Project presentation

Topic: **Fractional Fourier neural operator on partial differential equations**

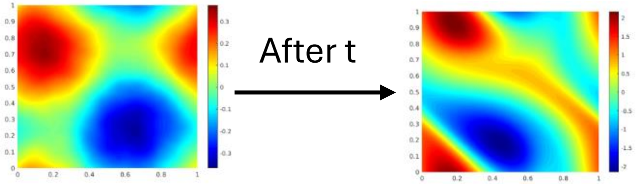
Student: Suwei Yang

Advisor: Shandian Zhe

Outline

- **Problem definition**
- Motivation
- What did I do?
- Result
- Conclusion

Problem definition



$$\frac{\partial u}{\partial t} + \alpha \frac{\partial u}{\partial x} = 0, \quad x \in [a,b], t > 0$$

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

$$u = u(x, t)$$

Nonlinear map $\quad G^{\dagger} : A \rightarrow U$

Goal $\quad G_{\theta} : A \rightarrow U$

Observation $\quad A \subset R^{d_a} \quad U \subset R^{d_u}$

$\{a_j \ u_j\}_{j=1}^{N}$ and a_j is i. i. d sequence from probability , μ, measurement.

u_j may be corrupted with noise.

Method $\quad G : A \times \Theta \rightarrow U \quad \min_{\theta \in \Theta} E_{a \sim \mu}[cost(G_{\theta}(a, \theta), G^{\dagger}(a))]$

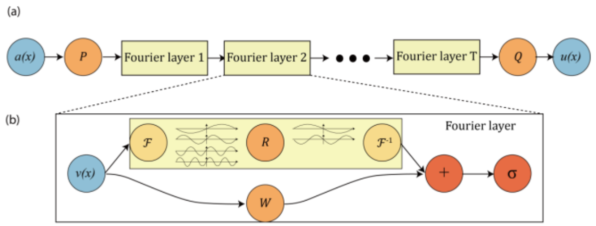
$\theta \in \Theta$

Ref: Fourier Neural Operator for Parametric Partial Differential Equations

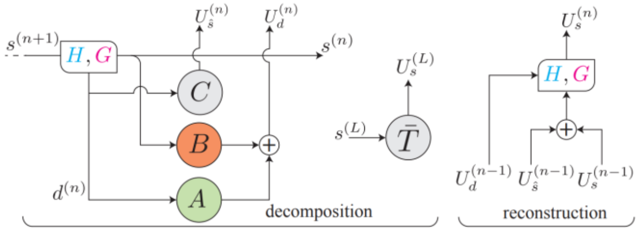
Problem definition

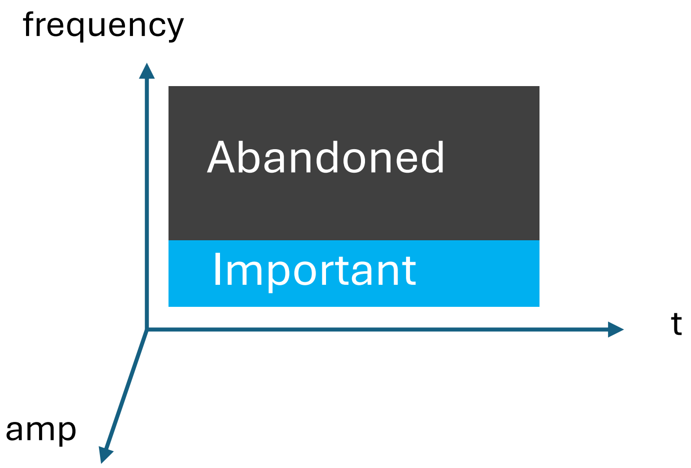
People have done...

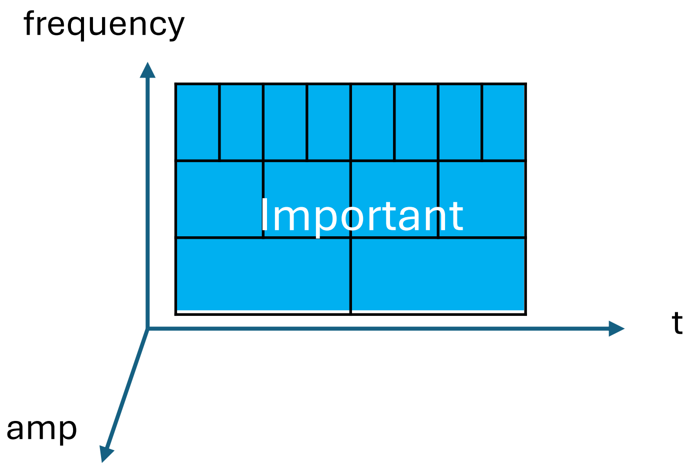
Fourier transform(FNO)



Wavelet transform (WNO)







Comparison:

FNO:

Fast

No time information

WNO:

Slow

With time information

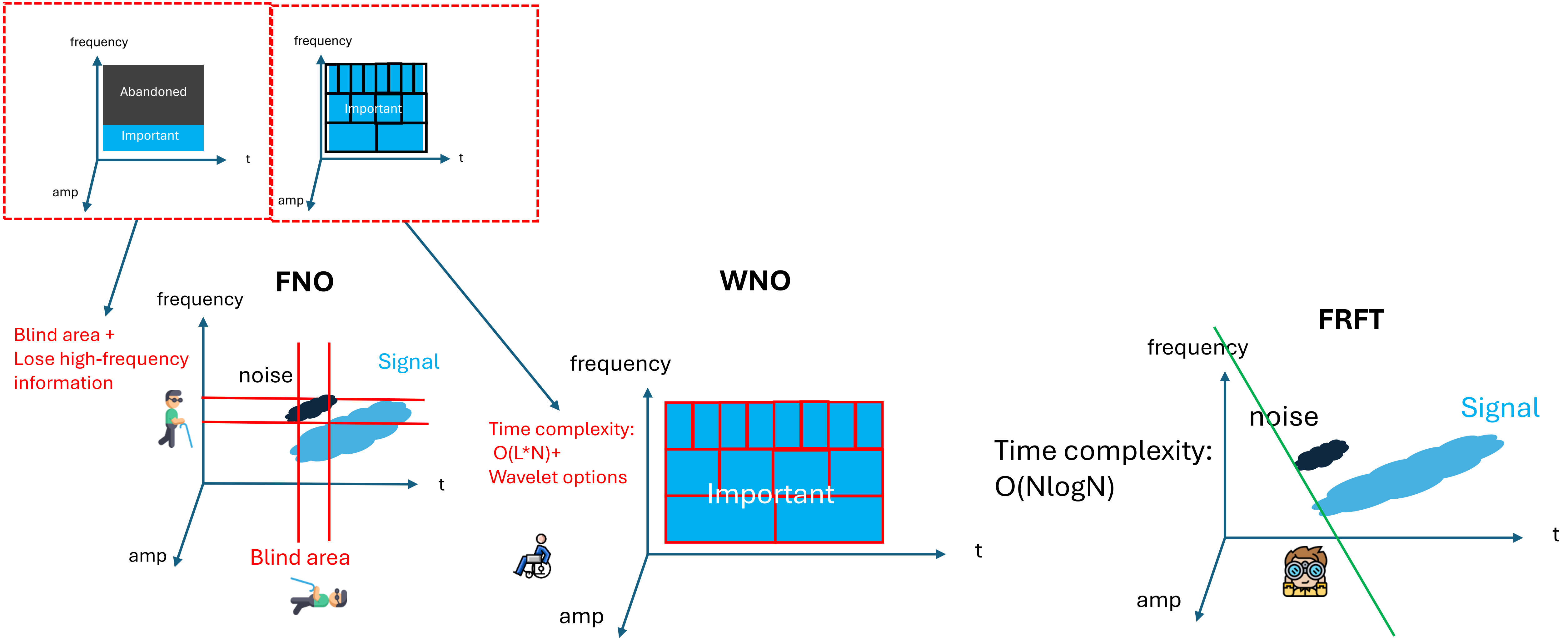
Fractional Fourier transform (FRFT)

Ref: Fourier Neural Operator for Parametric Partial Differential Equations
Multiwavelet-based Operator Learning for Differential Equations

Outline

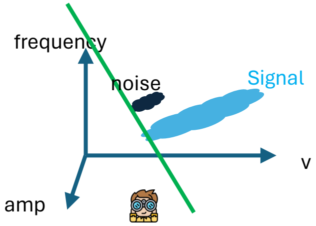
- Problem definition
- **Motivation**
- What did I do?
- Result
- Conclusion

Motivation



Ref: Optimal Filtering in Fractional Fourier Domains

Motivation



and $\{\mathcal{F}^4 f\}(x) = f(x)$. The ath-order fractional Fourier transform $\{\mathcal{F}^a f\}(x)$ of the function $f(x)$ may be defined for $0 < |a| < 2$ as

$$\boxed{\mathcal{F}^a[f(x)] \equiv \{\mathcal{F}^a f\}(x) \equiv \int_{-\infty}^{\infty} B_a(x, x') f(x')\, dx',}$$

$$B_a(x, x') \equiv A_\phi \exp\left[i\pi(x^2 \cot\phi - 2xx' \csc\phi + x'^2 \cot\phi)\right],$$

$$A_\phi \equiv \frac{\exp\left(-i\pi\, \mathrm{sgn}\,(\sin\phi)/4 + i\phi/2\right)}{|\sin\phi|^{1/2}} \tag{1}$$

where

$$\phi \equiv \frac{a\pi}{2} \tag{2}$$

and i is the imaginary unit. The kernel approaches $B_0(x, x') \equiv \delta(x - x')$ and $B_{\pm 2}(x, x') \equiv \delta(x + x')$ for $a = 0$ and $a = \pm 2$, respectively. The definition is easily extended outside

In this approach, we assume $a \in [-1, 1]$. Manipulating (1), we can write

$$f_a(x) = \exp\left[-i\pi x^2 \tan(\phi/2)\right] g'(x), \tag{14}$$

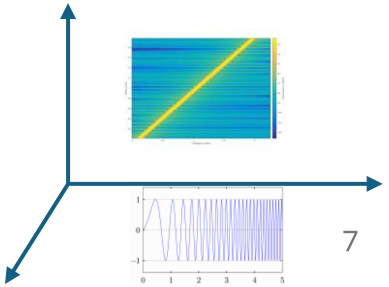
$$g'(x) = A_\phi \int_{-\infty}^{\infty} \exp\left[i\pi\beta(x - x')^2\right] g(x')\, dx', \tag{15}$$

$$g(x) = \exp\left[-i\pi x^2 \tan(\phi/2)\right] f(x) \tag{16}$$

where $g(x)$ and $g'(x)$ represent intermediate results, and $\beta = \csc\phi$.

$$\mathbf{Signal_\alpha = Chirp * Conv(Chirp, Chirp * signal_t)}$$

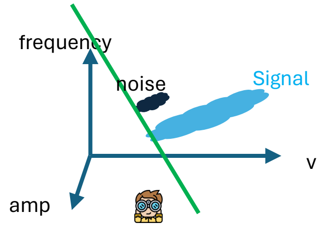
Time Complexity: NlogN



Ref: Optimal Filtering in Fractional Fourier Domains
Digital Computation of the Fractional Fourier Transform

Outline

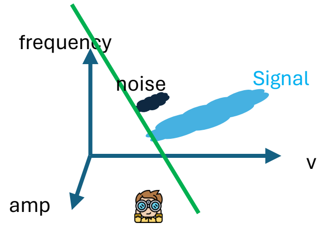
- Problem definition
- Motivation
- **What did I do?**
- Result
- Conclusion



What did I do?

- **Find cases**
  - High-frequency + Non-stationary dataset
- Prove it works
  - Applying FRFT to it

What did I do?



- **Find cases**
  - High-frequency + Non-stationary dataset
- Prove it works
  - Applying FRFT to it

$$(\mathbf{k^2} - \mathbf{\nabla H(s) \nabla}) \mathbf{u(s)} = \mathbf{W(s)}$$

$$k > 0, s \in D = [A_1, B_1] \times [A_2, B_2] \subset R^2,$$

$$\nabla = [\frac{\partial}{\partial s_1}, \frac{\partial}{\partial s_2}]$$

$$H(s) = \text{function with vector field},$$

$$u(s) = \text{nonstationary dataset},$$

$$W(s) = \text{standard gaussian white noise}$$

When H(s) is a constant,
u(s) is a **stationary dataset.**
When H(s) has vector fields,
u(s) is a **non-stationary dataset.**

$$H(s) = \gamma I_2 + Bv(s)v(s)^T, \gamma, B > 0$$

```python
def H_s(s1,s2):
    def vs(s1,s2):
        #vector field, the place creates nonstationary property
        v1 = 1.2*torch.sin(torch.tensor(2*torch.pi*10*s2*s2)*torch.cos(torch.tensor(s1*s2)))#
        v2 = 1.4*torch.sin(torch.tensor(2*torch.pi*5*s2*s2))+torch.cos(torch.tensor(s1*s2))#1
        v = torch.tensor([v1,v2])
        return v

    #H(s) = r*I2 + B*V(s)V(s)
    r, B = 1, 100
    I, V = torch.eye(2), vs(s1,s2)
    H = r*I +B*V.reshape(2,1) @ V.reshape(1,2)
    return H
```

$$\mathbf{u(s)} \propto \mathbf{exp}\left(-\frac{\mathbf{1}}{\mathbf{2}} \mathbf{s^T Q s}\right)$$

$$Q = A^T D_v A$$

$$A = D_v D_{k^2} - A_H$$

$$A_H \in R^{MN \times MN}$$

$$(\mathbf{A_H})_{jM+i,jM+i} = - \frac{h_y}{h_x} \left[H^{11}(\boldsymbol{s}_{i+1/2,j}) + H^{11}(\boldsymbol{s}_{i-1/2,j})\right] - \frac{h_x}{h_y} \left[H^{22}(\boldsymbol{s}_{i,j+1/2}) + H^{22}(\boldsymbol{s}_{i,j-1/2})\right].$$

The four closest neighbours have coefficients

$$(\mathbf{A_H})_{jM+i,jM+i_p} = \frac{h_y}{h_x} H^{11}(s_{i-1/2,j}) - \frac{1}{4}\left[H^{12}(\boldsymbol{s}_{i,j+1/2}) - H^{12}(\boldsymbol{s}_{i,j-1/2})\right],$$

$$(\mathbf{A_H})_{jM+i,jM+i_n} = \frac{h_y}{h_x} H^{11}(\boldsymbol{s}_{i+1/2,j}) + \frac{1}{4}\left[H^{12}(\boldsymbol{s}_{i,j+1/2}) - H^{12}(\boldsymbol{s}_{i,j-1/2})\right],$$

$$(\mathbf{A_H})_{jM+i,j_nM+i} = \frac{h_x}{h_y} H^{22}(\boldsymbol{s}_{i,j+1/2}) + \frac{1}{4}\left[H^{21}(\boldsymbol{s}_{i+1/2,j}) - H^{21}(\boldsymbol{s}_{i-1/2,j})\right],$$

$$(\mathbf{A_H})_{jM+i,j_pM+i} = \frac{h_x}{h_y} H^{22}(\boldsymbol{s}_{i,j-1/2}) - \frac{1}{4}\left[H^{21}(\boldsymbol{s}_{i+1/2,j}) - H^{21}(\boldsymbol{s}_{i-1/2,j})\right].$$

Lastly, the four diagonally closest neighbours have coefficients

$$(\mathbf{A_H})_{jM+i,j_pM+i_p} = +\frac{1}{4}\left[H^{12}(\boldsymbol{s}_{i,j-1/2}) + H^{21}(\boldsymbol{s}_{i-1/2,j})\right],$$

$$(\mathbf{A_H})_{jM+i,j_pM+i_n} = -\frac{1}{4}\left[H^{12}(\boldsymbol{s}_{i,j-1/2}) + H^{21}(\boldsymbol{s}_{i+1/2,j})\right],$$

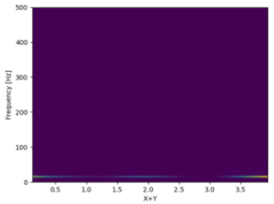
$$(\mathbf{A_H})_{jM+i,j_nM+i_p} = -\frac{1}{4}\left[H^{12}(\boldsymbol{s}_{i,j+1/2}) + H^{21}(\boldsymbol{s}_{i-1/2,j})\right],$$

$$(\mathbf{A_H})_{jM+i,j_nM+i_n} = +\frac{1}{4}\left[H^{12}(\boldsymbol{s}_{i,j+1/2}) + H^{21}(\boldsymbol{s}_{i+1/2,j})\right].$$

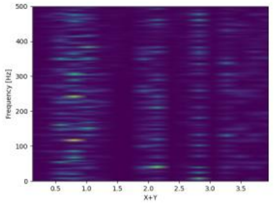
The rest of the elements of row $jM + i$ are 0.

Ref: Exploring a New Class of Non-stationary Spatial Gaussian Random Fields with Varying Local Anisotropy

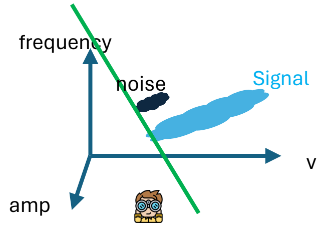
What did I do?



Low frequency + stationery



High frequency + non-stationery



- **Find cases**
  - High-frequency + Non-stationary dataset
- Prove it works
  - Applying FRFT to it

$$(\mathbf{k^2} - \mathbf{\nabla H(s) \nabla})\mathbf{u(s)} = \mathbf{W(s)}$$

$$k > 0, s \in D = [A_1, B_1] \times [A_2, B_2] \subset R^2,$$

$$\nabla = [\frac{\partial}{\partial s_1}, \frac{\partial}{\partial s_2}]$$

$$H(s) = \text{function with vector field,}$$

$$u(s) = \text{nonstationary dataset,}$$

$$W(s) = \text{standard gaussian white noise}$$

When H(s) is a constant,
u(s) is a **stationary dataset.**
When H(s) has vector fields,
u(s) is a **non-stationary dataset.**

$$H(s) = \gamma I_2 + Bv(s)v(s)^T, \gamma, B > 0$$

```
def H_s(s1,s2):
    def vs(s1,s2):
        #vector field, the place creates nonstationary property
        v1 = 1.2*torch.sin(torch.tensor(2*torch.pi*10*s2*s2)*torch.cos(torch.tensor(s1*s2)))#
        v2 = 1.4*torch.sin(torch.tensor(2*torch.pi*5*s2*s2))+torch.cos(torch.tensor(s1*s2))#1
        v = torch.tensor([v1,v2])
        return v

    #H(s) = r*I2 + B*V(s)V(s)
    r, B = 1, 100
    I, V = torch.eye(2), vs(s1,s2)
    H = r*I +B*V.reshape(2,1) @ V.reshape(1,2)
    return H
```

Gets:
1. High frequency when s2 goes up
2. Non-stationary with s2

$$\mathbf{u(s)} \propto \mathbf{exp}\left(-\frac{\mathbf{1}}{\mathbf{2}}\mathbf{s^T Q s}\right)$$

$$Q = A^T D_v A$$

$$A = D_v D_{k^2} - A_H$$

$$A_H \in R^{MN \times MN}$$

$$(\mathbf{A_H})_{jM+i,jM+i} = -\frac{h_y}{h_x}\left[H^{11}(\boldsymbol{s}_{i+1/2,j}) + H^{11}(\boldsymbol{s}_{i-1/2,j})\right] - \frac{h_x}{h_y}\left[H^{22}(\boldsymbol{s}_{i,j+1/2}) + H^{22}(\boldsymbol{s}_{i,j-1/2})\right].$$

The four closest neighbours have coefficients

$$(\mathbf{A_H})_{jM+i,jM+i_p} = \frac{h_y}{h_x}H^{11}(\boldsymbol{s}_{i-1/2,j}) - \frac{1}{4}\left[H^{12}(\boldsymbol{s}_{i,j+1/2}) - H^{12}(\boldsymbol{s}_{i,j-1/2})\right],$$

$$(\mathbf{A_H})_{jM+i,jM+i_n} = \frac{h_y}{h_x}H^{11}(\boldsymbol{s}_{i+1/2,j}) + \frac{1}{4}\left[H^{12}(\boldsymbol{s}_{i,j+1/2}) - H^{12}(\boldsymbol{s}_{i,j-1/2})\right],$$

$$(\mathbf{A_H})_{jM+i,j_nM+i} = \frac{h_x}{h_y}H^{22}(\boldsymbol{s}_{i,j+1/2}) + \frac{1}{4}\left[H^{21}(\boldsymbol{s}_{i+1/2,j}) - H^{21}(\boldsymbol{s}_{i-1/2,j})\right],$$

$$(\mathbf{A_H})_{jM+i,j_pM+i} = \frac{h_x}{h_y}H^{22}(\boldsymbol{s}_{i,j-1/2}) - \frac{1}{4}\left[H^{21}(\boldsymbol{s}_{i+1/2,j}) - H^{21}(\boldsymbol{s}_{i-1/2,j})\right].$$

Lastly, the four diagonally closest neighbours have coefficients

$$(\mathbf{A_H})_{jM+i,j_pM+i_p} = +\frac{1}{4}\left[H^{12}(\boldsymbol{s}_{i,j-1/2}) + H^{21}(\boldsymbol{s}_{i-1/2,j})\right],$$

$$(\mathbf{A_H})_{jM+i,j_pM+i_n} = -\frac{1}{4}\left[H^{12}(\boldsymbol{s}_{i,j-1/2}) + H^{21}(\boldsymbol{s}_{i+1/2,j})\right],$$

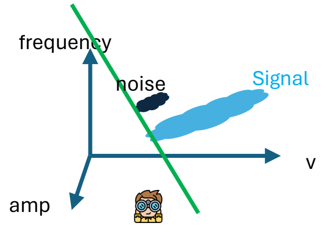
$$(\mathbf{A_H})_{jM+i,j_nM+i_p} = -\frac{1}{4}\left[H^{12}(\boldsymbol{s}_{i,j+1/2}) + H^{21}(\boldsymbol{s}_{i-1/2,j})\right],$$

$$(\mathbf{A_H})_{jM+i,j_nM+i_n} = +\frac{1}{4}\left[H^{12}(\boldsymbol{s}_{i,j+1/2}) + H^{21}(\boldsymbol{s}_{i+1/2,j})\right].$$

The rest of the elements of row $jM + i$ are 0.

Ref: Exploring a New Class of Non-stationary Spatial Gaussian Random Fields with Varying Local Anisotropy

What did I do?



- **Find cases**
  - High-frequency + Non-stationary dataset
- Prove it works
  - Applying FRFT to it

$$\frac{\partial u}{\partial t} + \alpha \frac{\partial u}{\partial x} = 0, \quad x \in [a, b], t > 0$$

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

$$u = u(x, t)$$

$$\mathbf{u}(s, \mathbf{0}) \propto \mathbf{exp}\left(-\frac{\mathbf{1}}{\mathbf{2}} \mathbf{s}^{\mathbf{T}} \mathbf{Q} \mathbf{s}\right)$$

Add it as the initial condition

$$\text{Burger's equation:} \quad \frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} = \frac{v \partial u}{\partial x^2},$$

$$\text{where } v = \text{viscosity}$$

Goal: u(x,0)-> u(x,t)

$$\text{Poisson equation:} \quad k\nabla^2 u + f = 0,$$

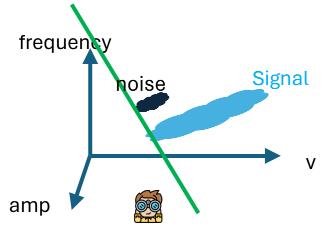
$$\text{where } \nabla = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}\right) \text{ and } f = \text{given function}$$

Goal: f(x,y)-> u(x,y)

$$\text{Wave equation:} \quad \frac{\partial}{\partial x}\left(\frac{\partial c^2 u}{\partial x}\right) + \frac{\partial}{\partial y}\left(\frac{\partial c^2 u}{\partial y}\right) = \frac{\partial^2 u}{\partial t^2},$$

$$\text{where } c = \text{wave velocity}$$

Goal: u(x,y,0)-> u(x,y,t)



What did I do?

- **Find cases**
  - High-frequency + Non-stationary dataset
- Prove it works
  - Applying FRFT to it

$$\frac{\partial u}{\partial t} + \alpha \frac{\partial u}{\partial x} = 0, \quad x \in [a, b], t > 0$$

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

$$u = u(x, t)$$

$$\mathbf{u}(s, \mathbf{0}) \propto \mathbf{exp}\left(-\frac{\mathbf{1}}{\mathbf{2}}\mathbf{s}^{\mathrm{T}}\mathbf{Q}\mathbf{s}\right)$$

Add it as the initial condition

$$\text{Burger's equation:} \quad \frac{\partial \mathrm{u}}{\partial \mathrm{t}} + \mathrm{u}\frac{\partial \mathrm{u}}{\partial \mathrm{x}} = \frac{\mathrm{v}\partial \mathrm{u}}{\partial \mathrm{x}^2},$$

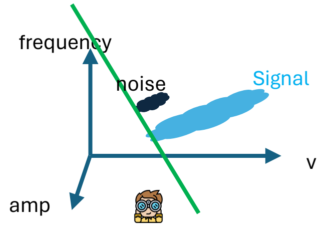
$$\text{where } \mathrm{v} = \text{viscosity}$$

Goal: u(x,0)-> u(x,t)

$\mathit{Two\ cases}$:

$v = 0$, $\mathit{inviscid\ Burgers'\ equation}$

$v \neq 0$, viscous Burgers' equation



What did I do?

- **Find cases**
  - High-frequency + Non-stationary dataset
- Prove it works
  - Applying FRFT to it

$$\frac{\partial u}{\partial t} + \alpha \frac{\partial u}{\partial x} = 0, \quad x \in [a, b], t > 0$$

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

$$u = u(x, t)$$

$$\mathbf{u}(s, \mathbf{0}) \propto \mathbf{exp}\left(-\frac{\mathbf{1}}{\mathbf{2}} \mathbf{s}^{\mathbf{T}} \mathbf{Q} \mathbf{s}\right)$$

Add it as the initial condition

Burger′s equation: $\frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} = \frac{v \partial u}{\partial x^2}$,

where v = viscosity

Goal: u(x,0)-> u(x,t)

Two cases:

$\mathbf{v = 0}$, **inviscid Burgers′ equation** (upwind scheme)

$v \neq 0$, viscous Burgers′ equation

$$\frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} = 0, \text{introduce } f(u) = \frac{u^2}{2}$$

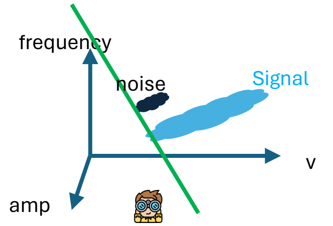
$$\Rightarrow \frac{\partial u}{\partial t} + \frac{\partial f}{\partial u} \frac{\partial u}{\partial x} = 0$$

$$\Rightarrow \frac{\partial u}{\partial t} + \frac{\partial f}{\partial x} = 0$$

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

Where n: time point, and j: x points

Ref: https://github.com/gilbertfrancois/partial-differential-equations/blob/master/notebook/1D%20Burgers%27%20equation%2C%20finite%20volume%2C%20upwind%20scheme.ipynb



What did I do?

- **Find cases**
  - High-frequency + Non-stationary dataset
- Prove it works
  - Applying FRFT to it

$$\frac{\partial u}{\partial t} + \alpha \frac{\partial u}{\partial x} = 0, \quad x \in [a,b], t > 0$$

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

$$u = u(x,t)$$

$$\mathbf{u}(s,\mathbf{0}) \propto \mathbf{exp}\left(-\frac{1}{2}\mathbf{s}^{\mathrm{T}}\mathbf{Q}\mathbf{s}\right)$$

Add it as the initial condition

```python
def dudt_upwind(u_bar, t, dx):
    f_interface[n] = 0

    # Compute the time derivative as the difference of
    dudt = (f_interface[0:n] - f_interface[1:n+1])/dx
    return dudt
```

Burger's equation: $\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \frac{v\partial u}{\partial x^2}$,

where v = viscosity

Goal: u(x,0)-> u(x,t)

Two cases:

$v = 0$, inviscid Burgers' equation (upwind scheme)

$v \neq 0$, viscous Burgers' equation

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = 0, \text{introduce } f(u) = \frac{u^2}{2}$$

$$\Rightarrow \frac{\partial u}{\partial t} + \frac{\partial f}{\partial u}\frac{\partial u}{\partial x} = 0$$

$$\Rightarrow \frac{\partial u}{\partial t} + \frac{\partial f}{\partial x} = 0$$

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

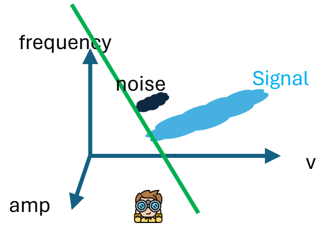
Where n: time point, and j: x points

```python
def Burger(n, timestep):

    #integral from initial condition
    u = odeint(dudt_upwind, u_init, t, args=(dx,))
```


Ref: https://github.com/gilbertfrancois/partial-differential-equations/blob/master/notebook/1D%20Burgers%27%20equation%2C%20finite%20volume%2C%20upwind%20scheme.ipynb

What did I do?



- **Find cases**
  - High-frequency + Non-stationary dataset
- Prove it works
  - Applying FRFT to it

$$\frac{\partial u}{\partial t} + \alpha \frac{\partial u}{\partial x} = 0, \quad x \in [a, b], t > 0$$

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

$$u = u(x, t)$$

$$\mathbf{u}(s, \mathbf{0}) \propto \mathbf{exp}\left(-\frac{\mathbf{1}}{\mathbf{2}}\mathbf{s}^{\mathbf{T}}\mathbf{Q}\mathbf{s}\right)$$

Add it as the initial condition

$$\text{Burger's equation:} \quad \frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \frac{v\partial u}{\partial x^2},$$

where $v = \text{viscosity}$

Goal: u(x,0)-> u(x,t)

Two cases:

$v = 0$, inviscid Burgers' equation

$v \neq 0$, viscous Burgers' equation (applying FFT)

$$u_t + uu_x = vu_{xx}$$

After FFT

$$\Rightarrow \hat{u}_t + u(ik * \hat{u}(w, t)) = v(-k^2\hat{u}(w, t))$$

where $w = x$ in fourier domain,

$\hat{u} =$ u in fourier domain,

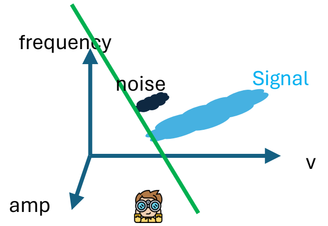
$k = 2\pi f$

After IFFT

$$\Rightarrow u_t = F^{-1}\left(v\left(-k^2\hat{u}(w, t)\right)\right) - F^{-1}\left(u\left(ik * \hat{u}(w, t)\right)\right)$$

$$\Rightarrow u(x, t) = \int F^{-1}\left(v\left(-k^2\hat{u}(w, t)\right)\right) - F^{-1}\left(u\left(ik * \hat{u}(w, t)\right)\right) dt$$

Ref: https://github.com/sachabinder/Burgers_equation_simulation/blob/main/Burgers_solver_SP.py



What did I do?

- **Find cases**
  - High-frequ
- Prove it work
  - Applying F

```
#Definition of ODE system (PDE ---(FFT
def burg_system(u,t,k,mu,nu):
    #Spatial derivative in the Fourier
    u_hat = np.fft.fft(u)
    u_hat_x = 1j*k*u_hat
    u_hat_xx = -k**2*u_hat

    #Switching in the spatial domain
    u_x = np.fft.ifft(u_hat_x)
    u_xx = np.fft.ifft(u_hat_xx)

    #ODE resolution
    u_t = -mu*u*u_x + nu*u_xx
    return u_t.real
```

$$\frac{\partial u}{\partial t} + \alpha \frac{\partial u}{\partial x} = 0, \quad x$$

$a = u(x,$

$u = u(x,t)$

$$\mathbf{u}(s,\mathbf{0}) \propto \mathbf{exp}\left(-\frac{\mathbf{1}}{\mathbf{2}}\mathbf{s}^{\mathbf{T}}\mathbf{Q}\mathbf{s}\right)$$

Add it as the initial condition

Burger's equation: $\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \frac{v\partial u}{\partial x^2}$,

```
#Def of the initial condition
u0 = NSGRF(N_x,2)#np.exp(-(X-3)**2/2) #Single space variable fonction th
# viz_tools.plot_a_frame_1D(X,u0,0,L_x,0,1.2,'Initial condition')
#PDE resolution (ODE system resolution)
U = odeint(burg_system, u0, T, args=(k,mu,nu,),mxstep=5000, hmin=1e-50)#
```

Two cases:

$v = 0$, inviscid Burgers' equation

$v \neq 0$, viscous Burgers' equation (applying FFT)

$u_t + uu_x = vu_{xx}$

After FFT

$\Rightarrow \hat{u}_t + u(ik * \hat{u}(w,t)) = v(-k^2\hat{u}(w,t))$

where $w = x$ in fourier domain,

$\hat{u} =$ u in fourier domain,

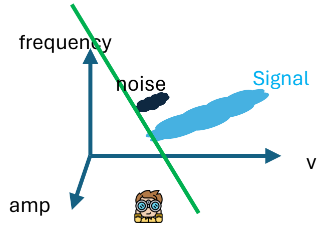
$k = 2\pi f$

After IFFT

$\Rightarrow u_t = F^{-1}\left(v\left(-k^2\hat{u}(w,t)\right)\right) - F^{-1}\left(u\left(ik * \hat{u}(w,t)\right)\right)$

$\Rightarrow u(x,t) = \int F^{-1}\left(v\left(-k^2\hat{u}(w,t)\right)\right) - F^{-1}\left(u\left(ik * \hat{u}(w,t)\right)\right) dt$

Ref: https://github.com/sachabinder/Burgers_equation_simulation/blob/main/Burgers_solver_SP.py



What did I do?

- **Find cases**
  - High-frequency + Non-stationary dataset
- Prove it works
  - Applying FRFT to it

$$\frac{\partial u}{\partial t} + \alpha \frac{\partial u}{\partial x} = 0, \quad x \in [a, b], t > 0$$

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

$$u = u(x, t)$$

$$\mathbf{u}(s, \mathbf{0}) \propto \mathbf{exp}\left(-\frac{\mathbf{1}}{\mathbf{2}}\mathbf{s}^{\mathrm{T}}\mathbf{Q}\mathbf{s}\right)$$

Add it as the initial condition

Poisson equation: $k\nabla^2 u + f = 0,$

where $\nabla = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}\right)$ and $f =$ given function, k is a constant

Goal: f(x,y)-> u(x,y)

Applying finite difference method:

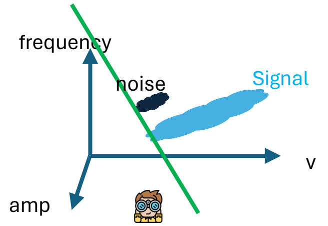
$$\frac{\partial^2 u}{\partial x^2} \cong \frac{\frac{u_{i+1,j} - u_{i,j}}{\Delta x} - \frac{u_{i,j} - u_{i-1,j}}{\Delta x}}{\Delta x} = \frac{u_{i+1,j} + u_{i-1,j} - 2u_{i,j}}{\Delta x^2}$$

$$\frac{\partial^2 u}{\partial y^2} \cong \frac{\frac{u_{i,j+1} - u_{i,j}}{\Delta x} - \frac{u_{i,j} - u_{i,j-1}}{\Delta x}}{\Delta y} = \frac{u_{i,j+1} + u_{i,j-1} - 2u_{i,j}}{\Delta y^2}$$

$Au = -f$, where

A= derivative matrix, u=solution, -f = given function

Ref: https://github.com/gilbertfrancois/partial-differential-equations/blob/master/notebook/2D%20Poisson%20equation%2C%20finite%20difference.ipynb



What did I do?

- **Find cases**
  - High-frequency + Non-stationary dataset
- Prove it works
  - Applying FF

```
def Poision2Dgen(numx,numy):
    # Take the inner nodes of F(x
    b = -F[1:-1,1:-1]
    # Reshape b into a 1D vector
    b = b.reshape(-1)
    # Solve the PDE
    u = np.linalg.solve(A, b)
    # Reshape the solution u to 2
```

$$\frac{\partial u}{\partial t} + \alpha \frac{\partial u}{\partial x} = 0, \quad x$$

$$a = u(x, ($$

$$u = u(x, t)$$

$$\mathbf{u}(s, \mathbf{0}) \propto \exp\left(-\frac{1}{2}\mathbf{s}^{\mathrm{T}}\mathbf{Q}\mathbf{s}\right)$$

Add it as the initial condition

Poisson equation: $k\nabla^2 u + f = 0,$

where $\nabla = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}\right)$ and f = given function, k is a constant

Goal: f(x,y)-> u(x,y)

Applying finite difference method:

$$\frac{\partial^2 u}{\partial x^2} \cong \frac{\frac{u_{i+1,j} - u_{i,j}}{\Delta x} - \frac{u_{i,j} - u_{i-1,j}}{\Delta x}}{\Delta x} = \frac{u_{i+1,j} + u_{i-1,j} - 2u_{i,j}}{\Delta x^2}$$

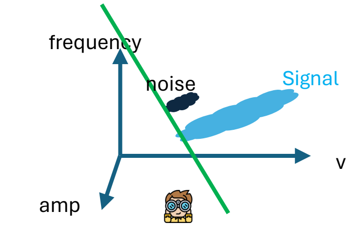
$$\frac{\partial^2 u}{\partial y^2} \cong \frac{\frac{u_{i,j+1} - u_{i,j}}{\Delta x} - \frac{u_{i,j} - u_{i,j-1}}{\Delta x}}{\Delta y} = \frac{u_{i,j+1} + u_{i,j-1} - 2u_{i,j}}{\Delta y^2}$$

Au = -f, where

A= derivative matrix, u=solution, -f = given function

Ref: https://github.com/gilbertfrancois/partial-differential-equations/blob/master/notebook/2D%20Poisson%20equation%2C%20finite%20difference.ipynb

What did I do?



- **Find cases**
  - High-frequency + Non-stationary dataset
- Prove it works
  - Applying FRFT to it

$$\frac{\partial u}{\partial t} + \alpha \frac{\partial u}{\partial x} = 0, \quad x \in [a, b], t > 0$$

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

$$u = u(x, t)$$

$$\mathbf{u}(s, \mathbf{0}) \propto \mathbf{exp}\left(-\frac{\mathbf{1}}{\mathbf{2}}\mathbf{s}^{\mathrm{T}}\mathbf{Q}\mathbf{s}\right)$$

Add it as the initial condition

Wave equation: $\frac{\partial}{\partial x}\left(\frac{\partial c^2 u}{\partial x}\right) + \frac{\partial}{\partial y}\left(\frac{\partial c^2 u}{\partial y}\right) = \frac{\partial^2 u}{\partial t^2}$,

where $c =$ wave velocity, set 1.

Goal: u(x,y,0)-> u(x,y,t)

Applying finite difference method to solve it

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

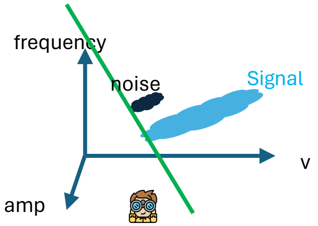
$$\Rightarrow \frac{u_{i,j}^{t+1} + u_{i,j}^{t-1} - 2u_{i,j}^{t}}{\Delta t^2} = \frac{u_{i+1,j}^{t} + u_{i-1,j}^{t} - 2u_{i,j}^{t}}{\Delta x^2} + \frac{u_{i,j+1}^{t} + u_{i,j-1}^{t} - 2u_{i,j}^{t}}{\Delta y^2}$$

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

Adding initial velocity(v) when t=1(some applications might need)

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

Ref: https://github.com/sachabinder/wave_equation_simulations/blob/main/2D_WAVE-EQ_variable-velocity.py




```
def wave_gen():

        #init cond - at t = 1
        #without boundary cond
        u_np1[1:N_x,1:N_y] = 2*u_n[1:N_x,1:N_y] - (u_n[1:N_x,1:N_y] - 2*dt*V_init[1:N_x,1:N_y])
        + Cx2*(  0.5*(q[1:N_x,1:N_y] + q[2:N_x+1,1:N_y ])*(u_n[2:N_x+1,1:N_y] - u_n[1:N_x,1:N_y])
                - 0.5*(q[0:N_x -1,1:N_y] + q[1:N_x,1:N_y ])*(u_n[1:N_x,1:N_y] - u_n[0:N_x -1,1:N_y]) )
        + Cy2*(  0.5*(q[1:N_x,1:N_y] + q[1:N_x ,2:N_y+1])*(u_n[1:N_x,2:N_y+1] - u_n[1:N_x,1:N_y])
                - 0.5*(q[1:N_x,0:N_y -1] + q[1:N_x ,1:N_y])*(u_n[1:N_x,1:N_y] - u_n[1:N_x,0:N_y -1]) )
```

$$\ldots\Big) + \frac{\partial}{\partial y}\left(\frac{\partial c^2 u}{\partial y}\right) = \frac{\partial^2 u}{\partial t^2},$$

et 1.

Goal: u(x,y,0)-> u(x,y,t)

- Prove it works
  - Applying FRFT to it

$$\frac{\partial u}{\partial t} + \alpha \frac{\partial u}{\partial x} = 0, \quad x \in [a,b], t > 0$$

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

$$u = u(x,t)$$

$$\mathbf{u}(s,\mathbf{0}) \propto \mathbf{exp}\left(-\frac{\mathbf{1}}{\mathbf{2}}\mathbf{s}^{\mathrm{T}}\mathbf{Q}\mathbf{s}\right)$$

Add it as the initial condition

Applying finite difference method to solve it

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

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

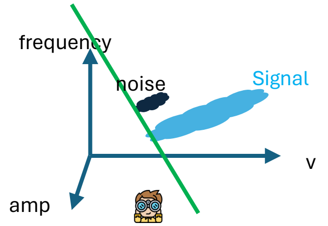
$$\Rightarrow u_{i,j}^{t+1} = 2u_{i,j}^{t} - u_{i,j}^{t-1} + \frac{\Delta t^2}{\Delta x^2}\left(u_{i+1,j}^{t} + u_{i-1,j}^{t} - 2u_{i,j}^{t}\right) + \frac{\Delta t^2}{\Delta y^2}\left(u_{i,j+1}^{t} + u_{i,j-1}^{t} - 2u_{i,j}^{t}\right)$$

Adding initial velocity(v) when t=1(some applications might need)

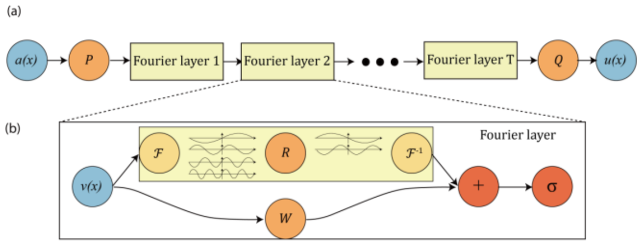
$$\Rightarrow u_{i,j}^{t+1} = 2u_{i,j}^{t} - u_{i,j}^{t-1} - 2\Delta t v_{i,j} + \frac{\Delta t^2}{\Delta x^2}\left(u_{i+1,j}^{t} + u_{i-1,j}^{t} - 2u_{i,j}^{t}\right) + \frac{\Delta t^2}{\Delta y^2}\left(u_{i,j+1}^{t} + u_{i,j-1}^{t} - 2u_{i,j}^{t}\right)$$

Ref: https://github.com/sachabinder/wave_equation_simulations/blob/main/2D_WAVE-EQ_variable-velocity.py
https://hplgit.github.io/num-methods-for-PDEs/doc/pub/wave/pdf/wave-4print-A4-2up.pdf

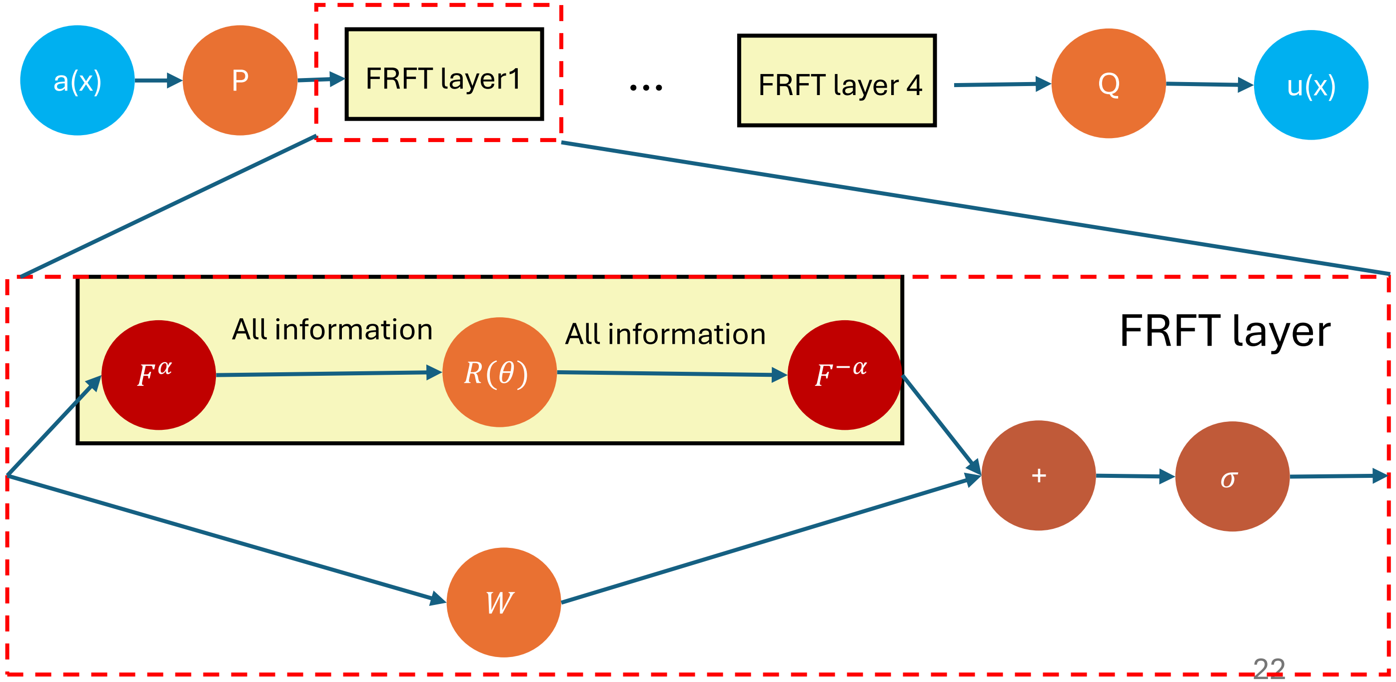
What did I do?



- Find cases
  - High-frequency + Non-stationary dataset
- **Prove it works**
  - Applying FRFT to it



Based on
FNO architecture



Ref: Fourier Neural Operator for Parametric Partial Differential Equations

Outline

- Problem definition
- Motivation
- What did I do?
- **Result**
- Conclusion

Result - burger's equation

- A: Fourier layer, B: FRFT layer
- Architecture: AABB(v=0), ABBB(v=0.001)
- Dataset size 200
  - u(x,0)=1x1024, u(x,t)=1x1024
- Test sequence:
  - Train1:dataset[:30], Test1: dataset[-100:]
  - Train2 dataset[30:60], Test2 dataset[-100:]
  - Train3 dataset[60:90], Test3 dataset[-100:]
  - Train4 dataset[-30:], Test 4 dataset[:100]

Burger Test l2 error	**V=0**	**V=0.001**
Train 1& Test 1	0.335549	0.371378
Train 2 & Test 2	0.335588	0.370999
Train 3 & Test 3	0.335466	0.371433
Train 4 & Test 4	0.332224	0.36674
FNO vanilla avg.	**0.334707**	**0.370138**

Test l2 error	**V=0**	**V=0.001**
Train 1& Test 1	0.083705	0.045617
Train 2 & Test 2	0.089063	0.045515
Train 3 & Test 3	0.082765	0.048091
Train 4 & Test 4	0.09321	0.04524
FNO all modes avg.	**0.087186**	**0.046116**

Test l2 error	**V=0**	**V=0.001**
Train 1& Test 1	0.069647	0.032559
Train 2 & Test 2	0.064908	0.031946
Train 3 & Test 3	0.067103	0.0264
Train 4 & Test 4	0.055544	0.026487
FRFTNO avg.	**0.064300**	**0.029348**

Result - poisson's equation

- A: Fourier layer, B: FRFT layer
- Architecture: BAAA
- Dataset size 200
  - f(x,y)=1x64x64, u(x,y)=1x64x64
- Test sequence:
  - Train1:dataset[:30], Test1: dataset[-100:]
  - Train2 dataset[30:60], Test2 dataset[-100:]
  - Train3 dataset[60:90], Test3 dataset[-100:]
  - Train4 dataset[-30:], Test 4 dataset[:100]

Poisson Test l2 error	
Train 1& Test 1	0.771226
Train 2 & Test 2	0.792926
Train 3 & Test 3	0.750082
Train 4 & Test 4	0.795369
FNO vanilla avg.	**0.777400**

Test l2 error	
Train 1& Test 1	0.448968
Train 2 & Test 2	0.449978
Train 3 & Test 3	0.444467
Train 4 & Test 4	0.445845
FNO all modes avg.	**0.447315**

Test l2 error	
Train 1& Test 1	0.371117
Train 2 & Test 2	0.354186
Train 3 & Test 3	0.334832
Train 4 & Test 4	0.355024
FRFTNO avg.	**0.353790**

Result - wave's equation

- A: Fourier layer, B: FRFT layer
- Architecture: BAAB(V=0), BBAB(V=0.001)
- Dataset size 200
  - u(x,y,t1)=1x64x64x1, u(x,y,t2)=1x64x64x1
- Test sequence:
  - Train1:dataset[:30], Test1: dataset[-100:]
  - Train2 dataset[30:60], Test2 dataset[-100:]
  - Train3 dataset[60:90], Test3 dataset[-100:]
  - Train4 dataset[-30:], Test 4 dataset[:100]

Wave Test l2 error	**V=0**	**V=0.001**
Train 1& Test 1	0.083385	0.084923
Train 2 & Test 2	0.08481	0.084671
Train 3 & Test 3	0.083464	0.084933
Train 4 & Test 4	0.082462	0.082445
FNO vanilla avg	**0.083530**	**0.084243**

Test l2 error	**V=0**	**V=0.001**
Train 1& Test 1	0.07214	0.067753
Train 2 & Test 2	0.069251	0.07335
Train 3 & Test 3	0.073995	0.072818
Train 4 & Test 4	0.072781	0.070534
FNO all modes avg.	**0.072042**	**0.071114**

Test l2 error	**V=0**	**V=0.001**
Train 1& Test 1	0.06064	0.062344
Train 2 & Test 2	0.060401	0.060734
Train 3 & Test 3	0.061143	0.060232
Train 4 & Test 4	0.059051	0.063212
FRFTNO avg	**0.06031**	**0.061631**

Outline

- Problem definition
- Motivation
- What did I do?
- Result
- **Conclusion**

Conclusion

- High-frequency dataset
- Non-stationary dataset
- Fractional Fourier transform
- Future work:
  - A general method for architecture
  - Weights decomposition
  - Linear canonical transform