A Hands-On Introduction to Computational Fluid Dynamics

Mohammad Umair[†], Gerardo Zampino[†] and Ricardo Vinuesa[†]

[†]FLOW, Engineering Mechanics, KTH Royal Institute of Technology, Sweden

VETENSKAP OCH KONST

MODELAIR

SERC Swedish e-Science Research Centre

Computational Fluid Dynamics

17th Century
France and England

PURE
EXPERIMENTS

Computational Fluid Dynamics

17th Century
France and England

PURE
EXPERIMENTS

PURE
THEORY

18th – 19th Century
Europe

Computational Fluid Dynamics

17th Century
France and England

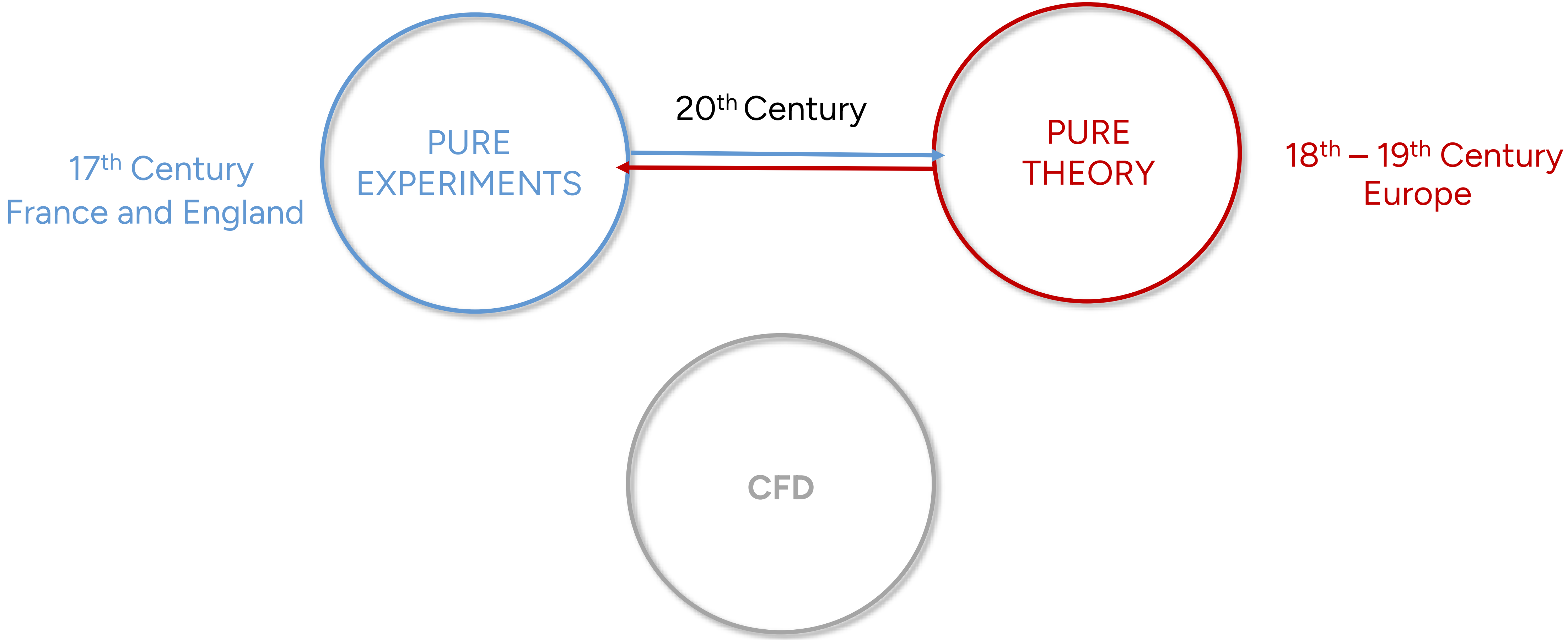
PURE
EXPERIMENTS

20th Century

PURE
THEORY

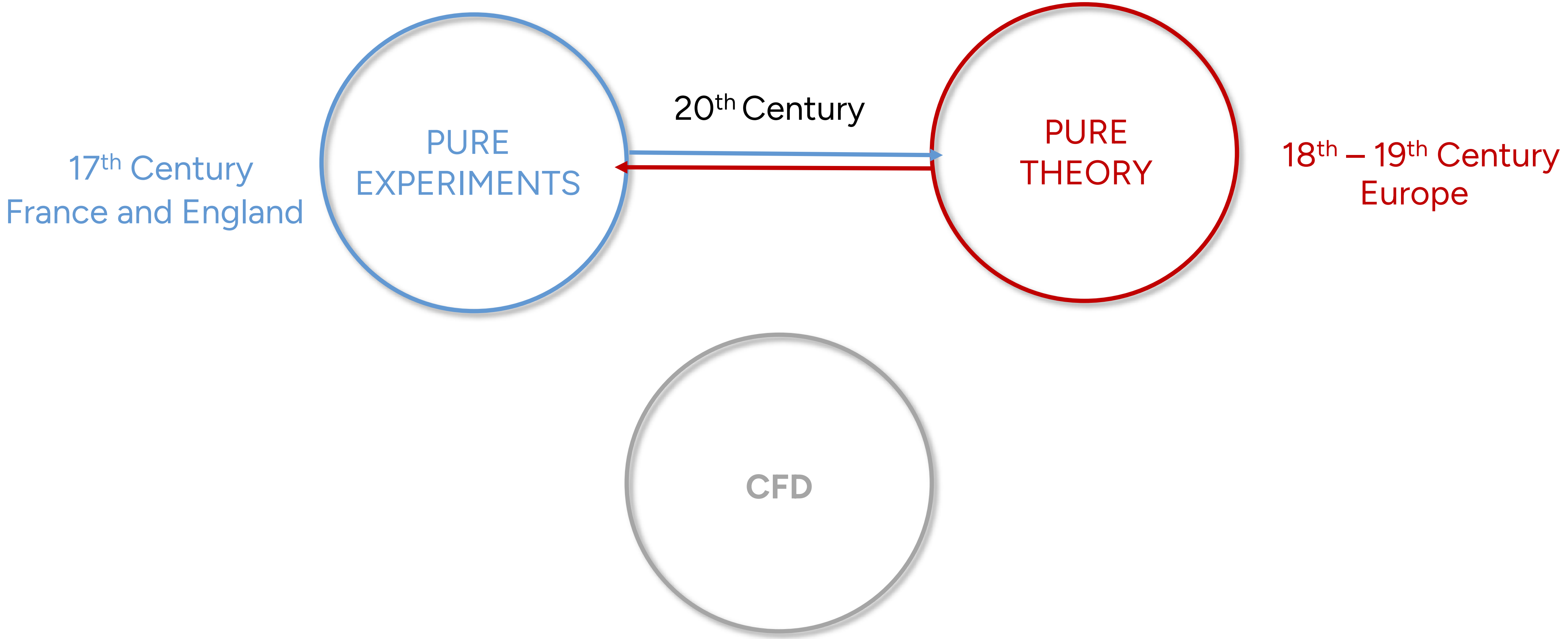
18th – 19th Century
Europe

Computational Fluid Dynamics



A new *"third approach"* in the philosophical study of fluid dynamics

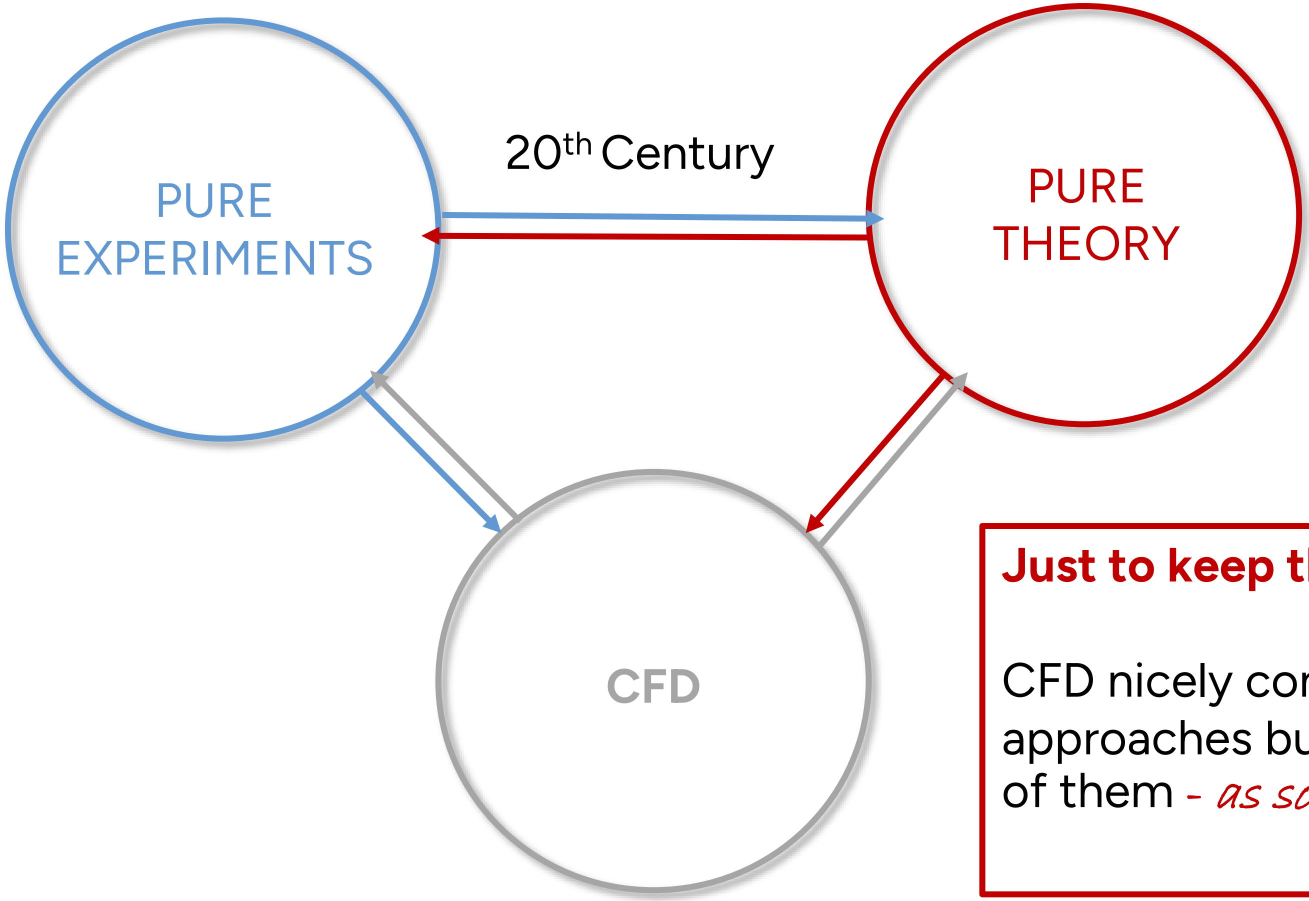
Computational Fluid Dynamics



A new *"third approach"* in the philosophical study of fluid dynamics

(Nothing more than that!)

Computational Fluid Dynamics



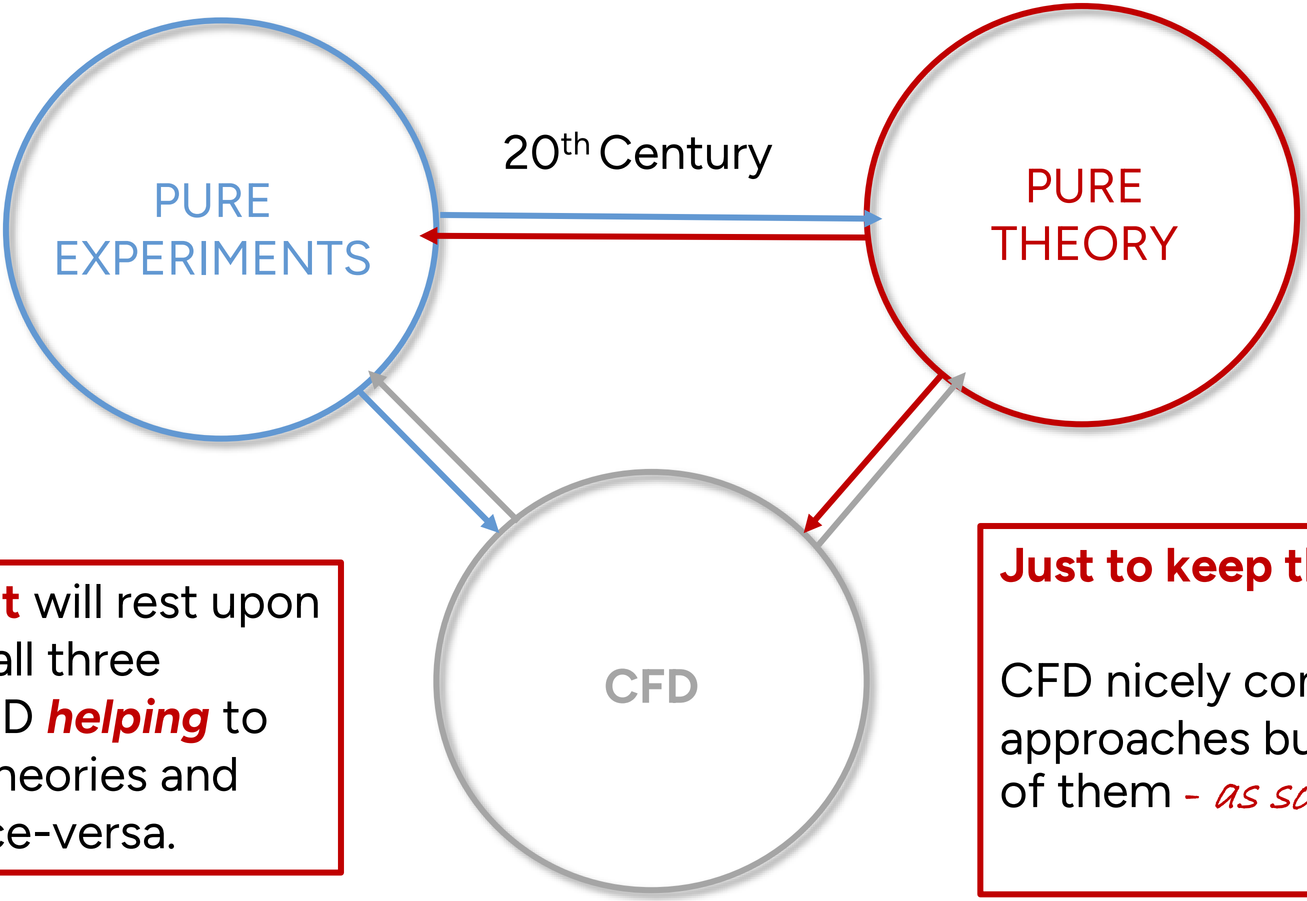
Just to keep things in perspective:

CFD nicely compliments the other two approaches but will ***never*** replace either of them - *as sometimes suggested!*

A new *"third approach"* in the philosophical study of fluid dynamics

(Nothing more than that!)

Computational Fluid Dynamics



Future advancement will rest upon a proper balance of all three approaches, with CFD ***helping*** to interpret results of theories and experiments, and vice-versa.

Just to keep things in perspective:

CFD nicely compliments the other two approaches but will ***never*** replace either of them - *as sometimes suggested!*

A new *"third approach"* in the philosophical study of fluid dynamics
(Nothing more than that!)

CFD as a research tool

CFD results are directly analogous to Wind Tunnel results.

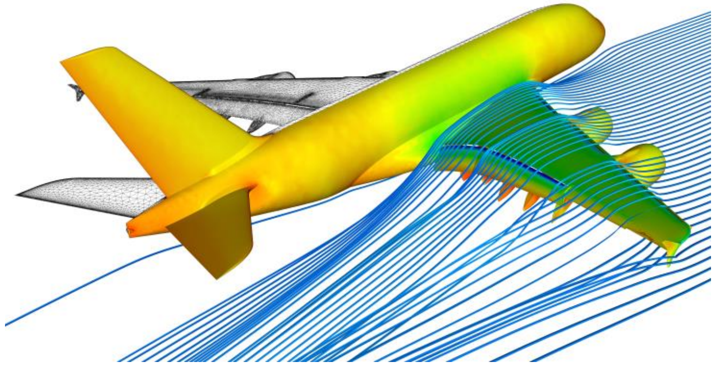
However, unlike a wind tunnel a computer program can be carried everywhere or can be accessed sitting *1000 miles away!*

A CFD program is, therefore, a *"readily transportable wind tunnel"*, where you can carry out *"numerical experiments".*

CFD as a research tool: Who cares?

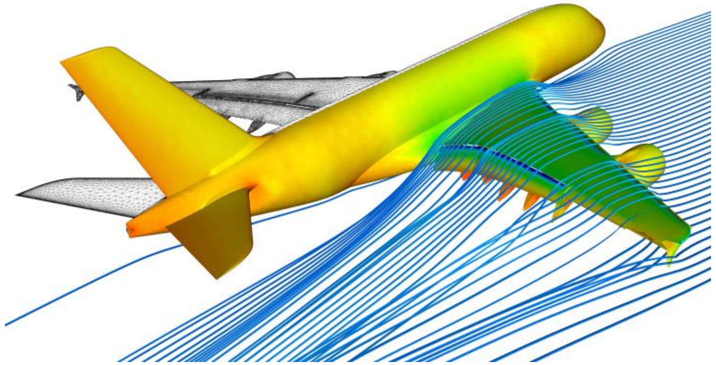
CFD as a research tool: Who cares?

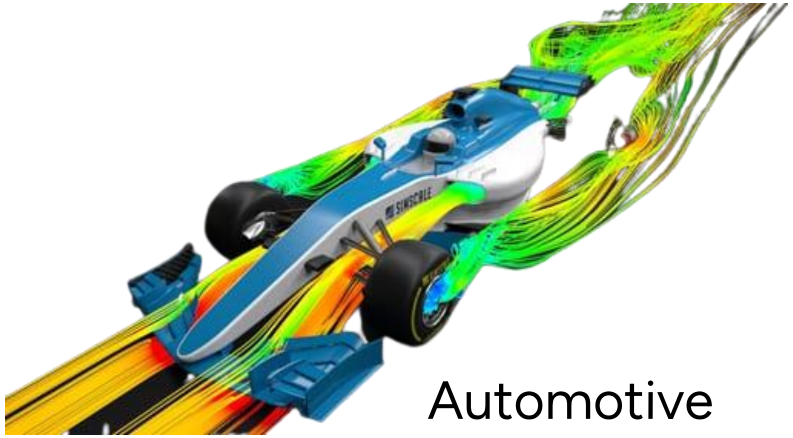
Aviation



CFD as a research tool: Who cares?

Aviation

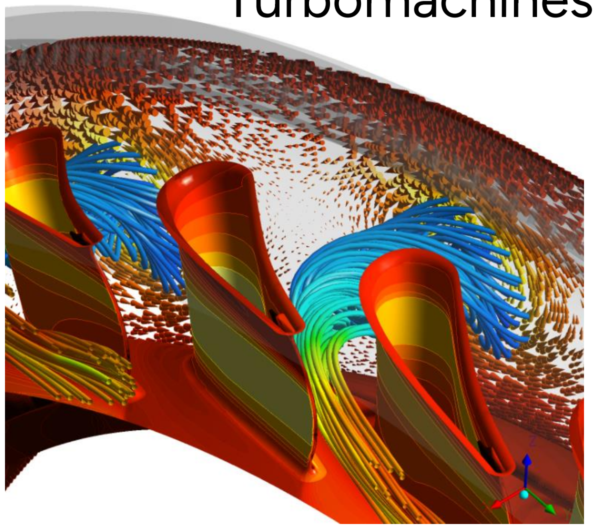




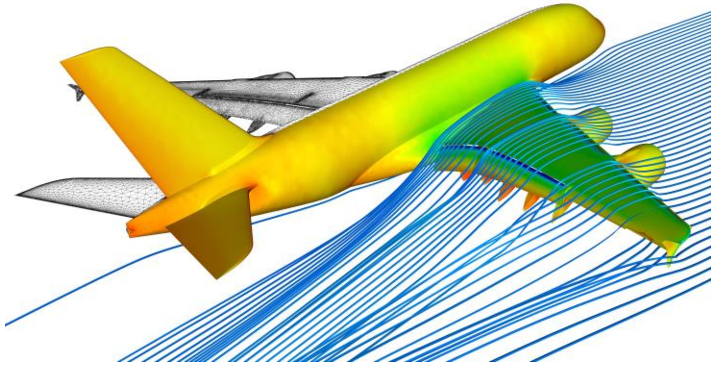
Automotive

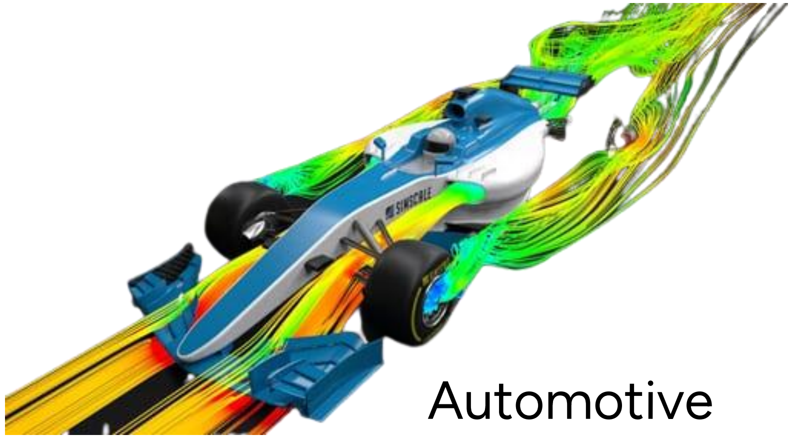
CFD as a research tool: Who cares?

Turbomachines



Aviation

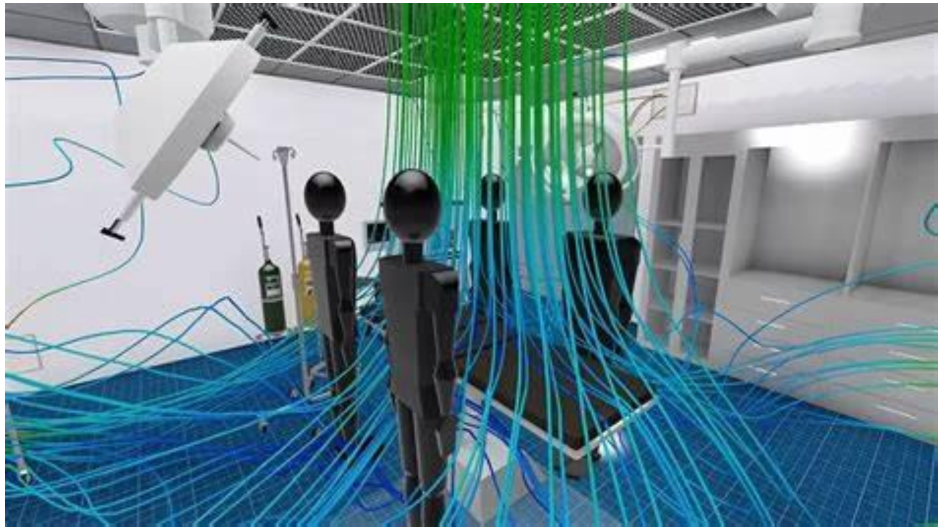




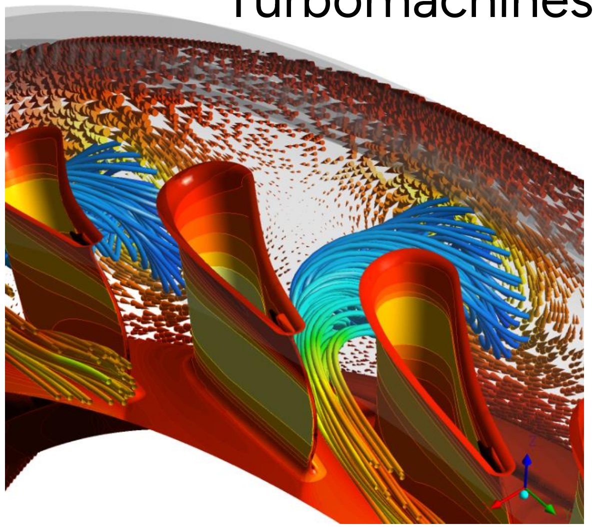
Automotive

CFD as a research tool: Who cares?

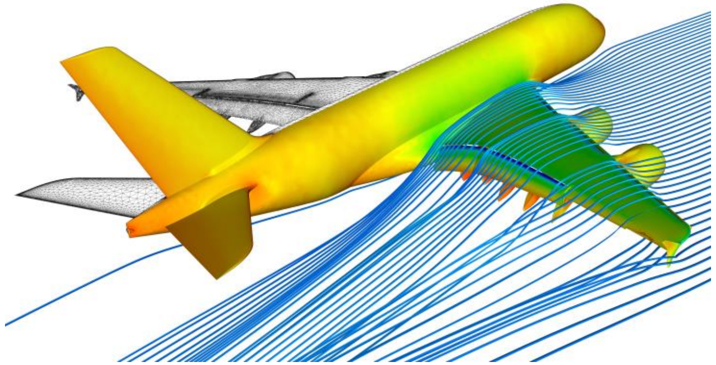
HVAC

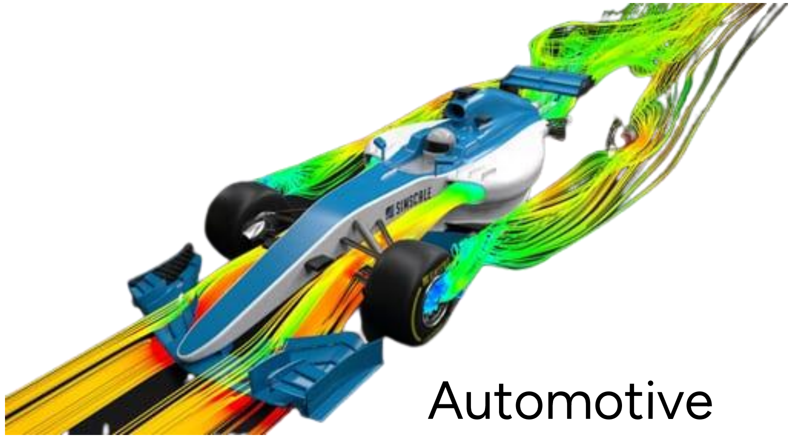


Turbomachines



Aviation

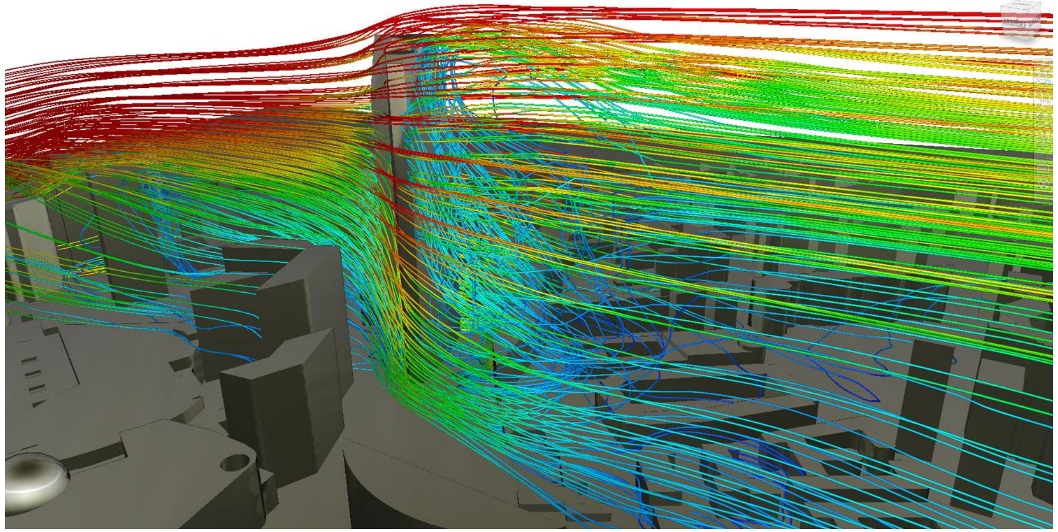




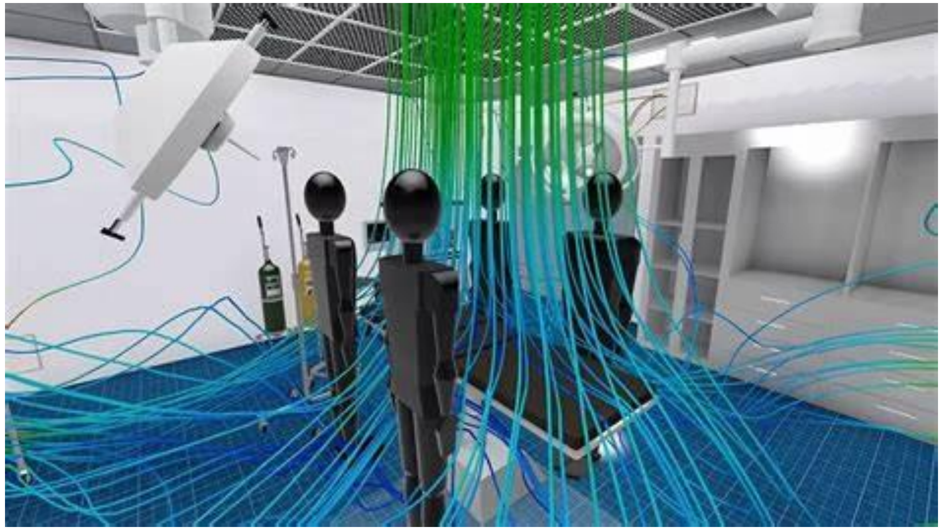
Automotive

CFD as a research tool: Who cares?

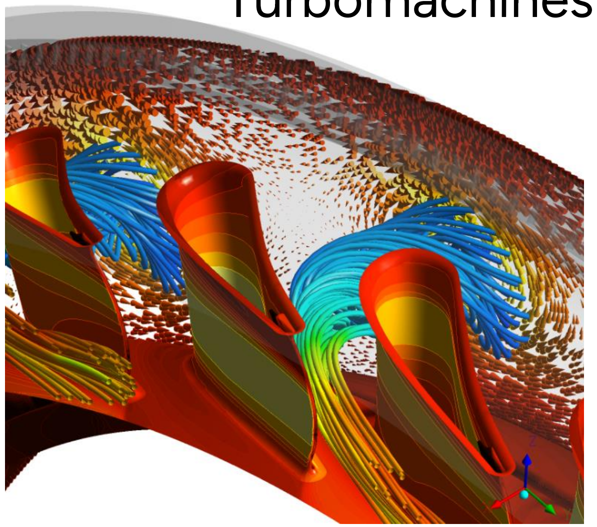
Cities

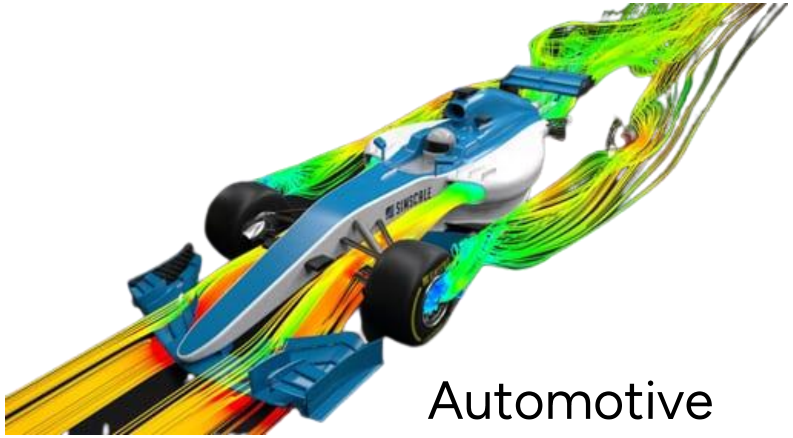


HVAC



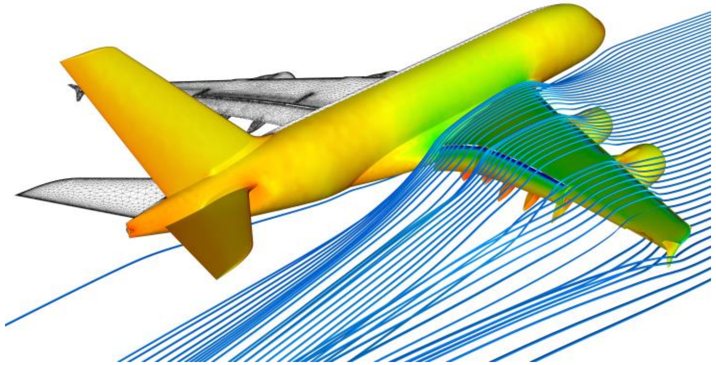
Turbomachines





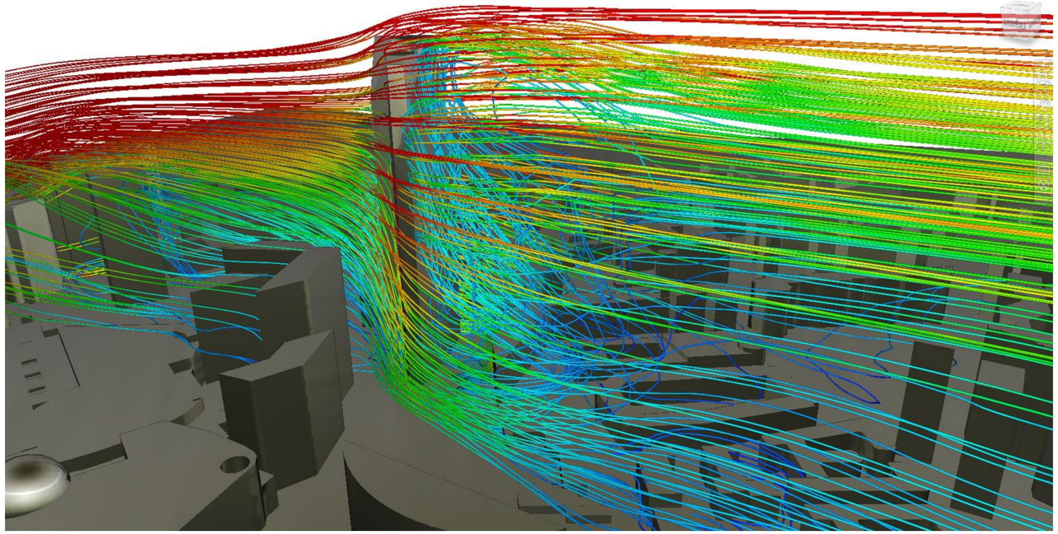
Automotive

Aviation

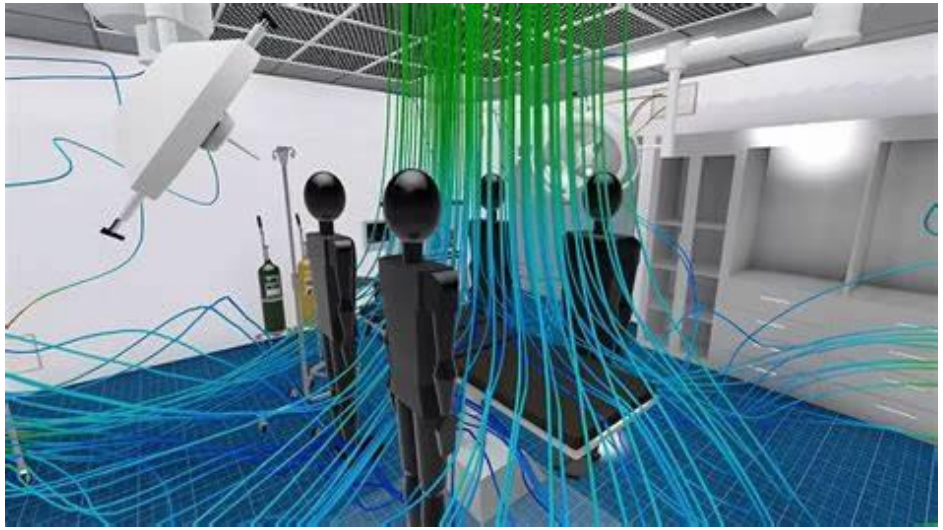


CFD as a research tool: Who cares?

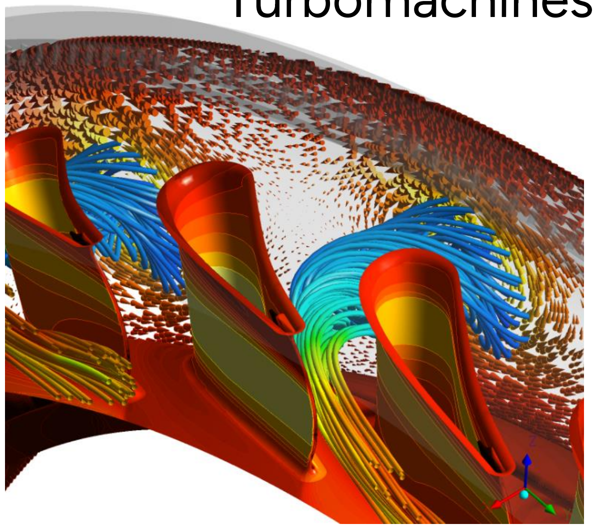
Cities

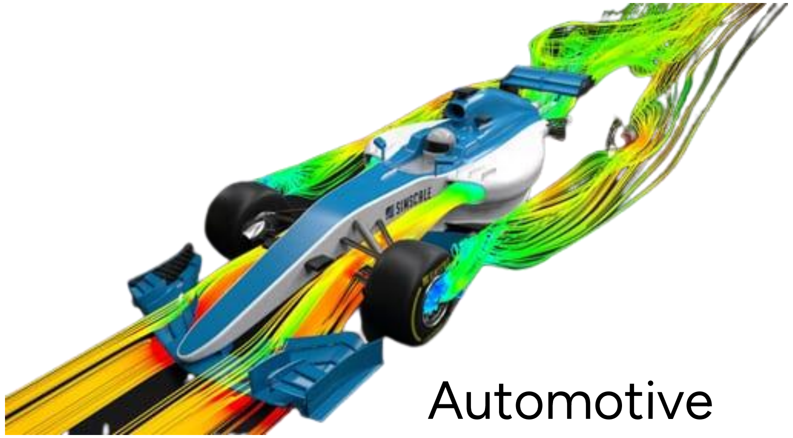


HVAC



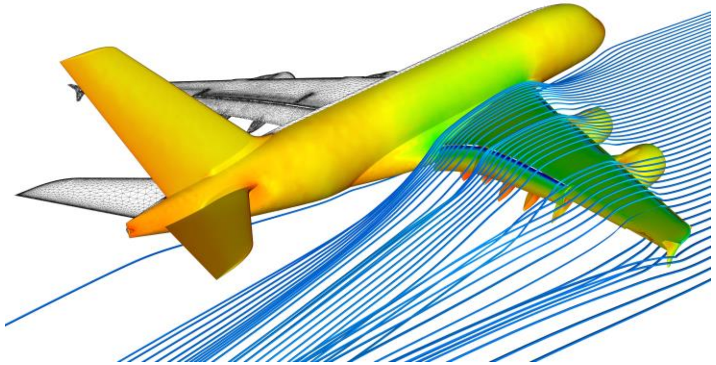
Turbomachines





Automotive

Aviation

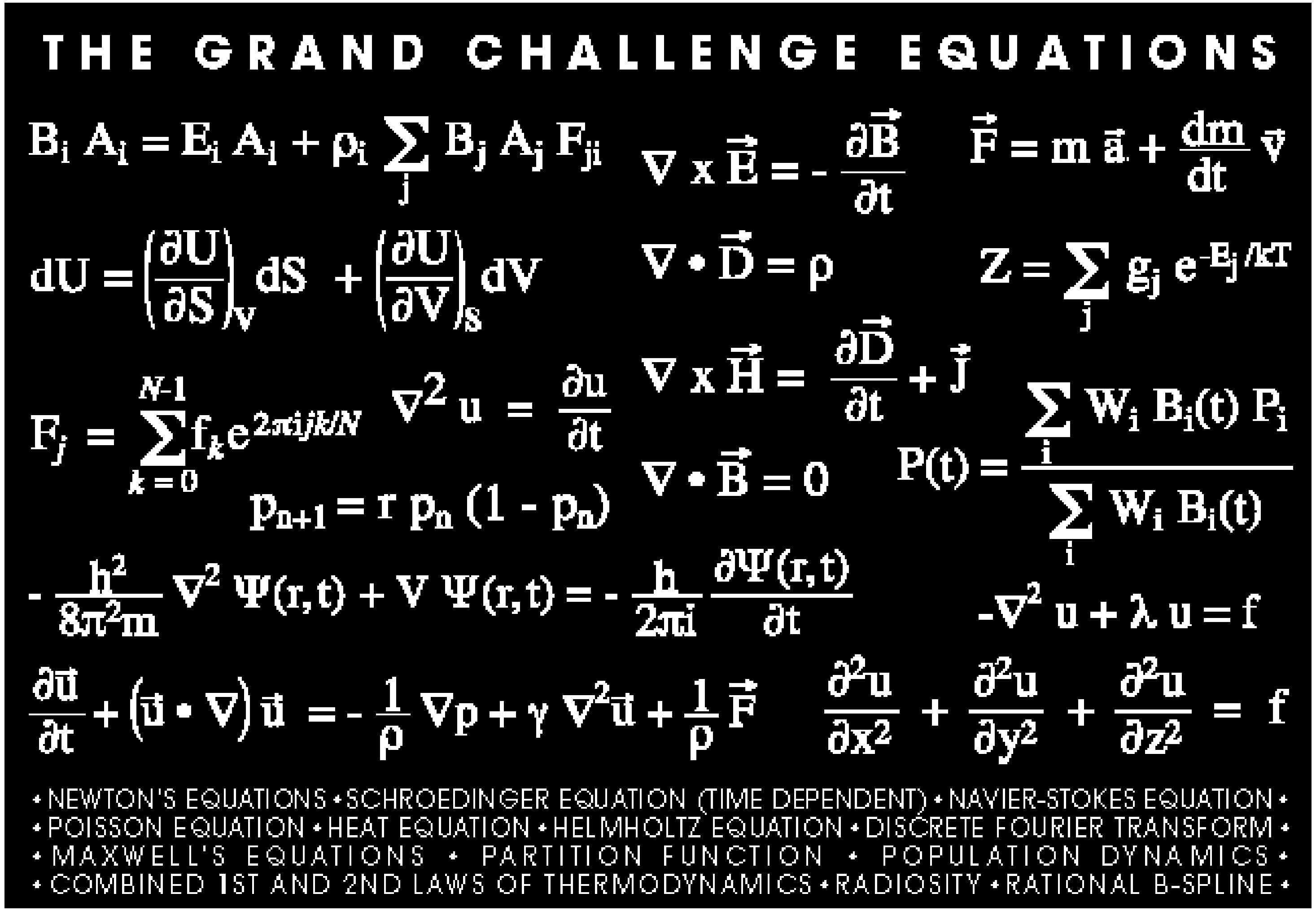


+ many more !

So How to CFD?

CFD involves:

- Identifying the physical phenomenon, that includes Governing equations, Initial & Boundary conditions.
- Breaking down the continuous problem to a discrete representation.
- Solving the discrete set of equations using adequate numerical methods.
- Post processing the results.



What are we solving?

MODELAIR

Navier Stokes equations

Continuity equation: $$\partial_t \rho + \nabla.(\rho \boldsymbol{u}) = 0$$

Momentum equation: $$\partial_t \rho \boldsymbol{u} + \nabla.(\rho \boldsymbol{u} \otimes \boldsymbol{u}) + \nabla p - \nabla.\boldsymbol{\tau} = \boldsymbol{f}$$

Energy equation: $$\partial_t E + \nabla.[(E + p)\boldsymbol{u}] - \nabla.(\boldsymbol{\tau u}) - \nabla.(\kappa \nabla T) = S$$

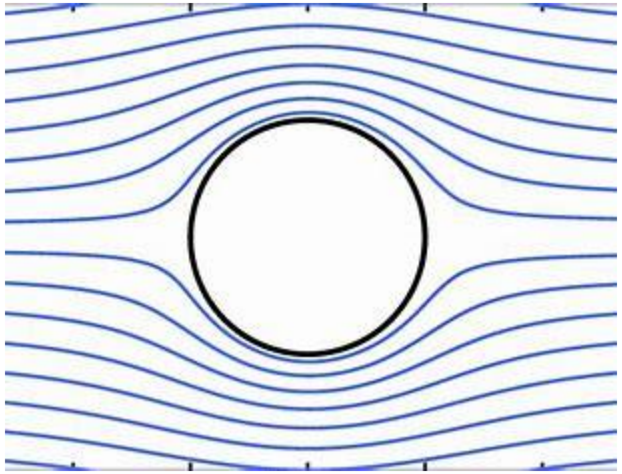
Depending on the nature of physics governing the fluid motion one or more terms might be negligible.

Presence of each term and their combinations determines the appropriate solution algorithm and the numerical procedure.

Classification of PDEs

Elliptic

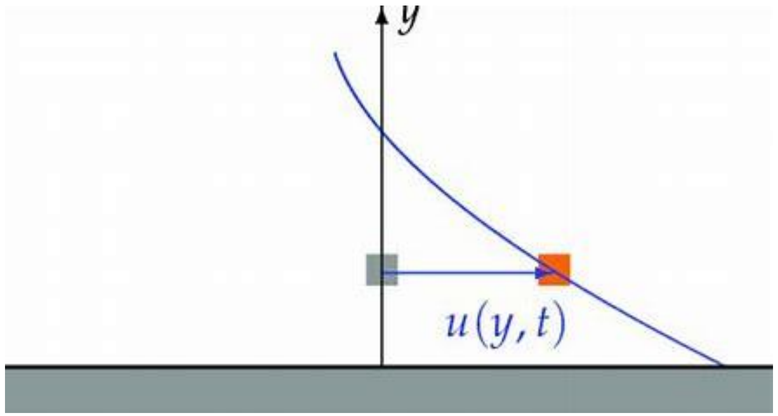
$$\nabla^2 u = 0$$



Potential Flow

Parabolic

$$\frac{\partial u}{\partial t} = \nu \nabla^2 u$$



Flow over an oscillating plate
(Stokes 2^{nd} Problem)

Hyperbolic

$$\frac{\partial^2 u}{\partial t^2} = c^2 \nabla^2 u$$

Wave motion

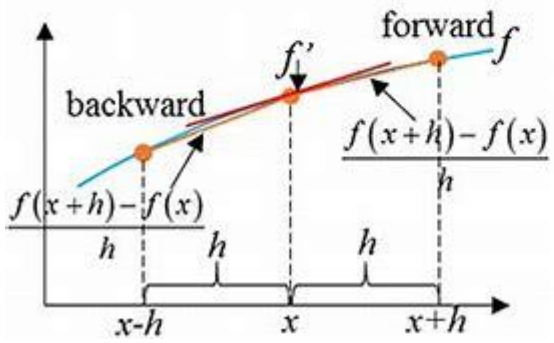
Equations belonging to each of these classifications behave in different ways both *physically* and *numerically*.

Techniques for Numerical Discretization

Governing Equations $\xrightarrow{\text{Discretization}}$ Numerical Analogue

Commonly used discretization methods

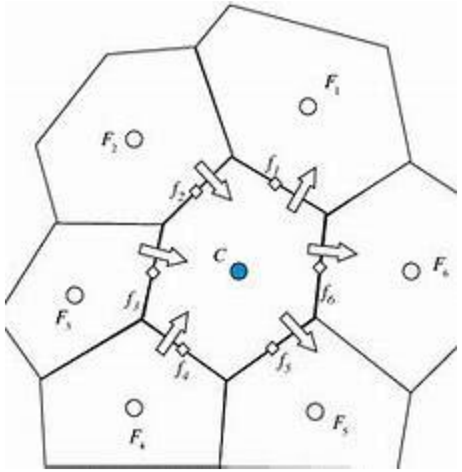
Finite Difference Methods



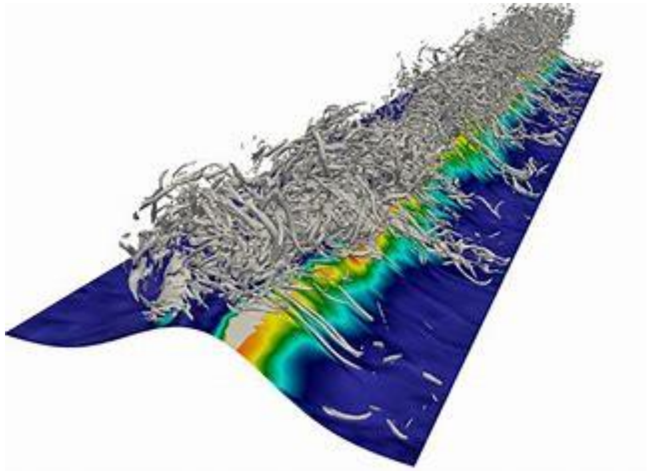
Spectral Methods

$$
\begin{aligned}
G(f) &= \Im\{\cos(2\pi A t)\} = \int_{-\infty}^{\infty} \frac{e^{i2\pi At} + e^{-i2\pi At}}{2} e^{-i2\pi ft}\,dt \\
&= \frac{1}{2}\left[\int_{-\infty}^{\infty} e^{i2\pi At} e^{-i2\pi ft}\,dt + \int_{-\infty}^{\infty} e^{-i2\pi At} e^{-i2\pi ft}\,dt\right] \\
&= \frac{1}{2}\left[\delta(f-A) + \delta(f+A)\right]
\end{aligned}
$$

Finite Volume Methods



Finite/Spectral Element Methods



Techniques for Numerical Discretization

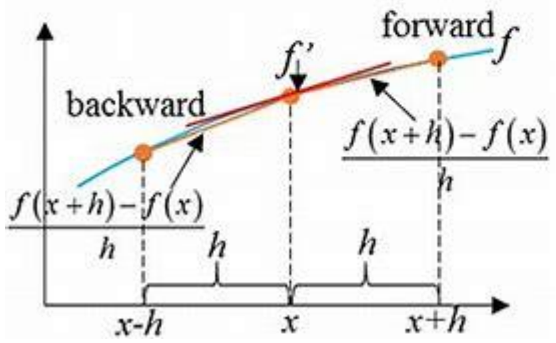
Governing Equations $\xrightarrow{\text{Discretization}}$ Numerical Analogue

Commonly used discretization methods

TODAY

TOMORROW

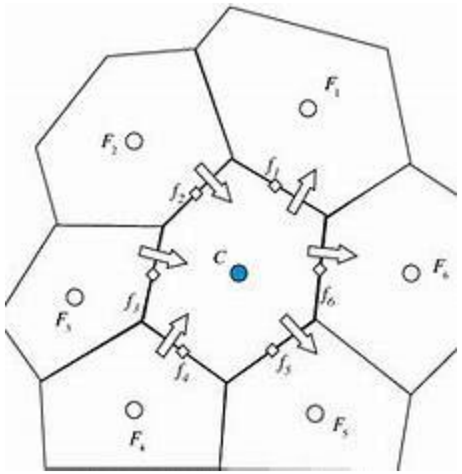
Finite Difference Methods



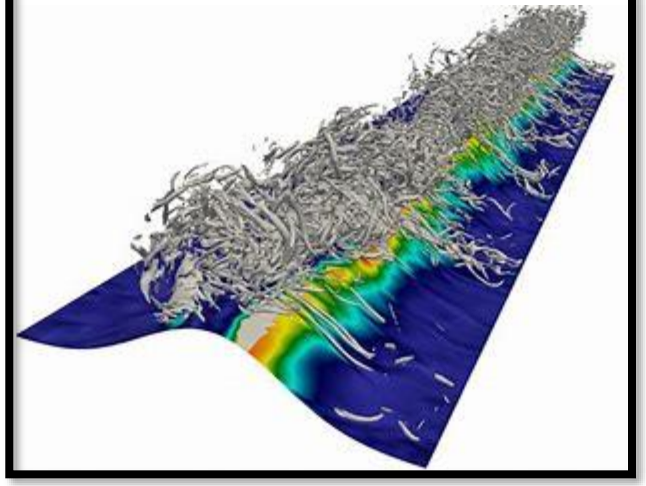
Spectral Methods

$$G(f)=\Im\{\cos(2\pi At)\}=\int_{-\infty}^{\infty}\frac{e^{i2\pi At}+e^{-i2\pi At}}{2}e^{-i2\pi ft}\,dt$$
$$=\frac{1}{2}\left[\int_{-\infty}^{\infty}e^{i2\pi At}e^{-i2\pi ft}\,dt+\int_{-\infty}^{\infty}e^{-i2\pi At}e^{-i2\pi ft}\,dt\right]$$
$$=\frac{1}{2}\left[\delta(f-A)+\delta(f+A)\right]$$

Finite Volume Methods



Finite/Spectral Element Methods



Finite Difference Method

Definition of a derivative

Exact

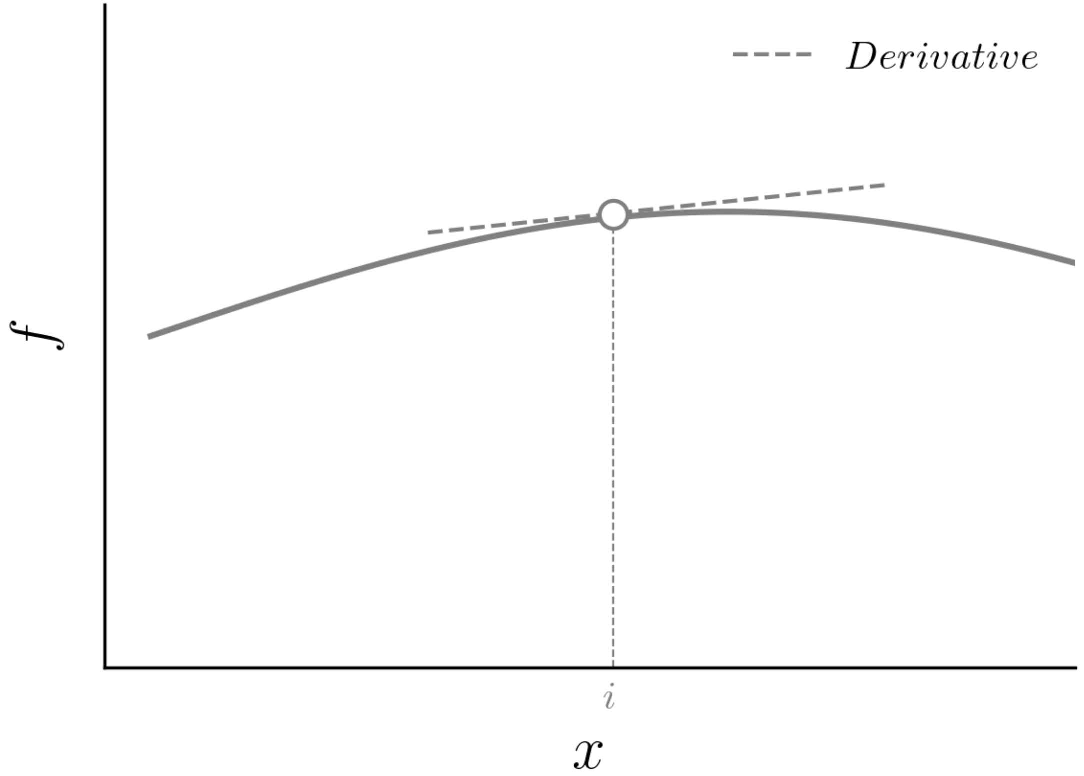
$$\frac{df}{dx} = \lim_{dx \to 0} \frac{f(x + dx) - f(x)}{dx}$$

Finite Difference Method

Definition of a derivative

Exact

$$\frac{df}{dx} = \lim_{dx \to 0} \frac{f(x + dx) - f(x)}{dx}$$



Finite Difference Method

MODELAIR

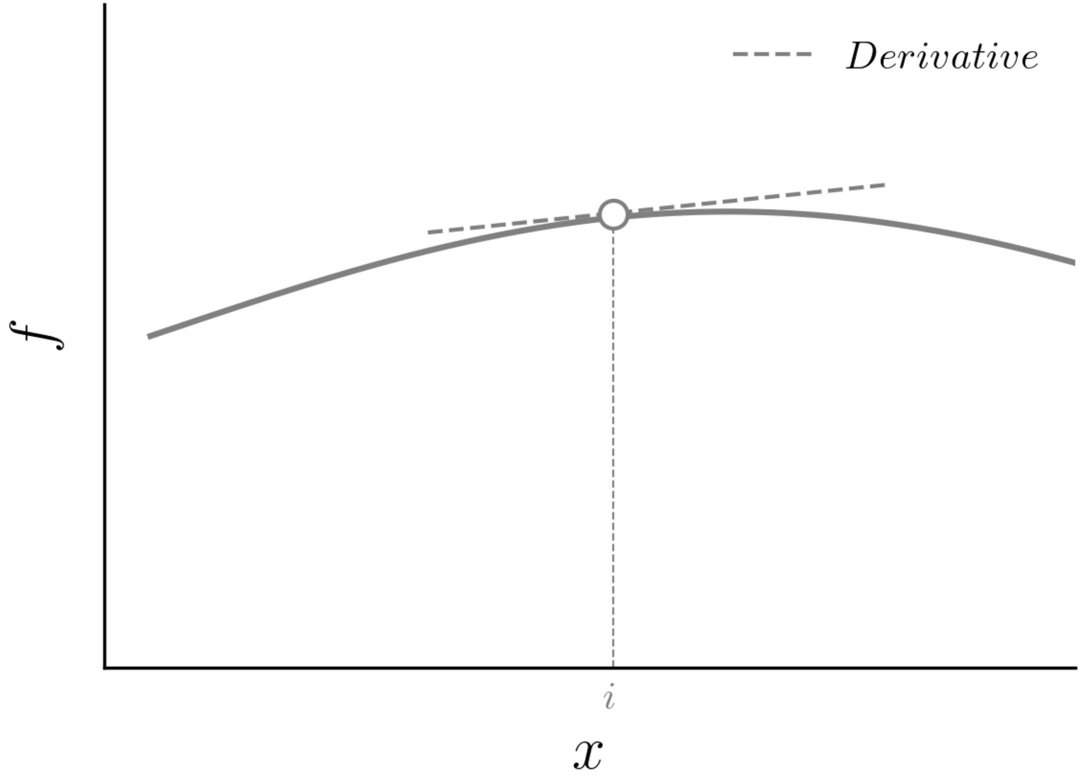
Definition of a derivative

Exact

$$\frac{df}{dx} = \lim_{dx \to 0} \frac{f(x + dx) - f(x)}{dx}$$

$$\frac{df}{dx} = \lim_{dx \to 0} \frac{f(x) - f(x - dx)}{dx}$$

$$\frac{df}{dx} = \lim_{dx \to 0} \frac{f(x + dx) - f(x - dx)}{2dx}$$



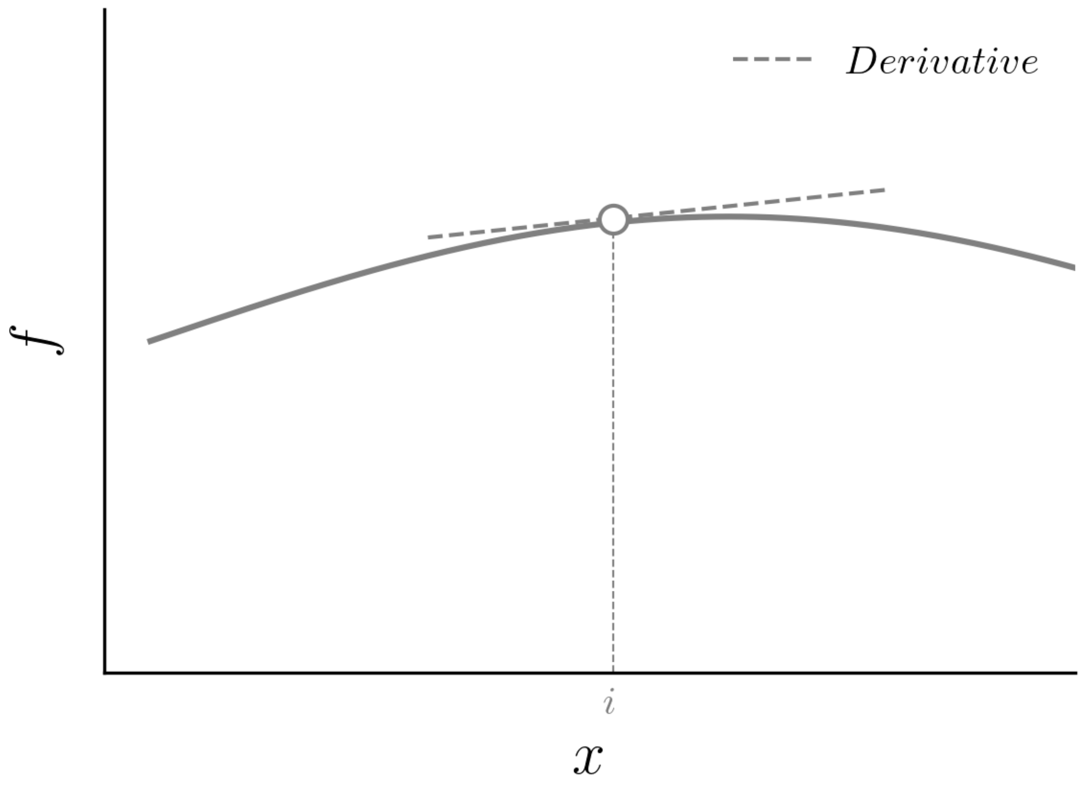
Finite Difference Method

Finite Difference

$$\frac{df}{dx} \approx \frac{f(x+dx)-f(x)}{dx}$$

$$\frac{df}{dx} \approx \frac{f(x)-f(x-dx)}{dx}$$

$$\frac{df}{dx} \approx \frac{f(x+dx)-f(x-dx)}{2dx}$$



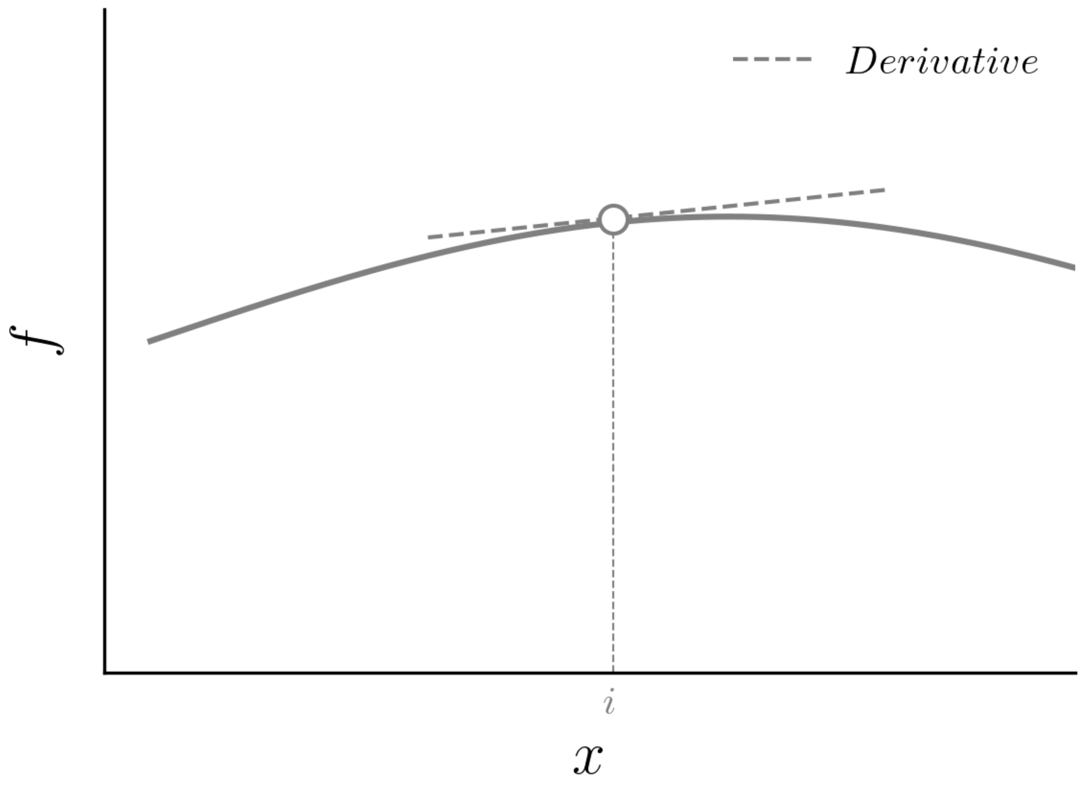
Finite Difference Method

Finite Difference

$$\frac{df}{dx} \approx \frac{f(x+dx)-f(x)}{dx}$$

$$\frac{df}{dx} \approx \frac{f(x)-f(x-dx)}{dx}$$

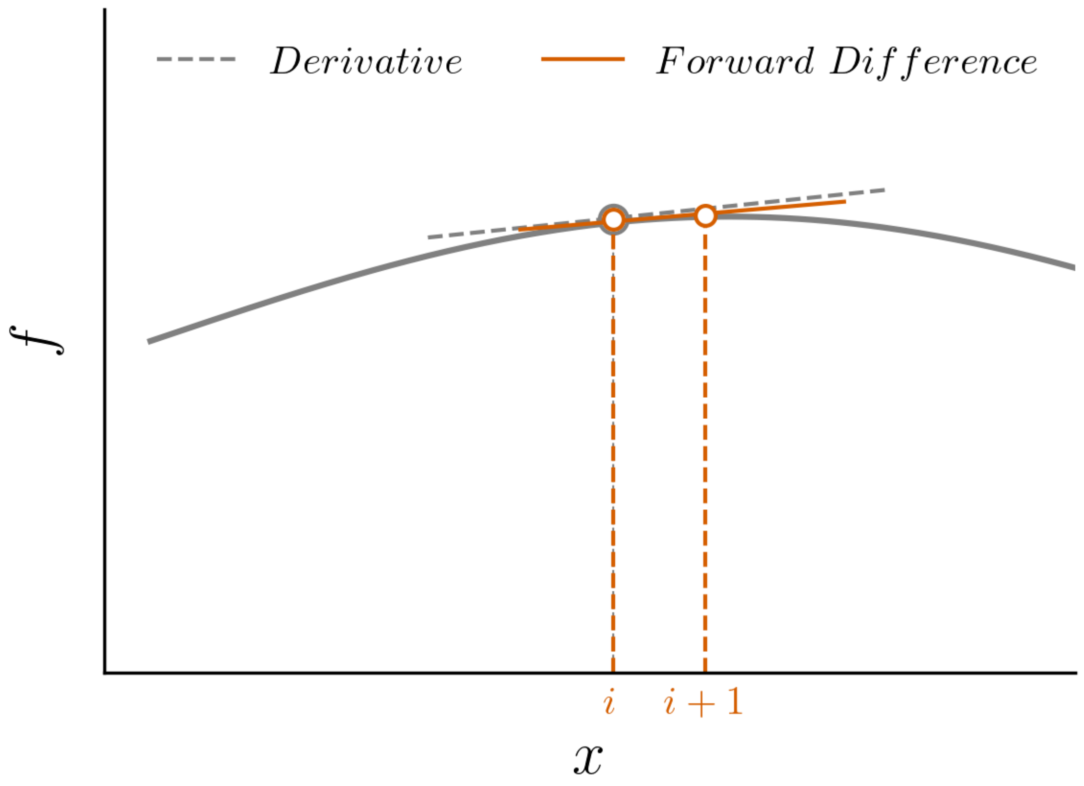
$$\frac{df}{dx} \approx \frac{f(x+dx)-f(x-dx)}{2dx}$$



Finite Difference Method

Finite Difference

Forward $$\frac{df}{dx} \approx \frac{f(x+dx) - f(x)}{dx}$$



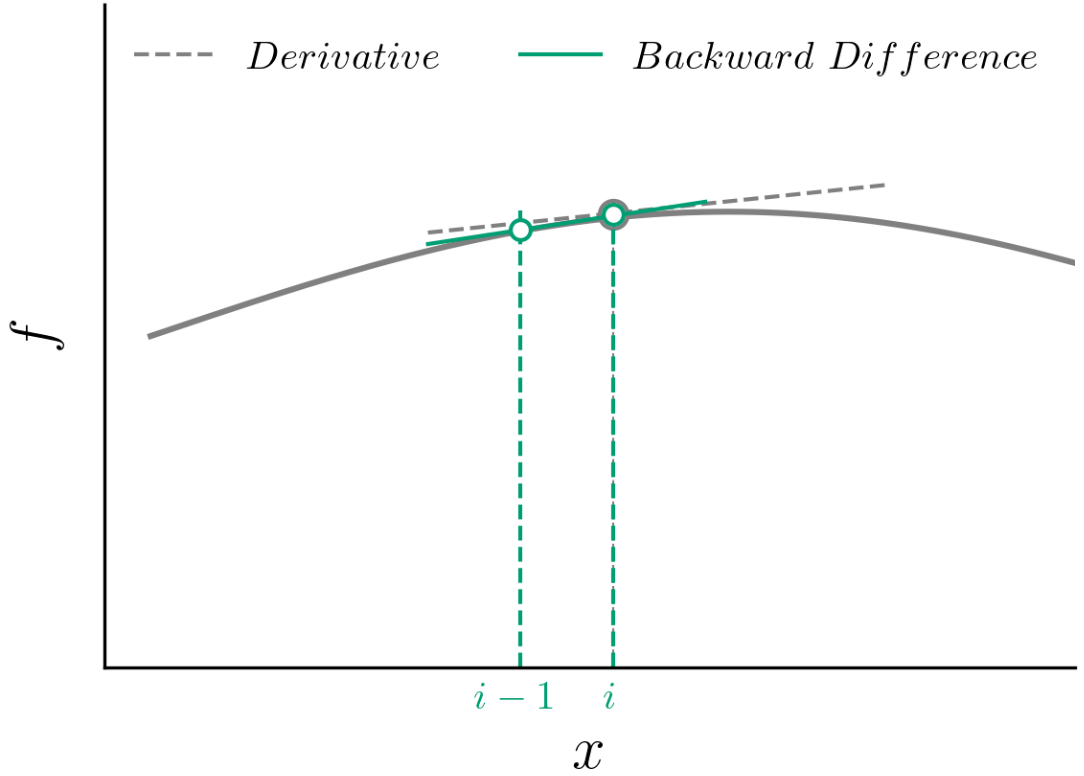
Finite Difference Method

MODELAIR

Finite Difference

Forward $$\frac{df}{dx} \approx \frac{f(x+dx) - f(x)}{dx}$$

Backward $$\frac{df}{dx} \approx \frac{f(x) - f(x-dx)}{dx}$$



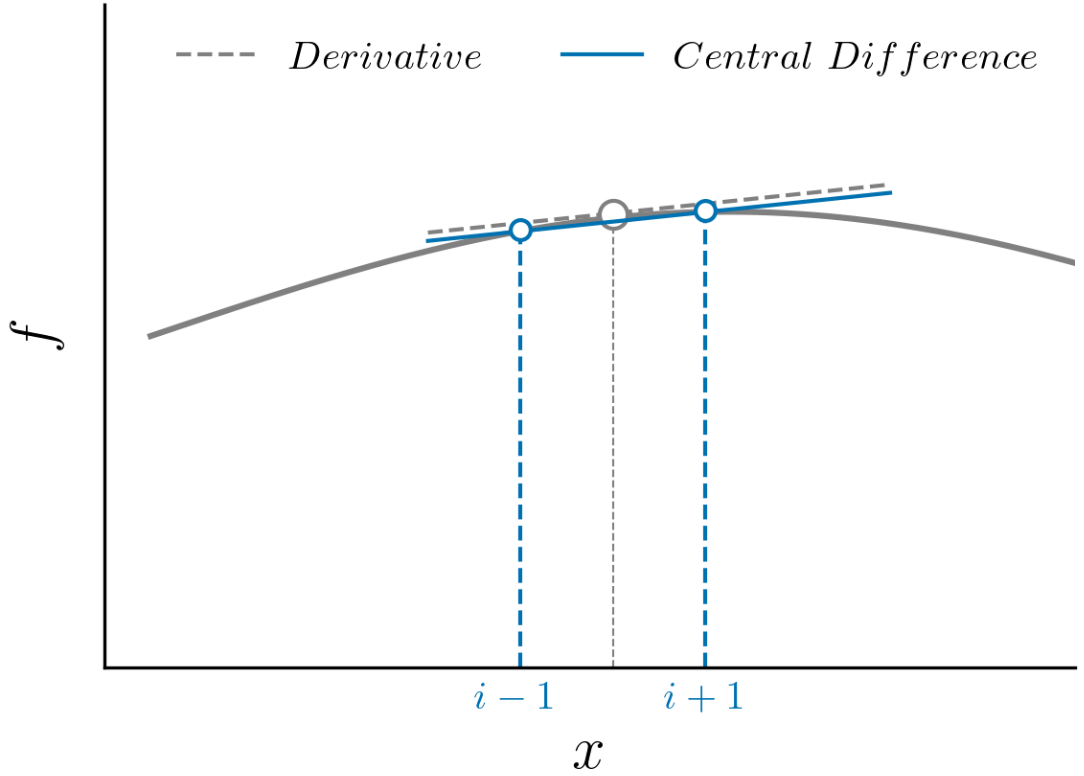
Finite Difference Method

Finite Difference

Forward $$\frac{df}{dx} \approx \frac{f(x+dx) - f(x)}{dx}$$

Backward $$\frac{df}{dx} \approx \frac{f(x) - f(x-dx)}{dx}$$

Centered $$\frac{df}{dx} \approx \frac{f(x+dx) - f(x-dx)}{2dx}$$



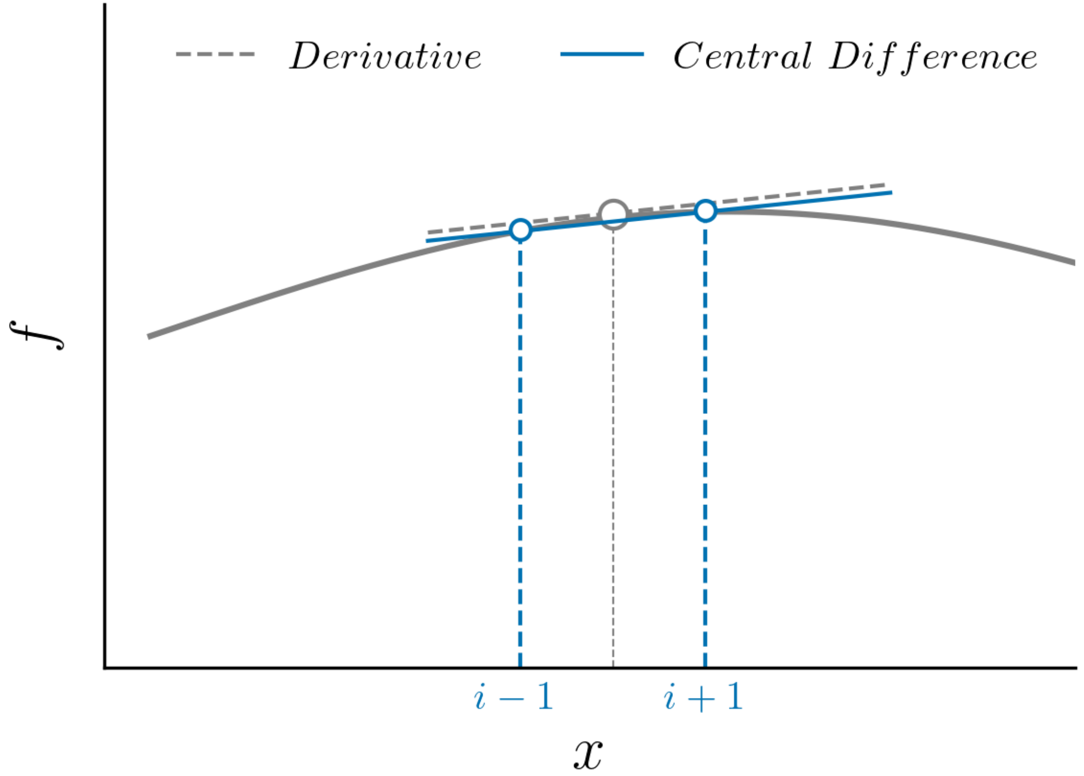
Finite Difference Method

Finite Difference

Forward $$\frac{df}{dx} \approx \frac{f(x+dx) - f(x)}{dx}$$

Backward $$\frac{df}{dx} \approx \frac{f(x) - f(x-dx)}{dx}$$

Centered $$\frac{df}{dx} \approx \frac{f(x+dx) - f(x-dx)}{2dx}$$



How good is the approximation?

How good is the approximation?

How good is the approximation?

Taylor Series

$$f(x_0 + dx) = f(x_0) + \frac{df}{dx}\Big|_{x_0} dx + \frac{1}{2!}\frac{d^2 f}{dx^2}\Big|_{x_0} dx^2 + \frac{1}{3!}\frac{d^3 f}{dx^3}\Big|_{x_0} dx^3 + \cdots$$

How good is the approximation?

Taylor Series

$$f(x_0+dx) = f(x_0) + \frac{df}{dx}\bigg|_{x_0} dx + \frac{1}{2!}\frac{d^2f}{dx^2}\bigg|_{x_0} dx^2 + \frac{1}{3!}\frac{d^3f}{dx^3}\bigg|_{x_0} dx^3 + \cdots$$

$$\frac{f(x_0+dx) - f(x_0)}{dx} = \frac{df}{dx}\bigg|_{x_0} + \frac{1}{2!}\frac{d^2f}{dx^2}\bigg|_{x_0} dx + \frac{1}{3!}\frac{d^3f}{dx^3}\bigg|_{x_0} dx^2 + \cdots$$

How good is the approximation?

Taylor Series

$$f(x_0+dx)=f(x_0)+\frac{df}{dx}\bigg|_{x_0}dx+\frac{1}{2!}\frac{d^2f}{dx^2}\bigg|_{x_0}dx^2+\frac{1}{3!}\frac{d^3f}{dx^3}\bigg|_{x_0}dx^3+\cdots$$

$$\frac{f(x_0+dx)-f(x_0)}{dx}=\frac{df}{dx}\bigg|_{x_0}+\frac{1}{2!}\frac{d^2f}{dx^2}\bigg|_{x_0}dx+\frac{1}{3!}\frac{d^3f}{dx^3}\bigg|_{x_0}dx^2+\cdots$$

$$\frac{f(x_0+dx)-f(x_0)}{dx}=\frac{df}{dx}\bigg|_{x_0}+O(dx)$$

How good is the approximation?

Taylor Series

$$f(x_0+dx)=f(x_0)+\frac{df}{dx}\bigg|_{x_0}dx+\frac{1}{2!}\frac{d^2f}{dx^2}\bigg|_{x_0}dx^2+\frac{1}{3!}\frac{d^3f}{dx^3}\bigg|_{x_0}dx^3+\cdots$$

$$\frac{f(x_0+dx)-f(x_0)}{dx}=\frac{df}{dx}\bigg|_{x_0}+\frac{1}{2!}\frac{d^2f}{dx^2}\bigg|_{x_0}dx+\frac{1}{3!}\frac{d^3f}{dx^3}\bigg|_{x_0}dx^2+\cdots$$

$$\frac{f(x_0+dx)-f(x_0)}{dx}=\frac{df}{dx}\bigg|_{x_0}+O(dx)$$

How good is the approximation?

Taylor Series

$$f(x_0+dx) = f(x_0) + \left.\frac{df}{dx}\right|_{x_0} dx + \frac{1}{2!}\left.\frac{d^2f}{dx^2}\right|_{x_0} dx^2 + \frac{1}{3!}\left.\frac{d^3f}{dx^3}\right|_{x_0} dx^3 + \cdots$$

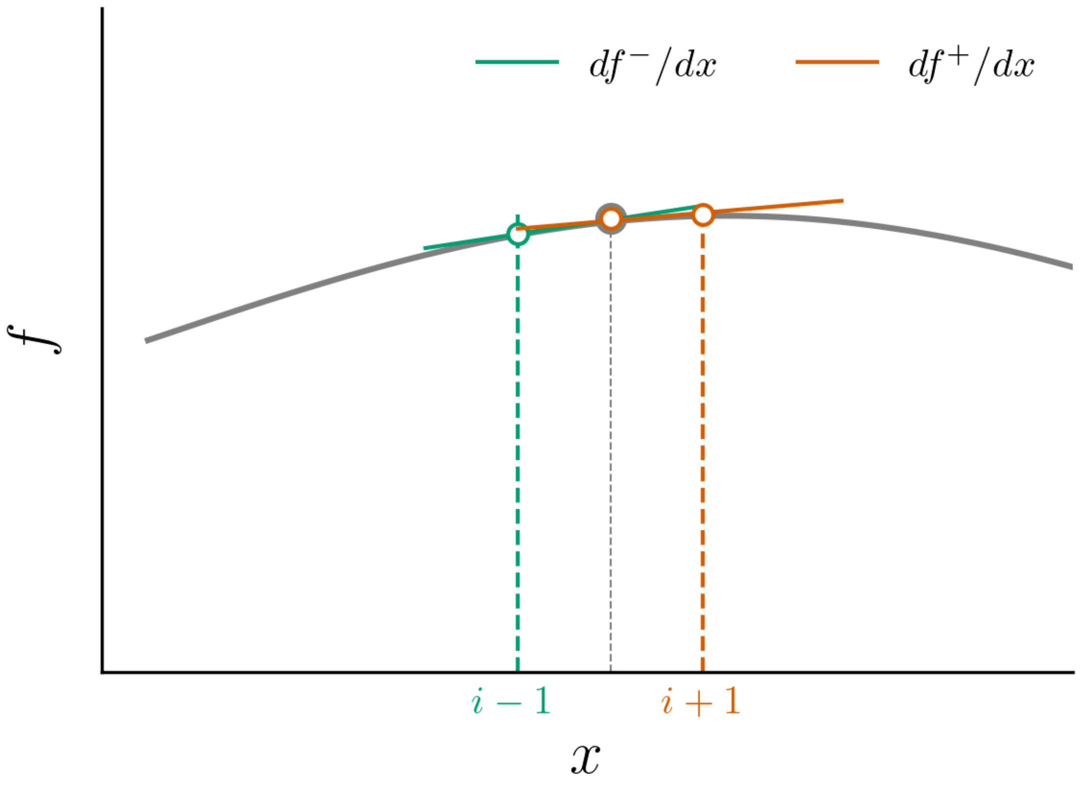
$$\frac{f(x_0+dx) - f(x_0)}{dx} = \left.\frac{df}{dx}\right|_{x_0} + \frac{1}{2!}\left.\frac{d^2f}{dx^2}\right|_{x_0} dx + \frac{1}{3!}\left.\frac{d^3f}{dx^3}\right|_{x_0} dx^2 + \cdots$$

$$\frac{f(x_0+dx) - f(x_0)}{dx} = \left.\frac{df}{dx}\right|_{x_0} + O(dx)$$

1st order accurate

Second derivative

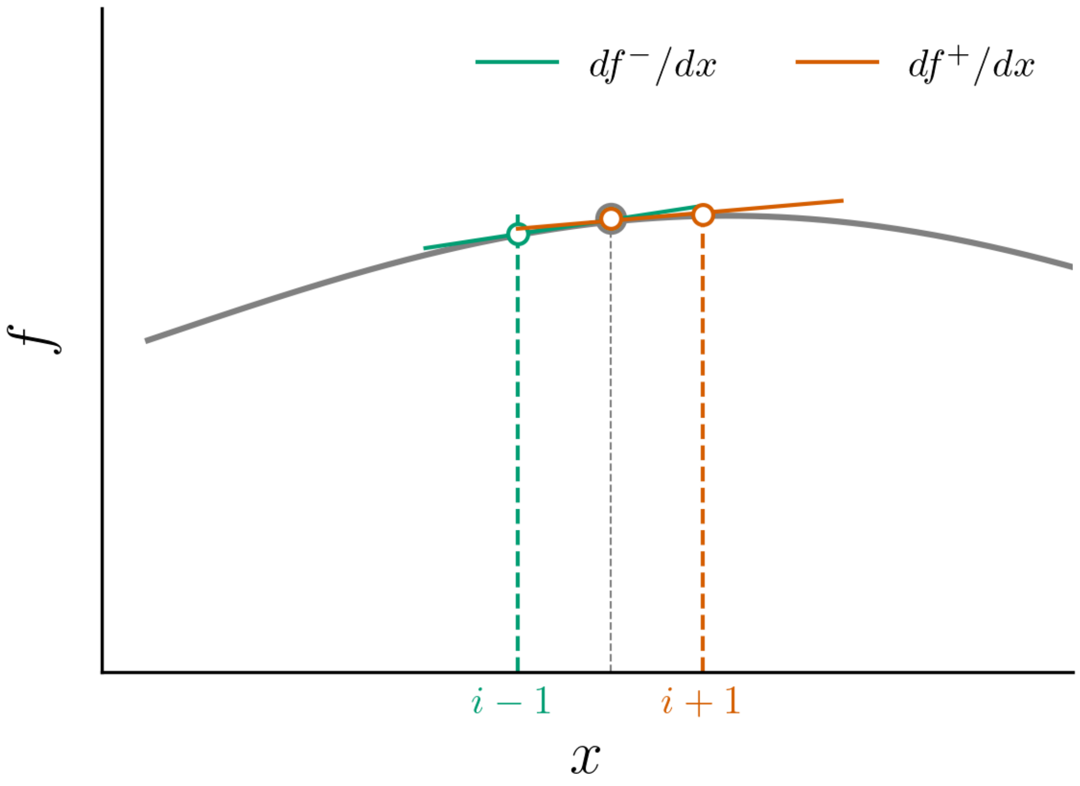
$$\frac{\left.\frac{df}{dx}\right|^{+} - \left.\frac{df}{dx}\right|^{-}}{dx} \approx \frac{d^2 f}{dx^2}$$



Second derivative

$$\frac{\left.\frac{df}{dx}\right|^{+} - \left.\frac{df}{dx}\right|^{-}}{dx} \approx \frac{d^2 f}{dx^2}$$

$$\frac{\frac{f(x+dx)-f(x)}{dx} - \frac{f(x+dx)-f(x)}{dx}}{dx} \approx \frac{d^2 f}{dx^2}$$

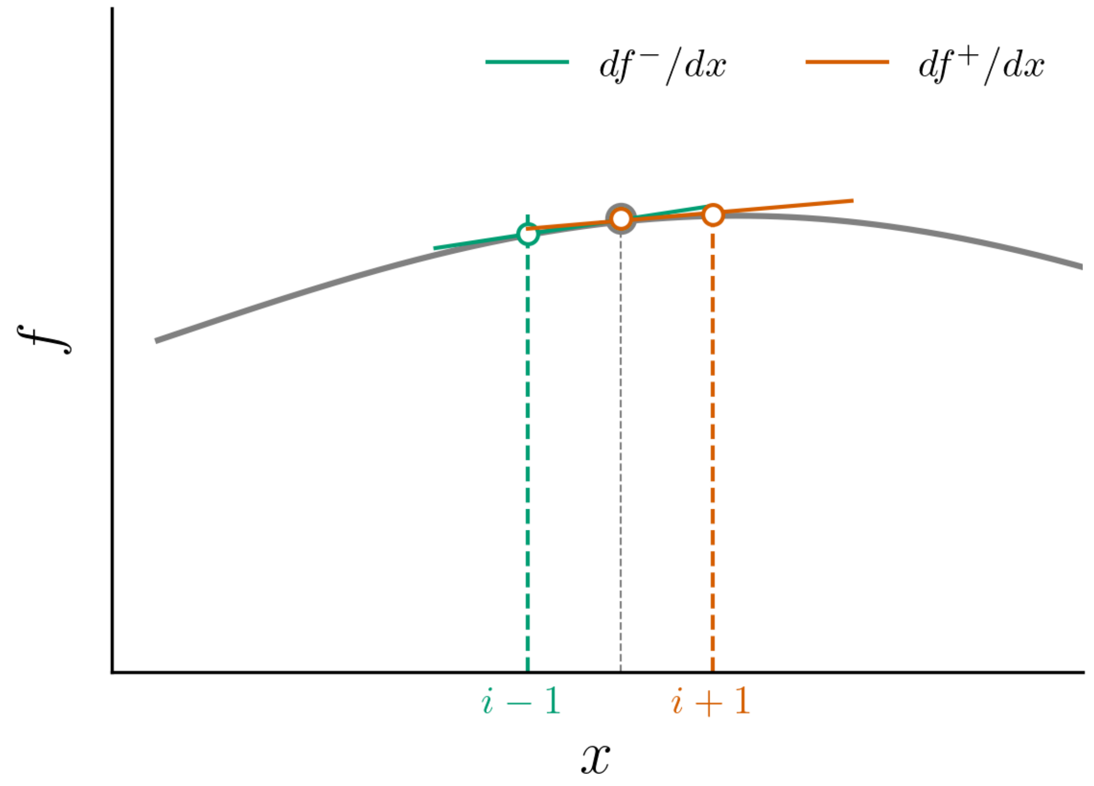


Second derivative

$$\frac{\left.\frac{df}{dx}\right|^{+} - \left.\frac{df}{dx}\right|^{-}}{dx} \approx \frac{d^2 f}{dx^2}$$

$$\frac{\frac{f(x+dx)-f(x)}{dx} - \frac{f(x+dx)-f(x)}{dx}}{dx} \approx \frac{d^2 f}{dx^2}$$

$$\frac{f(x+dx) - 2f(x) + f(x-dx)}{dx^2} \approx \frac{d^2 f}{dx^2}$$

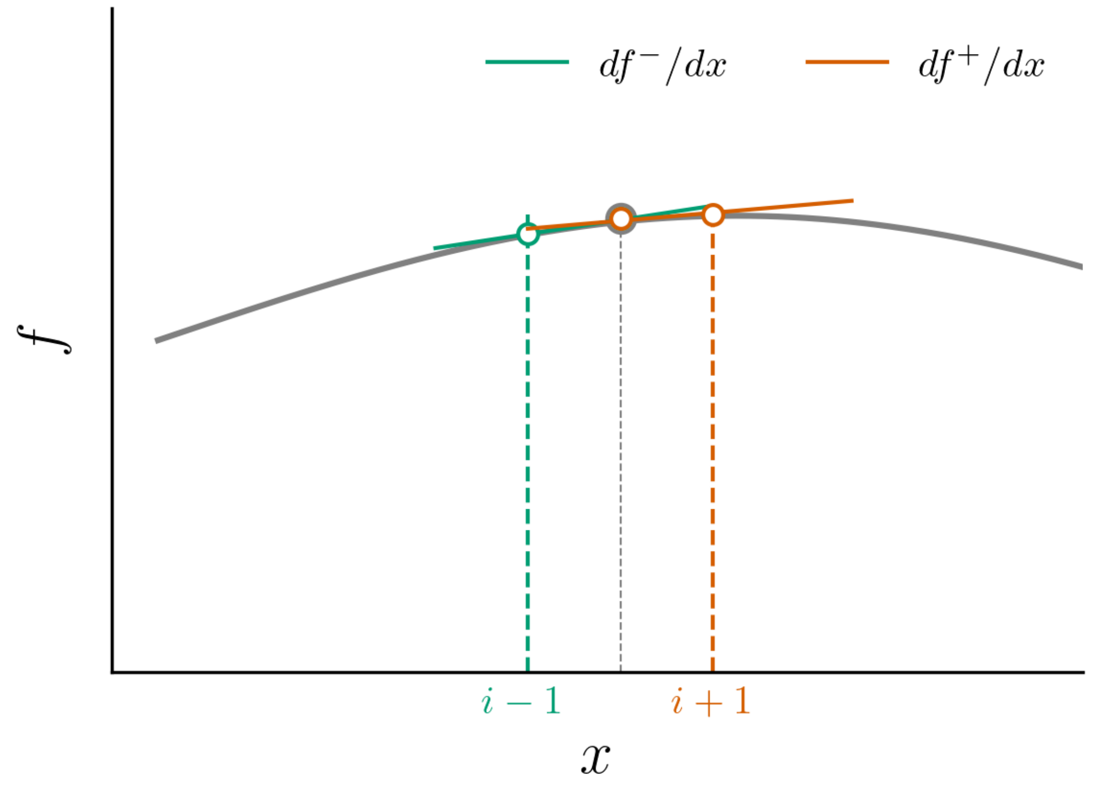


Second derivative

$$\frac{\left.\frac{df}{dx}\right|^{+} - \left.\frac{df}{dx}\right|^{-}}{dx} \approx \frac{d^2 f}{dx^2}$$

$$\frac{\frac{f(x+dx)-f(x)}{dx} - \frac{f(x+dx)-f(x)}{dx}}{dx} \approx \frac{d^2 f}{dx^2}$$

$$\boxed{\frac{f(x+dx) - 2f(x) + f(x-dx)}{dx^2} \approx \frac{d^2 f}{dx^2}}$$



Central Differencing formula for 2nd derivative

Derivative Operators

Taylor Series

$$[f(x+dx)] = \left[f(x) + f'(x)dx + \frac{1}{2!}f''(x)dx^2 + \cdots\right]$$

$$[f(x)] = [f(x)]$$

$$[f(x-dx)] = \left[f(x) - f'(x)dx + \frac{1}{2!}f''(x)dx^2 + \cdots\right]$$

Derivative Operators

MODELAIR

Taylor Series

$$a[f(x+dx)] = a\left[f(x) + f'(x)dx + \frac{1}{2!}f''(x)dx^2 + \cdots\right]$$

$$b[f(x)] = b[f(x)]$$

$$c[f(x-dx)] = c\left[f(x) - f'(x)dx + \frac{1}{2!}f''(x)dx^2 + \cdots\right]$$

Derivative Operators

Taylor Series

$$a[f(x+dx)] = a\left[f(x) + f'(x)dx + \frac{1}{2!}f''(x)dx^2 + \cdots\right]$$

$$b[f(x)] = b[f(x)]$$

$$c[f(x-dx)] = c\left[f(x) - f'(x)dx + \frac{1}{2!}f''(x)dx^2 + \cdots\right]$$

$$af(x+dx) + bf(x) + cf(x-dx) \approx (a+b+c)f(x) + (a-c)f'(x)dx + \frac{(a+c)}{2}f''(x)dx^2$$

Derivative Operators

$$af(x+dx)+bf(x)+cf(x-dx)\approx(a+b+c)f(x)+(a-c)f'(x)dx+\frac{(a+c)}{2}f''(x)dx^2$$

Derivative Operators

$$af(x+dx) + bf(x) + cf(x-dx) \approx (a+b+c)f(x) + (a-c)f'(x)dx + \frac{(a+c)}{2}f''(x)dx^2$$

1^{st} Derivative

$$(a+b+c) = 0$$

$$(a \quad -c) = 1/dx$$

$$(a \quad +c) = 0$$

Derivative Operators

$$af(x+dx) + bf(x) + cf(x-dx) \approx (a+b+c)f(x) + (a-c)f'(x)dx + \frac{(a+c)}{2}f''(x)dx^2$$

1st Derivative

$$(a+b+c) = 0$$
$$(a \qquad - c) = 1/dx$$
$$(a \qquad + c) = 0$$

$$\underset{A}{\begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & -1 \\ 1 & 0 & 1 \end{bmatrix}} \underset{w}{\begin{pmatrix} a \\ b \\ c \end{pmatrix}} = \underset{s}{\begin{pmatrix} 0 \\ 1/dx \\ 0 \end{pmatrix}}$$

Derivative Operators

$$af(x+dx) + bf(x) + cf(x-dx) \approx (a+b+c)f(x) + (a-c)f'(x)dx + \frac{(a+c)}{2}f''(x)dx^2$$

1st Derivative

$$(a+b+c) = 0$$
$$(a \quad\quad - c) = 1/dx$$
$$(a \quad\quad + c) = 0$$

$$\underbrace{\begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & -1 \\ 1 & 0 & 1 \end{bmatrix}}_{A} \underbrace{\begin{pmatrix} a \\ b \\ c \end{pmatrix}}_{w} = \underbrace{\begin{pmatrix} 0 \\ 1/dx \\ 0 \end{pmatrix}}_{s} \xrightarrow{\text{Inversion}}$$

$$a = \frac{1}{2dx}$$
$$b = 0$$
$$c = \frac{-1}{2dx}$$

Derivative Operators

$$af(x+dx)+bf(x)+cf(x-dx)\approx(a+b+c)f(x)+(a-c)f'(x)dx+\frac{(a+c)}{2}f''(x)dx^2$$

1st Derivative

$$(a+b+c)=0$$
$$(a\quad -c)=1/dx$$
$$(a\quad +c)=0$$

$$\begin{array}{ccccc} A & & w & & s \\ \begin{bmatrix}1 & 1 & 1\\ 1 & 0 & -1\\ 1 & 0 & 1\end{bmatrix} & & \begin{pmatrix}a\\ b\\ c\end{pmatrix} & = & \begin{pmatrix}0\\ 1/dx\\ 0\end{pmatrix}\end{array}\xrightarrow{\text{Inversion}}\quad a=\frac{1}{2dx},\quad b=0,\quad c=\frac{-1}{2dx}$$

2nd Derivative

$$(a+b+c)=0$$
$$(a\quad -c)=0$$
$$(a\quad +c)=2!/dx^2$$

$$\begin{array}{ccccc} A & & w & & s \\ \begin{bmatrix}1 & 1 & 1\\ 1 & 0 & -1\\ 1 & 0 & 1\end{bmatrix} & & \begin{pmatrix}a\\ b\\ c\end{pmatrix} & = & \begin{pmatrix}0\\ 0\\ 2/dx^2\end{pmatrix}\end{array}\xrightarrow{\text{Inversion}}\quad a=\frac{1}{dx^2},\quad b=\frac{-2}{dx^2},\quad c=\frac{1}{dx^2}$$

Finite Difference in action

1D Linear Convection

$$\frac{du}{dt} + c\frac{du}{dx} = 0$$

Finite Difference in action

1D Linear Convection $$\frac{du}{dt} + c\frac{du}{dx} = 0$$

Initial Condition: $$u(x, 0) = u_0(x)$$

Exact Solution: $$u(x, t) = u_0(x - ct)$$

Finite Difference in action

1D Linear Convection
$$\frac{du}{dt} + c\frac{du}{dx} = 0$$

With an Initial Condition (given as a wave), the equation represents propagation of a wave with a speed *c*, *without change in its shape*!

Initial Condition:
$$u(x,0) = u_0(x)$$

Exact Solution:
$$u(x,t) = u_0(x - ct)$$

Finite Difference in action

1D Linear Convection

$$\frac{du}{dt} + c\frac{du}{dx} = 0$$

With an Initial Condition (given as a wave), the equation represents propagation of a wave with a speed c, *without change in its shape*!

Initial Condition:

$$u(x,0) = u_0(x)$$

Exact Solution:

$$u(x,t) = u_0(x - ct)$$

Finite Difference in action

1D Linear Convection

$$\frac{du}{dt} + c\frac{du}{dx} = 0$$

Finite Difference in action

1D Linear Convection

$$\frac{du}{dt} + c\frac{du}{dx} = 0$$

Backward Difference

$$\frac{du}{dx} \approx \frac{u(x) - u(x - \Delta x)}{\Delta x}$$

Finite Difference in action

1D Linear Convection

$$\frac{du}{dt} + c\frac{du}{dx} = 0$$

Backward Difference

$$\frac{du}{dx} \approx \frac{u(x) - u(x - \Delta x)}{\Delta x}$$

Forward Euler

$$\frac{du}{dt} \approx \frac{u^{n+1}(x) - u^{n}(x)}{\Delta t}$$

Finite Difference in action

1D Linear Convection

$$\frac{du}{dt} + c\frac{du}{dx} = 0$$

Backward Difference

$$\frac{du}{dx} \approx \frac{u(x) - u(x - \Delta x)}{\Delta x}$$

Forward Euler

$$\frac{du}{dt} \approx \frac{u^{n+1}(x) - u^{n}(x)}{\Delta t}$$

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} + c\frac{u_i^n - u_{i-1}^n}{\Delta x} = 0$$

Finite Difference in action

1D Linear Convection $$\frac{du}{dt} + c\frac{du}{dx} = 0$$

Backward Difference $$\frac{du}{dx} \approx \frac{u(x) - u(x - \Delta x)}{\Delta x}$$

Forward Euler $$\frac{du}{dt} \approx \frac{u^{n+1}(x) - u^{n}(x)}{\Delta t}$$

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} + c\frac{u_i^n - u_{i-1}^n}{\Delta x} = 0$$

$$\boxed{u_i^{n+1} = u_i^n - \frac{c\Delta t}{\Delta \mathrm{x}}(u_i^n - u_{i-1}^n)}$$

Finite Difference in action

1D Linear Convection
$$\frac{du}{dt} + c\frac{du}{dx} = 0$$

Backward Difference
$$\frac{du}{dx} \approx \frac{u(x) - u(x - \Delta x)}{\Delta x}$$

Forward Euler
$$\frac{du}{dt} \approx \frac{u^{n+1}(x) - u^{n}(x)}{\Delta t}$$

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} + c\frac{u_i^n - u_{i-1}^n}{\Delta x} = 0$$

$$\boxed{u_i^{n+1} = u_i^n - \frac{c\Delta t}{\Delta \mathrm{x}}(u_i^n - u_{i-1}^n)}$$

Let's write our 1st CFD code !

Barba et al., (2018). CFD Python: the 12 steps to Navier-Stokes equations. Journal of Open-Source Education, 1(9), 21. https://doi.org/10. 21105/jose.00021

1st Code: 1D Linear Convection

```
import numpy as np
import matplotlib.pyplot as plt
import time, sys
```

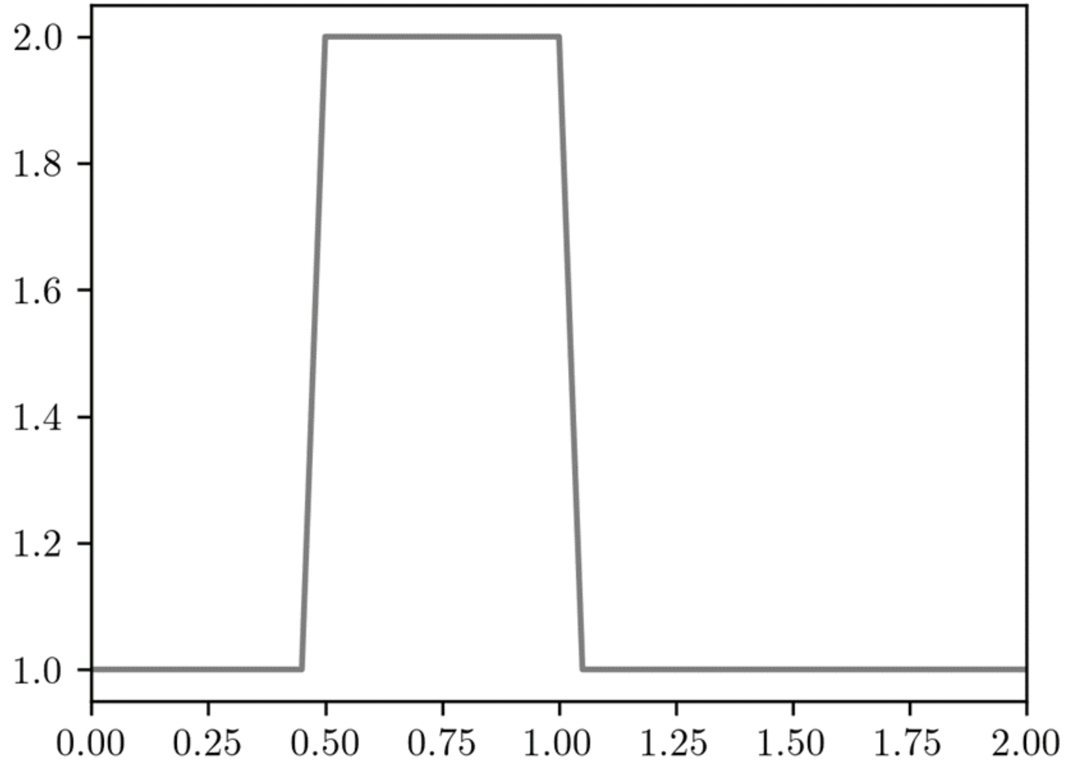
1st Code: 1D Linear Convection

```
import numpy as np
import matplotlib.pyplot as plt
import time, sys

nx = 41
dx = 2/(nx-1)
nt = 25
dt = 0.025
c = 1
```

1^{st} Code: 1D Linear Convection

```
import numpy as np
import matplotlib.pyplot as plt
import time, sys

nx = 41
dx = 2/(nx-1)
nt = 25
dt = 0.025
c = 1
```

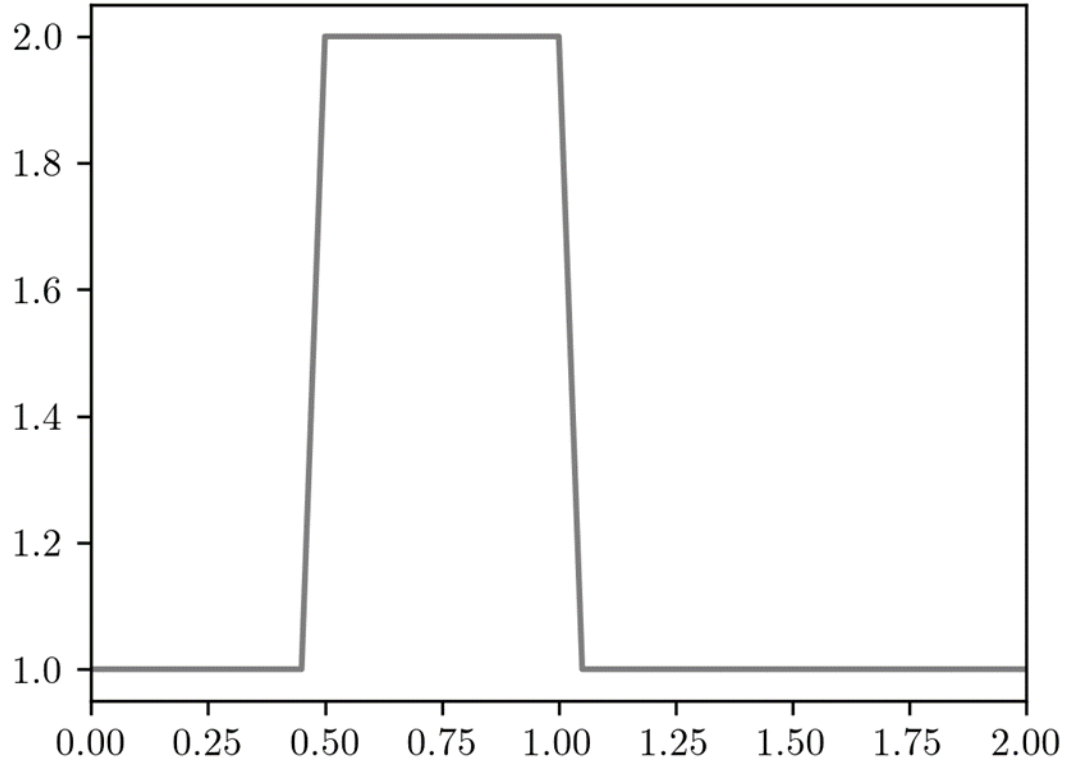
The initial velocity u_0 is given as 2 in the interval $0.5 \leq x \leq 1$ and 1 elsewhere in $(0,2) \Longrightarrow$ Hat function



1st Code: 1D Linear Convection

```
import numpy as np
import matplotlib.pyplot as plt
import time, sys

nx = 41
dx = 2/(nx-1)
nt = 25
dt = 0.025
c = 1

u = np.ones(nx)
u[int(0.5/dx):int(1/dx+1)] = 2
print(u)
plt.plot(np.linspace(0,2,nx), u)
```

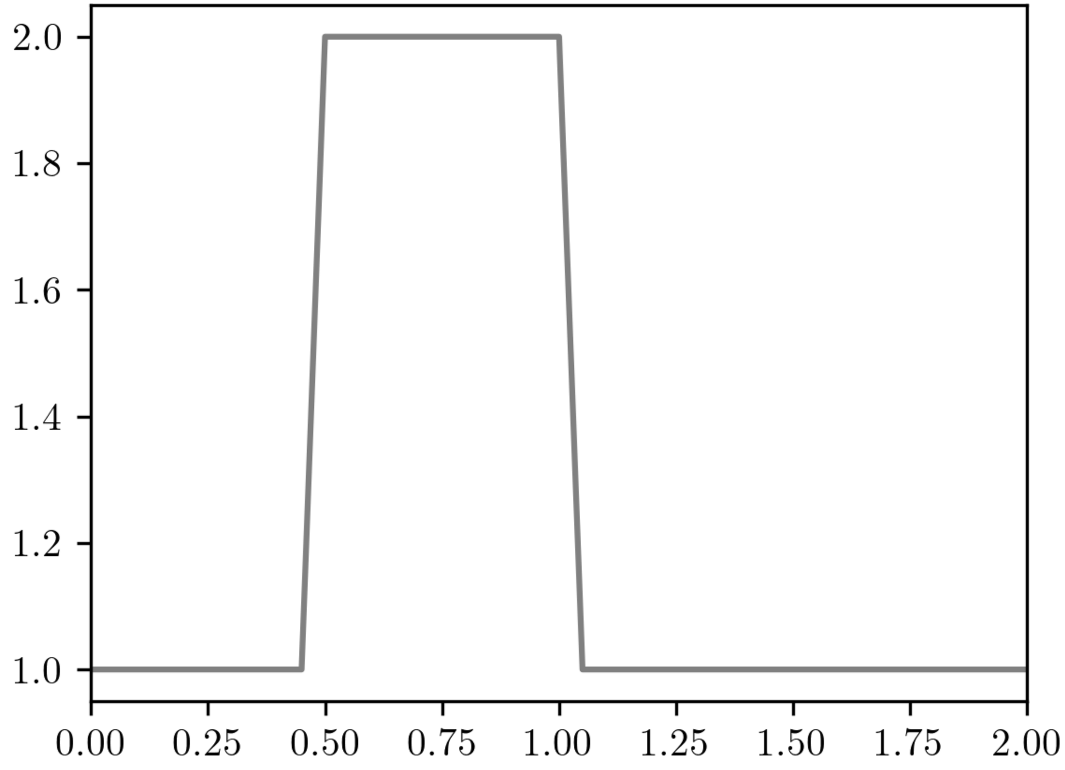
The initial velocity u_0 is given as 2 in the interval $0.5 \leq x \leq 1$ and 1 elsewhere in $(0,2) \Longrightarrow$ Hat function



1st Code: 1D Linear Convection

```
import numpy as np
import matplotlib.pyplot as plt
import time, sys

nx = 41
dx = 2/(nx-1)
nt = 25
dt = 0.025
c = 1

u = np.ones(nx)
u[int(0.5/dx):int(1/dx+1)] = 2
print(u)
plt.plot(np.linspace(0,2,nx), u)

un = np.ones(nx)
for n in range(nt):
    un=u.copy()
    for i in range(1,nx):
        u[i] = un[i] – c * dt/dx * (un[i]-un[i-1])
plt.plot(np.linspace(0,2,nx), u)
```

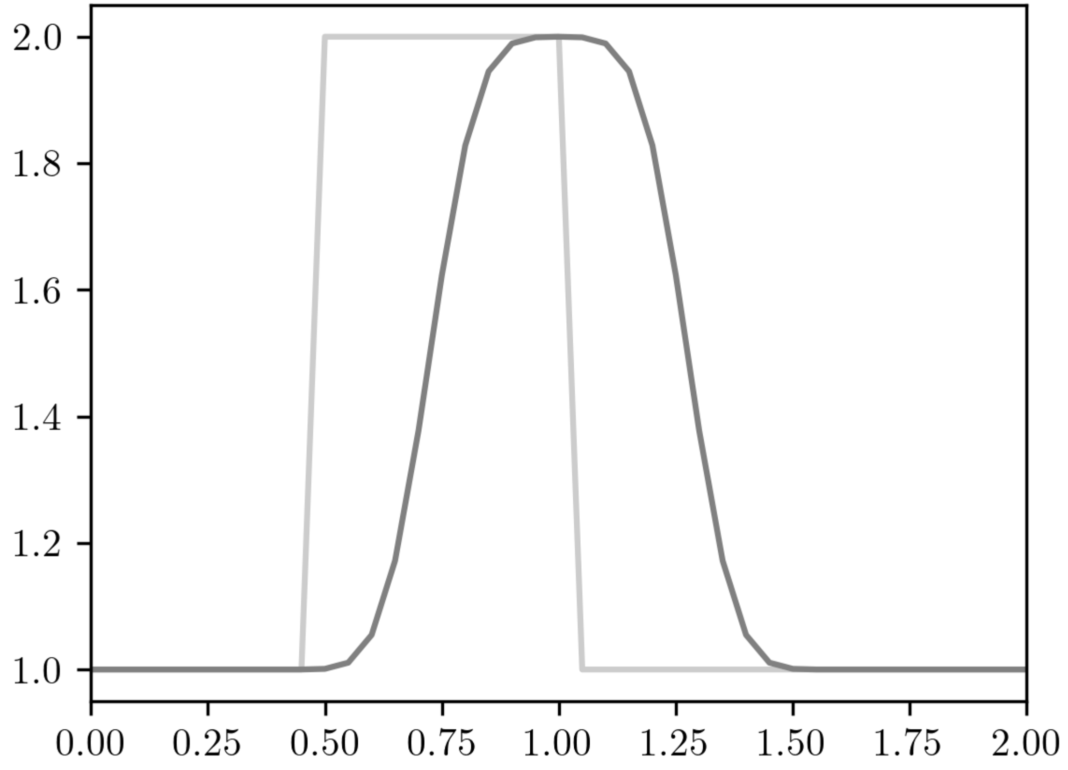
The initial velocity u_0 is given as 2 in the interval $0.5 \leq x \leq 1$ and 1 elsewhere in $(0,2)$ $\Longrightarrow$ Hat function



1st Code: 1D Linear Convection

```
import numpy as np
import matplotlib.pyplot as plt
import time, sys

nx = 41
dx = 2/(nx-1)
nt = 25
dt = 0.025
c = 1

u = np.ones(nx)
u[int(0.5/dx):int(1/dx+1)] = 2
print(u)
plt.plot(np.linspace(0,2,nx), u)

un = np.ones(nx)
for n in range(nt):
    un=u.copy()
    for i in range(1,nx):
        u[i] = un[i] – c * dt/dx * (un[i]-un[i-1])
plt.plot(np.linspace(0,2,nx), u)
```

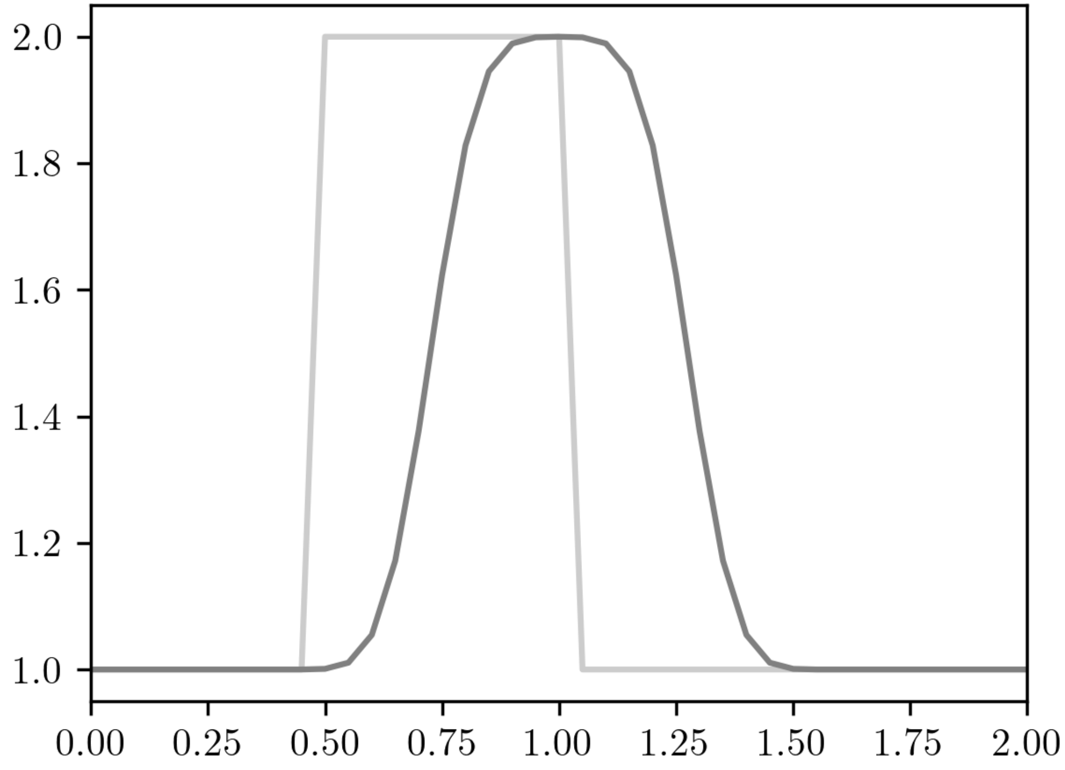
The initial velocity u_0 is given as 2 in the interval $0.5 \leq x \leq 1$ and 1 elsewhere in $(0,2) \Longrightarrow$ Hat function



1^{st} Code: 1D Linear Convection

```python
import numpy as np
import matplotlib.pyplot as plt
import time, sys

nx = 41
dx = 2/(nx-1)
nt = 25
dt = 0.025
c = 1

u = np.ones(nx)
u[int(0.5/dx):int(1/dx+1)] = 2
print(u)
plt.plot(np.linspace(0,2,nx), u)

un = np.ones(nx)
for n in range(nt):
    un=u.copy()
    for i in range(1,nx):
        u[i] = un[i] – c * dt/dx * (un[i]-un[i-1])
plt.plot(np.linspace(0,2,nx), u)
```

The initial velocity u_0 is given as 2 in the interval $0.5 \leq x \leq 1$ and 1 elsewhere in $(0,2) \Longrightarrow$ Hat function



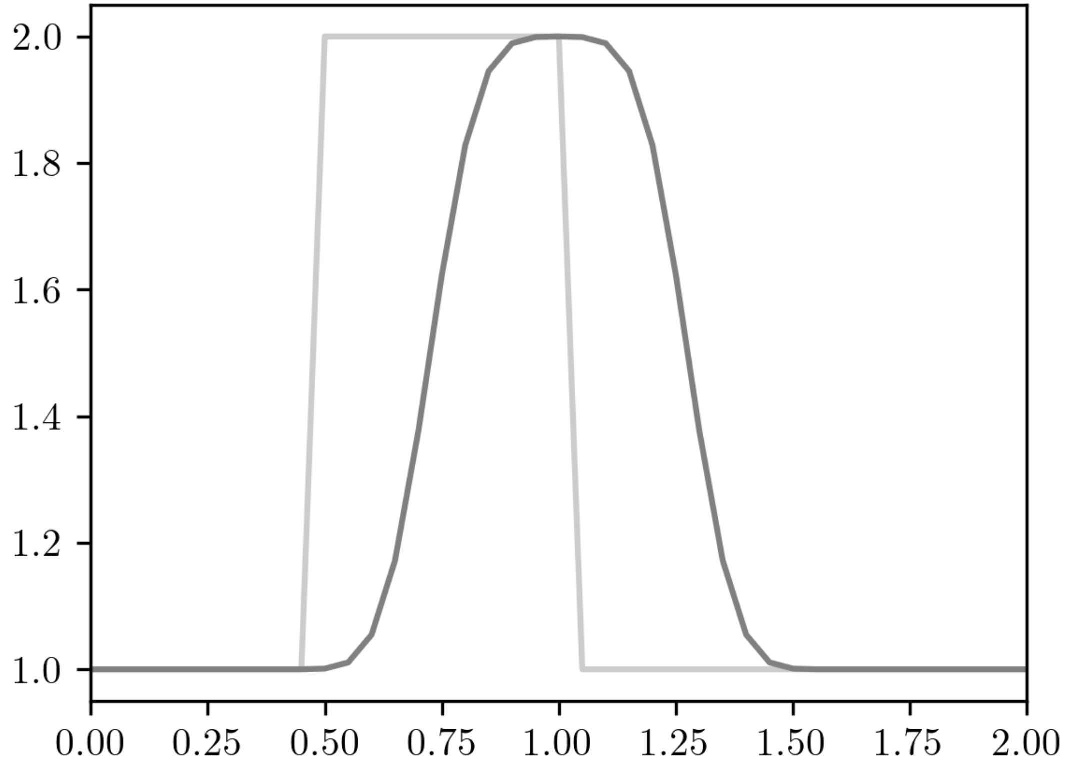
Ok, so our hat function moved to the right, but it's no longer a hat. *What's going on?*

1st Code: 1D Linear Convection

```python
import numpy as np
import matplotlib.pyplot as plt
import time, sys

nx = 41
dx = 2/(nx-1)
nt = 25
dt = 0.025
c = 1

u = np.ones(nx)
u[int(0.5/dx):int(1/dx+1)] = 2
print(u)
plt.plot(np.linspace(0,2,nx), u)

un = np.ones(nx)
for n in range(nt):
    un=u.copy()
    for i in range(1,nx):
        u[i] = un[i] – c * dt/dx * (un[i]-un[i-1])
plt.plot(np.linspace(0,2,nx), u)
```

Let's try with central differencing !

The initial velocity u_0 is given as 2 in the interval $0.5 \leq x \leq 1$ and 1 elsewhere in $(0,2) \Longrightarrow$ Hat function



Ok, so our hat function moved to the right, but it's no longer a hat. *What's going on?*

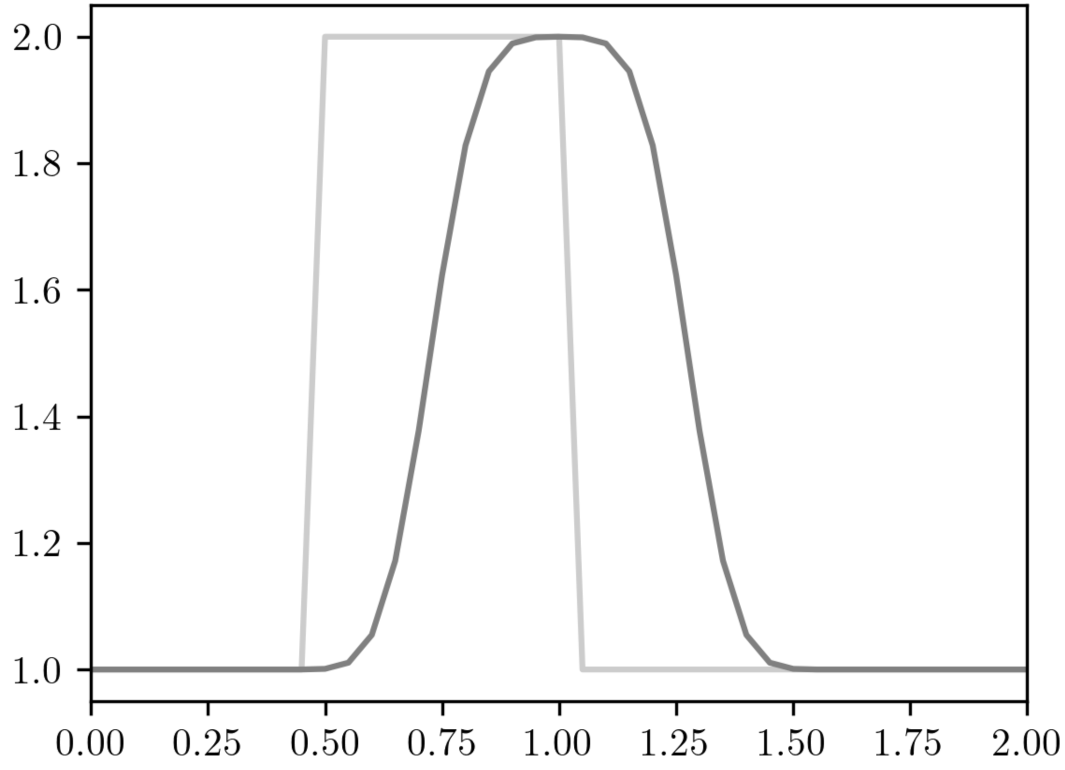
1st Code: 1D Linear Convection

```
import numpy as np
import matplotlib.pyplot as plt
import time, sys

nx = 41
dx = 2/(nx-1)
nt = 25
dt = 0.025
c = 1

u = np.ones(nx)
u[int(0.5/dx):int(1/dx+1)] = 2
print(u)
plt.plot(np.linspace(0,2,nx), u)

un = np.ones(nx)
for n in range(nt):
    un=u.copy()
    for i in range(1,nx):
        u[i] = un[i] – c * dt/dx * (un[i]-un[i-1])
plt.plot(np.linspace(0,2,nx), u)
```

Change these !

Let's try with central differencing !

The initial velocity u_0 is given as 2 in the interval $0.5 \leq x \leq 1$ and 1 elsewhere in $(0,2) \Longrightarrow$ Hat function



Ok, so our hat function moved to the right, but it's no longer a hat. *What's going on?*

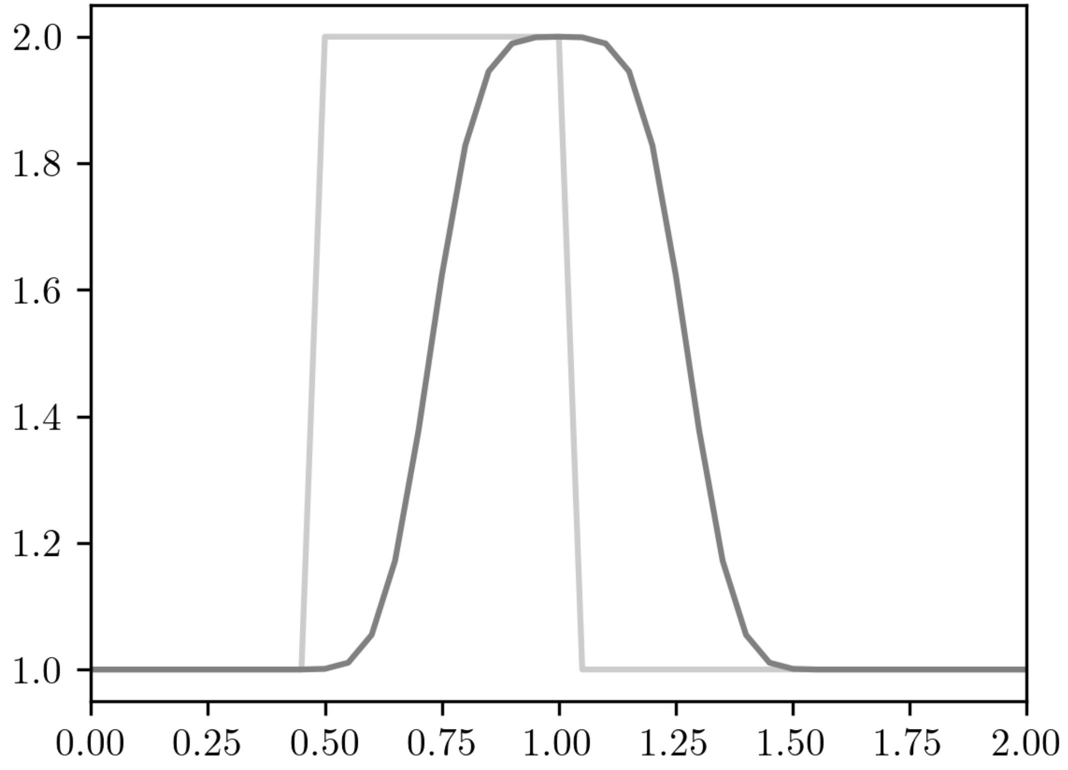
1st Code: 1D Linear Convection

```
import numpy as np
import matplotlib.pyplot as plt
import time, sys

nx = 41
dx = 2/(nx-1)
nt = 25
dt = 0.025
c = 1

u = np.ones(nx)
u[int(0.5/dx):int(1/dx+1)] = 2
print(u)
plt.plot(np.linspace(0,2,nx), u)

un = np.ones(nx)
for n in range(nt):
    un=u.copy()
    for i in range(1,nx-1):
        u[i] = un[i] – c * dt/dx * (un[i+1]-un[i-1])
plt.plot(np.linspace(0,2,nx), u)
```

Change these !

Let's try with central differencing !

The initial velocity u_0 is given as 2 in the interval $0.5 \leq x \leq 1$ and 1 elsewhere in $(0,2)$ $\Longrightarrow$ Hat function



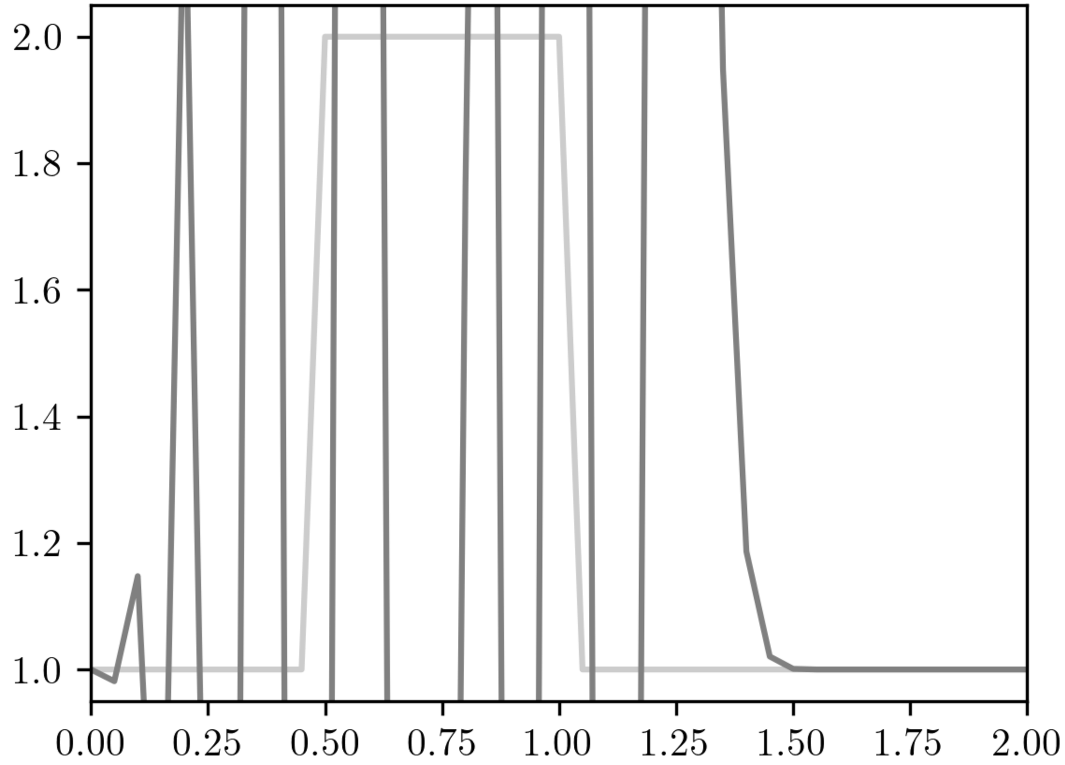
Ok, so our hat function moved to the right, but it's no longer a hat. What's going on?

1^{st} Code: 1D Linear Convection

```
import numpy as np
import matplotlib.pyplot as plt
import time, sys

nx = 41
dx = 2/(nx-1)
nt = 25
dt = 0.025
c = 1

u = np.ones(nx)
u[int(0.5/dx):int(1/dx+1)] = 2
print(u)
plt.plot(np.linspace(0,2,nx), u)

un = np.ones(nx)
for n in range(nt):
    un=u.copy()
    for i in range(1,nx-1):
        u[i] = un[i] – c * dt/dx * (un[i+1]-un[i-1])
plt.plot(np.linspace(0,2,nx), u)
```

Let's try with central differencing !

The initial velocity u_0 is given as 2 in the interval $0.5 \leq x \leq 1$ and 1 elsewhere in $(0,2)$ $\Longrightarrow$ Hat function



Ok, so our hat function moved to the right, but it's no longer a hat. *What's going on?*

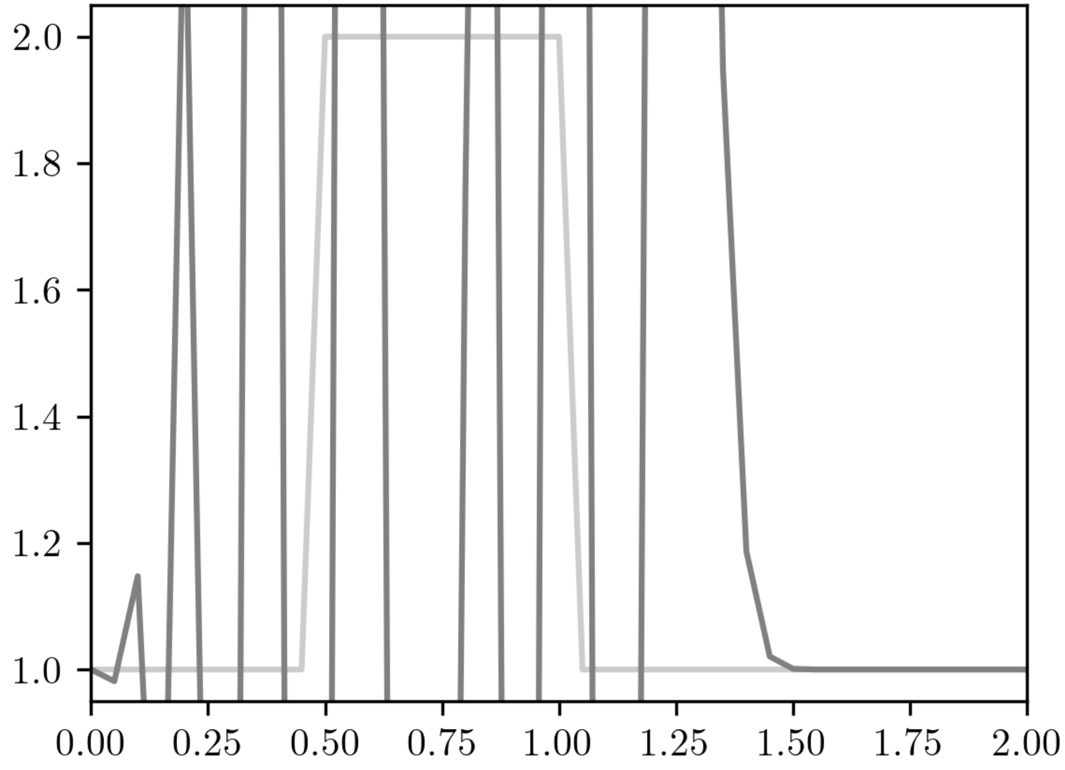
1^{st} Code: 1D Linear Convection

```
import numpy as np
import matplotlib.pyplot as plt
import time, sys

nx = 41
dx = 2/(nx-1)
nt = 25
dt = 0.025
c = 1

u = np.ones(nx)
u[int(0.5/dx):int(1/dx+1)] = 2
print(u)
plt.plot(np.linspace(0,2,nx), u)

un = np.ones(nx)
for n in range(nt):
    un=u.copy()
    for i in range(1,nx-1):
        u[i] = un[i] – c * dt/dx * (un[i+1]-un[i-1])
plt.plot(np.linspace(0,2,nx), u)
```

What's that !

Let's try with central differencing !

The initial velocity u_0 is given as 2 in the interval $0.5 \leq x \leq 1$ and 1 elsewhere in $(0,2) \Longrightarrow$ Hat function



Ok, so our hat function moved to the right, but it's no longer a hat. What's going on?

1D Linear Convection with Central Differencing in Space

$$\frac{du}{dt} + c\frac{du}{dx} = 0 \longrightarrow \frac{u_i^{n+1} - u_i^n}{\Delta t} + \frac{c}{2\Delta \mathrm{x}}(u_{i+1}^n - u_{i-1}^n) = 0$$

1D Linear Convection with Central Differencing in Space

$$\frac{du}{dt} + c\frac{du}{dx} = 0 \longrightarrow \frac{u_i^{n+1} - u_i^n}{\Delta t} + \frac{c}{2\Delta \mathrm{x}}(u_{i+1}^n - u_{i-1}^n) = 0$$

$$u_i^{n+1} = u_i^n + \Delta t \frac{\partial u}{\partial t}\bigg|_i^n + \frac{\Delta t^2}{2!}\frac{\partial^2 u}{\partial t^2}\bigg|_i^n + \frac{\Delta t^3}{3!}\frac{\partial^3 u}{\partial t^3}\bigg|_i^n + \cdots$$

1D Linear Convection with Central Differencing in Space

$$\frac{du}{dt} + c\frac{du}{dx} = 0 \longrightarrow \frac{u_i^{n+1} - u_i^n}{\Delta t} + \frac{c}{2\Delta \mathrm{x}}(u_{i+1}^n - u_{i-1}^n) = 0$$

$$u_i^{n+1} = u_i^n + \Delta t \frac{\partial u}{\partial t}\bigg|_i^n + \frac{\Delta t^2}{2!}\frac{\partial^2 u}{\partial t^2}\bigg|_i^n + \frac{\Delta t^3}{3!}\frac{\partial^3 u}{\partial t^3}\bigg|_i^n + \cdots$$

$$u_{i+1}^n = u_i^n + \Delta x \frac{\partial u}{\partial x}\bigg|_i^n + \frac{\Delta x^2}{2!}\frac{\partial^2 u}{\partial x^2}\bigg|_i^n + \frac{\Delta x^3}{3!}\frac{\partial^3 u}{\partial x^3}\bigg|_i^n + \cdots$$

1D Linear Convection with Central Differencing in Space

$$\frac{du}{dt} + c\frac{du}{dx} = 0 \longrightarrow \frac{u_i^{n+1} - u_i^n}{\Delta t} + \frac{c}{2\Delta \mathrm{x}}(u_{i+1}^n - u_{i-1}^n) = 0$$

$$u_i^{n+1} = u_i^n + \Delta t\frac{\partial u}{\partial t}\bigg|_i^n + \frac{\Delta t^2}{2!}\frac{\partial^2 u}{\partial t^2}\bigg|_i^n + \frac{\Delta t^3}{3!}\frac{\partial^3 u}{\partial t^3}\bigg|_i^n + \cdots$$

$$u_{i+1}^n = u_i^n + \Delta x\frac{\partial u}{\partial x}\bigg|_i^n + \frac{\Delta x^2}{2!}\frac{\partial^2 u}{\partial x^2}\bigg|_i^n + \frac{\Delta x^3}{3!}\frac{\partial^3 u}{\partial x^3}\bigg|_i^n + \cdots$$

$$u_{i-1}^n = u_i^n - \Delta x\frac{\partial u}{\partial x}\bigg|_i^n + \frac{\Delta x^2}{2!}\frac{\partial^2 u}{\partial x^2}\bigg|_i^n - \frac{\Delta x^3}{3!}\frac{\partial^3 u}{\partial x^3}\bigg|_i^n + \cdots$$

1D Linear Convection with Central Differencing in Space

$$\frac{du}{dt} + c\frac{du}{dx} = 0 \longrightarrow \frac{u_i^{n+1} - u_i^n}{\Delta t} + \frac{c}{2\Delta x}(u_{i+1}^n - u_{i-1}^n) = 0$$

$$u_i^{n+1} = u_i^n + \Delta t \left.\frac{\partial u}{\partial t}\right|_i^n + \frac{\Delta t^2}{2!}\left.\frac{\partial^2 u}{\partial t^2}\right|_i^n + \frac{\Delta t^3}{3!}\left.\frac{\partial^3 u}{\partial t^3}\right|_i^n + \cdots$$

$$u_{i+1}^n = u_i^n + \Delta x \left.\frac{\partial u}{\partial x}\right|_i^n + \frac{\Delta x^2}{2!}\left.\frac{\partial^2 u}{\partial x^2}\right|_i^n + \frac{\Delta x^3}{3!}\left.\frac{\partial^3 u}{\partial x^3}\right|_i^n + \cdots$$

$$u_{i-1}^n = u_i^n - \Delta x \left.\frac{\partial u}{\partial x}\right|_i^n + \frac{\Delta x^2}{2!}\left.\frac{\partial^2 u}{\partial x^2}\right|_i^n - \frac{\Delta x^3}{3!}\left.\frac{\partial^3 u}{\partial x^3}\right|_i^n + \cdots$$

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} + \frac{c}{2\Delta x}(u_{i+1}^n - u_{i-1}^n) - \left(\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x}\right)_i^n = \frac{\Delta t}{2!}\left.\frac{\partial^2 u}{\partial t^2}\right|_i^n + \frac{c\Delta x^3}{3!}\left.\frac{\partial^3 u}{\partial t^3}\right|_i^n + O(\Delta t^2, \Delta x^4)$$

1D Linear Convection with Central Differencing in Space

$$\frac{du}{dt} + c\frac{du}{dx} = 0 \longrightarrow \frac{u_i^{n+1} - u_i^n}{\Delta t} + \frac{c}{2\Delta x}(u_{i+1}^n - u_{i-1}^n) = 0$$

$$u_i^{n+1} = u_i^n + \Delta t \frac{\partial u}{\partial t}\bigg|_i^n + \frac{\Delta t^2}{2!}\frac{\partial^2 u}{\partial t^2}\bigg|_i^n + \frac{\Delta t^3}{3!}\frac{\partial^3 u}{\partial t^3}\bigg|_i^n + \cdots$$

$$u_{i+1}^n = u_i^n + \Delta x \frac{\partial u}{\partial x}\bigg|_i^n + \frac{\Delta x^2}{2!}\frac{\partial^2 u}{\partial x^2}\bigg|_i^n + \frac{\Delta x^3}{3!}\frac{\partial^3 u}{\partial x^3}\bigg|_i^n + \cdots$$

$$u_{i-1}^n = u_i^n - \Delta x \frac{\partial u}{\partial x}\bigg|_i^n + \frac{\Delta x^2}{2!}\frac{\partial^2 u}{\partial x^2}\bigg|_i^n - \frac{\Delta x^3}{3!}\frac{\partial^3 u}{\partial x^3}\bigg|_i^n + \cdots$$

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} + \frac{c}{2\Delta x}(u_{i+1}^n - u_{i-1}^n) - \left(\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x}\right)_i^n = \frac{\Delta t}{2!}\frac{\partial^2 u}{\partial t^2}\bigg|_i^n + \frac{c\Delta x^3}{3!}\frac{\partial^3 u}{\partial t^3}\bigg|_i^n + O(\Delta t^2, \Delta x^4)$$

Truncation Error (ϵ_t)

1D Linear Convection with Central Differencing in Space

$$\frac{du}{dt} + c\frac{du}{dx} = 0 \longrightarrow \frac{u_i^{n+1} - u_i^n}{\Delta t} + \frac{c}{2\Delta x}(u_{i+1}^n - u_{i-1}^n) = 0$$

$$u_i^{n+1} = u_i^n + \Delta t \left.\frac{\partial u}{\partial t}\right|_i^n + \frac{\Delta t^2}{2!}\left.\frac{\partial^2 u}{\partial t^2}\right|_i^n + \frac{\Delta t^3}{3!}\left.\frac{\partial^3 u}{\partial t^3}\right|_i^n + \cdots$$

$$u_{i+1}^n = u_i^n + \Delta x \left.\frac{\partial u}{\partial x}\right|_i^n + \frac{\Delta x^2}{2!}\left.\frac{\partial^2 u}{\partial x^2}\right|_i^n + \frac{\Delta x^3}{3!}\left.\frac{\partial^3 u}{\partial x^3}\right|_i^n + \cdots$$

$$u_{i-1}^n = u_i^n - \Delta x \left.\frac{\partial u}{\partial x}\right|_i^n + \frac{\Delta x^2}{2!}\left.\frac{\partial^2 u}{\partial x^2}\right|_i^n - \frac{\Delta x^3}{3!}\left.\frac{\partial^3 u}{\partial x^3}\right|_i^n + \cdots$$

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} + \frac{c}{2\Delta x}(u_{i+1}^n - u_{i-1}^n) - \left(\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x}\right)_i^n = \frac{\Delta t}{2!}\left.\frac{\partial^2 u}{\partial t^2}\right|_i^n + \frac{c\Delta x^3}{3!}\left.\frac{\partial^3 u}{\partial t^3}\right|_i^n + O(\Delta t^2, \Delta x^4)$$

Truncation Error (ϵ_t)

As Δt & $\Delta x \longrightarrow 0$: $\epsilon_T \longrightarrow 0 \Rightarrow$ Numerical Scheme is Consistent

1D Linear Convection with Central Differencing in Space

Consider the exact solution of the discretized equation:

$$\frac{\overline{u_i^{n+1}} - \overline{u_i^n}}{\Delta t} + \frac{c}{2\Delta x}\left(\overline{u_{i+1}^n} - \overline{u_{i-1}^n}\right) = 0$$

1D Linear Convection with Central Differencing in Space

Consider the exact solution of the discretized equation:

$$\frac{\overline{u_i^{n+1}} - \overline{u_i^n}}{\Delta t} + \frac{c}{2\Delta x}\left(\overline{u_{i+1}^n} - \overline{u_{i-1}^n}\right) = 0$$

$$-\left(\frac{\partial \bar{u}}{\partial t} + c\frac{\partial \bar{u}}{\partial x}\right)_i^n = \frac{\Delta t}{2!}\frac{\partial^2 \bar{u}}{\partial t^2}\bigg|_i^n + \frac{c\Delta x^3}{3!}\frac{\partial^3 \bar{u}}{\partial t^3}\bigg|_i^n + O(\Delta t^2, \Delta x^4)$$

1D Linear Convection with Central Differencing in Space

Consider the exact solution of the discretized equation:

$$\frac{\overline{u_i^{n+1}} - \overline{u_i^n}}{\Delta t} + \frac{c}{2\Delta \mathrm{x}}\left(\overline{u_{i+1}^n} - \overline{u_{i-1}^n}\right) = 0$$

$$-\left(\frac{\partial \bar{u}}{\partial t} + c\,\frac{\partial \bar{u}}{\partial x}\right)_i^n = \frac{\Delta t}{2!}\frac{\partial^2 \bar{u}}{\partial t^2}\bigg|_i^n + \frac{\mathrm{c}\Delta x^3}{3!}\frac{\partial^3 \bar{u}}{\partial t^3}\bigg|_i^n + O(\Delta t^2, \Delta x^4)$$

$$\overline{(u_t)}_i^n = -c\,\overline{(u_x)}_i^n + O(\Delta t, \Delta x)$$

1D Linear Convection with Central Differencing in Space

Consider the exact solution of the discretized equation:

$$\frac{\overline{u_i^{n+1}} - \overline{u_i^n}}{\Delta t} + \frac{c}{2\Delta \mathrm{x}}\left(\overline{u_{i+1}^n} - \overline{u_{i-1}^n}\right) = 0$$

$$-\left(\frac{\partial \bar{u}}{\partial t} + c\frac{\partial \bar{u}}{\partial x}\right)_i^n = \frac{\Delta t}{2!}\frac{\partial^2 \bar{u}}{\partial t^2}\bigg|_i^n + \frac{\mathrm{c}\Delta x^3}{3!}\frac{\partial^3 \bar{u}}{\partial t^3}\bigg|_i^n + O(\Delta t^2, \Delta x^4)$$

$$\overline{(u_t)}_i^n = -c\,\overline{(u_x)}_i^n + O(\Delta t, \Delta x)$$

$$\overline{(u_{tt})}_i^n = -c\,\overline{(u_{xt})}_i^n + O(\Delta t, \Delta x)$$

1D Linear Convection with Central Differencing in Space

Consider the exact solution of the discretized equation:

$$\frac{\overline{u_i^{n+1}} - \overline{u_i^n}}{\Delta t} + \frac{c}{2\Delta x}\left(\overline{u_{i+1}^n} - \overline{u_{i-1}^n}\right) = 0$$

$$-\left(\frac{\partial \bar{u}}{\partial t} + c\frac{\partial \bar{u}}{\partial x}\right)_i^n = \frac{\Delta t}{2!}\frac{\partial^2 \bar{u}}{\partial t^2}\bigg|_i^n + \frac{c\Delta x^3}{3!}\frac{\partial^3 \bar{u}}{\partial t^3}\bigg|_i^n + O(\Delta t^2, \Delta x^4)$$

$$\overline{(u_t)}_i^n = -c\,\overline{(u_x)}_i^n + O(\Delta t, \Delta x)$$

$$\overline{(u_{tt})}_i^n = -c\,\overline{(u_{xt})}_i^n + O(\Delta t, \Delta x)$$

$$\overline{(u_{tt})}_i^n = -c\,\overline{[(u_t)]_x}_i^n + O(\Delta t, \Delta x)$$

1D Linear Convection with Central Differencing in Space

Consider the exact solution of the discretized equation:

$$\frac{\overline{u_i^{n+1}} - \overline{u_i^n}}{\Delta t} + \frac{c}{2\Delta x}\left(\overline{u_{i+1}^n} - \overline{u_{i-1}^n}\right) = 0$$

$$-\left(\frac{\partial \bar{u}}{\partial t} + c\frac{\partial \bar{u}}{\partial x}\right)_i^n = \frac{\Delta t}{2!}\frac{\partial^2 \bar{u}}{\partial t^2}\bigg|_i^n + \frac{c\Delta x^3}{3!}\frac{\partial^3 \bar{u}}{\partial t^3}\bigg|_i^n + O(\Delta t^2, \Delta x^4)$$

$$\overline{(u_t)}_i^n = -c\,\overline{(u_x)}_i^n + O(\Delta t, \Delta x)$$

$$\overline{(u_{tt})}_i^n = -c\,\overline{(u_{xt})}_i^n + O(\Delta t, \Delta x)$$

$$\overline{(u_{tt})}_i^n = -c\,\overline{[(u_t)]_{x}}_i^n + O(\Delta t, \Delta x)$$

$$\overline{(u_{tt})}_i^n = c^2\,\overline{u_{xx}}_i^n + O(\Delta t, \Delta x)$$

1D Linear Convection with Central Differencing in Space

Consider the exact solution of the discretized equation:

$$\frac{\overline{u_i^{n+1}} - \overline{u_i^n}}{\Delta t} + \frac{c}{2\Delta x}\left(\overline{u_{i+1}^n} - \overline{u_{i-1}^n}\right) = 0$$

$$-\left(\frac{\partial \bar{u}}{\partial t} + c\frac{\partial \bar{u}}{\partial x}\right)_i^n = \frac{\Delta t}{2!}\frac{\partial^2 \bar{u}}{\partial t^2}\bigg|_i^n + \frac{c\Delta x^3}{3!}\frac{\partial^3 \bar{u}}{\partial t^3}\bigg|_i^n + O(\Delta t^2, \Delta x^4)$$

$$\overline{(u_t)}_i^n = -c\,\overline{(u_x)}_i^n + O(\Delta t, \Delta x)$$

$$\overline{(u_t)}_i^n + c\,\overline{(u_x)}_i^n = -\frac{\Delta t}{2!}c^2\,\overline{u_{xx}}_i^n + O(\Delta t^2, \Delta x^2)$$

$$\overline{(u_{tt})}_i^n = -c\,\overline{(u_{xt})}_i^n + O(\Delta t, \Delta x)$$

$$\overline{(u_{tt})}_i^n = -c\,\overline{[(u_t)]_x}_i^n + O(\Delta t, \Delta x)$$

$$\overline{(u_{tt})}_i^n = c^2\,\overline{u_{xx}}_i^n + O(\Delta t, \Delta x)$$

1D Linear Convection with Central Differencing in Space

Consider the exact solution of the discretized equation:

$$\frac{\overline{u_i^{n+1}} - \overline{u_i^n}}{\Delta t} + \frac{c}{2\Delta x}\left(\overline{u_{i+1}^n} - \overline{u_{i-1}^n}\right) = 0$$

$$-\left(\frac{\partial \bar{u}}{\partial t} + c\frac{\partial \bar{u}}{\partial x}\right)_i^n = \frac{\Delta t}{2!}\frac{\partial^2 \bar{u}}{\partial t^2}\Big|_i^n + \frac{c\Delta x^3}{3!}\frac{\partial^3 \bar{u}}{\partial t^3}\Big|_i^n + O(\Delta t^2, \Delta x^4)$$

$$\overline{(u_t)}_i^n = -c\,\overline{(u_x)}_i^n + O(\Delta t, \Delta x)$$

$$\overline{(u_{tt})}_i^n = -c\,\overline{(u_{xt})}_i^n + O(\Delta t, \Delta x)$$

$$\overline{(u_{tt})}_i^n = -c\,\overline{[(u_t)]_{x}}_i^n + O(\Delta t, \Delta x)$$

$$\overline{(u_{tt})}_i^n = c^2\,\overline{u_{xx}}_i^n + O(\Delta t, \Delta x)$$

$$\overline{(u_t)}_i^n + c\,\overline{(u_x)}_i^n = -\frac{\Delta t}{2!}c^2\,\overline{u_{xx}}_i^n + O(\Delta t^2, \Delta x^2)$$

Modified Differential Equation !

1D Linear Convection with Central Differencing in Space

Consider the exact solution of the discretized equation:

$$\frac{\overline{u_i^{n+1}} - \overline{u_i^n}}{\Delta t} + \frac{c}{2\Delta x}\left(\overline{u_{i+1}^n} - \overline{u_{i-1}^n}\right) = 0$$

$$-\left(\frac{\partial \bar{u}}{\partial t} + c\frac{\partial \bar{u}}{\partial x}\right)_i^n = \frac{\Delta t}{2!}\frac{\partial^2 \bar{u}}{\partial t^2}\bigg|_i^n + \frac{c\Delta x^3}{3!}\frac{\partial^3 \bar{u}}{\partial t^3}\bigg|_i^n + O(\Delta t^2, \Delta x^4)$$

$$\overline{(u_t)}_i^n = -c\,\overline{(u_x)}_i^n + O(\Delta t, \Delta x)$$

$$\overline{(u_{tt})}_i^n = -c\,\overline{(u_{xt})}_i^n + O(\Delta t, \Delta x)$$

$$\overline{(u_{tt})}_i^n = -c\,\overline{[(u_t)]_{x}}_i^n + O(\Delta t, \Delta x)$$

$$\overline{(u_{tt})}_i^n = c^2\,\overline{u_{xx}}_i^n + O(\Delta t, \Delta x)$$

$$\overline{(u_t)}_i^n + c\,\overline{(u_x)}_i^n = -\frac{\Delta t}{2!}c^2\,\overline{u_{xx}}_i^n + O(\Delta t^2, \Delta x^2)$$

Modified Differential Equation !

NOT a convection equation, It is a Convection-Diffusion equation !!

1D Linear Convection with Central Differencing in Space

Consider the exact solution of the discretized equation:

$$\frac{\overline{u_i^{n+1}} - \overline{u_i^n}}{\Delta t} + \frac{c}{2\Delta x}\left(\overline{u_{i+1}^n} - \overline{u_{i-1}^n}\right) = 0$$

-ve Diffusion Coefficient !!

$$-\left(\frac{\partial \bar{u}}{\partial t} + c\frac{\partial \bar{u}}{\partial x}\right)_i^n = \frac{\Delta t}{2!}\frac{\partial^2 \bar{u}}{\partial t^2}\bigg|_i^n + \frac{c\Delta x^3}{3!}\frac{\partial^3 \bar{u}}{\partial t^3}\bigg|_i^n + O(\Delta t^2, \Delta x^4)$$

$$\overline{(u_t)}_i^n = -c\,\overline{(u_x)}_i^n + O(\Delta t, \Delta x)$$

$$\overline{(u_{tt})}_i^n = -c\,\overline{(u_{xt})}_i^n + O(\Delta t, \Delta x)$$

$$\overline{(u_{tt})}_i^n = -c\,\overline{[(u_t)]_{x}}_i^n + O(\Delta t, \Delta x)$$

$$\overline{(u_{tt})}_i^n = c^2\,\overline{u_{xx}}_i^n + O(\Delta t, \Delta x)$$

$$\overline{(u_t)}_i^n + c\,\overline{(u_x)}_i^n = -\frac{\Delta t}{2!}c^2\,\overline{u_{xx}}_i^n + O(\Delta t^2, \Delta x^2)$$

Modified Differential Equation !

NOT a convection equation, It is a Convection-Diffusion equation !!

1D Linear Convection with Central Differencing in Space

Consider the exact solution of the discretized equation:

$$\frac{\overline{u_i^{n+1}} - \overline{u_i^n}}{\Delta t} + \frac{c}{2\Delta x}\left(\overline{u_{i+1}^n} - \overline{u_{i-1}^n}\right) = 0$$

$$-\left(\frac{\partial \bar{u}}{\partial t} + c\frac{\partial \bar{u}}{\partial x}\right)_i^n = \frac{\Delta t}{2!}\frac{\partial^2 \bar{u}}{\partial t^2}\bigg|_i^n + \frac{c\Delta x^3}{3!}\frac{\partial^3 \bar{u}}{\partial t^3}\bigg|_i^n + O(\Delta t^2, \Delta x^4)$$

$$\overline{(u_t)}_i^n = -c\,\overline{(u_x)}_i^n + O(\Delta t, \Delta x)$$

$$\overline{(u_{tt})}_i^n = -c\,\overline{(u_{xt})}_i^n + O(\Delta t, \Delta x)$$

$$\overline{(u_{tt})}_i^n = -c\,\overline{[(u_t)]_{x}}_i^n + O(\Delta t, \Delta x)$$

$$\overline{(u_{tt})}_i^n = c^2\,\overline{u_{xx}}_i^n + O(\Delta t, \Delta x)$$

Explosion : Unstable Scheme !!

-ve Diffusion Coefficient !!

$$\overline{(u_t)}_i^n + c\,\overline{(u_x)}_i^n = -\frac{\Delta t}{2!}c^2\,\overline{u_{xx}}_i^n + O(\Delta t^2, \Delta x^2)$$

Modified Differential Equation !

NOT a convection equation, It is a Convection-Diffusion equation !!

1D Linear Convection with Backward Differencing in Space

Consider the exact solution of the discretized equation:

$$\frac{\overline{u_i^{n+1}} - \overline{u_i^n}}{\Delta t} + \frac{c}{2\Delta \mathrm{x}}\left(\overline{u_i^n} - \overline{u_{i-1}^n}\right) = 0$$

1D Linear Convection with Backward Differencing in Space

Consider the exact solution of the discretized equation:

$$\frac{\overline{u_i^{n+1}} - \overline{u_i^n}}{\Delta t} + \frac{c}{2\Delta \mathrm{x}}\left(\overline{u_i^n} - \overline{u_{i-1}^n}\right) = 0$$

Modified Differential Equation: $\left(\frac{\partial \bar{u}}{\partial t} + c\frac{\partial \bar{u}}{\partial x}\right)_i^n = \frac{c\Delta x}{2}\left(1 - \frac{c\Delta \mathrm{t}}{\Delta x}\right)\frac{\partial^2 \bar{u}}{\partial x^2}\Big|_i^n$ Convection-Diffusion equation !!

1D Linear Convection with Backward Differencing in Space

Consider the exact solution of the discretized equation:

$$\frac{\overline{u_i^{n+1}} - \overline{u_i^n}}{\Delta t} + \frac{c}{2\Delta x}\left(\overline{u_i^n} - \overline{u_{i-1}^n}\right) = 0$$

Modified Differential Equation: $\left(\frac{\partial \bar{u}}{\partial t} + c\frac{\partial \bar{u}}{\partial x}\right)_i^n = \frac{c\Delta x}{2}\left(1 - \frac{c\Delta t}{\Delta x}\right)\frac{\partial^2 \bar{u}}{\partial x^2}\Big|_i^n$ Convection-Diffusion equation !!

Diffusion Coefficient must be +ve !!

1D Linear Convection with Backward Differencing in Space

Consider the exact solution of the discretized equation:

$$\frac{\overline{u_i^{n+1}} - \overline{u_i^n}}{\Delta t} + \frac{c}{2\Delta x}\left(\overline{u_i^n} - \overline{u_{i-1}^n}\right) = 0$$

Modified Differential Equation: $\left(\frac{\partial \bar{u}}{\partial t} + c\frac{\partial \bar{u}}{\partial x}\right)_i^n = \frac{c\Delta x}{2}\left(1 - \frac{c\Delta t}{\Delta x}\right)\frac{\partial^2 \bar{u}}{\partial x^2}\bigg|_i^n$ Convection-Diffusion equation !!

Diffusion Coefficient must be +ve !!

For stability, we need: $0 \leq \frac{c\Delta t}{\Delta x} \leq 1$

1D Linear Convection with Backward Differencing in Space

Consider the exact solution of the discretized equation:

$$\frac{\overline{u_i^{n+1}} - \overline{u_i^n}}{\Delta t} + \frac{c}{2\Delta x}\left(\overline{u_i^n} - \overline{u_{i-1}^n}\right) = 0$$

Modified Differential Equation: $\left(\frac{\partial \bar{u}}{\partial t} + c\frac{\partial \bar{u}}{\partial x}\right)_i^n = \frac{c\Delta x}{2}\left(1 - \frac{c\Delta t}{\Delta x}\right)\frac{\partial^2 \bar{u}}{\partial x^2}\Big|_i^n$ Convection-Diffusion equation !!

Diffusion Coefficient must be +ve !!

For stability, we need: $0 \leq \frac{c\Delta t}{\Delta x} \leq 1$

Courant-Friedrich-Lewy (CFL) Number!!

1D Linear Convection with Backward Differencing in Space

Consider the exact solution of the discretized equation:

$$\frac{\overline{u_i^{n+1}} - \overline{u_i^n}}{\Delta t} + \frac{c}{2\Delta \mathrm{x}}\left(\overline{u_i^n} - \overline{u_{i-1}^n}\right) = 0$$

Modified Differential Equation: $\left(\frac{\partial \bar{u}}{\partial t} + c\frac{\partial \bar{u}}{\partial x}\right)_i^n = \frac{\mathrm{c}\Delta x}{2}\left(1 - \frac{c\Delta \mathrm{t}}{\Delta x}\right)\frac{\partial^2 \bar{u}}{\partial x^2}\Big|_i^n$ Convection-Diffusion equation !!

Diffusion Coefficient must be +ve !!

For stability, we need: $0 \leq \frac{c\Delta t}{\Delta x} \leq 1$

For a CFL of < 1: Numerical Diffusion is of $O(\Delta x)$

Courant-Friedrich-Lewy (CFL) Number!!

1D Linear Convection with Backward Differencing in Space

Consider the exact solution of the discretized equation:

$$\frac{\overline{u_i^{n+1}} - \overline{u_i^n}}{\Delta t} + \frac{c}{2\Delta \mathrm{x}}\left(\overline{u_i^n} - \overline{u_{i-1}^n}\right) = 0$$

Modified Differential Equation: $$\left(\frac{\partial \bar{u}}{\partial t} + c\frac{\partial \bar{u}}{\partial x}\right)_i^n = \frac{c\Delta x}{2}\left(1 - \frac{c\Delta t}{\Delta x}\right)\frac{\partial^2 \bar{u}}{\partial x^2}\bigg|_i^n$$

Convection-Diffusion equation !!

Diffusion Coefficient must be +ve !!

For stability, we need: $0 \le \frac{c\Delta t}{\Delta x} \le 1$

Courant-Friedrich-Lewy (CFL) Number!!

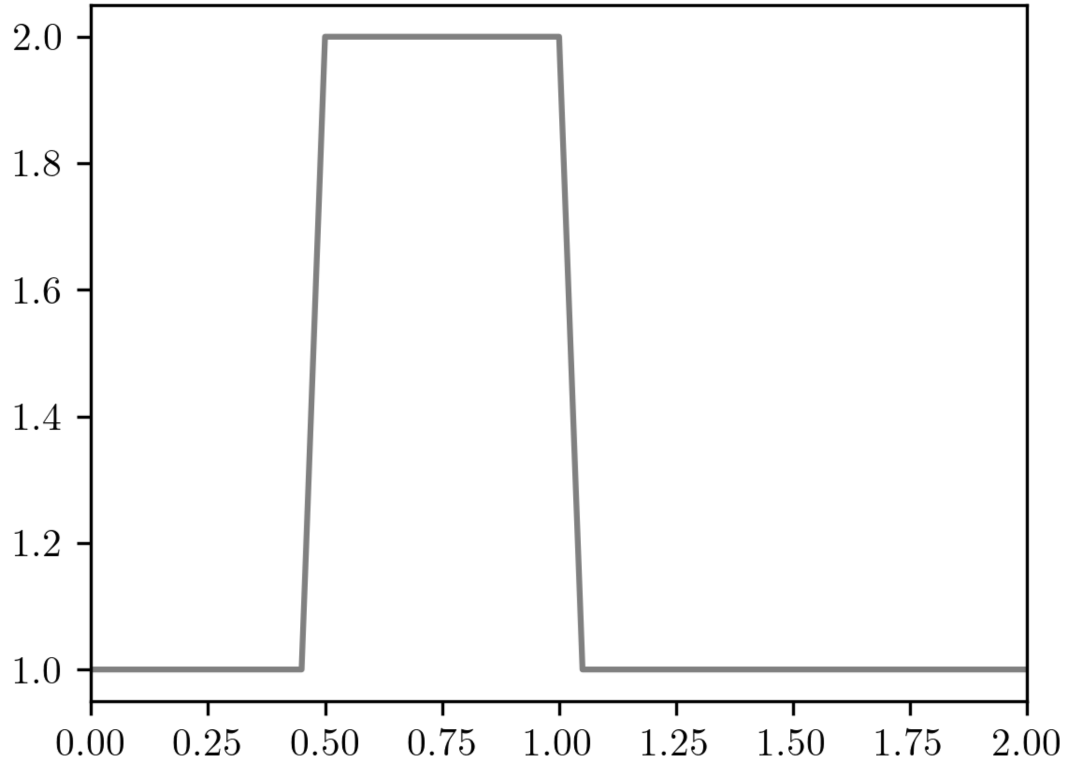
For a CFL of < 1: Numerical Diffusion is of $O(\Delta x)$

Poor Accuracy !!

2^{nd} Code: 1D non-Linear Convection

```
import numpy as np
import matplotlib.pyplot as plt
import time, sys

nx = 41
dx = 2/(nx-1)
nt = 25
dt = 0.025
c = 1

u = np.ones(nx)
u[int(0.5/dx):int(1/dx+1)] = 2
print(u)
plt.plot(np.linspace(0,2,nx), u)

un = np.ones(nx)
for n in range(nt):
    un=u.copy()
    for i in range(1,nx):
        u[i] = un[i] – c * dt/dx * (un[i]-un[i-1])
plt.plot(np.linspace(0,2,nx), u)
```

The initial velocity u_0 is given as 2 in the interval $0.5 \leq x \leq 1$ and 1 elsewhere in $(0,2)$ $\Longrightarrow$ Hat function

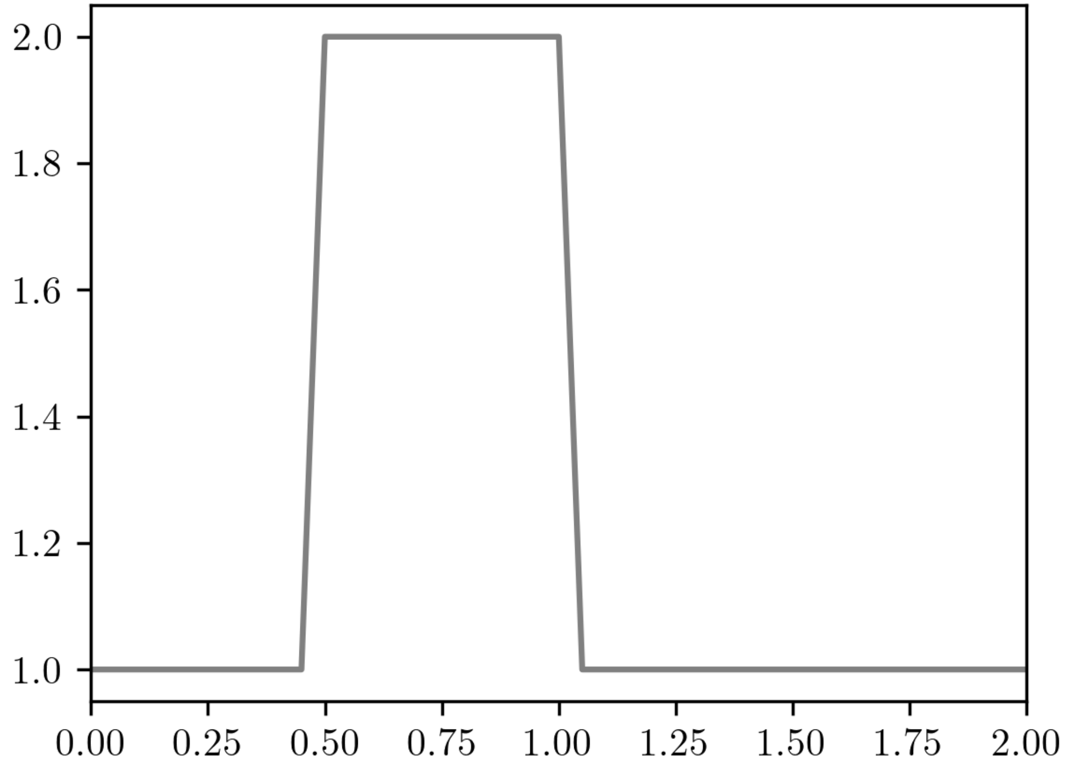


2^{nd} Code: 1D non-Linear Convection

```
import numpy as np
import matplotlib.pyplot as plt
import time, sys

nx = 41
dx = 2/(nx-1)
nt = 25
dt = 0.025
c = 1

u = np.ones(nx)
u[int(0.5/dx):int(1/dx+1)] = 2
print(u)
plt.plot(np.linspace(0,2,nx), u)

un = np.ones(nx)
for n in range(nt):
    un=u.copy()
    for i in range(1,nx):
        u[i] = un[i] - c * dt/dx * (un[i]-un[i-1])
plt.plot(np.linspace(0,2,nx), u)
```

Change this only !

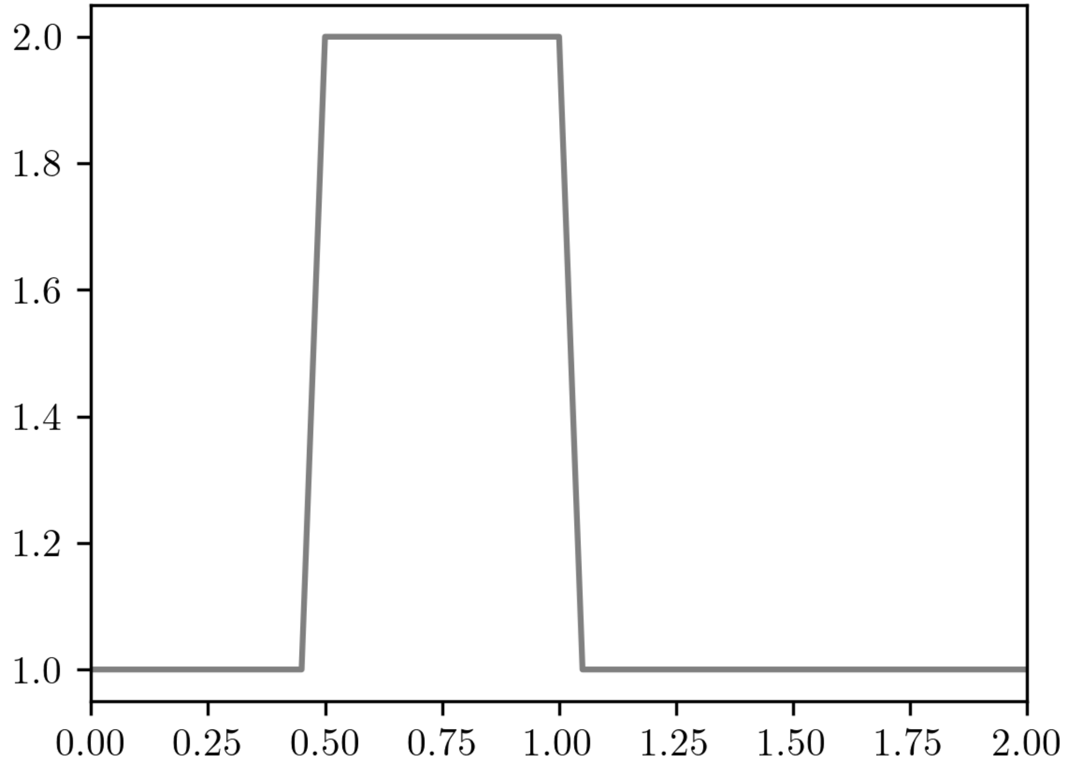
The initial velocity u_0 is given as 2 in the interval $0.5 \leq x \leq 1$ and 1 elsewhere in $(0,2)$ $\Longrightarrow$ Hat function



2nd Code: 1D non-Linear Convection

```
import numpy as np
import matplotlib.pyplot as plt
import time, sys

nx = 41
dx = 2/(nx-1)
nt = 25
dt = 0.025
c = 1

u = np.ones(nx)
u[int(0.5/dx):int(1/dx+1)] = 2
print(u)
plt.plot(np.linspace(0,2,nx), u)

un = np.ones(nx)
for n in range(nt):
    un=u.copy()
    for i in range(1,nx):
        u[i] = un[i] – un[i] * dt/dx * (un[i]-un[i-1])
plt.plot(np.linspace(0,2,nx), u)
```

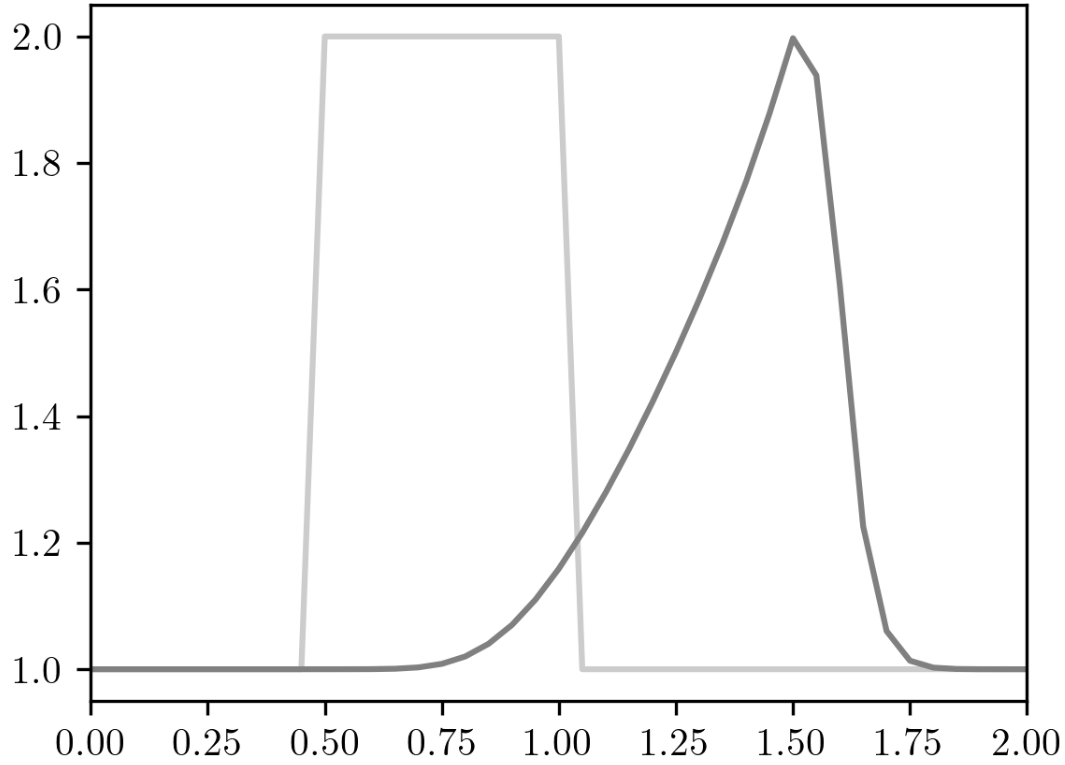
The initial velocity u_0 is given as 2 in the interval $0.5 \le x \le 1$ and 1 elsewhere in $(0,2)$ $\Longrightarrow$ Hat function



2nd Code: 1D non-Linear Convection

```
import numpy as np
import matplotlib.pyplot as plt
import time, sys

nx = 41
dx = 2/(nx-1)
nt = 25
dt = 0.025
c = 1

u = np.ones(nx)
u[int(0.5/dx):int(1/dx+1)] = 2
print(u)
plt.plot(np.linspace(0,2,nx), u)

un = np.ones(nx)
for n in range(nt):
    un=u.copy()
    for i in range(1,nx):
        u[i] = un[i] – un[i] * dt/dx * (un[i]-un[i-1])
plt.plot(np.linspace(0,2,nx), u)
```

The initial velocity u_0 is given as 2 in the interval $0.5 \leq x \leq 1$ and 1 elsewhere in $(0,2)$ $\Longrightarrow$ Hat function



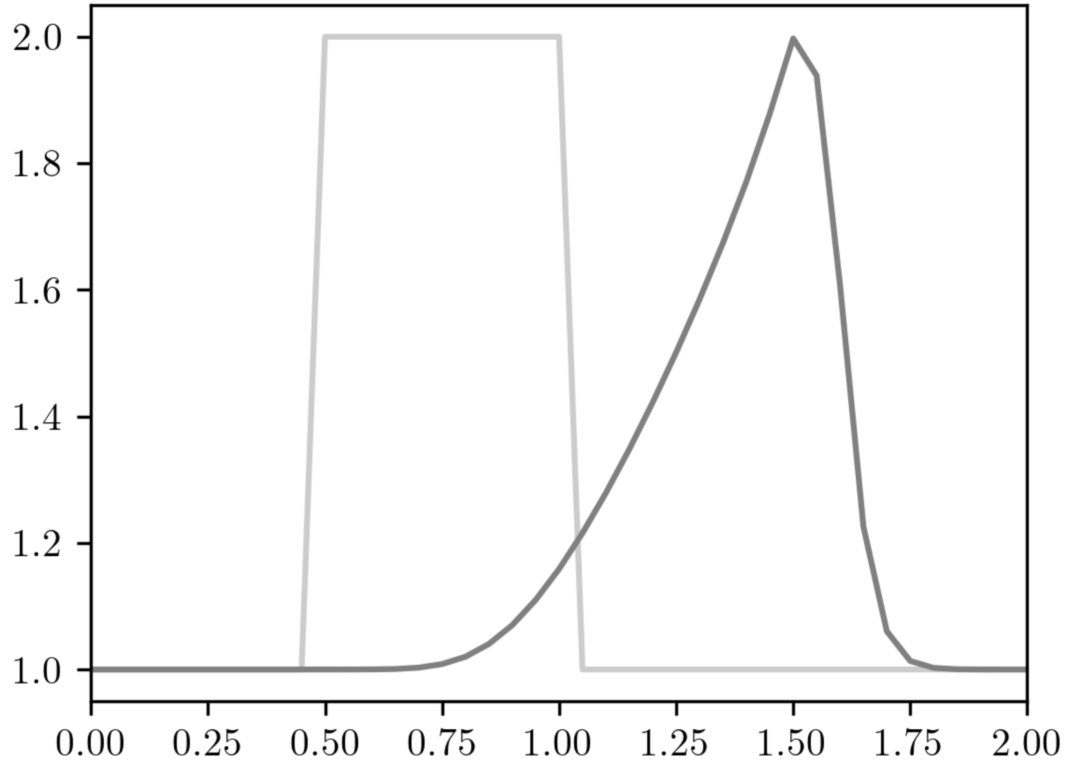
2^{nd} Code: 1D non-Linear Convection

MODELAIR

```
import numpy as np
import matplotlib.pyplot as plt
import time, sys

nx = 41
dx = 2/(nx-1)
nt = 25
dt = 0.025
c = 1

u = np.ones(nx)
u[int(0.5/dx):int(1/dx+1)] = 2
print(u)
plt.plot(np.linspace(0,2,nx), u)

un = np.ones(nx)
for n in range(nt):
    un=u.copy()
    for i in range(1,nx):
        u[i] = un[i] – un[i] * dt/dx * (un[i]-un[i-1])
plt.plot(np.linspace(0,2,nx), u)
```

Let's introduce CFL

The initial velocity u_0 is given as 2 in the interval $0.5 \leq x \leq 1$ and 1 elsewhere in $(0,2)$ $\Longrightarrow$ Hat function

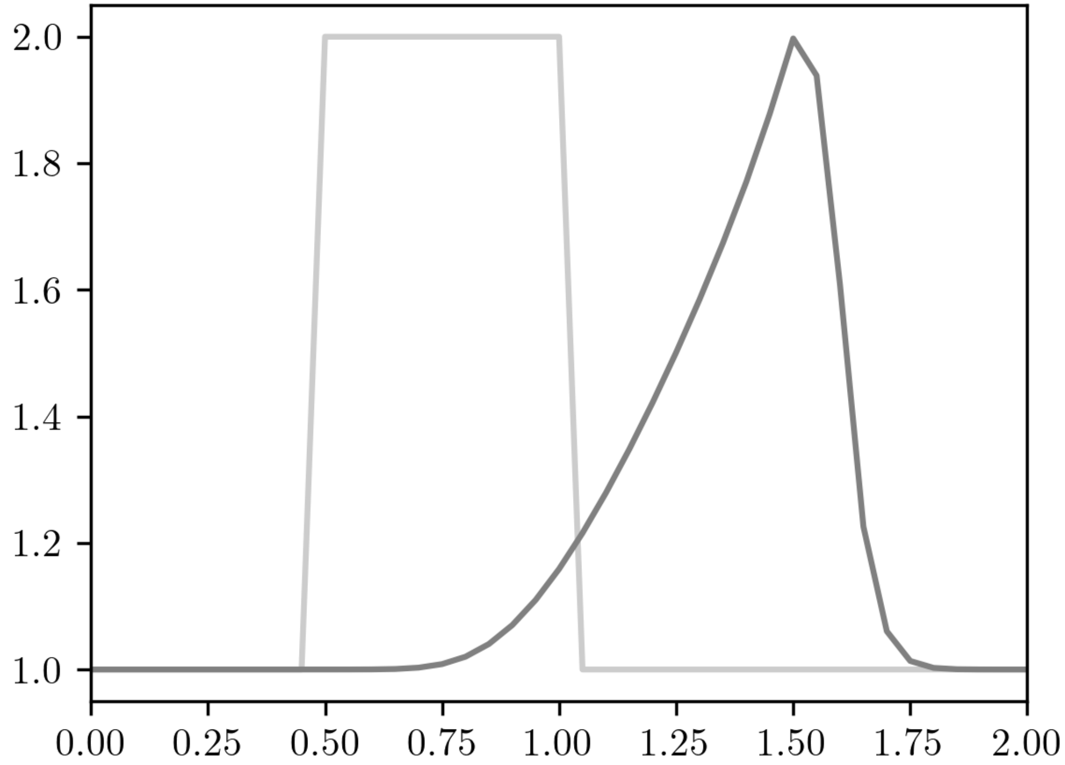


2^{nd} Code: 1D non-Linear Convection

```
import numpy as np
import matplotlib.pyplot as plt
import time, sys

nx = 41
dx = 2/(nx-1)
nt = 25
CFL = 0.9

u = np.ones(nx)
u[int(0.5/dx):int(1/dx+1)] = 2
print(u)
plt.plot(np.linspace(0,2,nx), u)

dt = CFL*dx/max(abs(u))

un = np.ones(nx)
for n in range(nt):
    un=u.copy()
    for i in range(1,nx):
        u[i] = un[i] – un[i] * dt/dx * (un[i]-un[i-1])
plt.plot(np.linspace(0,2,nx), u)
```

Let's introduce CFL

The initial velocity u_0 is given as 2 in the interval $0.5 \leq x \leq 1$ and 1 elsewhere in $(0,2) \Longrightarrow$ Hat function



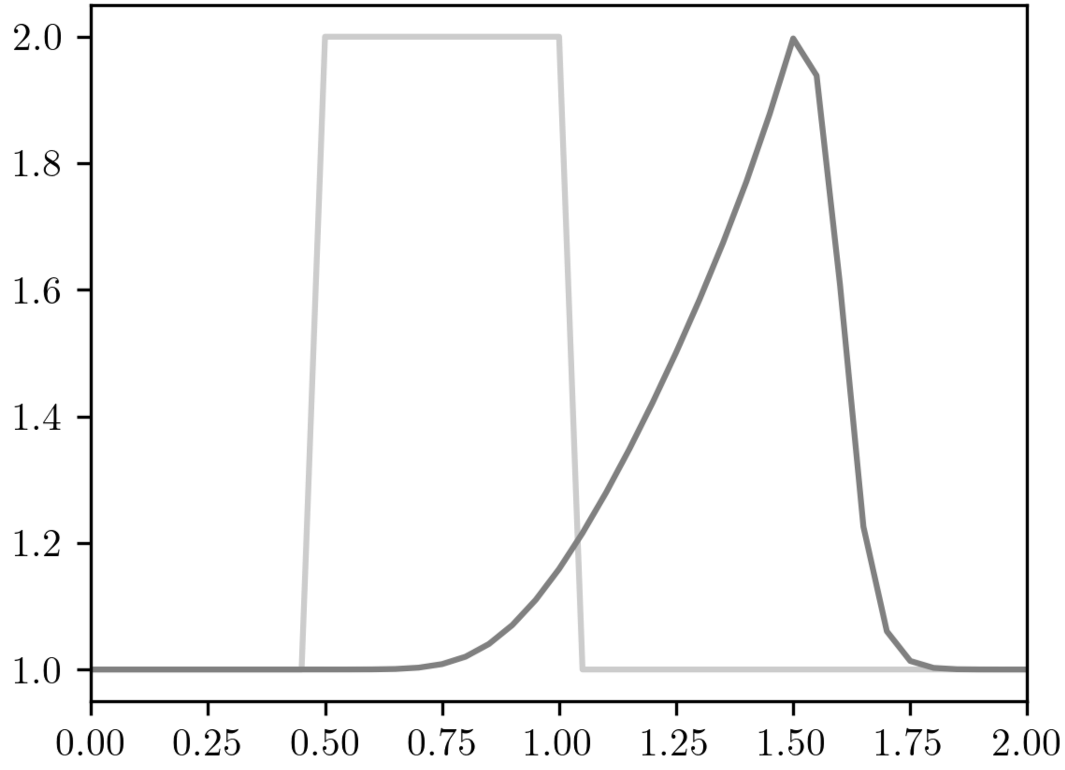
2nd Code: 1D non-Linear Convection

```
import numpy as np
import matplotlib.pyplot as plt
import time, sys

nx = 41
dx = 2/(nx-1)
nt = 25
CFL = 0.9

u = np.ones(nx)
u[int(0.5/dx):int(1/dx+1)] = 2
print(u)
plt.plot(np.linspace(0,2,nx), u)

dt = CFL*dx/max(abs(u))

un = np.ones(nx)
for n in range(nt):
    un=u.copy()
    for i in range(1,nx):
        u[i] = un[i] – un[i] * dt/dx * (un[i]-un[i-1])
plt.plot(np.linspace(0,2,nx), u)
```

Let's introduce CFL

Play a bit & see what happens!

The initial velocity u_0 is given as 2 in the interval $0.5 \leq x \leq 1$ and 1 elsewhere in $(0,2) \Longrightarrow$ Hat function



3rd Code: 1D Diffusion

$$\frac{du}{dt} = \nu \frac{d^2u}{dx^2}$$

3rd Code: 1D Diffusion

$$\frac{du}{dt} = \nu \frac{d^2u}{dx^2}$$

Exact Solution is known for constant ν

3rd Code: 1D Diffusion

$$\frac{du}{dt} = \nu \frac{d^2u}{dx^2}$$

Exact Solution is known for constant ν

Consider solution of type: $u = \hat{u}e^{i(\kappa x - \omega t)}$

3rd Code: 1D Diffusion

$$\frac{du}{dt} = \nu \frac{d^2u}{dx^2}$$

Exact Solution is known for constant ν

Consider solution of type: $u = \hat{u}e^{i(\kappa x - \omega t)}$

Introducing in the PDE, we obtain: $i\omega = \nu\kappa^2$

3^{rd} Code: 1D Diffusion

$$\frac{du}{dt} = \nu \frac{d^2u}{dx^2}$$

Exact Solution is known for constant ν

Consider solution of type: $u = \hat{u}e^{i(\kappa x - \omega t)}$

Introducing in the PDE, we obtain: $i\omega = \nu\kappa^2$

Leading to the solution: $u = \hat{u}e^{i\kappa x}e^{-\nu\kappa^2 t}$

3rd Code: 1D Diffusion

$$\frac{du}{dt} = \nu \frac{d^2u}{dx^2}$$

Exact Solution is known for constant ν

Consider solution of type: $u = \hat{u}e^{i(\kappa x - \omega t)}$

Introducing in the PDE, we obtain: $i\omega = \nu\kappa^2$

Leading to the solution: $u = \hat{u}e^{i\kappa x}e^{-\nu\kappa^2 t}$

Exponential Damping for $\nu > 0$

3rd Code: 1D Diffusion

$$\frac{du}{dt} = \nu \frac{d^2u}{dx^2}$$

Exact Solution is known for constant ν

Consider solution of type: $u = \hat{u}e^{i(\kappa x - \omega t)}$

Diffusion is isotropic in nature

Introducing in the PDE, we obtain: $i\omega = \nu\kappa^2$

Leading to the solution: $u = \hat{u}e^{i\kappa x}e^{-\nu\kappa^2 t}$

Exponential Damping for $\nu > 0$

3rd Code: 1D Diffusion

$$\frac{du}{dt} = \nu \frac{d^2u}{dx^2}$$

Exact Solution is known for constant ν

Consider solution of type: $u = \hat{u}e^{i(\kappa x - \omega t)}$

Introducing in the PDE, we obtain: $i\omega = \nu\kappa^2$

Leading to the solution: $u = \hat{u}e^{i\kappa x}e^{-\nu\kappa^2 t}$

Exponential Damping for $\nu > 0$

Diffusion is isotropic in nature

No Directional Bias

3rd Code: 1D Diffusion

$$\frac{du}{dt} = \nu \frac{d^2u}{dx^2}$$

Exact Solution is known for constant ν

Consider solution of type: $u = \hat{u}e^{i(\kappa x - \omega t)}$

Introducing in the PDE, we obtain: $i\omega = \nu\kappa^2$

Leading to the solution: $u = \hat{u}e^{i\kappa x}e^{-\nu\kappa^2 t}$

Exponential Damping for $\nu > 0$

Diffusion is isotropic in nature

No Directional Bias

Central-Differencing

3rd Code: 1D Diffusion

$$\frac{du}{dt} = \nu \frac{d^2u}{dx^2}$$

Central Difference $$\frac{d^2u}{dx^2} \approx \frac{u(x+\Delta x) - 2u(x) + u(x-\Delta x)}{\Delta x^2}$$

Forward Euler $$\frac{du}{dt} \approx \frac{u^{n+1}(x) - u^n(x)}{\Delta t}$$

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = \nu \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2}$$

3rd Code: 1D Diffusion

$$\frac{du}{dt} = \nu \frac{d^2u}{dx^2}$$

Central Difference $\quad \dfrac{d^2u}{dx^2} \approx \dfrac{u(x+\Delta x) - 2u(x) + u(x-\Delta x)}{\Delta x^2}$

Forward Euler $\quad \dfrac{du}{dt} \approx \dfrac{u^{n+1}(x) - u^n(x)}{\Delta t}$

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = \nu \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2}$$

$$\boxed{u_i^{n+1} = u_i^n + \frac{\nu \Delta t}{\Delta \mathrm{x}^2}(u_{i-1}^n - 2u_i^n + u_{i-1}^n)}$$

3rd Code: 1D Diffusion

$$\frac{du}{dt} = \nu \frac{d^2u}{dx^2}$$

Central Difference $\quad \frac{d^2u}{dx^2} \approx \frac{u(x+\Delta x) - 2u(x) + u(x-\Delta x)}{\Delta x^2}$

Forward Euler $\quad \frac{du}{dt} \approx \frac{u^{n+1}(x) - u^n(x)}{\Delta t}$

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = \nu \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2}$$

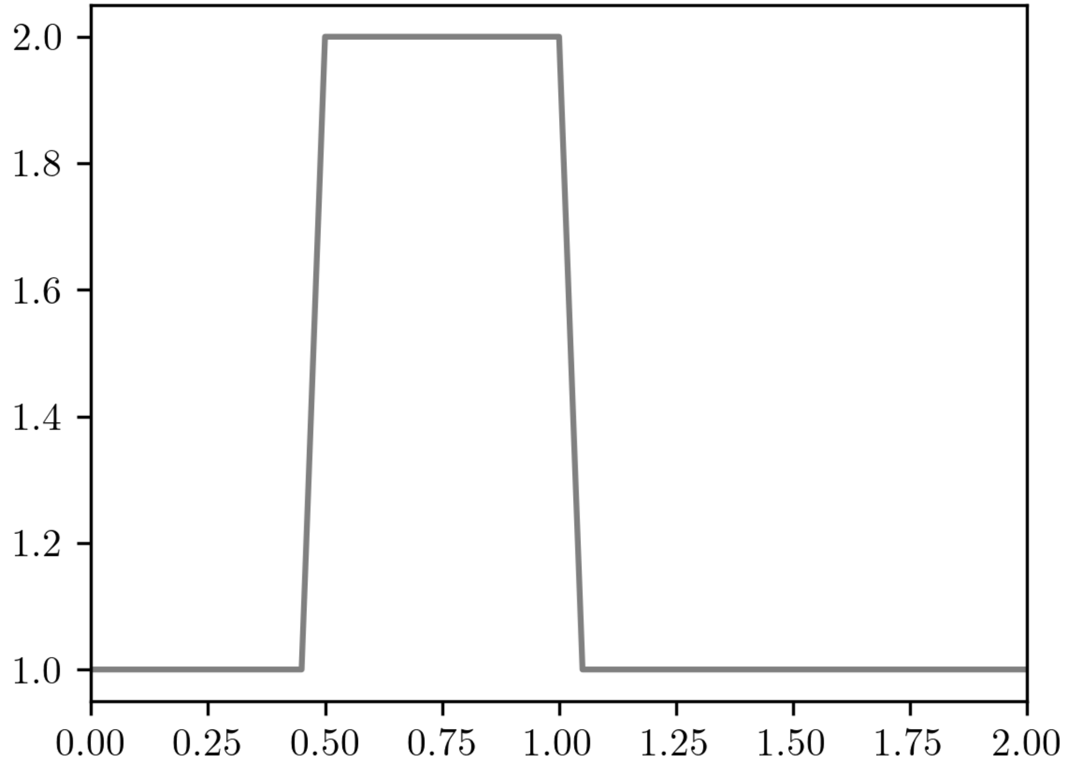
$$\boxed{u_i^{n+1} = u_i^n + \frac{\nu \Delta t}{\Delta \mathrm{x}^2}(u_{i-1}^n - 2u_i^n + u_{i-1}^n)}$$

Let's Code It !

3rd Code: 1D Diffusion

```
import numpy as np
import matplotlib.pyplot as plt
import time, sys

nx = 41; dx = 2/(nx-1)
nt = 25
nu = 0.3
sigma = 0.2
dt = sigma * dx**2 / nu

u = np.ones(nx)
u[int(0.5/dx):int(1/dx+1)] = 2
print(u)
plt.plot(np.linspace(0,2,nx), u)
```

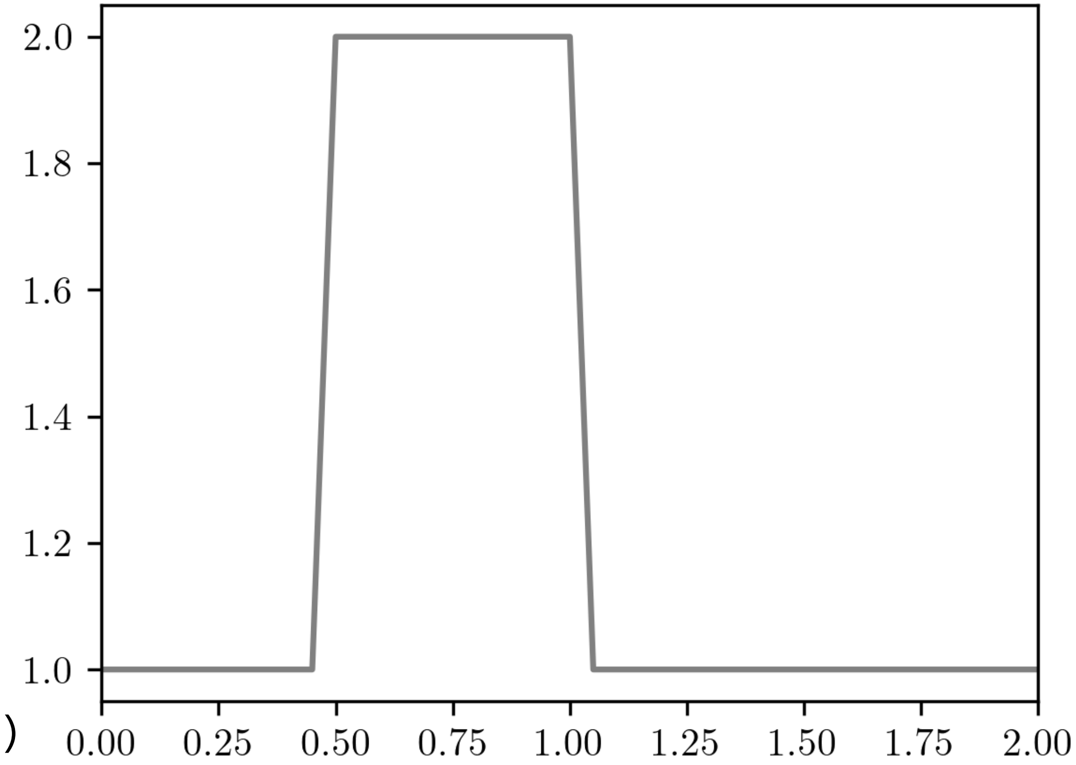
The initial velocity u_0 is given as 2 in the interval $0.5 \leq x \leq 1$ and 1 elsewhere in $(0,2)$ $\Longrightarrow$ Hat function



3rd Code: 1D Diffusion

```
import numpy as np
import matplotlib.pyplot as plt
import time, sys

nx = 41; dx = 2/(nx-1)
nt = 25
nu = 0.3
sigma = 0.2
dt = sigma * dx**2 / nu

u = np.ones(nx)
u[int(0.5/dx):int(1/dx+1)] = 2
print(u)
plt.plot(np.linspace(0,2,nx), u)

un = np.ones(nx)
for n in range(nt):
   un=u.copy()
   for i in range(1,nx-1):
      u[i] = un[i] + nu * dt/dx**2 * (un[i+1]-2*un[i]+un[i-1])
plt.plot(np.linspace(0,2,nx), u)
```

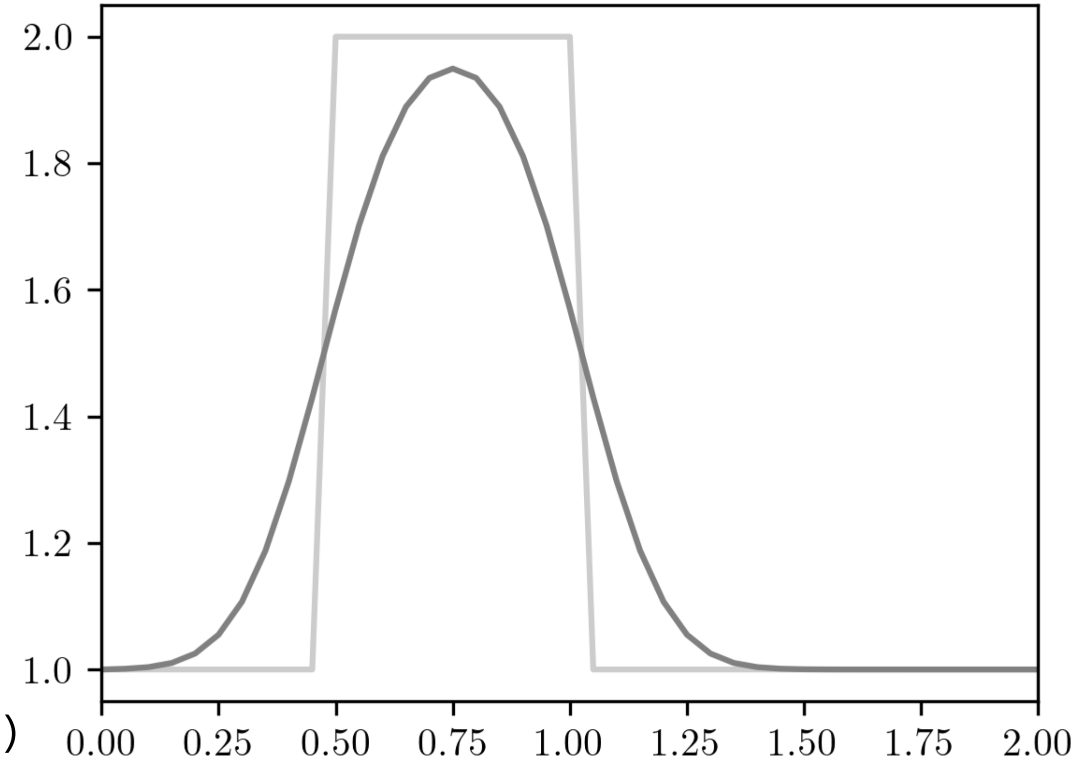
The initial velocity u_0 is given as 2 in the interval $0.5 \leq x \leq 1$ and 1 elsewhere in $(0,2) \Longrightarrow$ Hat function



3^{rd} Code: 1D Diffusion

```
import numpy as np
import matplotlib.pyplot as plt
import time, sys

nx = 41; dx = 2/(nx-1)
nt = 25
nu = 0.3
sigma = 0.2
dt = sigma * dx**2 / nu

u = np.ones(nx)
u[int(0.5/dx):int(1/dx+1)] = 2
print(u)
plt.plot(np.linspace(0,2,nx), u)

un = np.ones(nx)
for n in range(nt):
    un=u.copy()
    for i in range(1,nx-1):
        u[i] = un[i] + nu * dt/dx**2 * (un[i+1]-2*un[i]+un[i-1])
plt.plot(np.linspace(0,2,nx), u)
```

The initial velocity u_0 is given as 2 in the interval $0.5 \le x \le 1$ and 1 elsewhere in $(0,2) \Longrightarrow$ Hat function



1D Diffusion (Caution!)

$$\frac{du}{dt} = \nu \frac{d^2u}{dx^2}$$

1D Diffusion (Caution!)

Parabolic PDE

$$\frac{du}{dt} = \nu \frac{d^2u}{dx^2}$$

1D Diffusion (Caution!)

Parabolic PDE

$$\frac{du}{dt} = \nu \frac{d^2u}{dx^2}$$

Characteristic lines are lines of constant 't'

All information at a given time should affect the solution

1D Diffusion (Caution!)

Parabolic PDE

$$\frac{du}{dt} = \nu \frac{d^2u}{dx^2}$$

Characteristic lines are lines of constant 't'

All information at a given time should affect the solution

Our Numerical Formulation:

$$u_i^{n+1} = u_i^n + \frac{\nu \Delta t}{\Delta x^2}(u_{i-1}^n - 2u_i^n + u_{i-1}^n)$$

1D Diffusion (Caution!)

Parabolic PDE

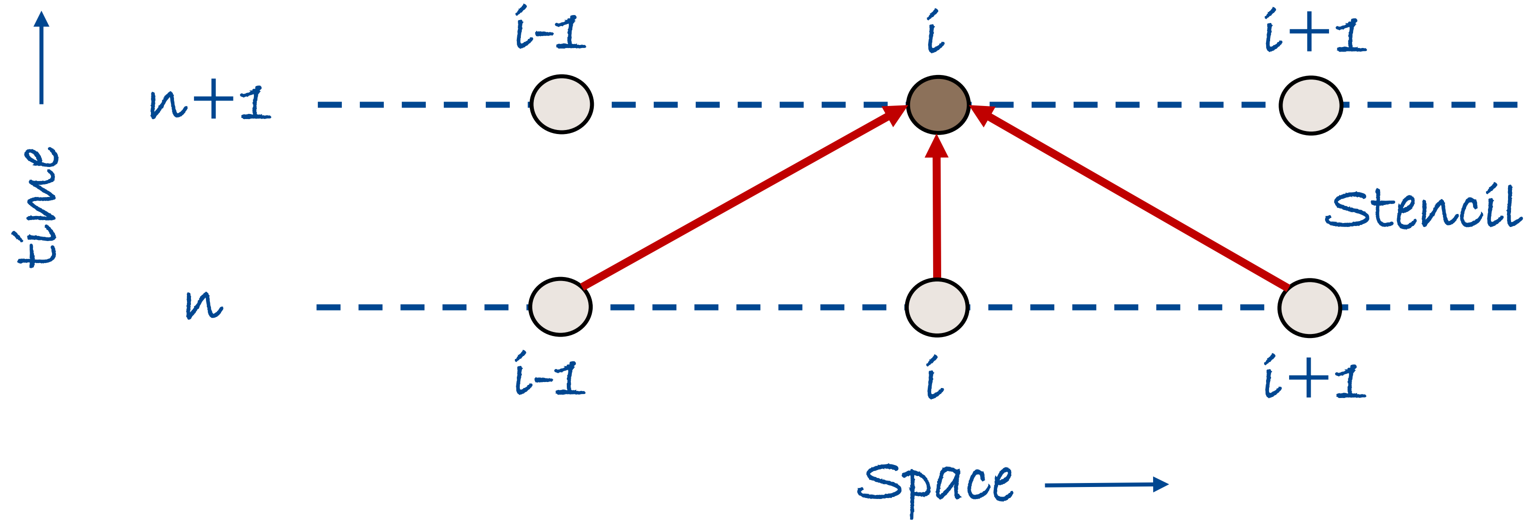
$$\frac{du}{dt} = \nu \frac{d^2u}{dx^2}$$

Characteristic lines are lines of constant 't'

All information at a given time should affect the solution

Our Numerical Formulation:

$$u_i^{n+1} = u_i^n + \frac{\nu\Delta t}{\Delta \mathrm{x}^2}(u_{i-1}^n - 2u_i^n + u_{i-1}^n)$$



1D Diffusion (Caution!)

Parabolic PDE

$$\frac{du}{dt} = \nu \frac{d^2u}{dx^2}$$

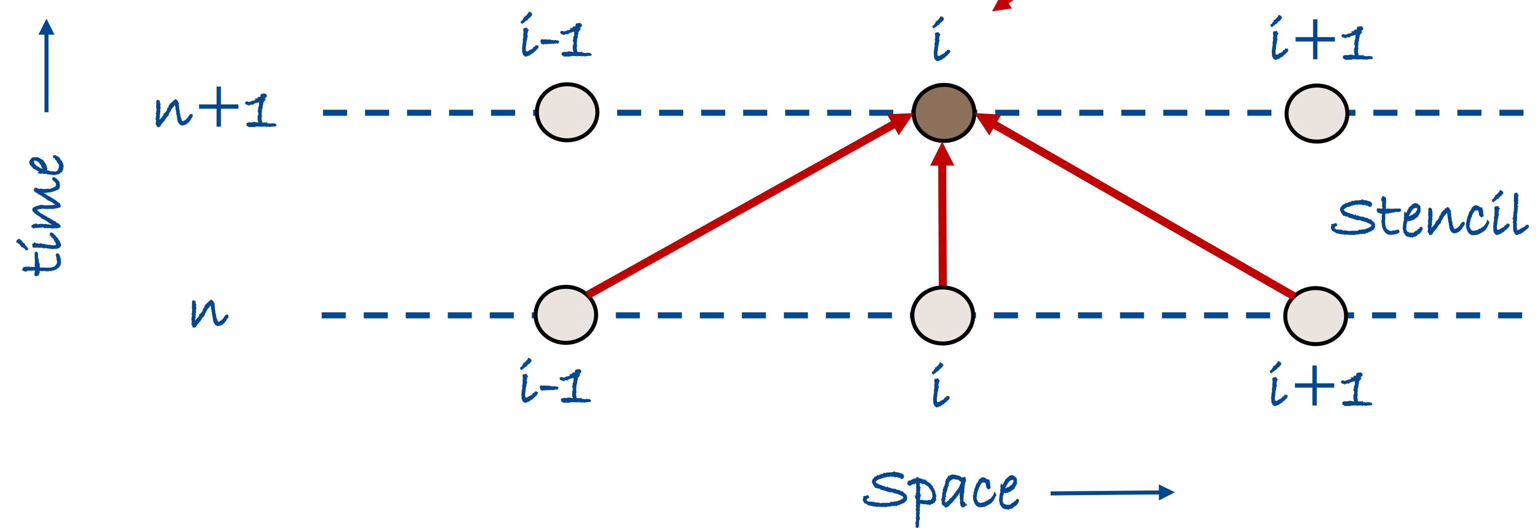
Characteristic lines are lines of constant 't'

All information at a given time should affect the solution

Our Numerical Formulation:

$$u_i^{n+1} = u_i^n + \frac{\nu \Delta t}{\Delta x^2}\left(u_{i-1}^n - 2u_i^n + u_{i-1}^n\right)$$

Our numerical scheme does not receive information from current time !



1D Diffusion (Caution!)

Parabolic PDE

Characteristic lines are lines of constant 't'

$$\frac{du}{dt} = \nu \frac{d^2u}{dx^2}$$

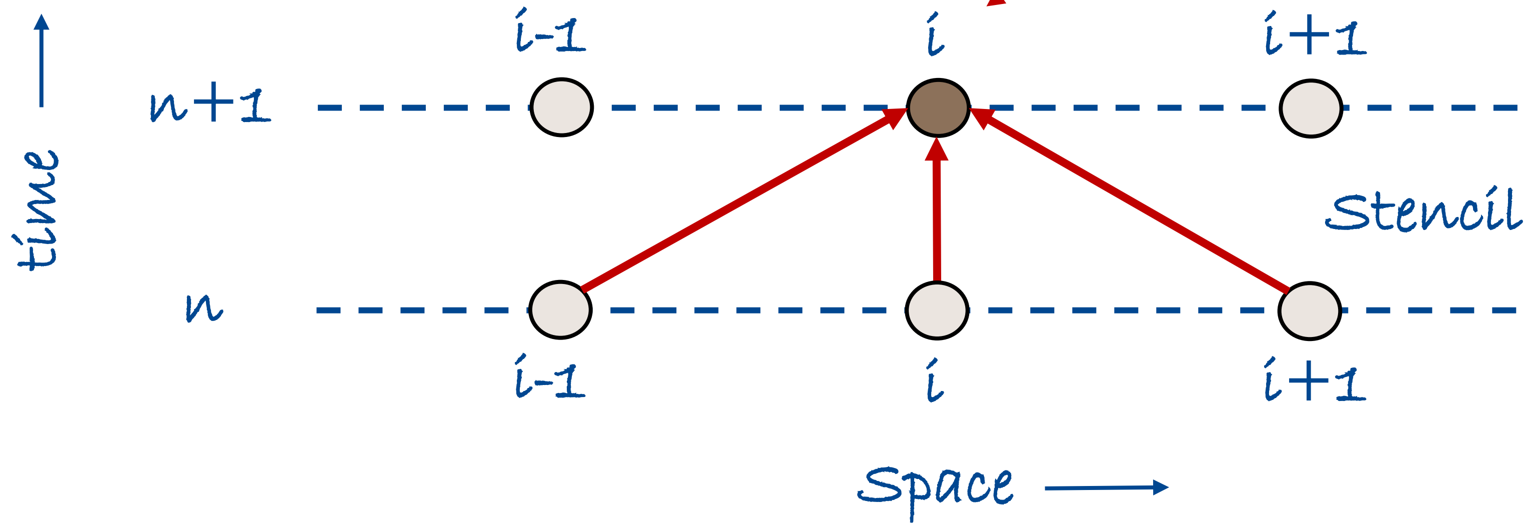
All information at a given time should affect the solution

Our Numerical Formulation:

Our numerical scheme does not receive information from current time !

$$u_i^{n+1} = u_i^n + \frac{\nu \Delta t}{\Delta x^2}(u_{i-1}^n - 2u_i^n + u_{i-1}^n)$$

"Explicit Formulation"



1D Diffusion (Caution!)

Parabolic PDE

$$\frac{du}{dt} = \nu \frac{d^2u}{dx^2}$$

Characteristic lines are lines of constant 't'

All information at a given time should affect the solution

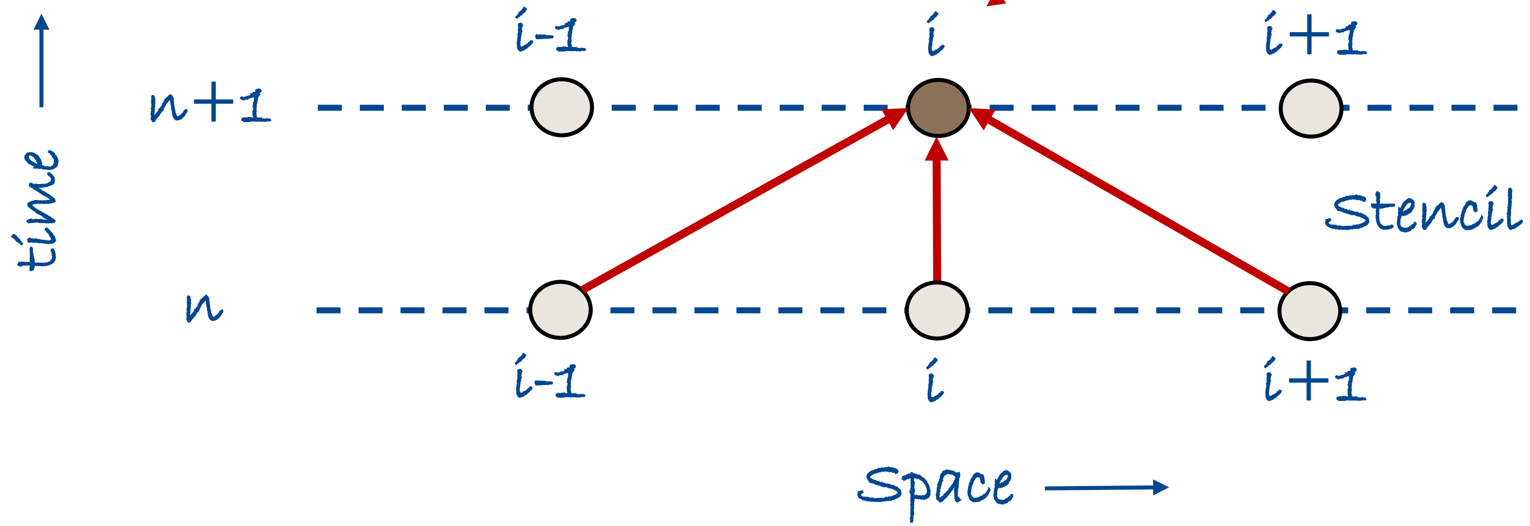
Our Numerical Formulation:

$$u_i^{n+1} = u_i^n + \frac{\nu\Delta t}{\Delta x^2}(u_{i-1}^n - 2u_i^n + u_{i-1}^n)$$

"Explicit Formulation"

Our numerical scheme does not receive information from current time !

Boundary Conditions are lagging!



1D Diffusion (Caution!)

Parabolic PDE

Characteristic lines are lines of constant 't'

$$\frac{du}{dt} = \nu \frac{d^2u}{dx^2}$$

All information at a given time should affect the solution

Our Numerical Formulation:

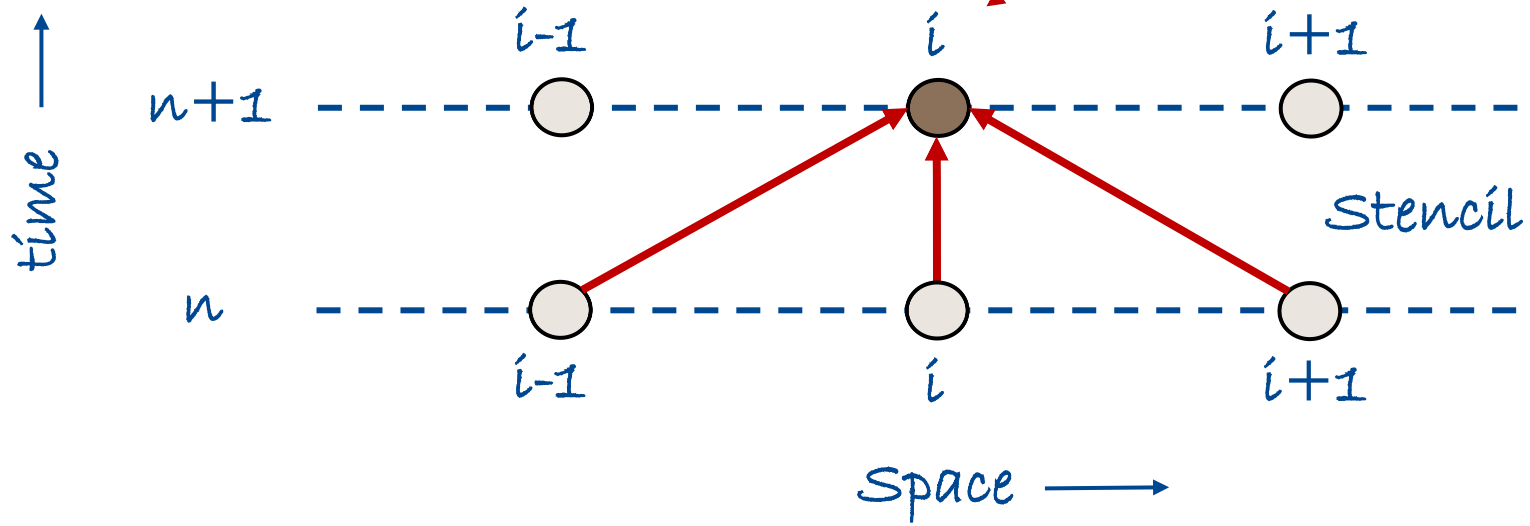
Our numerical scheme does not receive information from current time !

Boundary Conditions are lagging!

$$u_i^{n+1} = u_i^n + \frac{\nu \Delta t}{\Delta x^2}(u_{i-1}^n - 2u_i^n + u_{i-1}^n)$$

Problem if Boundary Conditions are time-dependent!

"Explicit Formulation"



1D Diffusion (Caution!)

Parabolic PDE

$$\frac{du}{dt} = \nu \frac{d^2u}{dx^2}$$

Characteristic lines are lines of constant 't'

All information at a given time should affect the solution

Our Numerical Formulation:

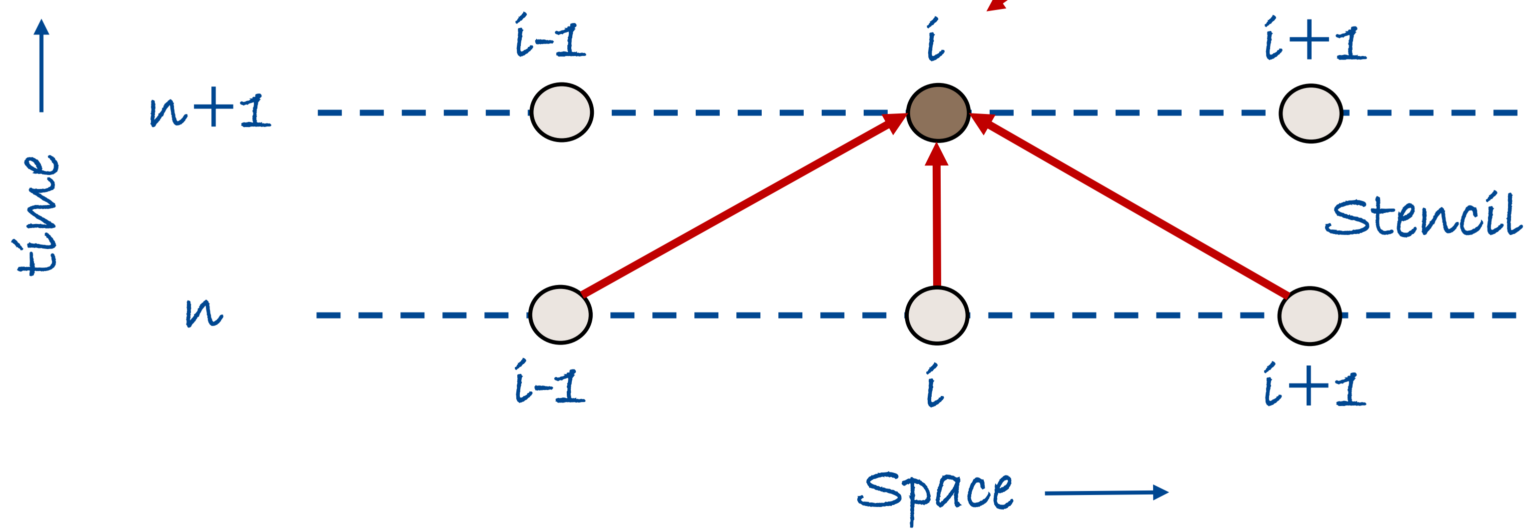
$$u_i^{n+1} = u_i^n + \frac{\nu \Delta t}{\Delta x^2}(u_{i-1}^n - 2u_i^n + u_{i-1}^n)$$

"Explicit Formulation"

Our numerical scheme does not receive information from current time !

Boundary Conditions are lagging!

Problem if Boundary Conditions are time-dependent!



How can we include BCs at current time?

1D Diffusion (Caution!)

Parabolic PDE

Characteristic lines are lines of constant 't'

All information at a given time should affect the solution

$$\frac{du}{dt} = \nu \frac{d^2u}{dx^2}$$

Our Numerical Formulation:

$$u_i^{n+1} = u_i^n + \frac{\nu \Delta t}{\Delta x^2}\left(u_{i-1}^n - 2u_i^n + u_{i-1}^n\right)$$

"Explicit Formulation"

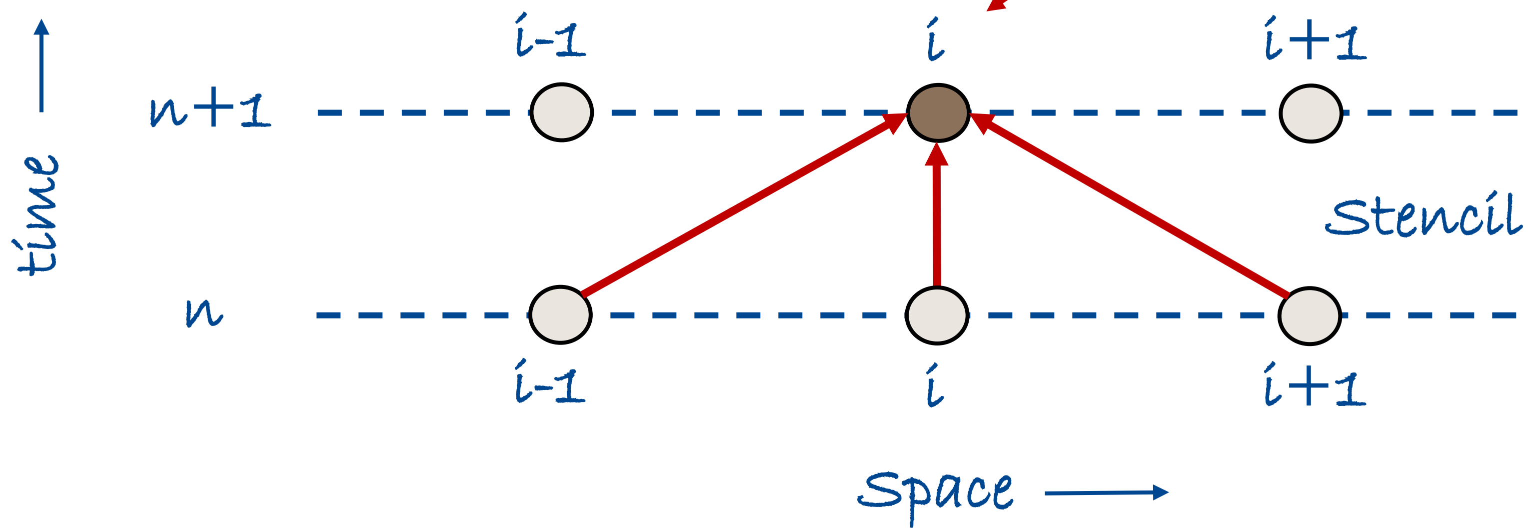
Our numerical scheme does not receive information from current time !

Boundary Conditions are lagging!

Problem if Boundary Conditions are time-dependent!

How can we include BCs at current time?

"Implicit Formulation"



1D Diffusion (Implicit Formulation)

Parabolic PDE

$$\frac{du}{dt} = \nu \frac{d^2u}{dx^2} \qquad \frac{u_i^{n+1} - u_i^n}{\Delta t} = \frac{\nu}{\Delta x^2}\left(u_{i-1}^n - 2u_i^n + u_{i-1}^n\right)$$

1D Diffusion (Implicit Formulation)

Parabolic PDE

$$\frac{du}{dt} = \nu \frac{d^2u}{dx^2} \qquad \frac{u_i^{n+1} - u_i^n}{\Delta t} = \frac{\nu}{\Delta x^2}\left(u_{i-1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}\right)$$

1D Diffusion (Implicit Formulation)

Parabolic PDE

$$\frac{du}{dt} = \nu \frac{d^2u}{dx^2} \qquad \frac{u_i^{n+1} - u_i^n}{\Delta t} = \frac{\nu}{\Delta x^2}\left(u_{i-1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}\right)$$

Rearrange:

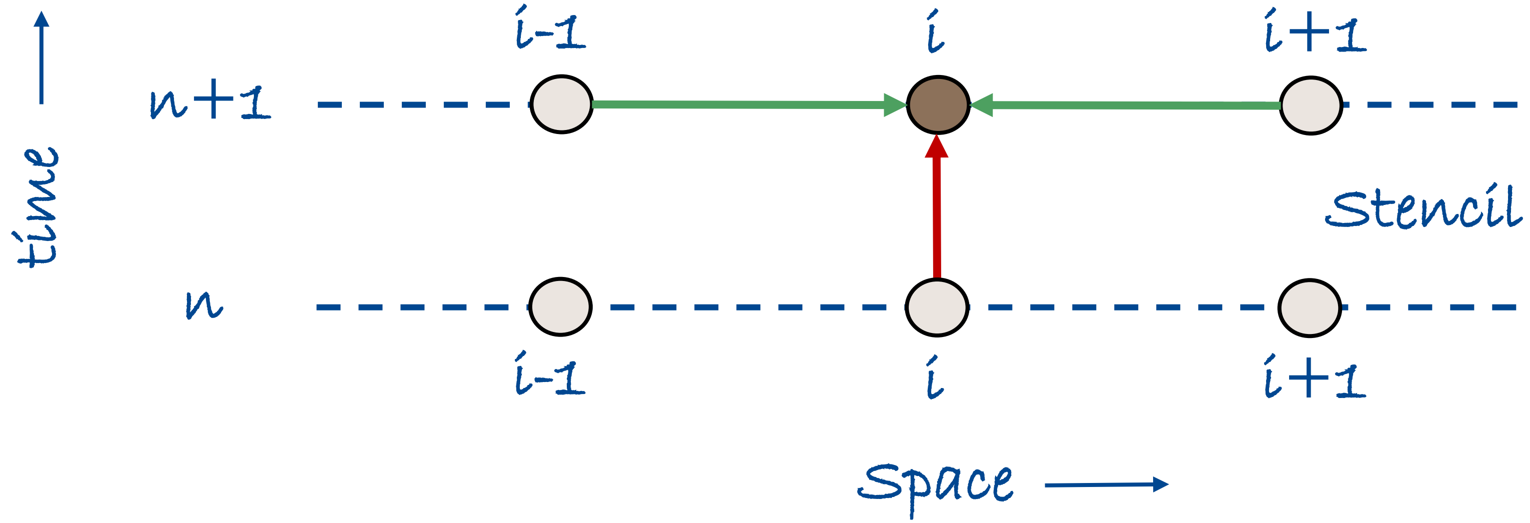
$$\frac{\nu\Delta t}{\Delta x^2} u_{i-1}^{n+1} - \left(1 + \frac{2\nu\Delta t}{\Delta x^2}\right) u_i^{n+1} - \frac{\nu\Delta t}{\Delta x^2} u_{i+1}^{n+1} = -u_i^n$$

1D Diffusion (Implicit Formulation)

Parabolic PDE

$$\frac{du}{dt} = \nu \frac{d^2u}{dx^2} \qquad \frac{u_i^{n+1} - u_i^n}{\Delta t} = \frac{\nu}{\Delta x^2}\left(u_{i-1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}\right)$$

Rearrange: $$\frac{\nu\Delta t}{\Delta x^2} u_{i-1}^{n+1} - \left(1 + \frac{2\nu\Delta t}{\Delta x^2}\right) u_i^{n+1} - \frac{\nu\Delta t}{\Delta x^2} u_{i+1}^{n+1} = -u_i^n$$

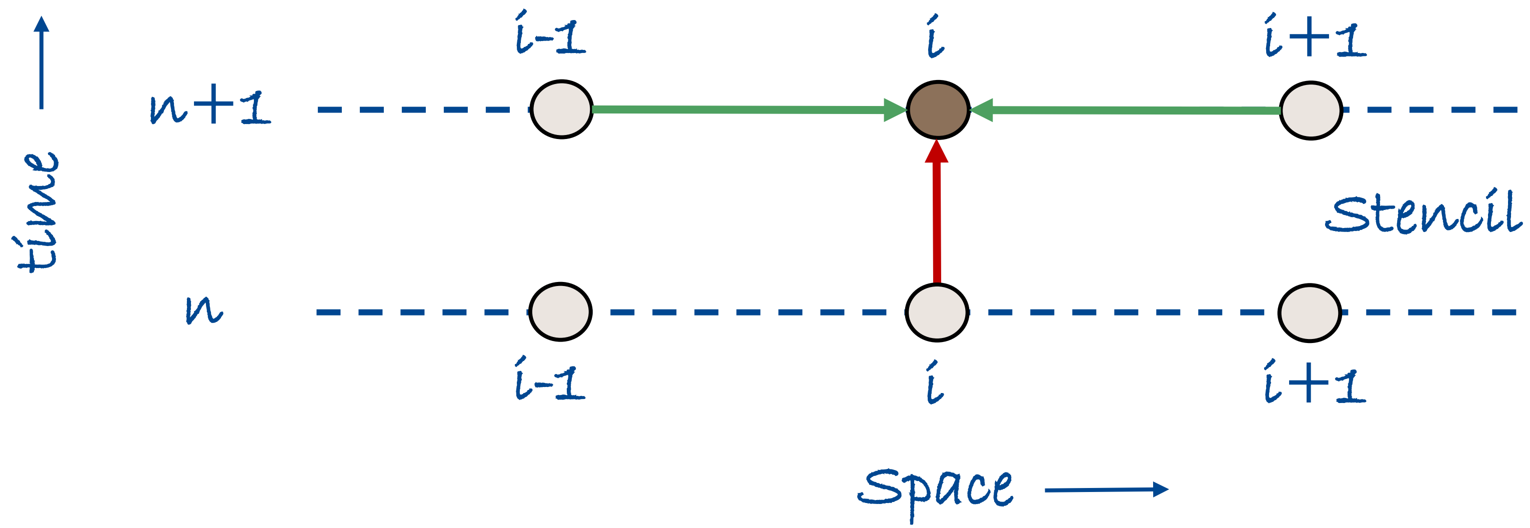


1D Diffusion (Implicit Formulation)

Parabolic PDE

$$\frac{du}{dt} = \nu \frac{d^2u}{dx^2} \qquad \frac{u_i^{n+1} - u_i^n}{\Delta t} = \frac{\nu}{\Delta x^2}\left(u_{i-1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}\right)$$

Rearrange: $$\frac{\nu\Delta t}{\Delta x^2} u_{i-1}^{n+1} - \left(1 + \frac{2\nu\Delta t}{\Delta x^2}\right) u_i^{n+1} - \frac{\nu\Delta t}{\Delta x^2} u_{i+1}^{n+1} = -u_i^n$$ "Implicit Formulation"



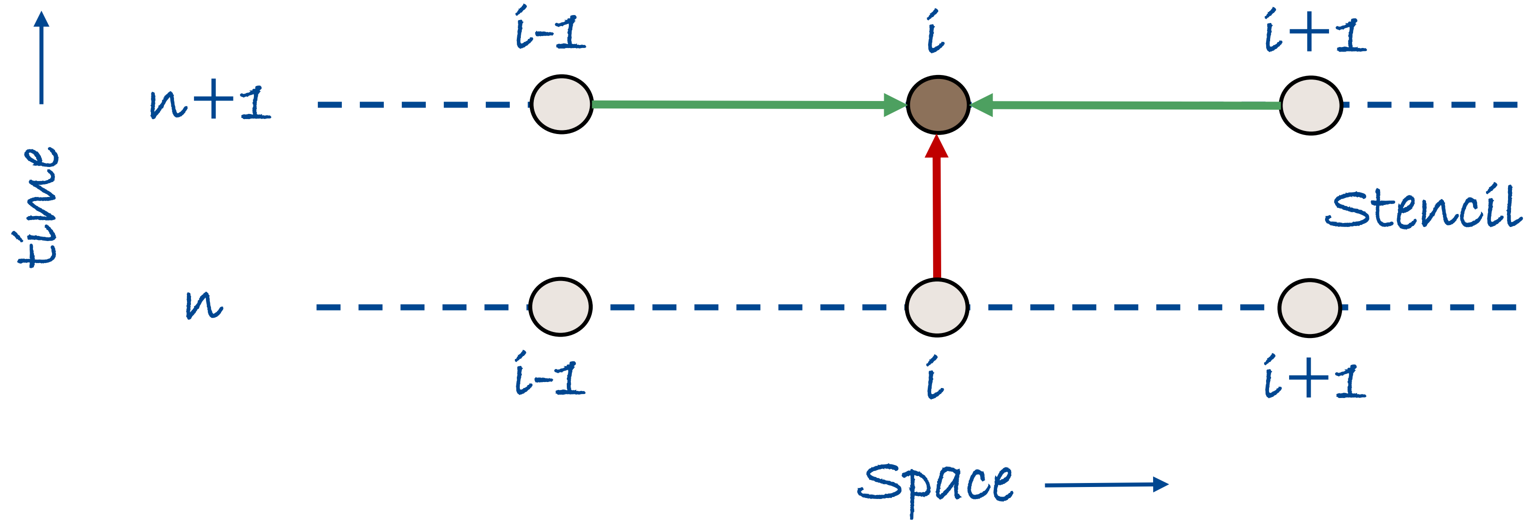
1D Diffusion (Implicit Formulation)

Parabolic PDE

$$\frac{du}{dt} = \nu \frac{d^2u}{dx^2} \qquad \frac{u_i^{n+1} - u_i^n}{\Delta t} = \frac{\nu}{\Delta x^2}\left(u_{i-1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}\right)$$

Rearrange: $$\frac{\nu\Delta t}{\Delta x^2} u_{i-1}^{n+1} - \left(1 + \frac{2\nu\Delta t}{\Delta x^2}\right) u_i^{n+1} - \frac{\nu\Delta t}{\Delta x^2} u_{i+1}^{n+1} = -u_i^n$$ "Implicit Formulation"

Need Algorithms to solve such a sparse matrix system!

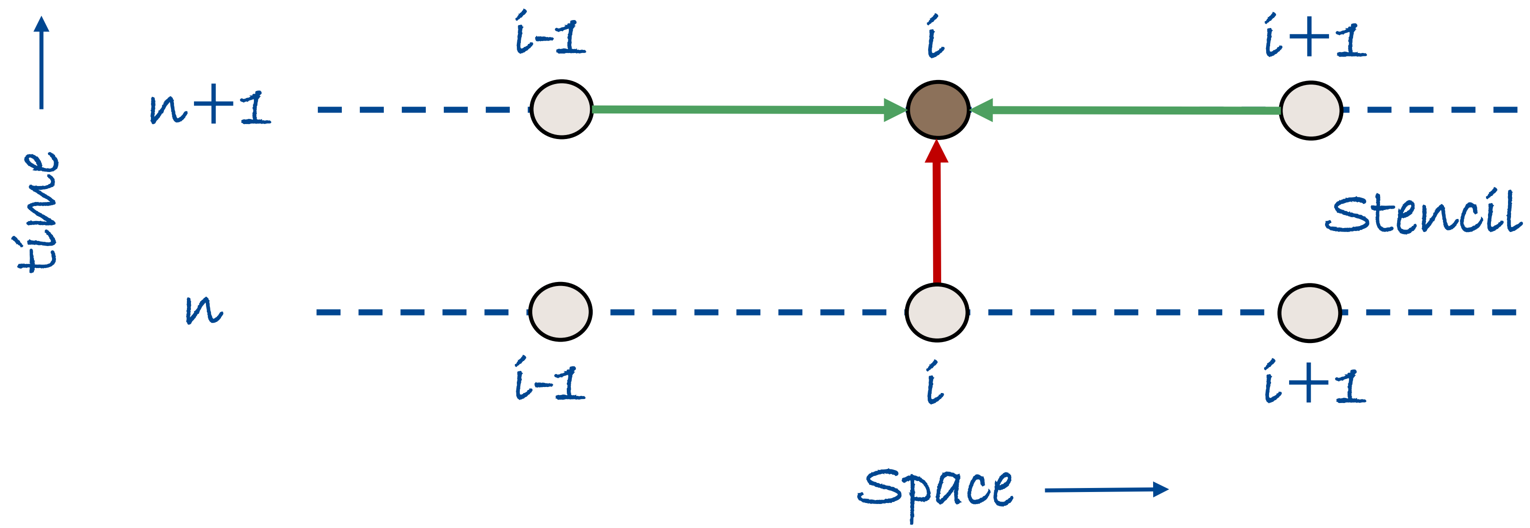


1D Diffusion (Implicit Formulation)

Parabolic PDE

$$\frac{du}{dt} = \nu \frac{d^2u}{dx^2} \qquad \frac{u_i^{n+1} - u_i^n}{\Delta t} = \frac{\nu}{\Delta x^2}\left(u_{i-1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}\right)$$

Rearrange: $$\frac{\nu \Delta t}{\Delta x^2} u_{i-1}^{n+1} - \left(1 + \frac{2\nu\Delta t}{\Delta x^2}\right) u_i^{n+1} - \frac{\nu\Delta t}{\Delta x^2} u_{i+1}^{n+1} = -u_i^n$$ "Implicit Formulation"



Need Algorithms to solve such a sparse matrix system!

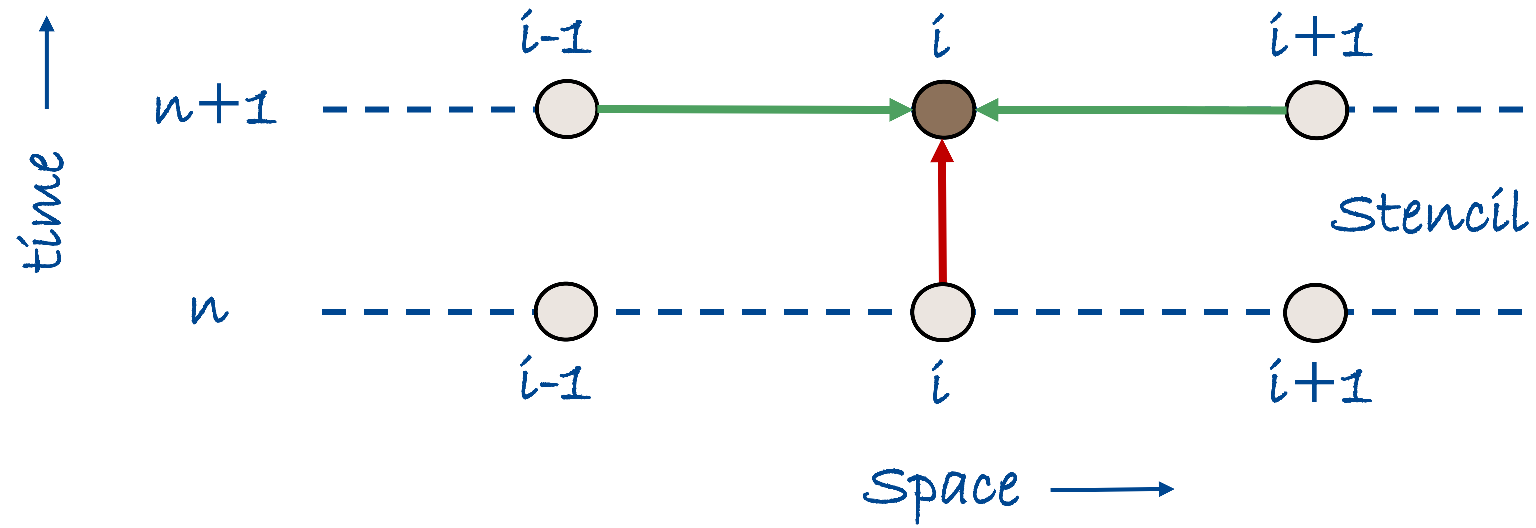
Direct or Iterative Methods

1D Diffusion (Implicit Formulation)

Parabolic PDE

$$\frac{du}{dt} = \nu \frac{d^2u}{dx^2} \qquad \frac{u_i^{n+1} - u_i^n}{\Delta t} = \frac{\nu}{\Delta x^2}\left(u_{i-1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}\right)$$

Rearrange: $$\frac{\nu\Delta t}{\Delta x^2} u_{i-1}^{n+1} - \left(1 + \frac{2\nu\Delta t}{\Delta x^2}\right) u_i^{n+1} - \frac{\nu\Delta t}{\Delta x^2} u_{i+1}^{n+1} = -u_i^n$$ "Implicit Formulation"



Need Algorithms to solve such a sparse matrix system!

Direct or Iterative Methods

Can be Expensive!

Crank-Nicholson Method

Well-known for Parabolic PDEs

$$\frac{1}{2}\left[\text{"Implicit Formulation"} \quad + \quad \text{"Explicit Formulation"}\right]$$

Crank-Nicholson Method

Well-known for Parabolic PDEs

$$\frac{1}{2}[\text{“Implicit Formulation”} \quad + \quad \text{“Explicit Formulation”}]$$

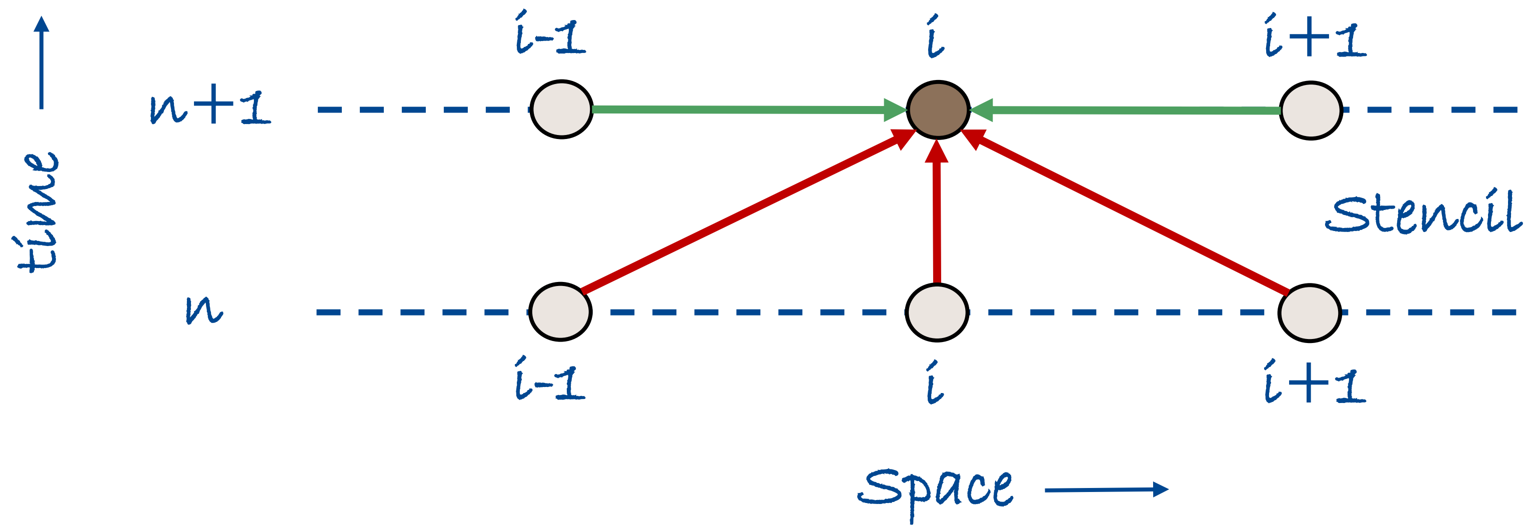
$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = \frac{1}{2}\left[\frac{\nu}{\Delta x^2}\left(u_{i-1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}\right) + \frac{\nu}{\Delta x^2}\left(u_{i-1}^{n} - 2u_i^{n} + u_{i-1}^{n}\right)\right]$$

Crank-Nicholson Method

Well-known for Parabolic PDEs

$$\frac{1}{2}\left[\text{“Implicit Formulation”} \quad + \quad \text{“Explicit Formulation”}\right]$$

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = \frac{1}{2}\left[\frac{\nu}{\Delta x^2}\left(u_{i-1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}\right) + \frac{\nu}{\Delta x^2}\left(u_{i-1}^{n} - 2u_i^{n} + u_{i-1}^{n}\right)\right]$$

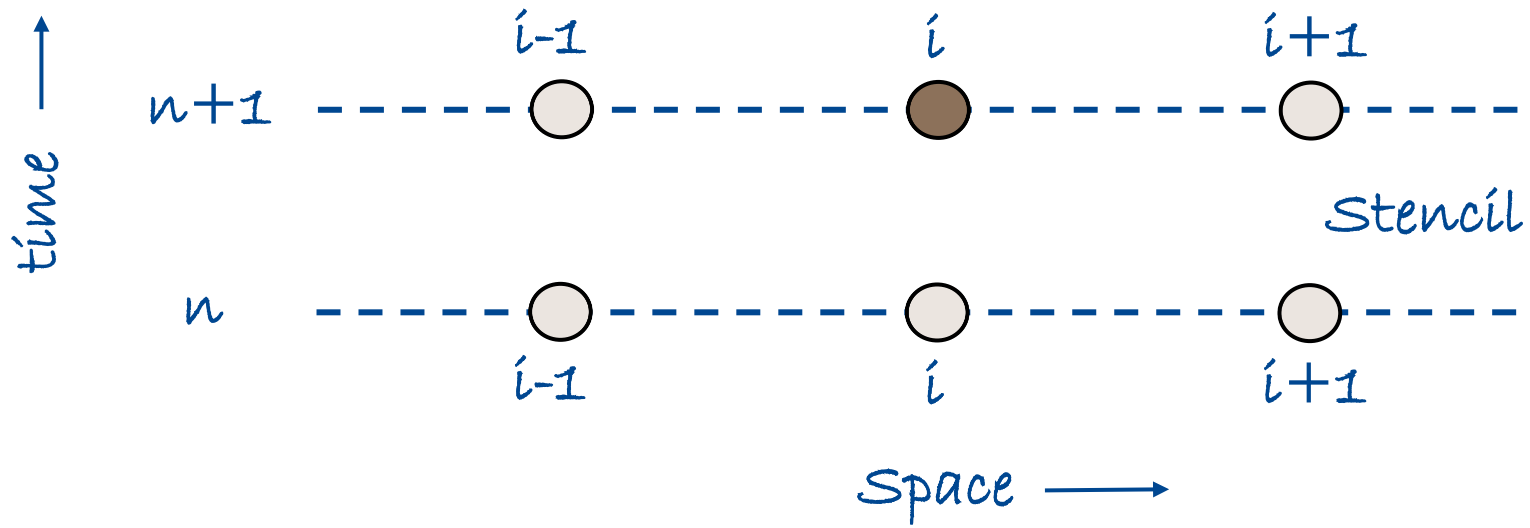


Crank-Nicholson Method

Well-known for Parabolic PDEs

$$\frac{1}{2}[\text{“Implicit Formulation”} \quad + \quad \text{“Explicit Formulation”}]$$

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = \frac{1}{2}\left[\frac{\nu}{\Delta x^2}\left(u_{i-1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}\right) + \frac{\nu}{\Delta x^2}\left(u_{i-1}^{n} - 2u_i^{n} + u_{i-1}^{n}\right)\right]$$

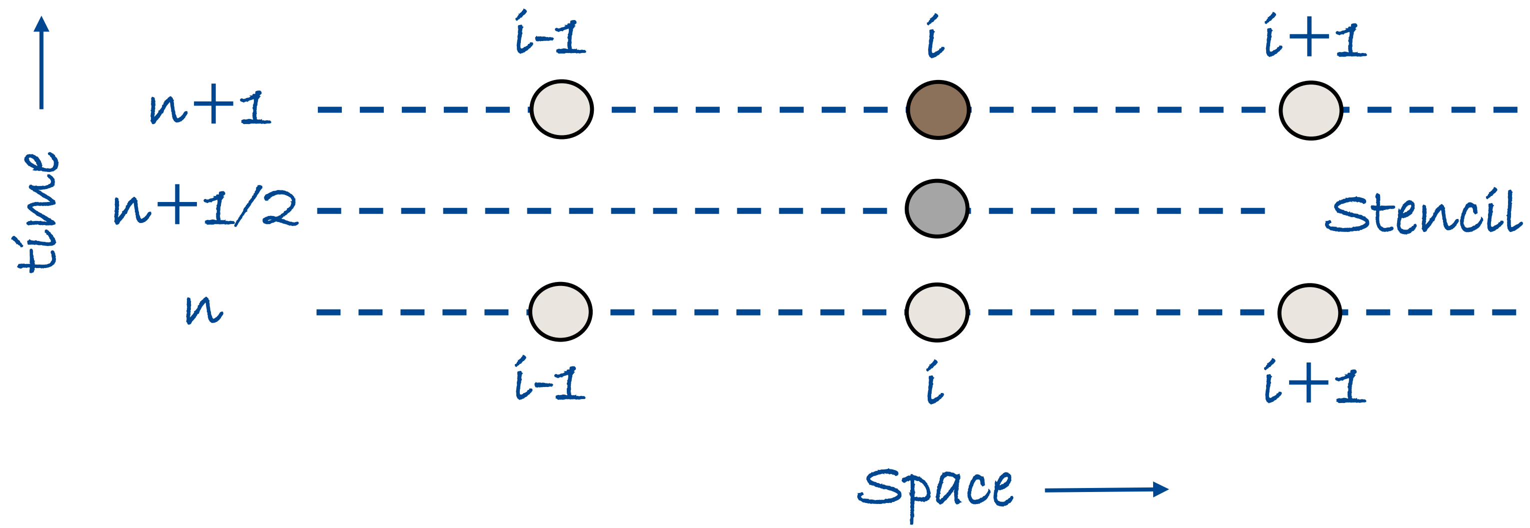


Crank-Nicholson Method

Well-known for Parabolic PDEs

$$\frac{1}{2}\left[\text{“Implicit Formulation”} \quad + \quad \text{“Explicit Formulation”}\right]$$

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = \frac{1}{2}\left[\frac{\nu}{\Delta x^2}\left(u_{i-1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}\right) + \frac{\nu}{\Delta x^2}\left(u_{i-1}^{n} - 2u_i^{n} + u_{i-1}^{n}\right)\right]$$

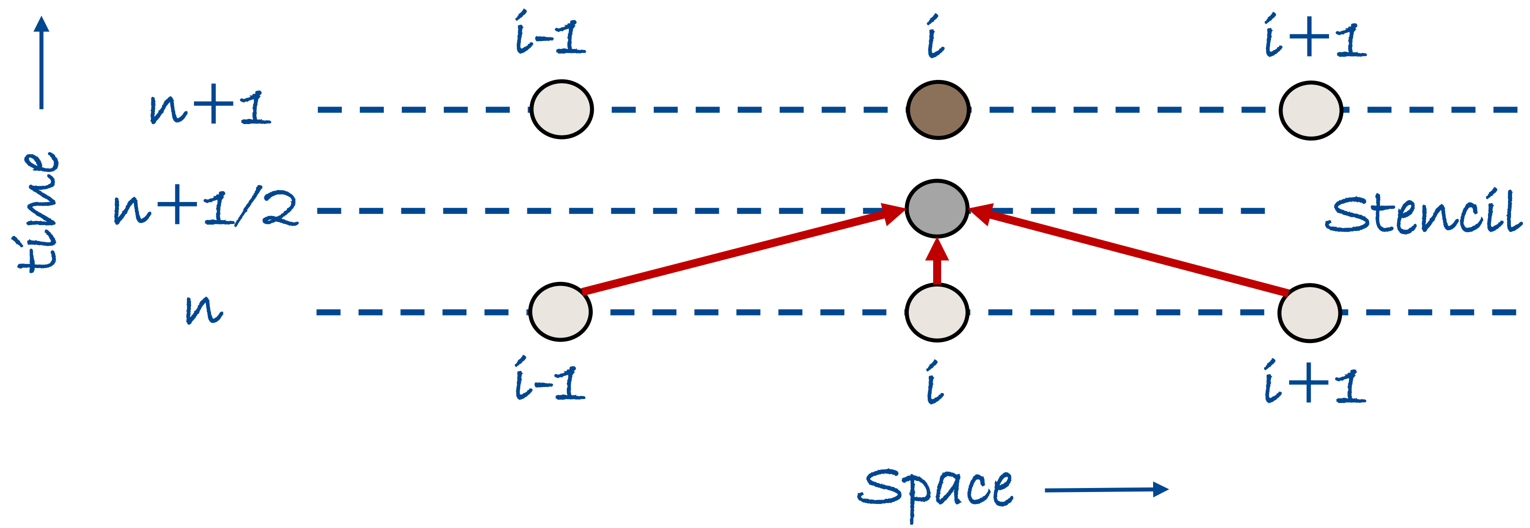


Crank-Nicholson Method

Well-known for Parabolic PDEs

$$\frac{1}{2}\,[\text{“Implicit Formulation”} \quad + \quad \text{“Explicit Formulation”}]$$

$$\frac{u_i^{n+1} - u_i^{n}}{\Delta t} = \frac{1}{2}\left[\frac{\nu}{\Delta x^2}\left(u_{i-1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}\right) + \frac{\nu}{\Delta x^2}\left(u_{i-1}^{n} - 2u_i^{n} + u_{i-1}^{n}\right)\right]$$



Explicit:

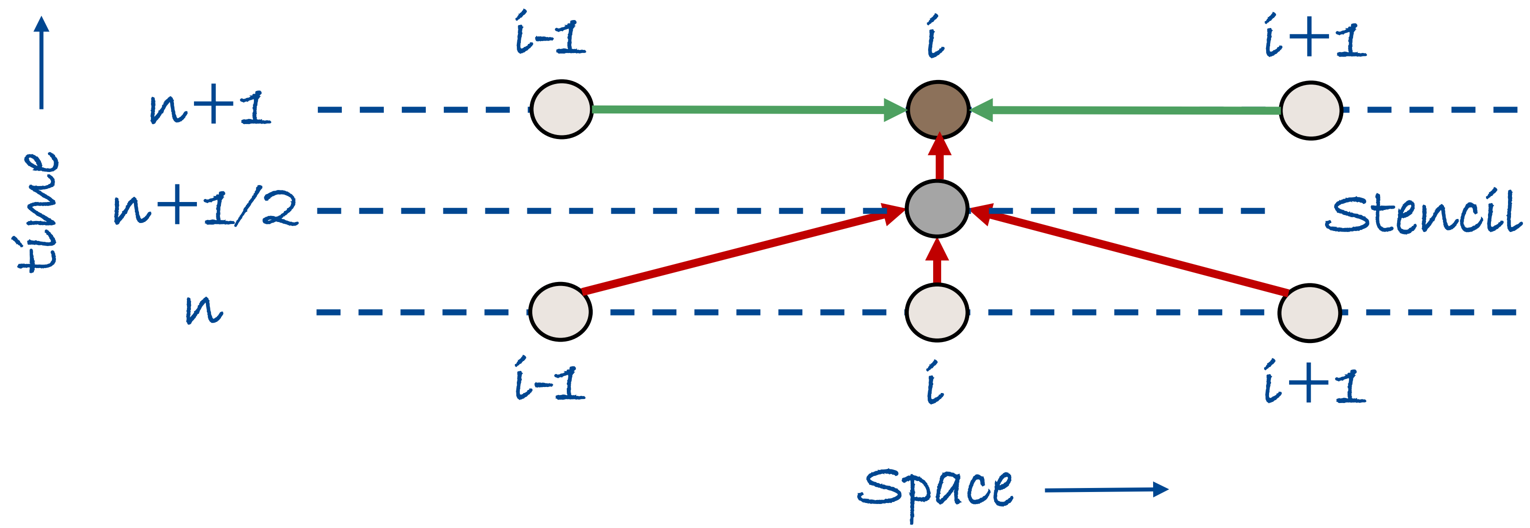
$$\frac{u_i^{n+1/2} - u_i^{n}}{\Delta t/2} = \frac{\nu}{\Delta x^2}\left(u_{i-1}^{n} - 2u_i^{n} + u_{i-1}^{n}\right)$$

Crank-Nicholson Method

Well-known for Parabolic PDEs

$$\frac{1}{2}[\text{"Implicit Formulation"} + \text{"Explicit Formulation"}]$$

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = \frac{1}{2}\left[\frac{\nu}{\Delta x^2}\left(u_{i-1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}\right) + \frac{\nu}{\Delta x^2}\left(u_{i-1}^{n} - 2u_i^{n} + u_{i-1}^{n}\right)\right]$$



Implicit:

$$\frac{u_i^{n+1} - u_i^{n+1/2}}{\Delta t/2} = \frac{\nu}{\Delta x^2}\left(u_{i-1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}\right)$$

Explicit:

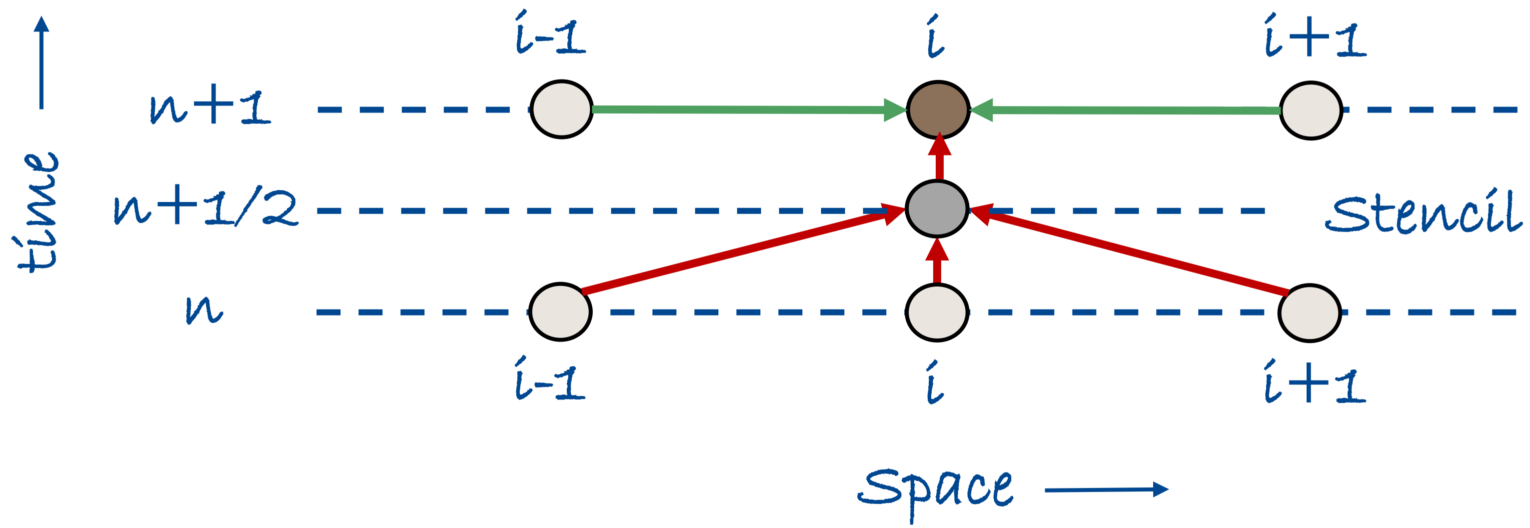
$$\frac{u_i^{n+1/2} - u_i^{n}}{\Delta t/2} = \frac{\nu}{\Delta x^2}\left(u_{i-1}^{n} - 2u_i^{n} + u_{i-1}^{n}\right)$$

Crank-Nicholson Method

Well-known for Parabolic PDEs

$$\frac{1}{2}\left[\text{"Implicit Formulation"} \quad + \quad \text{"Explicit Formulation"}\right]$$

$$\frac{u_i^{n+1} - u_i^{n}}{\Delta t} = \frac{1}{2}\left[\frac{\nu}{\Delta x^2}\left(u_{i-1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}\right) + \frac{\nu}{\Delta x^2}\left(u_{i-1}^{n} - 2u_i^{n} + u_{i-1}^{n}\right)\right]$$



Implicit:

$$\frac{u_i^{n+1} - u_i^{n+1/2}}{\Delta t/2} = \frac{\nu}{\Delta x^2}\left(u_{i-1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}\right)$$

Explicit:

$$\frac{u_i^{n+1/2} - u_i^{n}}{\Delta t/2} = \frac{\nu}{\Delta x^2}\left(u_{i-1}^{n} - 2u_i^{n} + u_{i-1}^{n}\right)$$

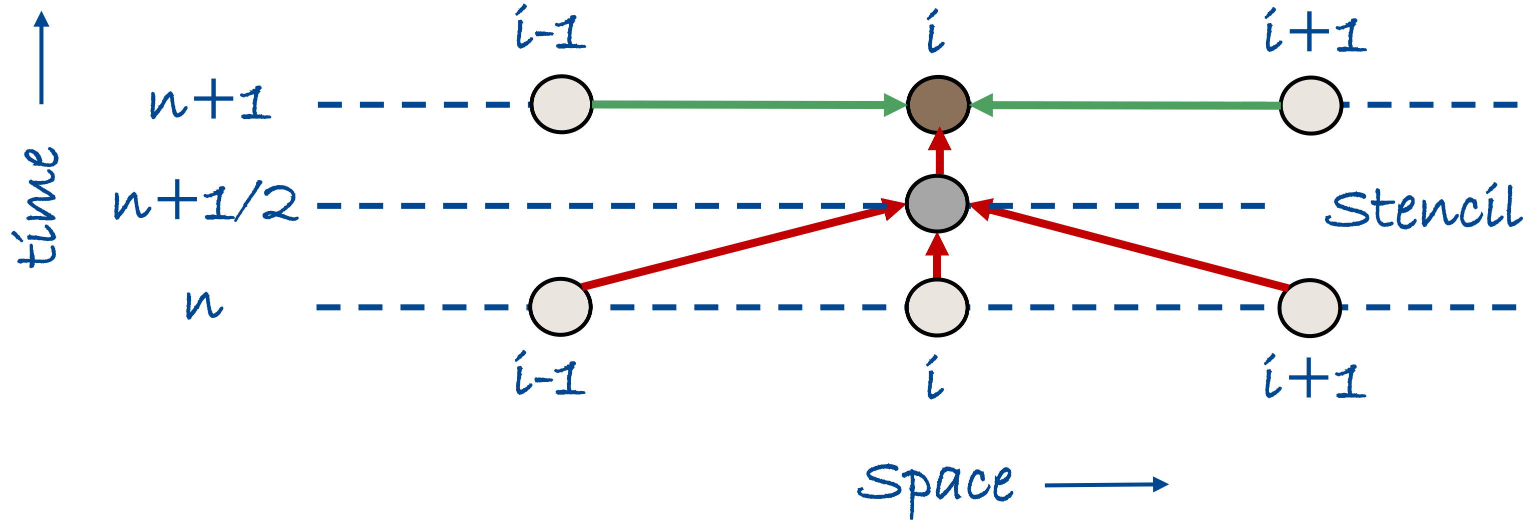
✓ 2nd order in time & space !

Crank-Nicholson Method

Well-known for Parabolic PDEs

$$\frac{1}{2}\left[\text{“Implicit Formulation”} + \text{“Explicit Formulation”}\right]$$

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = \frac{1}{2}\left[\frac{\nu}{\Delta x^2}\left(u_{i-1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}\right) + \frac{\nu}{\Delta x^2}\left(u_{i-1}^{n} - 2u_i^{n} + u_{i-1}^{n}\right)\right]$$



Implicit:

$$\frac{u_i^{n+1} - u_i^{n+1/2}}{\Delta t/2} = \frac{\nu}{\Delta x^2}\left(u_{i-1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}\right)$$

Explicit:

$$\frac{u_i^{n+1/2} - u_i^{n}}{\Delta t/2} = \frac{\nu}{\Delta x^2}\left(u_{i-1}^{n} - 2u_i^{n} + u_{i-1}^{n}\right)$$

✔ 2nd order in time & space !

⚠ Tridiagonal system to solve!

4th Code: 1D Burger's Equation

$$\frac{du}{dt} + u\frac{du}{dx} = \nu\frac{d^2u}{dx^2}$$

4th Code: 1D Burger's Equation

$$\frac{du}{dt} + u\frac{du}{dx} = \nu\frac{d^2u}{dx^2}$$

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} + u_i^n\frac{u_i^n - u_{i-1}^n}{\Delta x} = \nu\frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2}$$

4th Code: 1D Burger's Equation

$$\frac{du}{dt} + u\frac{du}{dx} = \nu\frac{d^2u}{dx^2}$$

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} + u_i^n\frac{u_i^n - u_{i-1}^n}{\Delta x} = \nu\frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2}$$

$$\boxed{u_i^{n+1} = u_i^n - \frac{c\Delta t}{\Delta \mathrm{x}}(u_i^n - u_{i-1}^n) + \frac{c\Delta t}{\Delta \mathrm{x}^2}(u_{i-1}^n - 2u_i^n + u_{i-1}^n)}$$

4^{th} Code: 1D Burger's Equation

$$\frac{du}{dt} + u\frac{du}{dx} = \nu\frac{d^2u}{dx^2}$$

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} + u_i^n\frac{u_i^n - u_{i-1}^n}{\Delta x} = \nu\frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2}$$

$$\boxed{u_i^{n+1} = u_i^n - \frac{c\Delta t}{\Delta x}(u_i^n - u_{i-1}^n) + \frac{c\Delta t}{\Delta x^2}(u_{i-1}^n - 2u_i^n + u_{i-1}^n)}$$

Initial Condition: $u(x,0) = -\frac{2\nu}{\phi}\frac{\partial\phi}{\partial x} + 4 \quad \phi = \exp\left(-\frac{x^2}{4\nu}\right) + \exp\left(-\frac{(x-2\pi)^2}{4\nu}\right)$ Boundary Condition: $u(0) = u(2\pi)$

Periodic BC!

Exact Solution: $u = -\frac{2\nu}{\phi}\frac{\partial\phi}{\partial x} + 4 \qquad \phi = \exp\left(-\frac{(x-4t)^2}{4\nu(t+1)}\right) + \exp\left(-\frac{(x-4t-2\pi)^2}{4\nu(t+1)}\right)$

4^{th} Code: 1D Burger's Equation

MODELAIR

$$\frac{du}{dt} + u\frac{du}{dx} = \nu\frac{d^2u}{dx^2}$$

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} + u_i^n\frac{u_i^n - u_{i-1}^n}{\Delta x} = \nu\frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2}$$

$$\boxed{u_i^{n+1} = u_i^n - \frac{c\Delta t}{\Delta \mathrm{x}}(u_i^n - u_{i-1}^n) + \frac{c\Delta t}{\Delta \mathrm{x}^2}(u_{i-1}^n - 2u_i^n + u_{i-1}^n)}$$

Initial Condition: $u(x,0) = -\frac{2\nu}{\phi}\frac{\partial\phi}{\partial x} + 4$ $\quad \phi = \exp\left(-\frac{x^2}{4\nu}\right) + \exp\left(-\frac{(\mathrm{x} - 2\pi)^2}{4\nu}\right)$ Boundary Condition: $u(0) = u(2\pi)$

Periodic BC!

Exact Solution: $u = -\frac{2\nu}{\phi}\frac{\partial\phi}{\partial x} + 4$ $\quad \phi = \exp\left(-\frac{(\mathrm{x} - 4\mathrm{t})^2}{4\nu(t+1)}\right) + \exp\left(-\frac{(\mathrm{x} - 4\mathrm{t} - 2\pi)^2}{4\nu(t+1)}\right)$

Let's Code It!

4th Code: 1D Burger's Equation

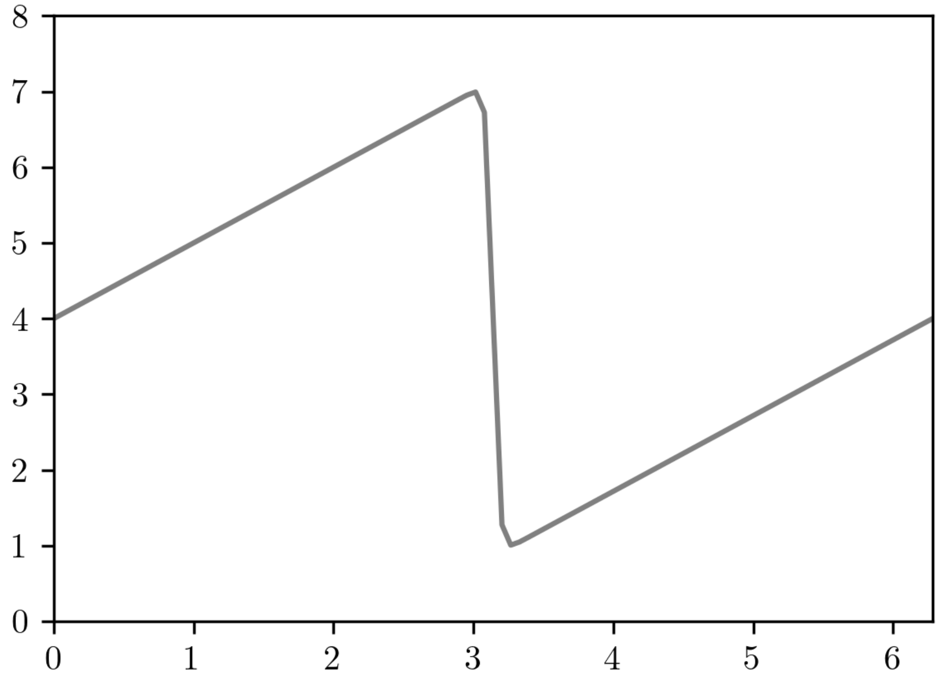
```
import numpy as np
import sympy as sp
import pylab as pl
pl.ion()
```

4th Code: 1D Burger's Equation

```
import numpy as np
import sympy as sp
import pylab as pl
pl.ion()

x, nu, t = sp.symbols('x nu t')
phi = sp.exp(-(x-4*t)**2/(4*nu*(t+1))) + sp.exp(-(x-4*t-2*sp.pi)**2/(4*nu*(t+1)))

phiprime = phi.diff(x)
u = -2*nu*(phiprime/phi)+4

from sympy.utilities.lambdify import lambdify
ufunc = lambdify ((t, x, nu), u)
```

4th Code: 1D Burger's Equation

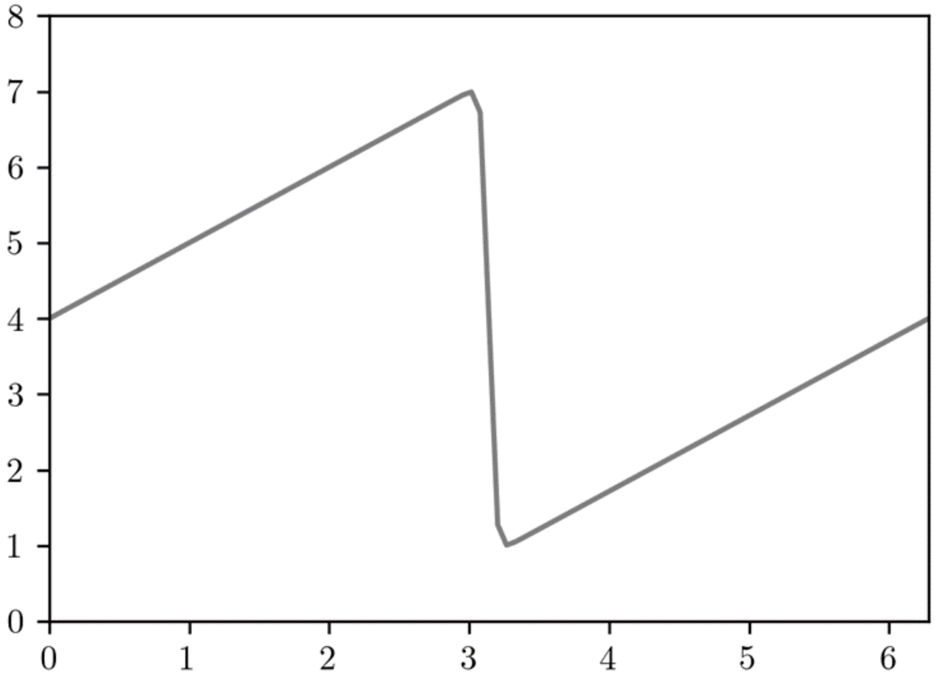
```
import numpy as np
import sympy as sp
import pylab as pl
pl.ion()

x, nu, t = sp.symbols('x nu t')
phi = sp.exp(-(x-4*t)**2/(4*nu*(t+1))) + sp.exp(-(x-4*t-2*sp.pi)**2/(4*nu*(t+1)))

phiprime = phi.diff(x)
u = -2*nu*(phiprime/phi)+4

from sympy.utilities.lambdify import lambdify
ufunc = lambdify ((t, x, nu), u)


nx = 101
dx = 2*np.pi/(nx-1)
nt = 100
nu = 0.07
dt = dx*nu
T = nt*dt

grid = np.linspace(0, 2*np.pi, nx)
u = np.empty(nx)
t = 0
u = np.asarray([ufunc(t, x, nu) for x in grid])


pl.figure(figsize=(11,7), dpi=100)
pl.plot(grid,u, marker='o', lw=2)
pl.xlim([0,2*np.pi])
pl.ylim([0,10])
pl.xlabel('X')
pl.ylabel('Velocity')
pl.title('1D Burgers Equation - Initial condition')
```



4th Code: 1D Burger's Equation

```python
import numpy as np
import sympy as sp
import pylab as pl
pl.ion()

x, nu, t = sp.symbols('x nu t')
phi = sp.exp(-(x-4*t)**2/(4*nu*(t+1))) + sp.exp(-(x-4*t-2*sp.pi)**2/(4*nu*(t+1)))

phiprime = phi.diff(x)
u = -2*nu*(phiprime/phi)+4

from sympy.utilities.lambdify import lambdify
ufunc = lambdify ((t, x, nu), u)


nx = 101
dx = 2*np.pi/(nx-1)
nt = 100
nu = 0.07
dt = dx*nu
T = nt*dt

grid = np.linspace(0, 2*np.pi, nx)
u = np.empty(nx)
t = 0
u = np.asarray([ufunc(t, x, nu) for x in grid])


pl.figure(figsize=(11,7), dpi=100)
pl.plot(grid,u, marker='o', lw=2)
pl.xlim([0,2*np.pi])
pl.ylim([0,10])
pl.xlabel('X')
pl.ylabel('Velocity')
pl.title('1D Burgers Equation - Initial condition')
```

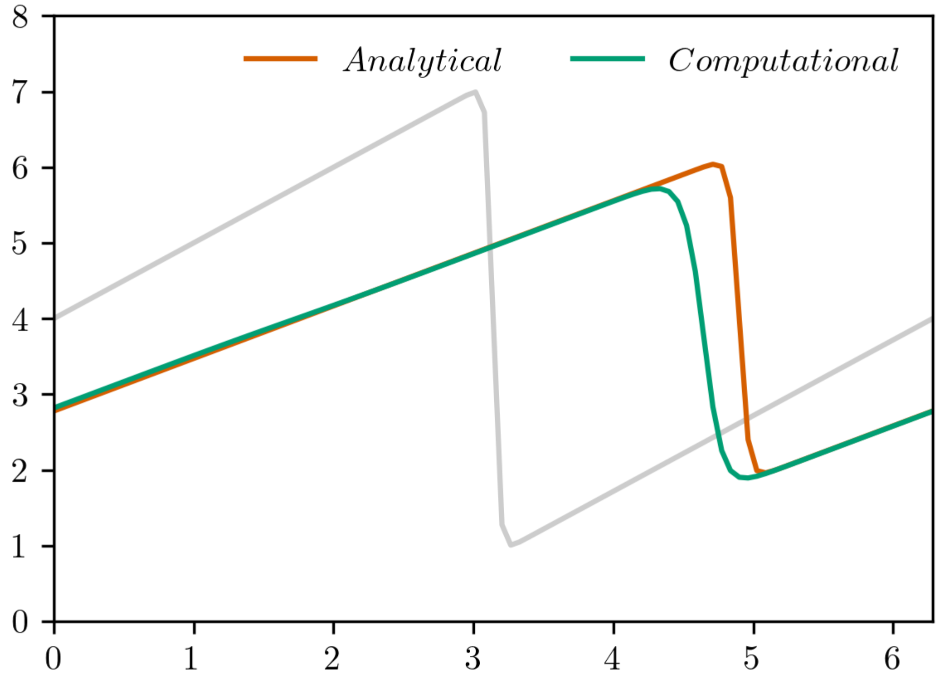
```python
for n in range(nt):
    un = u.copy()
    for i in range(nx-1):
        u[i] = un[i] - un[i] * dt/dx * (un[i]-un[i-1]) + \
            nu * dt/(dx**2) * (un[i+1] - 2*un[i] + un[i-1])
    # infer the periodicity
    u[-1] = un[-1] - un[-1] * dt/dx * (un[-1]-un[-2]) + \
            nu * dt/(dx**2) * (un[0] - 2*un[-1] + un[-2])

u_analytical = np.asarray([ufunc(T, xi, nu) for xi in grid])

pl.figure(figsize=(11,7), dpi=100)
pl.plot(grid, u, marker='o', lw=2, label='Computational')
pl.plot(grid, u_analytical, label='Analytical')
pl.xlim([0, 2*np.pi])
pl.ylim([0,10])
pl.legend()
pl.xlabel('X')
pl.ylabel('Velocity')
pl.title('1D Burgers Equation - Solutions')
```



4th Code: 1D Burger's Equation

```
import numpy as np
import sympy as sp
import pylab as pl
pl.ion()

x, nu, t = sp.symbols('x nu t')
phi = sp.exp(-(x-4*t)**2/(4*nu*(t+1))) + sp.exp(-(x-4*t-2*sp.pi)**2/(4*nu*(t+1)))

phiprime = phi.diff(x)
u = -2*nu*(phiprime/phi)+4

from sympy.utilities.lambdify import lambdify
ufunc = lambdify ((t, x, nu), u)


nx = 101
dx = 2*np.pi/(nx-1)
nt = 100
nu = 0.07
dt = dx*nu
T = nt*dt

grid = np.linspace(0, 2*np.pi, nx)
u = np.empty(nx)
t = 0
u = np.asarray([ufunc(t, x, nu) for x in grid])


pl.figure(figsize=(11,7), dpi=100)
pl.plot(grid,u, marker='o', lw=2)
pl.xlim([0,2*np.pi])
pl.ylim([0,10])
pl.xlabel('X')
pl.ylabel('Velocity')
pl.title('1D Burgers Equation - Initial condition')
```

```
for n in range(nt):
    un = u.copy()
    for i in range(nx-1):
        u[i] = un[i] - un[i] * dt/dx * (un[i]-un[i-1]) + \
            nu * dt/(dx**2) * (un[i+1] - 2*un[i] + un[i-1])
    # infer the periodicity
    u[-1] = un[-1] - un[-1] * dt/dx * (un[-1]-un[-2]) + \
            nu * dt/(dx**2) * (un[0] - 2*un[-1] + un[-2])

u_analytical = np.asarray([ufunc(T, xi, nu) for xi in grid])

pl.figure(figsize=(11,7), dpi=100)
pl.plot(grid, u, marker='o', lw=2, label='Computational')
pl.plot(grid, u_analytical, label='Analytical')
pl.xlim([0, 2*np.pi])
pl.ylim([0,10])
pl.legend()
pl.xlabel('X')
pl.ylabel('Velocity')
pl.title('1D Burgers Equation - Solutions')
```



5^{th} Code: 2D Laplace Equation

$$\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} = 0$$

5th Code: 2D Laplace Equation

$$\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} = 0$$

$$\frac{p_{i+1,j}^n - 2p_{i,j}^n + p_{i-1,j}^n}{\Delta x^2} + \frac{p_{i,j+1}^n - 2p_{i,j}^n + p_{i,j-1}^n}{\Delta y^2} = 0$$

5th Code: 2D Laplace Equation

$$\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} = 0$$

$$\frac{p_{i+1,j}^{n} - 2p_{i,j}^{n} + p_{i-1,j}^{n}}{\Delta x^2} + \frac{p_{i,j+1}^{n} - 2p_{i,j}^{n} + p_{i,j-1}^{n}}{\Delta y^2} = 0$$

$$p_{i,j}^{n} = \frac{\Delta y^2 (p_{i+1,j}^{n} + p_{i-1,j}^{n}) + \Delta x^2 (p_{i,j+1}^{n} + p_{i,j-1}^{n})}{2(\Delta x^2 + \Delta y^2)}$$

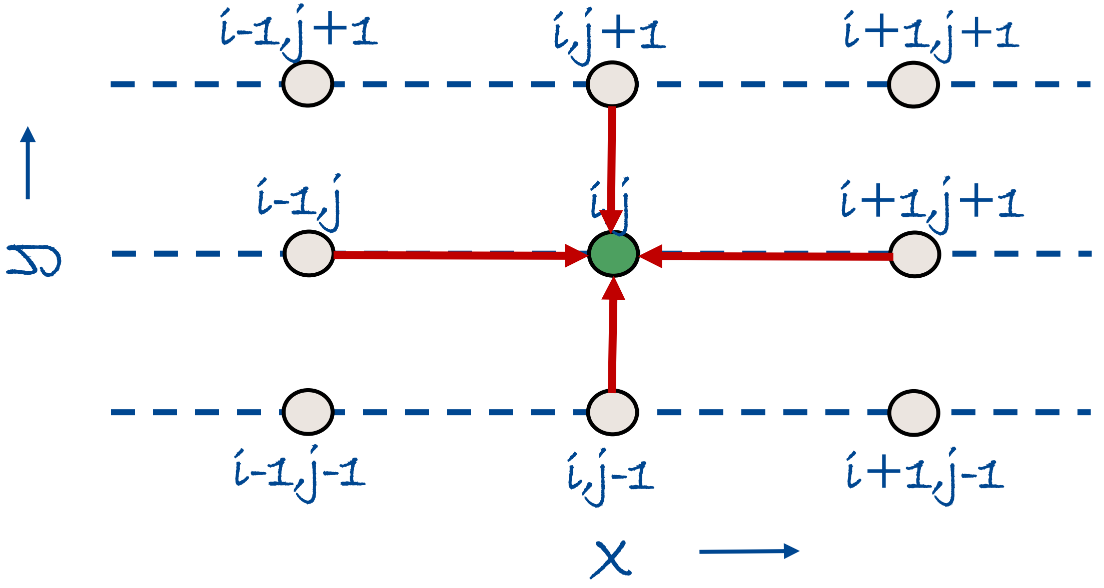
5th Code: 2D Laplace Equation

$$\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} = 0$$

$$\frac{p^{n}_{i+1,j} - 2p^{n}_{i,j} + p^{n}_{i-1,j}}{\Delta x^2} + \frac{p^{n}_{i,j+1} - 2p^{n}_{i,j} + p^{n}_{i,j-1}}{\Delta y^2} = 0$$

$$p^{n}_{i,j} = \frac{\Delta y^2 (p^{n}_{i+1,j} + p^{n}_{i-1,j}) + \Delta x^2 (p^{n}_{i,j+1} + p^{n}_{i,j-1})}{2(\Delta x^2 + \Delta y^2)}$$

5 points stencil



5^{th} Code: 2D Laplace Equation

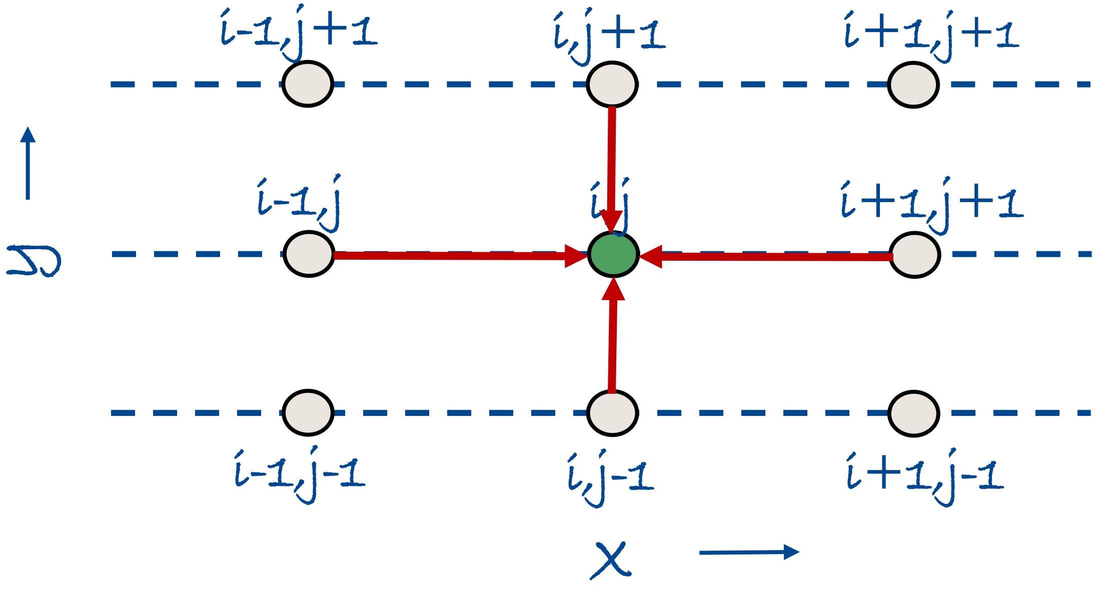
No time dependence !

$$\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} = 0$$

$$\frac{p_{i+1,j}^{n} - 2p_{i,j}^{n} + p_{i-1,j}^{n}}{\Delta x^2} + \frac{p_{i,j+1}^{n} - 2p_{i,j}^{n} + p_{i,j-1}^{n}}{\Delta y^2} = 0$$

$$p_{i,j}^{n} = \frac{\Delta y^2 (p_{i+1,j}^{n} + p_{i-1,j}^{n}) + \Delta x^2 (p_{i,j+1}^{n} + p_{i,j-1}^{n})}{2(\Delta x^2 + \Delta y^2)}$$

5 points stencil



5th Code: 2D Laplace Equation

No time dependence !

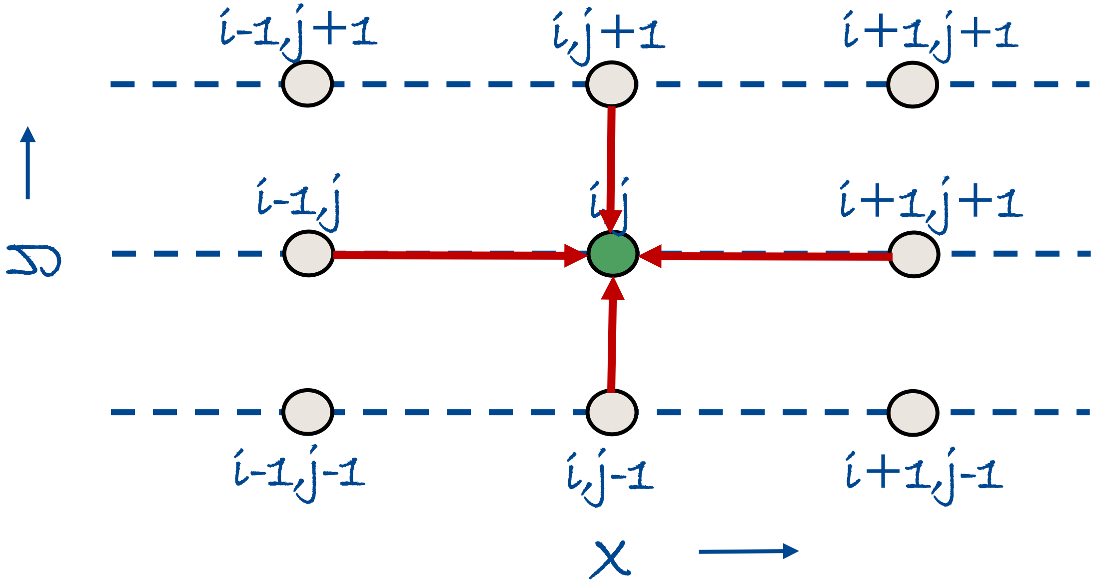
Calculates an equilibrium state given the specified BCs

$$\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} = 0$$

$$\frac{p^n_{i+1,j} - 2p^n_{i,j} + p^n_{i-1,j}}{\Delta x^2} + \frac{p^n_{i,j+1} - 2p^n_{i,j} + p^n_{i,j-1}}{\Delta y^2} = 0$$

$$p^n_{i,j} = \frac{\Delta y^2 (p^n_{i+1,j} + p^n_{i-1,j}) + \Delta x^2 (p^n_{i,j+1} + p^n_{i,j-1})}{2(\Delta x^2 + \Delta y^2)}$$

5 points stencil



5^{th} Code: 2D Laplace Equation

No time dependence !

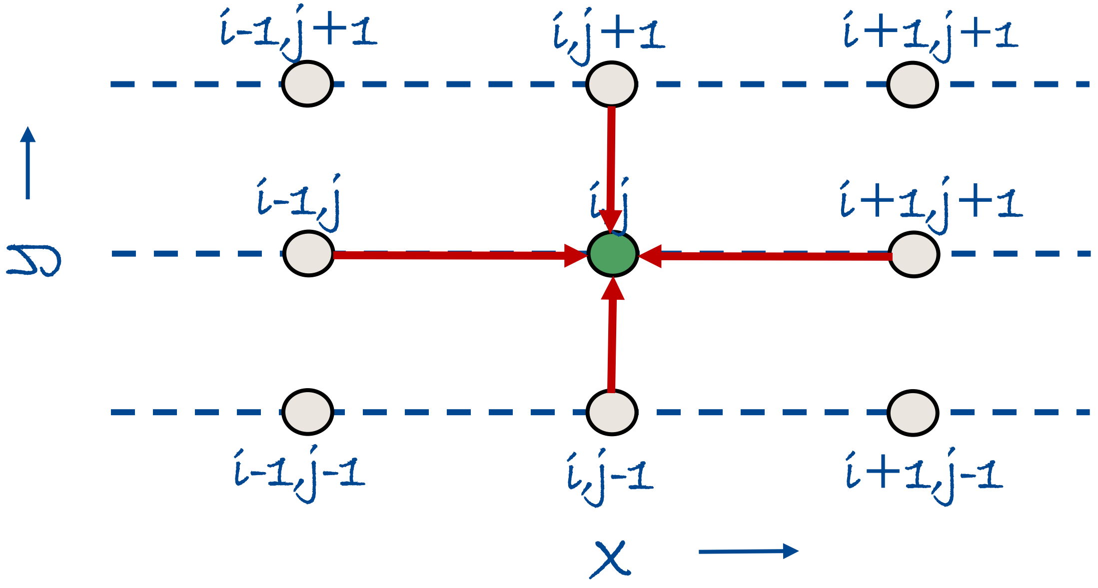
Calculates an equilibrium state given the specified BCs

$$\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} = 0$$

$$\frac{p_{i+1,j}^n - 2p_{i,j}^n + p_{i-1,j}^n}{\Delta x^2} + \frac{p_{i,j+1}^n - 2p_{i,j}^n + p_{i,j-1}^n}{\Delta y^2} = 0$$

$$p_{i,j}^n = \frac{\Delta y^2 (p_{i+1,j}^n + p_{i-1,j}^n) + \Delta x^2 (p_{i,j+1}^n + p_{i,j-1}^n)}{2(\Delta x^2 + \Delta y^2)}$$

5 points stencil



Need to solve iteratively

Will reach equilibrium as iterations $\rightarrow \infty$

Need to specify a threshold!

KTH VETENSKAP OCH KONST

5^{th} Code: 2D Laplace Equation

MODELAIR

No time dependence !

Calculates an equilibrium state given the specified BCs

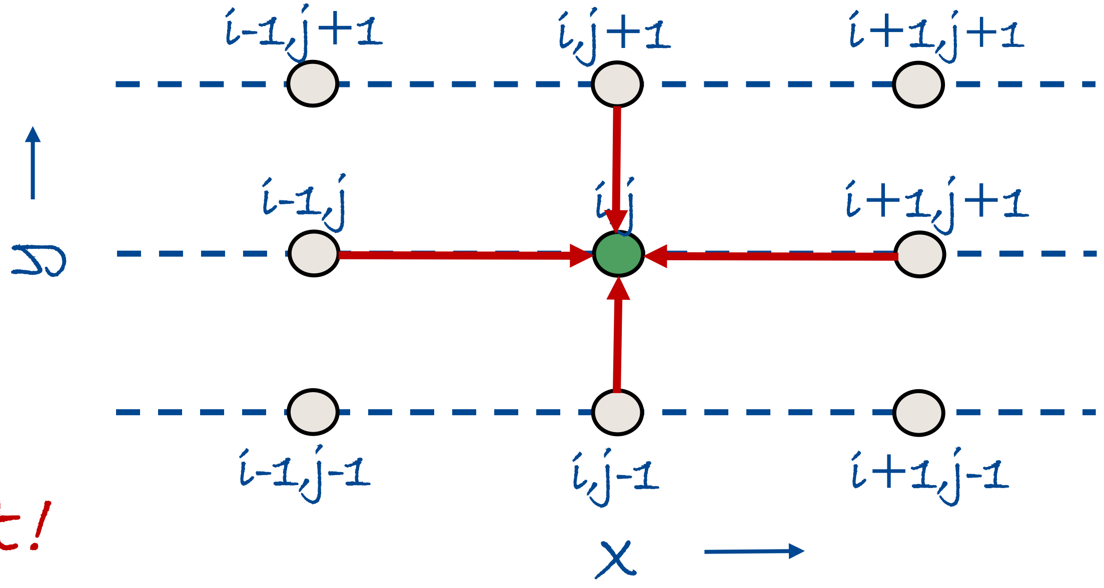
$$\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} = 0$$

BCs: $p = 0 \; at \; x = 0$, $p = y \; at \; x = 2$, $\partial p/\partial y = 0 \; at \; y = 0{,}1$

$$\frac{p_{i+1,j}^n - 2p_{i,j}^n + p_{i-1,j}^n}{\Delta x^2} + \frac{p_{i,j+1}^n - 2p_{i,j}^n + p_{i,j-1}^n}{\Delta y^2} = 0$$

$$p_{i,j}^n = \frac{\Delta y^2 (p_{i+1,j}^n + p_{i-1,j}^n) + \Delta x^2 (p_{i,j+1}^n + p_{i,j-1}^n)}{2(\Delta x^2 + \Delta y^2)}$$

5 points stencil



Need to solve iteratively

Will reach equilibrium as iterations → ∞

Need to specify a threshold!

Let's Code It!

5^{th} Code: 2D Laplace Equation

No time dependence !

Calculates an equilibrium state given the specified BCs

$$\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} = 0$$

BCs: $p = 0 \; at \; x = 0$, $p = y \; at \; x = 2$, $\partial p / \partial y = 0 \; at \; y = 0{,}1$

$$\frac{p_{i+1,j}^{n} - 2p_{i,j}^{n} + p_{i-1,j}^{n}}{\Delta x^2} + \frac{p_{i,j+1}^{n} - 2p_{i,j}^{n} + p_{i,j-1}^{n}}{\Delta y^2} = 0$$

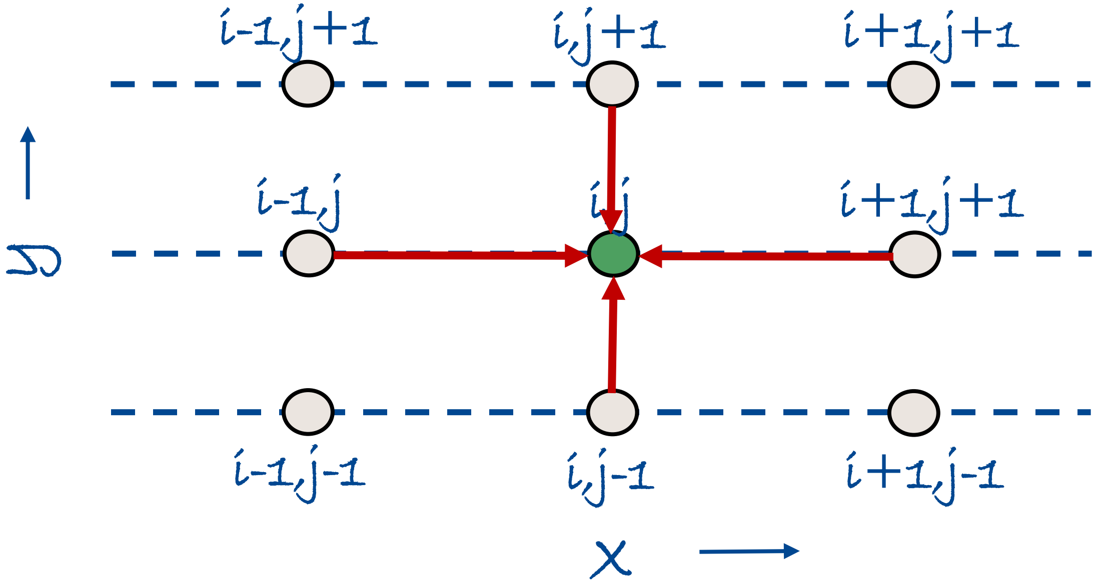
$$p_{i,j}^{n} = \frac{\Delta y^2 (p_{i+1,j}^{n} + p_{i-1,j}^{n}) + \Delta x^2 (p_{i,j+1}^{n} + p_{i,j-1}^{n})}{2(\Delta x^2 + \Delta y^2)}$$

Need to solve iteratively

Will reach equilibrium as iterations $\rightarrow \infty$

Need to specify a threshold!

5 points stencil



5th Code: 2D Laplace Equation

```
import numpy as np
from matplotlib import pyplot, cm
from mpl_toolkits.mplot3d import Axes3D
```

5th Code: 2D Laplace Equation

```
import numpy as np
from matplotlib import pyplot, cm
from mpl_toolkits.mplot3d import Axes3D

def plot2D(x,y,p):
    fig = pyplot.figure( figsize=(11,7), dpi=100 )
    ax  = fig.gca(projection='3d')
    X, Y = np.meshgrid(x,y)
    surf = ax.plot_surface( X, Y, p[:], rstride=1, cstride=1, cmap=cm.viridis,
                           linewidth=0, antialiased=False )
    ax.set_xlabel("$x$");  ax.set_xlim(0,2)
    ax.set_ylabel("$y$");  ax.set_ylim(0,1)
    ax.view_init(30,225)
```

5th Code: 2D Laplace Equation

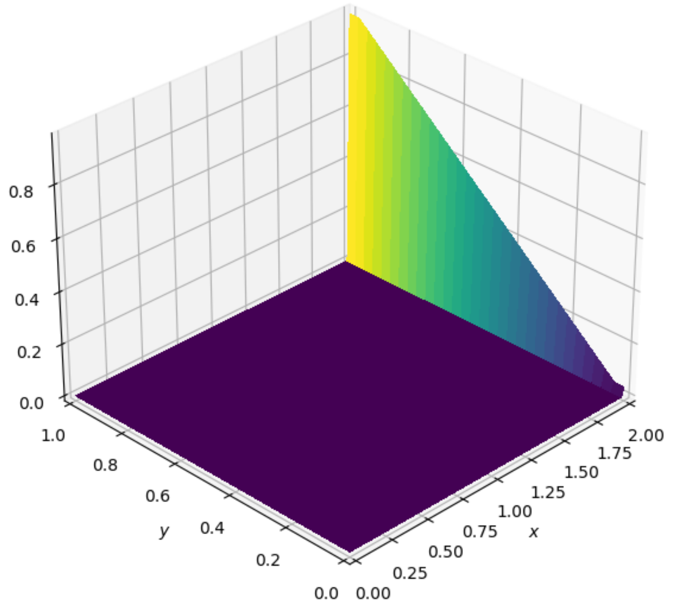
```
import numpy as np
from matplotlib import pyplot, cm
from mpl_toolkits.mplot3d import Axes3D

def plot2D(x,y,p):
   fig = pyplot.figure( figsize=(11,7), dpi=100 )
   ax  = fig.gca(projection='3d')
   X, Y = np.meshgrid(x,y)
   surf = ax.plot_surface( X, Y, p[:], rstride=1, cstride=1, cmap=cm.viridis,
                                    linewidth=0, antialiased=False )
   ax.set_xlabel("$x$");  ax.set_xlim(0,2)
   ax.set_ylabel("$y$");  ax.set_ylim(0,1)
   ax.view_init(30,225)

def laplace2D(p,y,dx,dy,target_norm):
   norm=1
   pn = np.empty_like(p)
   while norm > target_norm:
      pn = p.copy()
      p[1:-1,1:-1] = (
                        (dy**2 * (pn[1:-1,2:] – pn[1:-1,:-2]))   +
                        (dx**2 * (pn[2:,1:-1] – pn[:-2,1:-1]))
                     ) / (2* (dx**2 + dy**2))
      p[:,0] = 0                # p=0 at x=0
      p[:,-1]= y                # p=y at x=2
      p[0,:] = p[1,:]           # dp/dy=0 at y=0
      p[-1,:] = p[-2,:]         # dp/dy=0 at y=1
      norm = (np.sum(np.abs(p)) – np.sum(np.abs(pn)))/(np.sum(np.abs(pn)))
   return p
```

5th Code: 2D Laplace Equation

```
import numpy as np
from matplotlib import pyplot, cm
from mpl_toolkits.mplot3d import Axes3D

def plot2D(x,y,p):
   fig = pyplot.figure( figsize=(11,7), dpi=100 )
   ax  = fig.gca(projection='3d')
   X, Y = np.meshgrid(x,y)
   surf = ax.plot_surface( X, Y, p[:], rstride=1, cstride=1, cmap=cm.viridis,
                           linewidth=0, antialiased=False )
   ax.set_xlabel("$x$");  ax.set_xlim(0,2)
   ax.set_ylabel("$y$");  ax.set_ylim(0,1)
   ax.view_init(30,225)

def laplace2D(p,y,dx,dy,target_norm):
   norm=1
   pn = np.empty_like(p)
   while norm > target_norm:
      pn = p.copy()
      p[1:-1,1:-1] = (
                  (dy**2 * (pn[1:-1,2:] – pn[1:-1,:-2]))   +
                  (dx**2 * (pn[2:,1:-1] – pn[:-2,1:-1]))
                ) / (2* (dx**2 + dy**2))
      p[:,0] = 0               # p=0 at x=0
      p[:,-1]= y               # p=y at x=2
      p[0,:] = p[1,:]          # dp/dy=0 at y=0
      p[-1,:] = p[-2,:]        # dp/dy=0 at y=1
      norm = (np.sum(np.abs(p)) – np.sum(np.abs(pn)))/(np.sum(np.abs(pn)))
   return p

nx = 31; ny = 31; dx = 2/(nx-1); dy =1/(ny-1)
x = np.linspace(0, 2, nx); y = np.linspace(0, 1, ny)
```

5th Code: 2D Laplace Equation

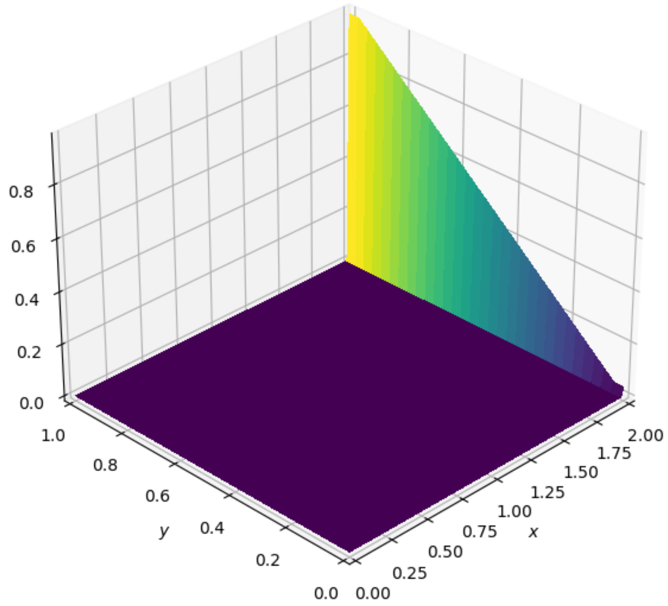
```python
import numpy as np
from matplotlib import pyplot, cm
from mpl_toolkits.mplot3d import Axes3D

def plot2D(x,y,p):
   fig = pyplot.figure( figsize=(11,7), dpi=100 )
   ax  = fig.gca(projection='3d')
   X, Y = np.meshgrid(x,y)
   surf = ax.plot_surface( X, Y, p[:], rstride=1, cstride=1, cmap=cm.viridis,
                           linewidth=0, antialiased=False )
   ax.set_xlabel("$x$");  ax.set_xlim(0,2)
   ax.set_ylabel("$y$");  ax.set_ylim(0,1)
   ax.view_init(30,225)

def laplace2D(p,y,dx,dy,target_norm):
    norm=1
    pn = np.empty_like(p)
    while norm > target_norm:
       pn = p.copy()
       p[1:-1,1:-1] = (
                      (dy**2 * (pn[1:-1,2:] – pn[1:-1,:-2]))   +
                      (dx**2 * (pn[2:,1:-1] – pn[:-2,1:-1]))
                     ) / (2* (dx**2 + dy**2))
       p[:,0] = 0                 # p=0 at x=0
       p[:,-1]= y                 # p=y at x=2
       p[0,:] = p[1,:]            # dp/dy=0 at y=0
       p[-1,:] = p[-2,:]          # dp/dy=0 at y=1
       norm = (np.sum(np.abs(p)) – np.sum(np.abs(pn)))/(np.sum(np.abs(pn)))
    return p

nx = 31; ny = 31; dx = 2/(nx-1); dy =1/(ny-1)
x = np.linspace(0, 2, nx); y = np.linspace(0, 1, ny)

p = np.zeros((ny,nx))
p[:,0] = 0              # p=0 at x=0
p[:,-1]= y              # p=y at x=2
p[0,:] = p[1,:]         # dp/dy=0 at y=0
p[-1,:] = p[-2,:]       # dp/dy=0 at y=1

plot2D(x,y,p)
```

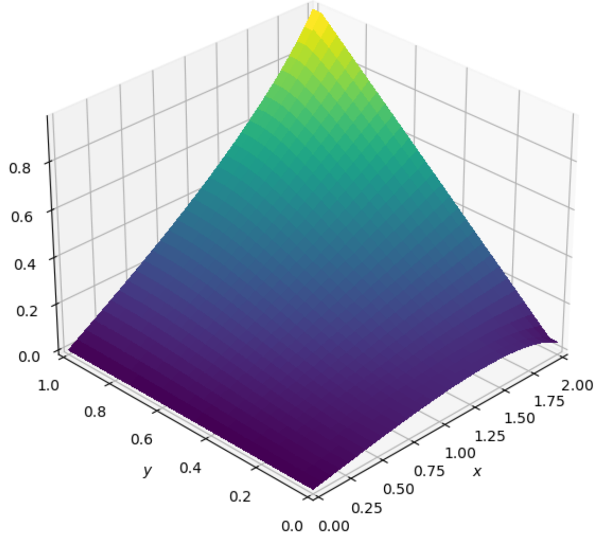


5th Code: 2D Laplace Equation

```
import numpy as np
from matplotlib import pyplot, cm
from mpl_toolkits.mplot3d import Axes3D

def plot2D(x,y,p):
   fig = pyplot.figure( figsize=(11,7), dpi=100 )
   ax  = fig.gca(projection='3d')
   X, Y = np.meshgrid(x,y)
   surf = ax.plot_surface( X, Y, p[:], rstride=1, cstride=1, cmap=cm.viridis,
                           linewidth=0, antialiased=False )
   ax.set_xlabel("$x$");  ax.set_xlim(0,2)
   ax.set_ylabel("$y$");  ax.set_ylim(0,1)
   ax.view_init(30,225)

def laplace2D(p,y,dx,dy,target_norm):
    norm=1
    pn = np.empty_like(p)
    while norm > target_norm:
       pn = p.copy()
       p[1:-1,1:-1] = (
                   (dy**2 * (pn[1:-1,2:] + pn[1:-1,:-2]))   +
                   (dx**2 * (pn[2:,1:-1] + pn[:-2,1:-1]))
                  ) / (2* (dx**2 + dy**2))
       p[:,0] = 0                # p=0 at x=0
       p[:,-1]= y                # p=y at x=2
       p[0,:] = p[1,:]           # dp/dy=0 at y=0
       p[-1,:] = p[-2,:]         # dp/dy=0 at y=1
       norm = (np.sum(np.abs(p)) – np.sum(np.abs(pn)))/(np.sum(np.abs(pn)))
    return p

nx = 31; ny = 31; dx = 2/(nx-1); dy =1/(ny-1)
x = np.linspace(0, 2, nx); y = np.linspace(0, 1, ny)
```

```
p = np.zeros((ny,nx))
p[:,0] = 0              # p=0 at x=0
p[:,-1]= y              # p=y at x=2
p[0,:] = p[1,:]         # dp/dy=0 at y=0
p[-1,:] = p[-2,:]       # dp/dy=0 at y=1

plot2D(x,y,p)
p = laplace2D(p,y,dx,dy,1e-4)
plot2D(x,y,p)
```





Navier-Stokes equations for incompressible flows

Continuity equation: $$\nabla . \vec{v} = 0$$

Momentum equation: $$\frac{\partial \vec{v}}{\partial t} + (\vec{v}.\nabla)\vec{v} = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \vec{v}$$

Navier-Stokes equations for incompressible flows

Continuity equation: $$\nabla . \vec{v} = 0$$

Momentum equation: $$\frac{\partial \vec{v}}{\partial t} + (\vec{v}.\nabla)\vec{v} = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \vec{v}$$

No equation for pressure !

Navier-Stokes equations for incompressible flows

Continuity equation: $$\nabla . \vec{v} = 0$$

Momentum equation: $$\frac{\partial \vec{v}}{\partial t} + (\vec{v}.\nabla)\vec{v} = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \vec{v}$$

No equation for pressure !

There is no obvious way of coupling these equations !

Navier-Stokes equations for incompressible flows

Continuity equation: $$\nabla . \vec{v} = 0$$

Provides a kinematic constraint that requires pressure field to evolve such that the velocity field remains solenoidal !

Momentum equation: $$\frac{\partial \vec{v}}{\partial t} + (\vec{v} . \nabla)\vec{v} = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \vec{v}$$

No equation for pressure !

There is no obvious way of coupling these equations !

Navier-Stokes equations for incompressible flows

Continuity equation: $\nabla . \vec{v} = 0$

Provides a kinematic constraint that requires pressure field to evolve such that the velocity field remains solenoidal !

Momentum equation: $$\frac{\partial \vec{v}}{\partial t} + (\vec{v}.\nabla)\vec{v} = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \vec{v}$$

No equation for pressure !

There is no obvious way of coupling these equations !

Remedy: "Construct" a pressure field that guarantees continuity constraint!

Navier-Stokes equations for incompressible flows

Continuity equation: $\nabla . \vec{v} = 0$

Provides a kinematic constraint that requires pressure field to evolve such that the velocity field remains solenoidal !

Momentum equation: $$\frac{\partial \vec{v}}{\partial t} + (\vec{v}.\nabla)\vec{v} = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \vec{v}$$

No equation for pressure !

There is no obvious way of coupling these equations !

Such a relation can be obtained by taking the divergence of the momentum equation.

Remedy: "Construct" a pressure field that guarantees continuity constraint!

Navier-Stokes equations for incompressible flows

Continuity equation: $$\nabla.\vec{v} = 0$$

Provides a kinematic constraint that requires pressure field to evolve such that the velocity field remains solenoidal !

Momentum equation: $$\frac{\partial \vec{v}}{\partial t} + (\vec{v}.\nabla)\vec{v} = -\frac{1}{\rho}\nabla p + \nu\nabla^2\vec{v}$$

No equation for pressure !

There is no obvious way of coupling these equations !

Such a relation can be obtained by taking the divergence of the momentum equation.

$$\nabla.\left(\frac{\partial \vec{v}}{\partial t} + (\vec{v}.\nabla)\vec{v}\right) = \nabla.(-\nabla p + \nu\nabla^2\vec{v})$$

Remedy: "Construct" a pressure field that guarantees continuity constraint!

Navier-Stokes equations for incompressible flows

Continuity equation: $$\nabla . \vec{v} = 0$$

Provides a kinematic constraint that requires pressure field to evolve such that the velocity field remains solenoidal !

Momentum equation: $$\frac{\partial \vec{v}}{\partial t} + (\vec{v}.\nabla)\vec{v} = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \vec{v}$$

No equation for pressure !

There is no obvious way of coupling these equations !

Such a relation can be obtained by taking the divergence of the momentum equation.

$$\nabla . \left(\frac{\partial \vec{v}}{\partial t} + (\vec{v}.\nabla)\vec{v} \right) = \nabla . \left(-\nabla p + \nu \nabla^2 \vec{v} \right)$$

$$\frac{\partial \nabla . \vec{v}}{\partial t} + \nabla . (\vec{v}.\nabla)\vec{v} = -\nabla^2 p + \nu \nabla^2 \nabla . \vec{v}$$

Remedy: "Construct" a pressure field that guarantees continuity constraint!

Navier-Stokes equations for incompressible flows

Continuity equation: $$\nabla.\vec{v} = 0$$

Provides a kinematic constraint that requires pressure field to evolve such that the velocity field remains solenoidal !

Momentum equation: $$\frac{\partial \vec{v}}{\partial t} + (\vec{v}.\nabla)\vec{v} = -\frac{1}{\rho}\nabla p + \nu\nabla^2\vec{v}$$

No equation for pressure !

There is no obvious way of coupling these equations !

Such a relation can be obtained by taking the divergence of the momentum equation.

$$\nabla.\left(\frac{\partial \vec{v}}{\partial t} + (\vec{v}.\nabla)\vec{v}\right) = \nabla.(-\nabla p + \nu\nabla^2\vec{v})$$

$$\cancel{\frac{\partial \nabla.\vec{v}}{\partial t}}^{\,0} + \nabla.(\vec{v}.\nabla)\vec{v} = -\nabla^2 p + \nu\cancel{\nabla^2\nabla.\vec{v}}^{\,0}$$

Remedy: "Construct" a pressure field that guarantees continuity constraint!

Navier-Stokes equations for incompressible flows

Continuity equation: $\nabla . \vec{v} = 0$ ← Provides a kinematic constraint that requires pressure field to evolve such that the velocity field remains solenoidal !

Momentum equation: $$\frac{\partial \vec{v}}{\partial t} + (\vec{v}.\nabla)\vec{v} = -\frac{1}{\rho}\nabla p + \nu\nabla^2\vec{v}$$

No equation for pressure !

There is no obvious way of coupling these equations !

Such a relation can be obtained by taking the divergence of the momentum equation.

$$\nabla .\left(\frac{\partial \vec{v}}{\partial t} + (\vec{v}.\nabla)\vec{v}\right) = \nabla .(-\nabla p + \nu\nabla^2\vec{v})$$

$$\cancel{\frac{\partial \nabla . \vec{v}}{\partial t}}^{0} + \nabla .(\vec{v}.\nabla)\vec{v} = -\nabla^2 p + \cancel{\nu\nabla^2\nabla . \vec{v}}^{0}$$

$$\nabla^2 p = -\nabla .(\vec{v}.\nabla)\vec{v}$$

Remedy: “Construct” a pressure field that guarantees continuity constraint!

Navier-Stokes equations for incompressible flows

Continuity equation: $$\nabla.\vec{v} = 0$$

Provides a kinematic constraint that requires pressure field to evolve such that the velocity field remains solenoidal !

Momentum equation: $$\frac{\partial \vec{v}}{\partial t} + (\vec{v}.\nabla)\vec{v} = -\frac{1}{\rho}\nabla p + \nu\nabla^2\vec{v}$$

No equation for pressure !

There is no obvious way of coupling these equations !

Such a relation can be obtained by taking the divergence of the momentum equation.

$$\nabla.\left(\frac{\partial \vec{v}}{\partial t} + (\vec{v}.\nabla)\vec{v}\right) = \nabla.(-\nabla p + \nu\nabla^2\vec{v})$$

$$\cancel{\frac{\partial \nabla.\vec{v}}{\partial t}}^{\,0} + \nabla.(\vec{v}.\nabla)\vec{v} = -\nabla^2 p + \nu\cancel{\nabla^2\nabla.\vec{v}}^{\,0}$$

Remedy: "Construct" a pressure field that guarantees continuity constraint!

Pressure Poisson Equation

$$\boxed{\nabla^2 p = -\nabla.(\vec{v}.\nabla)\vec{v}}$$

Navier-Stokes equations for incompressible flows

Continuity equation: $$\nabla.\vec{v} = 0$$

Provides a kinematic constraint that requires pressure field to evolve such that the velocity field remains solenoidal !

Momentum equation: $$\frac{\partial \vec{v}}{\partial t} + (\vec{v}.\nabla)\vec{v} = -\frac{1}{\rho}\nabla p + \nu\nabla^2\vec{v}$$

No equation for pressure !

There is no obvious way of coupling these equations !

Such a relation can be obtained by taking the divergence of the momentum equation.

$$\nabla.\left(\frac{\partial \vec{v}}{\partial t} + (\vec{v}.\nabla)\vec{v}\right) = \nabla.(-\nabla p + \nu\nabla^2\vec{v})$$

$$\cancel{\frac{\partial \nabla.\vec{v}}{\partial t}}^{0} + \nabla.(\vec{v}.\nabla)\vec{v} = -\nabla^2 p + \nu\nabla^2\cancel{\nabla.\vec{v}}^{0}$$

Remedy: "Construct" a pressure field that guarantees continuity constraint!

Pressure Poisson Equation

$$\boxed{\nabla^2 p = -\nabla.(\vec{v}.\nabla)\vec{v}}$$

Let's look at the 6th code to learn how to solve a Poisson's equation.

6^{th} Code: 2D Poisson's Equation

Obtained by adding a source term on the right-hand side of the Laplace's equation

$$\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} = b$$

6th Code: 2D Poisson's Equation

Obtained by adding a source term on the right-hand side of the Laplace's equation

$$\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} = b$$

Poisson's equation act to "relax" the initial sources in the field

6th Code: 2D Poisson's Equation

Obtained by adding a source term on the right-hand side of the Laplace's equation

$$\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} = b$$

Poisson's equation act to "relax" the initial sources in the field

Discretized form:

$$\frac{p_{i+1,j}^{n} - 2p_{i,j}^{n} + p_{i-1,j}^{n}}{\Delta x^2} + \frac{p_{i,j+1}^{n} - 2p_{i,j}^{n} + p_{i,j-1}^{n}}{\Delta y^2} = b_{i,j}^{n}$$

6th Code: 2D Poisson's Equation

Obtained by adding a source term on the right-hand side of the Laplace's equation

$$\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} = b$$

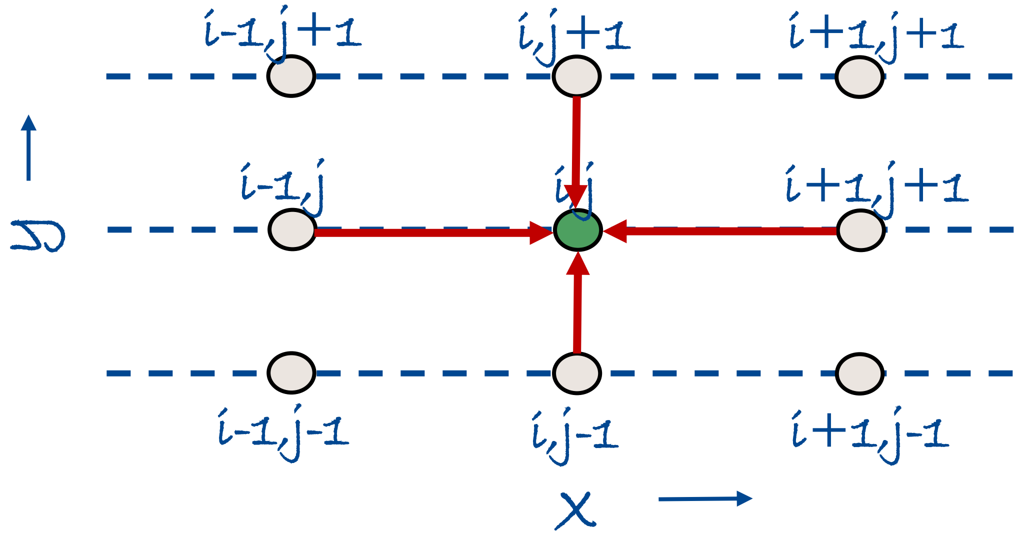
Poisson's equation act to "relax" the initial sources in the field

Discretized form:

$$\frac{p^n_{i+1,j} - 2p^n_{i,j} + p^n_{i-1,j}}{\Delta x^2} + \frac{p^n_{i,j+1} - 2p^n_{i,j} + p^n_{i,j-1}}{\Delta y^2} = b^n_{i,j}$$

$$p^n_{i,j} = \frac{\Delta y^2 (p^n_{i+1,j} + p^n_{i-1,j}) + \Delta x^2 (p^n_{i,j+1} + p^n_{i,j-1}) - b^n_{i,j} \Delta x^2 \Delta y^2}{2(\Delta x^2 + \Delta y^2)}$$

5 points stencil



6^{th} Code: 2D Poisson's Equation

Obtained by adding a source term on the right-hand side of the Laplace's equation

$$\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} = b$$

Poisson's equation act to "relax" the initial sources in the field

Discretized form:

$$\frac{p_{i+1,j}^{n} - 2p_{i,j}^{n} + p_{i-1,j}^{n}}{\Delta x^2} + \frac{p_{i,j+1}^{n} - 2p_{i,j}^{n} + p_{i,j-1}^{n}}{\Delta y^2} = b_{i,j}^{n}$$

$$p_{i,j}^{n} = \frac{\Delta y^2 (p_{i+1,j}^{n} + p_{i-1,j}^{n}) + \Delta x^2 (p_{i,j+1}^{n} + p_{i,j-1}^{n}) - b_{i,j}^{n} \Delta x^2 \Delta y^2}{2(\Delta x^2 + \Delta y^2)}$$

Source term:

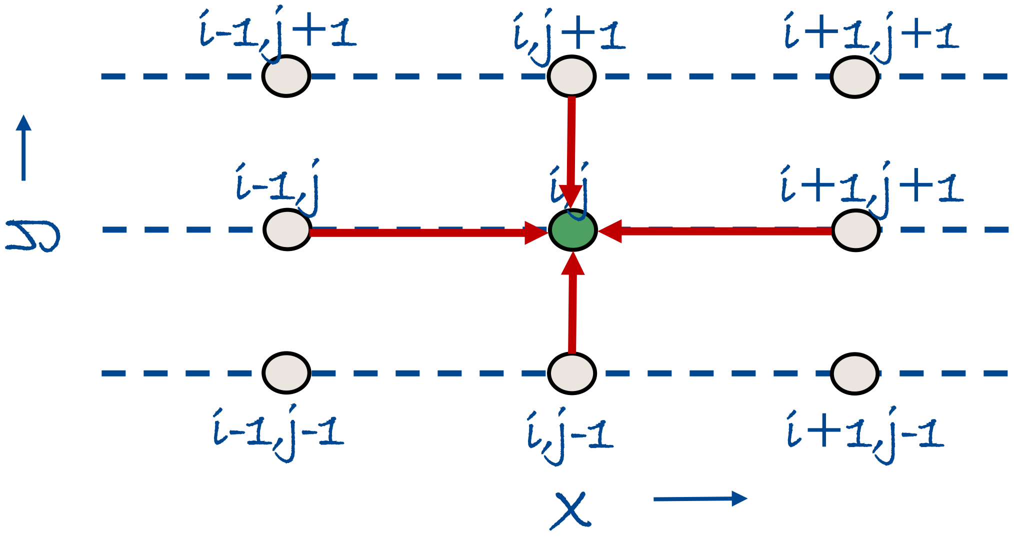
2 Spikes

$$b_{i,j} = -100 \ at \ i = \frac{nx}{4}, j = \frac{3ny}{4}$$

$$b_{i,j} = 100 \ at \ i = \frac{3nx}{4}, j = \frac{ny}{4}$$

$$b_{i,j} = 0 \ \ elsewhere$$

5 points stencil



6^{th} Code: 2D Poisson's Equation

Obtained by adding a source term on the right-hand side of the Laplace's equation

$$\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} = b$$

Poisson's equation act to "relax" the initial sources in the field

Discretized form:

$$\frac{p^n_{i+1,j} - 2p^n_{i,j} + p^n_{i-1,j}}{\Delta x^2} + \frac{p^n_{i,j+1} - 2p^n_{i,j} + p^n_{i,j-1}}{\Delta y^2} = b^n_{i,j}$$

$$p^n_{i,j} = \frac{\Delta y^2 (p^n_{i+1,j} + p^n_{i-1,j}) + \Delta x^2 (p^n_{i,j+1} + p^n_{i,j-1}) - b^n_{i,j} \Delta x^2 \Delta y^2}{2(\Delta x^2 + \Delta y^2)}$$

Source term:

$$b_{i,j} = -100 \ at \ i = \frac{nx}{4}, j = \frac{3ny}{4}$$

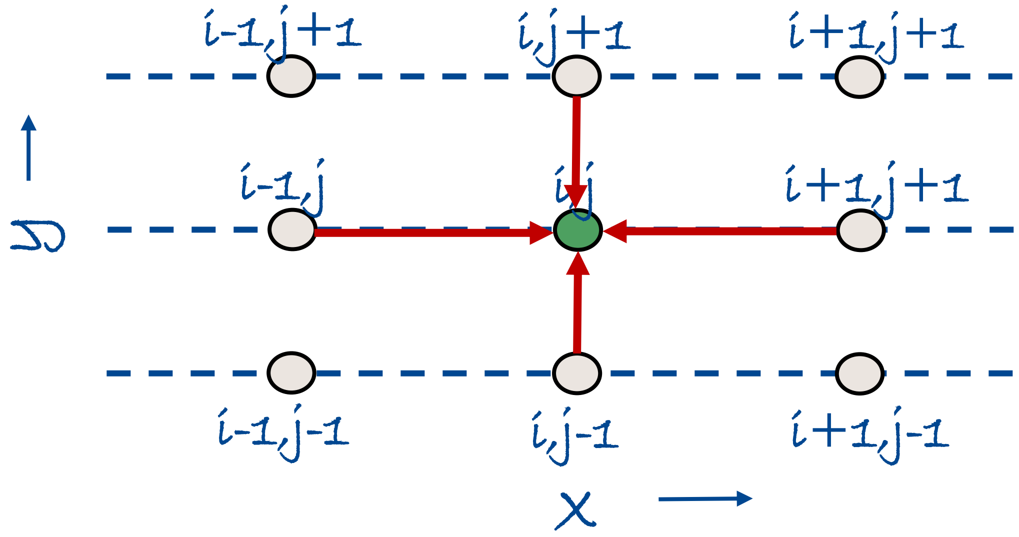
2 Spikes

$$b_{i,j} = 100 \ at \ i = \frac{3nx}{4}, j = \frac{ny}{4}$$

$$b_{i,j} = 0 \ elsewhere$$

The iteration will relax the initial spikes!

5 points stencil



6th Code: 2D Poisson's Equation

Obtained by adding a source term on the right-hand side of the Laplace's equation

$$\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} = b$$

Poisson's equation act to "relax" the initial sources in the field

Discretized form:

$$\frac{p^n_{i+1,j} - 2p^n_{i,j} + p^n_{i-1,j}}{\Delta x^2} + \frac{p^n_{i,j+1} - 2p^n_{i,j} + p^n_{i,j-1}}{\Delta y^2} = b^n_{i,j}$$

BCs:

$$p = 0 \; at \; x = 0,2$$

$$p = 0 \; at \; y = 0,2$$

$$p^n_{i,j} = \frac{\Delta y^2 (p^n_{i+1,j} + p^n_{i-1,j}) + \Delta x^2 (p^n_{i,j+1} + p^n_{i,j-1}) - b^n_{i,j} \Delta x^2 \Delta y^2}{2(\Delta x^2 + \Delta y^2)}$$

Source term:

2 Spikes

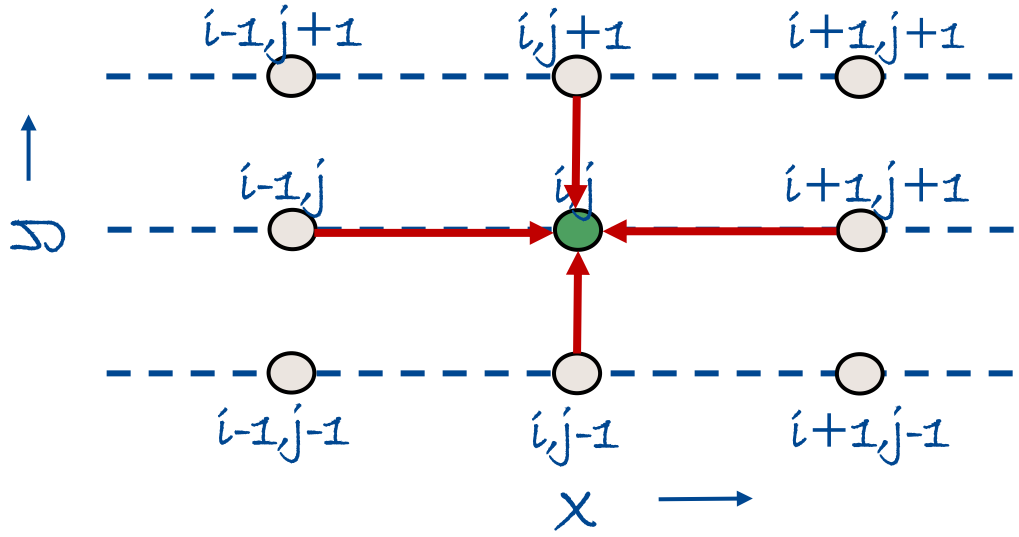
$$b_{i,j} = -100 \; at \; i = \frac{nx}{4}, j = \frac{3ny}{4}$$

$$b_{i,j} = 100 \; at \; i = \frac{3nx}{4}, j = \frac{ny}{4}$$

$$b_{i,j} = 0 \; elsewhere$$

The iteration will relax the initial spikes!

5 points stencil



6^{th} Code: 2D Poisson's Equation

Obtained by adding a source term on the right-hand side of the Laplace's equation

$$\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} = b$$

Poisson's equation act to "relax" the initial sources in the field

Discretized form:

$$\frac{p_{i+1,j}^n - 2p_{i,j}^n + p_{i-1,j}^n}{\Delta x^2} + \frac{p_{i,j+1}^n - 2p_{i,j}^n + p_{i,j-1}^n}{\Delta y^2} = b_{i,j}^n$$

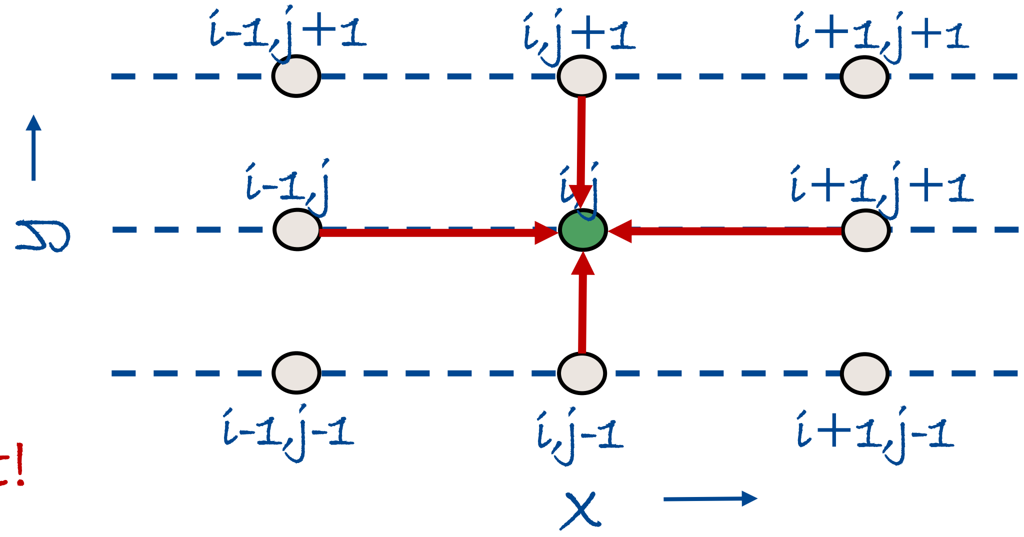
BCs:

$$p = 0 \ at \ x = 0,2$$

$$p = 0 \ at \ y = 0,2$$

$$p_{i,j}^n = \frac{\Delta y^2 (p_{i+1,j}^n + p_{i-1,j}^n) + \Delta x^2 (p_{i,j+1}^n + p_{i,j-1}^n) - b_{i,j}^n \Delta x^2 \Delta y^2}{2(\Delta x^2 + \Delta y^2)}$$

5 points stencil



Source term:

2 Spikes

$$b_{i,j} = -100 \ at \ i = \frac{nx}{4}, j = \frac{3ny}{4}$$

$$b_{i,j} = 100 \ at \ i = \frac{3nx}{4}, j = \frac{ny}{4}$$

$$b_{i,j} = 0 \ elsewhere$$

The iteration will relax the initial spikes!

Let's Code it!

6th Code: 2D Poisson's Equation

```
import numpy as np
from matplotlib import pyplot, cm
from mpl_toolkits.mplot3d import Axes3D
```

6^{th} Code: 2D Poisson's Equation

```
import numpy as np
from matplotlib import pyplot, cm
from mpl_toolkits.mplot3d import Axes3D

def plot2D(x,y,p):
    fig = pyplot.figure( figsize=(11,7), dpi=100 )
    ax  = fig.gca(projection='3d')
    X, Y = np.meshgrid(x,y)
    surf = ax.plot_surface( X, Y, p[:], rstride=1, cstride=1, cmap=cm.viridis,
                            linewidth=0, antialiased=False )
    ax.set_xlabel("$x$");  ax.set_xlim(0,2)
    ax.set_ylabel("$y$");  ax.set_ylim(0,1)
    ax.view_init(30,225)
```

6th Code: 2D Poisson's Equation

```
import numpy as np
from matplotlib import pyplot, cm
from mpl_toolkits.mplot3d import Axes3D

def plot2D(x,y,p):
   fig = pyplot.figure( figsize=(11,7), dpi=100 )
   ax  = fig.gca(projection='3d')
   X, Y = np.meshgrid(x,y)
   surf = ax.plot_surface( X, Y, p[:], rstride=1, cstride=1, cmap=cm.viridis,
                                    linewidth=0, antialiased=False )
   ax.set_xlabel("$x$");  ax.set_xlim(0,2)
   ax.set_ylabel("$y$");  ax.set_ylim(0,1)
   ax.view_init(30,225)

def  poisson2D(p,b,dx,dy,target_norm):
    norm=1; small=1e-8; niter=0
    pn = np.zeros_like(p)
    while norm > target_norm:
       pn = p.copy(); niter+=1
       p[1:-1,1:-1] = ( ( dy**2 * (pn[2:,1:-1] – pn[:-2,1:-1])    +
                          dx**2 * (pn[1:-1,2:] – pn[1:-1,:-2])    -
                          dx**2 * dy**2 * b[1:-1,1:-1] ) /
                         (2* (dx**2 + dy**2)))
       p[0,:]  = 0            # p=0 at x=0
       p[-1,:] = 0            # p=0 at x=2
       p[:,0]  = 0            # p=0 at y=0
       p[:,-1] = 0            # p=0 at y=2
       norm = (np.sum(np.abs(p)) – np.sum(np.abs(pn)))/(np.sum(np.abs(pn))+small)
    return p
```

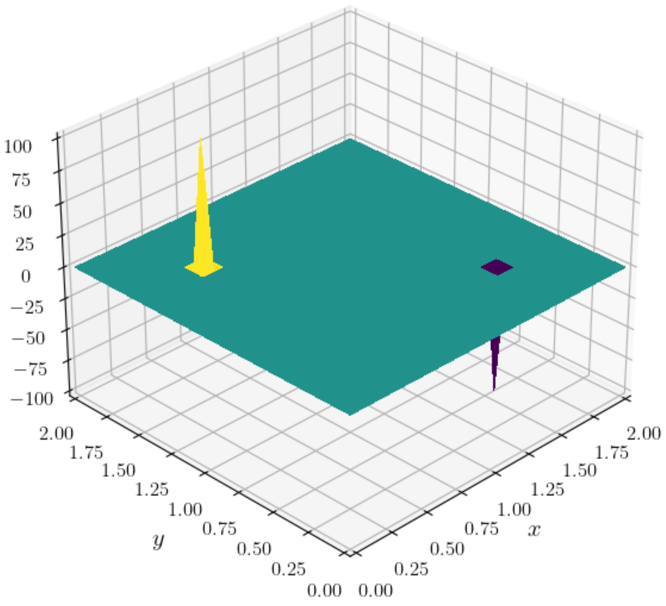
6th Code: 2D Poisson's Equation

```
import numpy as np
from matplotlib import pyplot, cm
from mpl_toolkits.mplot3d import Axes3D

def plot2D(x,y,p):
   fig = pyplot.figure( figsize=(11,7), dpi=100 )
   ax  = fig.gca(projection='3d')
   X, Y = np.meshgrid(x,y)
   surf = ax.plot_surface( X, Y, p[:], rstride=1, cstride=1, cmap=cm.viridis,
                           linewidth=0, antialiased=False )
   ax.set_xlabel("$x$");  ax.set_xlim(0,2)
   ax.set_ylabel("$y$");  ax.set_ylim(0,2)
   ax.view_init(30,225)

def  poisson2D(p,b,dx,dy,target_norm):
   norm=1; small=1e-8; niter=0
   pn = np.zeros_like(p)
   while norm > target_norm:
      pn = p.copy(); niter+=1
      p[1:-1,1:-1] = ( ( dy**2 * (pn[2:,1:-1] – pn[:-2,1:-1])    +
                         dx**2 * (pn[1:-1,2:] – pn[1:-1,:-2])    -
                         dx**2 * dy**2 * b[1:-1,1:-1] ) /
                        (2* (dx**2 + dy**2)))
      p[0,:]  = 0          # p=0 at x=0
      p[-1,:] = 0          # p=0 at x=2
      p[:,0]  = 0          # p=0 at y=0
      p[:,-1] = 0          # p=0 at y=2
      norm = (np.sum(np.abs(p)) – np.sum(np.abs(pn)))/(np.sum(np.abs(pn))+small)
   return p


nx = 51; ny = 51; dx = 2/(nx-1); dy = 2/(ny-1); nt = 100;
x = np.linspace(0, 2, nx); y = np.linspace(0, 2, ny)
```

6th Code: 2D Poisson's Equation

```python
import numpy as np
from matplotlib import pyplot, cm
from mpl_toolkits.mplot3d import Axes3D

def plot2D(x,y,p):
   fig = pyplot.figure( figsize=(11,7), dpi=100 )
   ax  = fig.gca(projection='3d')
   X, Y = np.meshgrid(x,y)
   surf = ax.plot_surface( X, Y, p[:], rstride=1, cstride=1, cmap=cm.viridis,
                                   linewidth=0, antialiased=False )
   ax.set_xlabel("$x$");  ax.set_xlim(0,2)
   ax.set_ylabel("$y$");  ax.set_ylim(0,1)
   ax.view_init(30,225)

def  poisson2D(p,b,dx,dy,target_norm):
    norm=1; small=1e-8; niter=0
    pn = np.zeros_like(p)
    while norm > target_norm:
       pn = p.copy(); niter+=1
       p[1:-1,1:-1] = ( ( dy**2 * (pn[2:,1:-1] – pn[:-2,1:-1])    +
                          dx**2 * (pn[1:-1,2:] – pn[1:-1,:-2])    -
                          dx**2 * dy**2 * b[1:-1,1:-1] ) /
                        (2* (dx**2 + dy**2)))
       p[0,:]  = 0            # p=0 at x=0
       p[-1,:] = 0            # p=0 at x=2
       p[:,0]  = 0            # p=0 at y=0
       p[:,-1] = 0            # p=0 at y=2
       norm = (np.sum(np.abs(p)) – np.sum(np.abs(pn)))/(np.sum(np.abs(pn))+small)
    return p


nx = 51; ny = 51; dx = 2/(nx-1); dy = 2/(ny-1); nt = 100;
x = np.linspace(0, 2, nx); y = np.linspace(0, 1, ny)
```

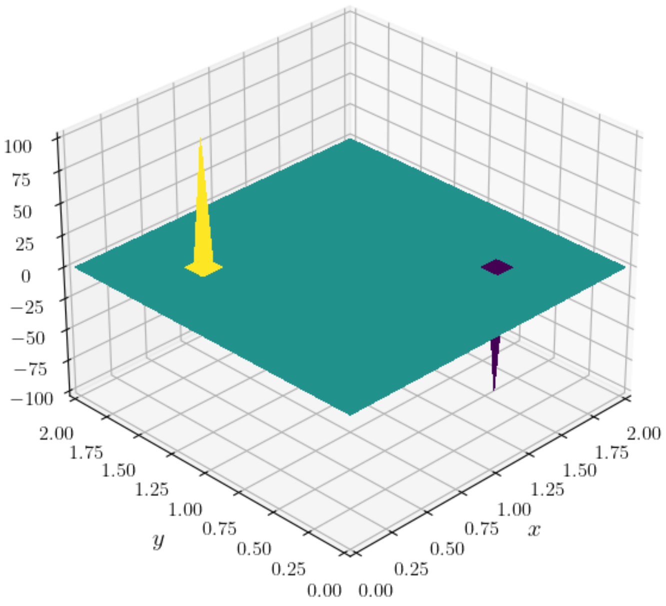
```python
p  = np.zeros((nx,ny))
b  = np.zeros((nx,ny))

b[int(nx/4), int(3*ny/4)] = -100
b[int(3*nx/4), int(ny/4)] = 100

plot2D(x,y,p)
plot2D(x,y,b)
```



6th Code: 2D Poisson's Equation

```python
import numpy as np
from matplotlib import pyplot, cm
from mpl_toolkits.mplot3d import Axes3D

def plot2D(x,y,p):
   fig = pyplot.figure( figsize=(11,7), dpi=100 )
   ax  = fig.gca(projection='3d')
   X, Y = np.meshgrid(x,y)
   surf = ax.plot_surface( X, Y, p[:], rstride=1, cstride=1, cmap=cm.viridis,
                           linewidth=0, antialiased=False )
   ax.set_xlabel("$x$");  ax.set_xlim(0,2)
   ax.set_ylabel("$y$");  ax.set_ylim(0,1)
   ax.view_init(30,225)

def  poisson2D(p,b,dx,dy,target_norm):
    norm=1; small=1e-8; niter=0
    pn = np.zeros_like(p)
    while norm > target_norm:
       pn = p.copy(); niter+=1
       p[1:-1,1:-1] = ( ( dy**2 * (pn[2:,1:-1] – pn[:-2,1:-1])    +
                          dx**2 * (pn[1:-1,2:] – pn[1:-1,:-2])    -
                          dx**2 * dy**2 * b[1:-1,1:-1] ) /
                         (2* (dx**2 + dy**2)))
       p[0,:]  = 0           # p=0 at x=0
       p[-1,:] = 0           # p=0 at x=2
       p[:,0]  = 0           # p=0 at y=0
       p[:,-1] = 0           # p=0 at y=2
       norm = (np.sum(np.abs(p)) – np.sum(np.abs(pn)))/(np.sum(np.abs(pn))+small)
    return p


nx = 51; ny = 51; dx = 2/(nx-1); dy = 2/(ny-1); nt = 100;
x = np.linspace(0, 2, nx); y = np.linspace(0, 1, ny)
```

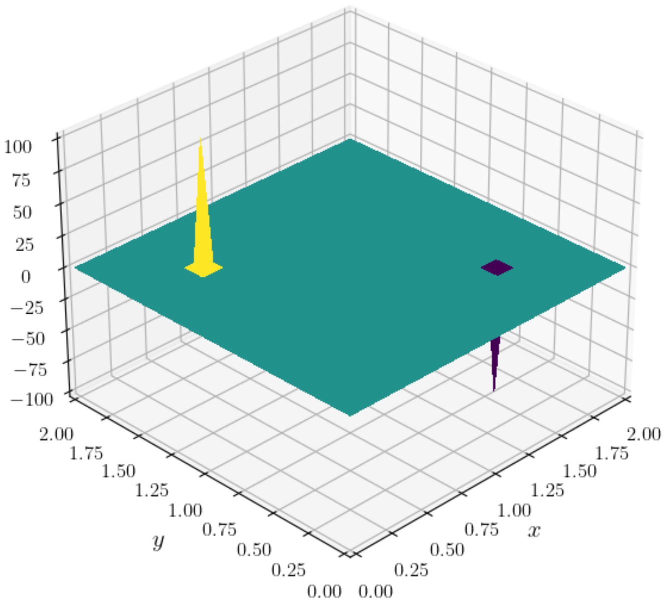
```python
p  = np.zeros((nx,ny))
b  = np.zeros((nx,ny))

b[int(nx/4), int(3*ny/4)] = -100
b[int(3*nx/4), int(ny/4)] = 100

plot2D(x,y,p)
plot2D(x,y,b)

p, niter = poisson2D(p,b,dx,dy,1e-4)
```

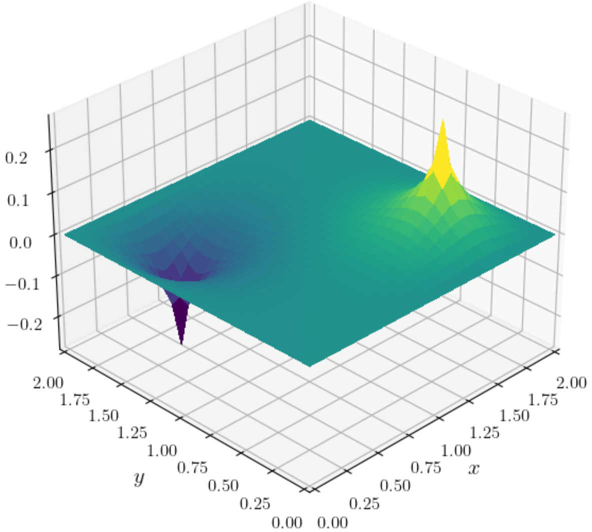


6th Code: 2D Poisson's Equation

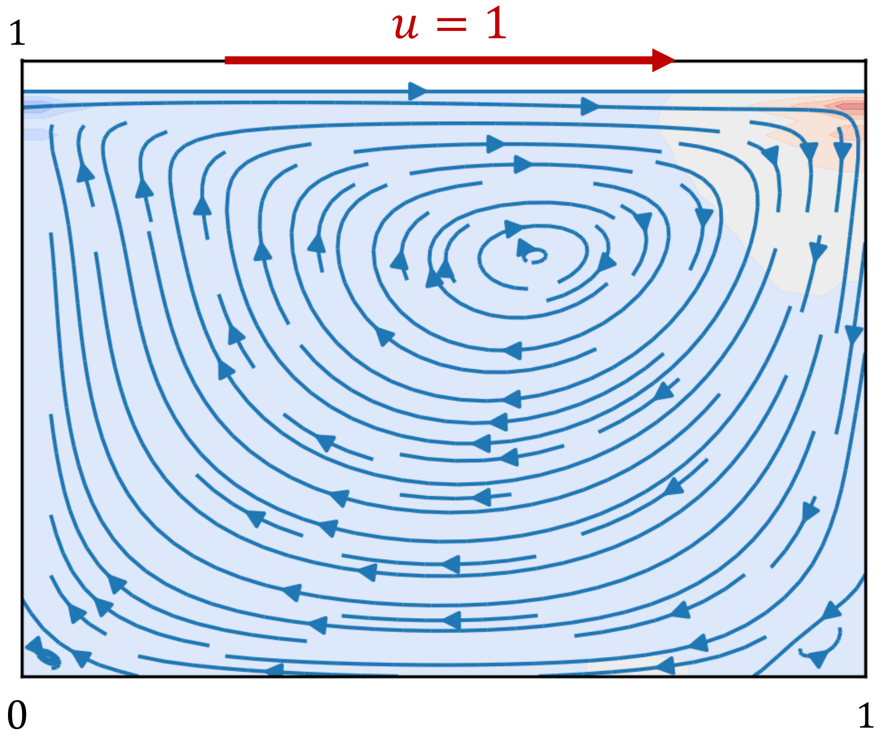
```
import numpy as np
from matplotlib import pyplot, cm
from mpl_toolkits.mplot3d import Axes3D

def plot2D(x,y,p):
   fig = pyplot.figure( figsize=(11,7), dpi=100 )
   ax  = fig.gca(projection=‘3d’)
   X, Y = np.meshgrid(x,y)
   surf = ax.plot_surface( X, Y, p[:], rstride=1, cstride=1, cmap=cm.viridis,
                           linewidth=0, antialiased=False )
   ax.set_xlabel(“$x$”);  ax.set_xlim(0,2)
   ax.set_ylabel(“$y$”);  ax.set_ylim(0,1)
   ax.view_init(30,225)

def  poisson2D(p,b,dx,dy,target_norm):
    norm=1; small=1e-8; niter=0
    pn = np.zeros_like(p)
    while norm > target_norm:
       pn = p.copy(); niter+=1
       p[1:-1,1:-1] = ( ( dy**2 * (pn[2:,1:-1] – pn[:-2,1:-1])    +
                          dx**2 * (pn[1:-1,2:] – pn[1:-1,:-2])    -
                          dx**2 * dy**2 * b[1:-1,1:-1] ) /
                         (2* (dx**2 + dy**2)))
       p[0,:]  = 0           # p=0 at x=0
       p[-1,:] = 0           # p=0 at x=2
       p[:,0]  = 0           # p=0 at y=0
       p[:,-1] = 0           # p=0 at y=2
       norm = (np.sum(np.abs(p)) – np.sum(np.abs(pn)))/(np.sum(np.abs(pn))+small)
    return p, niter


nx = 51; ny = 51; dx = 2/(nx-1); dy = 2/(ny-1); nt = 100;
x = np.linspace(0, 2, nx); y = np.linspace(0, 1, ny)
```

```
p  = np.zeros((nx,ny))
b  = np.zeros((nx,ny))

b[int(nx/4), int(3*ny/4)] = -100
b[int(3*nx/4), int(ny/4)] = 100

plot2D(x,y,p)
plot2D(x,y,b)

p, niter = poisson2D(p,b,dx,dy,1e-4)
plot2D(x,y,p)
```





Practical Session: 2D Lid Driven Cavity Flow

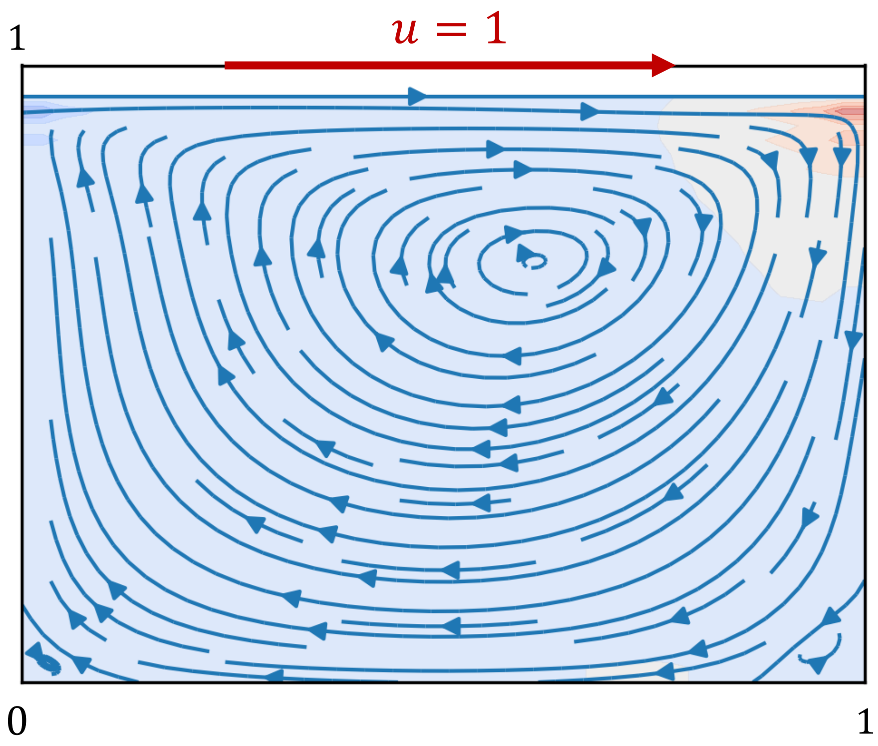


Practical Session: 2D Lid Driven Cavity Flow

MODELAIR

Continuity: $\nabla . \vec{v} = 0$

Momentum: $\dfrac{\partial \vec{v}}{\partial t} + (\vec{v}.\nabla)\vec{v} = -\dfrac{1}{\rho}\nabla p + \nu \nabla^2 \vec{v}$



Practical Session: 2D Lid Driven Cavity Flow

KTH VETENSKAP OCH KONST

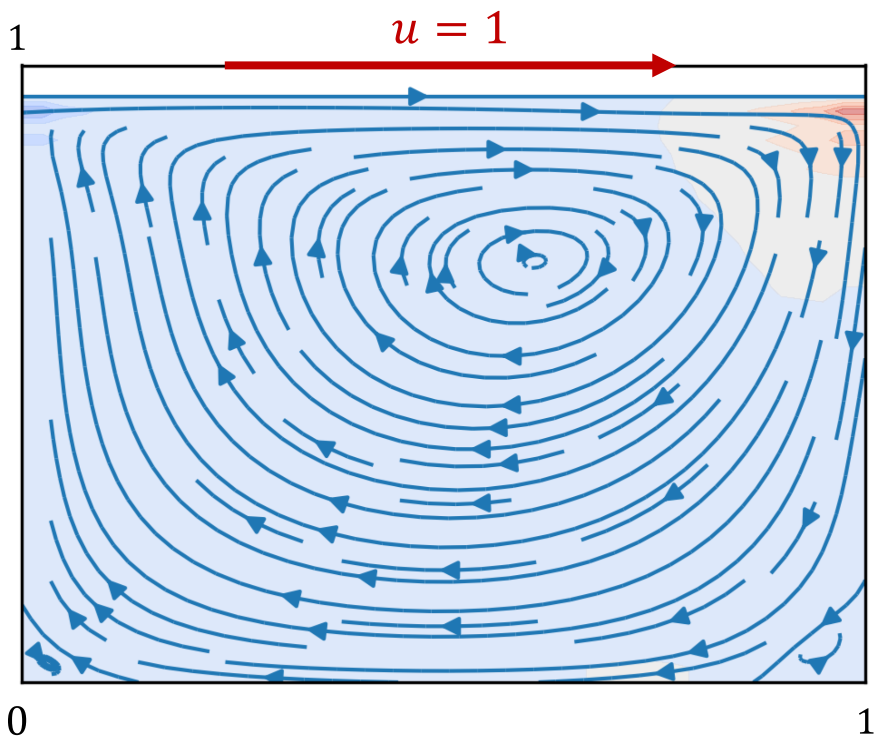
MODELAIR

Continuity: $\nabla . \vec{v} = 0$

Momentum: $\dfrac{\partial \vec{v}}{\partial t} + (\vec{v}.\nabla)\vec{v} = -\dfrac{1}{\rho}\nabla p + \nu \nabla^2 \vec{v}$

X-velocity: $\dfrac{\partial u}{\partial t} + u\left(\dfrac{\partial u}{\partial x}\right) + v\left(\dfrac{\partial u}{\partial y}\right) = -\dfrac{1}{\rho}\left(\dfrac{\partial p}{\partial x}\right) + \nu\left(\dfrac{\partial^2 u}{\partial x^2} + \dfrac{\partial^2 u}{\partial y^2}\right)$

Y-velocity: $\dfrac{\partial v}{\partial t} + u\left(\dfrac{\partial v}{\partial x}\right) + v\left(\dfrac{\partial v}{\partial y}\right) = -\dfrac{1}{\rho}\left(\dfrac{\partial p}{\partial y}\right) + \nu\left(\dfrac{\partial^2 v}{\partial x^2} + \dfrac{\partial^2 v}{\partial y^2}\right)$



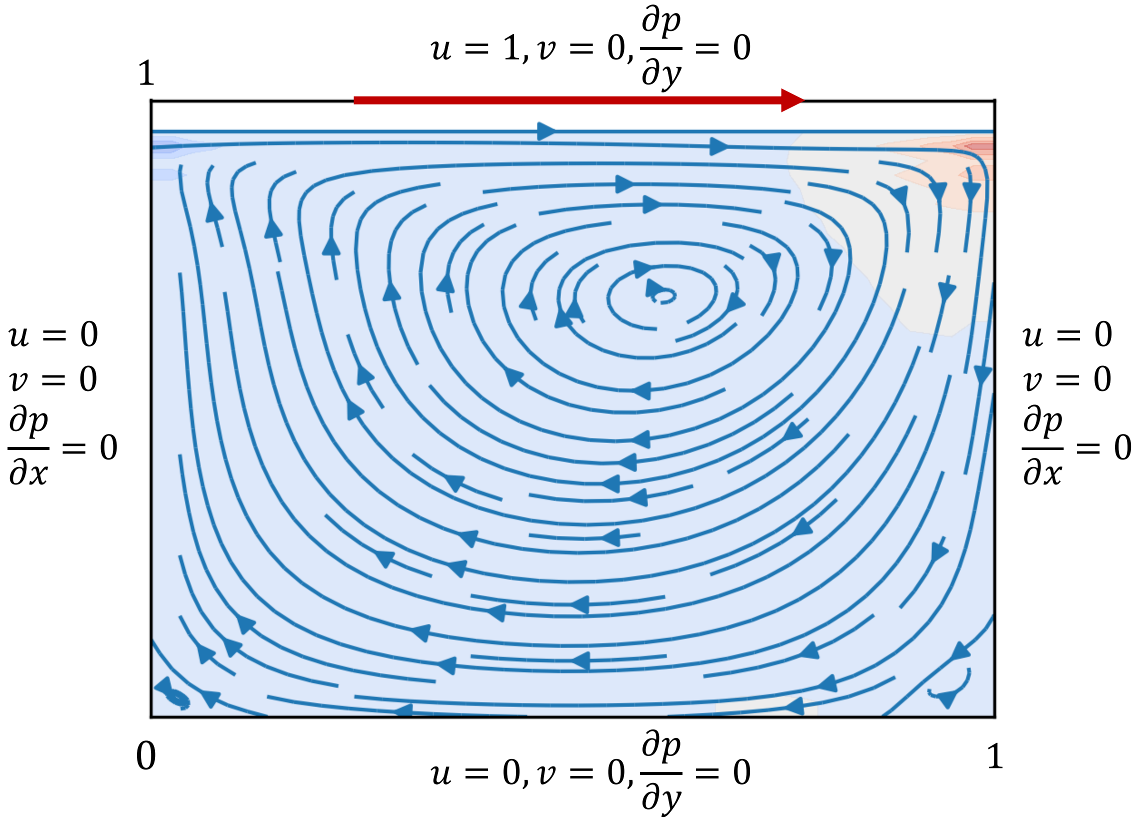
Practical Session: 2D Lid Driven Cavity Flow

Continuity: $\nabla . \vec{v} = 0$

Momentum: $\dfrac{\partial \vec{v}}{\partial t} + (\vec{v}.\nabla)\vec{v} = -\dfrac{1}{\rho}\nabla p + \nu \nabla^2 \vec{v}$

X-velocity: $\dfrac{\partial u}{\partial t} + u\left(\dfrac{\partial u}{\partial x}\right) + v\left(\dfrac{\partial u}{\partial y}\right) = -\dfrac{1}{\rho}\left(\dfrac{\partial p}{\partial x}\right) + \nu\left(\dfrac{\partial^2 u}{\partial x^2} + \dfrac{\partial^2 u}{\partial y^2}\right)$

Y-velocity: $\dfrac{\partial v}{\partial t} + u\left(\dfrac{\partial v}{\partial x}\right) + v\left(\dfrac{\partial v}{\partial y}\right) = -\dfrac{1}{\rho}\left(\dfrac{\partial p}{\partial y}\right) + \nu\left(\dfrac{\partial^2 v}{\partial x^2} + \dfrac{\partial^2 v}{\partial y^2}\right)$



Practical Session: 2D Lid Driven Cavity Flow

Continuity: $\nabla.\vec{v} = 0$

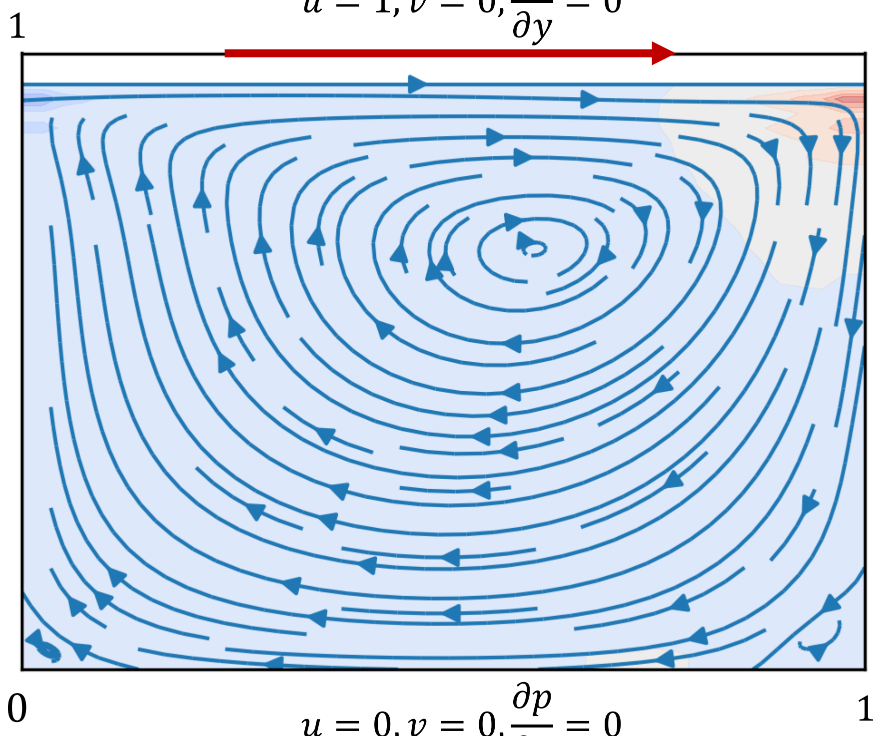
Momentum: $\dfrac{\partial \vec{v}}{\partial t} + (\vec{v}.\nabla)\vec{v} = -\dfrac{1}{\rho}\nabla p + \nu \nabla^2 \vec{v}$

X-velocity: $\dfrac{\partial u}{\partial t} + u\left(\dfrac{\partial u}{\partial x}\right) + v\left(\dfrac{\partial u}{\partial y}\right) = -\dfrac{1}{\rho}\left(\dfrac{\partial p}{\partial x}\right) + \nu\left(\dfrac{\partial^2 u}{\partial x^2} + \dfrac{\partial^2 u}{\partial y^2}\right)$

Y-velocity: $\dfrac{\partial v}{\partial t} + u\left(\dfrac{\partial v}{\partial x}\right) + v\left(\dfrac{\partial v}{\partial y}\right) = -\dfrac{1}{\rho}\left(\dfrac{\partial p}{\partial y}\right) + \nu\left(\dfrac{\partial^2 v}{\partial x^2} + \dfrac{\partial^2 v}{\partial y^2}\right)$

What about the pressure?

$u = 1, v = 0, \dfrac{\partial p}{\partial y} = 0$



$u = 0$, $v = 0$, $\dfrac{\partial p}{\partial x} = 0$ (left wall)

$u = 0$, $v = 0$, $\dfrac{\partial p}{\partial x} = 0$ (right wall)

$u = 0, v = 0, \dfrac{\partial p}{\partial y} = 0$

Practical Session: 2D Lid Driven Cavity Flow

Continuity: $\nabla.\vec{v} = 0$

Momentum: $$\frac{\partial \vec{v}}{\partial t} + (\vec{v}.\nabla)\vec{v} = -\frac{1}{\rho}\nabla p + \nu\nabla^2\vec{v}$$

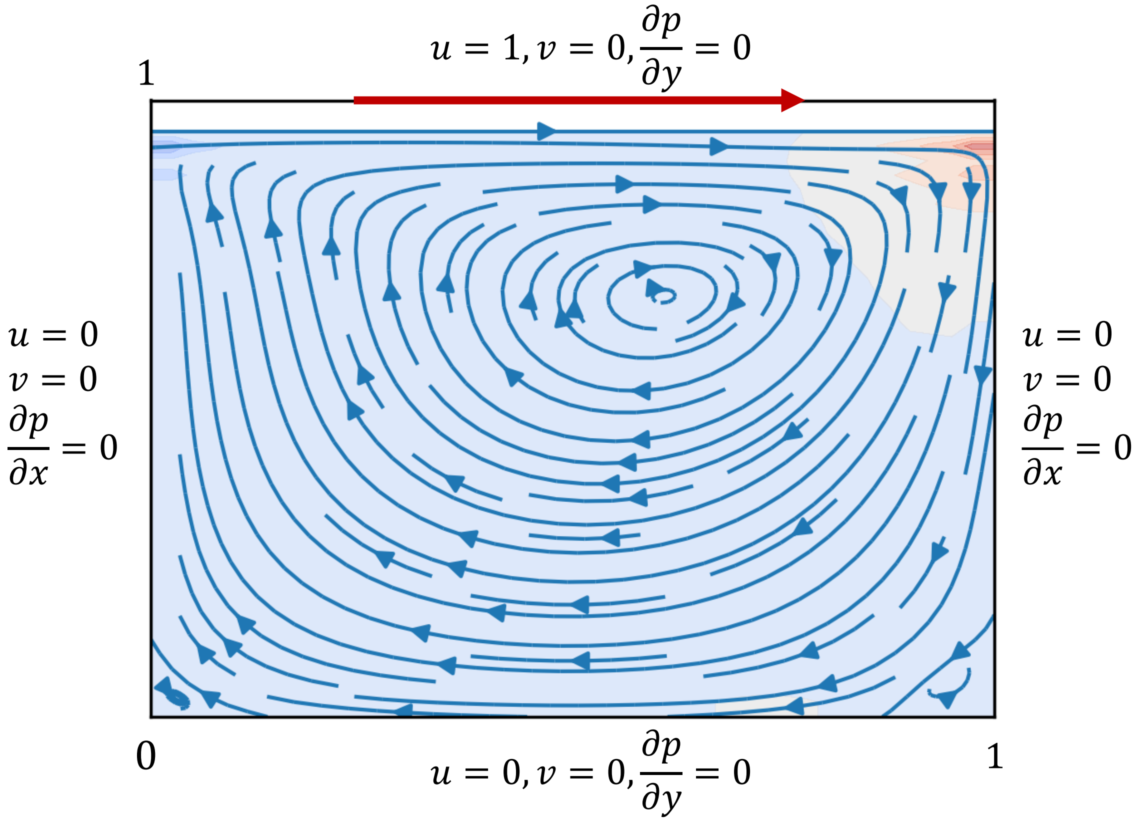
X-velocity: $$\frac{\partial u}{\partial t} + u\left(\frac{\partial u}{\partial x}\right) + v\left(\frac{\partial u}{\partial y}\right) = -\frac{1}{\rho}\left(\frac{\partial p}{\partial x}\right) + \nu\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right)$$

Y-velocity: $$\frac{\partial v}{\partial t} + u\left(\frac{\partial v}{\partial x}\right) + v\left(\frac{\partial v}{\partial y}\right) = -\frac{1}{\rho}\left(\frac{\partial p}{\partial y}\right) + \nu\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right)$$

What about the pressure?

Recall: The pressure Poisson's Equation

$$\nabla^2 p = -\nabla.(\vec{v}.\nabla)\vec{v}$$



Practical Session: 2D Lid Driven Cavity Flow

Continuity: $\nabla . \vec{v} = 0$

Momentum: $\dfrac{\partial \vec{v}}{\partial t} + (\vec{v}.\nabla)\vec{v} = -\dfrac{1}{\rho}\nabla p + \nu \nabla^2 \vec{v}$

X-velocity: $\dfrac{\partial u}{\partial t} + u\left(\dfrac{\partial u}{\partial x}\right) + v\left(\dfrac{\partial u}{\partial y}\right) = -\dfrac{1}{\rho}\left(\dfrac{\partial p}{\partial x}\right) + \nu\left(\dfrac{\partial^2 u}{\partial x^2} + \dfrac{\partial^2 u}{\partial y^2}\right)$

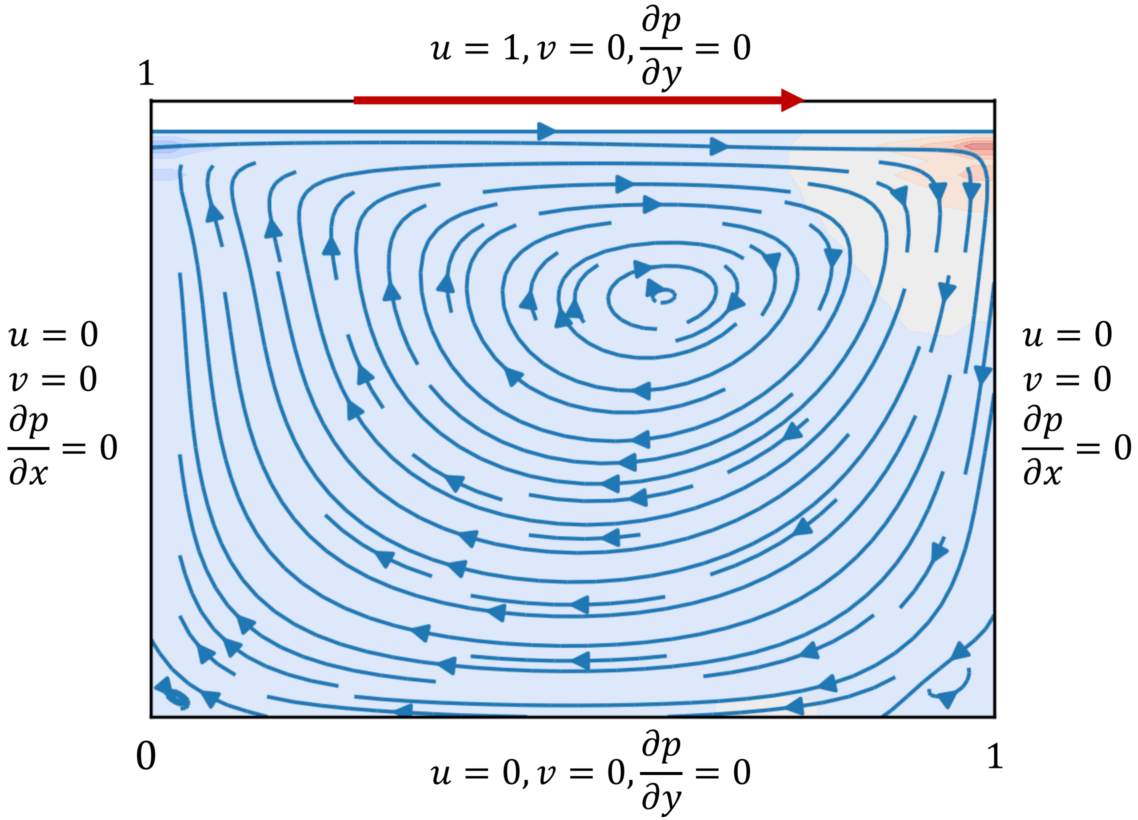
Y-velocity: $\dfrac{\partial v}{\partial t} + u\left(\dfrac{\partial v}{\partial x}\right) + v\left(\dfrac{\partial v}{\partial y}\right) = -\dfrac{1}{\rho}\left(\dfrac{\partial p}{\partial y}\right) + \nu\left(\dfrac{\partial^2 v}{\partial x^2} + \dfrac{\partial^2 v}{\partial y^2}\right)$

What about the pressure?

Recall: The pressure Poisson's Equation

$$\nabla^2 p = -\nabla . (\vec{v}.\nabla)\vec{v}$$

$$\left(\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2}\right) = -\rho\left(\frac{\partial u}{\partial x}\frac{\partial u}{\partial x} + 2\frac{\partial u}{\partial y}\frac{\partial v}{\partial x} + \frac{\partial v}{\partial y}\frac{\partial v}{\partial y}\right)$$



Practical Session: 2D Lid Driven Cavity Flow

Continuity: $\nabla . \vec{v} = 0$

Momentum: $\dfrac{\partial \vec{v}}{\partial t} + (\vec{v}.\nabla)\vec{v} = -\dfrac{1}{\rho}\nabla p + \nu \nabla^2 \vec{v}$

X-velocity: $\dfrac{\partial u}{\partial t} + u\left(\dfrac{\partial u}{\partial x}\right) + v\left(\dfrac{\partial u}{\partial y}\right) = -\dfrac{1}{\rho}\left(\dfrac{\partial p}{\partial x}\right) + \nu\left(\dfrac{\partial^2 u}{\partial x^2} + \dfrac{\partial^2 u}{\partial y^2}\right)$

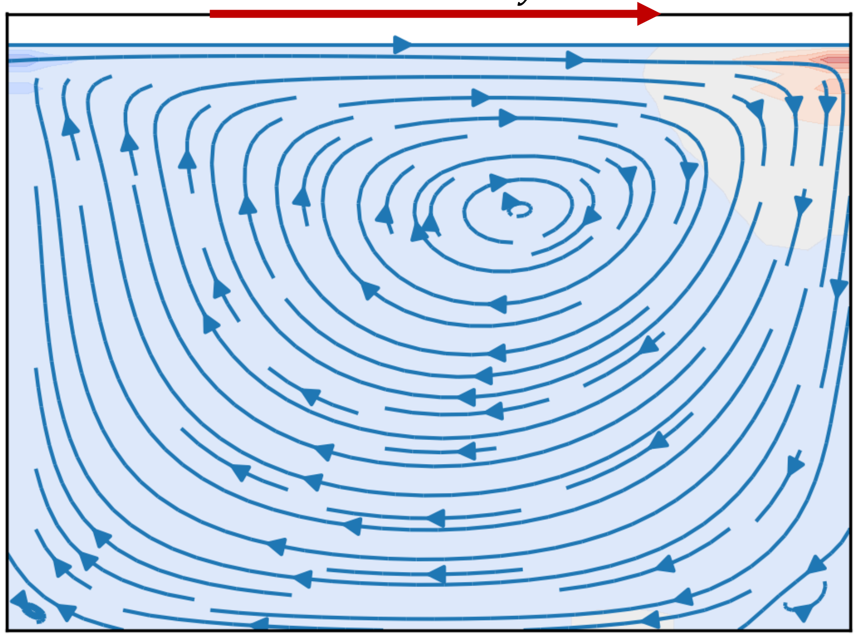
Y-velocity: $\dfrac{\partial v}{\partial t} + u\left(\dfrac{\partial v}{\partial x}\right) + v\left(\dfrac{\partial v}{\partial y}\right) = -\dfrac{1}{\rho}\left(\dfrac{\partial p}{\partial y}\right) + \nu\left(\dfrac{\partial^2 v}{\partial x^2} + \dfrac{\partial^2 v}{\partial y^2}\right)$

What about the pressure?

Recall: The pressure Poisson's Equation

$$\nabla^2 p = -\nabla . (\vec{v}.\nabla)\vec{v}$$

$$\left(\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2}\right) = -\rho\left(\frac{\partial u}{\partial x}\frac{\partial u}{\partial x} + 2\frac{\partial u}{\partial y}\frac{\partial v}{\partial x} + \frac{\partial v}{\partial y}\frac{\partial v}{\partial y}\right)$$



Now it's your turn to discretize the equations, code it & visualize the results!

Practical Session: 2D Lid Driven Cavity Flow

Continuity: $\nabla . \vec{v} = 0$

Momentum: $\dfrac{\partial \vec{v}}{\partial t} + (\vec{v}.\nabla)\vec{v} = -\dfrac{1}{\rho}\nabla p + \nu \nabla^2 \vec{v}$

X-velocity: $\dfrac{\partial u}{\partial t} + u\left(\dfrac{\partial u}{\partial x}\right) + v\left(\dfrac{\partial u}{\partial y}\right) = -\dfrac{1}{\rho}\left(\dfrac{\partial p}{\partial x}\right) + \nu\left(\dfrac{\partial^2 u}{\partial x^2} + \dfrac{\partial^2 u}{\partial y^2}\right)$

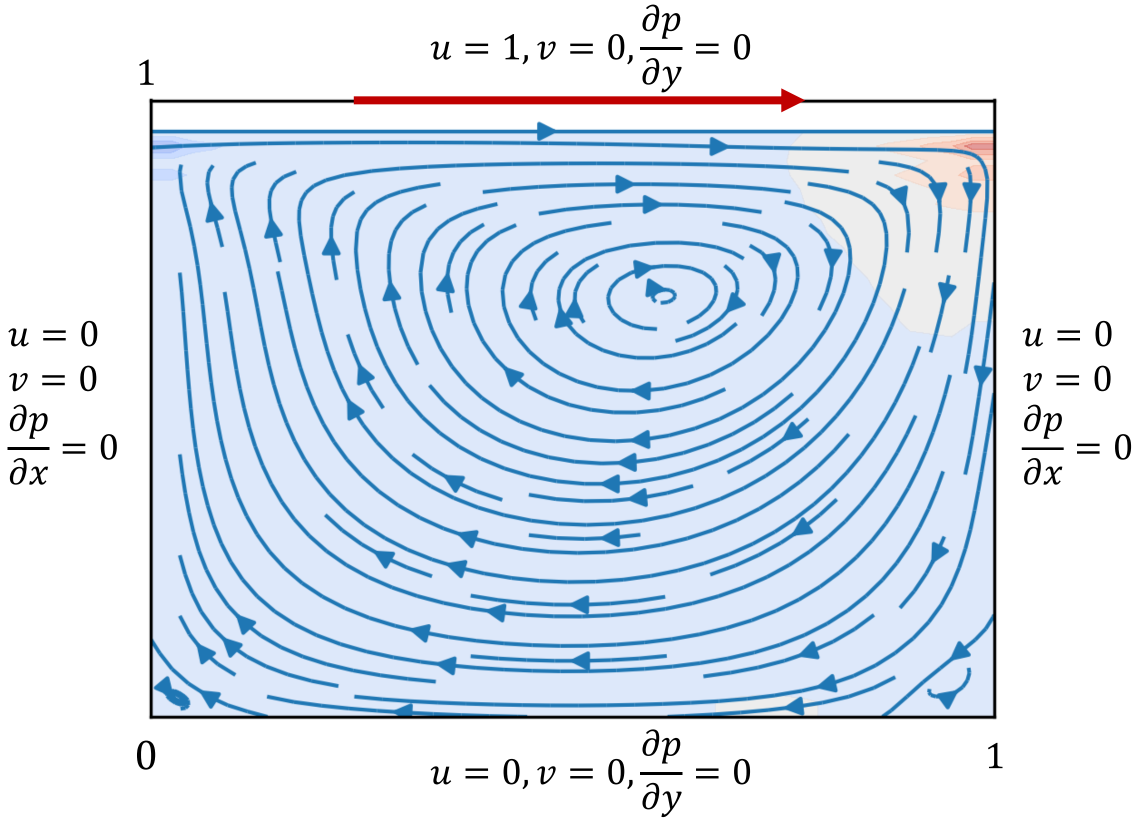
Y-velocity: $\dfrac{\partial v}{\partial t} + u\left(\dfrac{\partial v}{\partial x}\right) + v\left(\dfrac{\partial v}{\partial y}\right) = -\dfrac{1}{\rho}\left(\dfrac{\partial p}{\partial y}\right) + \nu\left(\dfrac{\partial^2 v}{\partial x^2} + \dfrac{\partial^2 v}{\partial y^2}\right)$

What about the pressure?

Recall: The pressure Poisson's Equation

$$\nabla^2 p = -\nabla . (\vec{v}.\nabla)\vec{v}$$

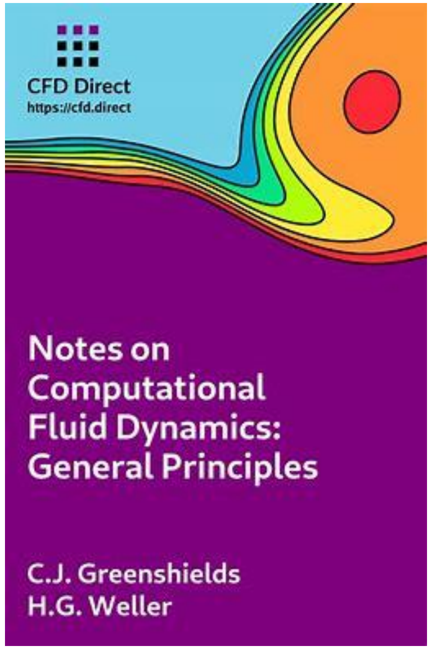
$$\left(\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2}\right) = -\rho\left(\frac{\partial u}{\partial x}\frac{\partial u}{\partial x} + 2\frac{\partial u}{\partial y}\frac{\partial v}{\partial x} + \frac{\partial v}{\partial y}\frac{\partial v}{\partial y}\right)$$

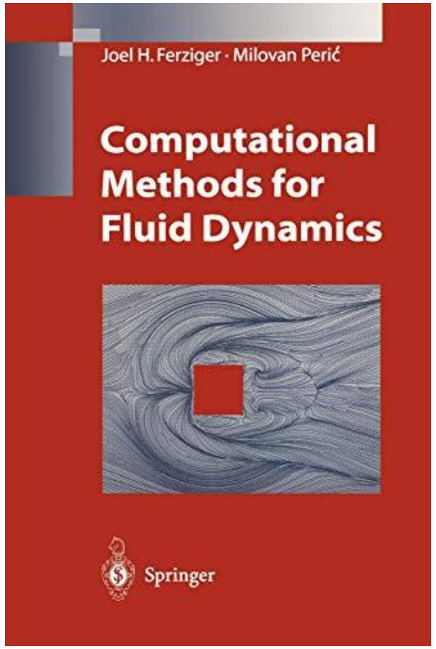


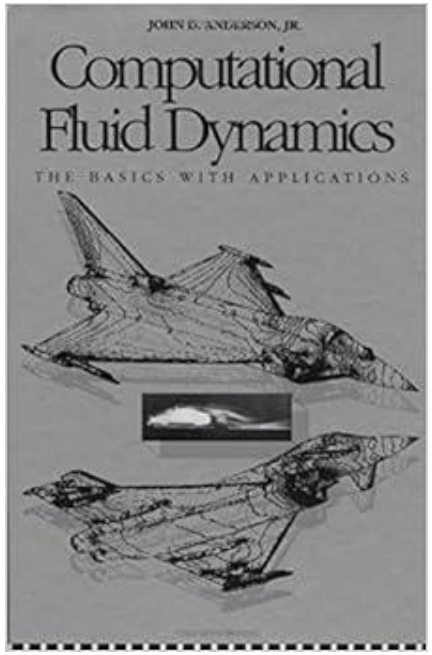
Now it's your turn to discretize the equations, code it & visualize the results!

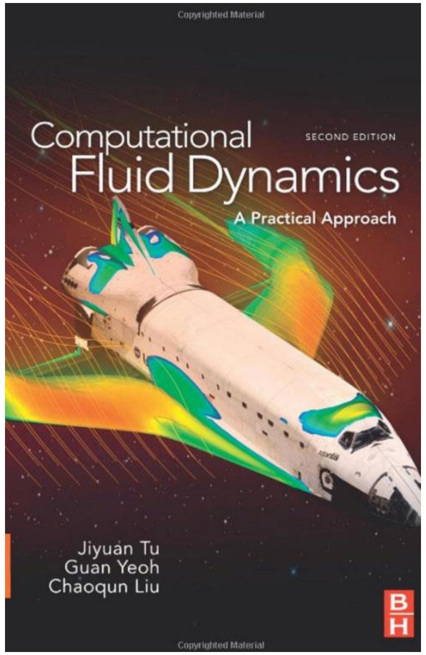
Good Luck!

References









CFD Python: 12 steps to Navier-Stokes :: Lorena A. Barba Group (lorenabarba.com)

Huge Thanks to Marcial Sanchis, Francesco Mario D'Afiero, Gerardo Zampino, Abhijeet Vishwasrao, and Ricardo Vinuesa for their feedback on this material.

FLOW SeRC Swedish e-Science Research Centre MODELAIR VINUESA LAB umair-kth/ModelAir_CFD (github.com)