Python and Numerical Methods

Physics Simulation

Objectives

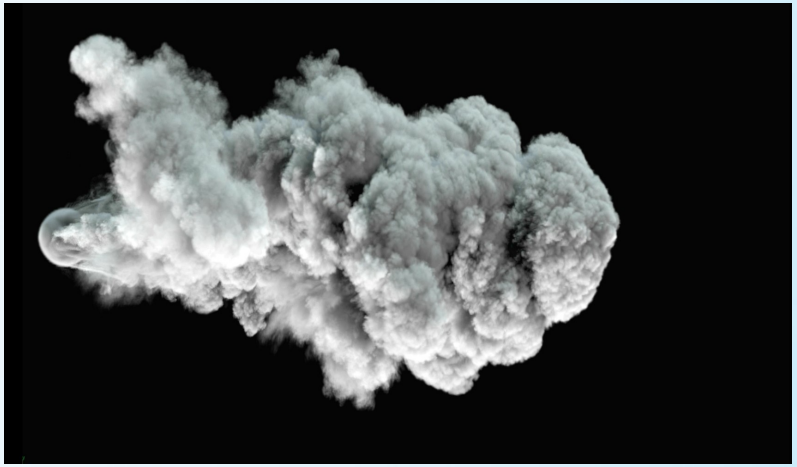
- Python Ecosystem
- Performance Tests
- Examples – 2D simulation

Concepts

Physics Simulation

Physics Simulation

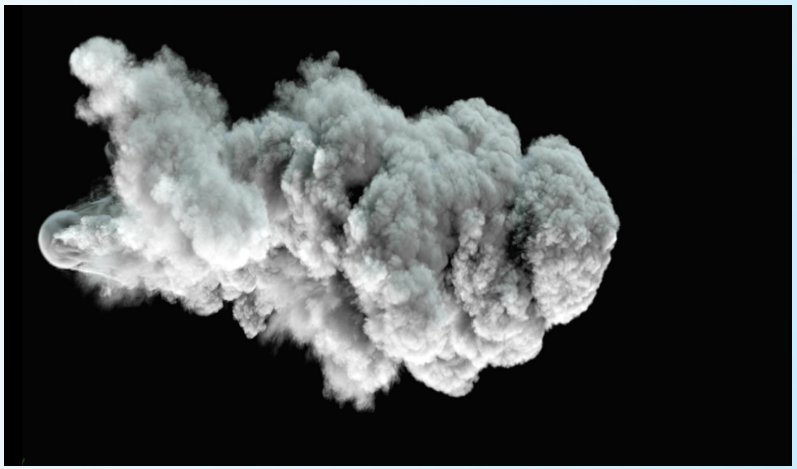
Basic Concepts



Physics Simulation

Basic Concepts

Point / Navier Stokes



Cell / Navier Stokes

Triangles / Hookes Law

Prototype

- N-dimensional array
- Apply function on multiple data (array)
- Easy to code, do not re-invent the wheel
- Fast to evaluate a simulation
- Open Source

Prototype

- N-dimensional array
- Apply function on multiple data (array)
- Easy to code, do not re-invent the wheel
- Fast to evaluate a simulation
- Open Source

C++ is fast but bad for prototyping...
Lets find something else !

Python Ecosystem

Numerical Methods

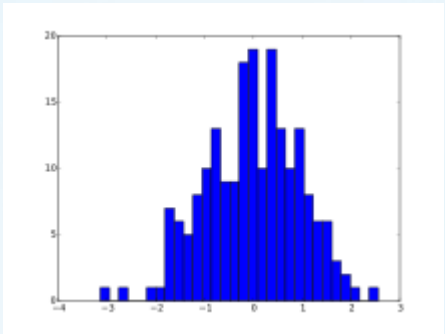
Prototype a solver –

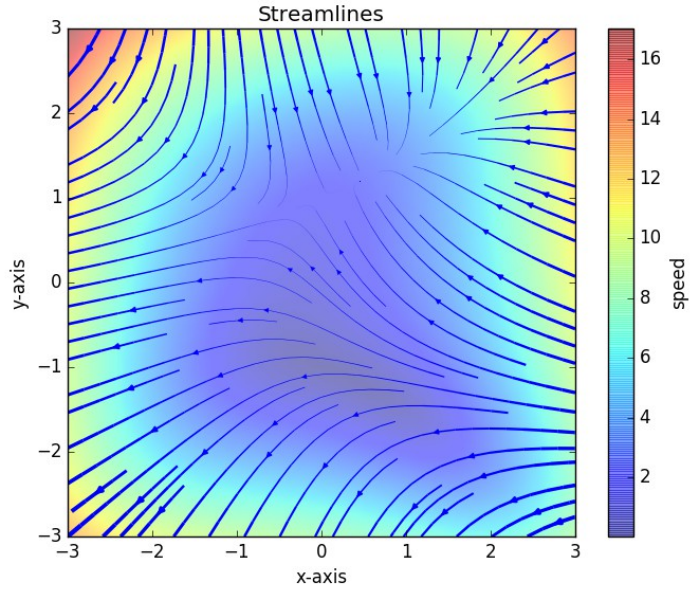
- N-dimensional array
- Memory alignment

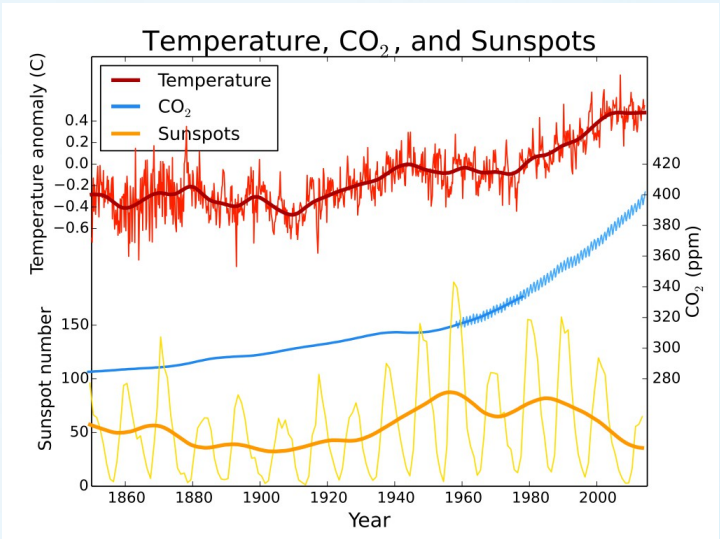
Prototype a solver –

- Optimization
- Linear Algebra
- Signal Processing
- ODE Solver
- ...

Prototype a solver –



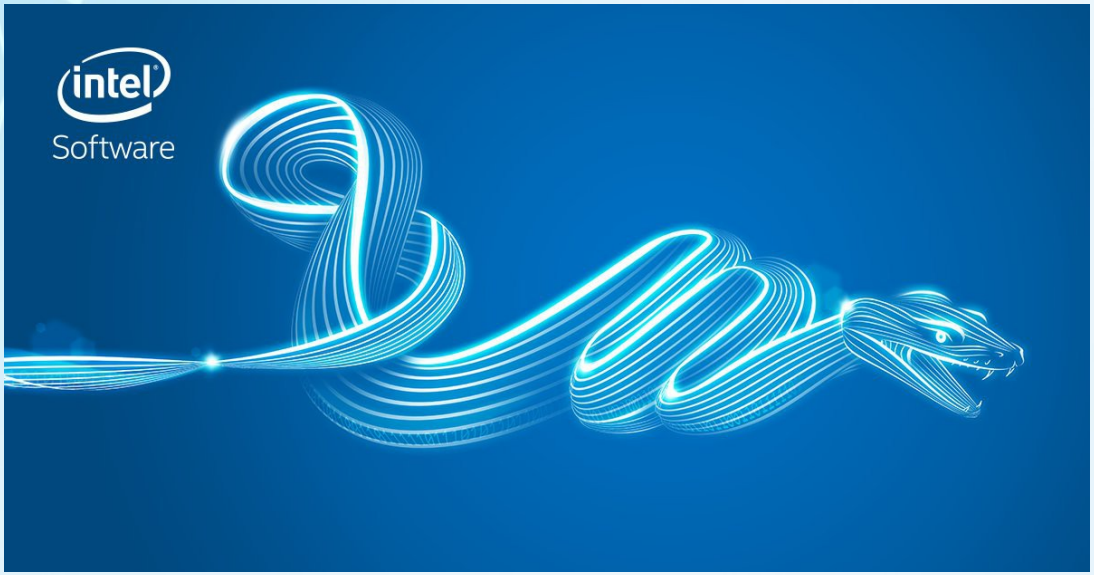




Prototype a solver –

- Translates Python functions to optimized machine code at runtime
- CPU and GPU support

Prototype a solver –



Accelerate compute-intense applications—including numeric, scientific, data analytics, machine learning–that use NumPy, SciPy

Performance Tests

1D-Array Test

Numba Introduction

Objectives

. Update random array with function

. Test Numba

```
def create_data(N):
    return np.random.rand(N)
```

```
def function(value):
    return math.sqrt(math.tan(value) * value * math.cos(value))
```

```
def run(array):
    for i in range(array.shape[0]):
        array[i] = function(array[i])
    return array
```

	Standard	Numba	Numba Parallel
1e6	0.295 sec		
1e7	3.01 sec		
1e8	29.50 sec		
1e9	-		

1D-Array Test

Numba Introduction

Objectives

. Update random array with function

. Test Numba

```
def create_data(N):
    return np.random.rand(N)

@njit
def function(value):
    return math.sqrt(math.tan(value) * value * math.cos(value))

@njit
def run(array):
    for i in range(array.shape[0]):
        array[i] = function(array[i])
    return array
```

	Standard	Numba	Numba Parallel
1e6	0.295 sec	0.009 sec	
1e7	3.01 sec	0.069 sec	
1e8	29.50 sec	0.689 sec	
1e9	-	10.939 sec	

1D-Array Test

Numba Introduction

Objectives

. Update random array with function

. Test Numba

```
def create_data(N):
    return np.random.rand(N)

@njit
def function(value):
    return math.sqrt(math.tan(value) * value * math.cos(value))

@njit(parallel=True)
def run(array):
    for i in prange(array.shape[0]):
        array[i] = function(array[i])
    return array
```

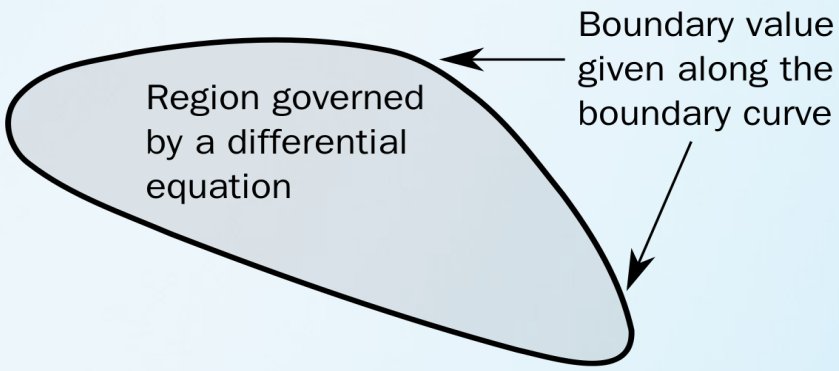
	Standard	Numba	Numba Parallel
1e6	0.295 sec	0.009 sec	0.002 sec
1e7	3.01 sec	0.069 sec	0.016 sec
1e8	29.50 sec	0.689 sec	0.165 sec
1e9	-	10.939 sec	1.681 sec

2D-Stencil Test

Boundary value problem

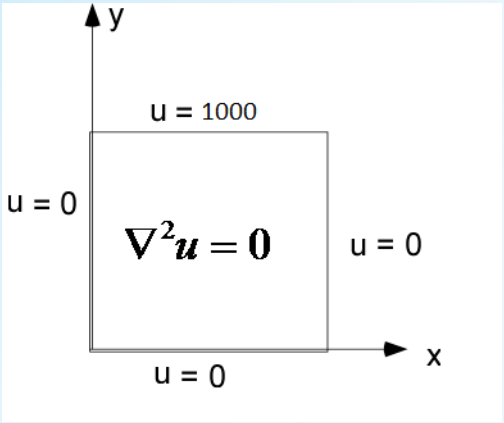
Objectives

. Solve Laplace's Equation
. CPU / GPU
. Stencil operations

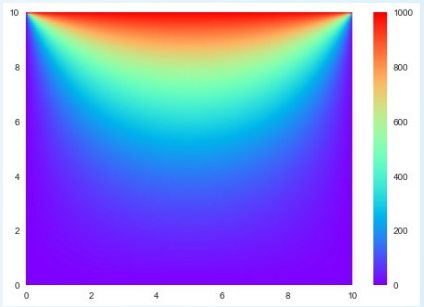


2D-Stencil Test

Boundary value problem



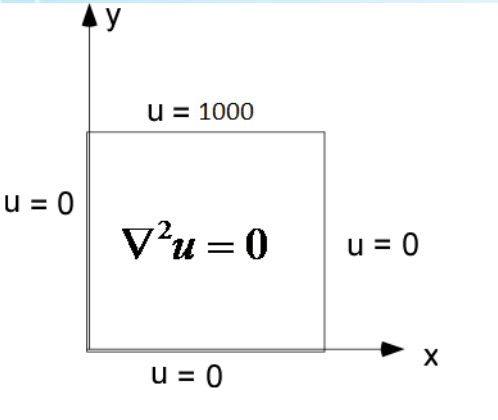
Problem



Solution

2D-Stencil Test

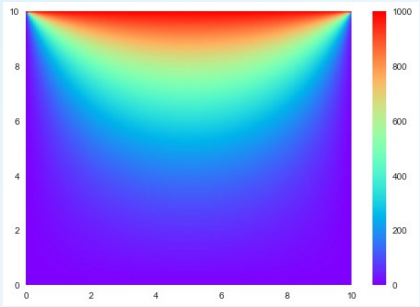
Boundary value problem



Solve a linear system Au = b

A : sparse matrix [N^2, N^2]
u : unknown [N^2]
b : zero vector [N^2]

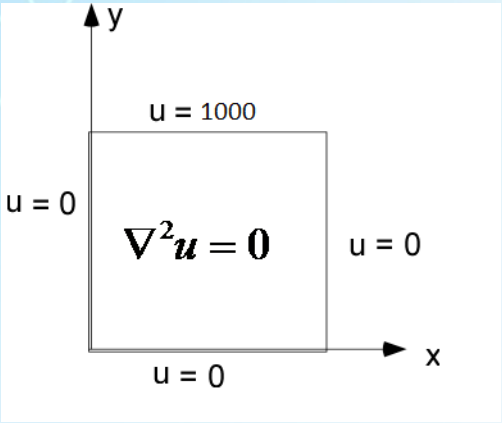
N : number of values along on axis (x, y)



Solve Au = b with Jacobi Iterative Method

2D-Stencil Test

Boundary value problem

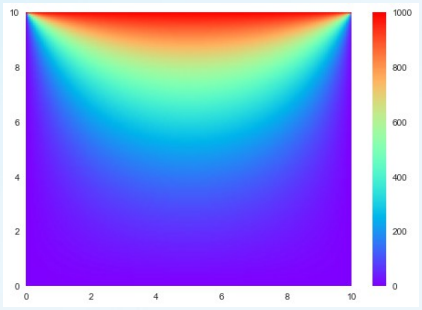


Solve a linear system Au = b

A : sparse matrix [N^2, N^2]
u : unknown [N^2]
b : zero vector [N^2]

N : number of values along on axis (x, y)

Solve Au = b with Jacobi Iterative Method



2D-Stencil Test

Boundary value problem

```
def create_domain(N):
    values = zeros((num_nodes, num_nodes), np.float)
    values[num_nodes-1:num_nodes] = 1000.0


def iteration(x, next_x):
    num_dof_x = x.shape[0]
    num_dof_y = x.shape[1]

    for i in range(1, num_dof_x - 1):
        for j in range(1, num_dof_y - 1):
            next_x[i][j] = (x[i - 1][j] +
                            x[i + 1][j] +
                            x[i][j - 1] +
                            x[i][j + 1]) * 0.25
```

	Standard	Numba	Numba Parallel	Numba Cuda
64x64 2000 iter				
128x128 10000 iter.				
256x256 50000 iter.				
512x512 250000 iter.				

2D-Stencil Test

Boundary value problem

Call that many times

```
def create_domain(N):
    values = zeros((num_noc                                    t)
    values[num_nodes-1:num_n                                   ]
```

```
def iteration(x, next_x):
    num_dof_x = x.shape[0]
    num_dof_y = x.shape[1]

    for i in range(1, num_dof_x - 1):
        for j in range(1, num_dof_y - 1):
            next_x[i][j] = (x[i - 1][j] +
                            x[i + 1][j] +
                            x[i][j - 1] +
                            x[i][j + 1]) * 0.25
```

	Standard	Numba	Numba Parallel	Numba Cuda
64x64 2000 iter	9.696 sec			
128x128 10000 iter.	-			
256x256 50000 iter.	-			
512x512 250000 iter.	-			

2D-Stencil Test

Boundary value problem

```
def create_domain(N):
    values = zeros((num_nodes, num_nodes), np.float)
    values[num_nodes-1:num_nodes] = 1000.0


@njit
def iteration(x, next_x):
    num_dof_x = x.shape[0]
    num_dof_y = x.shape[1]

    for i in range(1, num_dof_x - 1):
        for j in range(1, num_dof_y - 1):
            next_x[i][j] = (x[i - 1][j] +
                            x[i + 1][j] +
                            x[i][j - 1] +
                            x[i][j + 1]) * 0.25
```

	Standard	Numba	Numba Parallel	Numba Cuda
64x64 2000 iter	9.696 sec	0.007 sec		
128x128 10000 iter.	-	0.133 sec		
256x256 50000 iter.	-	2.459 sec		
512x512 250000 iter.	-	50.343 sec		

2D-Stencil Test

Boundary value problem

```
def create_domain(N):
    values = zeros((num_nodes, num_nodes), np.float)
    values[num_nodes-1:num_nodes] = 1000.0


@njit(parallel=True)
def iteration(x, next_x):
    num_dof_x = x.shape[0]
    num_dof_y = x.shape[1]

    for i in prange(1, num_dof_x - 1):
        for j in range(1, num_dof_y - 1):
            next_x[i][j] = (x[i - 1][j] +
                            x[i + 1][j] +
                            x[i][j - 1] +
                            x[i][j + 1]) * 0.25
```

	Standard	Numba	Numba Parallel	Numba Cuda
64x64 2000 iter	9.696 sec	0.007 sec	0.012 sec	
128x128 10000 iter.	-	0.133 sec	0.131 sec	
256x256 50000 iter.	-	2.459 sec	1.401 sec	
512x512 250000 iter.	-	50.343 sec	19.872 sec	

2D-Stencil Test

Boundary value problem

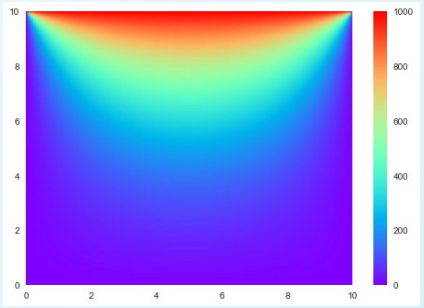
```
def create_domain(N):
    values = zeros((num_nodes, num_nodes), np.float)
    values[num_nodes-1:num_nodes] = 1000.0


@cuda.jit
def cuda_iteration(x, next_x):
      i, j = cuda.grid(2)
      next_x[i][j] = (x[i - 1][j] +
                      x[i + 1][j] +
                      x[i][j - 1] +
                      x[i][j + 1]) * 0.25
```

	Standard	Numba	Numba Parallel	Numba Cuda
64x64 2000 iter	9.696 sec	0.007 sec	0.012 sec	0.34 sec
128x128 10000 iter.	-	0.133 sec	0.131 sec	1.21 sec
256x256 50000 iter.	-	2.459 sec	1.401 sec	5.56 sec
512x512 250000 iter.	-	50.343 sec	19.872 sec	27.30 sec

2D-Stencil Test

Boundary value problem

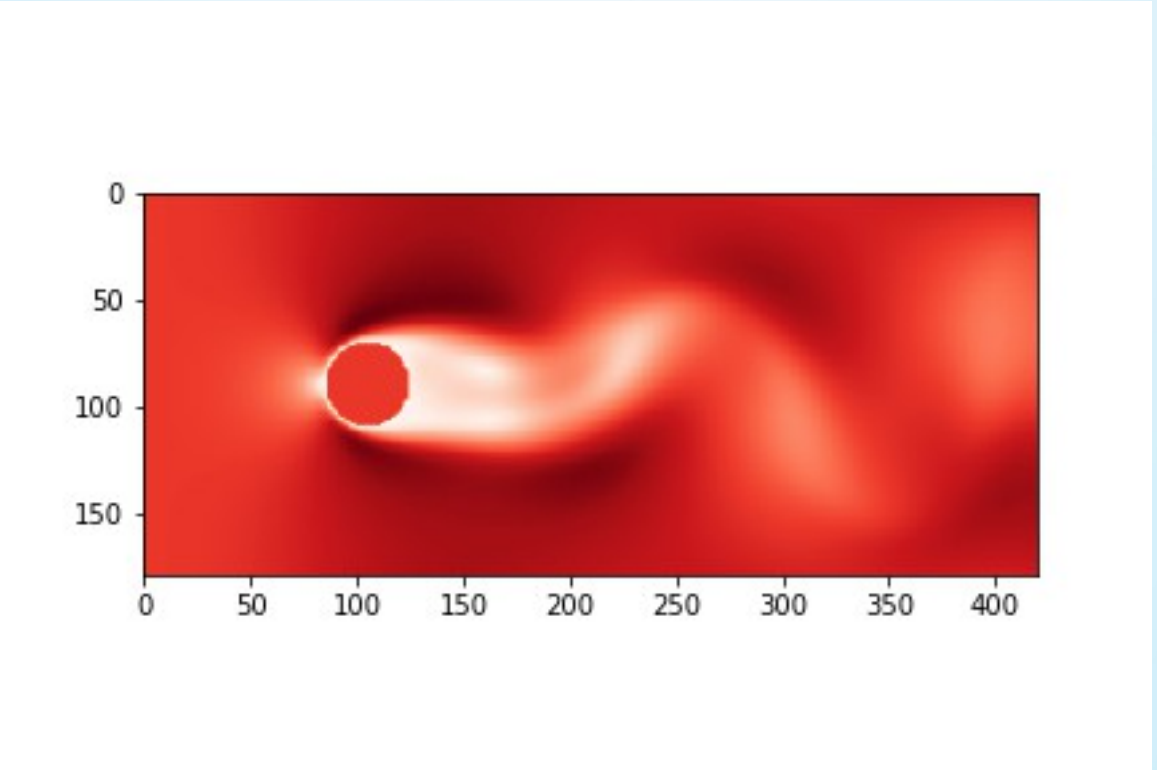
Stencil Operations		Standard	Numba	Numba Parallel	Numba Cuda
8.192.000	**64x64 2000 iter**	9.696 sec	0.007 sec	0.012 sec	0.34 sec
163.840.000	**128x128 10000 iter.**	-	0.133 sec	0.131 sec	1.21 sec
3.276.800.000	**256x256 50000 iter.**	-	2.459 sec	1.401 sec	5.56 sec
65.536.000.000	**512x512 250000 iter.**	-	50.343 sec	19.872 sec	27.30 sec



Examples

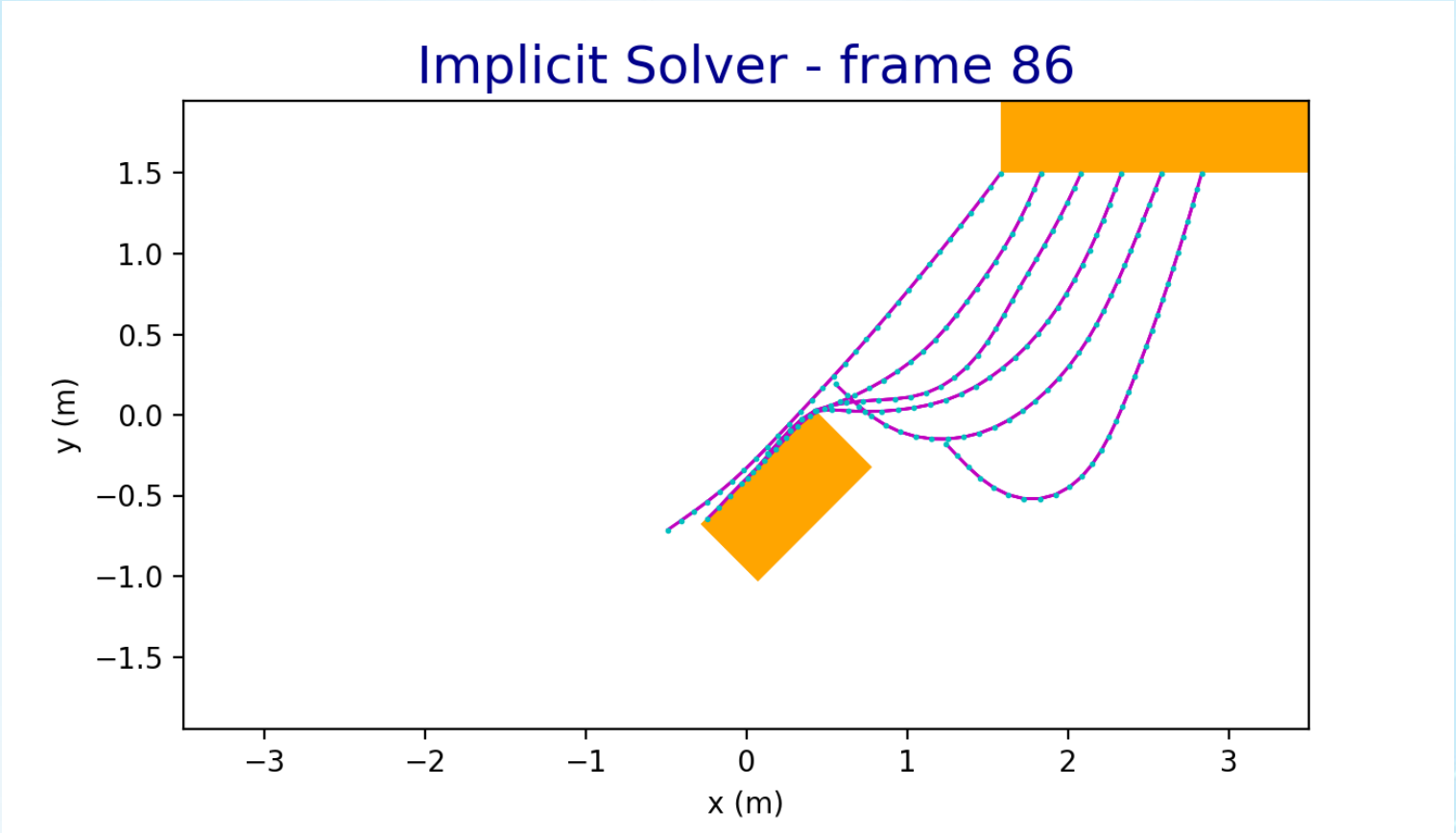
2D – Fluid Simulation

Lattice Boltzmann



2D – Elastic Deformation

FEM - Implicit Solver



Conclusion

- Fail Fast – Development is an iterative process
- Focus more on problem-solving and developing interesting programs

Next Step

- 2D => 3D
- More complex examples
- Debugging / Profiling / Testing

?