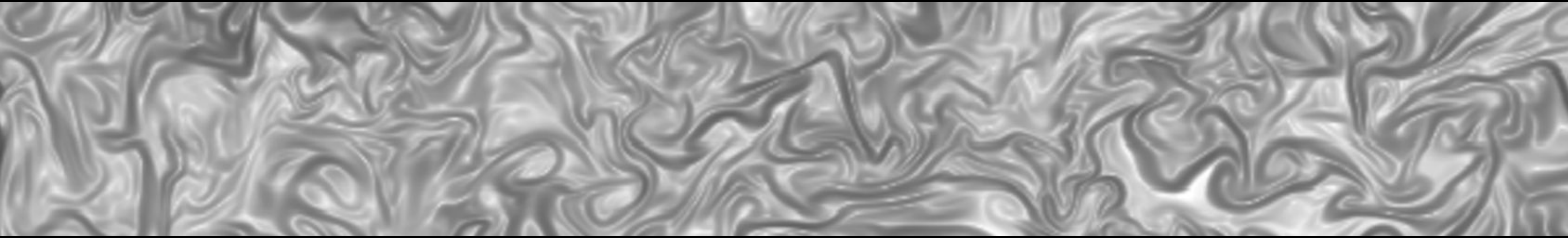
# 2D Wind Farm

## Project Documentation

### Update: 15th May 24’



Add link to github preview here

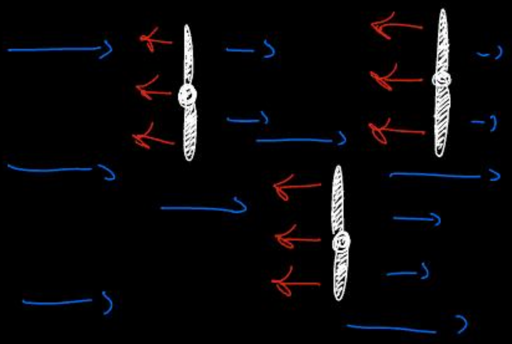
# Axial Induction - Theoretical concept

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 an ~~ān~~ external body force. ($f_{AD}$)

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



- $\rho$ : air density $\rho = 1$
- $A_n$ (turbine rotor area) $A_n = \pi \cdot R^2$
- $C_t'$ (modified thrust coefficient) $C_t' = \frac{C_t}{(1-a)^2} \underset{(C_t = 4a(1-a))}{=} \frac{4a(1-a)}{(1-a)^2} \underset{(a=1/4)}{=} \boxed{4/3}$
- $\beta$ (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...

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

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

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

timestep

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

in my case, this is simply $\rho$, ok?
(it wouldn't make sense to take the mass from a 2D flow!)

updated velocity

original velocity

the direction the force is the same as the direction of the velocity

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

density

timestep

# 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

$$F_{TOTAL} = \frac{1}{2} \cdot \rho \cdot A_N \cdot C_t' \cdot \bar{u} \quad \text{where} \quad A_N = \pi \cdot (R \cdot \cos(yaw))^2 \text{ (Normal area)}$$

$$C_t' = 4/3 \text{ (from King's paper)}$$

2) Apply force to u component of velocity

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

## Axial Induction Implementation - Approach B (current)

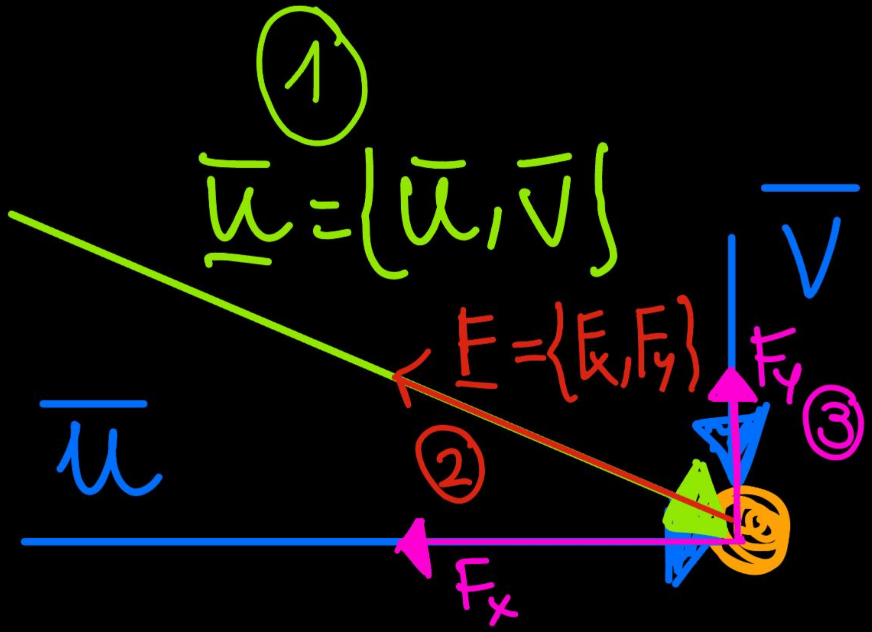
This approach only considers both vertical (v) and horizontal (u) components of velocity.

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

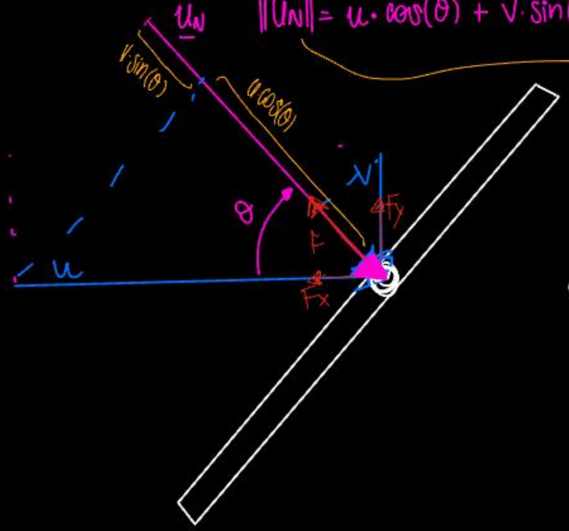
2 – Calculate force $\underline{F}(\underline{u})$

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

4 – Apply $\left| \begin{array}{l} F_x \text{ to } u \\ F_y \text{ to } v \end{array} \right.$


1
$\underline{\bar{u}} = \{\bar{u}, \bar{v}\}$
$\bar{v}$
$\underline{F} = \{F_x, F_y\}$
$F_y$
3
$\bar{u}$
2
$F_x$

# Axial Induction Implementation - Approach B



$\|U_N\| = u \cdot \cos(\theta) + v \cdot \sin(\theta)$ Does this make sense?

→ this is all we need to calculate the force.

## So, for implementation...

- Calculate force

$$F_{TOTAL} = \frac{1}{2} \cdot \rho \cdot A_N \cdot C_t' \cdot \|\underbrace{\bar{u}_N}_{\text{mean velocity}}\|$$

verify this

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

- Decompose

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

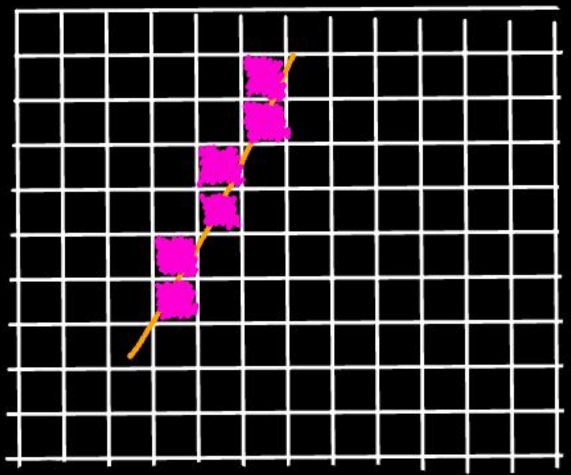
- Normalize force among cells where it is going to be applied

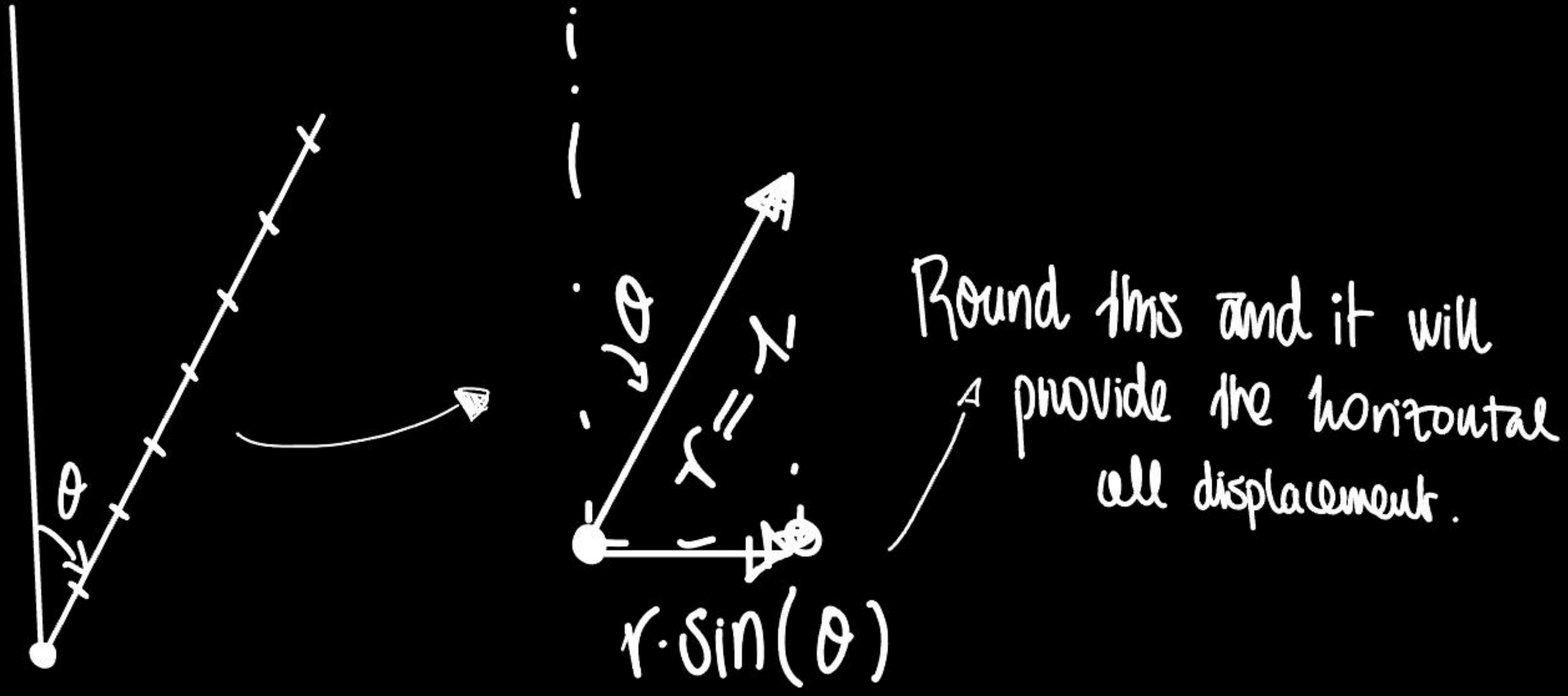
$F = F_{TOTAL} / (2 \cdot R)$ → Number of cells where force will be applied

- Apply force

$$U_{new} = u - \frac{F_x}{\rho} \cdot dt$$
$$V_{new} = v - \frac{F_y}{\rho} \cdot dt$$

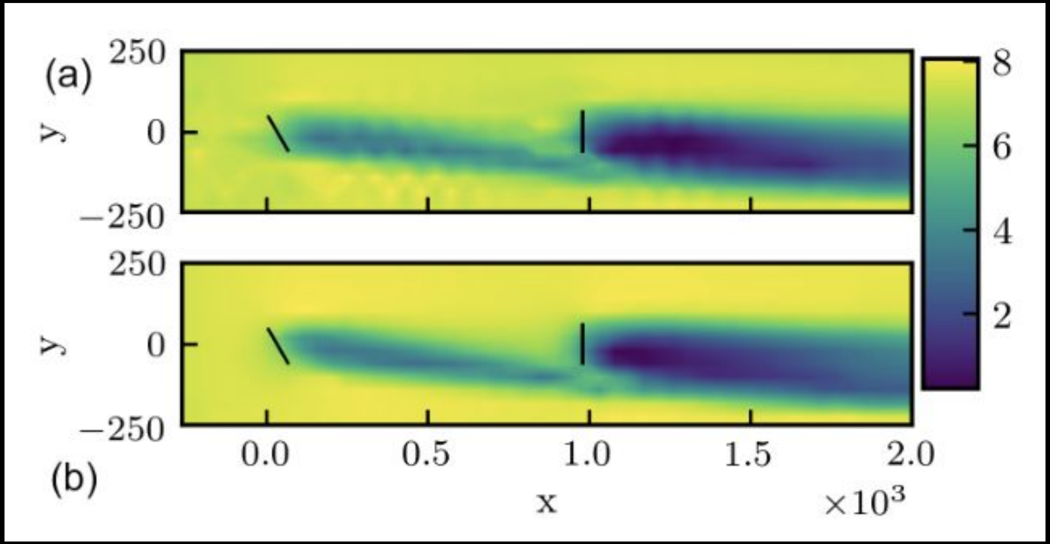
## Implement forces into the corresponding cells of a yawed turbine

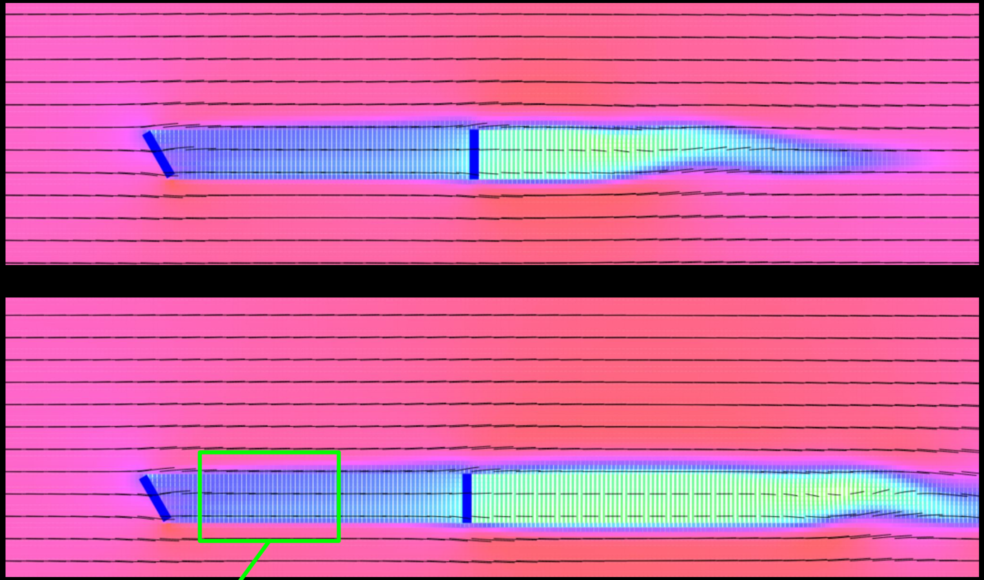



θ
θ
r = λ
r·sin(θ)
Round this and it will provide the horizontal cell displacement.

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

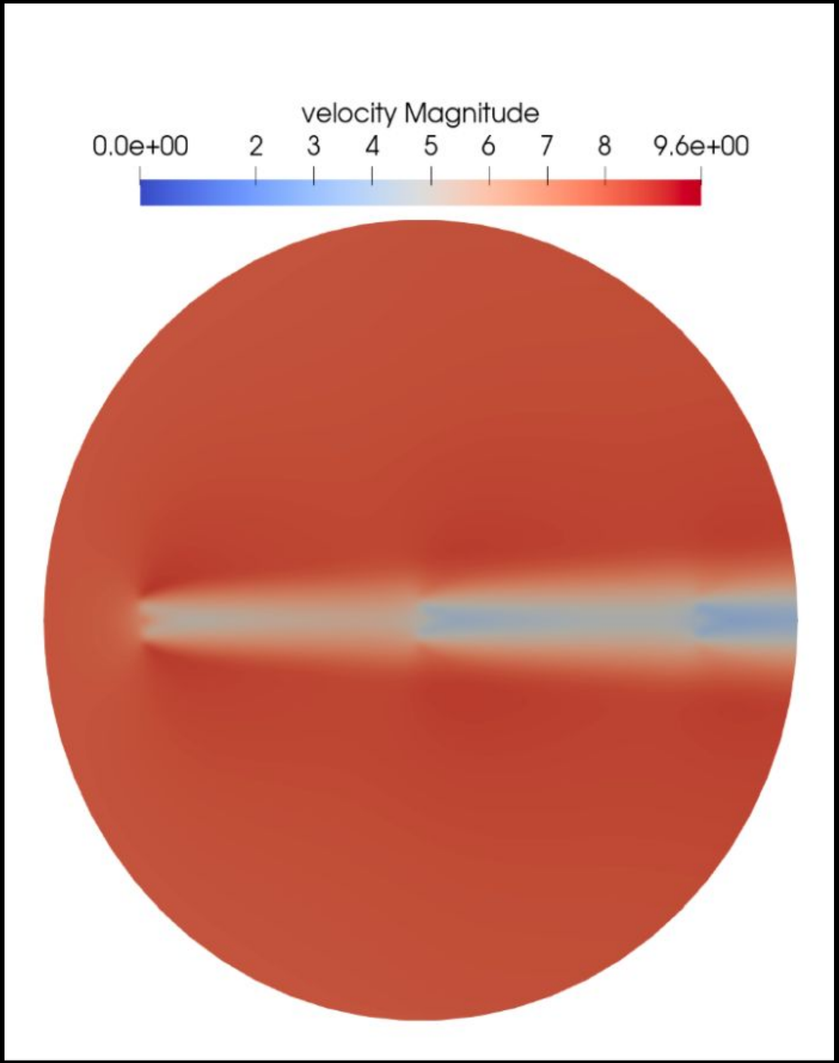
# Comparison with J.Quick wake model

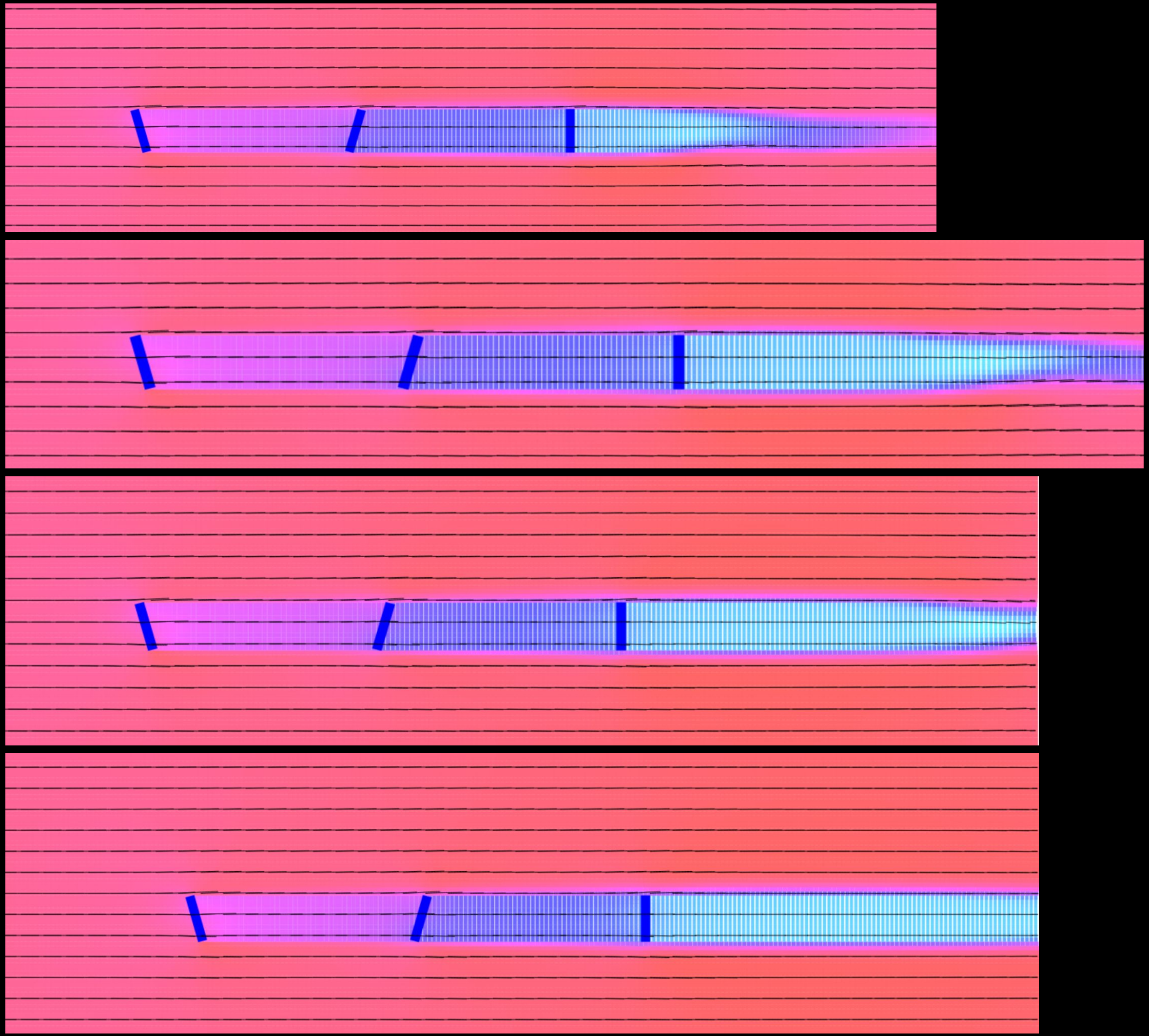




- The velocity is larger at the top than at the bottom (agreement with J.Quick results).
- Wakes do not displace (Disagreement with J.Quick results).

# More on the yaw effect: WindSE





Steady state is eventually reached but the 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 ($u$).

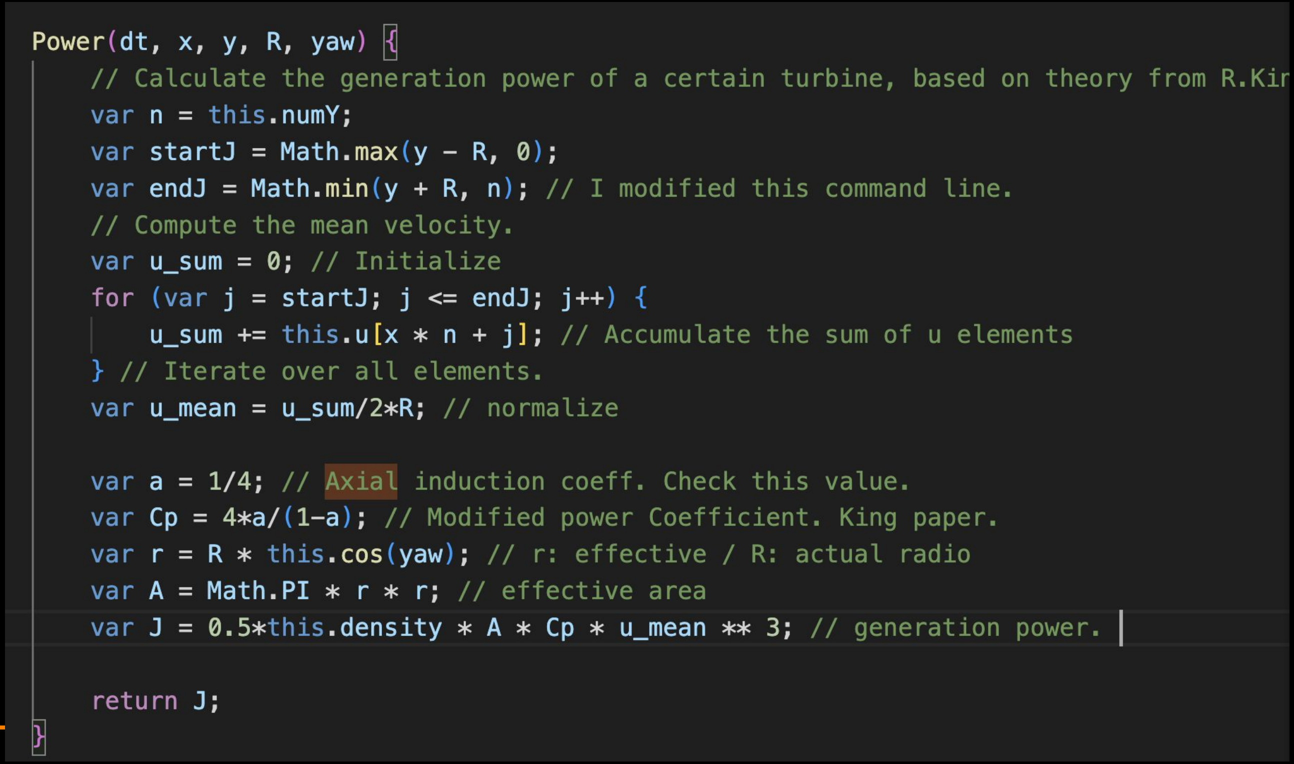
Calculated from the "R" input. $[A = \pi \cdot R^2]$

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;
}
```

## Issues to solve

- On yawed turbines, the force is applied in a slightly longer “blade” due to trigonometric issues on the way it is implemented (using the same number of cells as if it was vertical). I would have to shorten the j loop depending on trigonometric relations.
- I need to add a diffusive term; the wake has to disappear due to velocity diffusion, which is something that is not happening at the moment with the axial induction force from Actuator Disk Theory. It does happen with a higher force due to numerical viscosity; a Von Karman vortex street is created, which is a form of oscillatory vorticity that provokes the dissipation of the wake.

## 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 dont want to sacrifice resolution.
  - Change advection scheme. Too complex
  - Larger timestep. Easy implementation. Does not work.
  - Additional term. Interesting idea.