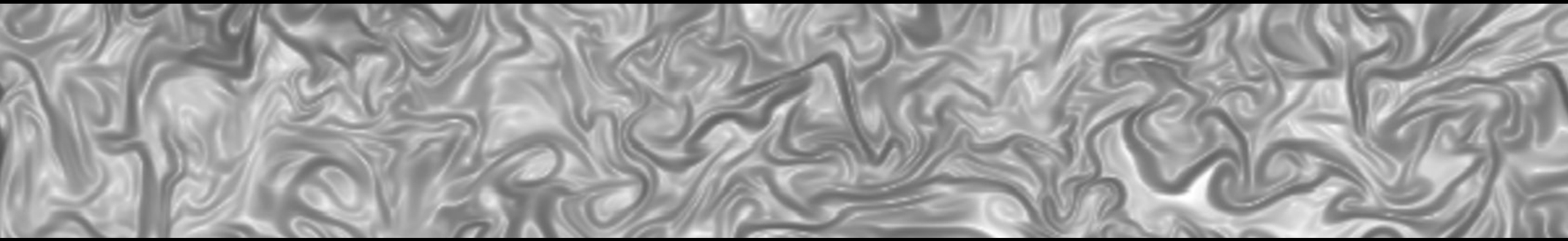
# 2D Wind Farm

## Project Documentation

Update: 30th May 24’



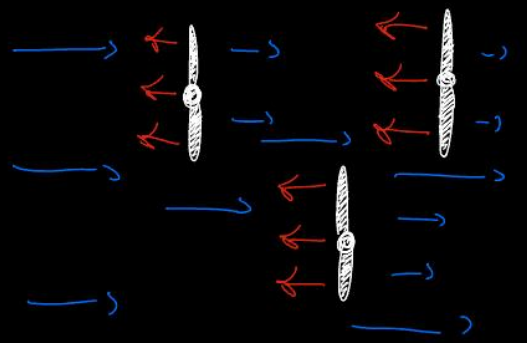
# Axial Induction - Theoretical concept

Source : “Optimization wind plant layouts using adjoint approach” R.King et al.

## Axial induction implementation

The effect of the presence of the turbines on the flow is represented as an external body force. ($f_{AD}$)

$$f_{AD} = \frac{1}{2} \cdot \rho \cdot A_m \cdot C_t' \cdot \beta^{-1} \cdot \varphi \, \| \bar{u}(m) \cdot \hat{n} \|$$



- $\rho$ : air density $\rho = 1$
- $A_m$ (turbine rotor area) $A_m = \pi \cdot R^2$
- $C_t'$ (modified thrust coefficient) $C_t' = \frac{C_t}{(1-a)^2} \underset{(C_t = 4a(1-a))}{=} \frac{4a(1-a)}{(1-a)^2} \underset{(a=1/4)}{=} 4/3$
- $\beta'$ (normalizing constant) used to normalize the values of the smoothed loading. $\beta = R\,(\#cells)$ The force is divided among the number of cells the rotor ext
- $\varphi$ : (geometric smoothing kernel) Uniform , Gaussian . . .
- $\| \bar{u} \cdot \hat{n} \|$ : absolute value of the mean velocity

Currently, the force is multiplied by a constant 10, in order to make the axial induction noticeable. Review this in the future.

# Axial Induction - How to apply the force

From Newton's Second Law ...

$$F = m \cdot \frac{dv}{dt}$$

$$\frac{dv}{dt} = \frac{F}{m}$$

$$\int dv = \int \frac{F}{m} \cdot dt = \frac{F}{m} \int dt$$

$$v_{new} - v_{old} = \frac{F}{m}(t_{new} - t_{old}) = \frac{F}{m} \cdot \Delta t$$

($\Delta t$: timestep)

$$\underline{u}_{new} = \underline{u} - \frac{F}{\rho} \cdot dt$$

- $\underline{u}_{new}$: updated velocities
- $\underline{u}$: original velocities
- $F$: the direction of the force is the same as the direction of the velocity
- $\rho$: density
- $dt$: timestep

in my case, this is simply $\rho$, or?

(it wouldn't make sense to take the mass from a 2D flow!)

# Axial Induction Implement - Approach A

This approach only considers the u component of velocity to calculate the axial induction force and apply the force.

1) Compute axial induction force from the u component of velocity

$$F_{TOTAL} = \frac{1}{2} \cdot \rho \cdot A_N \cdot C_t' \cdot \bar{u}$$

where $A_N = \pi \cdot (R \cdot \cos(yaw))^2$ (Normal area)

$C_t' = 4/3$ (from King's paper)

2) Apply force to u component of velocity

$$u_{new} = u - \frac{F}{\rho} \cdot dt$$

# Axial Induction Implementation - Approach B (current)

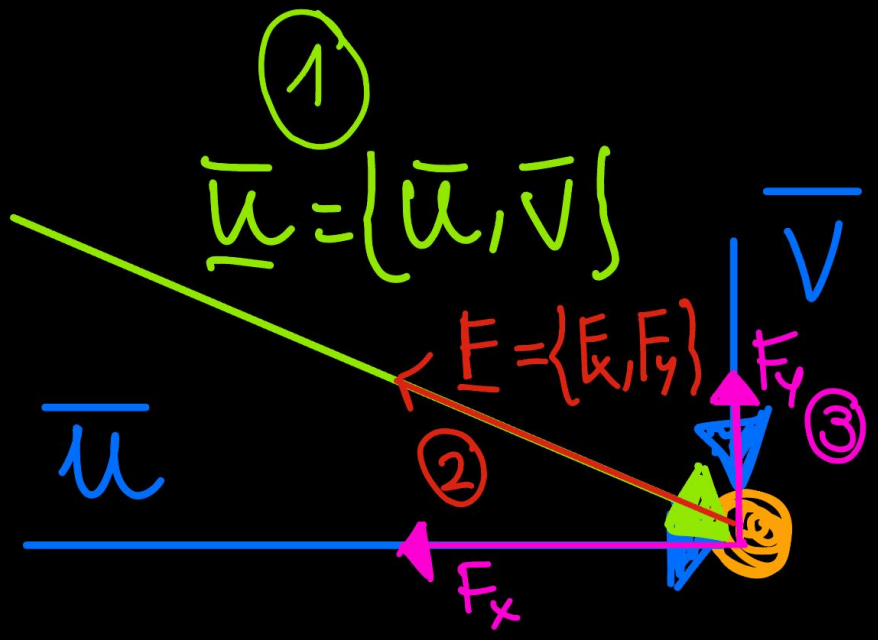
This approach only considers both vertical (v) and horizontal (u) components of velocity.

1 – Compose vector $\underline{u} = \{u, v\}$

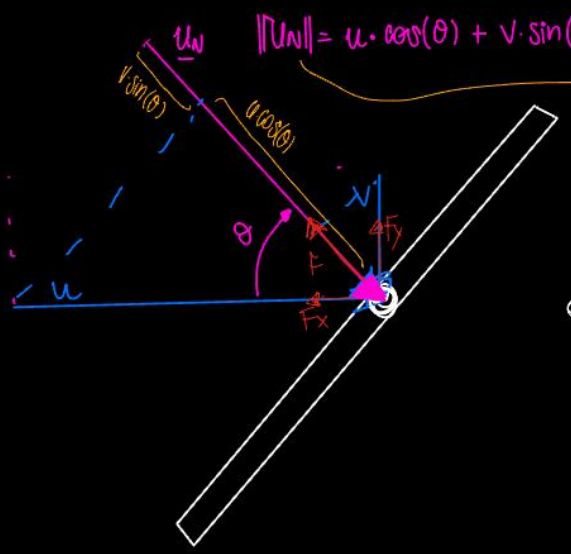
2 – Calculate force $\underline{F}(\underline{u})$

3 – Decompose force $\underline{F} \rightarrow F_x, F_y$

4 – Apply $F_x$ to $u$
$F_y$ to $v$



# Axial Induction Implementation - Approach B




$\|\underline{u}_N\| = u \cdot \cos(\theta) + v \cdot \sin(\theta)$ Does this make sense?

→ this is all we need to calculate the force.

## So, for implementation...

- Calculate force

$$F_{TOTAL} = \frac{1}{2} \cdot \rho \cdot A_n \cdot C_t \cdot \|\bar{u}_N\|$$

(normal velocity)

verify this

where $\|\bar{u}_N\| = \bar{u} \cdot \cos(\theta) + \bar{v} \cdot \sin(\theta)$

- Decompose

$$F_x = F \cdot \cos(\theta)$$
$$f_y = F \cdot \sin(\theta)$$

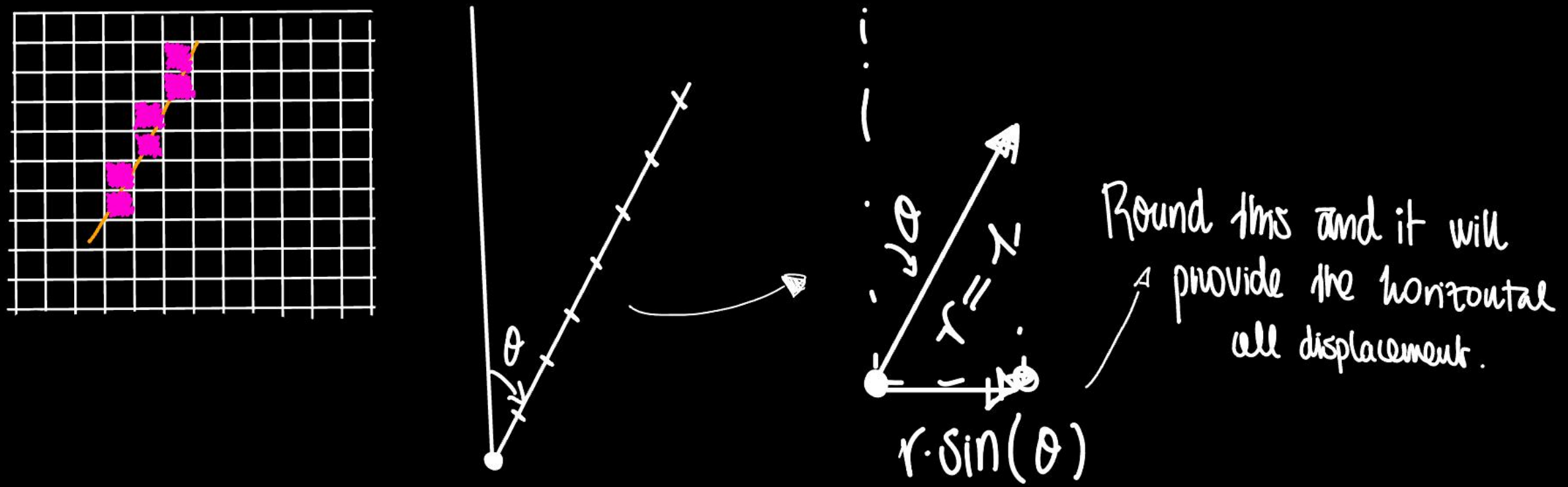
- Normalize force among cells where it is going to be applied

$F = F_{TOTAL} / (2 \cdot R)$ → Number of cells where force will be applied

- Apply force

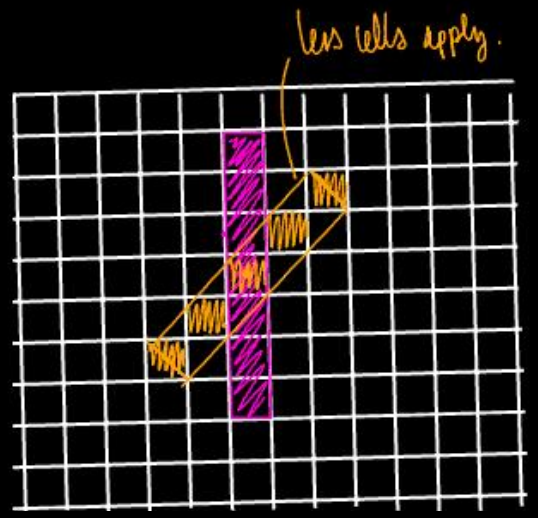
$$\begin{cases} u_{new} = u - \frac{F_x}{\rho} \cdot dt \\ v_{new} = v - \frac{F_y}{\rho} \cdot dt \end{cases}$$

# Implement forces into the corresponding cells of a yawed turbine

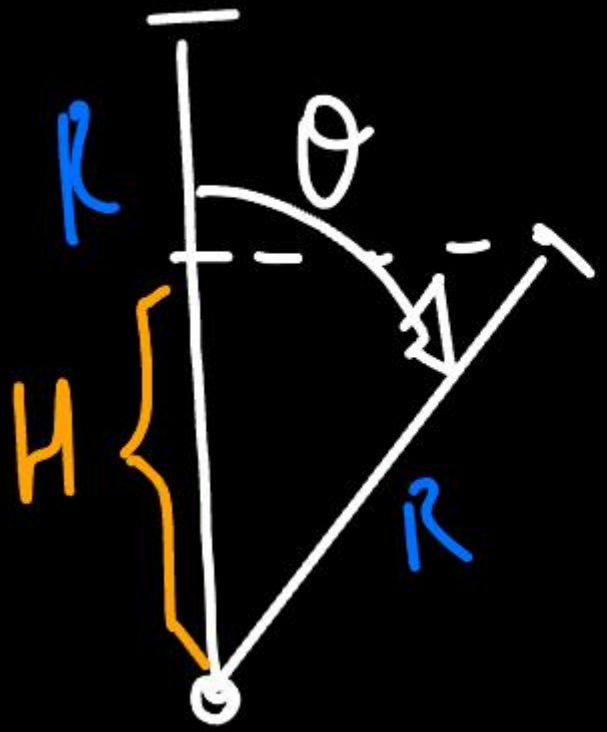



The cell displacement is computed using trigonometry laws. Using the known yaw angle, the solver iterates through all the cells calculating the horizontal displacement and applies the AI force to the corresponding cells. Rounding the horizontal displacement is key

# Crop the number of vertical cells according to trigonometry rules



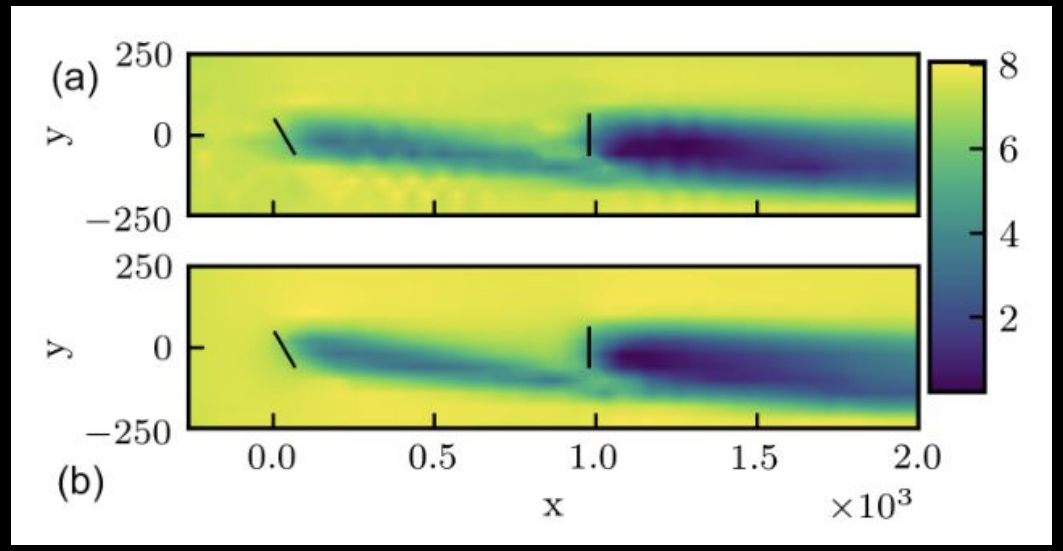

The range of j (number of rows that correspond to a wind turbine gets reduced when the yaw angle increases.

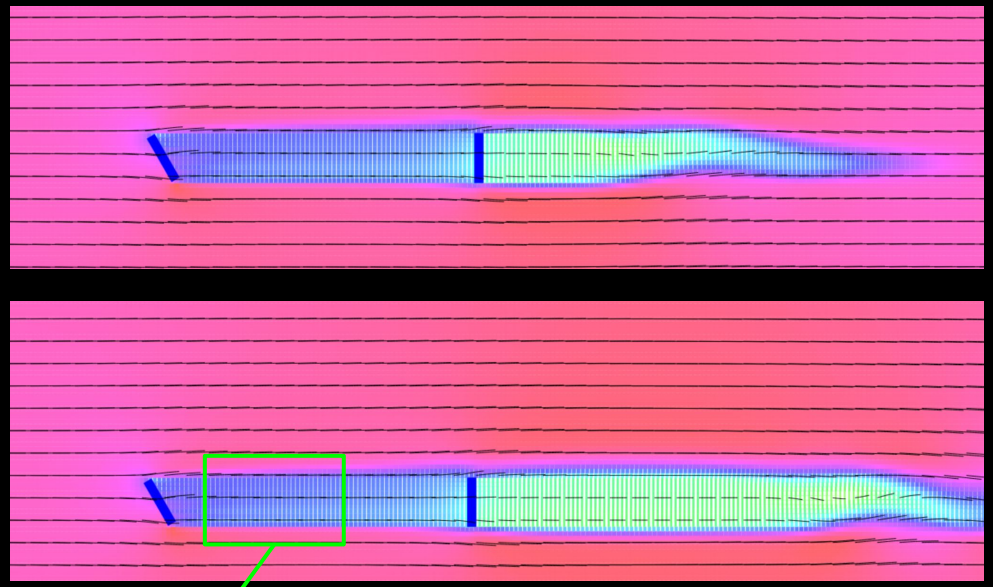



$$H = R \cdot \cos(\theta)$$

**Prior extra step:**
Projects the radius into the turbine axis. This projection (H) is used to iterate through the turbine cells when applying the AI force. It is also used to normalize the force.

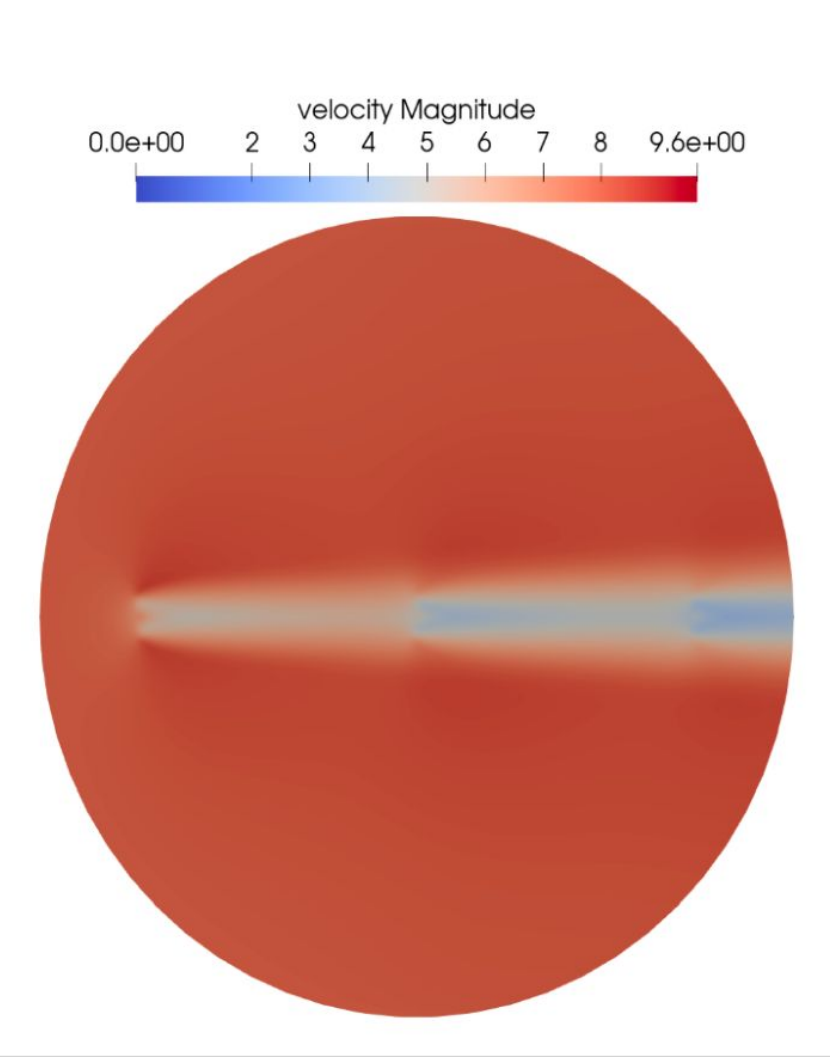
# Comparison with J.Quick wake model

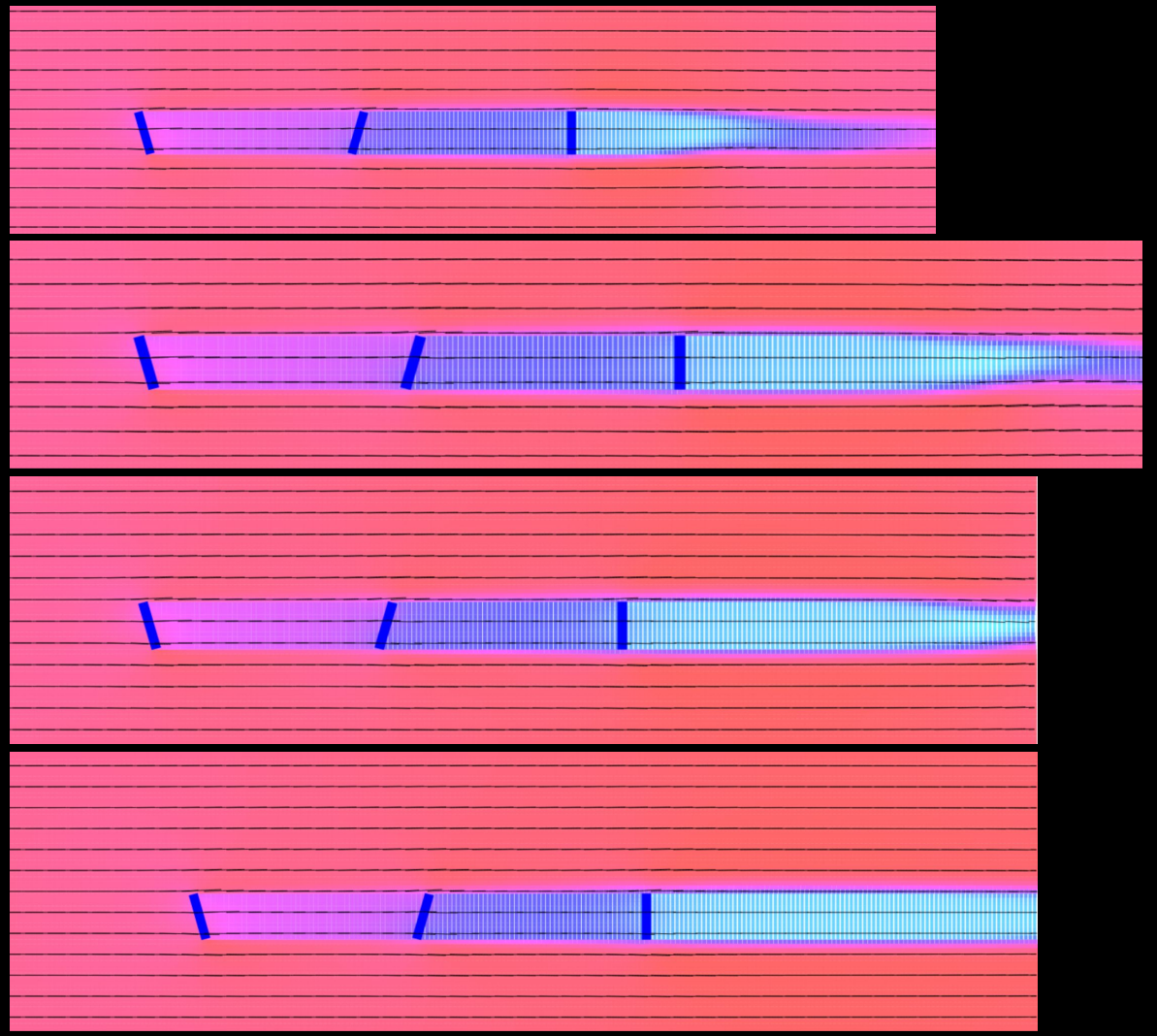


- The velocity is larger at the top than at the bottom (agreement with J.Quick results).
- Wakes do not displace (Disagreement with J.Quick results).

# More on the yaw effect: WindSE



Steady state is eventually reached: wakes are not realistic because they do not diffuse.

# Generation Power

$$J = \frac{1}{2} \cdot \rho \cdot A \cdot C_p' \cdot \|\bar{u} \cdot \hat{n}\|^3 \qquad \text{where } C_p' = \frac{C_p}{(1-a)^3} = \frac{4a(1-a)^2}{(1-a)^3} = \frac{4a}{1-a}$$

In my case, this is simply the horizontal component of velocity ($u$).

Calculated from the "R" input. [$A = \pi \cdot R^2$]

So, in practice...

$$J = \frac{1}{2} \cdot \rho \cdot \pi \cdot R^2 \cdot \frac{4a}{1-a} \cdot u^3$$

let's code it...

Now, I just need to calculate the power for each of the turbines and add counters.
Finally, do the total power summation and add a total counter.

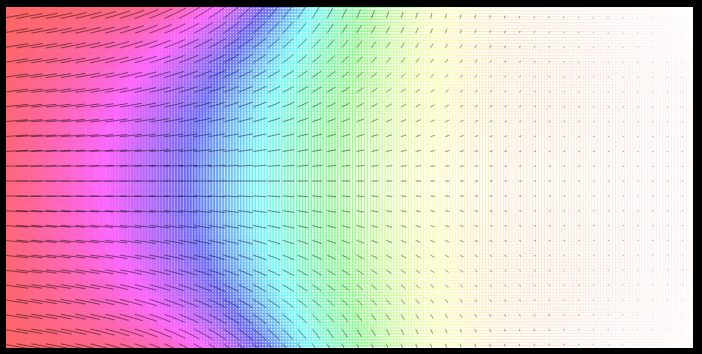
```
Power(dt, x, y, R, yaw) {
    // Calculate the generation power of a certain turbine, based on theory from R.Kir
    var n = this.numY;
    var startJ = Math.max(y - R, 0);
    var endJ = Math.min(y + R, n); // I modified this command line.
    // Compute the mean velocity.
    var u_sum = 0; // Initialize
    for (var j = startJ; j <= endJ; j++) {
        u_sum += this.u[x * n + j]; // Accumulate the sum of u elements
    } // Iterate over all elements.
    var u_mean = u_sum/2*R; // normalize

    var a = 1/4; // Axial induction coeff. Check this value.
    var Cp = 4*a/(1-a); // Modified power Coefficient. King paper.
    var r = R * this.cos(yaw); // r: effective / R: actual radio
    var A = Math.PI * r * r; // effective area
    var J = 0.5*this.density * A * Cp * u_mean ** 3; // generation power.

    return J;
}
```

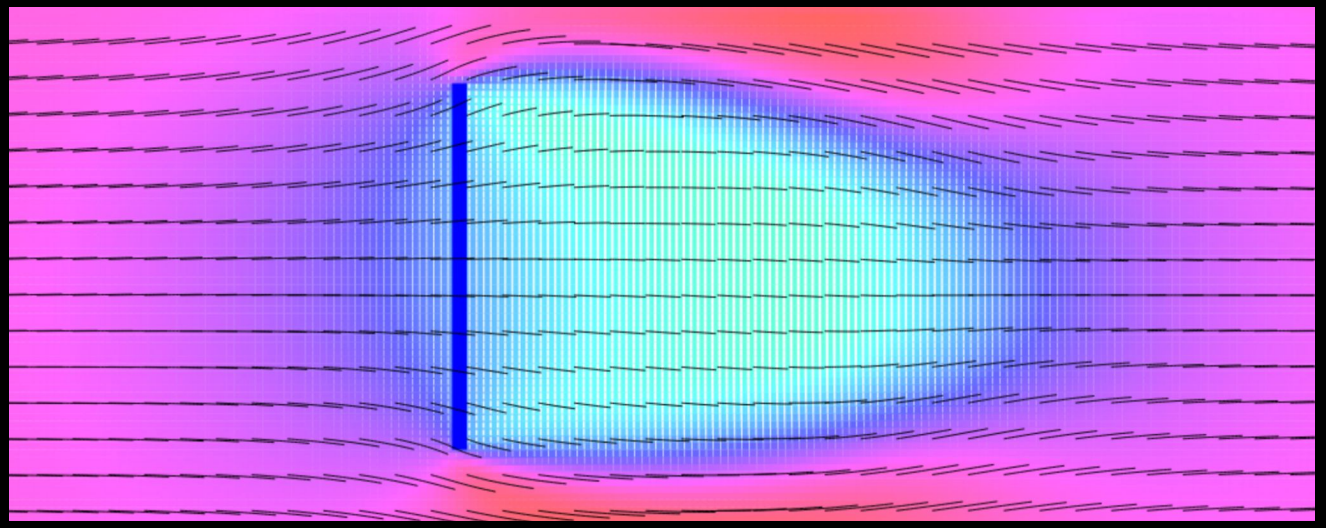
# What is Numerical Viscosity?

- Due to discretization. Interpolations can smooth out the motions of the flow, producing effects that are similar to what viscosity does in a real fluid.
- It helps replicate the shedding mechanism that occurs in real physics due to actual viscosity.
- How can I increase it further?
  - Coarse Grid. I don't want to sacrifice resolution.
  - Change advection scheme. Too complex
  - Larger timestep. Easy implementation. Does not work.
  - Additional term. Interesting idea.

# Remove Top and Bottom walls

- Removing the top and bottom walls could help create the deflection effect on the wake that we expect from a yawed turbine.
- I removed the walls.
- A “zero gradient (du/dy=o & dv/dy=0)” BC was not necessary to apply (anything changed).
- The issue is that the field is initialized with zero velocity and the flow diverges and takes a while to get the inlet velocity advected through the domain:
- Solved this by initializing u=invel and v=0 thru all domain.
- Example of how the flow can exit the domain:





# Add diffusivity: Viscous term

The problem of the current model is that the velocities don't diffuse so the wakes never dissipate.
We want to add an extra step that diffuses them, so I will use the viscous term of the momentum eq of the NS.

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\frac{1}{\rho}\nabla p + \nu\nabla^2\mathbf{u} + \mathbf{f}$$

v: kinematic viscosity/diffusion coef

The viscous term accounts for the internal friction within the fluid due to its viscosity, which tends to dampen the fluid motion and diffuses the momentum.

$$\text{Viscous term} = \nu\left(\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right)\hat{\mathbf{i}} + \left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right)\hat{\mathbf{j}}\right)$$

In order to implement this as an extra step, we need to linearize and discretize it.
As proposed by Peter, I'll use a finite difference discretization scheme:

U component

$$(\nabla^2 u)_{i,j} \approx \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h^2}$$

V component

$$(\nabla^2 v)_{i,j} \approx \frac{v_{i+1,j} - 2v_{i,j} + v_{i-1,j}}{h^2} + \frac{v_{i,j+1} - 2v_{i,j} + v_{i,j-1}}{h^2}$$

*Where h is the length of the cell facet (assuming squared cells).

Finally, I included an extra step in the simulation, which updates the velocity field applying diffusion.

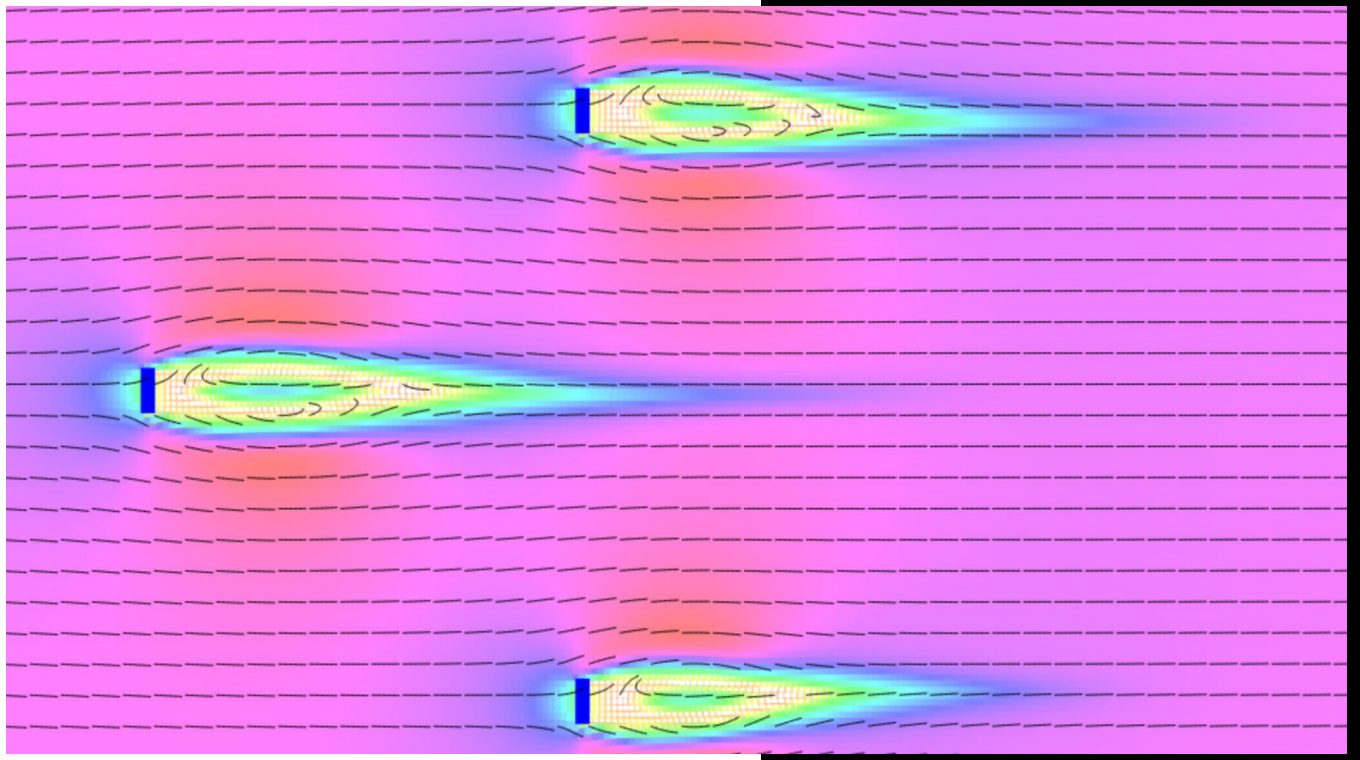
$$u_{i,j}^{n+1} = u_{i,j}^{n} + \Delta t\left[\nu\left(\nabla^2 u\right)_{i,j}\right]$$

$$v_{i,j}^{n+1} = v_{i,j}^{n} + \Delta t\left[\nu\left(\nabla^2 v\right)_{i,j}\right]$$

# Kinematic viscosity

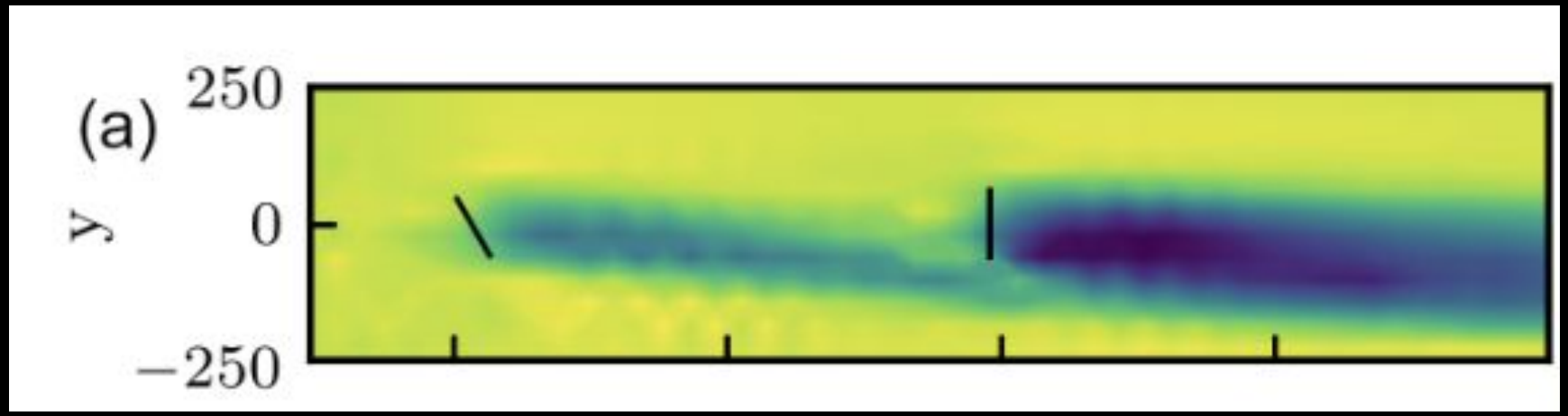
## Calculation Steps

1. **Characteristic Length and Velocity:**
   - $L_{\text{ref}} = 3000 \text{ m}$
   - $U_{\text{ref}} = 8 \text{ m/s}$
2. **Kinematic Viscosity of Air:**
   - $\nu_{\text{real}} = 1.5 \times 10^{-5} \text{ m}^2/\text{s}$
3. **Non-Dimensional Viscosity:**

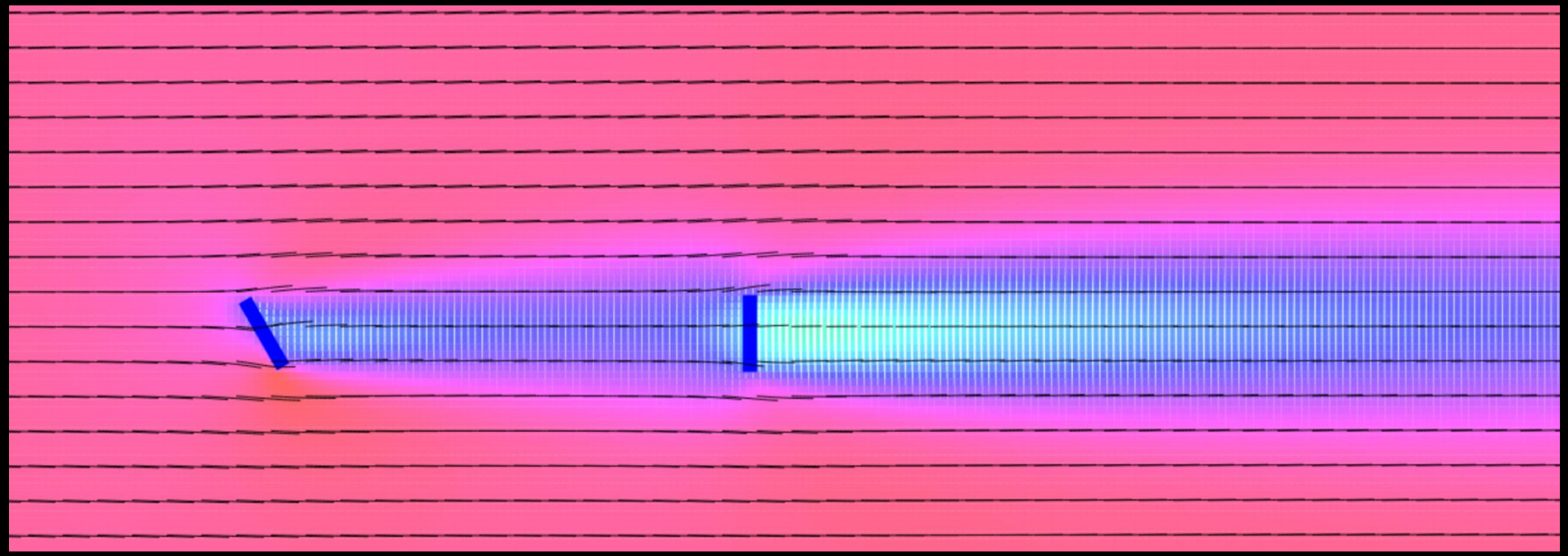


$$\nu_{\text{non-dim}} = \frac{1.5 \times 10^{-5} \text{ m}^2/\text{s}}{3000 \text{ m} \times 8 \text{ m/s}} = \frac{1.5 \times 10^{-5}}{24000} = \boxed{6.25 \times 10^{-10}}$$

# Comparison with J.Quick wake model (with viscosity)

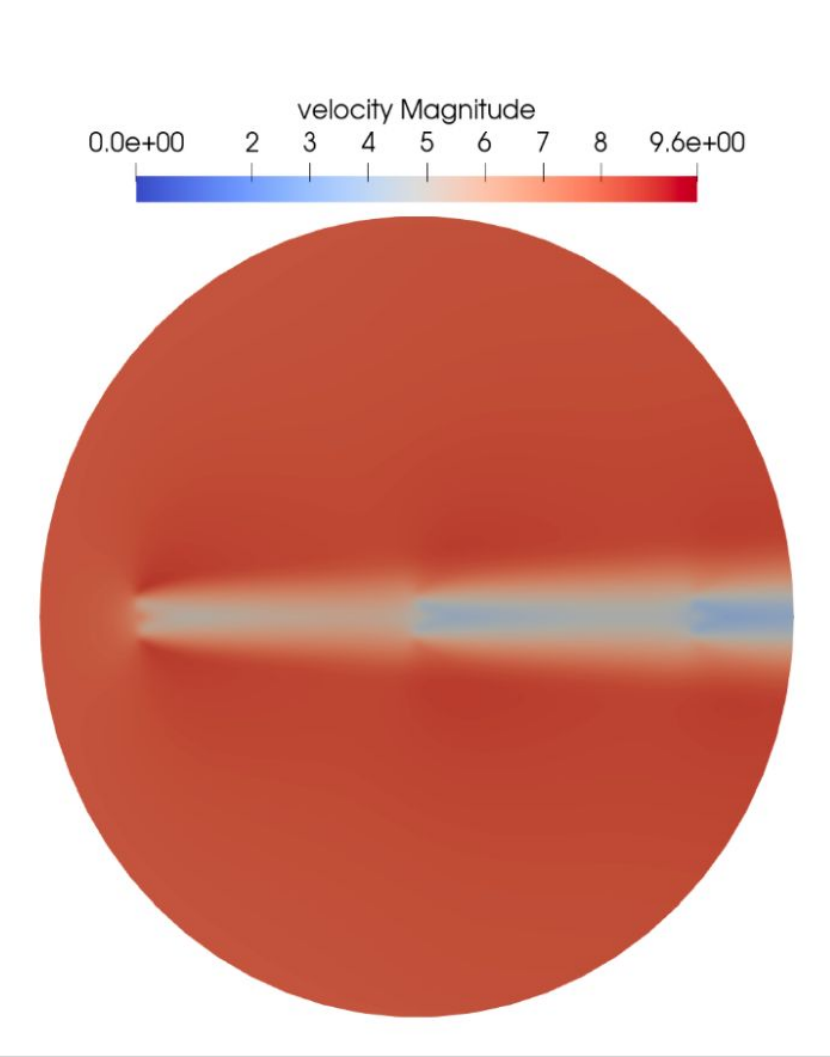



My wake does not deflect like Julian's...

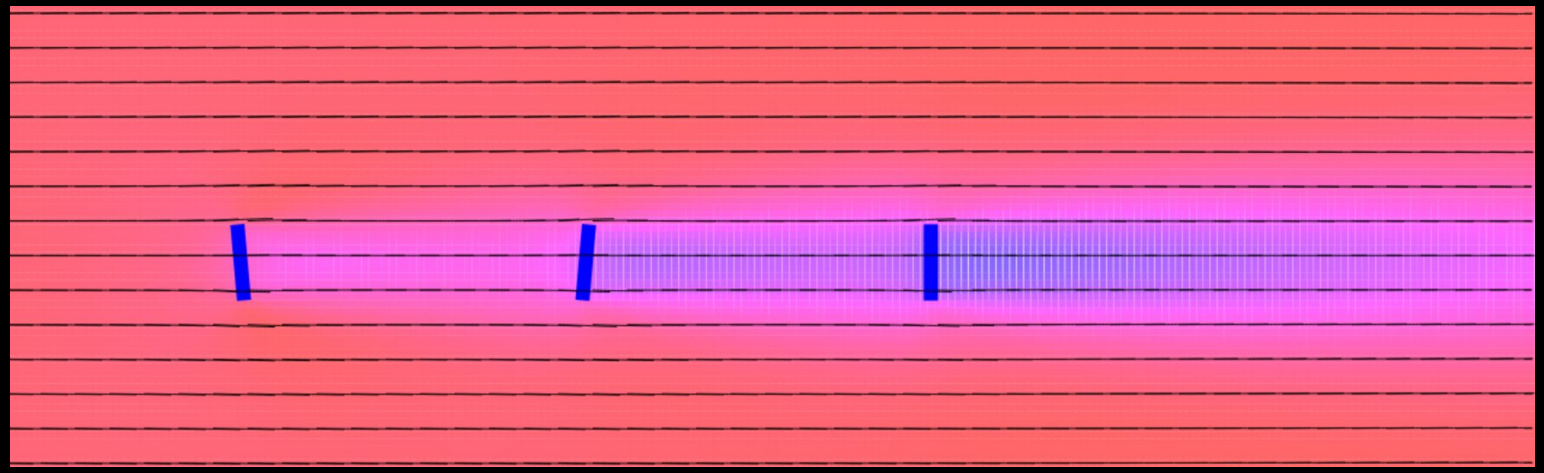


*3x theoretical force. I exagerated it in order to

# More on the yaw effect: WindSE (with viscosity)




* Unmodified theoretical force



This looks pretty much realistic.
But I need to validate magnitudes make sense with generation calculations.
Viscosity value needs to be discussed…
Another difference with WindSE is that a sinusoidal force (zero at center and edges) is applied in his while a uniform force is applied in mine.

Steady state is eventually reached: wakes are not realistic because they do not diffuse.

# Dimensional analysis

Axial Induction

$$F = \frac{1}{2} \cdot \rho \cdot A \cdot C_t \cdot \bar{u}^2$$

$$\left[\frac{kg \cdot m}{s^2}\right] = \left[\frac{kg}{m^3}\right] \cdot \left[m^2\right] \cdot [\,] \cdot \left[\frac{m}{s}\right]^2 \checkmark$$

Which are the length/velocity units of my code??

There are several different concepts.

- canvas.width/height. This is given the canvas size, which depends on the browser's window size. It does not affect the simulation.
- simHeight/Width. These define the aspect ratio of the simulation domain, which depends on the canvas shape.
- domainHeight/Width. This is the physical height in the simulation. These are the only quantities of our interest! Height is defined and width is defined in terms of the height and the aspect ratio. This is a normalized quantity. The height is actually defined as 1, so we need to denormalize in order to get the actual physical quantities in order to calculate the actual force. But then.. Do we need to normalize the force back? Or can we just calculate the force with the normalized quantities?

Used values:

- inVel= 0.5
- domainHeight = 1

Check SetUpScene() with Peter !!!

I believe that I don't necessarily need to denormalize the quantities before calculating the force. I can perform the calculations directly using the normalized values. This approach maintains the consistency of dimensional analysis and is typically used in fluid dynamics simulations to keep the equations dimensionless and comparable. Do I need to adjust your force equation accordingly to reflect these normalizations?

# Dimensional Analysis

| Dimension | Characteristic |
|---|---|
| Length (m) | 3000 |
| Velocity (m/s) | 8 |
| Time (s) | 375 |

| Resolution | 150 |
|---|---|
| CFL | |
| 0.01 | |

(Number of domain rows)

| | Real (m) (m/s) (s) | Simulation |
|---|---|---|
| Domain Height | 3000 | 1 |
| Blade radius (R) | 60 | 3 |
| Inlet Velocity | 8 | 1 |
| Time Step | 1 | 0.00267 |
| One Cell | 20 | 1 |

Issues/Comments:

- Orange brackets are the editable parameters.
- The real timestep, computed from the CFL condition does not match with the one computed from the simulation time step using the characteristic time (table). Have I wrongly computed the characteristic time? Is is better to calculate CFL from real or simulation parameters?
- The simulation timestep is the adimensional time step. Does this correspond to the real time that a timestep takes to compute? How can I know that one? Is it modifiable?
- It is better to maintain the simulation values and simply modify the characteristic ones.