Delta t) - u(x_i, t_j - \Delta t)}{2\Delta t} = \frac{u_{i,j+1} - u_{i,j-1}}{2\Delta t}$$

- Second derivatives can be approximated too, e.g. using central difference

$$\frac{\partial^2 u}{\partial x^2}(x_i, t_j) \approx \frac{u(x_i + \Delta x, t_j) - 2u(x_i, t_j) + u(x_i - \Delta x, t_j)}{\Delta x^2} = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2}$$

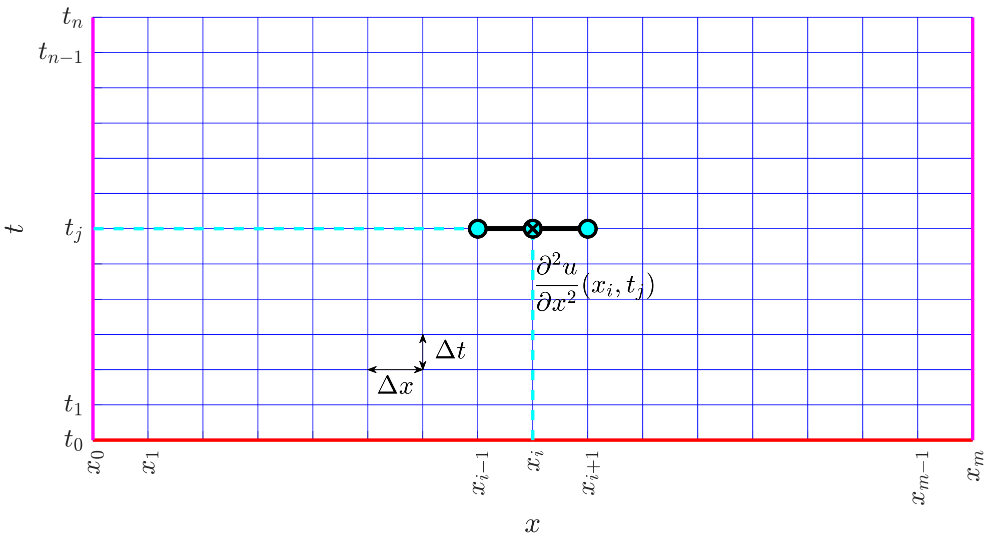
- Many other approximations exist: more accuracy, higher derivatives

Grid



t_n
t_{n-1}
t_{j+1}
t_j
t
t_1
t_0
$\frac{\partial u}{\partial t}(x_i, t_j)$
Δt
Δx
x_0
x_1
x_i
x_{m-1}
x_m
x

Grid


t_n
t_{n-1}
t
t_j
t_1
t_0
$\frac{\partial^2 u}{\partial x^2}(x_i, t_j)$
Δt
Δx
x_0
x_1
x_{i-1}
x_i
x_{i+1}
x_{m-1}
x_m
x

Starter problem

- Solve the heat equation

$$\frac{\partial u}{\partial t} = \kappa \frac{\partial^2 u}{\partial x^2}, \quad 0 < x < L, 0 < t \leqslant T \tag{1}$$

with boundary & initial conditions

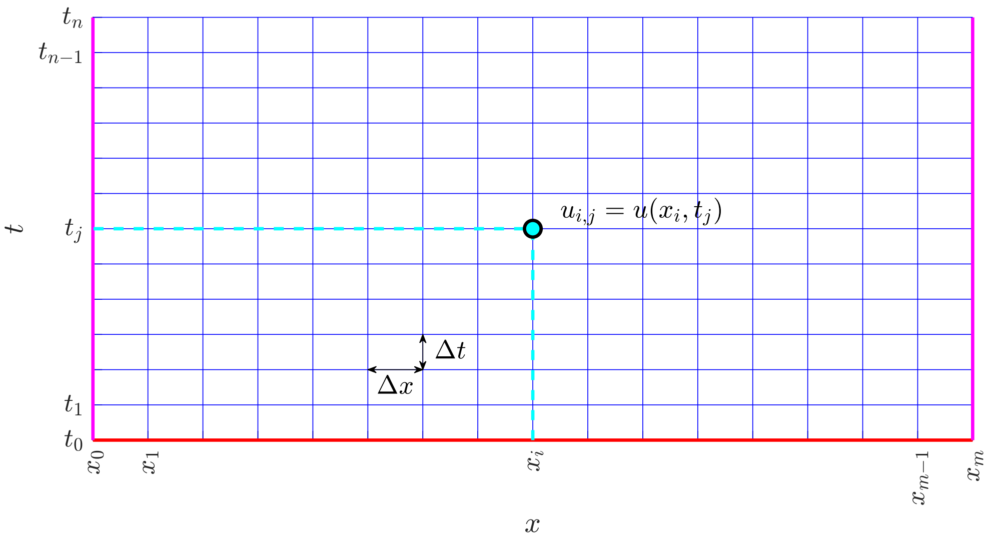
$$u(0,t) = u(L,t) = 0, \quad t > 0, \qquad u(x,0) = f(x), \quad 0 < x < L \tag{2}$$

- Select mesh constants Δx, Δt, so that

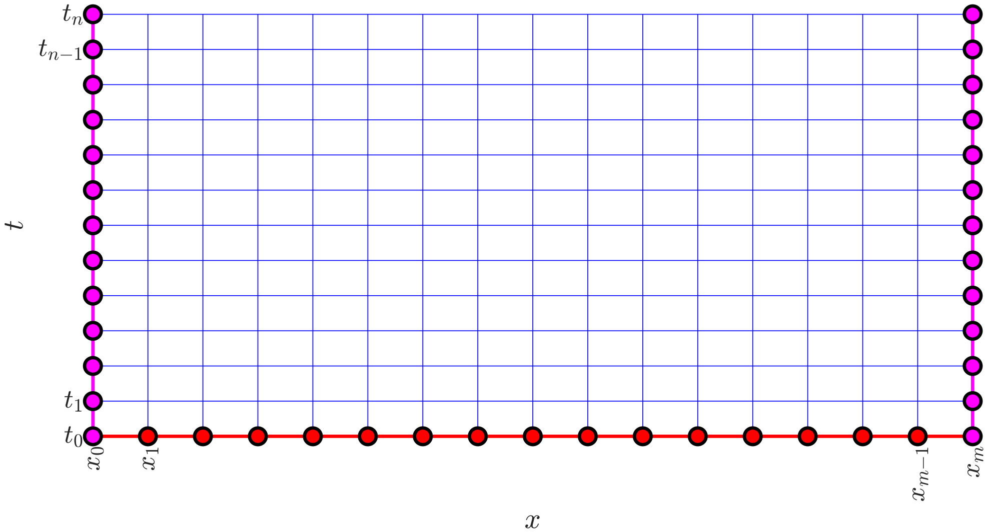
$$\frac{L}{\Delta x} = m, \quad \frac{T}{\Delta t} = n, \qquad m, n \in \mathbb{Z}$$

- Grid points are $(x_i, t_j) = (i\Delta x, j\Delta t)$, where $i = 0, 1, \ldots, m$, and $j = 0, 1, 2, \ldots, n$
- Find the approximate solution $u_{i,j} = u(x_i, t_j)$

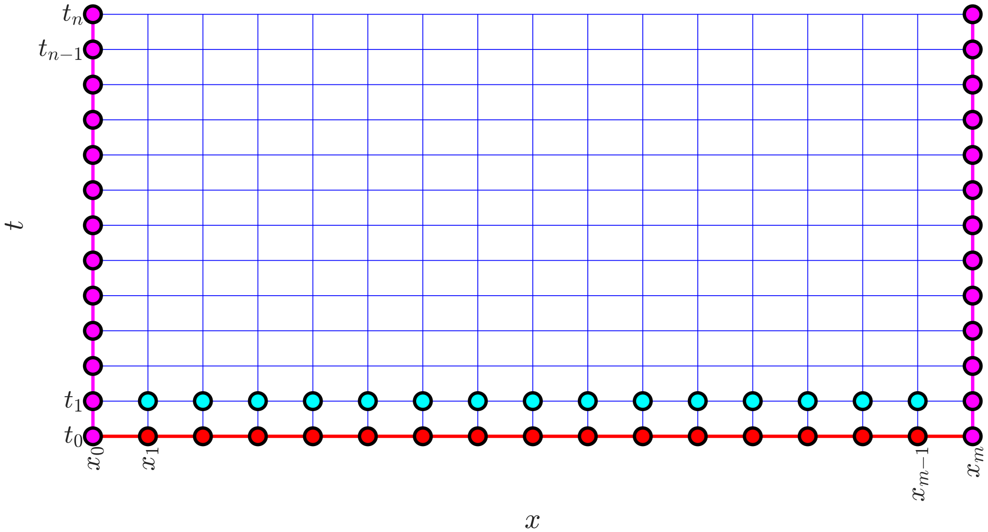
Grid



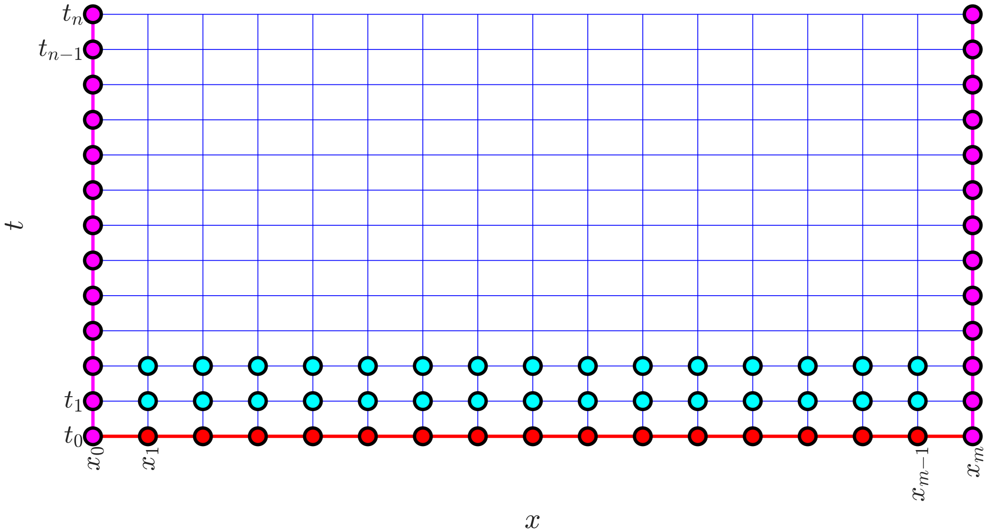
Grid



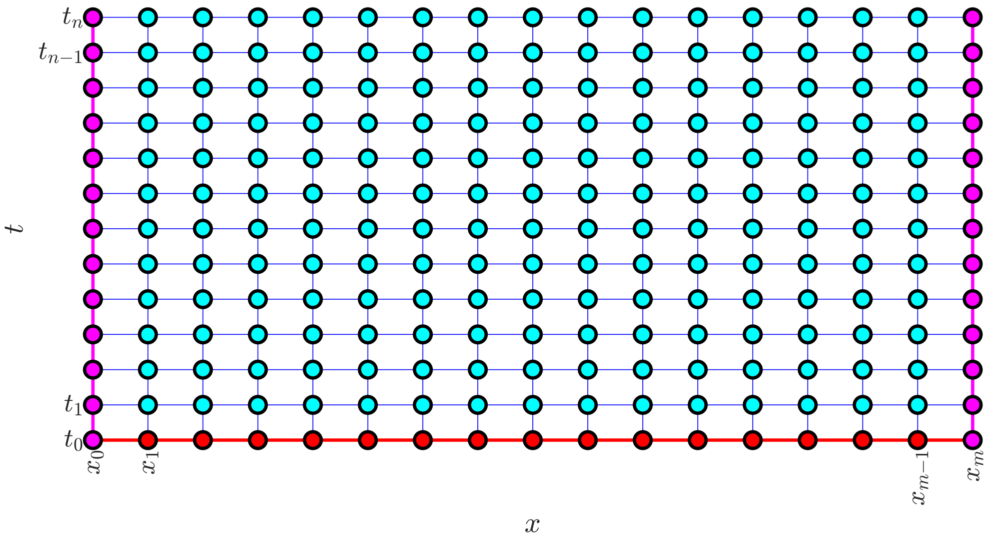
Grid



Grid



Grid



Discretizing the PDE

- Expand the PDE about $(x, y) = (x_i, t_j)$, by discretizing the derivatives
- Use forward difference in time, central difference in space

$$\frac{\partial u}{\partial t} = \kappa \frac{\partial^2 u}{\partial x^2}$$

Discretizing the PDE

- Expand the PDE about $(x, y) = (x_i, t_j)$, by discretizing the derivatives
- Use forward difference in time, central difference in space

$$\frac{\partial u}{\partial t}(x_i, t_j) = \kappa \frac{\partial^2 u}{\partial x^2}(x_i, t_j)$$

Discretizing the PDE

- Expand the PDE about $(x, y) = (x_i, t_j)$, by discretizing the derivatives
- Use forward difference in time, central difference in space

$$\frac{u_{i,j+1} - u_{i,j}}{\Delta t} = \kappa \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2}$$

- Rearrange to give an equation for $u_{i,j+1}$ in terms of the $u_{i,j}$s

$$u_{i,j+1} = u_{i,j} + \frac{\kappa \Delta t}{\Delta x^2} \left(u_{i+1,j} - 2u_{i,j} + u_{i-1,j} \right)$$

- We can do timestepping! Solution at next timestep $(j+1)$ is given *explicitly* in terms of solution at previous timestep (j)
- This is the *forward Euler* scheme: an *explicit* method

Discretizing the PDE

- Expand the PDE about $(x, y) = (x_i, t_j)$, by discretizing the derivatives
- Use forward difference in time, central difference in space

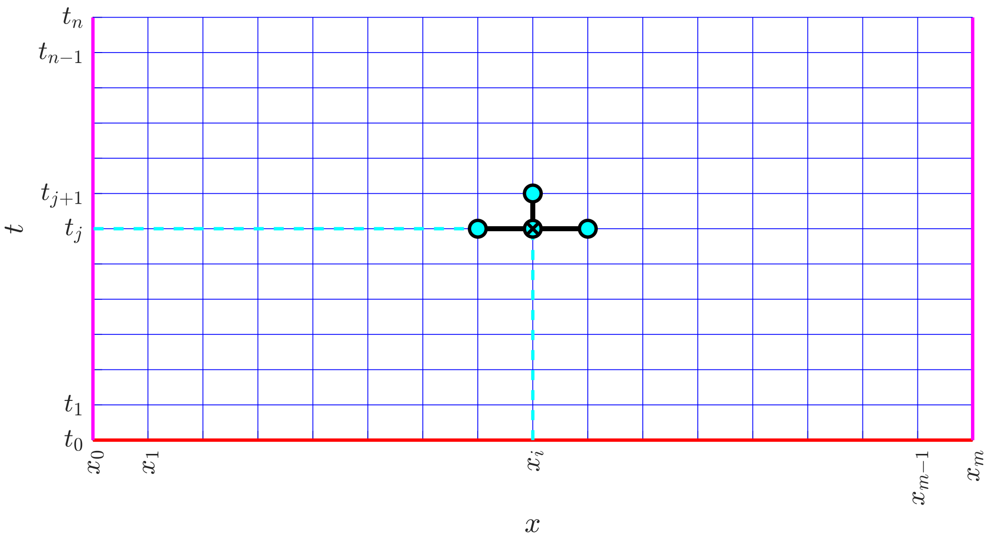
$$\frac{u_{i,j+1} - u_{i,j}}{\Delta t} = \kappa \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2}$$

- Rearrange to give an equation for $u_{i,j+1}$ in terms of the $u_{i,j}$s

$$u_{i,j+1} = u_{i,j} + \lambda \left(u_{i+1,j} - 2u_{i,j} + u_{i-1,j} \right), \qquad \lambda = \frac{\kappa \Delta t}{\Delta x^2}$$

- We can do timestepping! Solution at next timestep $(j+1)$ is given *explicitly* in terms of solution at previous timestep (j)
- This is the *forward Euler* scheme: an *explicit* method

Forward Euler — stencil



Boundary/initial conditions

- Dealing with the boundary and initial conditions is easy
- Boundary conditions give, for all j

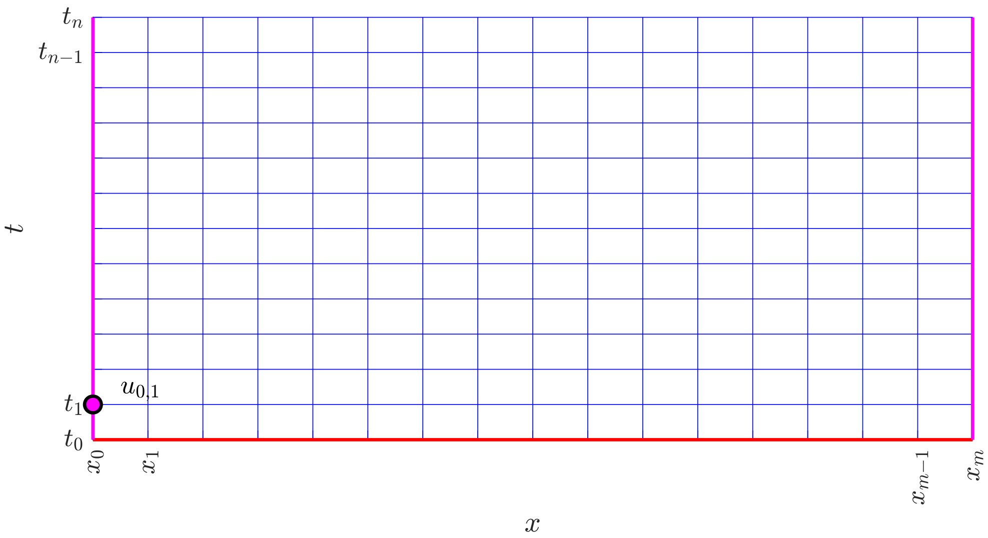
$$u_{0,j} = u_{m,j} = 0$$

- Use initial condition (at time $t = 0$, i.e. for $j = 0$) to start the iteration

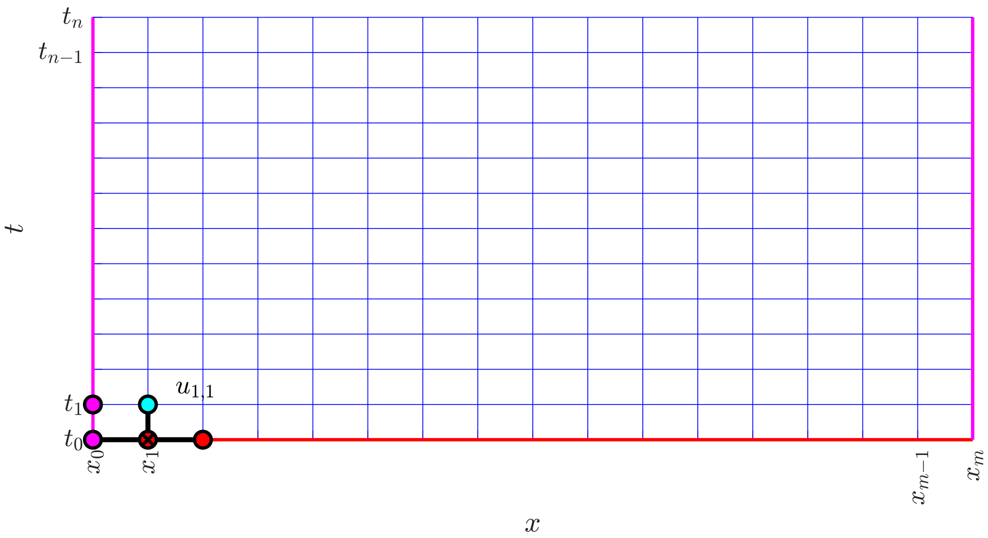
$$u_{i,0} = f(x_i)$$

- The (unknown) solution values we need to find are $u_{1,j}, u_{2,j}, \ldots, u_{m-1,j}$ for $j = 1, 2, \ldots, n$
- Find them one row at a time: compute all the $u_{i,j}$ for $j = 1$, then for $j = 2$, etc.

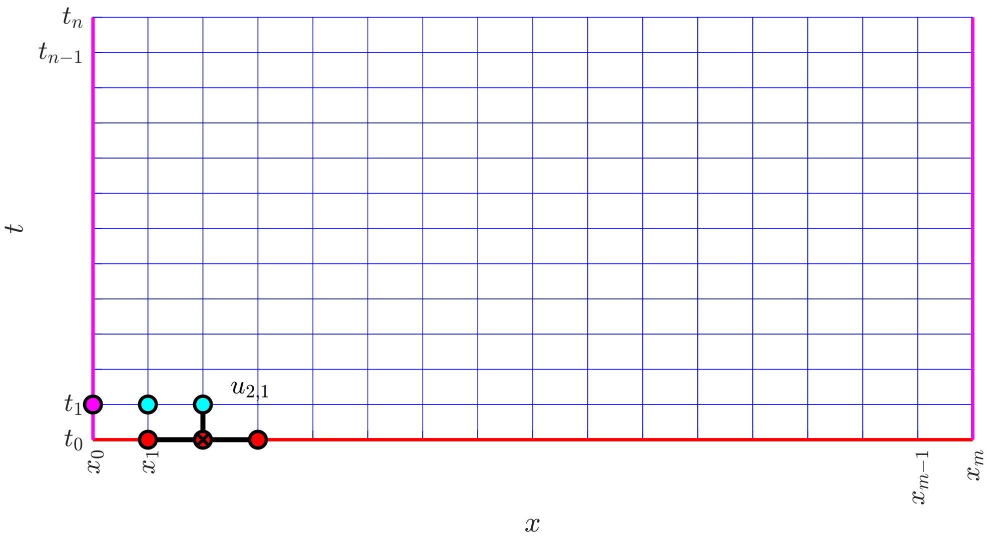
Forward Euler — iteration



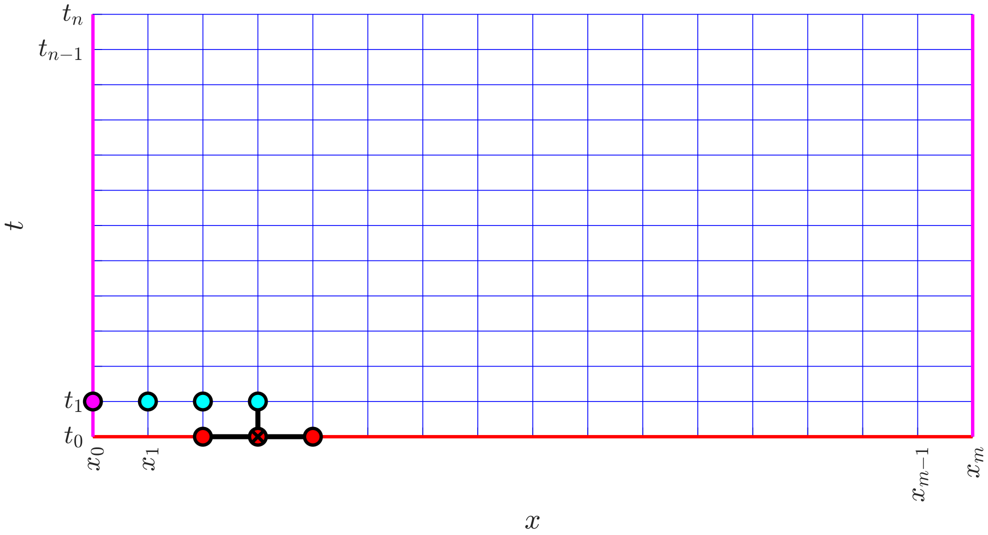
Forward Euler — iteration



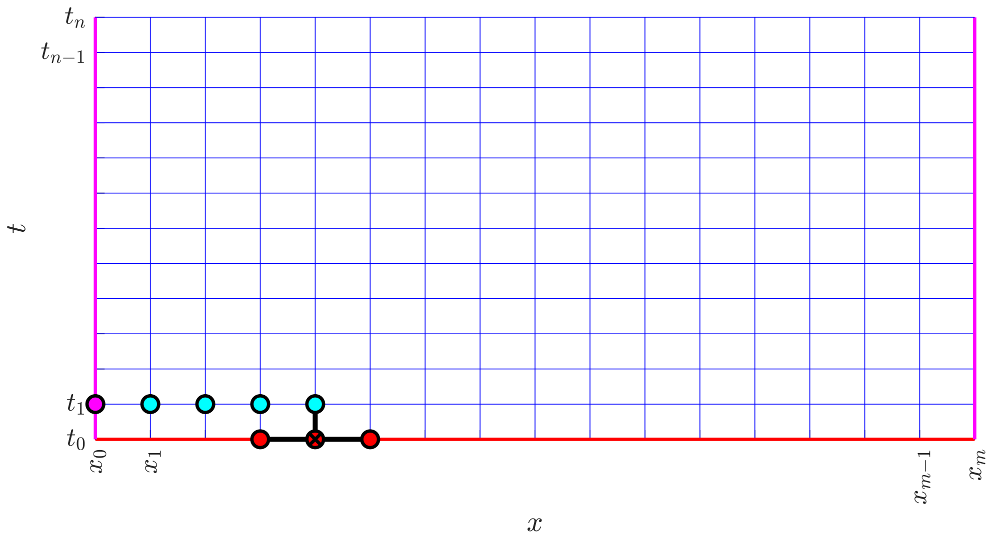
Forward Euler — iteration



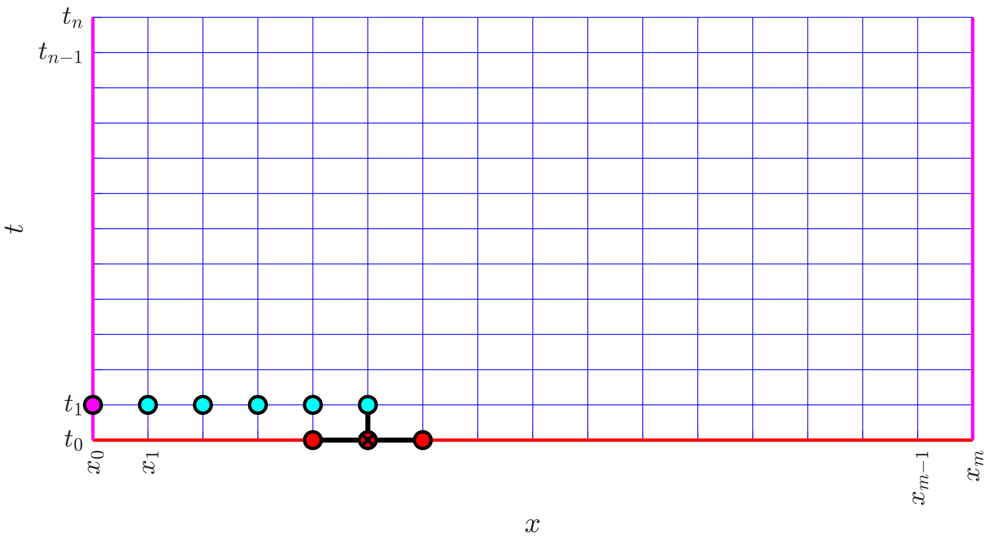
Forward Euler — iteration



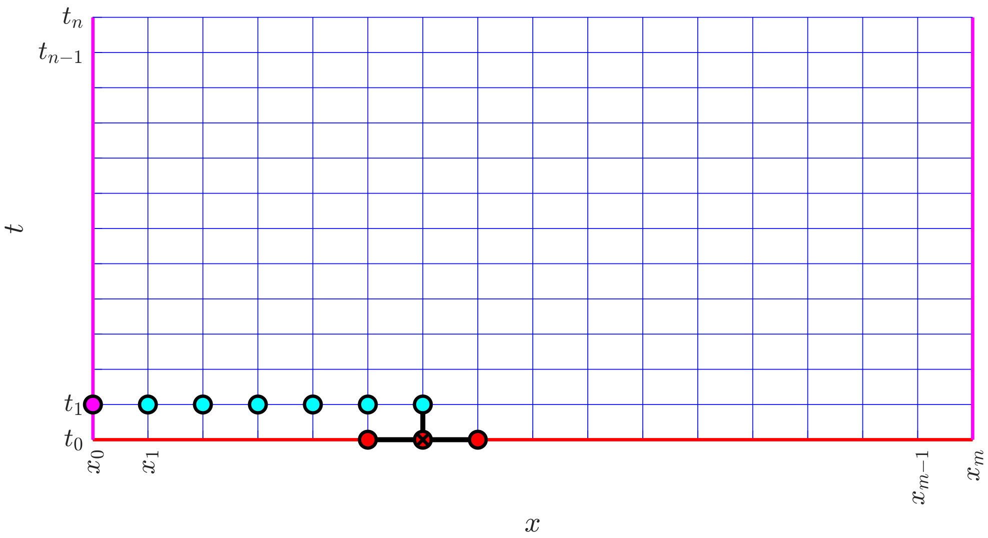
Forward Euler — iteration



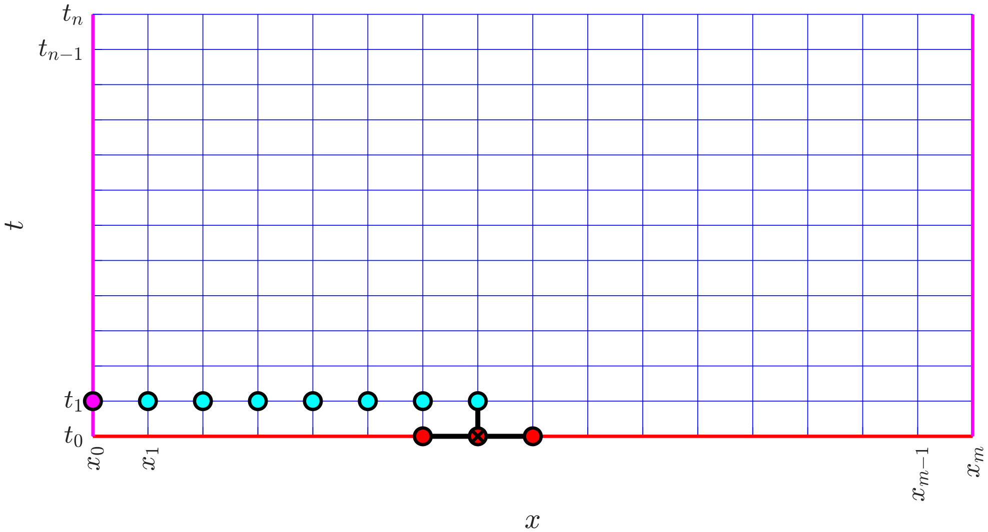
Forward Euler — iteration



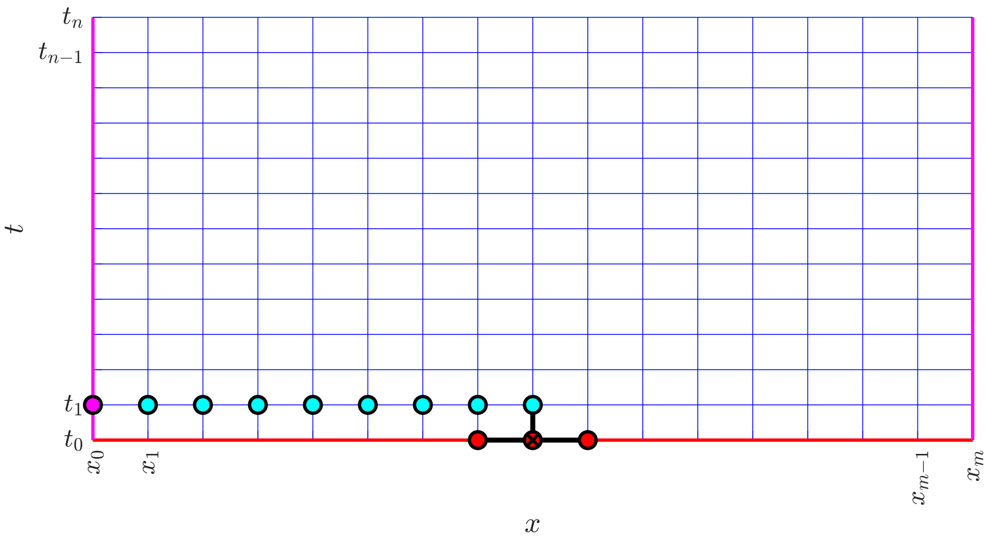
Forward Euler — iteration



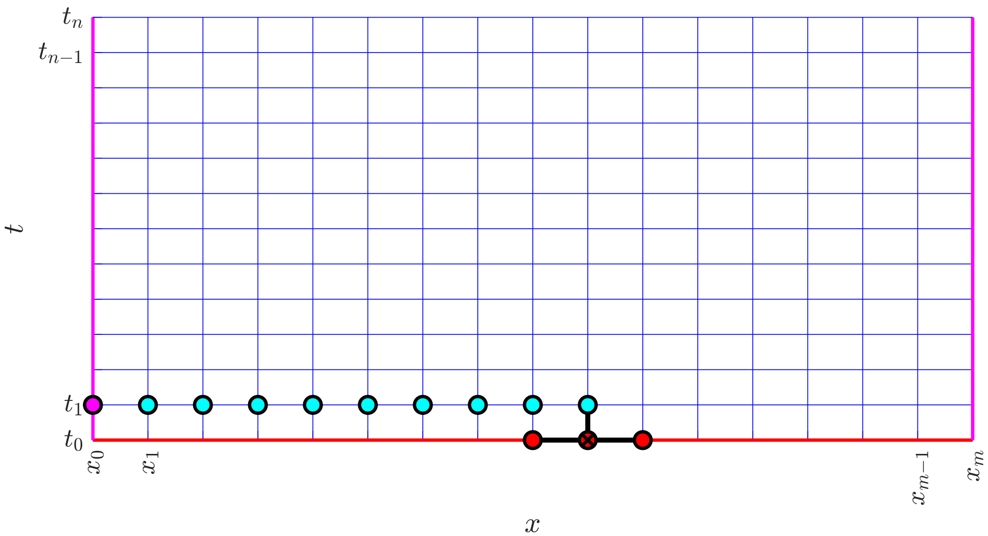
Forward Euler — iteration



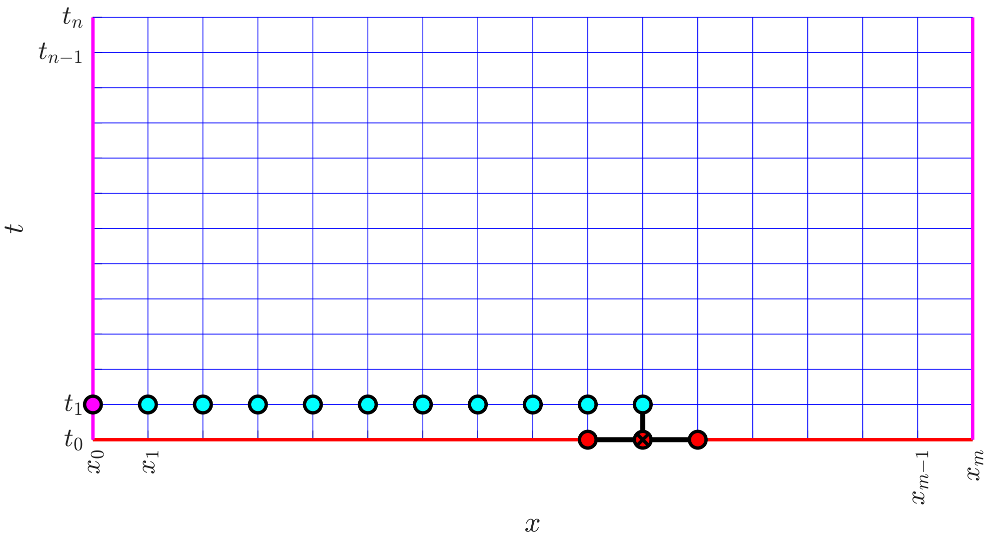
Forward Euler — iteration



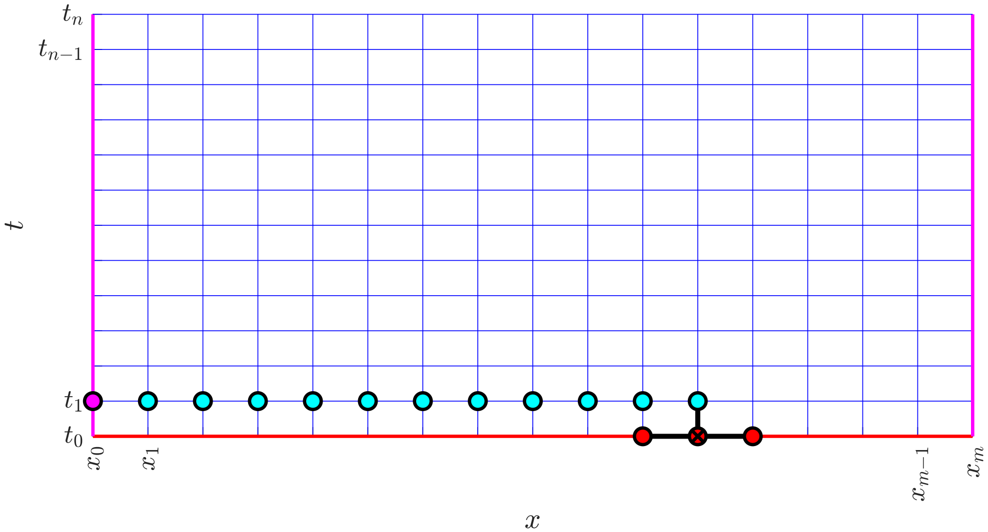
Forward Euler — iteration



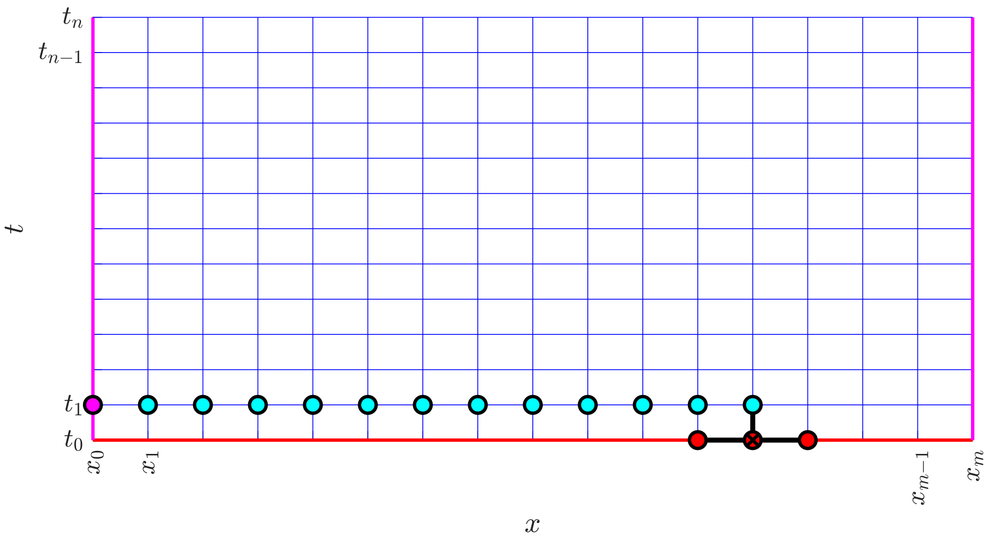
Forward Euler — iteration



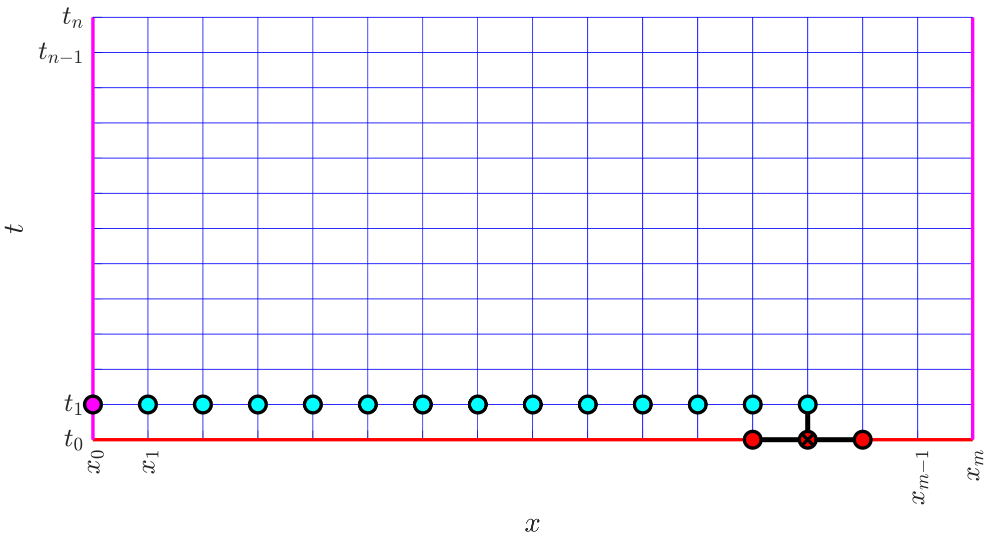
Forward Euler — iteration



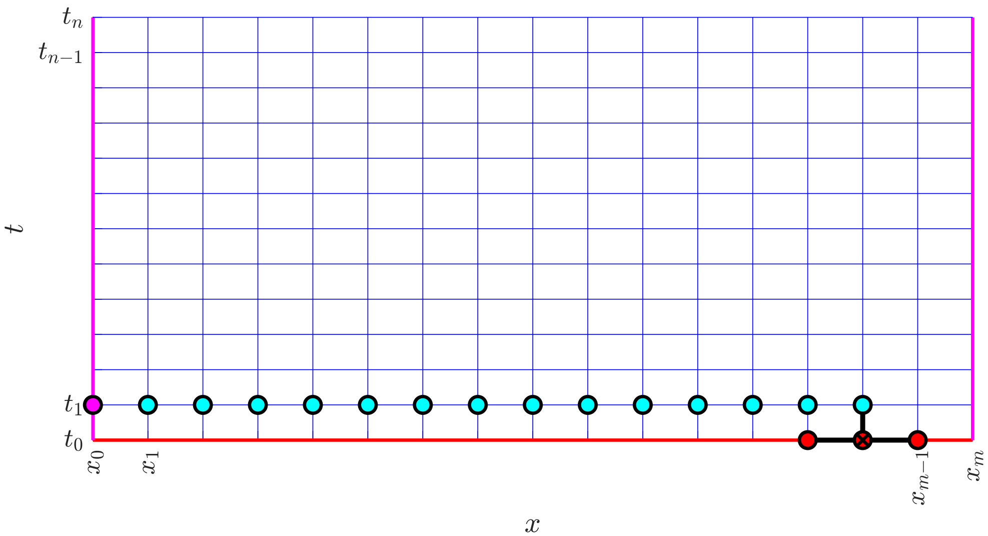
Forward Euler — iteration



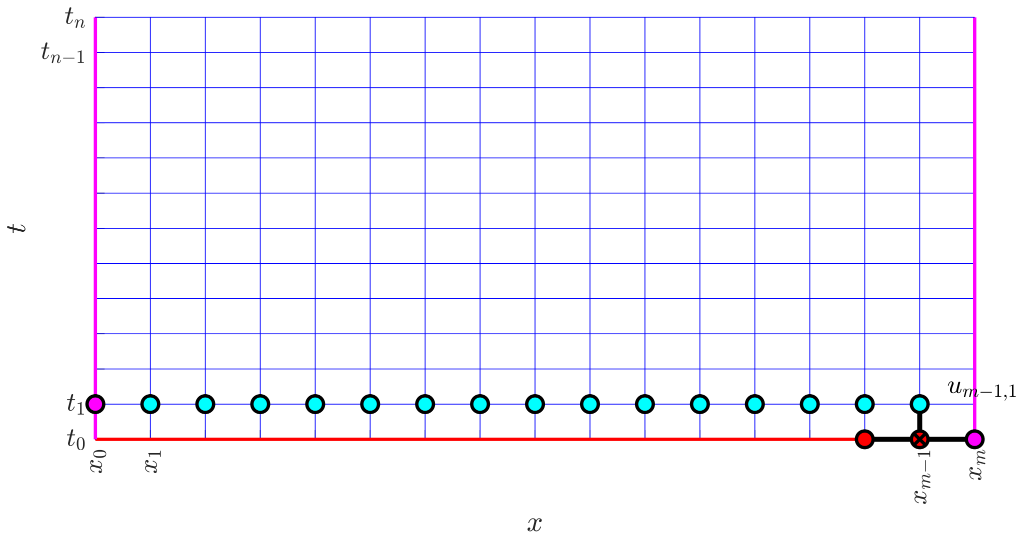
Forward Euler — iteration



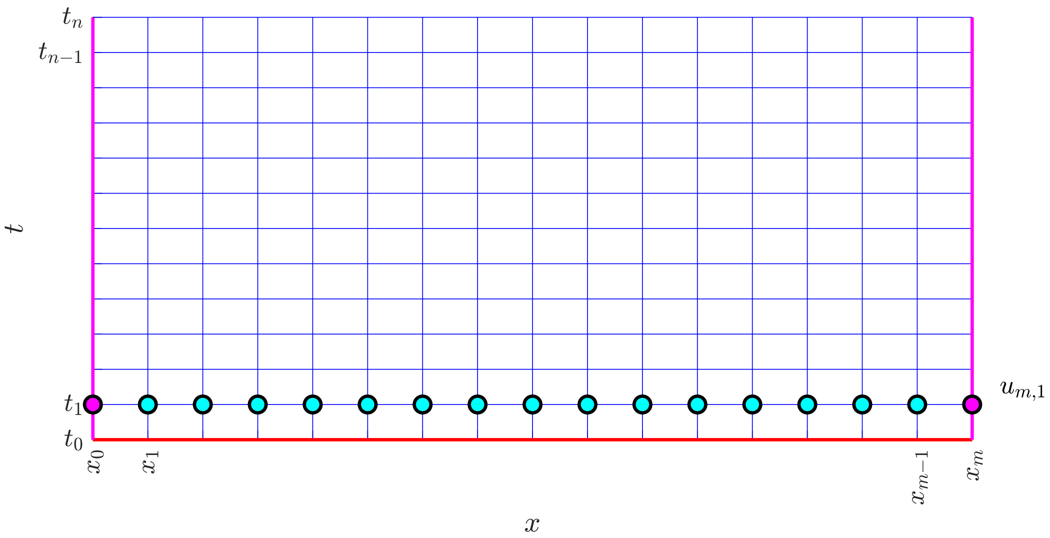
Forward Euler — iteration



Forward Euler — iteration



Forward Euler — iteration



Forward Euler — matrix form

- Can conveniently rewrite forward Euler scheme in matrix/vector form
- Let $\boldsymbol{u}^{(j)} = (u_{1,j}, u_{2,j}, \ldots, u_{m-1,j})^T$. Then

$$\boldsymbol{u}^{(j+1)} = A_{\mathsf{FE}} \boldsymbol{u}^{(j)} \qquad \text{for } j \geqslant 1$$

- A_{FE} is a *tridiagonal* $(m-1) \times (m-1)$ matrix:

$$A_{\mathsf{FE}} = \begin{pmatrix} 1-2\lambda & \lambda & & & & 0 \\ \lambda & 1-2\lambda & \lambda & & & \\ & \lambda & 1-2\lambda & \lambda & & \\ & & \ddots & \ddots & \ddots & \\ & & & \lambda & 1-2\lambda & \lambda \\ 0 & & & & \lambda & 1-2\lambda \end{pmatrix}, \qquad \lambda = \frac{\kappa \Delta t}{\Delta x^2}$$

- Initial conditions: $\boldsymbol{u}^{(0)} = (f(x_1), f(x_2), \ldots, f(x_{m-1}))^T$
- Boundary conditions: $u_{0,j} = u_{m,j} = 0$ for all j

Forward Euler — stability

- Numerical experiments suggest that whether the Forward Euler method gives the right answer depends on the values of Δx and Δt
- Simple calculation shows that errors grow as

$$\boldsymbol{e}^{(j)} = A_{\mathsf{FE}}^{j} \boldsymbol{e}^{(0)}$$

- Errors will decay to zero (i.e. the scheme is *stable*) if all the eigenvalues of A_{FE} are inside the unit circle; true if

$$0 < \lambda < \frac{1}{2} \qquad \Leftrightarrow \qquad \frac{\kappa \Delta t}{\Delta x^2} < \frac{1}{2}$$

- Forward Euler scheme is *conditionally stable* (that's bad)

Alternative schemes

- In order to solve the Forward Euler stability problem we could try alternative discretizations for the time derivative
- E.g., approximate the PDE at the point (x_i, t_{j+1}), and use a backward difference approximation for $\frac{\partial u}{\partial t}$ and central difference for $\frac{\partial^2 u}{\partial x^2}$:

$$\frac{\partial u}{\partial t} = \kappa \frac{\partial^2 u}{\partial x^2}$$

Alternative schemes

- In order to solve the Forward Euler stability problem we could try alternative discretizations for the time derivative
- E.g., approximate the PDE at the point (x_i, t_{j+1}), and use a backward difference approximation for $\frac{\partial u}{\partial t}$ and central difference for $\frac{\partial^2 u}{\partial x^2}$:

$$\frac{\partial u}{\partial t}(x_i, t_{j+1}) = \kappa \frac{\partial^2 u}{\partial x^2}(x_i, t_{j+1})$$

Alternative schemes

- In order to solve the Forward Euler stability problem we could try alternative discretizations for the time derivative
- E.g., approximate the PDE at the point (x_i, t_{j+1}), and use a backward difference approximation for $\frac{\partial u}{\partial t}$ and central difference for $\frac{\partial^2 u}{\partial x^2}$:

$$\frac{u_{i,j+1} - u_{i,j}}{\Delta t} = \kappa \frac{u_{i+1,j+1} - 2u_{i,j+1} + u_{i-1,j+1}}{\Delta x^2}$$

- Rearrange so all the timestep $j+1$ terms are on the left, and timestep j on the right

$$u_{i,j+1} - \lambda \left(u_{i+1,j+1} - 2u_{i,j+1} + u_{i-1,j+1} \right) = u_{i,j}$$
$$\lambda = \kappa \frac{\Delta t}{\Delta x^2}$$

- We can't explicitly solve for $u_{i,j+1}$ in terms of $u_{\star,j}$: an *implicit* method

Alternative schemes

- In order to solve the Forward Euler stability problem we could try alternative discretizations for the time derivative
- E.g., approximate the PDE at the point (x_i, t_{j+1}), and use a backward difference approximation for $\frac{\partial u}{\partial t}$ and central difference for $\frac{\partial^2 u}{\partial x^2}$:

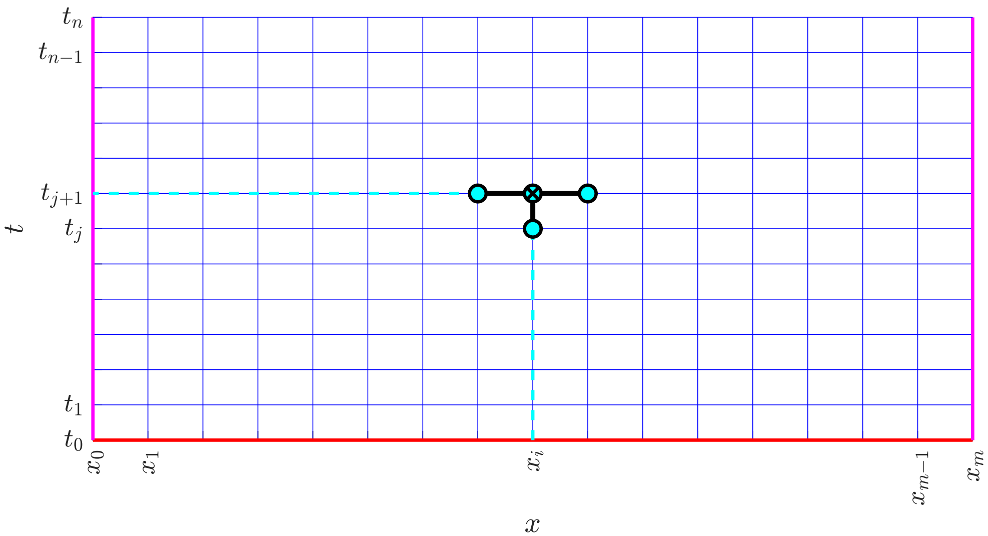
$$\frac{u_{i,j+1} - u_{i,j}}{\Delta t} = \kappa \frac{u_{i+1,j+1} - 2u_{i,j+1} + u_{i-1,j+1}}{\Delta x^2}$$

- Rearrange so all the timestep $j+1$ terms are on the left, and timestep j on the right

$$-\lambda u_{i+1,j+1} + (1 + 2\lambda) u_{i,j+1} - \lambda u_{i-1,j+1} = u_{i,j}$$
$$\lambda = \kappa \frac{\Delta t}{\Delta x^2}$$

- We can't explicitly solve for $u_{i,j+1}$ in terms of $u_{\star,j}$: an *implicit* method

Backward Euler — stencil



Backward Euler — matrix form

- Again write $\boldsymbol{u}^{(j)} = (u_{1,j}, u_{2,j}, \ldots, u_{m-1,j})^T$. Matrix/vector form:

$$A_{\mathsf{BE}} \boldsymbol{u}^{(j+1)} = \boldsymbol{u}^{(j)}$$

  with $(m-1) \times (m-1)$ tridiagonal matrix A_{BE}

$$A_{\mathsf{BE}} = \begin{pmatrix} 1+2\lambda & -\lambda & & & 0 \\ -\lambda & 1+2\lambda & -\lambda & & \\ & -\lambda & 1+2\lambda & -\lambda & \\ & & \ddots & \ddots & \ddots \\ 0 & & & -\lambda & 1+2\lambda \end{pmatrix}$$

- This is the *Backward Euler* scheme. It is an *implicit* method. We must solve a matrix equation at each time step.
- Initial and boundary conditions are as for Forward Euler

Alternative schemes (2)

- Or we could try central differences for the time derivative
- To get $u_{\star,j+1}$ in terms of $u_{\star,j}$, expand the PDE about $(x_i, t_{j+1/2})$

$$\frac{\partial u}{\partial t}(x_i, t_{j+1/2}) = \frac{u_{i,j+1} - u_{i,j}}{\Delta t}$$

- But now the spatial derivative contains terms we don't really want

$$\frac{\partial^2 u}{\partial x^2}(x_i, t_{j+1/2}) = \frac{u_{i+1,j+1/2} - 2u_{i,j+1/2} + u_{i-1,j+1/2}}{\Delta x^2}$$

- Approximate the "half" timestep terms by averaging over the two nearest gridpoints (at times j and $j+1$)

$$\frac{\partial^2 u}{\partial x^2}(x_i, t_{j+1/2}) = \frac{\frac{u_{i+1,j+1} + u_{i+1,j}}{2} - 2\frac{u_{i,j+1} + u_{i,j}}{2} + \frac{u_{i-1,j+1} + u_{i-1,j}}{2}}{\Delta x^2}$$

- Or we could try central differences for the time derivative
- To get $u_{\star,j+1}$ in terms of $u_{\star,j}$, expand the PDE about $(x_i, t_{j+1/2})$

$$\frac{\partial u}{\partial t}(x_i, t_{j+1/2}) = \frac{u_{i,j+1} - u_{i,j}}{\Delta t}$$

- But now the spatial derivative contains terms we don't really want

$$\frac{\partial^2 u}{\partial x^2}(x_i, t_{j+1/2}) = \frac{u_{i+1,j+1/2} - 2u_{i,j+1/2} + u_{i-1,j+1/2}}{\Delta x^2}$$

- Approximate the "half" timestep terms by averaging over the two nearest gridpoints (at times j and $j+1$)

$$\frac{\partial^2 u}{\partial x^2}(x_i, t_{j+1/2}) = \frac{\frac{u_{i+1,j+1} + u_{i+1,j}}{2} - 2\frac{u_{i,j+1} + u_{i,j}}{2} + \frac{u_{i-1,j+1} + u_{i-1,j}}{2}}{\Delta x^2}$$

Alternative schemes (2)

- Or we could try central differences for the time derivative
- To get $u_{\star,j+1}$ in terms of $u_{\star,j}$, expand the PDE about $(x_i, t_{j+1/2})$

$$\frac{\partial u}{\partial t}(x_i, t_{j+1/2}) = \frac{u_{i,j+1} - u_{i,j}}{\Delta t}$$

- But now the spatial derivative contains terms we don't really want

$$\frac{\partial^2 u}{\partial x^2}(x_i, t_{j+1/2}) = \frac{u_{i+1,j+1/2} - 2u_{i,j+1/2} + u_{i-1,j+1/2}}{\Delta x^2}$$

- Approximate the "half" timestep terms by averaging over the two nearest gridpoints (at times j and $j+1$)

$$\frac{\partial^2 u}{\partial x^2}(x_i, t_{j+1/2}) = \frac{\frac{u_{i+1,j+1} + u_{i+1,j}}{2} - 2\frac{u_{i,j+1} + u_{i,j}}{2} + \frac{u_{i-1,j+1} + u_{i-1,j}}{2}}{\Delta x^2}$$

Alternative schemes (2)

- Or we could try central differences for the time derivative
- To get $u_{\star,j+1}$ in terms of $u_{\star,j}$, expand the PDE about $(x_i, t_{j+1/2})$

$$\frac{\partial u}{\partial t}(x_i, t_{j+1/2}) = \frac{u_{i,j+1} - u_{i,j}}{\Delta t}$$

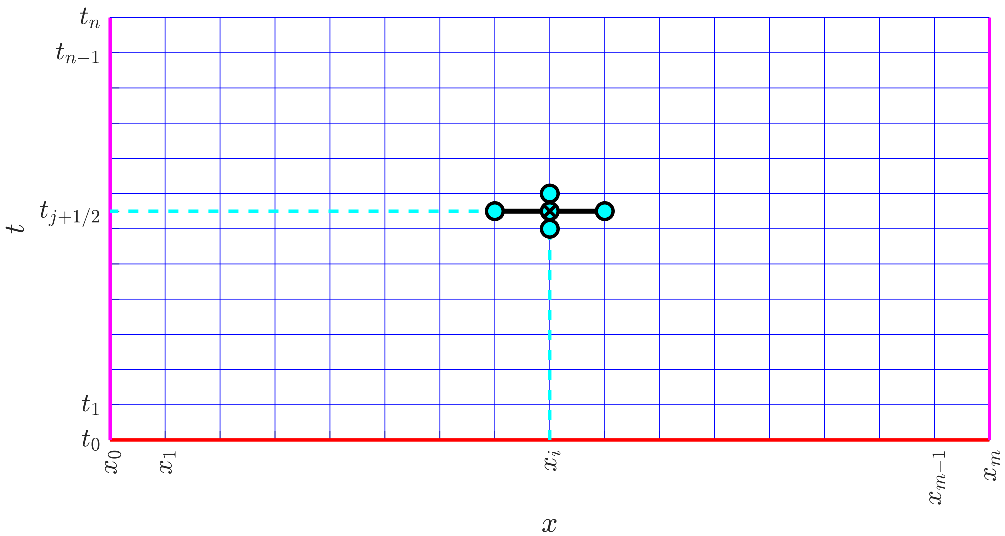
- But now the spatial derivative contains terms we don't really want

$$\frac{\partial^2 u}{\partial x^2}(x_i, t_{j+1/2}) = \frac{u_{i+1,j+1/2} - 2u_{i,j+1/2} + u_{i-1,j+1/2}}{\Delta x^2}$$

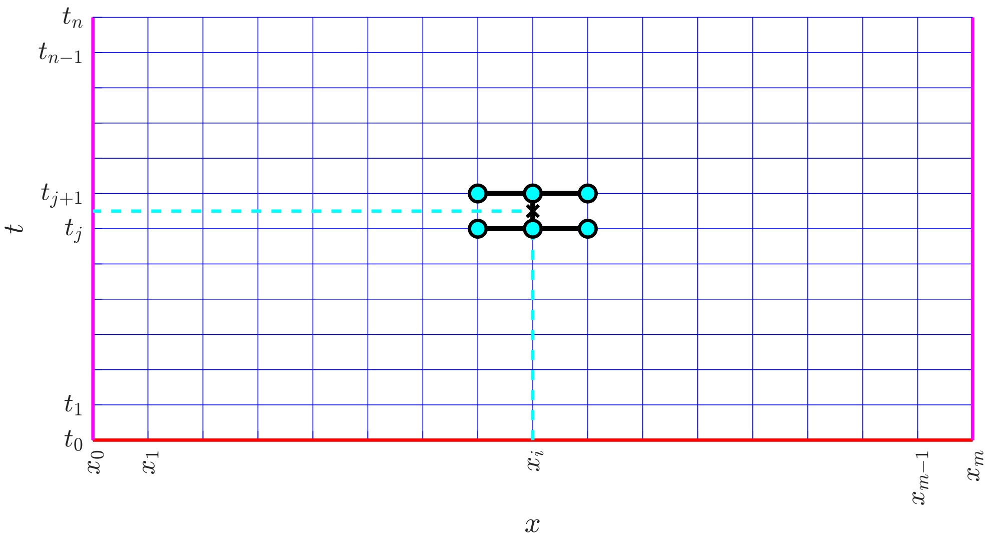
- Approximate the "half" timestep terms by averaging over the two nearest gridpoints (at times j and $j+1$)

$$\frac{\partial^2 u}{\partial x^2}(x_i, t_{j+1/2}) = \frac{\frac{u_{i+1,j+1} + u_{i+1,j}}{2} - 2\frac{u_{i,j+1} + u_{i,j}}{2} + \frac{u_{i-1,j+1} + u_{i-1,j}}{2}}{\Delta x^2}$$

Alternative schemes (2)



Alternative schemes (2)



Alternative schemes (2)

- Putting the pieces together, we get

$$\frac{u_{i,j+1} - u_{i,j}}{\Delta t} = \frac{\kappa}{2}\left[\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2} + \frac{u_{i+1,j+1} - 2u_{i,j+1} + u_{i-1,j+1}}{\Delta x^2}\right]$$

- Rearrange as usual: put timestep $j+1$ terms on the left-hand-side, timestep j terms on the right-hand-side, and set $\lambda = \kappa\Delta t/\Delta x^2$, to get

$$u_{i,j+1} - \frac{\lambda}{2}\left[u_{i+1,j+1} - 2u_{i,j+1} + u_{i-1,j+1}\right] = u_{i,j} + \frac{\lambda}{2}\left[u_{i+1,j} - 2u_{i,j} + u_{i-1,j}\right]$$

- Called the *Crank-Nicholson* scheme
- Like backward Euler it's an *implicit* method; we have to solve a matrix equation at each time step to find $\boldsymbol{u}^{(j+1)}$

Alternative schemes (2)

- Putting the pieces together, we get

$$\frac{u_{i,j+1} - u_{i,j}}{\Delta t} = \frac{\kappa}{2}\left[\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2} + \frac{u_{i+1,j+1} - 2u_{i,j+1} + u_{i-1,j+1}}{\Delta x^2}\right]$$

- Rearrange as usual: put timestep $j+1$ terms on the left-hand-side, timestep j terms on the right-hand-side, and set $\lambda = \kappa\Delta t/\Delta x^2$, to get

$$-\frac{\lambda}{2}u_{i+1,j+1} + (1+\lambda)u_{i,j+1} - \frac{\lambda}{2}u_{i-1,j+1} = \frac{\lambda}{2}u_{i+1,j} + (1-\lambda)u_{i,j} + \frac{\lambda}{2}u_{i-1,j}$$

- Called the *Crank-Nicholson* scheme
- Like backward Euler it's an *implicit* method; we have to solve a matrix equation at each time step to find $\boldsymbol{u}^{(j+1)}$

- In matrix/vector form, the Crank-Nicholson scheme is

$$A_{\mathsf{CN}}\boldsymbol{u}^{(j+1)} = B_{\mathsf{CN}}\boldsymbol{u}^{(j)}$$

- As usual, $\boldsymbol{u}^{(j)} = (u_{1,j}, \ldots, u_{m-1,j})^T$
- This time there are *two* $(m-1) \times (m-1)$ tridiagonal matrices

$$A_{\mathsf{CN}} = \mathrm{tridiag}\left(-\frac{\lambda}{2}, 1+\lambda, -\frac{\lambda}{2}\right), \quad B_{\mathsf{CN}} = \mathrm{tridiag}\left(\frac{\lambda}{2}, 1-\lambda, \frac{\lambda}{2}\right)$$

- Initial and boundary conditions implemented as usual

Stability

- Is any of this worth it?
- Yes! Both backward Euler and Crank-Nicholson schemes are *unconditionally stable*
- Recall: forward Euler is only *conditionally stable*; stability criterion is

$$\lambda = \kappa \frac{\Delta t}{\Delta x^2} < \frac{1}{2}$$

- However, there is a price to pay: solving a matrix equation at each time step
- We must be careful not to make the algorithm inefficient as a result

Error/accuracy

- There is also a benefit to the extra work involved in coding a Crank-Nicholson solver
- Straightforward error analysis (Taylor series) shows that, to leading order
  - forward Euler has truncation error $E = C_t \Delta t + C_x \Delta x^2$
  - backward Euler has truncation error $E = C_t \Delta t + C_x \Delta x^2$
  - Crank-Nicholson has truncation error $E = C_t \Delta t^2 + C_x \Delta x^2$
- The Crank-Nicholson scheme is *second order* accurate in time; i.e. it converges to the true solution quicker as $\Delta t \to 0$

Implementation: solving $A\boldsymbol{x} = \boldsymbol{b}$

- *What you should not do*: multiply by the inverse. It's very expensive, computationally
- For an $n \times n$ matrix
  - matrix inversion takes $\mathcal{O}(n^3)$ operations
  - multiplying two matrices together takes $\mathcal{O}(n^3)$ operations
  - multiplying a matrix and a vector takes $\mathcal{O}(n^2)$ operations
- Using a linear solver is much better
  - linear solve typically takes $\mathcal{O}(n^2)$ operations
  - if A is tridiagonal, the Thomas algorithm[2] finds a solution in $\mathcal{O}(n)$ operations:

[2] see, e.g., https://en.wikipedia.org/wiki/Tridiagonal_matrix_algorithm

Implementation: sparse matrices

- There's also a lot of wasted memory (most our matrix entries are zeros)
- There's a whole suite of methods in Python (and every other serious programming language) to take advantage of the fact that our matrices are *sparse* (almost all the entries are zero)
- With even only a couple of hundred nodes in space, code that uses sparse operations can run tens of thousands times faster, and reduce storage by factors of hundreds!
- `scipy.sparse` is the package to use
  - `scipy.sparse.diags` to define a sparse diagonal matrix
  - `scipy.sparse.linalg.spsolve` to solve a sparse linear system
  - `A.dot(v)` as usual to multiply sparse matrix `A` and (regular) vector `v` (NB `numpy.dot` doesn't support sparse matrices)

Recap

- We've derived three finite difference methods to solve the PDE problem (1)–(2)
- In component form, for $i = 1, 2, \ldots, m-1$, $j = 1, 2, \ldots, n-1$:
  1. Forward Euler (discretize PDE at (x_i, t_j), forward difference $\partial u/\partial t$):

$$u_{i,j+1} = u_{i,j} + \lambda \left(u_{i+1,j} - 2u_{i,j} + u_{i-1,j} \right)$$

  2. Backward Euler (discretize PDE at (x_i, t_{j+1}), backward difference $\partial u/\partial t$):

$$u_{i,j+1} - \lambda \left(u_{i+1,j+1} - 2u_{i,j+1} + u_{i-1,j+1} \right) = u_{i,j}$$

  3. Crank Nicholson (discretize PDE at $(x_i, t_{j+1/2})$), central difference $\partial u/\partial t$):

$$u_{i,j+1} - \frac{\lambda}{2} \left(u_{i+1,j+1} - 2u_{i,j+1} + u_{i-1,j+1} \right) = u_{i,j} + \frac{\lambda}{2} \left(u_{i+1,j} - 2u_{i,j} + u_{i-1,j} \right)$$

- $\lambda = \kappa \Delta x / \Delta t^2$
- Boundary conditions: $u_{0,j} = u_{m,j} = 0$ $(j = 1, 2, \ldots, n)$

Extensions

- It'd be nice to extend the range of systems we can solve
- Some (reasonably) straightforward modifications
  - other boundary conditions; e.g.

$$u(0,t) = p(t), \quad \text{or} \quad \frac{\partial u}{\partial x}(0,t) = q(t)$$

  - a heat source

$$\frac{\partial u}{\partial t} = \kappa \frac{\partial^2 u}{\partial x^2} + F(x,t)$$

  - variable diffusion coefficient

$$\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left(\kappa(x)\frac{\partial u}{\partial x}\right)$$

  - more general parabolic PDE problems
- We'll illustrate these in the case of forward Euler
- Other schemes can be adapted similarly, by taking care to remember at which point(s) the PDE is being discretized

Non-homogeneous Dirichlet boundary conditions

- General non-homogeneous Dirichlet boundary conditions are

$$u(0,t) = p(t), \qquad u(L,t) = q(t) \qquad \text{for all } t > 0$$

- Forward Euler discretization is still valid; for $i = 1, \dots, m-1$

$$u_{i,j+1} = u_{i,j} + \lambda(u_{i-1,j} - 2u_{i,j} + u_{i+1,j})$$

- For $i = 1, m-1$ this refers to $u_{0,j}$ and $u_{m,j}$: the value of u at $x = x_0 = 0$ and $x = x_m = L$, time t_j:

$$u_{0,j} = u(0,t_j) = p(t_j) = p_j, \quad u_{m,j} = u(L,t_j) = q(t_j) = q_j$$

- Grid points next to boundaries get information from the boundary

Non-homogeneous Dirichlet boundary conditions

- General non-homogeneous Dirichlet boundary conditions are

$$u(0,t) = p(t), \qquad u(L,t) = q(t) \qquad \text{for all } t > 0$$

- Forward Euler discretization is still valid; for $i = 1, \ldots, m-1$

$$u_{i,j+1} = u_{i,j} + \lambda(u_{i-1,j} - 2u_{i,j} + u_{i+1,j})$$

- For $i = 1, m-1$ this refers to $u_{0,j}$ and $u_{m,j}$: the value of u at $x = x_0 = 0$ and $x = x_m = L$, time t_j:

$$u_{0,j} = u(0,t_j) = p(t_j) = p_j, \qquad u_{m,j} = u(L,t_j) = q(t_j) = q_j$$

- Grid points next to boundaries get information from the boundary

Non-homogeneous Dirichlet boundary conditions

- General non-homogeneous Dirichlet boundary conditions are

$$u(0,t) = p(t), \qquad u(L,t) = q(t) \qquad \text{for all } t > 0$$

- Forward Euler discretization is still valid; for $i = 1, \dots, m-1$

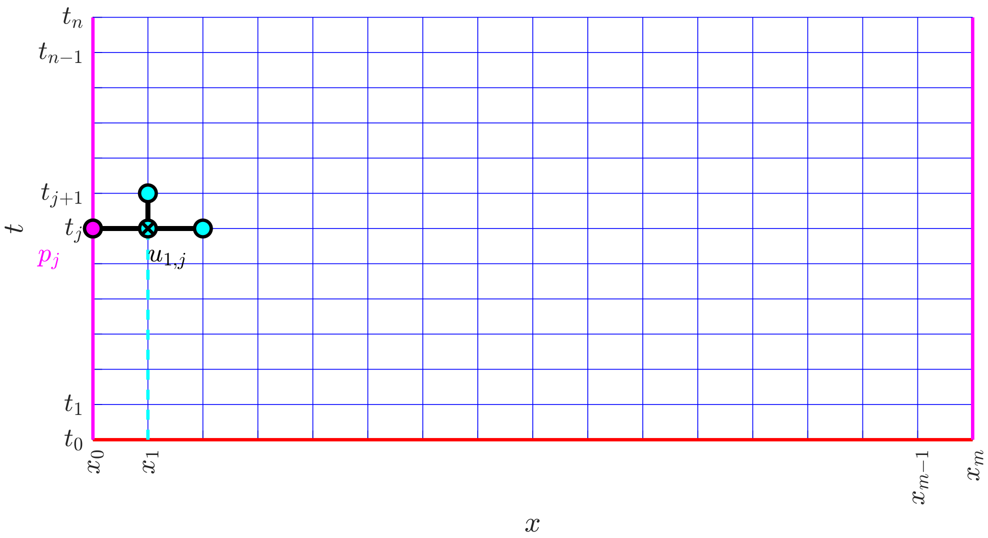
$$u_{i,j+1} = u_{i,j} + \lambda(u_{i-1,j} - 2u_{i,j} + u_{i+1,j})$$

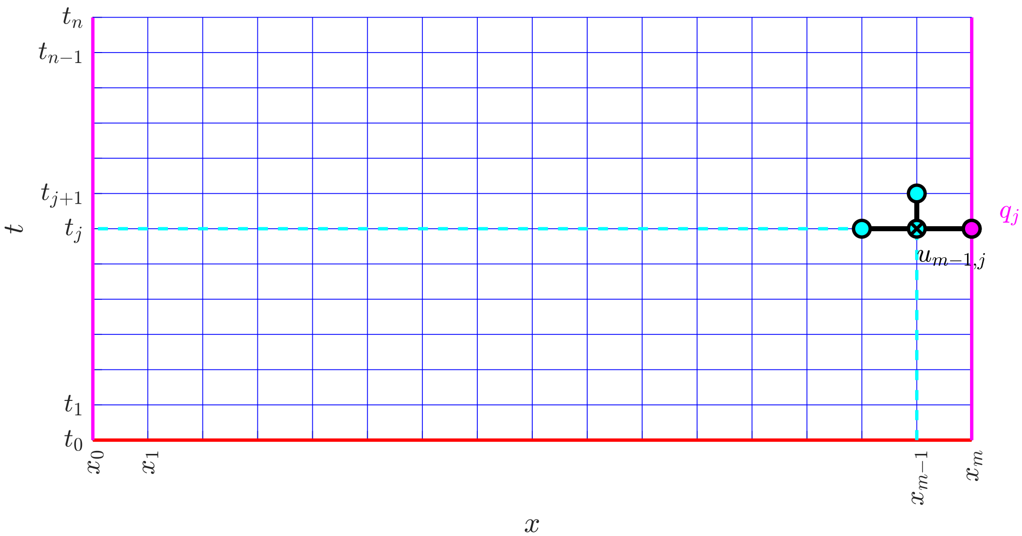
- For $i = 1, m-1$ this refers to $u_{0,j}$ and $u_{m,j}$: the value of u at $x = x_0 = 0$ and $x = x_m = L$, time t_j:

$$u_{0,j} = u(0,t_j) = p(t_j) = p_j, \qquad u_{m,j} = u(L,t_j) = q(t_j) = q_j$$

- Grid points next to boundaries get information from the boundary

Dirichlet grid





t_n
t_{n-1}
t_{j+1}
t
t_j
t_1
t_0
q_j
$u_{m-1,j}$
x_0
x_1
x_{m-1}
x_m
x

Component form

- In component form, we get

$$
\begin{aligned}
u_{1,j+1} &= u_{1,j} + \lambda(-2u_{1,j} + u_{2,j}) + \lambda p_j && i = 1 \\
u_{i,j+1} &= u_{i,j} + \lambda(u_{i-1,j} - 2u_{i,j} + u_{i+1,j}) && i = 2, \ldots, m-2 \\
u_{m-1,j+1} &= u_{m-1,j} + \lambda(u_{m-2,j} - 2u_{m-1,j}) + \lambda q_j && i = m-1
\end{aligned}
$$

- p_j and q_j come from the Dirichlet boundary conditions

$$
p_j = p(t_j) = u(0, t_j), \qquad q_j = q(t_j) = u(L, t_j)
$$

- Dirichlet boundary conditions give rise to two additive terms, in the equations next to the boundary

Matrix form

- Writing as a matrix equation as usual, $\boldsymbol{u}^{(j)} = (u_{1,j}, \ldots, u_{m-1,j})^T$,

$$\boldsymbol{u}^{(j+1)} = A_{\mathsf{FE}}\boldsymbol{u}^{(j)} + \lambda \begin{pmatrix} p_j \\ 0 \\ \vdots \\ 0 \\ q_j \end{pmatrix}, \qquad p_j = p(t_j), q_j = q(t_j)$$

- A_{FE} is the usual $(m-1) \times (m-1)$ forward Euler tridiagonal matrix

$$A_{\mathsf{FE}} = \mathrm{tridiag}\,(\lambda, 1-2\lambda, \lambda)$$

- Boundary conditions give us an extra (known) RHS vector
- Must keep the boundary nodes up to date too: $u_{0,j} = p_j, u_{m,j} = q_j$
- Scheme is still only *conditionally stable*

Neumann boundary conditions

- Suppose that $\frac{\partial u}{\partial n}$ is given at the boundaries $x = 0, L$, i.e.,

$$\frac{\partial u}{\partial x}(0,t) = P(t), \qquad \frac{\partial u}{\partial x}(L,t) = Q(t)$$

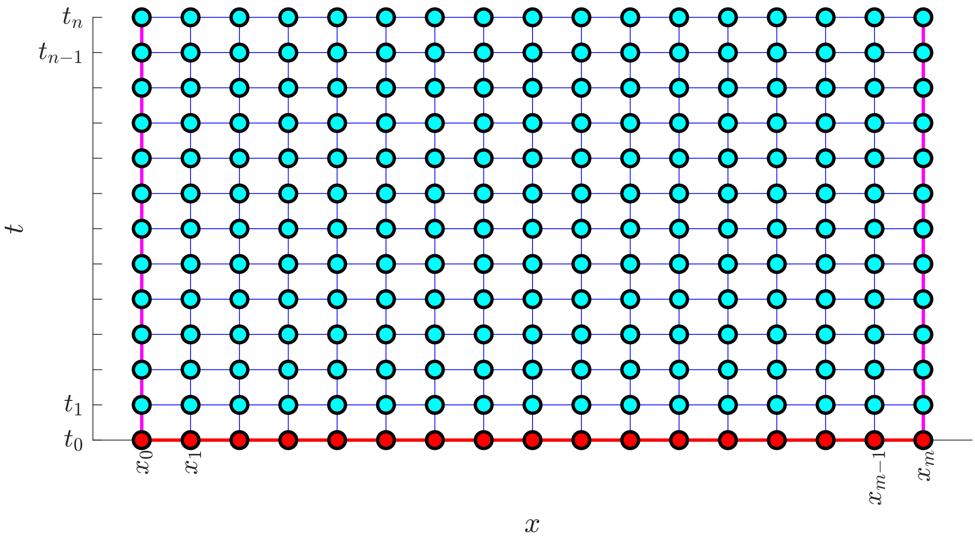
- The values $u_{0,j} = u(0,t)$ and $u_{m,j} = u(L,t)$ are no longer prescribed: must solve for them at each time step too
- The Forward Euler discretization still applies, now for all $i = 0, \ldots, m$

$$u_{i,j+1} = u_{i,j} + \lambda(u_{i-1,j} - 2u_{i,j} + u_{i+1,j})$$

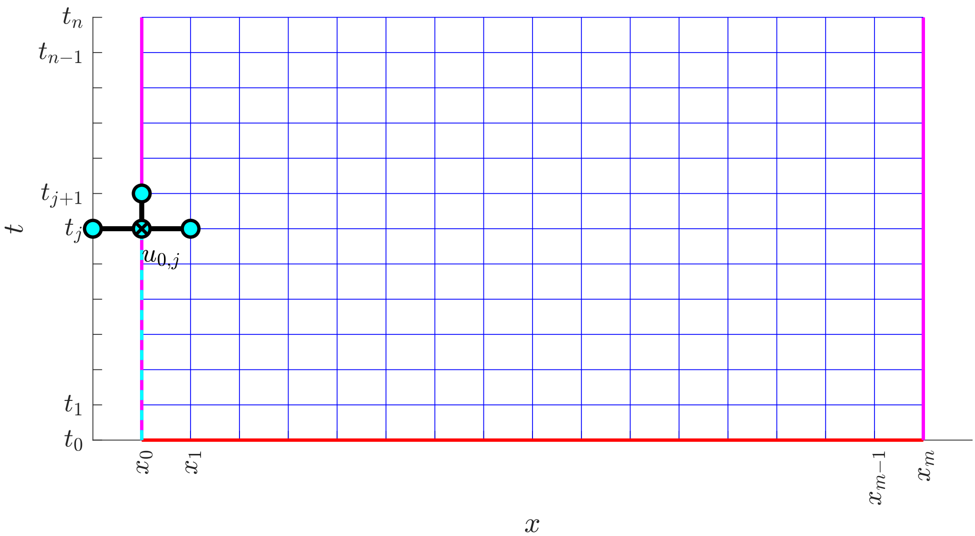
- At the boundaries, $i = 0, m$, we need to know $u_{-1,j}$ and $u_{m+1,j}$?!

$$\begin{aligned} u_{0,j+1} &= u_{0,j} + \lambda(u_{-1,j} - 2u_{0,j} + u_{1,j}) \\ u_{m,j+1} &= u_{m,j} + \lambda(u_{m-1,j} - 2u_{m,j} + u_{m+1,j}) \end{aligned}$$

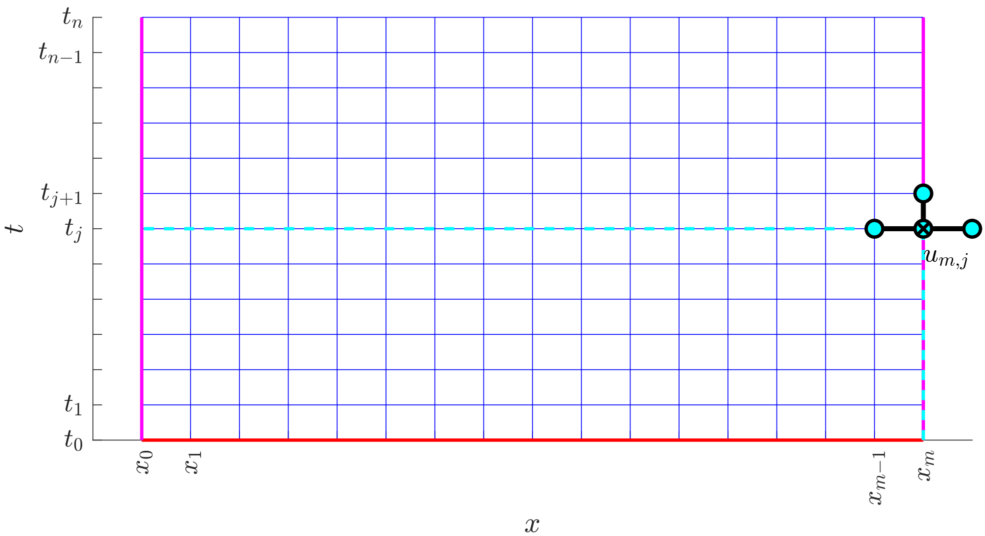
Neumann boundary conditions — grid



Neumann boundary conditions — grid



Neumann boundary conditions — grid



Removing the fictitious nodes

- We can use central difference to discretize the (gradient) boundary conditions

$$\frac{\partial u}{\partial x}(x_i, t_j) \approx \frac{u(x_i + \Delta x, t_j) - u(x_i - \Delta x, t_j)}{2\Delta x} = \frac{u_{i+1,j} - u_{i-1,j}}{2\Delta x}$$

- So, for example, at $i = 0$, we get

$$P_j = \frac{u_{1,j} - u_{-1,j}}{2\Delta x}, \qquad \text{where } P_j = P(t_j) = \frac{\partial u}{\partial x}(0, t_j)$$

- Rearranging gives us an expression for $u_{-1,j}$

$$u_{-1,j} = u_{1,j} - 2\Delta x P_j$$

- So forward difference at $i = 0$ becomes

$$u_{0,j+1} = u_{0,j} + \lambda(u_{-1,j} - 2u_{0,j} + u_{1,j})$$

Removing the fictitious nodes

- We can use central difference to discretize the (gradient) boundary conditions

$$\frac{\partial u}{\partial x}(x_i, t_j) \approx \frac{u(x_i + \Delta x, t_j) - u(x_i - \Delta x, t_j)}{2\Delta x} = \frac{u_{i+1,j} - u_{i-1,j}}{2\Delta x}$$

- So, for example, at $i = 0$, we get

$$P_j = \frac{u_{1,j} - u_{-1,j}}{2\Delta x}, \qquad \text{where } P_j = P(t_j) = \frac{\partial u}{\partial x}(0, t_j)$$

- Rearranging gives us an expression for $u_{-1,j}$

$$u_{-1,j} = u_{1,j} - 2\Delta x P_j$$

- So forward difference at $i = 0$ becomes

$$u_{0,j+1} = u_{0,j} + \lambda(-2u_{0,j} + 2u_{1,j}) - 2\lambda\Delta x P_j$$

Component form

- The complete forward difference discretization of the PDE is

$$
\begin{aligned}
u_{0,j+1} &= u_{0,j} + \lambda(-2u_{0,j} + 2u_{1,j}) - 2\lambda\Delta x P_j && i = 0 \\
u_{i,j+1} &= u_{i,j} + \lambda(u_{i-1,j} - 2u_{i,j} + u_{i+1,j}) && i = 1, \ldots, m-1 \\
u_{m,j+1} &= u_{m,j} + \lambda(2u_{m-1,j} - 2u_{m,j}) + 2\lambda\Delta x Q_j && i = m
\end{aligned}
$$

- P_j and Q_j come from the Neumann boundary conditions

$$P_j = P(t_j) = \frac{\partial u}{\partial x}(0, t_j), \quad Q_j = Q(t_j) = \frac{\partial u}{\partial x}(L, t_j)$$

- Differences from our previous discretizations
  - $u_{0,j}$ and $u_{m,j}$ are unknowns, to be solved for
  - Neumann boundary conditions create two additional terms, and modified coefficients

Component form

- The complete forward difference discretization of the PDE is

$$
\begin{aligned}
u_{0,j+1} &= u_{0,j} + \lambda(-2u_{0,j} + 2u_{1,j}) - 2\lambda\Delta x P_j && i = 0 \\
u_{i,j+1} &= u_{i,j} + \lambda(u_{i-1,j} - 2u_{i,j} + u_{i+1,j}) && i = 1, \ldots, m-1 \\
u_{m,j+1} &= u_{m,j} + \lambda(2u_{m-1,j} - 2u_{m,j}) + 2\lambda\Delta x Q_j && i = m
\end{aligned}
$$

- P_j and Q_j come from the Neumann boundary conditions

$$
P_j = P(t_j) = \frac{\partial u}{\partial x}(0, t_j), \quad Q_j = Q(t_j) = \frac{\partial u}{\partial x}(L, t_j)
$$

- Differences from our previous discretizations
  - $u_{0,j}$ and $u_{m,j}$ are unknowns, to be solved for
  - Neumann boundary conditions create two additional terms, and modified coefficients

Component form

- The complete forward difference discretization of the PDE is

$$
\begin{aligned}
u_{0,j+1} &= u_{0,j} + \lambda(-2u_{0,j} + 2u_{1,j}) - 2\lambda\Delta x P_j && i = 0 \\
u_{i,j+1} &= u_{i,j} + \lambda(u_{i-1,j} - 2u_{i,j} + u_{i+1,j}) && i = 1, \ldots, m-1 \\
u_{m,j+1} &= u_{m,j} + \lambda(2u_{m-1,j} - 2u_{m,j}) + 2\lambda\Delta x Q_j && i = m
\end{aligned}
$$

- P_j and Q_j come from the Neumann boundary conditions

$$
P_j = P(t_j) = \frac{\partial u}{\partial x}(0, t_j), \quad Q_j = Q(t_j) = \frac{\partial u}{\partial x}(L, t_j)
$$

- Differences from our previous discretizations
  - $u_{0,j}$ and $u_{m,j}$ are unknowns, to be solved for
  - Neumann boundary conditions create two additional terms, and modified coefficients

Matrix form

- As a matrix equation

$$\boldsymbol{u}^{(j+1)} = \overline{A}_{\mathsf{FE}}\boldsymbol{u}^{(j)} + 2\lambda\Delta x(-P_j, 0, \ldots, 0, Q_j)^T$$

with $\boldsymbol{u}^{(j)} = (u_{0,j}, u_{1,j}, \ldots, u_{m-1,j}, u_{m,j})^T$ and

$$\overline{A}_{\mathsf{FE}} = \begin{pmatrix} 1-2\lambda & 2\lambda & & & & 0 \\ \lambda & 1-2\lambda & \lambda & & & \\ & \lambda & 1-2\lambda & \lambda & & \\ & & \ddots & \ddots & \ddots & \\ & & & \lambda & 1-2\lambda & \lambda \\ 0 & & & & 2\lambda & 1-2\lambda \end{pmatrix}$$

- Neumann boundary conditions appear as a RHS vector, and modify the evolution matrix
- Note that $\boldsymbol{u}^{(j)}$ now has $m+1$ entries, and $\overline{A}_{\mathsf{FE}}$ is a $(m+1)\times(m+1)$ matrix

Matrix form

- As a matrix equation

$$\boldsymbol{u}^{(j+1)} = \overline{A}_{\mathsf{FE}}\boldsymbol{u}^{(j)} + 2\lambda\Delta x(-P_j, 0, \ldots, 0, Q_j)^T$$

with $\boldsymbol{u}^{(j)} = (u_{0,j}, u_{1,j}, \ldots, u_{m-1,j}, u_{m,j})^T$ and

$$\overline{A}_{\mathsf{FE}} = \begin{pmatrix} 1-2\lambda & 2\lambda & & & & 0 \\ \lambda & 1-2\lambda & \lambda & & & \\ & \lambda & 1-2\lambda & \lambda & & \\ & & \ddots & \ddots & \ddots & \\ & & & \lambda & 1-2\lambda & \lambda \\ 0 & & & & 2\lambda & 1-2\lambda \end{pmatrix}$$

- Neumann boundary conditions appear as a RHS vector, and modify the evolution matrix
- Note that $\boldsymbol{u}^{(j)}$ now has $m+1$ entries, and $\overline{A}_{\mathsf{FE}}$ is a $(m+1) \times (m+1)$ matrix

Matrix form

- As a matrix equation

$$\boldsymbol{u}^{(j+1)} = \overline{A}_{\mathsf{FE}}\boldsymbol{u}^{(j)} + 2\lambda\Delta x(-P_j, 0, \ldots, 0, Q_j)^T$$

with $\boldsymbol{u}^{(j)} = (u_{0,j}, u_{1,j}, \ldots, u_{m-1,j}, u_{m,j})^T$ and

$$\overline{A}_{\mathsf{FE}} = \begin{pmatrix} 1-2\lambda & 2\lambda & & & & 0 \\ \lambda & 1-2\lambda & \lambda & & & \\ & \lambda & 1-2\lambda & \lambda & & \\ & & \ddots & \ddots & \ddots & \\ & & & \lambda & 1-2\lambda & \lambda \\ 0 & & & & 2\lambda & 1-2\lambda \end{pmatrix}$$

- Neumann boundary conditions appear as a RHS vector, and modify the evolution matrix
- Note that $\boldsymbol{u}^{(j)}$ now has $m+1$ entries, and $\overline{A}_{\mathsf{FE}}$ is a $(m+1) \times (m+1)$ matrix

Periodic boundary conditions

- Periodic boundary conditions are often used when simulating travelling solutions in large domains:

$$u(0,t) = u(L,t) \qquad \text{for all } t$$

- Straightforward to implement; in components, the boundary conditions are

$$u_{0,j} = u_{m,j} \qquad \text{for all } j$$

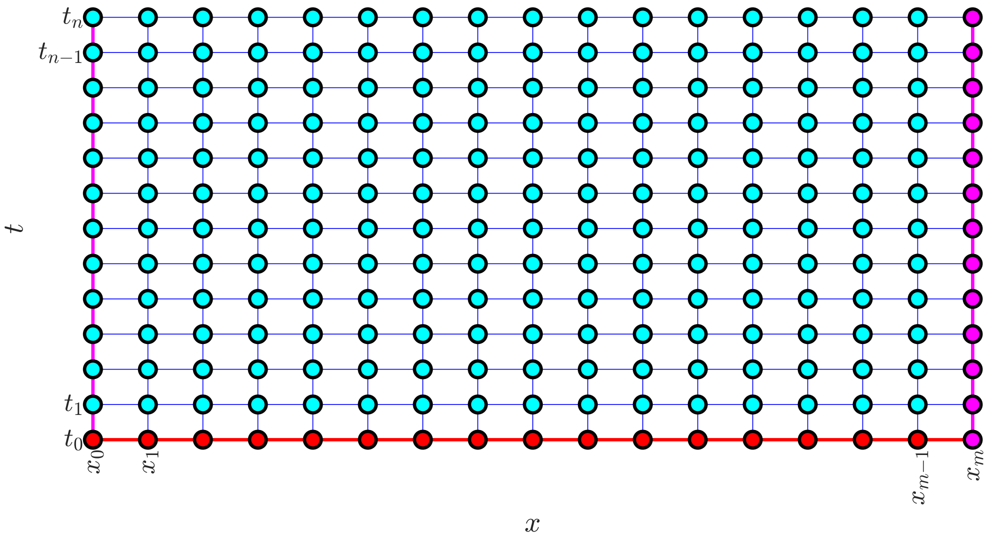
- Leaves m unknowns to find, per time-step: $u_{0,j}$, $u_{1,j}$, $\ldots u_{m-1,j}$
- Usual forward Euler update rule for $i = 1, 2, \ldots, m-2$:

$$u_{i,j+1} = u_{i,j} + \lambda(u_{i-1,j} - 2u_{i,j} + u_{i+1,j})$$

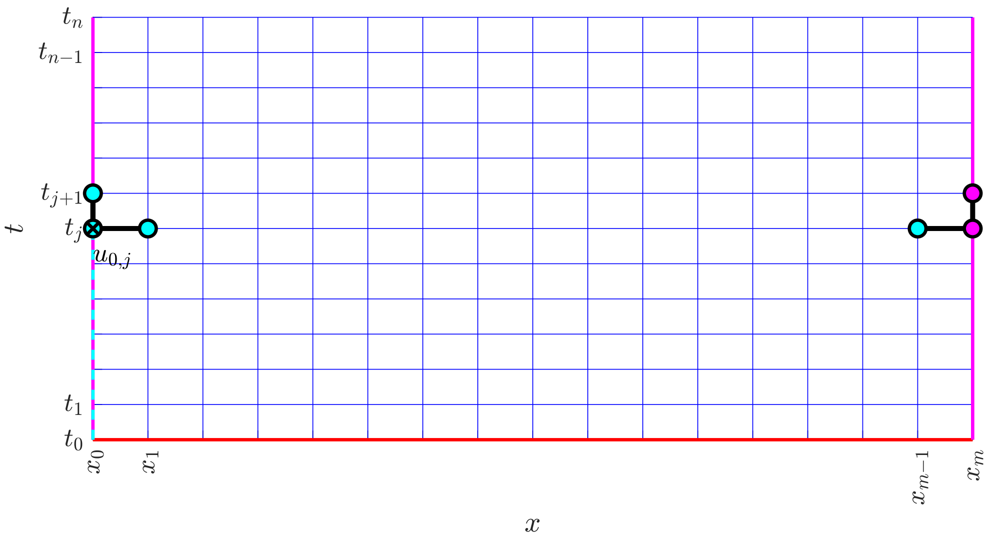
- At the two ends, $i = 0, m-1$, the nodes wrap around:

$$\begin{aligned} u_{0,j+1} &= u_{0,j} + \lambda(u_{m-1,j} - 2u_{0,j} + u_{1,j}) \\ u_{m-1,j+1} &= u_{m,j} + \lambda(u_{m-2,j} - 2u_{m-1,j} + u_{0,j}) \end{aligned}$$

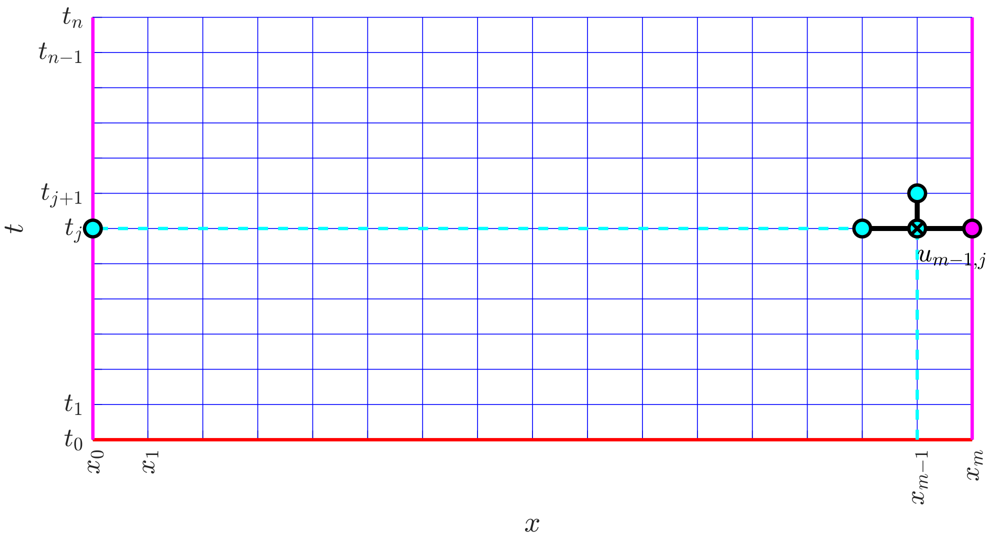
Periodic boundary conditions — grid



Periodic boundary conditions — grid



Periodic boundary conditions — grid



Periodic boundary conditions — matrix form

- In matrix form, we get
$$\boldsymbol{u}^{(j+1)} = \tilde{A}_{\mathsf{FE}}\boldsymbol{u}^{(j)}$$
with $\boldsymbol{u}^{(j)} = (u_{0,j}, u_{1,j}, \ldots, u_{m-1,j}, u_{m-1,j})^T$ and
$$\tilde{A}_{\mathsf{FE}} = \begin{pmatrix} 1-2\lambda & \lambda & & & & \lambda \\ \lambda & 1-2\lambda & \lambda & & & \\ & \lambda & 1-2\lambda & \lambda & & \\ & & \ddots & \ddots & \ddots & \\ & & & \lambda & 1-2\lambda & \lambda \\ \lambda & & & & \lambda & 1-2\lambda \end{pmatrix}$$
- Periodic boundary conditions appear as a modified evolution matrix
- The matrix is now no longer tridiagonal, though it is still sparse
- Note that $\boldsymbol{u}^{(j)}$ now has m entries, and $\tilde{A}_{\mathsf{FE}}$ is a $m \times m$ matrix

Periodic boundary conditions — matrix form

- In matrix form, we get

$$\boldsymbol{u}^{(j+1)} = \tilde{A}_{\mathsf{FE}}\boldsymbol{u}^{(j)}$$

with $\boldsymbol{u}^{(j)} = (u_{0,j}, u_{1,j}, \ldots, u_{m-1,j}, u_{m-1,j})^T$ and

$$\tilde{A}_{\mathsf{FE}} = \begin{pmatrix} 1-2\lambda & \lambda & & & & \lambda \\ \lambda & 1-2\lambda & \lambda & & & \\ & \lambda & 1-2\lambda & \lambda & & \\ & & \ddots & \ddots & \ddots & \\ & & & \lambda & 1-2\lambda & \lambda \\ \lambda & & & & \lambda & 1-2\lambda \end{pmatrix}$$

- Periodic boundary conditions appear as a modified evolution matrix
- The matrix is now no longer tridiagonal, though it is still sparse
- Note that $\boldsymbol{u}^{(j)}$ now has m entries, and $\tilde{A}_{\mathsf{FE}}$ is a $m \times m$ matrix

Adding a right-hand-side function

- Consider the heat equation with a source term, independent of u

$$\frac{\partial u}{\partial t} = \kappa \frac{\partial^2 u}{\partial x^2} + F(x,t)$$

- Standard forward Euler discretization at (x_i, t_j):

$$\frac{u_{i,j+1} - u_{i,j}}{\Delta t} = \kappa \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2} + F(x_i, t_j)$$

- Writing as a matrix equation

$$\boldsymbol{u}^{(j+1)} = A_{\mathsf{FE}} \boldsymbol{u}^{(j)} + \Delta t \, \boldsymbol{F}^{(j)}$$

- A_{FE} = usual forward Euler matrix, and $\boldsymbol{F}_i^{(j)} = F_{i,j} = F(x_i, t_j)$

Adding a right-hand-side function

- Consider the heat equation with a source term, independent of u

$$\frac{\partial u}{\partial t} = \kappa \frac{\partial^2 u}{\partial x^2} + F(x,t)$$

- Standard forward Euler discretization at (x_i, t_j), set $\lambda = \kappa \Delta x / \Delta t^2$, $F_{i,j} = F(x_i, t_j)$:

$$u_{i,j+1} = u_{i,j} + \lambda \left(u_{i+1,j} - 2u_{i,j} + u_{i-1,j} \right) + \Delta t \, F_{i,j}$$

- Writing as a matrix equation

$$\boldsymbol{u}^{(j+1)} = A_{\mathsf{FE}} \boldsymbol{u}^{(j)} + \Delta t \, \boldsymbol{F}^{(j)}$$

- A_{FE} = usual forward Euler matrix, and $\boldsymbol{F}_i^{(j)} = F_{i,j} = F(x_i, t_j)$

Variable diffusion coefficient

- Consider the heat equation with a variable diffusion coefficient

$$\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left[\kappa(x)\frac{\partial u}{\partial x}\right]$$

- Discretise the RHS at (x_i, t_j) in two steps, using central difference with spatial difference $\pm\Delta x/2$ for each derivative
- Outer space derivative

$$\frac{\partial}{\partial x}\left(\kappa(x)\frac{\partial u}{\partial x}\right)\bigg|_{x_i,t_j} = \frac{\left(\kappa(x)\frac{\partial u}{\partial x}\right)\Big|_{x_i+\frac{\Delta x}{2},t_j} - \left(\kappa(x)\frac{\partial u}{\partial x}\right)\Big|_{x_i-\frac{\Delta x}{2},t_j}}{\Delta x}$$

$$= \frac{\kappa(x_i + \frac{\Delta x}{2})\frac{\partial u}{\partial x}\Big|_{x_i+\frac{\Delta x}{2},t_j} - \kappa(x_i - \frac{\Delta x}{2})\frac{\partial u}{\partial x}\Big|_{x_i-\frac{\Delta x}{2},t_j}}{\Delta x}$$

- Can evaluate the diffusion coefficient $\kappa(x)$ at any value of x; set $\kappa_{i\pm 1/2} = \kappa(x_i \pm \Delta x/2)$

Variable diffusion coefficient

- Consider the heat equation with a variable diffusion coefficient

$$\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left[\kappa(x)\frac{\partial u}{\partial x}\right]$$

- Discretise the RHS at (x_i, t_j) in two steps, using central difference with spatial difference $\pm\Delta x/2$ for each derivative
- Outer space derivative

$$\frac{\partial}{\partial x}\left(\kappa(x)\frac{\partial u}{\partial x}\right)\bigg|_{x_i,t_j} = \frac{\left(\kappa(x)\frac{\partial u}{\partial x}\right)\Big|_{x_i+\frac{\Delta x}{2},t_j} - \left(\kappa(x)\frac{\partial u}{\partial x}\right)\Big|_{x_i-\frac{\Delta x}{2},t_j}}{\Delta x}$$

$$= \frac{\kappa(x_i + \frac{\Delta x}{2})\frac{\partial u}{\partial x}\Big|_{x_i+\frac{\Delta x}{2},t_j} - \kappa(x_i - \frac{\Delta x}{2})\frac{\partial u}{\partial x}\Big|_{x_i-\frac{\Delta x}{2},t_j}}{\Delta x}$$

- Can evaluate the diffusion coefficient $\kappa(x)$ at any value of x; set $\kappa_{i\pm 1/2} = \kappa(x_i \pm \Delta x/2)$

Variable diffusion coefficient

- Consider the heat equation with a variable diffusion coefficient

$$\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left[\kappa(x)\frac{\partial u}{\partial x}\right]$$

- Discretise the RHS at (x_i, t_j) in two steps, using central difference with spatial difference $\pm\Delta x/2$ for each derivative
- Outer space derivative

$$\frac{\partial}{\partial x}\left(\kappa(x)\frac{\partial u}{\partial x}\right)\Big|_{x_i,t_j} = \frac{\left(\kappa(x)\frac{\partial u}{\partial x}\right)\Big|_{x_i+\frac{\Delta x}{2},t_j} - \left(\kappa(x)\frac{\partial u}{\partial x}\right)\Big|_{x_i-\frac{\Delta x}{2},t_j}}{\Delta x}$$

$$= \frac{\kappa(x_i + \frac{\Delta x}{2})\frac{\partial u}{\partial x}\big|_{x_i+\frac{\Delta x}{2},t_j} - \kappa(x_i - \frac{\Delta x}{2})\frac{\partial u}{\partial x}\big|_{x_i-\frac{\Delta x}{2},t_j}}{\Delta x}$$

- Can evaluate the diffusion coefficient $\kappa(x)$ at any value of x; set $\kappa_{i\pm1/2} = \kappa(x_i \pm \Delta x/2)$

Variable diffusion coefficient

- Consider the heat equation with a variable diffusion coefficient

$$\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left[\kappa(x)\frac{\partial u}{\partial x}\right]$$

- Discretise the RHS at (x_i, t_j) in two steps, using central difference with spatial difference $\pm\Delta x/2$ for each derivative
- Outer space derivative

$$\frac{\partial}{\partial x}\left(\kappa(x)\frac{\partial u}{\partial x}\right)\bigg|_{x_i,t_j} = \frac{\left(\kappa(x)\frac{\partial u}{\partial x}\right)\Big|_{x_i+\frac{\Delta x}{2},t_j} - \left(\kappa(x)\frac{\partial u}{\partial x}\right)\Big|_{x_i-\frac{\Delta x}{2},t_j}}{\Delta x}$$

$$= \frac{\kappa(x_i+\frac{\Delta x}{2})\frac{\partial u}{\partial x}\Big|_{x_i+\frac{\Delta x}{2},t_j} - \kappa(x_i-\frac{\Delta x}{2})\frac{\partial u}{\partial x}\Big|_{x_i-\frac{\Delta x}{2},t_j}}{\Delta x}$$

- Can evaluate the diffusion coefficient $\kappa(x)$ at any value of x; set $\kappa_{i\pm1/2} = \kappa(x_i \pm \Delta x/2)$

Discretizing the spatial derivative term

- Same approach for the inner space derivatives

$$\left.\frac{\partial u}{\partial x}\right|_{x_i+\frac{\Delta x}{2},t_j} = \frac{u(x_i+\Delta x,t_j)-u(x_i,t_j)}{\Delta x} = \frac{u_{i+1,j}-u_{i,j}}{\Delta x}$$

$$\left.\frac{\partial u}{\partial x}\right|_{x_i-\frac{\Delta x}{2},t_j} = \frac{u(x_i,t_j)-u(x_i-\Delta x,t_j)}{\Delta x} = \frac{u_{i,j}-u_{i-1,j}}{\Delta x}$$

- Hence

$$\left.\frac{\partial}{\partial x}\left(\kappa(x)\frac{\partial u}{\partial x}\right)\right|_{x_i,t_j} = \frac{\kappa_{i+1/2}(u_{i+1,j}-u_{i,j})-\kappa_{i-1/2}(u_{i,j}-u_{i-1,j})}{\Delta x^2}$$

- Use forward difference as usual to discretize the time derivative

Variable diffusion coefficient — scheme

- Collecting all the terms

$$u_{i,j+1} = u_{i,j} + \frac{\Delta t}{\Delta x^2}\left\{\kappa_{i+1/2}u_{i+1,j} - (\kappa_{i+1/2} + \kappa_{i-1/2})u_{i,j} + \kappa_{i-1/2}u_{i-1,j}\right\}$$

  where $\kappa_{i\pm1/2} = \kappa(x_i \pm \Delta x/2)$

- Write as a matrix equation as usual

$$\boldsymbol{u}^{(j+1)} = \mathcal{A}_{\mathsf{FE}}\boldsymbol{u}^{(j)}$$

- $\mathcal{A}_{\mathsf{FE}}$ is tridiagonal, but the entries in each row vary: row i entries are

$$\frac{\Delta t}{\Delta x^2}\kappa_{i-1/2},\ 1 - \frac{\Delta t}{\Delta x^2}(\kappa_{i+1/2} + \kappa_{i-1/2}),\ \frac{\Delta t}{\Delta x^2}\kappa_{i+1/2}$$

- Scheme is stable if $\kappa(x) \geqslant \kappa^\star > 0\ \forall x$, and $\kappa^\star\Delta t/\Delta x^2 < 1/2$
- NB: if κ is constant, this is just regular forward Euler!

Nonlinear PDE problems

- More general source terms can pose significant problems; consider

$$\frac{\partial u}{\partial t} = \kappa \frac{\partial^2 u}{\partial x^2} + F(u, x, t)$$

- If F is linear in u we can discretize as before, with no issues
- If F is nonlinear in u, life can become quite difficult
  - solutions might develop sharp transitions that require lots of spatial resolution
  - we don't know the stability criterion for explicit methods (e.g., forward Euler)
  - implicit methods require us to solve a nonlinear equation at each time step
  - testing is difficult because there are few (if any) analytic solutions available

Forward Euler

- Consider the nonlinear diffusion PDE

$$\frac{\partial u}{\partial t} = \kappa \frac{\partial^2 u}{\partial x^2} + F(u)$$

- We could use forward Euler; discretizing at (x_i, t_j) gives an explicit rule:

$$\frac{u_{i,j+1} - u_{i,j}}{\Delta t} = \kappa \frac{u_{i+1,j} - 2u_{i,j} + u_{i+1,j}}{\Delta x^2} + F(u_{i,j})$$
$$u_{i,j+1} = u_{i,j} + \lambda \left(u_{i+1,j} - 2u_{i,j} + u_{i+1,j}\right) + \Delta t F(u_{i,j})$$

- Downsides:
    - only conditionally stable, must experiment with Δx and Δt (and hope!)
    - slow convergence (first order in time)

Crank-Nicholson

- Backward Euler and Crank-Nicholson lead to a set of coupled nonlinear equations at each timestep. E.g., for Crank-Nicholson,

$$\frac{u_{i,j+1} - u_{i,j}}{\Delta t} = \frac{\kappa}{2}\left[\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2} + \frac{u_{i+1,j+1} - 2u_{i,j+1} + u_{i-1,j+1}}{\Delta x^2}\right] + \frac{1}{2}\left[F(u_{i,j+1}) + F(u_{i,j})\right]$$

or, in vector form,

$$\boldsymbol{u}^{(j+1)} - \boldsymbol{u}^{(j)} = \frac{\lambda}{2}\left[D\boldsymbol{u}^{(j)} + D\boldsymbol{u}^{(j+1)}\right] + \frac{\Delta t}{2}\left[\boldsymbol{F}(\boldsymbol{u}^{(j)}) + \boldsymbol{F}(\boldsymbol{u}^{(j+1)})\right] \tag{3}$$

where $\lambda = \kappa\Delta t/\Delta x^2$, $D = \mathrm{tridiag}(1, -2, 1)$, $\boldsymbol{u}_i^{(j)} = u_{i,j}$, $\boldsymbol{F}_i(\boldsymbol{x}) = F(\boldsymbol{x}_i)$
- Must solve the nonlinear equation (3) for $\boldsymbol{u}^{(j+1)}$ at each timestep
- We have a reasonable initial guess: the solution at the previous timestep $\boldsymbol{u}^{(j)}$

Diffusion in more space dimensions

- Methods so far have been to solve parabolic PDEs in one space dimension and time
- They can be generalised (to some extent) to more space dimensions
- E.g., find solution $u = u(x, y, t)$ of

$$\frac{\partial u}{\partial t} = \kappa \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) \qquad \text{for } 0 < x < L_x,\ 0 < y < L_y,\ 0 < t \leqslant T$$

- Dirichlet boundary conditions for all $t > 0$

$$u(0, y, 0) = u(L_x, y, 0) = 0,$$
$$u(x, 0, 0) = u(x, L_y, 0) = 0$$

- Initial condition

$$u(x, y, 0) = u_I(x, y)$$

Discretization

- We can use the same finite difference approach as before
- Grid spacing $\Delta x, \Delta y$ in x, y directions, Δt in time
- Approximate solution at grid points by $u(x_i, y_j, t_k) = u_{i,j}^k$
- Approximate derivatives as before; e.g. for forward Euler, discretize PDE at (x_i, t_j) using

$$
\begin{aligned}
\frac{\partial u}{\partial t}(x_i, t_j) &= \frac{u_{i,j}^{k+1} - u_{i,j}^k}{\Delta t} \\
\frac{\partial^2 u}{\partial x^2}(x_i, t_j) &= \frac{u_{i-1,j}^k - 2u_{i,j}^k + u_{i+1,j}^k}{\Delta x^2} \\
\frac{\partial^2 u}{\partial y^2}(x_i, t_j) &= \frac{u_{i,j-1}^k - 2u_{i,j}^k + u_{i,j+1}^k}{\Delta y^2}
\end{aligned}
$$

2D Forward Euler scheme

- Turning the handle gives us the full scheme

$$
\begin{aligned}
u_{i,j}^{k+1} &= u_{i,j}^{k} + \lambda_x \left(u_{i+1,j}^{k} - 2u_{i,j}^{k} + u_{i-1,j}^{k} \right) + \lambda_y \left(u_{i,j+1}^{k} - 2u_{i,j}^{k} + u_{i,j-1}^{k} \right) \\
&= (1 - 2(\lambda_x + \lambda_y))u_{i,j}^{k} + \lambda_x \left(u_{i+1,j}^{k} + u_{i-1,j}^{k} \right) + \lambda_y \left(u_{i,j+1}^{k} + u_{i,j-1}^{k} \right)
\end{aligned}
$$

- Two mesh Fourier numbers, one in each direction

$$
\lambda_x = \kappa \frac{\Delta t}{\Delta x^2}, \qquad \lambda_y = \kappa \frac{\Delta t}{\Delta y^2}
$$

2D Forward Euler matrix

- Exactly as before, write as a matrix/vector equation

$$\boldsymbol{u}^{k+1} = A_{\mathsf{FE2}}\boldsymbol{u}^k$$

- Solution vector $\boldsymbol{u}^k$ made up of all $u_{i,j}^k$ in sequence (here, in column-major order)

$$\begin{aligned}\boldsymbol{u}^k = (u_{1,1}^k, u_{1,2}^k, \ldots u_{1,m_y-1}^k, \quad u_{2,1}^k, u_{2,2}^k, \ldots, u_{2,m_y-1}^k, \quad \ldots \\ \ldots, \quad u_{m_x-1,1}^k, u_{m_x-1,2}^k, \ldots, u_{m_x-1,m_y-1}^k)\end{aligned}$$

- A_{FE2} is a tridiagonal block matrix, with structure

$$A_{\mathsf{FE2}} = \begin{pmatrix} \mathcal{A} & \mathcal{B} & & & 0 \\ \mathcal{B} & \mathcal{A} & \mathcal{B} & & \\ & \ddots & \ddots & \ddots & \\ & & \mathcal{B} & \mathcal{A} & \mathcal{B} \\ 0 & & & \mathcal{B} & \mathcal{A} \end{pmatrix}$$

2D Forward Euler matrix (2)

- $\mathcal{A}$ and $\mathcal{B}$ are tridiagonal and diagonal $(m_y - 1) \times (m_y - 1)$ matrices

$$\mathcal{A} = \begin{pmatrix} 1-2(\lambda_x+\lambda_y) & \lambda_y & & 0 \\ \lambda_y & 1-2(\lambda_x+\lambda_y) & \lambda_y & \\ & \ddots & \ddots & \ddots \\ & \lambda_y & 1-2(\lambda_x+\lambda_y) & \lambda_y \\ 0 & & \lambda_y & 1-2(\lambda_x+\lambda_y) \end{pmatrix}$$

$$\mathcal{B} = \begin{pmatrix} \lambda_x & & & 0 \\ & \lambda_x & & \\ & & \ddots & \\ 0 & & & \lambda_x \end{pmatrix}$$

- They are repeated a total of $m_x - 1$ (for $\mathcal{A}$) and $m_x - 2$ (for $\mathcal{B}$) times down the leading diagonal, and the two off-diagonals, of A_{FE2}, respectively

2D Forward Euler properties

- The 2D forward Euler scheme is straightforward (if a bit cumbersome) to implement
- Sparse matrix techniques help a lot with storage requirements
- It is an *explicit* scheme
- It is only conditionally stable. We need, for stability

$$\kappa \frac{\Delta t}{\Delta x^2 + \Delta y^2} \leqslant \frac{1}{8}$$

- Truncation error is $\mathcal{O}(\Delta t) + \mathcal{O}(\Delta x^2) + \mathcal{O}(\Delta y^2)$
- Again, we'd like to use an implicit scheme for unconditional stability, and also to have quicker convergence

Better 2D schemes

- In principle, we could turn the handle as usual to obtain e.g. a 2D Crank-Nicholson scheme[3]
- Unfortunately, this leads to a big mess: an $(m_x - 1)(m_y - 1)$ dimensional linear system to solve at each time step, with a coefficient matrix that isn't tridiagonal
- This gets prohibitively expensive very quickly
- Same problem with backward Euler
- There are several possible ways out
  - sparse solvers might be clever enough on moderate size problems
  - solve the matrix equation via an iterative method: e.g., SOR (successive over-relaxation), Gauss-Seidel, Jacobi
  - break each timestep down into simpler bits: the ADI (alternating direction implicit) method

[3] Central differences to find u_t, u_{xx} and u_{yy} at $(x_i, y_j, t_{k+1/2})$, then average over t_k and t_{k+1} in the spatial terms

The ADI method

- The idea: take two backward-Euler-like half-steps per timestep
  1. time $k \to k+\frac{1}{2}$: evaluate u_{yy} at time t_k, update u_t, u_{xx} to $t_{k+\frac{1}{2}}$

$$\frac{u_{i,j}^{k+\frac{1}{2}} - u_{i,j}^{k}}{\Delta t/2} = \kappa\left(\frac{u_{i+1,j}^{k+\frac{1}{2}} - 2u_{i,j}^{k+\frac{1}{2}} + u_{i-1,j}^{k+\frac{1}{2}}}{\Delta x^2} + \frac{u_{i,j+1}^{k} - 2u_{i,j}^{k} + u_{i,j-1}^{k}}{\Delta y^2}\right)$$

  2. time $k+\frac{1}{2} \to k+1$: evaluate u_{xx} at time $t_{k+\frac{1}{2}}$, update u_t, u_{yy} to t_{k+1}

$$\frac{u_{i,j}^{k+1} - u_{i,j}^{k+\frac{1}{2}}}{\Delta t/2} = \kappa\left(\frac{u_{i+1,j}^{k+\frac{1}{2}} - 2u_{i,j}^{k+\frac{1}{2}} + u_{i-1,j}^{k+\frac{1}{2}}}{\Delta x^2} + \frac{u_{i,j+1}^{k+1} - 2u_{i,j}^{k+1} + u_{i,j-1}^{k+1}}{\Delta y^2}\right)$$

- Each half-step in time can be written in matrix/vector form $A_\star \boldsymbol{u}^{(\star+\frac{1}{2})} = B_\star \boldsymbol{u}^{(\star)}$ with a tridiagonal matrix $A_\star$: cheap to solve

The ADI method (2)

- The ADI method is unconditionally stable
- Second-order accurate in time and space; the truncation error is $\mathcal{O}(\Delta t^2) + \mathcal{O}(\Delta x^2) + \mathcal{O}(\Delta y^2)$
- It's no more computationally intensive than Crank-Nicholson for 1D in space systems
- Technical challenge: to make $A_\star$ tridiagonal in each half-step in time, we need to re-order the solution vector (rows/columns first) at each half-step
- Sadly, it doesn't generalize to 3D in space parabolic PDEs

Summary

- Finite difference methods can be used to solve parabolic PDE problems
  - fixed steps Δt in time, and Δx in space
  - expand the PDE: replace derivatives with finite difference approximations
  - solve for $u_{i,j} = u(x_0 + i\Delta x, t_0 + j\Delta t)$
- Forward Euler scheme
  - explicit
  - only conditionally stable: $\kappa \Delta t / \Delta x^2 < 1/2$ for 1D diffusion
  - converges slowly: first order in time, second order in space
- Crank-Nicholson scheme
  - implicit; must solve an algebraic system at each time step
  - unconditionally stable
  - converges fast: second order in both time and space
- Boundary conditions, source terms, etc., lead to modified matrices and/or additional right-hand-side vectors