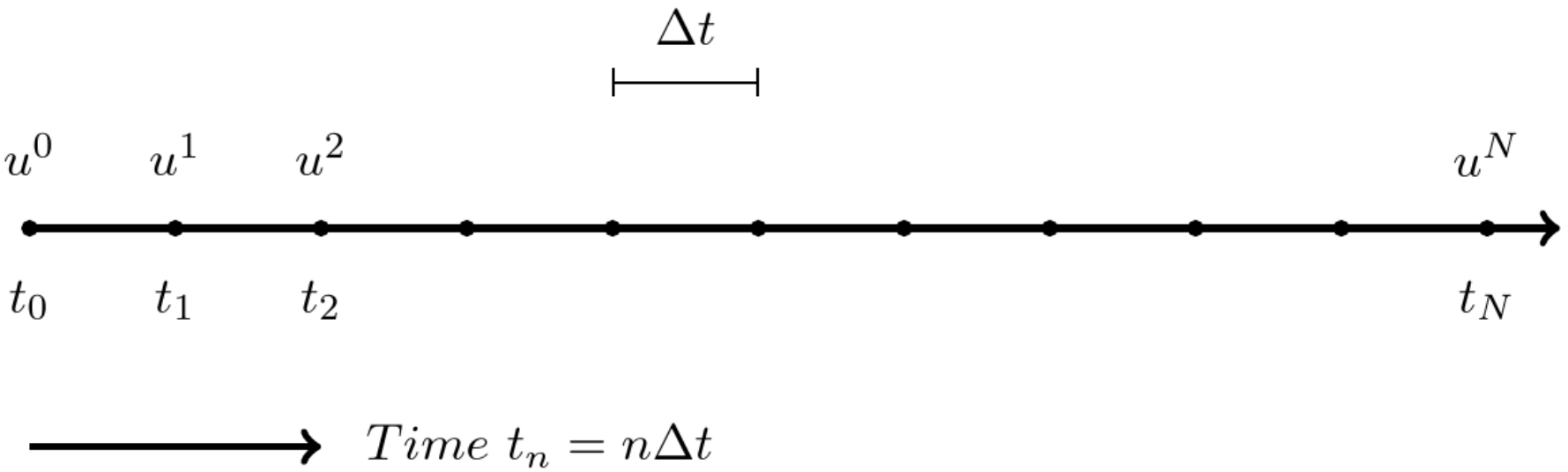
The finite difference method

MATMEK-4270

Prof. Mikael Mortensen, University of Oslo

The finite difference method

The finite difference method divides (in 1D) the line into a mesh and solves equations only for specific locations (nodes) in the mesh. A mesh is created with $t = 0, \Delta t, 2\Delta t, \ldots, N\Delta t$, where $t_n = n\Delta t$ and $T = t_N = N\Delta t$.



Up until now we have solved equations by using a recurrence approach. This is very easy to implement intuitively using for-loops. However, the most common and general use of finite difference methods is through **explicit** assembling of matrices.

The recurrence approach for exponential decay

The exponential decay problem is

$$u' + au = 0, t \in (0, T], \, u(0) = I.$$

with difference equation

$$u^{n+1} = \frac{1 - (1 - \theta)a\Delta t}{1 + \theta a \Delta t} u^n = g u^n.$$

A solution vector $\boldsymbol{u} = (u^0, u^1, \ldots, u^{N_t})$. The recurrence algorithm is:

- $u^0 = I$
- for n = 0, 1, … , N-1
  - Compute $u^{n+1} = g u^n$

Easy to understand and easy to solve. Equations are never **assembled** into matrix form

The matrix approach

The matrix approach solves the difference equations by assembling matrices and vectors. Each **row** in the matrix-problem represents one difference equation.

$$A\boldsymbol{u} = \boldsymbol{b}$$

For the exponential decay problem the matrix problem looks like

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -g & 1 & 0 & 0 & 0 \\ 0 & -g & 1 & 0 & 0 \\ 0 & 0 & -g & 1 & 0 \\ 0 & 0 & 0 & -g & 1 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} u^0 \\ u^1 \\ u^2 \\ u^3 \\ u^4 \\ u^5 \end{bmatrix}}_{\boldsymbol{u}} = \underbrace{\begin{bmatrix} I \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}}_{\boldsymbol{b}}$$

Understand that the matrix problem is the same as the recurrence

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -g & 1 & 0 & 0 & 0 \\ 0 & -g & 1 & 0 & 0 \\ 0 & 0 & -g & 1 & 0 \\ 0 & 0 & 0 & -g & 1 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} u^0 \\ u^1 \\ u^2 \\ u^3 \\ u^4 \\ u^5 \end{bmatrix}}_{\boldsymbol{u}} = \underbrace{\begin{bmatrix} I \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}}_{\boldsymbol{b}}$$

simply means

$$\begin{aligned} u^0 &= I \\ -gu^0 + u^1 &= 0 \\ -gu^1 + u^2 &= 0 \\ &\vdots \end{aligned}$$

Solve matrix problem

The matrix problem

$$A\boldsymbol{u} = \boldsymbol{b}$$

is solved as

$$\boldsymbol{u} = A^{-1}\boldsymbol{b}$$

Assemble matrix problem

```
N = 8
a = 2
I = 1
theta = 0.5
dt = 0.5
T = N*dt
t = np.linspace(0, N*dt, N+1)
u = np.zeros(N+1)
g = (1 - (1-theta) * a * dt)/(1 + theta * a * dt)
```

Assemble

```
from scipy import sparse
A = sparse.diags([-g, 1], np.array([-1, 0]), (N+1, N+1), 'csr')
b = np.zeros(N+1); b[0] = I
```

Solve and compare with recurrence

Recurrence:

```
u[0] = I
for n in range(N):
  u[n+1] = g * u[n]
```

Matrix

```
um = sparse.linalg.spsolve(A, b)
print(u-um)
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

Ok, no difference. Which is faster?

```
def reccsolve(u, N, I):
  u[0] = I
  for n in range(N):
    u[n+1] = g * u[n]
N = 1000
u = np.zeros(N+1)
%timeit -n100 reccsolve(u, N, I)
```

```
105 µs ± 3.92 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

```
%timeit -n100 um = sparse.linalg.spsolve(A, b)
```

```
19.2 µs ± 1.77 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

Compute the inverse

A unit lower triangular matrix has a unit lower triangular inverse

```
np.set_printoptions(precision=3, suppress=True)
Ai = np.linalg.inv(A.toarray())
print(Ai)
```

```
[[1.    0.    0.    0.    0.    0.    0.    0.    0.   ]
 [0.333 1.    0.    0.    0.    0.    0.    0.    0.   ]
 [0.111 0.333 1.    0.    0.    0.    0.    0.    0.   ]
 [0.037 0.111 0.333 1.    0.    0.    0.    0.    0.   ]
 [0.012 0.037 0.111 0.333 1.    0.    0.    0.    0.   ]
 [0.004 0.012 0.037 0.111 0.333 1.    0.    0.    0.   ]
 [0.001 0.004 0.012 0.037 0.111 0.333 1.    0.    0.   ]
 [0.    0.001 0.004 0.012 0.037 0.111 0.333 1.    0.   ]
 [0.    0.    0.001 0.004 0.012 0.037 0.111 0.333 1.   ]]
```

Now solve directly using A^{-1}

```
ui = Ai @ b
```

Compare with previous

```
um = sparse.linalg.spsolve(A, b)
print(um-ui)
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

```
%timeit -n 100 ui = Ai @ b
```

```
548 ns ± 36.3 ns per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

The vibration problem

$$u'' + \omega^2 u = 0,\ t \in (0, T] \quad u(0) = I, u'(0) = 0,$$

is solved using a central difference for $n = 1, 2, \ldots, N-1$

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

The recurrence algorithm is

- $u^0 = I$
- $u^1 = u^0(1 - 0.5\omega^2 \Delta t^2)$
- for n = 1, 2, … , N-1
  - $u^{n+1} = (2 - \omega^2 \Delta t^2)u^n - u^{n-1}$

Vibration on matrix form

The matrix problem is now, using $g = 2 - \omega^2 \Delta t^2$,

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -g/2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -g & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -g & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & -g & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & -g & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & -g & 1 \end{bmatrix} \begin{bmatrix} u^0 \\ u^1 \\ u^2 \\ u^3 \\ u^4 \\ u^5 \\ u^6 \\ u^7 \end{bmatrix} = \begin{bmatrix} I \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Note

Notice that the matrix is still lower triangular. Matrices that are lower or upper triangular are especially quick to solve for using forward or backward substitution. This is due to the explicit recurrence relation. Just optimize the for-loop using Numba/Cython.

Stencils from Taylor expansions

Create one backward and three forward Taylor expansions starting from $u(t_n)$

$$
\begin{aligned}
(-1) \quad & u^{n-1} = u^n - hu' + \frac{h^2}{2}u'' - \frac{h^3}{6}u''' + \frac{h^4}{24}u'''' - \cdots \\
(1) \quad & u^{n+1} = u^n + hu' + \frac{h^2}{2}u'' + \frac{h^3}{6}u''' + \frac{h^4}{24}u'''' + \cdots \\
(2) \quad & u^{n+2} = u^n + 2hu' + \frac{2h^2}{1}u'' + \frac{4h^3}{3}u''' + \frac{2h^4}{3}u'''' + \cdots \\
(3) \quad & u^{n+3} = u^n + 3hu' + \frac{9h^2}{2}u'' + \frac{9h^3}{2}u''' + \frac{27h^4}{8}u'''' + \cdots
\end{aligned}
$$

$u^{n+a} = u(t_{n+a})$ and $t_{n+a} = (n+a)h$ and we use $h = \Delta t$ for simplicity.

Now add equations (-1) and (1) and isolate $u''(t_n)$

$$
u''(t_n) = \frac{u^{n+1} - 2u^n + u^{n-1}}{h^2} + \frac{h^2}{12}u'''' +
$$

Matrix form

Let the mesh function mesh function $\boldsymbol{u}^{(2)} \in \mathbb{R}^N$ be defined as

$$\boldsymbol{u}^{(2)} = (u''(t_n))_{n=0}^{N}$$

We can compute this using a finite difference matrix $D^{(2)} \in \mathbb{R}^{N \times N}$ and $\boldsymbol{u} \in \mathbb{R}^N$

$$\boldsymbol{u}^{(2)} = D^{(2)}\boldsymbol{u},$$

$$\underbrace{\begin{bmatrix} u_0^{(2)} \\ u_1^{(2)} \\ u_2^{(2)} \\ \vdots \\ u_{N-2}^{(2)} \\ u_{N-1}^{(2)} \\ u_N^{(2)} \end{bmatrix}}_{\boldsymbol{u}^{(2)}} = \underbrace{\begin{bmatrix} ? & ? & ? & ? & ? & ? & ? & ? \\ 1 & -2 & 1 & 0 & 0 & 0 & 0 & \cdots \\ 0 & 1 & -2 & 1 & 0 & 0 & 0 & \cdots \\ \vdots & & & \ddots & & & & \vdots \\ \vdots & 0 & 0 & 0 & 1 & -2 & 1 & 0 \\ \vdots & 0 & 0 & 0 & 0 & 1 & -2 & 1 \\ ? & ? & ? & ? & ? & ? & ? & ? \end{bmatrix}}_{D^{(2)}} \underbrace{\begin{bmatrix} u^0 \\ u^1 \\ u^2 \\ \vdots \\ u^{N-1} \\ u^{N-1} \\ u^N \end{bmatrix}}_{\boldsymbol{u}}$$

Boundaries

$$\underbrace{\begin{bmatrix} u_0^{(2)} \\ u_1^{(2)} \\ u_2^{(2)} \\ \vdots \\ u_{N-2}^{(2)} \\ u_{N-1}^{(2)} \\ u_N^{(2)} \end{bmatrix}}_{\boldsymbol{u}^{(2)}} = \underbrace{\frac{1}{h^2} \begin{bmatrix} ? & ? & ? & ? & ? & ? & ? & ? \\ 1 & -2 & 1 & 0 & 0 & 0 & 0 & \cdots \\ 0 & 1 & -2 & 1 & 0 & 0 & 0 & \cdots \\ \vdots & & & \ddots & & & & \vdots \\ \vdots & 0 & 0 & 0 & 1 & -2 & 1 & 0 \\ \vdots & 0 & 0 & 0 & 0 & 1 & -2 & 1 \\ ? & ? & ? & ? & ? & ? & ? & ? \end{bmatrix}}_{D^{(2)}} \underbrace{\begin{bmatrix} u^0 \\ u^1 \\ u^2 \\ \vdots \\ u^{N-1} \\ u^{N-1} \\ u^N \end{bmatrix}}_{\boldsymbol{u}}$$

What to do with the first and last rows? The central stencils do not work here.

Forward and backward stencils will do

Forward difference at $n = 0$

Remember:

$$(1) \quad u^{n+1} = u^n + hu' + \frac{h^2}{2}u'' + \frac{h^3}{6}u''' + \frac{h^4}{24}u'''' + \cdots$$

$$(2) \quad u^{n+2} = u^n + 2hu' + \frac{2h^2}{1}u'' + \frac{4h^3}{3}u''' + \frac{2h^4}{3}u'''' + \cdots$$

Subtract 2 times Eq. (1) from Eq. (2) and rearrange

$$(2) - 2(1): \; u^{n+2} - 2u^{n+1} = -u^n + \frac{h^2}{1}u'' + h^3u''' + \frac{7h^4}{12}u''''+$$

Rearrange to isolate a first order accurate stencil for $u''(0)$

$$u''(0) = \frac{u^2 - 2u^1 + u^0}{h^2} - hu'''(0) - \frac{7h^2}{12}u''''(0)+$$

Backward difference at $n = N$

Use two backward Taylor expansions:

$$(-2) \quad u^{n-2} = u^n - 2hu' + \frac{2h^2}{1}u'' - \frac{4h^3}{3}u''' + \frac{2h^4}{3}u'''' - \cdots$$

$$(-1) \quad u^{n-1} = u^n - hu' + \frac{h^2}{2}u'' - \frac{h^3}{6}u''' + \frac{h^4}{24}u'''' - \cdots$$

Subtract 2 times Eq. (-1) from Eq. (-2) and rearrange

$$(-2) - 2(-1) : \; u^{n-2} - 2u^{n-1} = -u^n + \frac{h^2}{1}u'' - h^3u''' + \frac{7h^4}{12}u''''+$$

Rearrange to isolate a first order accurate backward stencil for $u''(T)$

$$u''(T) = \frac{u^{N-2} - 2u^{N-1} + u^N}{h^2} - hu'''(T) - \frac{7h^2}{12}u''''(T)+$$

Second derivative matrix

$$D^{(2)} = \frac{1}{h^2}\begin{bmatrix} 1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 & 0 & 0 & \cdots \\ 0 & 1 & -2 & 1 & 0 & 0 & 0 & \cdots \\ \vdots & & & \ddots & & & & \cdots \\ \vdots & 0 & 0 & 0 & 1 & -2 & 1 & 0 \\ \vdots & 0 & 0 & 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 \end{bmatrix}$$

But with first order only at the boundaries. Can we do better?

Second order forward stencil

Use three forward points $u^{n+1}, u^{n+2}, u^{n+3}$

$$(1) \quad u^{n+1} = u^n + hu' + \frac{h^2}{2}u'' + \frac{h^3}{6}u''' + \frac{h^4}{24}u'''' + \cdots$$

$$(2) \quad u^{n+2} = u^n + 2hu' + \frac{2h^2}{1}u'' + \frac{4h^3}{3}u''' + \frac{2h^4}{3}u'''' + \cdots$$

$$(3) \quad u^{n+3} = u^n + 3hu' + \frac{9h^2}{2}u'' + \frac{9h^3}{2}u''' + \frac{27h^4}{8}u'''' + \cdots$$

Now to eliminate both u' and u''' terms add the three equations as $-(3) + 4 \cdot (2) - 5 \cdot (1)$ (don't worry about how I know this yet)

$$-(3) + 4 \cdot (2) - 5 \cdot (1): \; -u^{n+3} + 4u^{n+2} - 5u^{n+1} = -2u^n + h^2u'' - \frac{11h^4}{12}u'''' +$$

Isolate $u''(0)$

$$u''(0) = \frac{-u^3 + 4u^2 - 5u^1 + 2u^n}{h^2} + \frac{11h^2}{12}u'''' +$$

Fully second order $D^{(2)}$

$$D^{(2)} = \frac{1}{h^2}\begin{bmatrix} 2 & -5 & 4 & -1 & 0 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 & 0 & 0 & \cdots \\ 0 & 1 & -2 & 1 & 0 & 0 & 0 & \cdots \\ \vdots & & & \ddots & & & & \cdots \\ \vdots & 0 & 0 & 0 & 1 & -2 & 1 & 0 \\ \vdots & 0 & 0 & 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 0 & -1 & 4 & -5 & 2 \end{bmatrix}$$

Assemble in Python

```
N = 8
D2 = sparse.diags([1, -2, 1], np.array([-1, 0, 1]), (N+1, N+1), 'lil')
D2[0, :4] = 2, -5, 4, -1
D2[-1, -4:] = -1, 4, -5, 2
D2 *= (1/dt**2) # don't forget h
D2.toarray()*dt**2
```

```
array([[ 2., -5.,  4., -1.,  0.,  0.,  0.,  0.,  0.],
       [ 1., -2.,  1.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  1., -2.,  1.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  1., -2.,  1.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  1., -2.,  1.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  1., -2.,  1.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  1., -2.,  1.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  1., -2.,  1.],
       [ 0.,  0.,  0.,  0.,  0., -1.,  4., -5.,  2.]])
```

Apply matrix to a mesh function $f(t_n) = t_n^2$

```
array([2., 2., 2., 2., 2., 2., 2., 2., 2.])
```

Exact for all n!

First order boundary

```
D21 = sparse.diags([1, -2, 1], np.array([-1, 0, 1]), (N+1, N+1), 'lil')
D21[0, :4] = 1, -2, 1, 0
D21[-1, -4:] = 0, 1, -2, 1
D21 *= (1/dt**2)
D21 @ f
```

```
array([2., 2., 2., 2., 2., 2., 2., 2., 2.])
```

Still exact! Why?

Consider the forward first order stencil

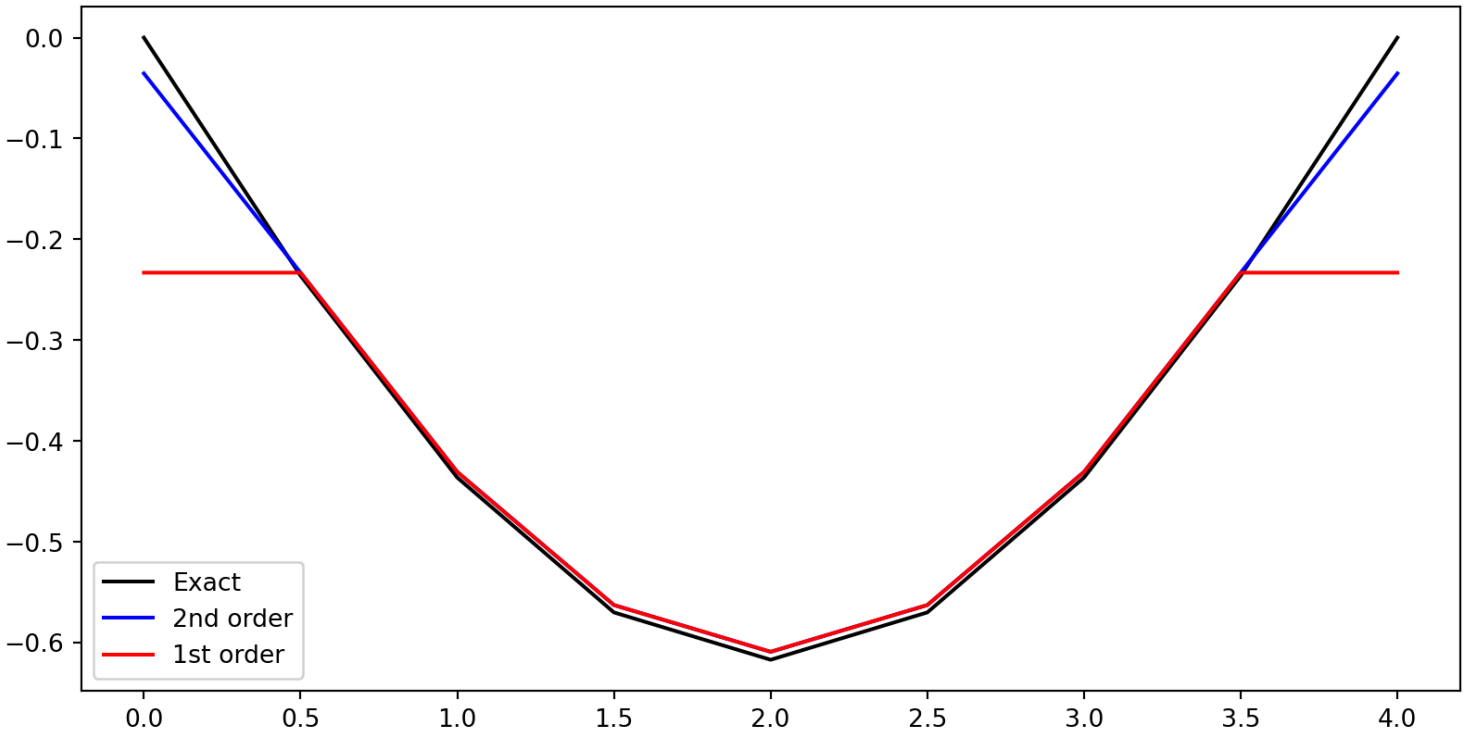
$$u''(0) = \frac{u^{n+2} - 2u^{n+1} + u^n}{h^2} - hu'''(0) - \frac{7h^2}{12}u''''(0) +$$

The leading error is $hu'''(0)$. What is $u'''(0)$ if $u(t) = t^2$? The second order stencil captures the second order polynomial t^2 exactly!

More challenging example

Let $f(t) = \sin(\pi t/T)$ such that $f''(t) = -(\pi/T)^2 f(t)$. Here none of the error terms will disappear and we see the effect of the poor first order boundary.

```
f = np.sin(np.pi*t / T)
d2fe = -(np.pi/T)**2*f
d2f = D2 @ f
d2f1 = D21 @ f
plt.plot(t, d2fe, 'k', t, d2f, 'b', t, d2f1, 'r')
plt.legend(['Exact', '2nd order', '1st order']);
```



First derivative matrix

Lets create a similar matrix for a second order accurate single derivative.

$$(-1) \quad u^{n-1} = u^n - hu' + \frac{h^2}{2}u'' - \frac{h^3}{6}u''' + \frac{h^4}{24}u'''' + \cdots$$

$$(1) \quad u^{n+1} = u^n + hu' + \frac{h^2}{2}u'' + \frac{h^3}{6}u''' + \frac{h^4}{24}u'''' + \cdots$$

Here Eq. (1) minus Eq. (-1) leads to

$$u'(t_n) = \frac{u^{n+1} - u^{n-1}}{2h} + \frac{h^2}{6}u''' +$$

which is second order accurate.

The central scheme cannot be used for $n = 0$ or $n = N$.

Boundaries single derivative

Forward for $n = 0$

$$(1) \quad u^{n+1} = u^n + hu' + \frac{h^2}{2}u'' + \frac{h^3}{6}u''' + \frac{h^4}{24}u'''' + \cdots$$

$$(2) \quad u^{n+2} = u^n + 2hu' + \frac{2h^2}{1}u'' + \frac{4h^3}{3}u''' + \frac{2h^4}{3}u'''' + \cdots$$

We get a first order approximation for u' using merely Eq. (1):

$$u'(t_n) = \frac{u^{n+1} - u^n}{h} - \frac{h}{2}u'' -$$

Adding one more equation (2) we get second order: $(2) - 4 \cdot (1)$ (Note that the terms with u'' then cancel)

$$u'(t_n) = \frac{-u^{n+2} + 4u^{n+1} - 3u^n}{2h} + \frac{h^2}{3}u''' +$$

First derivative matrix

The backward for $n = N$ is the same, only with different sign from $n = 0$. The derivative matrix is

$$D^{(1)} = \frac{1}{2h}\begin{bmatrix} -3 & 4 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 & 0 & \cdots \\ 0 & -1 & 0 & 1 & 0 & 0 & 0 & \cdots \\ \vdots & & & \ddots & & & & \cdots \\ \vdots & 0 & 0 & 0 & -1 & 0 & 1 & 0 \\ \vdots & 0 & 0 & 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & -4 & 3 \end{bmatrix}$$

```
D1 = sparse.diags([-1, 1], np.array([-1, 1]), (N+1, N+1), 'lil')
D1[0, :3] = -3, 4, -1
D1[-1, -3:] = 1, -4, 3
D1 *= (1/(2*dt))
f = t
D1 @ f
```

```
array([1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

Is $D^{(2)}$ the same as $D^{(1)}D^{(1)}$?

```
D22 = D1 @ D1
D22.toarray()*4*dt**2
```

```
array([[  5., -11.,   7.,  -1.,   0.,   0.,   0.,   0.,   0.],
       [  3.,  -5.,   1.,   1.,   0.,   0.,   0.,   0.,   0.],
       [  1.,   0.,  -2.,   0.,   1.,   0.,   0.,   0.,   0.],
       [  0.,   1.,   0.,  -2.,   0.,   1.,   0.,   0.,   0.],
       [  0.,   0.,   1.,   0.,  -2.,   0.,   1.,   0.,   0.],
       [  0.,   0.,   0.,   1.,   0.,  -2.,   0.,   1.,   0.],
       [  0.,   0.,   0.,   0.,   1.,   0.,  -2.,   0.,   1.],
       [  0.,   0.,   0.,   0.,   0.,   1.,   1.,  -5.,   3.],
       [  0.,   0.,   0.,   0.,   0.,  -1.,   7., -11.,   5.]])
```

No! This stencil is wider, using neighboring points farther away. The central stencil is

$$u''(t_n) = \frac{u^{n+2} - 2u^n + u^{n-2}}{4h^2}$$

How accurate is $D^{(1)}D^{(1)}$?

Internal points

$$(-2) \quad u^{n-2} = u^n - 2hu' + \frac{2h^2}{1}u'' - \frac{4h^3}{3}u''' + \frac{2h^4}{3}u'''' - \cdots$$

$$(2) \quad u^{n+2} = u^n + 2hu' + \frac{2h^2}{1}u'' + \frac{4h^3}{3}u''' + \frac{2h^4}{3}u'''' + \cdots$$

Take Eq. (2) plus Eq. (-2)

$$u^{n+2} + u^{n-2} = 2u^n + 4h^2u'' + \frac{4h^4}{3}u'''' +$$

and

$$u''(t_n) = \frac{u^{n+2} - 2u^n + u^{n-2}}{4h^2} - \frac{h^2}{3}u'''' + \cdots$$

Second order!

How about boundaries?

$$u''(t_1) = \frac{u^{n+2} + u^{n+1} - 5u^n + 3u^{n-1}}{4h^2}$$

$$(-1) \quad u^{n-1} = u^n - hu' + \frac{h^2}{2}u'' - \frac{h^3}{6}u''' + \frac{h^4}{24}u'''' - \cdots$$

$$(1) \quad u^{n+1} = u^n + hu' + \frac{h^2}{2}u'' + \frac{h^3}{6}u''' + \frac{h^4}{24}u'''' + \cdots$$

$$(2) \quad u^{n+2} = u^n + 2hu' + \frac{2h^2}{1}u'' + \frac{4h^3}{3}u''' + \frac{2h^4}{3}u'''' + \cdots$$

Take Eq. $(2) + (1) + 3 \cdot (-1)$

$$u''(t_1) = \frac{u^{n+2} + u^{n+1} - 5u^n + 3u^{n-1}}{4h^2} + \frac{h}{4}u''' + \cdots$$

Only first order.

First point

$$u''(t_0) = \frac{-u^{n+3} + 7u^{n+2} - 11u^{n+1} + 5u^n}{4h^2}$$

$$(1) \quad u^{n+1} = u^n + hu' + \frac{h^2}{2}u'' + \frac{h^3}{6}u''' + \frac{h^4}{24}u'''' + \cdots$$

$$(2) \quad u^{n+2} = u^n + 2hu' + \frac{2h^2}{1}u'' + \frac{4h^3}{3}u''' + \frac{2h^4}{3}u'''' + \cdots$$

$$(3) \quad u^{n+3} = u^n + 3hu' + \frac{9h^2}{2}u'' + \frac{9h^3}{2}u''' + \frac{27h^4}{8}u'''' + \cdots$$

Take Eq. $-(3) + 7 \cdot (2) - 11 \cdot (1)$ to obtain

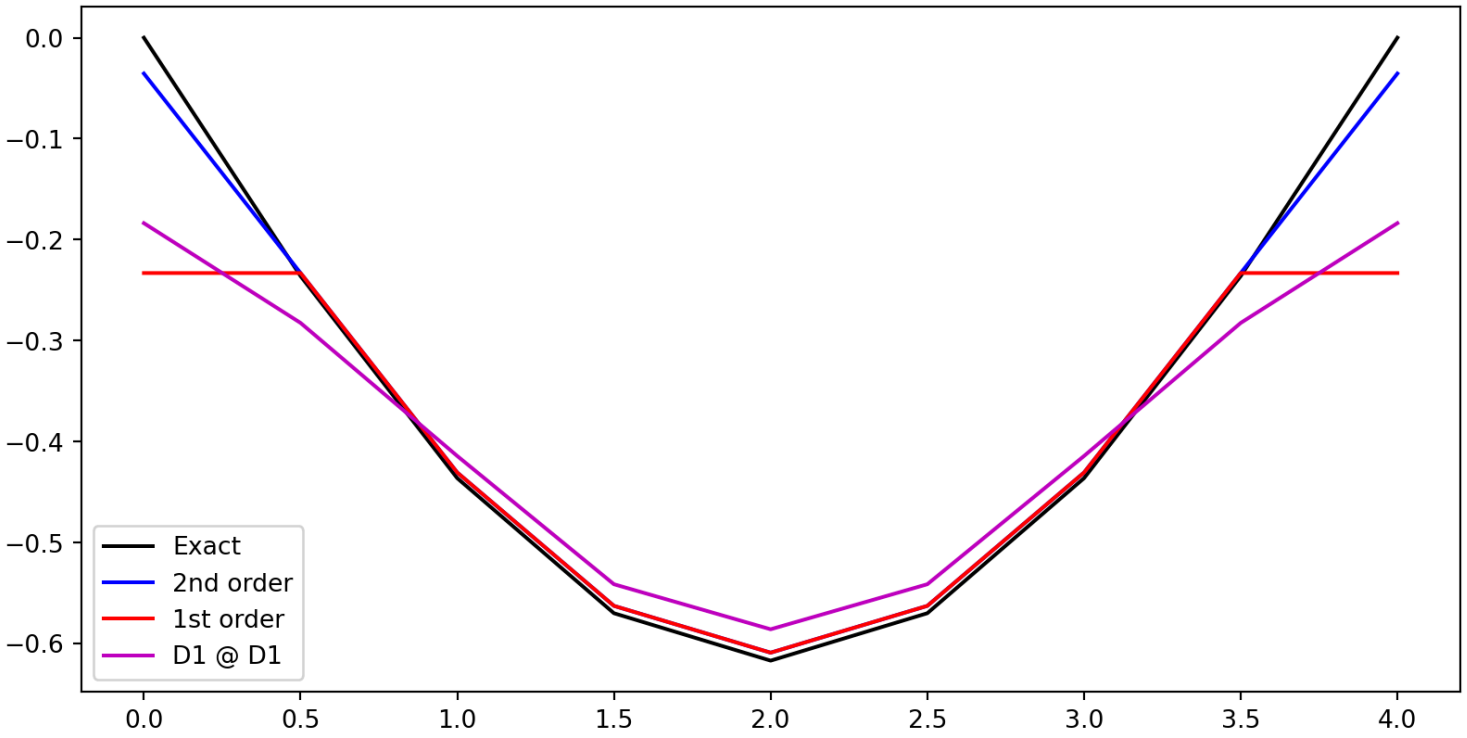
$$u''(t_0) = \frac{-u^{n+3} + 7u^{n+2} - 11u^{n+1} + 5u^n}{4h^2} - \frac{3h}{4}u''' \cdots$$

First order. So $D^{(1)}D^{(1)}$ is still an approximation of the second derivative, only first order accurate near the boundaries.

Test accuracy

Testing $D^{(2)}$, $D^{(2)}$ with only first order boundaries, and $D^{(1)}D^{(1)}$

```
f = np.sin(np.pi*t / T)
d2fe = -(np.pi/T)**2*f
d2f = D2 @ f
d2f1 = D21 @ f
d2f2 = D1 @ D1 @ f
plt.plot(t, d2fe, 'k', t, d2f, 'b', t, d2f1, 'r', t, d2f2, 'm')
plt.legend(['Exact', '2nd order', '1st order', 'D1 @ D1']);
```



Solving equations using FD matrices

Consider the exponential decay model

$$u' + au = 0,\ t \in (0, T], u(0) = I.$$

Without considering boundary conditions we can assemble this problem as

$$(D^{(1)} + a\mathbb{I})\boldsymbol{u} = \boldsymbol{0},$$

where $\mathbb{I} \in \mathbb{R}^{N+1 \times N+1}$ is the identity matrix and $\boldsymbol{0} \in \mathbb{R}^{N+1}$ is a null-vector. The matrix problem is

$$A\boldsymbol{u} = \boldsymbol{0}, \quad A = D^{(1)} + a\mathbb{I}$$

which is only trivially solved to $\boldsymbol{u} = \boldsymbol{0}$ before adding boundary conditions.

Modify the matrix

We need to modify the matrix problem to enforce the boundary condition. Assume first

$$A\boldsymbol{u} = \boldsymbol{b}, \quad \text{for} \quad \boldsymbol{b} \in \mathbb{R}^{N+1}$$

$$\underbrace{\frac{1}{2h}\begin{bmatrix} -3+2ah & 4 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 2ah & 1 & 0 & 0 & 0 & 0 & \cdots \\ 0 & -1 & 2ah & 1 & 0 & 0 & 0 & \cdots \\ \vdots & & & \ddots & & & & \cdots \\ \vdots & 0 & 0 & 0 & -1 & 2ah & 1 & 0 \\ \vdots & 0 & 0 & 0 & 0 & -1 & -2ah & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & -4 & 3+2ah \end{bmatrix}}_{D^{(1)}+a\mathbb{I}} \underbrace{\begin{bmatrix} u^0 \\ u^1 \\ u^2 \\ \vdots \\ u^{N-2} \\ u^{N-1} \\ u^N \end{bmatrix}}_{\boldsymbol{u}} = \underbrace{\begin{bmatrix} b^0 \\ b^1 \\ b^2 \\ \vdots \\ b^{N-2} \\ b^{N-1} \\ b^N \end{bmatrix}}_{\boldsymbol{b}} =$$

In order to enforce that $u(0) = I$, we can modify the first row of the coefficient matrix A and the right hand side vector $\boldsymbol{b}$

Modify matrix

$$\underbrace{\frac{1}{2h}\begin{bmatrix} 2h & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 2ah & 1 & 0 & 0 & 0 & 0 & \cdots \\ 0 & -1 & 2ah & 1 & 0 & 0 & 0 & \cdots \\ \vdots & & & \ddots & & & & \cdots \\ \vdots & 0 & 0 & 0 & -1 & 2ah & 1 & 0 \\ \vdots & 0 & 0 & 0 & 0 & -1 & -2ah & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & -4 & 3+2ah \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} u^0 \\ u^1 \\ u^2 \\ \vdots \\ u^{N-2} \\ u^{N-1} \\ u^N \end{bmatrix}}_{u} = \underbrace{\begin{bmatrix} I \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \end{bmatrix}}_{b}$$

Now the equation in the first row states that $u^0 = I$, whereas the remaining rows are unchanged and solve the central and **implicit** problem for row n

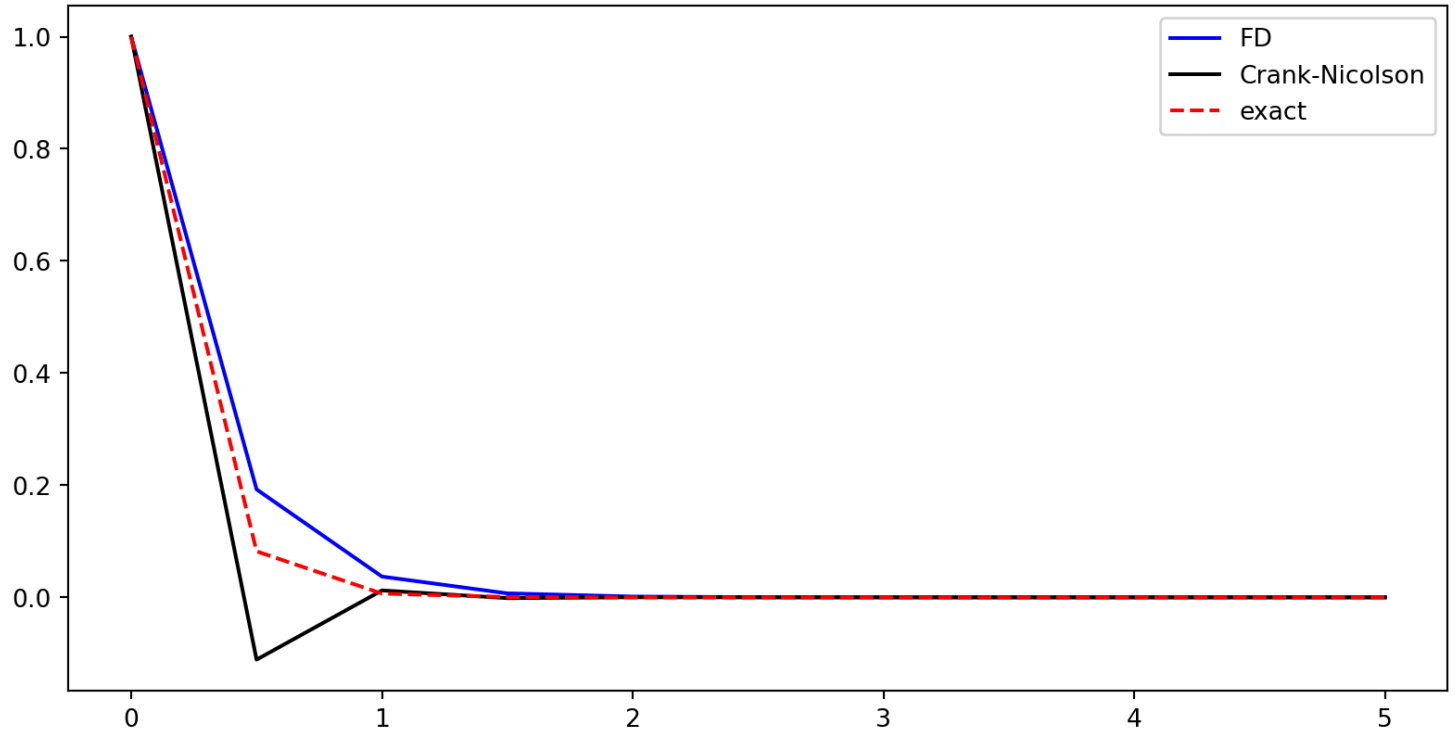
$$\frac{u^{n+1} - u^{n-1}}{2h} + au^n = 0$$

Implement decay problem

```
N = 10
dt = 0.5
a = 5
T = N*dt
t = np.linspace(0, N*dt, N+1)
D1 = sparse.diags([-1, 1], np.array([-1, 1]), (N+1, N+1), 'lil')
D1[-1, -3:] = 1, -4, 3
#D1[-1, -3:] = 0, -2, 2
D1 *= (1/(2*dt))
Id = sparse.eye(N+1)
A = D1 + a*Id
b = np.zeros(N+1)
b[0] = I
A[0, :3] = 1, 0, 0 # boundary condition
A.toarray()
```

```
array([[ 1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [-1.,  5.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0., -1.,  5.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0., -1.,  5.,  1.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0., -1.,  5.,  1.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0., -1.,  5.,  1.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0., -1.,  5.,  1.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0., -1.,  5.,  1.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0., -1.,  5.,  1.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0., -1.,  5.,  1.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1., -4.,  8.]])
```

Compare with exact solution



The accuracy is like for the Crank-Nicolson method discussed in lectures 1-2. However, the FD method is not a recursive **marching method**, since the equation for u^n depends on the solution at u^{n+1}! This **implicit** FD method is unconditionally stable. Normally, only marching methods are analysed for stability.

Generic Finite difference stencils

We have seen that it is quite simple to develop finite difference stencils for any derivative, using either forward or backward points. Can this be generalized?

Yes! Of course it can. The generic Taylor expansion around $x = x_0$ reads

$$u(x) = \sum_{i=0}^{N} \frac{(x - x_0)^i}{i!} u^{(i)}(x_0) + \mathcal{O}((x - x_0)^{N+1}),$$

where $u^{(i)}(x_0) = \frac{d^i u}{dx^i}\big|_{x=x_0}$.

Use only $x = x_0 + mh$, where m is an integer and h is a constant (Δt or Δx)

$$u^{n+m} = \sum_{i=0}^{N} \frac{(mh)^i}{i!} u^{(i)}(x_0) + \mathcal{O}(h^{N+1}),$$

where we use the finite difference notation $u^{n+m} = u(x_0 + mh)$

Generic FD

The Taylor expansions for a given m can be written

$$u^{n+m} = \sum_{i=0}^{N} \frac{(mh)^i}{i!} u^{(i)}(x_0) = \sum_{i=0}^{N} c_{mi} du_i$$

using $c_{mi} = \frac{(mh)^i}{i!}$ and $du_i = u^{(i)}(x_0)$.

This can be understood as a matrix-vector product

$$\boldsymbol{u} = C\boldsymbol{du},$$

where $\boldsymbol{u} = (u^{n+m})_{m=m_0}^{N+m_0}$, $C = (c_{m+m_0,i})_{m,i=0}^{N,N}$ and $\boldsymbol{du} = (du_i)_{i=0}^{N}$. Here m_0 is an integer representing the lowest value of m in the stencil.

For $m_0 = -2$ and $N = 4$:

$$\boldsymbol{u} = (u^{n-2}, u^{n-1}, u^n, u^{n+1}, u^{n+2})^T \quad \boldsymbol{du} = (u^{(0)}, u^{(1)}, u^{(2)}, u^{(3)}, u^{(4)})^T$$

The stencil matrix

For $m_0 = -2$ and $N = 4$ we get 5 Taylor expansions

$$
\begin{aligned}
u^{n-2} &= \sum_{i=0}^{N} \frac{(-2h)^i}{i!} du_i \\
u^{n-1} &= \sum_{i=0}^{N} \frac{(-h)^i}{i!} du_i \\
u^{n} &= u^{n} \\
u^{n+1} &= \sum_{i=0}^{N} \frac{(h)^i}{i!} du_i \\
u^{n+2} &= \sum_{i=0}^{N} \frac{(2h)^i}{i!} du_i
\end{aligned}
$$

The stencil matrix ctd

Expanding the sums these 5 Taylor expansions can be written in matrix form

$$\underbrace{\begin{bmatrix} u^{n-2} \\ u^{n-1} \\ u^{n} \\ u^{n+1} \\ u^{n+2} \end{bmatrix}}_{\boldsymbol{u}} = \underbrace{\begin{bmatrix} \frac{(-2h)^0}{0!} & \frac{(-2h)^1}{1!} & \frac{(-2h)^2}{2!} & \frac{(-2h)^3}{3!} & \frac{(-2h)^4}{4!} \\ \frac{(-h)^0}{0!} & \frac{(-h)^1}{1!} & \frac{(-h)^2}{2!} & \frac{(-h)^3}{3!} & \frac{(-h)^4}{4!} \\ 1 & 0 & 0 & 0 & 0 \\ \frac{(h)^0}{0!} & \frac{(h)^1}{1!} & \frac{(h)^2}{2!} & \frac{(h)^3}{3!} & \frac{(h)^4}{4!} \\ \frac{(2h)^0}{0!} & \frac{(2h)^1}{1!} & \frac{(2h)^2}{2!} & \frac{(2h)^3}{3!} & \frac{(2h)^4}{4!} \end{bmatrix}}_{C} \underbrace{\begin{bmatrix} du_0 \\ du_1 \\ du_2 \\ du_3 \\ du_4 \end{bmatrix}}_{\boldsymbol{du}}$$

Invert to obtain

$$\boldsymbol{du} = C^{-1}\boldsymbol{u}$$

This is how we can get any derivative stencil! Remember du_i is an approximation to the i'th derivative.

Second order second derivative stencil

We have been using the following stencil

$$du_2 = u^{(2)}(x_0) = \frac{u^{n+1} - 2u^n + u^{n-1}}{h^2}.$$

Let's derive this with the approach above. The scheme is central and second order so we use $m_0 = -1$ and $N = 2$ (hence $m = (-1, 0, 1)$). Insert into the recipe for C

$$C = \begin{bmatrix} 1 & -h & \frac{h^2}{2} \\ 1 & 0 & 0 \\ 1 & h & \frac{h^2}{2} \end{bmatrix}$$

In Sympy

```
import sympy as sp
x, h = sp.symbols('x,h')
C = sp.Matrix([[1, -h, h**2/2], [1, 0, 0], [1, h, h**2/2]])
```

Print C matrix

```
C
```

$$\begin{bmatrix} 1 & -h & \frac{h^2}{2} \\ 1 & 0 & 0 \\ 1 & h & \frac{h^2}{2} \end{bmatrix}$$

Print C^{-1} matrix

```
C.inv()
```

$$\begin{bmatrix} 0 & 1 & 0 \\ -\frac{1}{2h} & 0 & \frac{1}{2h} \\ \frac{1}{h^2} & -\frac{2}{h^2} & \frac{1}{h^2} \end{bmatrix}$$

The last row in C^{-1} represents u''! The middle row represents a second order central u'.

Create a vector for $\boldsymbol{u}$ and print u' and u'

```
u = sp.Function('u')
coef = sp.Matrix([u(x-h), u(x), u(x+h)])
```

```
display((C.inv()[1, :] @ coef)[0])
```

$$-\frac{u(-h+x)}{2h} + \frac{u(h+x)}{2h}$$

```
display((C.inv()[2, :] @ coef)[0])
```

$$-\frac{2u(x)}{h^2} + \frac{u(-h+x)}{h^2} + \frac{u(h+x)}{h^2}$$