

# Time-dependent variational methods

MATMEK-4270

Prof. Mikael Mortensen, University of Oslo

# Time dependent variational forms

In lectures 11-12 we have considered steady equations like

$$\mathcal{L}(u) = f$$

for a variety of operators  $\mathcal{L}$  and a numerical solution depending only on space  $u_N(\boldsymbol{x})$ .

In this lecture we will add consider two types of time-dependency

$$\begin{aligned}\frac{\partial u}{\partial t} + \mathcal{L}(u) &= f \\ \frac{\partial^2 u}{\partial t^2} + \mathcal{L}(u) &= f\end{aligned}$$

and approximations  $u_N(\boldsymbol{x}, t) \in V_N = \text{span}\{\psi_j\}_{j=0}^N$ :

$$u_N(\boldsymbol{x}, t) = \sum_{j=0}^N \hat{u}_j(t) \psi_j(\boldsymbol{x})$$

where the expansion coefficients are time dependent.

# Finite difference in time, Galerkin in space

Consider the equation

$$\frac{\partial u}{\partial t} + \mathcal{L}(u) = f$$

A finite difference approximation in time, with forward Euler:

$$\frac{u^{n+1} - u^n}{\Delta t} + \mathcal{L}(u^n) = f^n$$

where

$$u^n(\mathbf{x}) = u(\mathbf{x}, t_n), f^n(\mathbf{x}) = f(\mathbf{x}, t_n)$$

and time is discretized as always for time domain  $[0, T]$

$$t_n = n\Delta t, \quad n = 0, 1, \dots, N_t, \quad N_t = T/\Delta t$$



Note

We normally drop the spatial dependence and write only  $u^n$  or  $f^n$

# The residual is now a function of time

$$\mathcal{R}(u^{n+1}; u^n) = \frac{u^{n+1} - u^n}{\Delta t} + \mathcal{L}(u^n) - f^n$$



Note

The notation  $\mathcal{R}(u^{n+1}; u^n)$  is used to indicate that  $u^{n+1}$  is the unknown and  $u^n$  (right of the semicolon) is known.

Introduce the approximation to  $u^n(\mathbf{x}) = u(\mathbf{x}, t_n)$

$$u(\mathbf{x}, t_n) \approx u_N^n(\mathbf{x}) = \sum_{j=0}^N \hat{u}_j^n \psi_j(\mathbf{x})$$

where  $\hat{u}_j^n = \hat{u}_j(t_n)$  are the time dependent expansion coefficients (the unknowns at time  $t_n$ ). Insert into the residual to get the **numerical residual**

$$\mathcal{R}_N = \mathcal{R}(u_N^{n+1}; u_N^n) = \frac{u_N^{n+1} - u_N^n}{\Delta t} + \mathcal{L}(u_N^n) - f^n$$

# The method of weighted residuals

The method of weighted residuals is to find  $u_N^{n+1} \in V_N$  such that

$$(\mathcal{R}_N, v) = 0, \quad \forall v \in W$$

and for the Galerkin method  $W = V_N$ .

In order to solve the time-dependent problem we need to solve this Galerkin problem many times. The procedure is

1. Initialize  $u_N^0(\mathbf{x})$  by projecting the given initial condition  $u(\mathbf{x}, 0)$  to  $V_N$  (function approximation, see lectures 8, 9). That is, find  $u_N^0 \in V_N$  such that

$$(u_N^0 - u(\mathbf{x}, 0), v) = 0, \quad \forall v \in V_N$$

2. For  $n = 0, 1, \dots, N_t - 1$  find  $u_N^{n+1} \in V_N$  such that

$$(\mathcal{R}_N, v) = 0, \quad \forall v \in V_N$$

# Stability of time- dependent variational methods

# Recap

Remember the early lectures 2, 3 and 5

Exponential decay  $\frac{\partial u(t)}{\partial t} = -au(t)$

Vibration equation  $\frac{\partial^2 u(t)}{\partial t^2} + \omega^2 u(t) = 0$

Wave equation  $\frac{\partial^2 u(x, t)}{\partial t^2} = c^2 \frac{\partial^2 u(x, t)}{\partial x^2}$

In order to study stability we introduced the ansatz

$$u^{n+1} = gu^n \rightarrow u^n = g^n u^0$$

into the discretized equations. For stability we required  $|g| \leq 1$ .



We actually called the amplification factor  $A$ , but since then we have started using matrices that we rather have as  $A$ . Hence the amplification factor is now called  $g$ .

# Recap vibration equation

$$\frac{\partial^2 u(t)}{\partial t^2} + \omega^2 u(t) = 0$$

Discretize with central finite difference

$$\frac{u^{n+1} - 2u^n + u^{n-1}}{\Delta t^2} + \omega^2 u^n = 0$$

Insert for ansatz  $u^n = g^n u^0$

$$g^n (g - 2 + g^{-1}) u^0 + (\omega \Delta t)^2 g^n u^0 = 0$$

Divide by  $g^n u^0$

$$g + g^{-1} = 2 - (\omega \Delta t)^2$$

and solve quadratic equation ( $g$  may be complex) to find stability  $|g| = 1$  for

$$\omega \Delta t \leq 2$$



# Recap - stability of the diffusion equation

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

Use forward Euler in time and central difference in space

$$u_j^{n+1} - u_j^n = \frac{\Delta t}{\Delta x^2} (u_{j+1}^n - 2u_j^n + u_{j-1}^n)$$

where  $u_j^n = u(j\Delta x, n\Delta t)$  and  $x_j = j\Delta x$  and  $t_n = n\Delta t$ . Make the ansatz as always

$$u_j^{n+1} = g u_j^n \rightarrow u_j^n = g^n u_j^0$$

and let the initial condition be a periodic Fourier wave with wavenumber  $k$

$$u_j^0 = e^{\hat{i} k x_j}$$

where  $\hat{i} = \sqrt{-1}$ .

# Write the equation in matrix form

$$u_j^{n+1} - u_j^n = \frac{\Delta t}{\Delta x^2} (u_{j+1}^n - 2u_j^n + u_{j-1}^n)$$

$$\mathbf{u}^n = (u_j^n)_{j=0}^N$$

$$\mathbf{u}^{n+1} - \mathbf{u}^n = \frac{\Delta t}{\Delta x^2} \tilde{D}^{(2)} \mathbf{u}^n$$

$$D^{(2)} = \frac{1}{\Delta x^2} \tilde{D}^{(2)} = \frac{1}{\Delta x^2} \underbrace{\begin{bmatrix} -2 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 & 0 & 0 & 0 \\ \vdots & & & \ddots & & & & \dots \\ 0 & 0 & 0 & 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & -2 \end{bmatrix}}_{\tilde{D}^{(2)}}$$

# Find the amplification factor

$$\mathbf{u}^{n+1} - \mathbf{u}^n = \frac{\Delta t}{\Delta x^2} \tilde{D}^{(2)} \mathbf{u}^n$$

Insert for ansatz  $\mathbf{u}^n = g^n \mathbf{u}^0$

$$(g - 1) \mathbf{u}^0 = \frac{\Delta t}{\Delta x^2} \tilde{D}^{(2)} \mathbf{u}^0$$

Inserting for  $u_j^0 = e^{ikj\Delta x}$  we get

$$(g - 1) e^{ikj\Delta x} = \frac{\Delta t}{\Delta x^2} (e^{ik(j+1)\Delta x} - 2e^{ikj\Delta x} + e^{ik(j-1)\Delta x})$$

Divide by  $e^{ikj\Delta x}$  to obtain

$$g - 1 = \frac{\Delta t}{\Delta x^2} (e^{ik\Delta x} - 2 + e^{-ik\Delta x})$$

**Simplify**  $g - 1 = \frac{\Delta t}{\Delta x^2} (e^{ik\Delta x} - 2 + e^{-ik\Delta x})$

Use  $e^{ix} + e^{-ix} = 2 \cos x$ ,  $\cos(2x) = \cos^2 x - \sin^2 x$  and  $1 = \cos^2 x + \sin^2 x$

$$g = 1 - \frac{4\Delta t}{\Delta x^2} \sin^2 \left( \frac{k\Delta x}{2} \right)$$

We want  $|g| \leq 1$ , so we need

$$\left| 1 - \frac{4\Delta t}{\Delta x^2} \sin^2 \left( \frac{k\Delta x}{2} \right) \right| \leq 1$$

Since  $\sin^2(k\Delta x/2) \leq 1$  and positive, we get that  $1 - \frac{4\Delta t}{\Delta x^2}$  is always less than 1.

However,  $1 - \frac{4\Delta t}{\Delta x^2}$  can be smaller than  $-1$ , and that gives us the stability limit

$$1 - \frac{4\Delta t}{\Delta x^2} \geq -1 \Rightarrow \boxed{\Delta t \leq \frac{\Delta x^2}{2}}$$

# A lot of hard work to find

$$\left| 1 - \frac{4\Delta t}{\Delta x^2} \right| \leq 1 \Rightarrow \Delta t \leq \frac{\Delta x^2}{2}$$

Can we find an easier approach?

We have the simple looking formula

$$(g - 1)\mathbf{u}^0 = \frac{\Delta t}{\Delta x^2} \tilde{D}^{(2)} \mathbf{u}^0$$

But we cannot eliminate the two  $\mathbf{u}^0$  terms on each side because of the matrix  $\tilde{D}^{(2)}$ !

Eigenvalues! The eigenvalues  $\lambda$  of a matrix  $A$  with associated eigenvectors  $\mathbf{x}$  are

$$A\mathbf{x} = \lambda\mathbf{x}$$

Hence the simplification

$$\tilde{D}^{(2)} \mathbf{u}^0 = \lambda \mathbf{u}^0 \Rightarrow (g - 1)\mathbf{u}^0 = \frac{\Delta t}{\Delta x^2} \lambda \mathbf{u}^0 \Rightarrow g = 1 + \frac{\lambda \Delta t}{\Delta x^2}$$

# Stability limit through eigenvalues

Assume the ansatz  $\mathbf{u}^n = g^n \mathbf{u}^0$ :

$$(g - 1)\mathbf{u}^0 = \frac{\Delta t}{\Delta x^2} \tilde{D}^{(2)} \mathbf{u}^0$$

Insert for  $\tilde{D}^{(2)} \mathbf{u}^0 = \lambda \mathbf{u}^0$  to get

$$g = 1 + \frac{\lambda \Delta t}{\Delta x^2}$$

To satisfy  $|g| \leq 1$

$$|g| = \left| 1 + \frac{\lambda \Delta t}{\Delta x^2} \right| \leq 1$$

But we need to compute the eigenvalues..

# Compute eigenvalues

```
1 import numpy as np
2 from scipy import sparse
3 N = 11
4 D2 = sparse.diags((1, 1, -2, 1, 1), (-N, -1, 0, 1, N), shape=(N+1, N+1))
5 Lambda = np.linalg.eig(D2.toarray())[0]
6 Lambda
```

```
array([-4.00000000e+00, -3.73205081e+00, -3.00000000e+00, -2.00000000e+00,
       -1.00000000e+00, -4.53075522e-16, -2.67949192e-01, -2.67949192e-01,
       -1.00000000e+00, -3.73205081e+00, -2.00000000e+00, -3.00000000e+00])
```

The minimum eigenvalue is -4 (worst case). Hence we obtain again exactly the same limit

$$\left|1 + \frac{\lambda \Delta t}{\Delta x^2}\right| \leq 1 \Rightarrow \left|1 - \frac{4\Delta t}{\Delta x^2}\right| \leq 1 \Rightarrow \Delta t \leq \frac{\Delta x^2}{2}$$

But with a lot less work!



## Note

The complex exponentials  $e^{\hat{i}kx}$  are eigenfunctions of the Laplace operator in 1D, so we actually used eigenvalue theory also for the hard approach, we just did not mention it!

# Stability of the forward Euler method for variational forms

For the forward Euler method we now consider the discretized Galerkin equation

$$(u_N^{n+1}, v) = (u_N^n, v) + \Delta t (\mathcal{L}(u_N^n), v)$$

and make the same ansatz as before

$$u_N^{n+1} = g u_N^n \quad \text{and thus} \quad u_N^n = g^n u_N^0$$

Insert into the variational form

$$(g^{n+1} u_N^0, v) = (g^n u_N^0, v) + \Delta t (\mathcal{L}(g^n u_N^0), v)$$

Divide by  $g^n$  ( $g$  is not dependent on  $x$ ) to obtain

$$g (u_N^0, v) = (u_N^0, v) + \Delta t (\mathcal{L}(u_N^0), v)$$



# Derive the linear algebra form

Insert for  $u_N^0 = \sum_{j=0}^N \hat{u}_j^0 \psi_j$  and  $v = \psi_i$  to obtain

$$g \sum_{j=0}^N (\psi_j, \psi_i) \hat{u}_j^0 = \sum_{j=0}^N (\psi_j, \psi_i) \hat{u}_j^0 + \Delta t \sum_{j=0}^N (\mathcal{L}(\psi_j), \psi_i) \hat{u}_j^0$$

In matrix form this is

$$gA\hat{\mathbf{u}}^0 = A\hat{\mathbf{u}}^0 + \Delta t M\hat{\mathbf{u}}^0$$

where  $a_{ij} = (\psi_j, \psi_i)$  and  $m_{ij} = (\mathcal{L}(\psi_j), \psi_i)$ . Multiply from the left by  $A^{-1}$  to obtain

$$g\hat{\mathbf{u}}^0 = \hat{\mathbf{u}}^0 + \Delta t A^{-1} M \hat{\mathbf{u}}^0$$

Compute the eigenvalues  $\lambda$  of the matrix  $H = A^{-1}M$  such that  $H\hat{\mathbf{u}}^0 = \lambda\hat{\mathbf{u}}^0$  and

$$g\hat{\mathbf{u}}^0 = \hat{\mathbf{u}}^0 + \lambda\Delta t\hat{\mathbf{u}}^0 \Rightarrow \boxed{g = 1 + \lambda\Delta t}$$

$$\text{Stability limit: } |g| = |1 + \lambda\Delta t| \leq 1$$

# Complete worked example, the diffusion equation

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}, \quad x, t \in (0, L) \times (0, T]$$

$$u(x, 0) = p(x) = \sin(\pi x/L) + \sin(10\pi x/L)$$

$$u(0, t) = 0, u(L, t) = 0$$

Forward Euler in time, Galerkin in space, and Dirichlet Legendre polynomials:

$$\text{Basis:} \quad \psi_j(x) = P_j(X) - P_{j+2}(X) \quad V_N = \text{span}\{\psi_j\}_{j=0}^N$$

$$\text{Mapping:} \quad x(X) = \frac{L}{2}(1 + X) \quad X(x) = \frac{2x}{L} - 1$$

We get the variational form used on the previous slide with diffusion for  $\mathcal{L}(\psi_j)$  and integration by parts (boundary terms canceled since  $\psi_j(0) = \psi_j(L) = 0$ ):

$$\sum_{j=0}^N (\psi_j, \psi_i) \hat{u}_j^{n+1} = \sum_{j=0}^N (\psi_j, \psi_i) \hat{u}_j^n - \Delta t \left( \sum_{j=0}^N (\psi'_j, \psi'_i) \hat{u}_j^n - \cancel{[u_N^n \psi_i]_{x=0}^{x=L}} \right)$$

# The stiffness matrix $(\psi'_j, \psi'_i)$

$$\begin{aligned}\int_0^L \psi'_j(x) \psi'_i(x) dx &= \int_{-1}^1 (P'_j(X) - P'_{j+2}(X)) \frac{dX}{dx} (P'_i(X) - P'_{i+2}(X)) \frac{dX}{dx} \frac{dx}{dX} dX \\ &= \frac{2}{L} \int_{-1}^1 (P'_j(X) - P'_{j+2}(X)) (P'_i(X) - P'_{i+2}(X)) dX \\ &= \frac{2}{L} (P'_j - P'_{j+2}, P'_i - P'_{i+2})_{L^2(-1,1)}\end{aligned}$$

Using equality:  $(2j+3)P_{j+1} = P'_{j+2} - P'_j$  and  $(P_{i+1}, P_{j+1})_{L^2(-1,1)} = \|P_{i+1}\|^2 \delta_{ij}$   
and  $\|P_i\|^2 = \frac{2}{2j+1}$

$$\begin{aligned}\int_0^L \psi'_j(x) \psi'_i(x) dx &= \frac{2}{L} (-(2j+3)P_{j+1}, -(2j+3)P_{i+1}) \\ &= \frac{2}{L} (2j+3)^2 \|P_{i+1}\|^2 \delta_{ij} \\ &= \frac{8j+12}{L} \delta_{ij}\end{aligned}$$

# The mass matrix $(\psi_j, \psi_i)$

$$\begin{aligned}
 \int_0^L \psi_j(x) \psi_i(x) dx &= \int_{-1}^1 (P_j(X) - P_{j+2}(X))(P_i(X) - P_{i+2}(X)) \frac{dx}{dX} dX \\
 &= \frac{L}{2} (P_j - P_{j+2}, P_i - P_{i+2})_{L^2(-1,1)} \\
 &= \frac{L}{2} ((P_j, P_i) - (P_{j+2}, P_i) - (P_j, P_{i+2}) + (P_{j+2}, P_{i+2}))
 \end{aligned}$$

Using now  $(P_j, P_i) = \|P_i\|^2 \delta_{ij} = \frac{2}{2i+1} \delta_{ij}$ :

$$\begin{aligned}
 j = i \quad (\psi_i, \psi_i) &= \frac{L}{2} ((P_i, P_i) + (P_{i+2}, P_{i+2})) = \frac{L}{2} (\|P_i\|^2 + \|P_{i+2}\|^2) \\
 j = i + 2 \quad (\psi_{i+2}, \psi_i) &= -(P_{i+2}, P_{i+2}) = -\frac{L}{2} \|P_{i+2}\|^2
 \end{aligned}$$

Since the mass matrix  $a_{ij} = a_{ji} = (\psi_j, \psi_i)$  is symmetric, we get the tri-diagonal

$$a_{ij} = a_{ji} = \frac{L}{2} \begin{cases} \|P_i\|^2 + \|P_{i+2}\|^2, & i = j \\ -\|P_{i+2}\|^2, & j = i + 2 \\ 0, & \text{otherwise} \end{cases}$$

# Implementation

It is rather messy to work with composite basis functions. A very smart trick is thus to use a **stencil matrix**  $S$ , such that (summation on repeated  $j$  index)

$$\psi_i = P_i - P_{i+2} = s_{ij}P_j$$

This defines  $S \in \mathbb{R}^{(N+1) \times (N+3)}$  for the Dirichlet problem as

$$s_{ij} = \begin{cases} 1 & i = j \\ -1 & j = i + 2 \end{cases} \quad S = \begin{bmatrix} 1 & 0 & -1 & 0 & 0 & \dots \\ 0 & 1 & 0 & -1 & 0 & \dots \\ 0 & 0 & 1 & 0 & -1 & \dots \\ \vdots & \vdots & \vdots & \ddots & \ddots & \ddots \\ 0 & 0 & 0 & \dots & 1 & 0 & -1 \end{bmatrix}$$

With  $\boldsymbol{\psi} = \{\psi_i\}_{i=0}^N$  and  $\boldsymbol{P} = \{P_i\}_{i=0}^N$  we get all basis functions in matrix form:

$$\boldsymbol{\psi} = S\boldsymbol{P}$$

# Why is the stencil matrix smart?

Consider the mass matrix

$$\begin{aligned}(\psi_j, \psi_i) &= (P_j - P_{j+2}, P_i - P_{i+2}) \\ &= (P_j, P_i) - (P_j, P_{i+2}) - (P_{j+2}, P_i) + (P_{j+2}, P_{i+2})\end{aligned}$$

You need to sort out the four (diagonal) matrices on the right.

Now use stencil matrix instead (summation on  $k$  and  $l$ ):

$$\begin{aligned}(\psi_j, \psi_i) &= (s_{jk}P_k, s_{il}P_l) \\ &= s_{il}(P_l, P_k)s_{jk}\end{aligned}$$

You need only the diagonal matrix  $P = ((P_l, P_k))_{k,l=0}^{N,N}$ , where  $(P_l, P_k) = \frac{2}{2l+1}\delta_{kl}$ .

Matrix form

$$A = ((\psi_j, \psi_i))_{i,j=0}^{N,N} = SPS^T$$

# Stiffness matrix through stencil matrix

$$\begin{aligned}(\psi_j'', \psi_i) &= (s_{jk}P_k'', s_{il}P_l) \\ &= s_{il}(P_l'', P_k)s_{jk}\end{aligned}$$

Again you need only the single matrix using orthogonal polynomials

$$q_{kl} = (P_l'', P_k) = -(P_l', P_k')$$

$$M = ((\psi_j'', \psi_i))_{i,j=0}^{N,N} = SQS^T$$

# Implementation using shenfun

```
1 import numpy as np
2 import sympy as sp
3 from scipy import sparse
4 from shenfun import TrialFunction, TestFunction, FunctionSpace, \
5     project, inner, Dx, la, Function
6
7 x, t, c, L = sp.symbols('x,t,c,L')
8
9 class FE_Diffusion:
10     def __init__(self, N, L0=1, u0=sp.sin(sp.pi*x/L)+sp.sin(10*sp.pi*x/L), family='L'):
11         self.N = N
12         self.L = L0
13         self.u0 = u0.subs({L: L0})
14         self.V = self.get_function_space(family=family)
15         self.uh_np1 = Function(self.V)
16         self.uh_n = Function(self.V)
17         self.A, self.S = self.get_mats()
18         self.sol = la.Solver(self.A)
19         self.A = self.A.diags()
20         self.S = self.S.diags()
21
22     def get_function_space(self, family='L'):
23         return FunctionSpace(self.N+1, family, bc=(0, 0), domain=(0, self.L))
24
25     def get_mats(self):
26         N = self.N
27         u = TrialFunction(self.V)
```

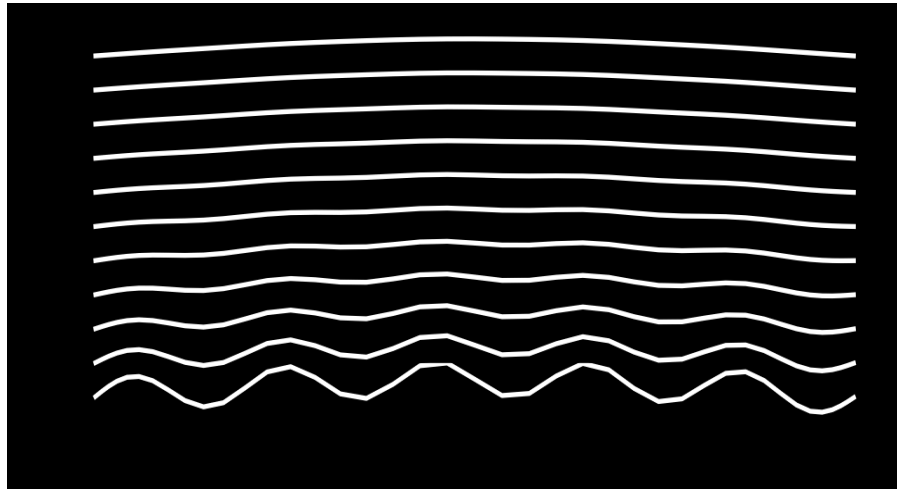


# Run solver

Use longest stable time step

$$dt = 2 / \max(\lambda)$$

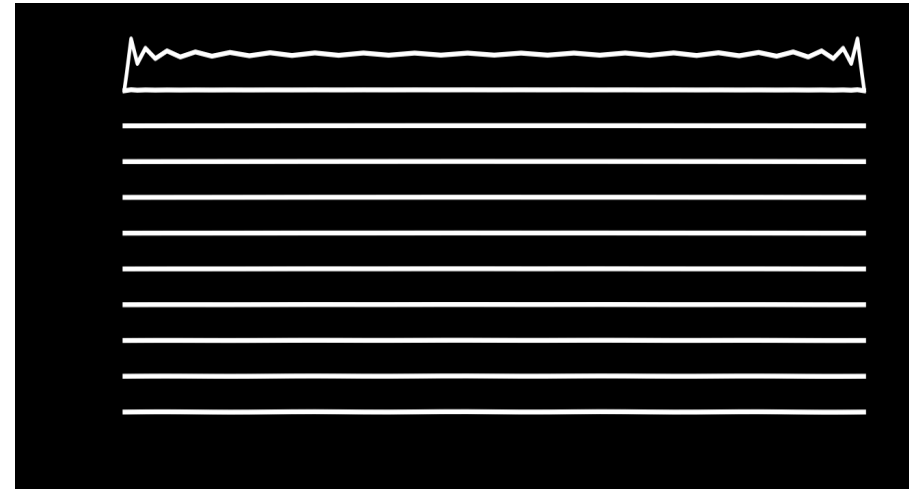
```
1 from utilities import plot_with_offset
2 sol = FE_Diffusion(42, L0=2)
3 dt = 2/max(sol.get_eigenvalues())
4 data = sol(1000, dt=dt, save_step=100)
5 plot_with_offset(data, sol.V.mesh(), figsize=(8, 3))
```



Use slightly too large time step

$$dt = 2.04 / \max(\lambda)$$

```
1 #
2 #
3 #
4 data = sol(1000, dt=dt*1.02, save_step=100)
5 plot_with_offset(data, sol.V.mesh(), figsize=(8, 3))
```



# Backward Euler for diffusion equation

The forward Euler method requires a very short time step for the diffusion equation. How about the backward Euler method

$$(u_N^{n+1}, v) = (u_N^n, v) + \Delta t \left( \frac{\partial^2 u_N^{n+1}}{\partial x^2}, v \right)$$

The linear algebra problem is now

$$\sum_{j=0}^N ((\psi_j, \psi_i) + \Delta t(\psi'_j, \psi'_i)) \hat{u}_j^{n+1} = \sum_{j=0}^N (\psi_j, \psi_i) \hat{u}_j^n$$

Insert the ansatz  $u_N^{n+1} = g u_N^n$

$$\sum_{j=0}^N g ((\psi_j, \psi_i) + \Delta t(\psi'_j, \psi'_i)) \hat{u}_j^0 = \sum_{j=0}^N (\psi_j, \psi_i) \hat{u}_j^0$$

# Backward Euler on matrix form

$$\sum_{j=0}^N g ((\psi_j, \psi_i) + \Delta t(\psi'_j, \psi'_i)) \hat{u}_j^0 = \sum_{j=0}^N (\psi_j, \psi_i) \hat{u}_j^0$$

becomes

$$g(A + \Delta t S) \hat{\mathbf{u}}^0 = A \hat{\mathbf{u}}^0$$

If we multiply from left to right by  $A^{-1}$  we get

$$g(I + \Delta t A^{-1} S) \hat{\mathbf{u}}^0 = \hat{\mathbf{u}}^0$$

Here we can once again use  $H = A^{-1} S$  and  $H \hat{\mathbf{u}}^0 = \lambda \hat{\mathbf{u}}^0$  and thus we find

$$g = \frac{1}{1 + \Delta t \lambda}$$

Since all time steps and eigenvalues are real numbers larger than 0,  $g$  is always less than 1 and positive and the backward Euler method is as such *unconditionally* stable.

# The wave equation as a variational problem

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2}, \quad (x, t) \in (0, L) \times (0, T]$$

$$u(x, 0) = f(x) \quad \frac{\partial u}{\partial t}(x, 0) = 0$$

We will consider both Dirichlet and Neumann boundary conditions in space and use finite differences in time such that

$$\frac{u^{n+1} - 2u^n + u^{n-1}}{\Delta t^2} = c^2 \frac{\partial^2 u^n}{\partial x^2}$$

A numerical residual  $\mathcal{R}_N = \mathcal{R}(u_N^{n+1}; u_N^n, u_N^{n-1})$  can now be defined as

$$\mathcal{R}_N = u_N^{n+1} - 2u_N^n + u_N^{n-1} - \alpha^2 (u_N^n)''$$

with  $\alpha = c\Delta t$  and  $u' = \frac{\partial u}{\partial x}$ .

# The Galerkin methods for the wave equation

Initialize  $u_N^0$  and  $u_N^1$ . The Galerkin method is to find  $u_N^{n+1} \in V_N$  such that

$$(u_N^{n+1} - 2u_N^n + u_N^{n-1} - \alpha^2(u_N^n)'', v) = 0, \quad \forall v \in V_N$$

for  $n = 1, \dots, N_t - 1$ . We can also integrate the last term by parts

$$(u_N^{n+1} - 2u_N^n + u_N^{n-1}, v) + \alpha^2 \left( ((u_N^n)', v') - [(u_N^n)'v]_{x=0}^{x=L} \right) = 0, \quad \forall v \in V_N$$

Insert for  $u_N^{n-1}, u_N^n, u_N^{n+1} \in V_N$  and use  $v = \psi_i$  to get the linear algebra problem

$$\sum_{j=0}^N (\psi_j, \psi_i) \left( \hat{u}_j^{n+1} - 2\hat{u}_j^n + \hat{u}_j^{n-1} \right) + \alpha^2 \left( \sum_{j=0}^N (\psi_j', \psi_i') \hat{u}_j^n - [(u_N^n)' \psi_i]_{x=0}^{x=L} \right) = 0$$



Note

All time dependency is in the expansion coefficients  $\hat{u}_j^n, \hat{u}_j^n, \hat{u}_j^{n-1}$ .

# Matrix form

Rearrange such that the unknown  $\hat{u}_j^{n+1}$  is on the left hand side and the rest of the known terms are on the right hand side

$$\sum_{j=0}^N (\psi_j, \psi_i) \hat{u}_j^{n+1} = \sum_{j=0}^N (\psi_j, \psi_i) (2\hat{u}_j^n - \hat{u}_j^{n-1}) - \alpha^2 \left( \sum_{j=0}^N (\psi'_j, \psi'_i) \hat{u}_j^n - [(u_N^n)' \psi_i]_{x=0}^{x=L} \right)$$

With matrix notation,  $a_{ij} = (\psi_j, \psi_i)$  and  $s_{ij} = (\psi'_j, \psi'_i)$ , we get

$$A\hat{\mathbf{u}}^{n+1} = A(2\hat{\mathbf{u}}^n - \hat{\mathbf{u}}^{n-1}) - \alpha^2 (S\hat{\mathbf{u}}^n - \mathbf{b}^n)$$

where  $\mathbf{b}^n = ((u_N^n)'(L)\psi_i(L) - (u_N^n)'(0)\psi_i(0))_{i=0}^N$ .

We can write this as

$$A\hat{\mathbf{u}}^{n+1} = \mathbf{f}^n$$

where the vector  $\mathbf{f}^n = A(2\hat{\mathbf{u}}^n - \hat{\mathbf{u}}^{n-1}) - \alpha^2 (S\hat{\mathbf{u}}^n - \mathbf{b}^n)$ .

**Dirichlet boundary conditions**  $u(0, t) = a$  and  $u(L, t) = b$

$$b_i^n = (u_N^n)'(L)\psi_i(L) - (u_N^n)'(0)\psi_i(0), \quad i = 0, 1, \dots, N$$

**Global variational methods**

$$\psi_i(0) = \psi_i(L) = 0, \quad i = 0, 1, \dots, N$$

$$\Rightarrow \mathbf{b}^n = 0, \quad \forall n = 1, 2, \dots, N_t - 1$$

Use a boundary function (with reference coordinate  $X \in [-1, 1]$ )

$$B(x) = \tilde{B}(X) = \frac{b}{2}(1 + X) + \frac{a}{2}(1 - X)$$

and solve for  $\tilde{u}_N^n \in V_N = \text{span}\{\psi_j\}_{j=0}^N$ . Final solution is then

$$u_N^n = \tilde{u}_N^n + B(x)$$

# Dirichlet with the finite element method

Makes use of Lagrange polynomials and  $\psi_i(0) = 1$  for  $i = 0$  and zero otherwise. Similarly,  $\psi_i(L) = 1$  for  $i = N$  and zero otherwise. Hence it is only  $b_0^n$  and  $b_N^n$  that potentially are different from zero. However,

$$u(0, t) = a = \hat{u}_0^{n+1} \quad \text{and} \quad u(L, t) = b = \hat{u}_N^{n+1}$$



## Note

Since  $\hat{u}_0$  and  $\hat{u}_N$  are known we do not solve the linear PDE for  $i = 0$  or  $i = N$ . Simply ident the coefficient matrix  $A$  and set  $f_0^n = a$  and  $f_N^n = b$ .

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ a_{10} & a_{11} & a_{12} & \cdots & a_{1N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{N-1,0} & a_{N-1,1} & a_{N-1,2} & \cdots & a_{N-1,N} \\ 0 & \cdots & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \hat{u}_0^{n+1} \\ \hat{u}_1^{n+1} \\ \vdots \\ \hat{u}_{N-1}^{n+1} \\ \hat{u}_N^{n+1} \end{bmatrix} = \begin{bmatrix} a \\ f_1^n \\ \vdots \\ f_{N-1}^n \\ b \end{bmatrix}$$



## Neumann boundary conditions

$$u'(0, t) = g_0 \quad \text{and} \quad u'(L, t) = g_L$$

$$b_i^n = (u_N^n)'(L)\psi_i(L) - (u_N^n)'(0)\psi_i(0), \quad i = 0, 1, \dots, N$$

## Apply Neumann strongly through basis

Use basis functions satisfying

$$\psi_j'(0) = \psi_j'(L) = 0, \quad j = 0, 1, \dots, N$$

Add a boundary function  $B(x)$  that satisfied  $B'(0) = g_0$  and  $B'(L) = g_L$

$$B(x) = \tilde{B}(X) = \frac{L}{8} (g_L(1 + X)^2 - g_0(1 - X)^2)$$

The  $L$  is there since  $X = -1 + \frac{2x}{L}$  and  $\frac{\partial X}{\partial x} = \frac{2}{L}$  and thus

$$\frac{\partial B}{\partial x} = \frac{\partial \tilde{B}}{\partial X} \frac{\partial X}{\partial x} = \frac{2}{L} \frac{\partial \tilde{B}}{\partial X}$$

# Neumann boundary conditions weakly

## Apply boundary condition weakly through variational form

$$b_i^n = (u_N^n)'(L)\psi_i(L) - (u_N^n)'(0)\psi_i(0), \quad i = 0, 1, \dots, N$$

Insert for  $(u_N^n)'(0) = g_0$  and  $(u_N^n)'(L) = g_L$

$$b_i^n = g_L\psi_i(L) - g_0\psi_i(0)$$

Use any basis function, except those where all  $\psi_i(0) = 0$  and  $\psi_i(L) = 0$ , for all  $i = 0, 1, \dots, N$ .

- This works for the finite element method, where  $\psi_0(0) = 1$  and  $\psi_N(L) = 1$ . Nothing else is required
- Also works for the global Galerkin method, e.g., Legendre polynomials  $\psi_i(x) = P_i(X)$ .
- Does not work for Chebyshev because the method relies on integration by parts, which does not work with the Chebyshev weights  $\omega(x) = 1/\sqrt{1 - X^2}$ .

# Stability of the wave equation in variational form

We have derived the linear algebra problem

$$A(\hat{\mathbf{u}}^{n+1} - 2\hat{\mathbf{u}}^n + \hat{\mathbf{u}}^{n-1}) = -\alpha^2 (S\hat{\mathbf{u}}^n - \mathbf{b}^n)$$

The ansatz  $u_N^{n+1} = gu_N^n = g^{n+1}u_N^0$  also implies that

$$\hat{\mathbf{u}}^{n+1} = g\hat{\mathbf{u}}^n = g^{n+1}\hat{\mathbf{u}}^0 \quad (1)$$

since

$$u_N^{n+1} = gu_N^n \Rightarrow \sum_{j=0}^N \hat{u}_j^{n+1} \psi_j = g \sum_{j=0}^N \hat{u}_j^n \psi_j = \sum_{j=0}^N (g\hat{u}_j^n) \psi_j$$

Hence  $\hat{u}_j^{n+1} = g\hat{u}_j^n$ . Recursively, we get (1).

# Find amplification factor

$$A(\hat{\mathbf{u}}^{n+1} - 2\hat{\mathbf{u}}^n + \hat{\mathbf{u}}^{n-1}) = -\alpha^2 (S\hat{\mathbf{u}}^n - \mathbf{b}^n)$$

Insert for  $\hat{\mathbf{u}}^n = g^n \hat{\mathbf{u}}^0$  and assume that  $\mathbf{b}^n = \mathbf{0}$

$$(g^{n+1} - 2g^n + g^{n-1})A\hat{\mathbf{u}}^0 = -\alpha^2 g^n S\hat{\mathbf{u}}^0$$

Divide by  $g^n$  and multiply by  $A^{-1}$  from the left

$$(g - 2 + g^{-1})I\hat{\mathbf{u}}^0 = -\alpha^2 A^{-1}S\hat{\mathbf{u}}^0$$



Note

The identity matrix  $I$  on the left is optional and can be removed.

# Find amplification factor 2

$$(g - 2 + g^{-1})I\hat{\mathbf{u}}^0 = -\alpha^2 A^{-1}S\hat{\mathbf{u}}^0$$

Use  $H = A^{-1}S$  and the eigenvalues  $H\hat{\mathbf{u}}^0 = \lambda I\hat{\mathbf{u}}^0$  to get

$$(g - 2 + g^{-1})I\hat{\mathbf{u}}^0 = -\alpha^2 \lambda I\hat{\mathbf{u}}^0$$

and after eliminating the equal terms on both sides

$$g + g^{-1} = \beta \tag{1}$$

where  $\beta = -\alpha^2 \lambda + 2$ .

**For stability we need  $|g| \leq 1$ .**



Note

If  $g < 1$  is a root of (1), then  $g^{-1} > 1$  is also a root, which is unstable. All roots must be  $\leq 1$ . Hence  $|g| = 1$  is the only possible solution.

# Find the longest stable time step

$$g + g^{-1} = \beta$$

Solve quadratic equation for  $g$

$$g = \frac{\beta \pm \sqrt{\beta^2 - 4}}{2}$$

Assume  $-2 \leq \beta \leq 2$  such that  $\beta^2 - 4 \leq 0$  and  $\sqrt{\beta^2 - 4} = a\hat{i}$ , with  $a = \sqrt{4 - \beta^2}$  real

$$2|g| = \left| \beta \pm \sqrt{\beta^2 - 4} \right| = \left| \beta \pm a\hat{i} \right| = \sqrt{\beta^2 + a^2} = \sqrt{\beta^2 + 4 - \beta^2} = 2$$

For  $-2 \leq \beta \leq 2$  we get that  $|g| = 1$  and the numerical solution is stable!

Using  $\beta = -\alpha^2\lambda + 2 = -(c\Delta t)^2\lambda + 2 = -2$  the longest possible time step is

$$\Delta t \leq \frac{1}{c} \sqrt{\frac{4}{\max(\lambda)}}$$

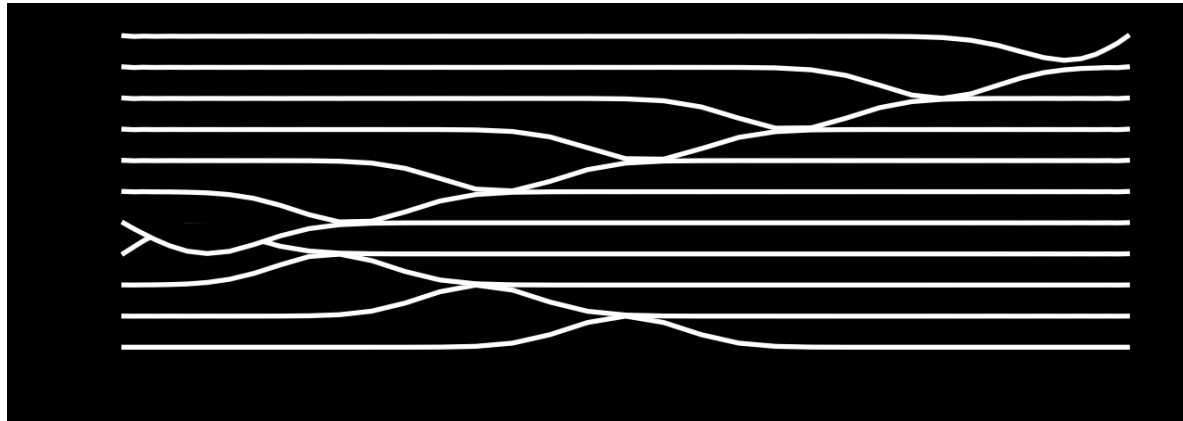
# Implementation - Overload diffusion solver

We are using the same function space and thus the same matrices  $A$  and  $S$  as the diffusion equation. We only need to reimplement the time stepper in `__call__`.

```
1 class Wave(FE_Diffusion):
2     def __init__(self, N, L0=1, c0=1, u0=sp.sin(sp.pi*x/L)):
3         u0 = u0.subs({c: c0})
4         FE_Diffusion.__init__(self, N, L0=L0, u0=u0)
5         self.c = c0
6         self.uh_nm1 = np.zeros_like(self.uh_n)
7
8     def __call__(self, Nt, dt=0.1, save_step=100):
9         self.uh_nm1[:] = project(self.u0.subs(t, 0), self.V)
10        xj = self.V.mesh()
11        plotdata = {0: self.uh_nm1(xj)}
12        self.uh_n[:] = project(self.u0.subs(t, dt), self.V)
13        if save_step == 1: plotdata[1] = self.uh_n(xj)
14
15        f = np.zeros(self.N+1)
16        alfa = self.c*dt
17        for n in range(2, Nt+1):
18            f[:-2] = self.A @ (2*self.uh_n[:-2]-self.uh_nm1[:-2]) \
19                - alfa**2*(self.S @ self.uh_n[:-2])
20            self.uh_np1[:] = self.sol(f)
21            self.uh_nm1[:] = self.uh_n
22            self.uh_n[:] = self.uh_np1
23            if n % save_step == 0: # save every save_step timestep
24                plotdata[n] = self.uh_np1(xj)
25        return plotdata
```

# Check stability

```
1 sol = Wave(40, L0=2, c0=1, u0=sp.exp(-40*(x-L/2+c*t)**2))
2 dt = 1/sol.c*np.sqrt(4/max(sol.get_eigenvalues()))
3 data = sol(400, dt=dt, save_step=40)
4 plot_with_offset(data, sol.V.mesh(), figsize=(8, 2.5))
```



```
1 data = sol(400, dt=dt*1.01, save_step=40)
2 plot_with_offset(data, sol.V.mesh(), figsize=(8, 2.5))
```

