

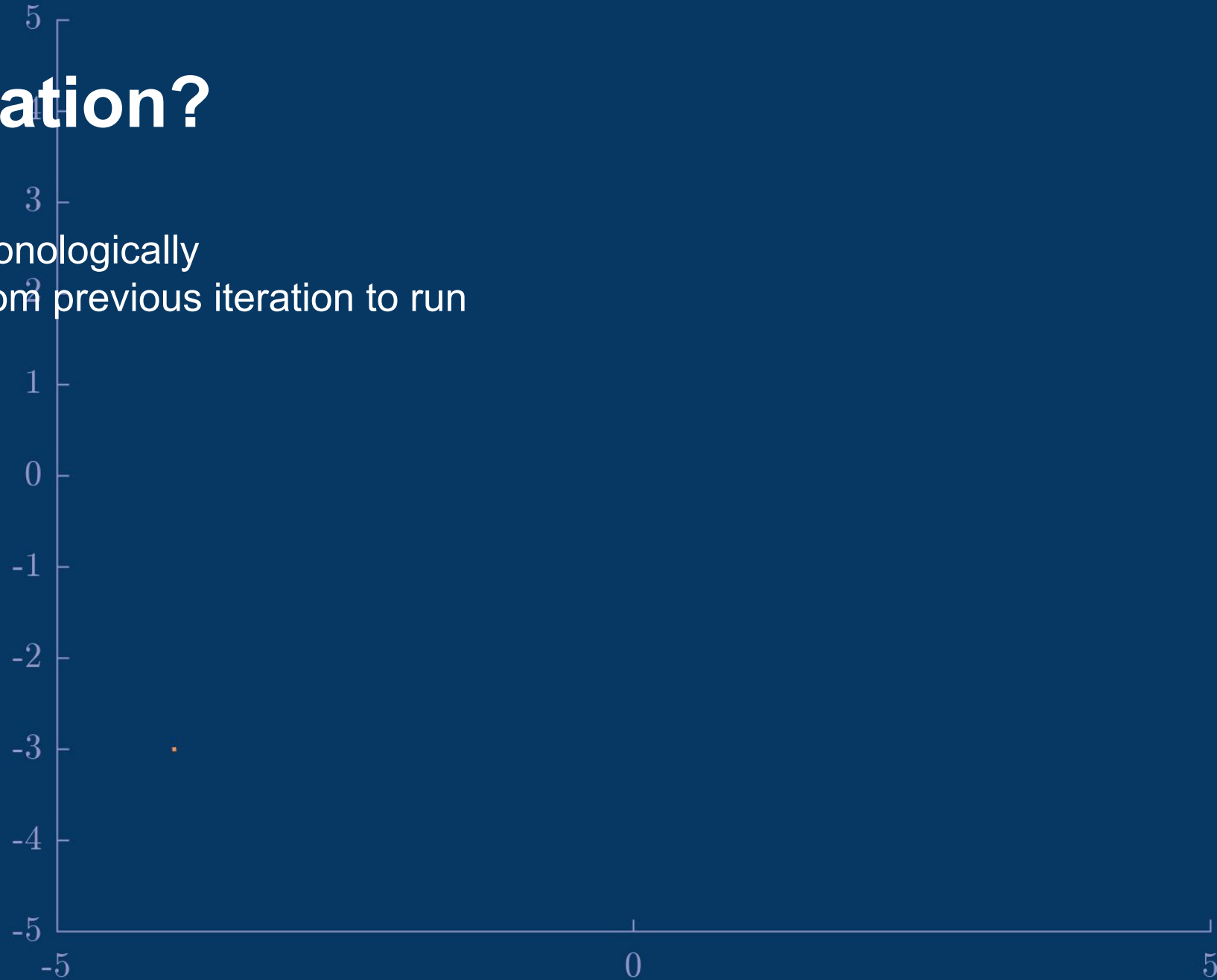
# Parallel Computing Toolbox

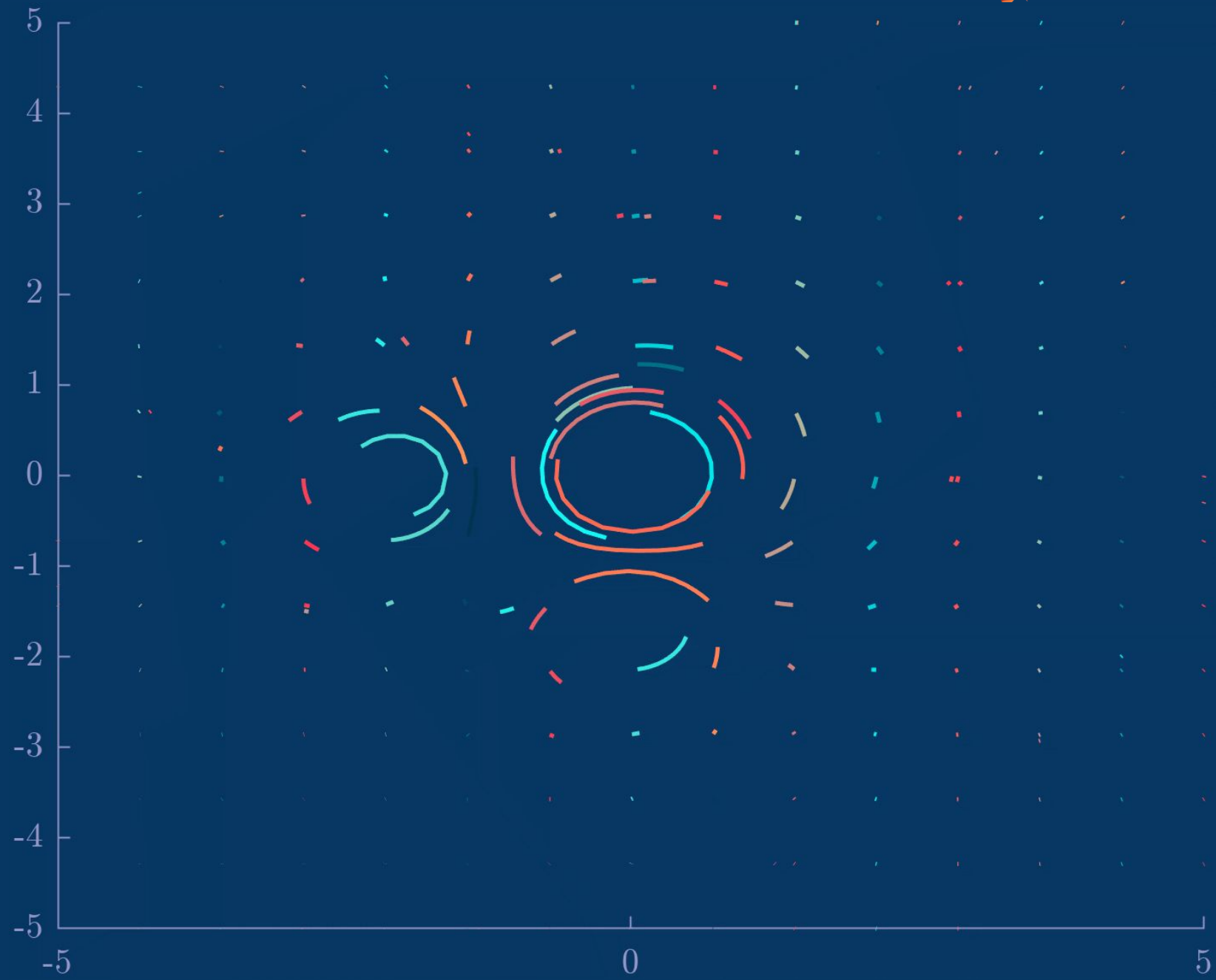
Perform parallel computations on multicore computers, GPUs, and clusters



# What is parallelization?

- Most MATLAB scripts run chronologically
- for-loops, need information from previous iteration to run





# Problems suitable for parallelization:

- Problems where one process is not dependant on the other
- No (\*) need for communication between processes



# Why parallelize?

Well, depends on your problem!

- SPEED AND POWER!!!

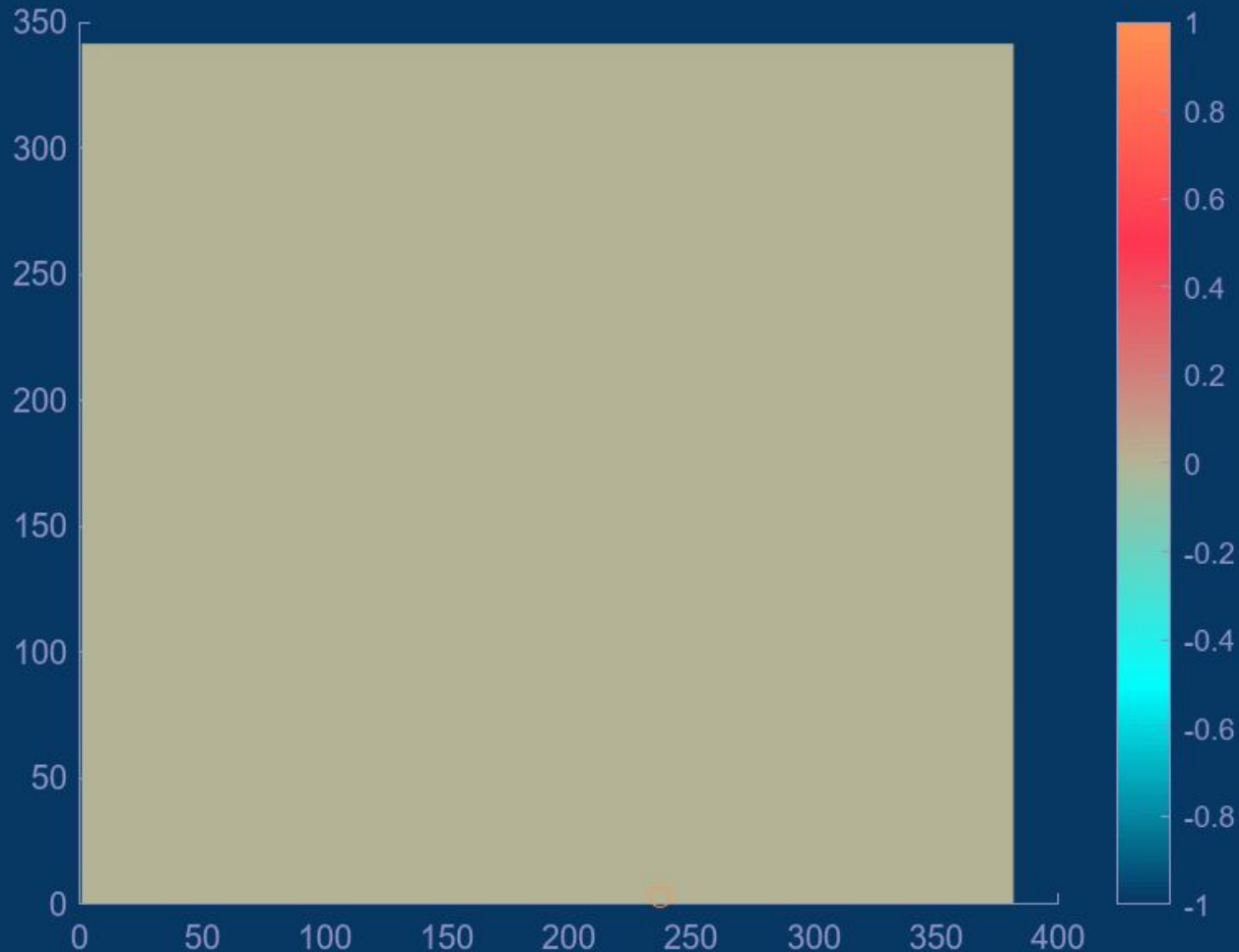


# Different ways of running MATLAB in parallel:

- parfor
- arrayfun, gpuArray
- parfeval
- tall arrays
- batch

```
parfor  
arrayfun()  
gpuArray()  
parfeval()  
batch()
```

# Normal for-loop:



```
X = -2.3:0.01:1.5;
Y = -1.7:0.01:1.7;
iterations = zeros(width(Y), width(X));

for y_index = 1:width(Y)
for x_index = 1:width(X)

    c = X(x_index) + Y(y_index)*1i;
    z = c;
    iteration = 0;

    while true
        z = z^2 + c;

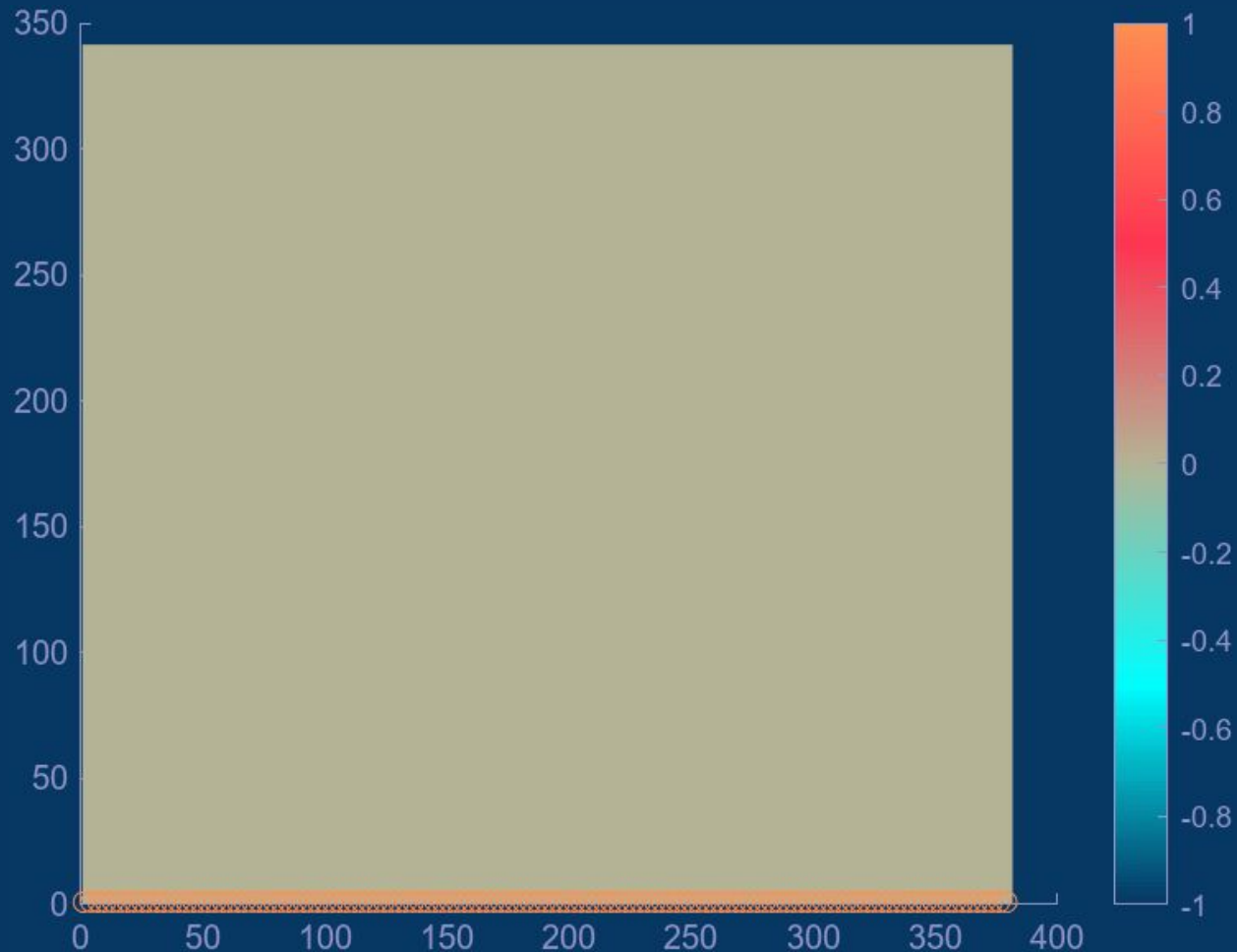
        if iteration > 1000 | norm(z) > 2; break; end

        iteration = iteration +1;
    end

    if iteration < 100; iterations(y_index, x_index) = iteration; end

end
end
```

# parfor-loop:



```
X = -2.3:0.01:1.5;  
Y = -1.7:0.01:1.7;  
iterations = zeros(width(Y), width(X));
```

```
for y_index = 1:width(Y)  
    parfor x_index = 1:width(X)
```

```
        c = X(x_index) + Y(y_index)*1i;  
        z = c;  
        iteration = 0;
```

```
        while true  
            z = z^2 + c;
```

```
            if iteration > 1000 | norm(z) > 2; break; end
```

```
            iteration = iteration + 1;  
        end
```

```
        if iteration < 100; iterations(y_index, x_index) = iteration; end  
    end  
end
```



# parfor-loop:

```
A = 1:2:100;  
B = rand(100);  
C = zeros(100);  
  
parfor i=A  
  
C(i,i) = B(A(i));  
  
end
```

# parfor-loop:

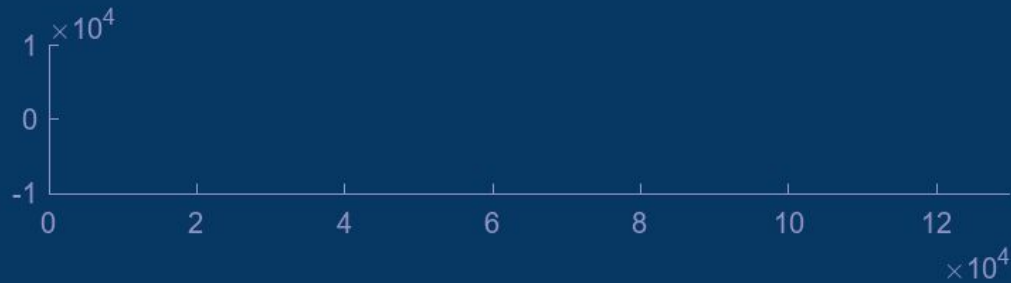
```
A = 1:2:100;  
B = rand(100);  
C = zeros(100);
```

```
parfor i=A
```

! The PARFOR loop cannot run due to the way variable 'C' is used. [Details ▼](#)

```
end
```

# parfor-loop:



```
X = -2.3:0.01:1.5;
Y = -1.7:0.01:1.7;
iterations = zeros(width(Y), width(X));
C = X + Y'*1i;
```

```
iterations = reshape(iterations,1,[]);
C           = reshape(C,1,[]);
```

```
parfor index = 1:width(C)

    c = C(index);
    z = c;
    iteration = 0;

    while true
        z = z^2 + c;

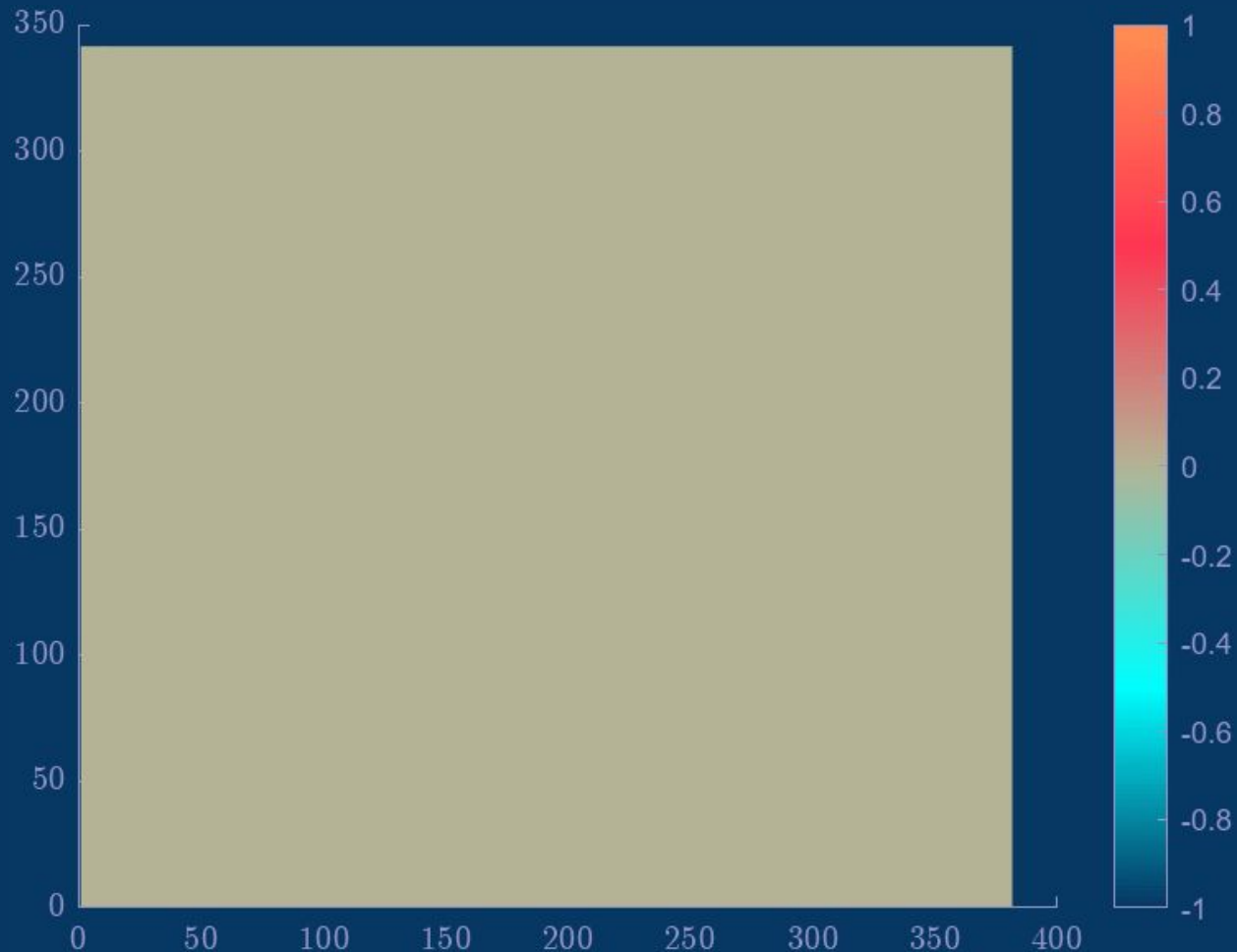
        if iteration > 1000 | norm(z) > 2; break; end

        iteration = iteration +1;
    end

    if iteration < 100; iterations(index) = iteration; end
end
```

```
iterations = reshape(iterations, width(Y), []);
```

# Vectorizing and arrayfun



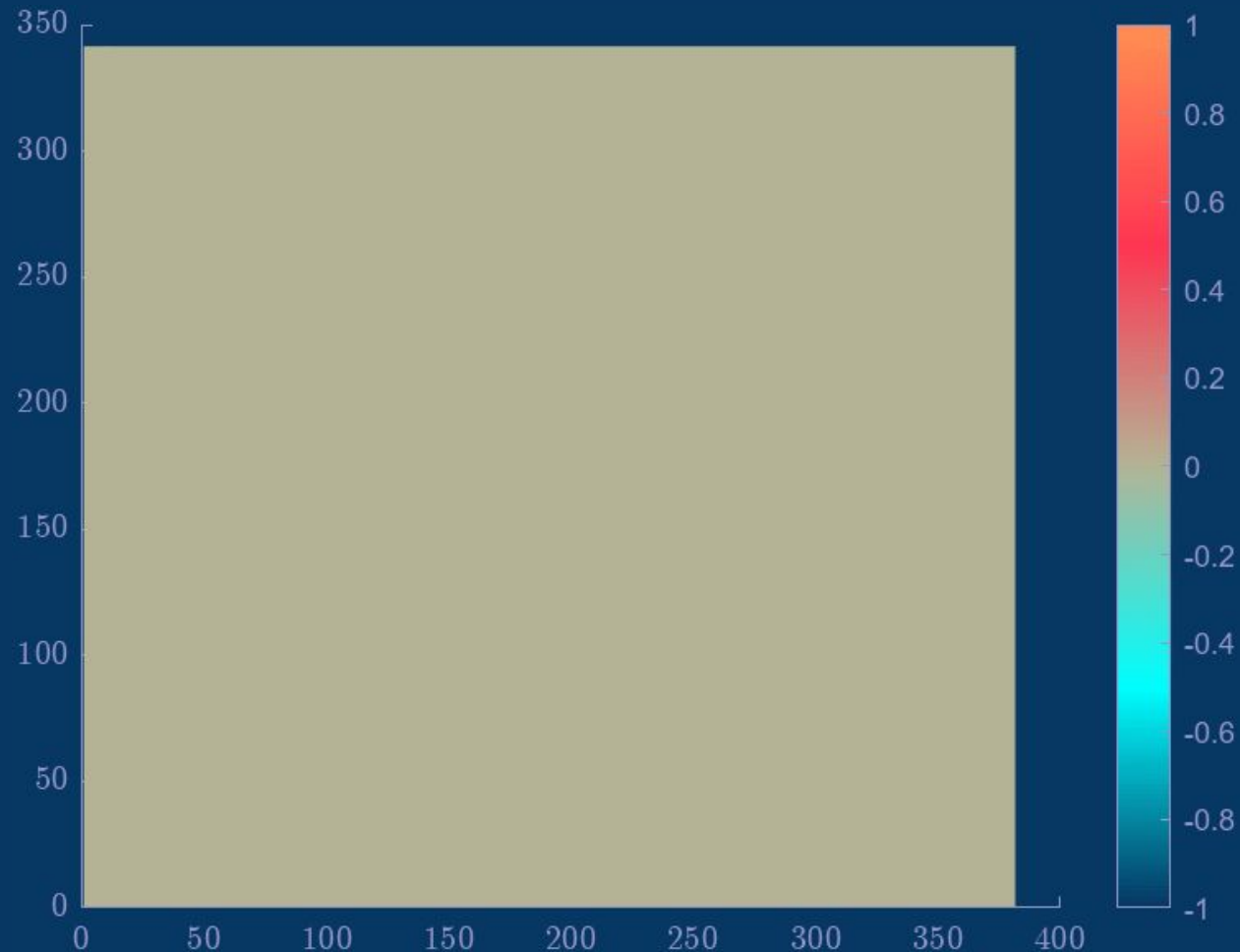
```
X = -2.3:0.01:1.5;  
Y = -1.7:0.01:1.7;  
  
C = X + Y'*1i;  
  
iterations = arrayfun(@evaluate_iterations, C);
```

```
function iteration = evaluate_iterations(c)  
  
    z = c;  
    iteration = 0;  
  
    while 100 > iteration && 2 > norm(z)  
  
        z = z^2 + c;  
        iteration = iteration + 1;  
    end  
  
    if iteration == 100; iteration = 0; end  
end
```

# Vectorizing and arrayfun

```
A = rand(100);  
B = rand(100);  
  
C = A + B;  
D = A.*B;  
E = arrayfun( @(a,b) a + b, A, B)
```

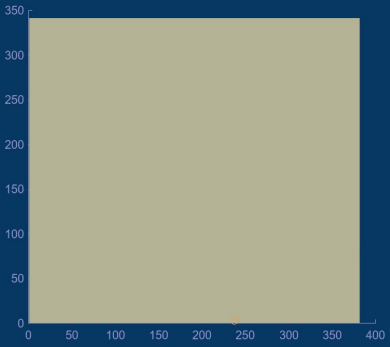
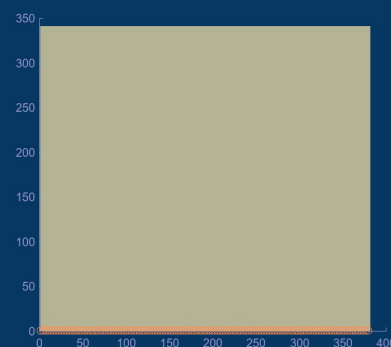
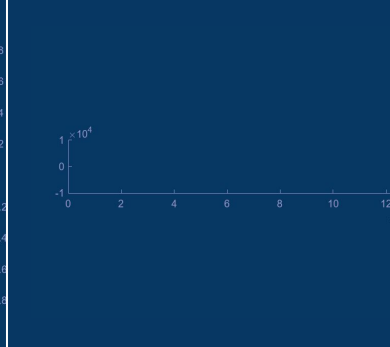
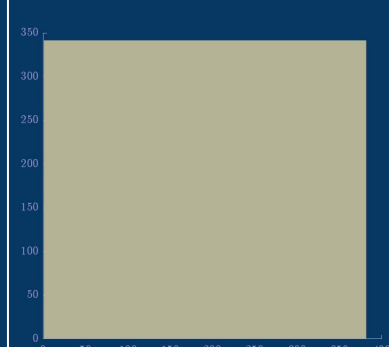
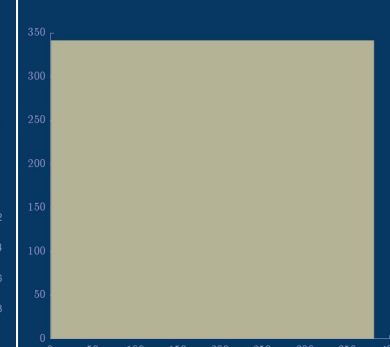
# gpuArray:



```
X = -2.3:0.01:1.5;  
Y = -1.7:0.01:1.7;  
  
C = X + Y'*1i;  
C = gpuArray(C);  
  
iterations = arrayfun(@evaluate_iterations, C);
```

```
function iteration = evaluate_iterations(c)  
  
    z = c;  
    iteration = 0;  
  
    while 100 > iteration && 2 > norm(z)  
  
        z = z^2 + c;  
        iteration = iteration + 1;  
    end  
  
    if iteration == 100; iteration = 0; end  
end
```

# Benchmark $10^5$ datapoints:

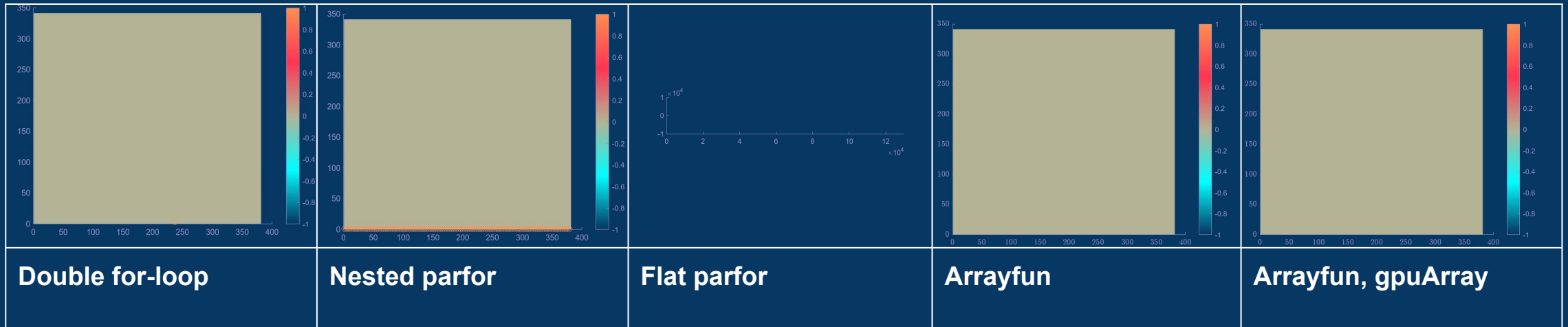
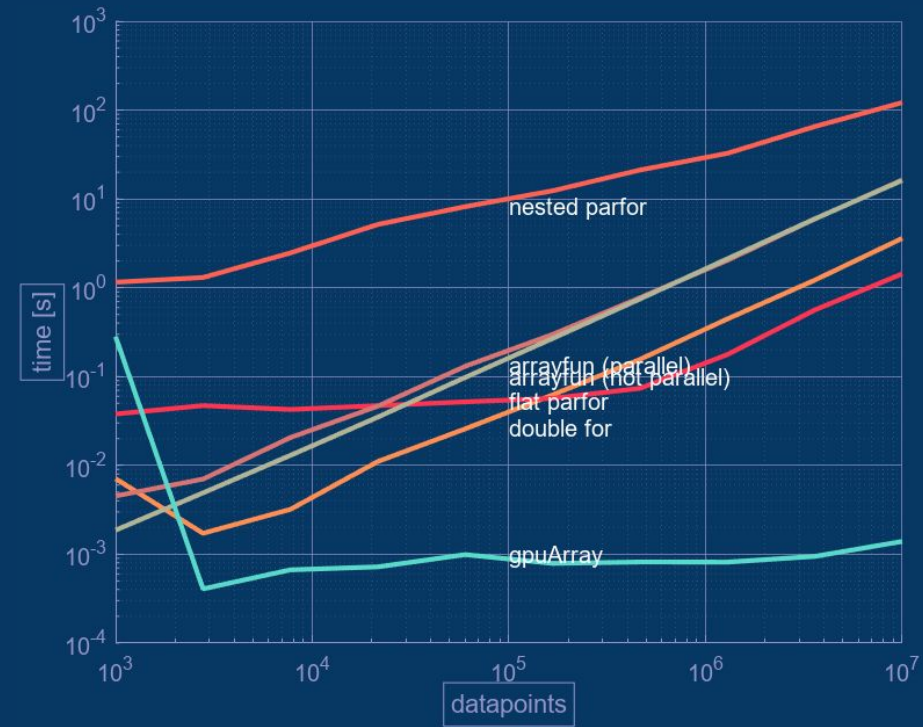
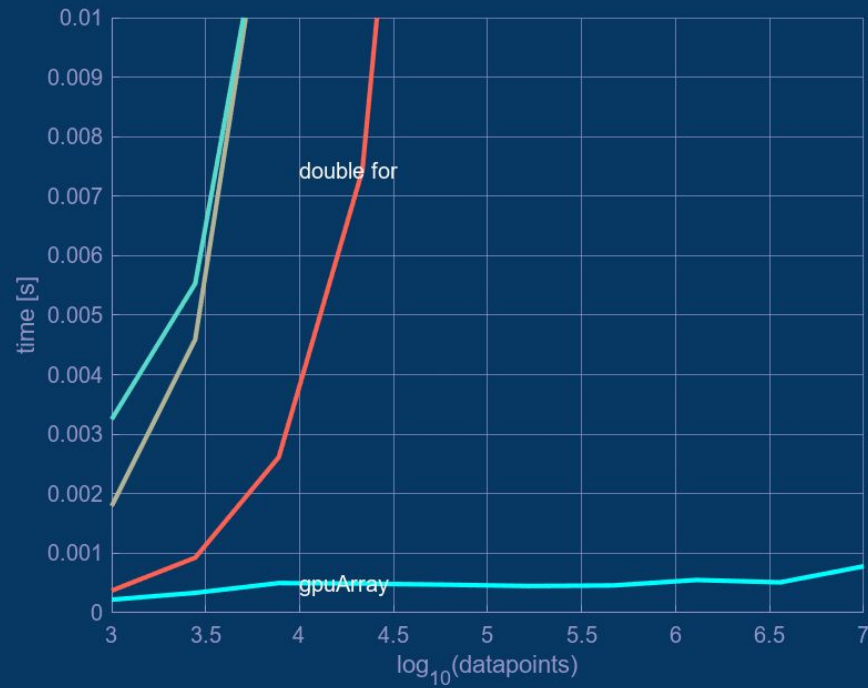
					
	<b>Double for-loop</b>	<b>Nested parfor</b>	<b>Flat parfor</b>	<b>Arrayfun</b>	<b>Arrayfun, gpuArray</b>
s	0.026	8.192	0.052	0.097 (not parallel) 0.1306 (parallel)	0.00099
ms	26	8 192	52	97 130	0.99

## Computer specs:

Processor: 11th Gen Intel(R) Core(TM) i7-11700 @ 2.50GHz, 2496 Mhz, 8 Core(s), 16 Logical Processor(s)

GPU: NVIDIA Quadro P1000

Ram: 32 GB



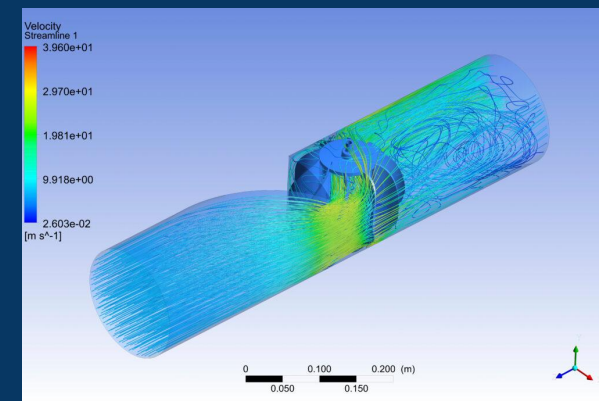
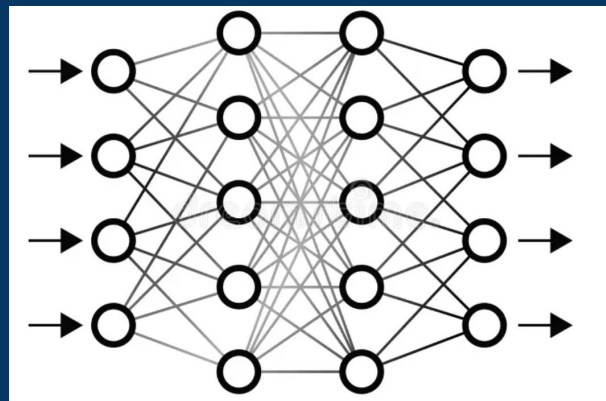
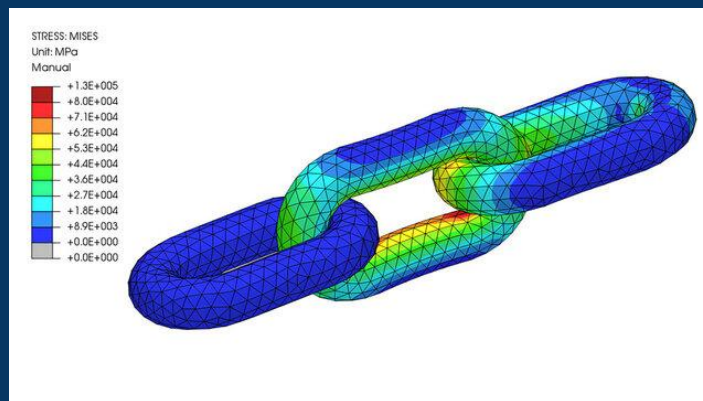


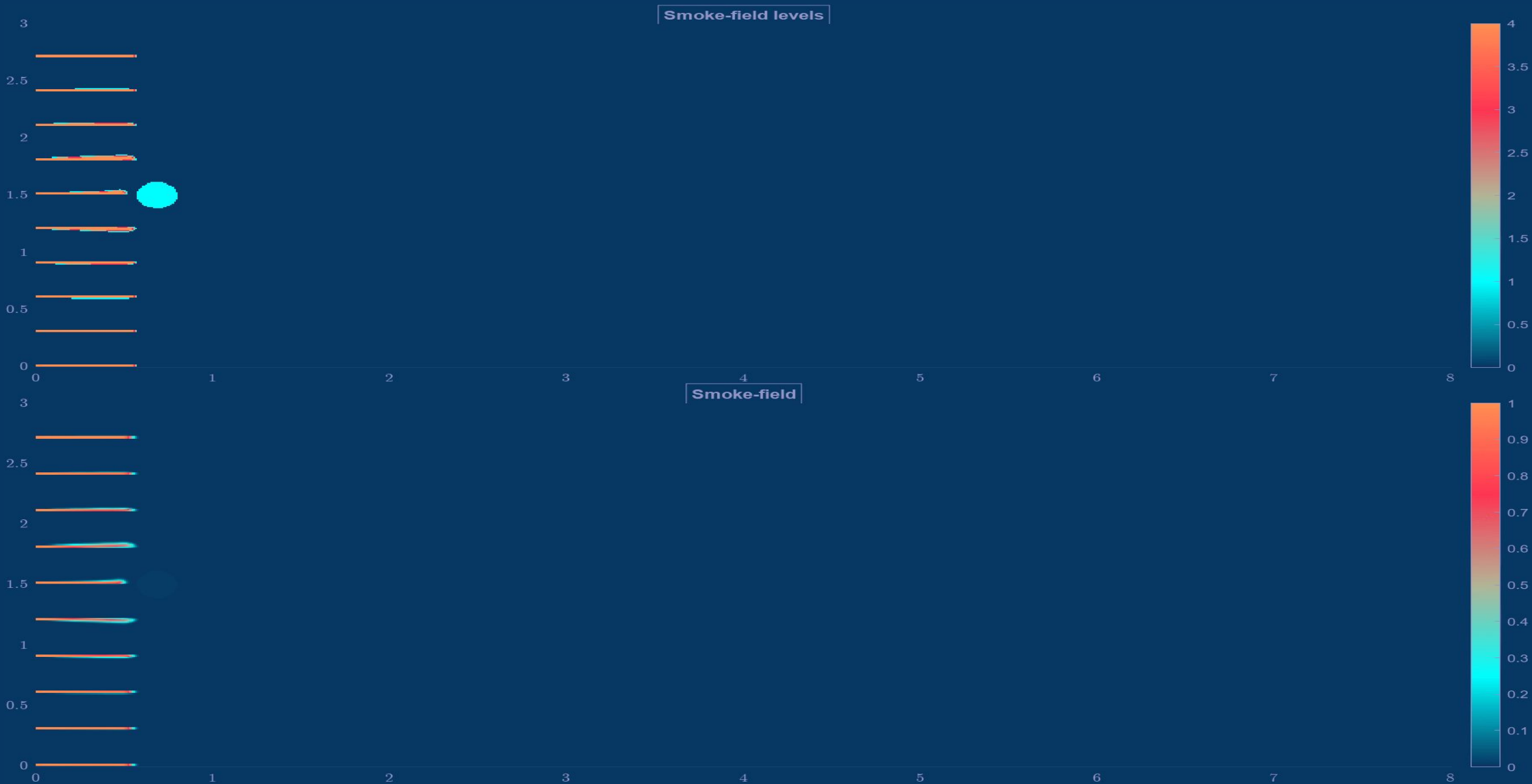
# Applications?

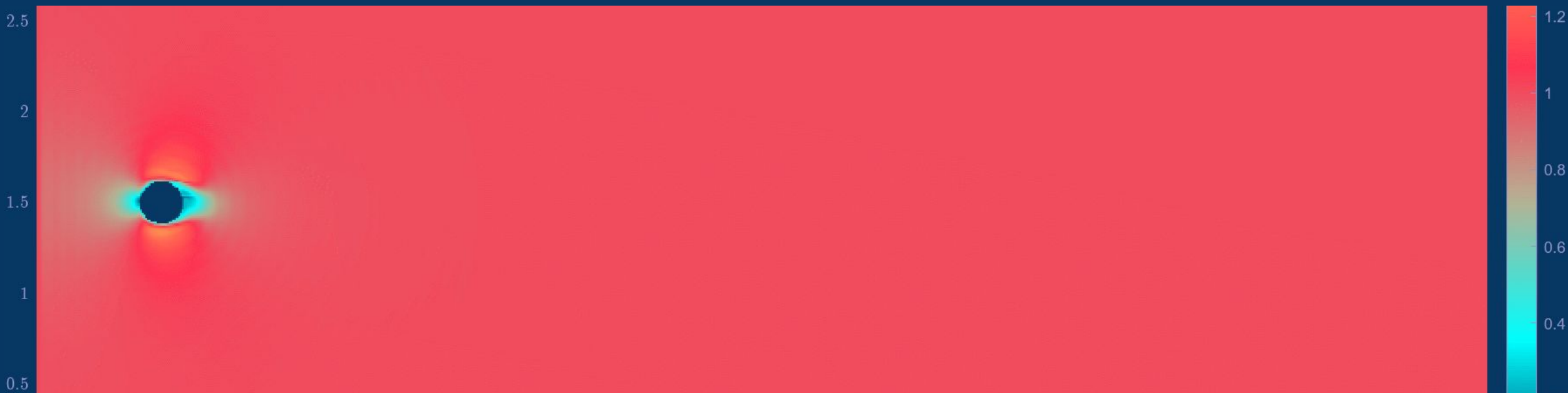
# Applications?

...Well there are many!

- AI, neural networks & machine learning
- Graphics
- Solving matrix-problems
- Fluid mechanics, CFD

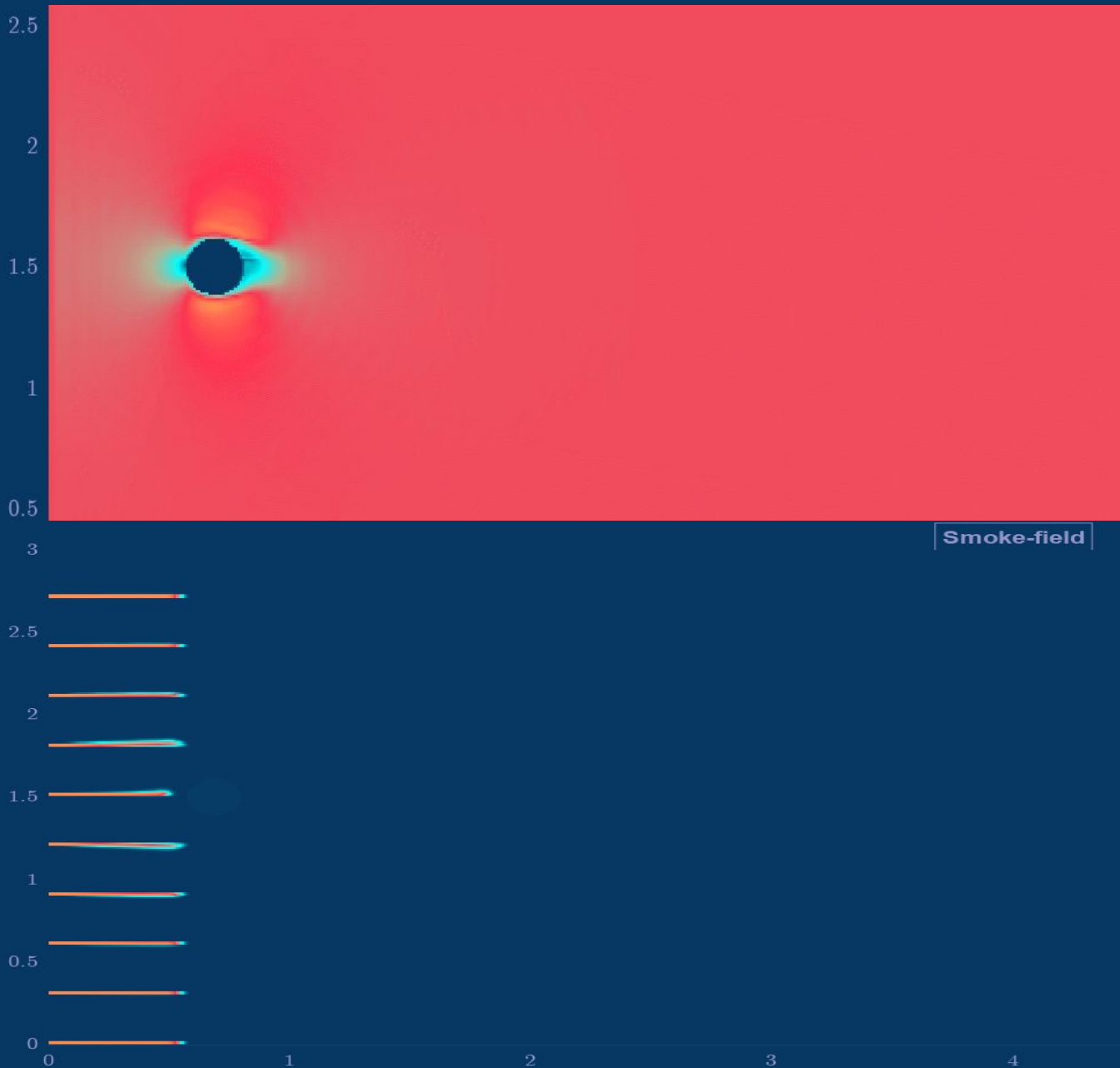






Smoke-field





With gpuArray:

0.0778 s/iteration = **77.84 ms/it**

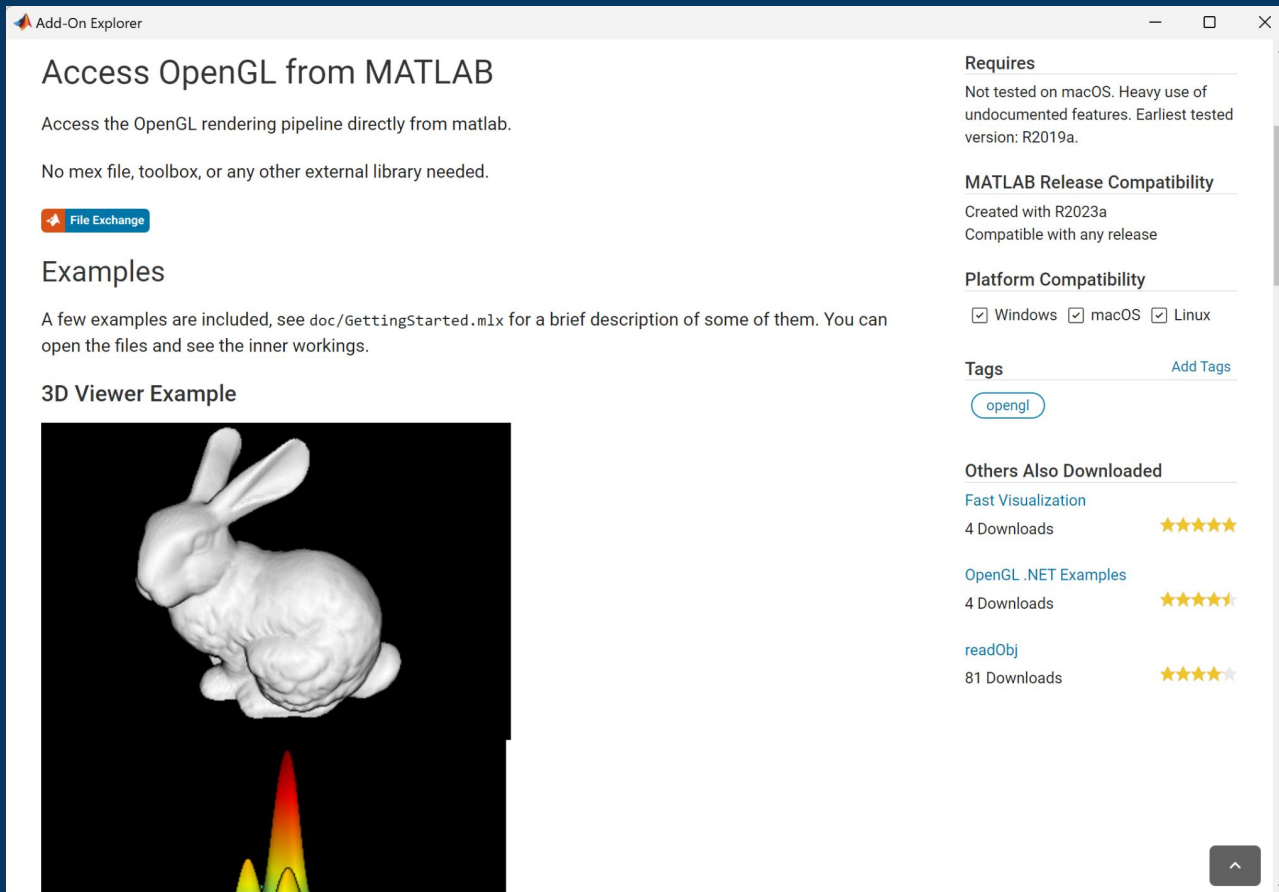
Without gpuArray:

1.7432 s/iteration = **1743.2 ms/it**

(we want the parallel computing toolbox to support Intel and AMD GPUs!!! >:( )

# Access OpenGL from MATLAB

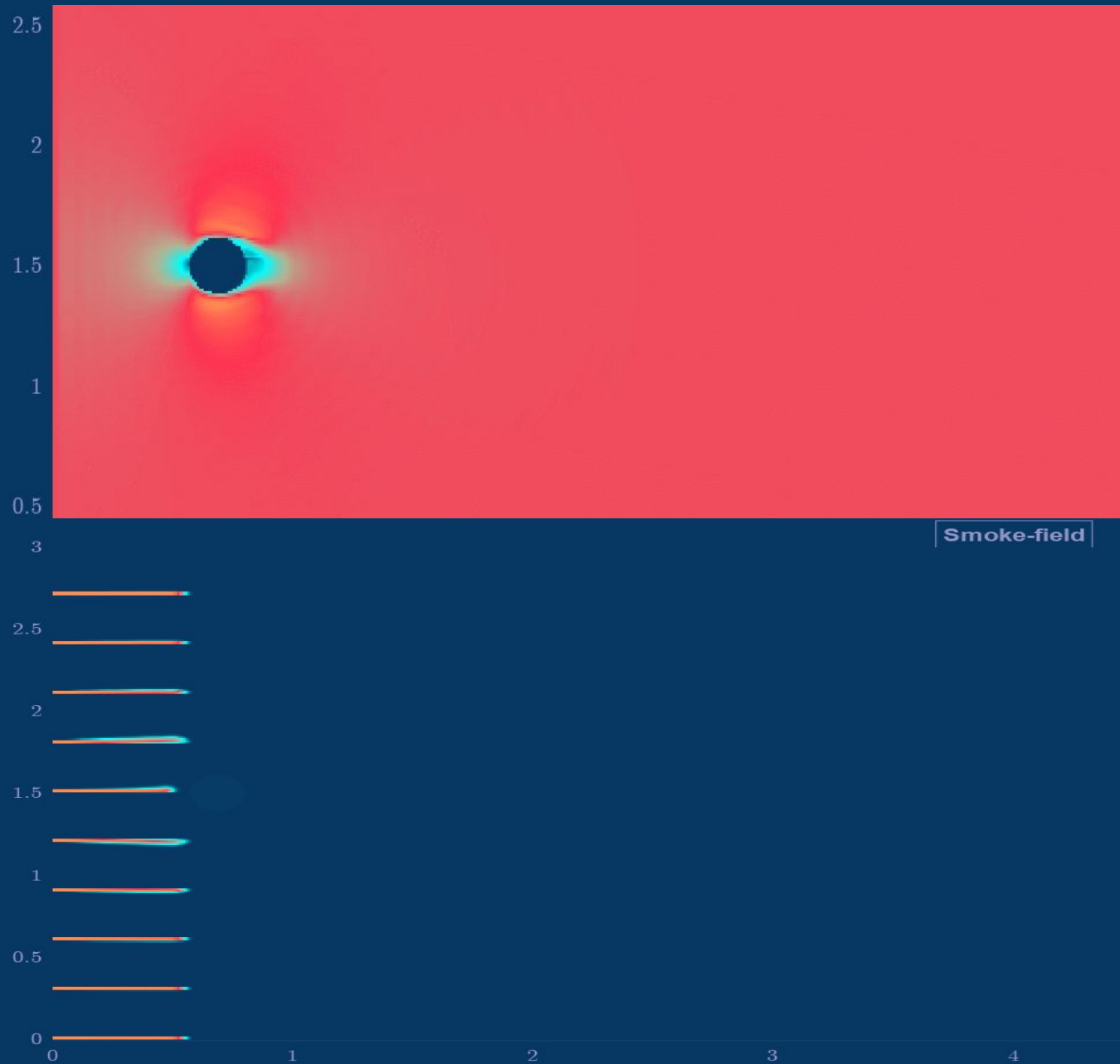
A standalone library developed by community-member **Francis Plamondon**



The screenshot shows the MATLAB Add-On Explorer window for the 'Access OpenGL from MATLAB' add-on. The window title is 'Add-On Explorer'. The main content area displays the add-on's name, a brief description ('Access the OpenGL rendering pipeline directly from matlab.'), and a note that no mex file, toolbox, or other external library is needed. There is a 'File Exchange' button. Below this, the 'Examples' section mentions that a few examples are included and refers to a 'doc/GettingStarted.mlx' file. A '3D Viewer Example' is shown with a 3D rendering of a white rabbit on a black background, with a small color bar at the bottom. The right sidebar contains metadata: 'Requires' (Not tested on macOS, heavy use of undocumented features, earliest tested version: R2019a), 'MATLAB Release Compatibility' (Created with R2023a, Compatible with any release), 'Platform Compatibility' (checked for Windows, macOS, and Linux), 'Tags' (a tag for 'opengl'), and 'Others Also Downloaded' (listing 'Fast Visualization' with 4 downloads and 5 stars, 'OpenGL .NET Examples' with 4 downloads and 5 stars, and 'readObj' with 81 downloads and 5 stars).

<https://github.com/fr0nkk/matogl>





```
for iteration = 1:max_iterations  
  
    projection  
    staggered2centered  
    advection  
    viscosity  
    centered2staggered  
    stabilize  
  
end
```

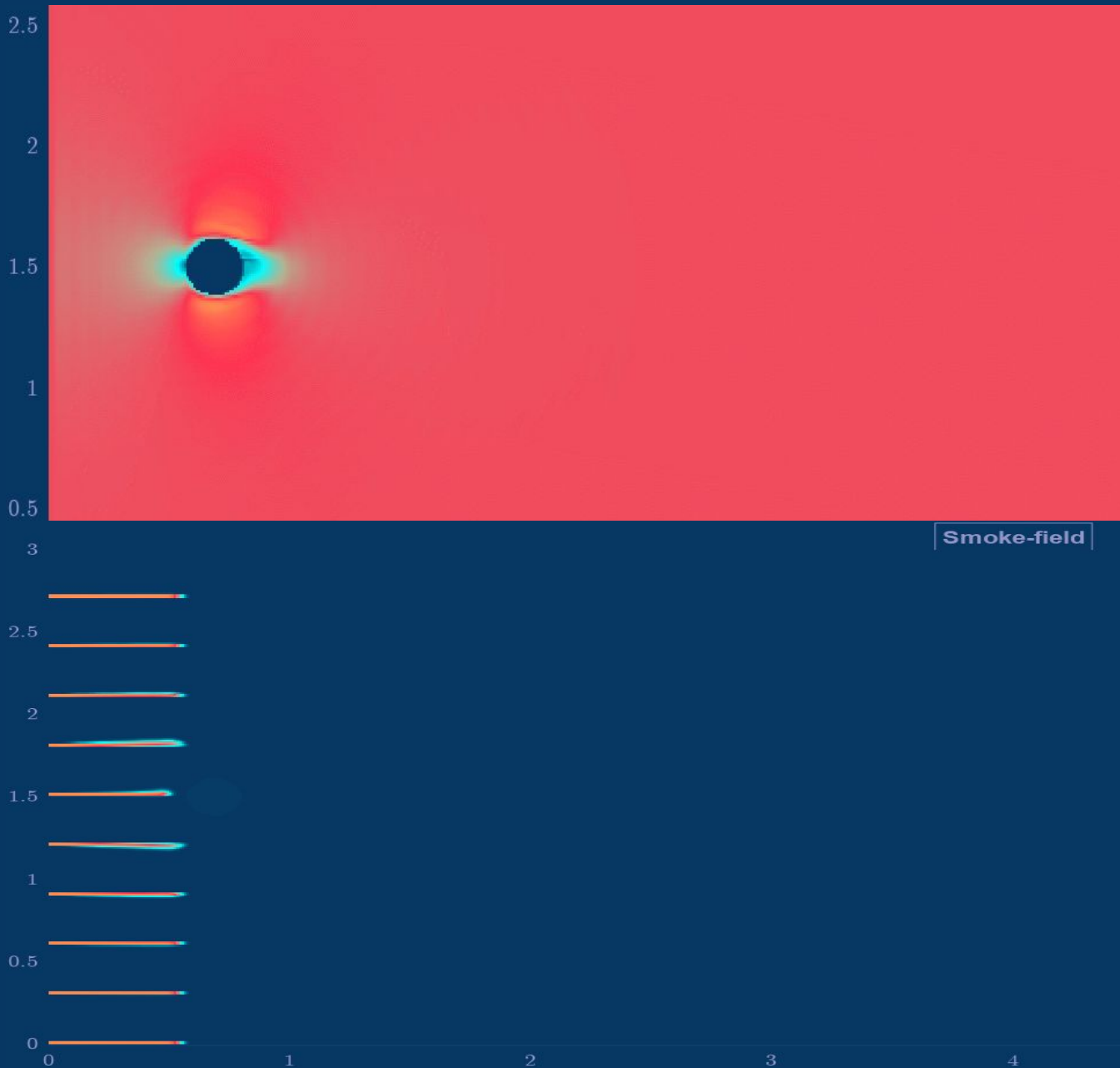
projection



advection



viscosity



The project is up on **Github** and the **MATLAB file-exchange** for anyone interested!

(Just don't use it for anything important bcs it kinda bad 🤪🤪🤪)

<https://github.com/spiggen/MATLAB-fluid-sim>





# Parallel Computing with MATLAB: Hands-on workshop

24th of April 2024, 13:00

- Multithreading vs multiprocessing
- When to use parfor vs parfeval constructs
- Creating data queues for data transfer
- Leveraging NVIDIA GPUs
- Parallelizing Simulink models
- Working with large data



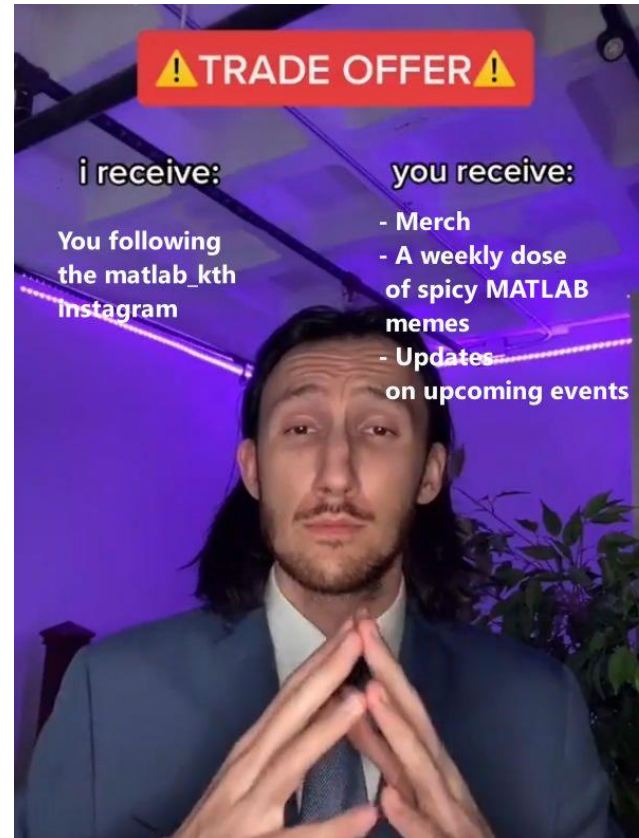
[tinyurl.com/ParallelComputing24](https://tinyurl.com/ParallelComputing24)

# Merch giveaway requirements:

- Follow matlab\_kth on instagram Or join the facebook-group!



matlab\_kth  
facebook  
group



matlab\_kth  
instagram