





# IDEFIX: USING MULTIFLUIDS AND PASSIVE TRACERS

<https://idefix.readthedocs.io/latest/modules/dust.html>



# MULTIFLUIDS

## *Introduction & motivation*

**AIM:** study the evolution of multiple dust species in a gaseous environment.

# MULTIFLUIDS

## *Introduction & motivation*

**AIM:** study the evolution of multiple dust species in a gaseous environment.

**APPROACHES:** Lagrangian particles or pressureless fluids.

# MULTIFLUIDS

## *Introduction & motivation*

**AIM:** study the evolution of multiple dust species in a gaseous environment.

**APPROACHES:** Lagrangian particles or pressureless fluids.

**METHOD:** solve the continuity and momentum equations, the same way as the gas.

$$\frac{\partial \rho_i}{\partial t} + \vec{\nabla} \cdot \rho_i \vec{v}_i = 0$$

$$\frac{\partial(\rho_i \vec{v}_i)}{\partial t} + \vec{\nabla} \cdot \rho_i \vec{v}_i \otimes \vec{v}_i = -\rho_i \vec{\nabla} \Psi - \vec{\nabla} P$$

# MULTIFLUIDS

## *Introduction & motivation*

**AIM:** study the evolution of multiple dust species in a gaseous environment.

**APPROACHES:** Lagrangian particles or pressureless fluids.

**METHOD:** solve the continuity and momentum equations, the same way as the gas.

**BUT:** the fluid is **pressureless** and subject to an **aerodynamic drag**.

$$\frac{\partial \rho_i}{\partial t} + \vec{\nabla} \cdot \rho_i \vec{v}_i = 0$$

$$\frac{\partial(\rho_i \vec{v}_i)}{\partial t} + \vec{\nabla} \cdot \rho_i \vec{v}_i \otimes \vec{v}_i = -\rho_i \vec{\nabla} \Psi - \cancel{\vec{\nabla} P} + \underbrace{\gamma_i \rho_i \rho (\vec{v} - \vec{v}_i)}_{\text{Aerodynamic drag force}}$$

# MULTIFLUIDS

## *Introduction & motivation*

**AIM:** study the evolution of multiple dust species in a gaseous environment.

**APPROACHES:** Lagrangian particles or pressureless fluids.

**METHOD:** solve the continuity and momentum equations, the same way as the gas.

**BUT:** the fluid is **pressureless** and subject to an **aerodynamic drag**.

**APPLICATION:** gas/dust interactions in a circumstellar disk, varying the dust size.

$$\frac{\partial \rho_i}{\partial t} + \vec{\nabla} \cdot \rho_i \vec{v}_i = 0$$

$$\frac{\partial(\rho_i \vec{v}_i)}{\partial t} + \vec{\nabla} \cdot \rho_i \vec{v}_i \otimes \vec{v}_i = -\rho_i \vec{\nabla} \Psi - \cancel{\vec{\nabla} P} + \underbrace{\gamma_i \rho_i \rho (\vec{v} - \vec{v}_i)}_{\text{Aerodynamic drag force}}$$

# MULTIFLUIDS

## *Introduction & motivation*

**AIM:** study the evolution of multiple dust species in a gaseous environment.

**APPROACHES:** Lagrangian particles or pressureless fluids.

**METHOD:** solve the continuity and momentum equations, the same way as the gas.

**BUT:** the fluid is **pressureless** and subject to an **aerodynamic drag**.

**APPLICATION:** gas/dust interactions in a circumstellar disk, varying the dust size.

$$\frac{\partial \rho_i}{\partial t} + \vec{\nabla} \cdot \rho_i \vec{v}_i = 0$$

$$\frac{\partial(\rho_i \vec{v}_i)}{\partial t} + \vec{\nabla} \cdot \rho_i \vec{v}_i \otimes \vec{v}_i = -\rho_i \vec{\nabla} \Psi - \cancel{\vec{\nabla} P} + \underbrace{\gamma_i \rho_i \rho (\vec{v} - \vec{v}_i)}_{\text{Aerodynamic drag force}}$$

**NOTE:** possible to test the non-ideal MHD framework with a bifluid approach

- one fluid of ions/electrons globally neutral (MHD fluid)
- one fluid of neutral particles (purely hydro fluid).



## MULTIFLUIDS

### *Implementation in Idefix*

We make use of the template class `Fluid<Phys>`, described by the template parameter `Phys`. `Phys` is `DefaultPhysics` for a gas fluid and `DustPhysics` for a dust fluid.

# MULTIFLUIDS

## *Implementation in Idefix*

We make use of the template class `Fluid<Phys>`, described by the template parameter `Phys`. `Phys` is `DefaultPhysics` for a gas fluid and `DustPhysics` for a dust fluid.

```
struct DustPhysics {
    static constexpr bool dust{true};
    static constexpr bool pressure{false};
    static constexpr bool isothermal{false};
    static constexpr bool eos{false};

    static constexpr bool mhd{false};
    static constexpr int nvar{1+COMPONENTS};

    // prefix
    static constexpr std::string_view prefix = "Dust";
};
```

# MULTIFLUIDS

## *Implementation in Idefix*

We make use of the template class `Fluid<Phys>`, described by the template parameter `Phys`. `Phys` is `DefaultPhysics` for a gas fluid and `DustPhysics` for a dust fluid.

```
struct DustPhysics {
    static constexpr bool dust{true};
    static constexpr bool pressure{false};
    static constexpr bool isothermal{false};
    static constexpr bool eos{false};

    static constexpr bool mhd{false};
    static constexpr int nvar{1+COMPONENTS};

    // prefix
    static constexpr std::string_view prefix = "Dust";
};
```

Note that *Idefix* defines an alias for the default fluid which is often found in the provided examples:

```
using Hydro = Fluid<DefaultPhysics>;
```

## PASSIVE TRACERS

### *Implementation in Idefix*

A passive tracer/scalar allows us to trace how much material of the attached fluid is advected during the simulation, given an initial condition. Idefix can follow such passive tracer by solving the incompressible continuity equation for the scalar  $\mathcal{T}$ , with  $\vec{v}$  being the velocity of the corresponding fluid.

$$\text{Advection equation } \frac{\partial \mathcal{T}}{\partial t} + \vec{v} \cdot \vec{\nabla} \mathcal{T} = 0$$

Idefix supports an arbitrary number of tracers per fluid. For example, you can divide radially your domain into N tracers for the gas fluid and check how the tracers are advected during your simulation

## EXAMPLE OF SETUP

*Model: Planet+Dust+Tracers*

**SETUP:** similar to `$IDEFIX_DIR/test/Dust/FargoPlanet`.

## EXAMPLE OF SETUP

*Model: Planet+Dust+Tracers*

**SETUP:** similar to \$IDEFIX\_DIR/test/Dust/FargoPlanet.

**GEOMETRY:** polar coordinates ( $x_1 = R$ ,  $x_2 = \phi$ )

## EXAMPLE OF SETUP

*Model: Planet+Dust+Tracers*

**SETUP:** similar to \$IDEFIX\_DIR/test/Dust/FargoPlanet.

**GEOMETRY:** polar coordinates ( $x_1 = R$ ,  $x_2 = \phi$ )

**SYSTEM:**

- 2D Keplerian thin disk:  $\Sigma_g \propto R^{-1}$ ,  $h_0 = 0.05$

## EXAMPLE OF SETUP

*Model: Planet+Dust+Tracers*

**SETUP:** similar to \$IDEFIX\_DIR/test/Dust/FargoPlanet.

**GEOMETRY:** polar coordinates ( $x_1 = R$ ,  $x_2 = \phi$ )

**SYSTEM:**

- 2D Keplerian thin disk:  $\Sigma_g \propto R^{-1}$ ,  $h_0 = 0.05$
- locally isothermal e.o.s.: fixed  $c_s = h_0 v_K$



## EXAMPLE OF SETUP

*Model: Planet+Dust+Tracers*

**SETUP:** similar to \$IDEFIX\_DIR/test/Dust/FargoPlanet.

**GEOMETRY:** polar coordinates ( $x_1 = R$ ,  $x_2 = \phi$ )

**SYSTEM:**

- 2D Keplerian thin disk:  $\Sigma_g \propto R^{-1}$ ,  $h_0 = 0.05$
- locally isothermal e.o.s.: fixed  $c_s = h_0 v_K$
- 1 Jupiter-mass planet in a fixed circular orbit:  $q_p = M_p/M_\star = 10^{-3}$ ,  $R_p = 1$

## EXAMPLE OF SETUP

*Model: Planet+Dust+Tracers*

**SETUP:** similar to \$IDEFIX\_DIR/test/Dust/FargoPlanet.

**GEOMETRY:** polar coordinates ( $x_1 = R$ ,  $x_2 = \phi$ )

**SYSTEM:**

- 2D Keplerian thin disk:  $\Sigma_g \propto R^{-1}$ ,  $h_0 = 0.05$
- locally isothermal e.o.s.: fixed  $c_s = h_0 v_K$
- 1 Jupiter-mass planet in a fixed circular orbit:  $q_p = M_p/M_\star = 10^{-3}$ ,  $R_p = 1$
- 1 dust fluid:  $\tau_{\text{stop}} = 1$  (see Thomas' section),  $\Sigma_d/\Sigma_g = 10^{-2}$

## EXAMPLE OF SETUP

*Model: Planet+Dust+Tracers*

**SETUP:** similar to \$IDEFIX\_DIR/test/Dust/FargoPlanet.

**GEOMETRY:** polar coordinates ( $x_1 = R$ ,  $x_2 = \phi$ )

**SYSTEM:**

- 2D Keplerian thin disk:  $\Sigma_g \propto R^{-1}$ ,  $h_0 = 0.05$
- locally isothermal e.o.s.: fixed  $c_s = h_0 v_K$
- 1 Jupiter-mass planet in a fixed circular orbit:  $q_p = M_p/M_\star = 10^{-3}$ ,  $R_p = 1$
- 1 dust fluid:  $\tau_{\text{stop}} = 1$  (see Thomas' section),  $\Sigma_d/\Sigma_g = 10^{-2}$
- 2 gas tracers

## EXAMPLE OF SETUP

*Model: Planet+Dust+Tracers*

**SETUP:** similar to \$IDEFIX\_DIR/test/Dust/FargoPlanet.

**GEOMETRY:** polar coordinates ( $x_1 = R$ ,  $x_2 = \phi$ )

**SYSTEM:**

- 2D Keplerian thin disk:  $\Sigma_g \propto R^{-1}$ ,  $h_0 = 0.05$
- locally isothermal e.o.s.: fixed  $c_s = h_0 v_K$
- 1 Jupiter-mass planet in a fixed circular orbit:  $q_p = M_p/M_\star = 10^{-3}$ ,  $R_p = 1$
- 1 dust fluid:  $\tau_{\text{stop}} = 1$  (see Thomas' section),  $\Sigma_d/\Sigma_g = 10^{-2}$
- 2 gas tracers
- 2 dust tracers

# EXAMPLE OF SETUP

*Using Idefix*

*idefix.ini*



# EXAMPLE OF SETUP

*Using Idefix*

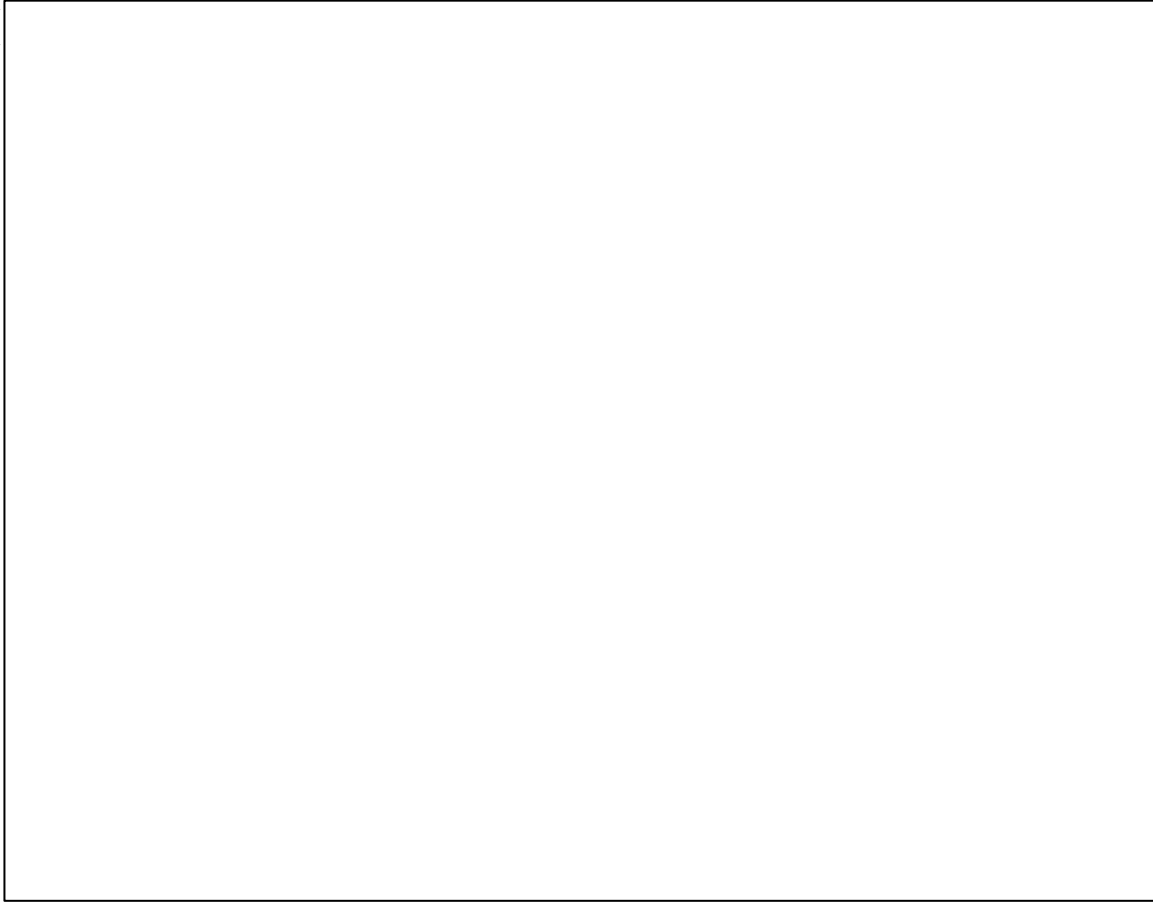
definitions.hpp

idefix.ini

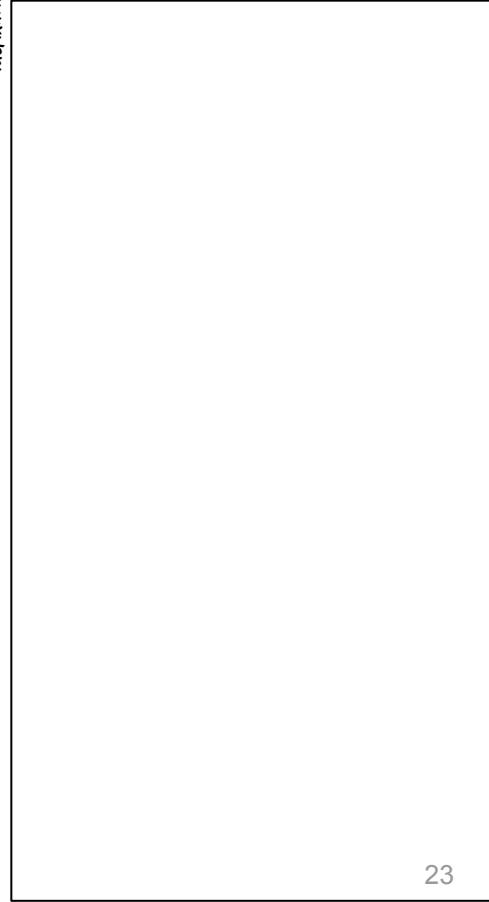
# EXAMPLE OF SETUP

*Using Idefix*

*setup.cpp*



*idefix.ini*



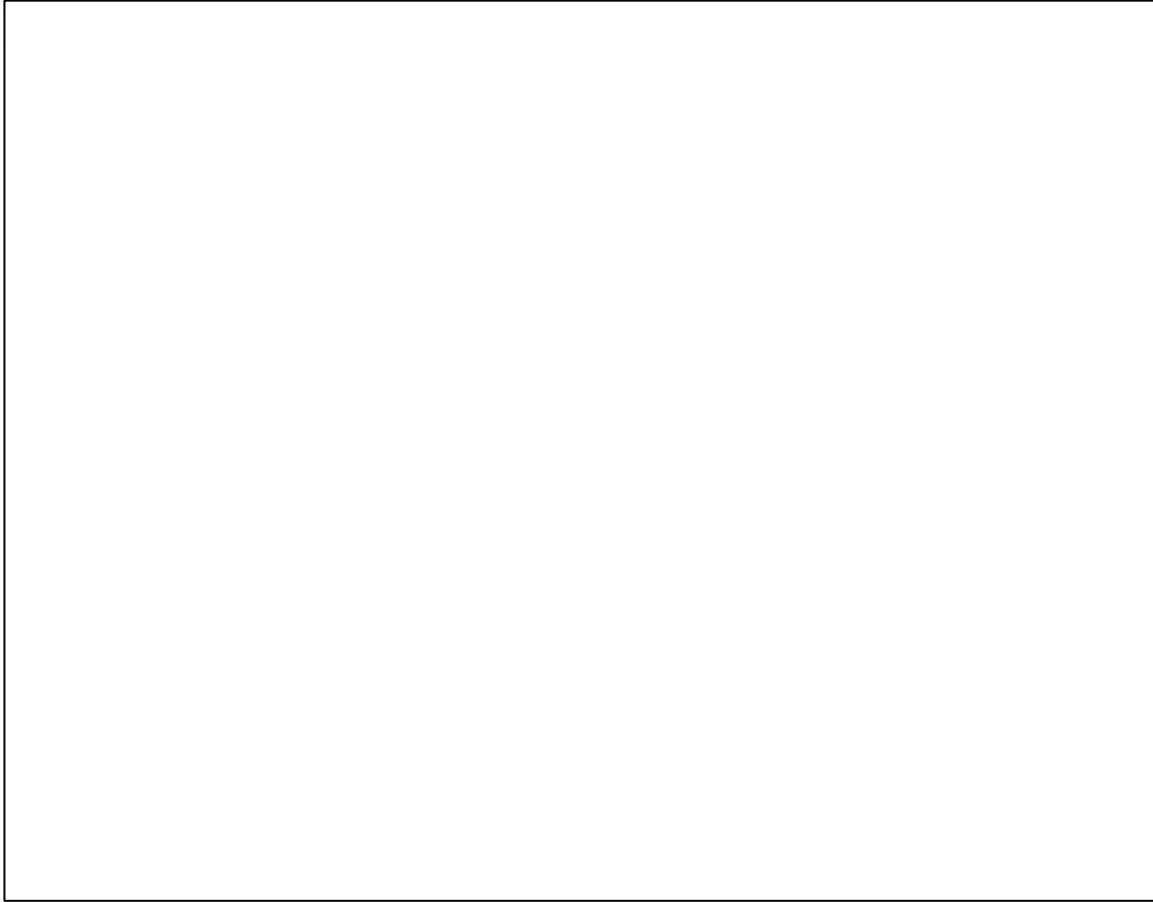
*definitions.hpp*



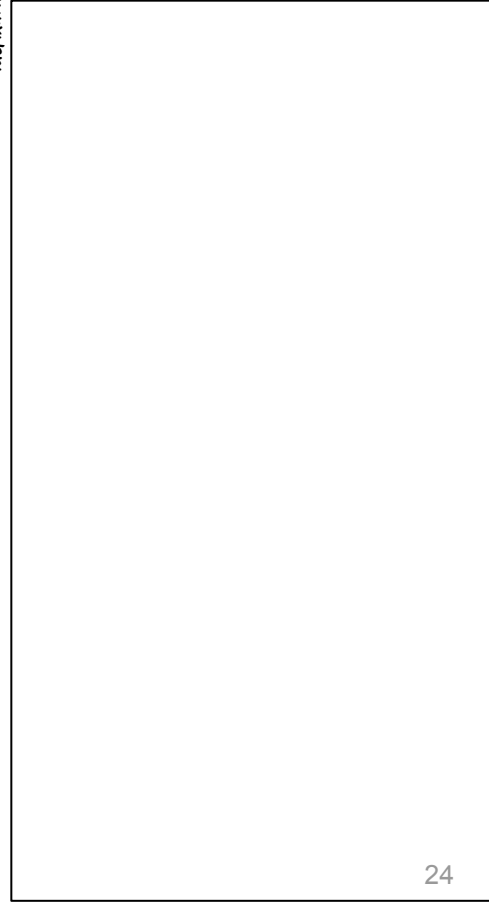
# EXAMPLE OF SETUP

*Using Idefix*

*setup.cpp*



*idefix.ini*



*definitions.hpp*

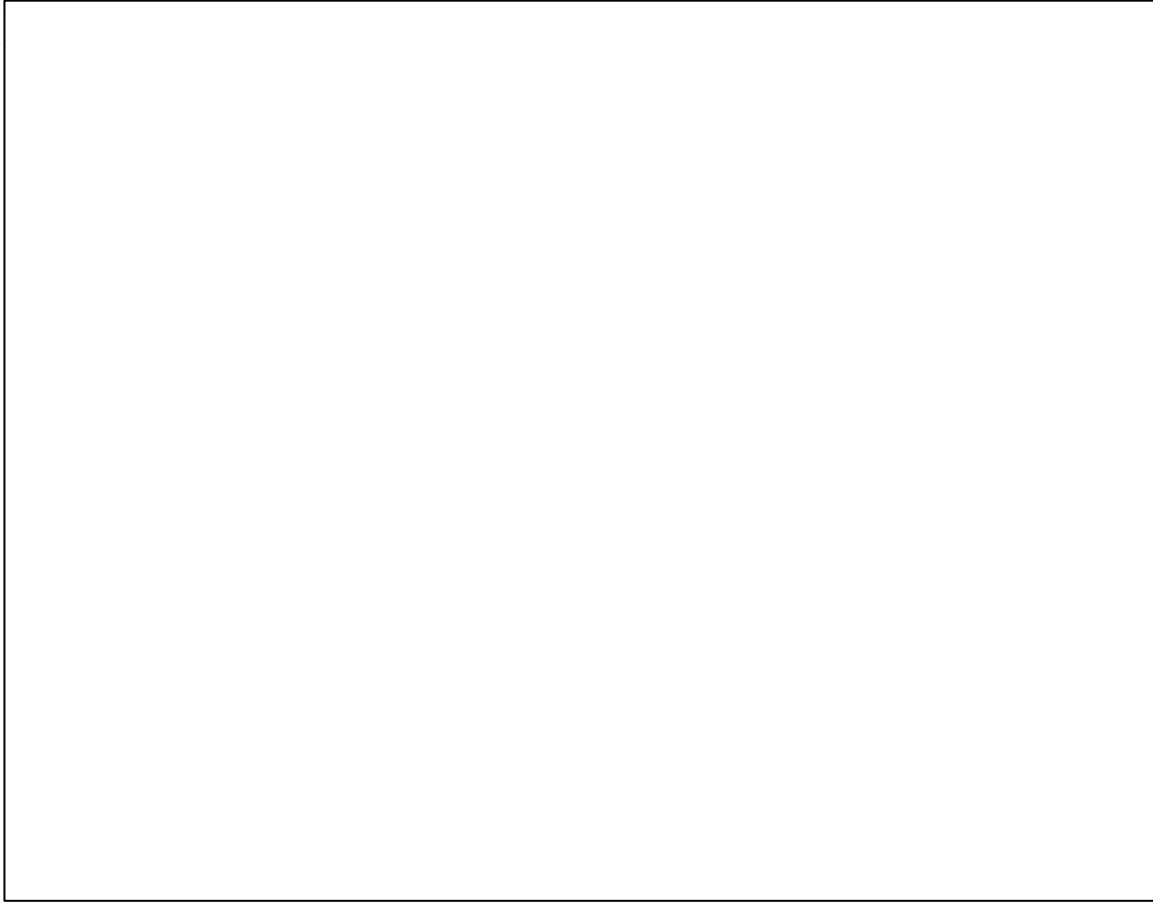
```
1 #define COMPONENTS 2
2 #define DIMENSIONS 2
3
4 #define ISOTHERMAL
5
6 #define GEOMETRY POLAR
```



# EXAMPLE OF SETUP

*Using Idefix*

setup.cpp



definitions.hpp

```

1 #define COMPONENTS 2
2 #define DIMENSIONS 2
3
4 #define ISOTHERMAL
5
6 #define GEOMETRY POLAR

```

idefix.ini

```

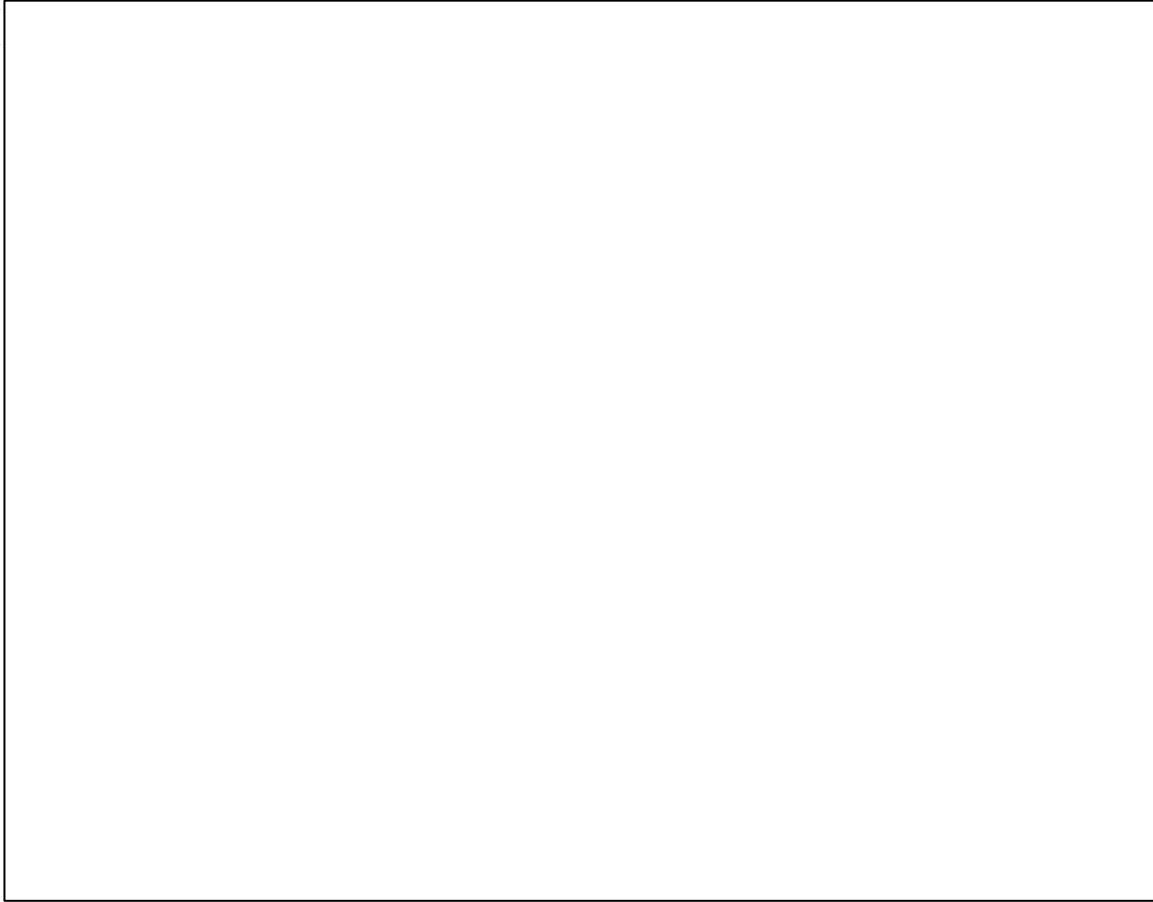
1 [Grid]
2 X1-grid 1 0.4 128 l 2.5
3 X2-grid 1 0.0 256 u 6.283185307179586
4 X3-grid 1 -1 1 u 1
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49

```

# EXAMPLE OF SETUP

*Using Idefix*

setup.cpp



definitions.hpp

```

1 #define COMPONENTS 2
2 #define DIMENSIONS 2
3
4 #define ISOTHERMAL
5
6 #define GEOMETRY POLAR

```

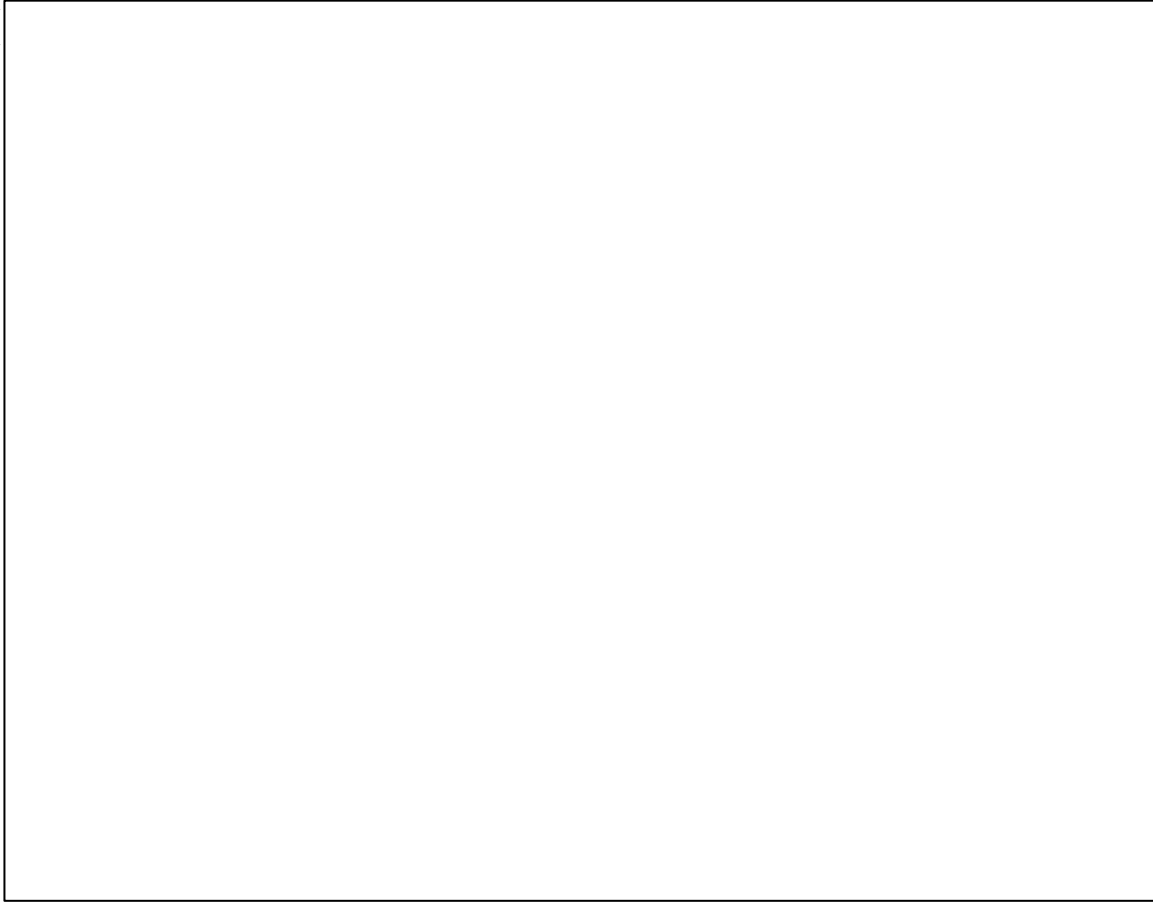
idefix.ini

```

1 [Grid]
2 X1-grid 1 0.4 128 l 2.5
3 X2-grid 1 0.0 256 u 6.283185307179586
4 X3-grid 1 -1 1 u 1
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23 [Gravity]
24 potential central planet
25 Mcentral 1.0
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49

```

## EXAMPLE OF SETUP

*Using Idefix**setup.cpp**definitions.hpp*

```

1 #define COMPONENTS 2
2 #define DIMENSIONS 2
3
4 #define ISOTHERMAL
5
6 #define GEOMETRY POLAR

```

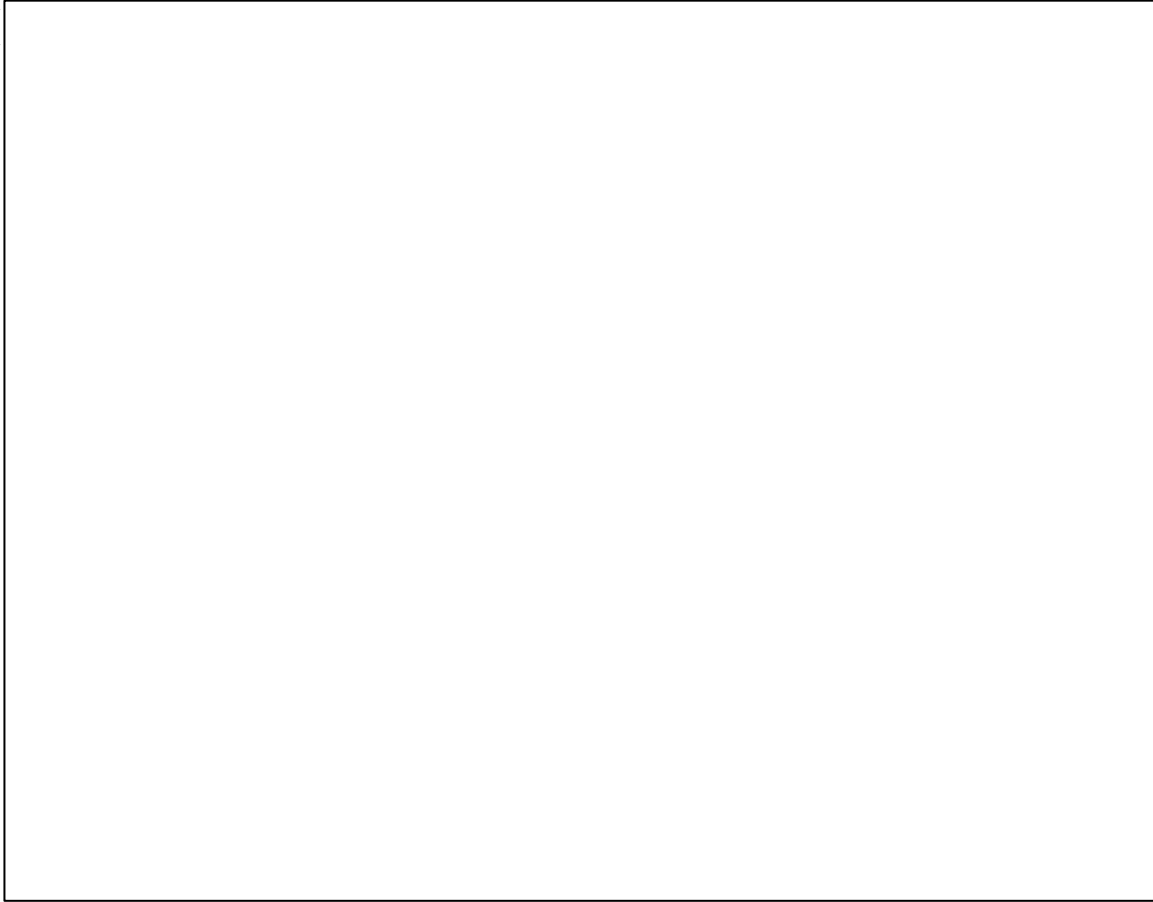
*idefix.ini*

```

1 [Grid]
2 X1-grid 1 0.4 128 l 2.5
3 X2-grid 1 0.0 256 u 6.283185307179586
4 X3-grid 1 -1 1 u 1
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23 [Gravity]
24 potential central planet
25 Mcentral 1.0
26
27 [Planet]
28 integrator analytical
29 planetToPrimary 1.0e-3
30 initialDistance 1.0
31 feelDisk false
32 feelPlanets false
33 smoothing plummer 0.03 0.0
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49

```

## EXAMPLE OF SETUP

*Using Idefix**setup.cpp**definitions.hpp*

```

1 #define COMPONENTS 2
2 #define DIMENSIONS 2
3
4 #define ISOTHERMAL
5
6 #define GEOMETRY POLAR

```

*idefix.ini*

```

1 [Grid]
2 X1-grid 1 0.4 128 l 2.5
3 X2-grid 1 0.0 256 u 6.283185307179586
4 X3-grid 1 -1 1 u 1
5
6
7
8
9
10
11
12 [Hydro]
13 solver hllc
14 csiso userdef
15
16
17
18
19
20
21
22
23 [Gravity]
24 potential central planet
25 Mcentral 1.0
26
27 [Planet]
28 integrator analytical
29 planetToPrimary 1.0e-3
30 initialDistance 1.0
31 feelDisk false
32 feelPlanets false
33 smoothing plummer 0.03 0.0
34
35
36
37
38
39
40
41
42
43 [Setup]
44 h0 0.05
45
46
47
48
49

```

# EXAMPLE OF SETUP

## Using Idefix

setup.cpp

```

1 #include <algorithm> 73
2 #include "idefix.hpp" 74
3 #include "setup.hpp" 75
4 76
5 namespace SetupVariables { 76
6 real h0; 77
7 } 78
8 79
9 void MySoundSpeed(DataBlock &data, const real t, IdefixArray3D<real> &cs) { 80
10 IdefixArray3D<real> x1 = data.x[IDIR]; 81
11 real h0 = SetupVariables::h0; 82
12 idefix_for("MyCS", 0, data.np_tot[KDIR], 0, data.np_tot[JDIR], 0, data.np_tot[IDIR], 83
13 KOKKOS_LAMBDA (int k, int j, int i) { 84
14     real R = x1(i); 85
15     cs(k,j,1) = h0/sqrt(R); 86
16 }); 87
17 } 88
18 89
19 90
20 91
21 92
22 93
23 94
24 95
25 96
26 97
27 98
28 99
29 100
30 101
31 102
32 103
33 104
34 105
35 106
36 107
37 108
38 109
39 110
40 111
41 112
42 113
43 114
44 115
45 116
46 117
47 118
48 119
49 120
50 121
51 122
52 123
53 124
54 125
55 126
56 127
57 128
58 129
59 130
60 131
61 132
62 133
63 134
64 135
65 136
66 137
67
68
69
70
71

```

definitions.hpp

```

1 #define COMPONENTS 2
2 #define DIMENSIONS 2
3
4 #define ISOTHERMAL
5
6 #define GEOMETRY POLAR

```

idefix.ini

```

1 [Grid]
2 X1-grid 1 0.4 128 l 2.5
3 X2-grid 1 0.0 256 u 6.283185307179586
4 X3-grid 1 -1 1 u 1
5
6
7
8
9
10
11
12 [Hydro]
13 solver hllc
14 csiso userdef
15
16
17
18
19
20
21
22
23 [Gravity]
24 potential central planet
25 Mcentral 1.0
26
27 [Planet]
28 integrator analytical
29 planetToPrimary 1.0e-3
30 initialDistance 1.0
31 feelDisk false
32 feelPlanets false
33 smoothing plummer 0.03 0.0
34
35
36
37
38
39
40
41
42
43 [Setup]
44 h0 0.05
45
46
47
48
49

```

# EXAMPLE OF SETUP

## Using Idefix

<https://idefix.readthedocs.io/latest/index.html>

setup.cpp

```

1 #include <algorithm> 73
2 #include "idefix.hpp" 74
3 #include "setup.hpp" 75
4 76
5 namespace SetupVariables { 76
6 real h0; 77
7 } 78
8 79
9 void MySoundSpeed(DataBlock &data, const real t, IdefixArray3D<real> &cs) { 80
10 IdefixArray3D<real> x1 = data.x[IDIR]; 81
11 real h0 = SetupVariables::h0; 82
12 idefix_for("MyCS", 0, data.np_tot[KDIR], 0, data.np_tot[JDIR], 0, data.np_tot[IDIR], 83
13 KOKKOS_LAMBDA (int k, int j, int i) { 84
14     real R = x1(i); 85
15     cs(k,j,1) = h0/sqrt(R); 86
16 }); 87
17 } 88
18 89
19 90
20 91
21 92
22 93
23 94
24 95
25 96
26 97
27 98
28 99
29 100
30 101
31 102
32 103
33 104
34 105
35 106
36 107
37 108
38 109
39 110
40 111
41 112
42 113
43 114
44 115
45 116
46 117
47 118
48 119
49 120
50 121
51 122
52 123
53 124
54 125
55 126
56 127
57 128
58 129
59 130
60 131
61 132
62 133
63 134
64 135
65 136
66 137
67
68
69
70
71

```

definitions.hpp

```

1 #define COMPONENTS 2
2 #define DIMENSIONS 2
3
4 #define ISOTHERMAL
5
6 #define GEOMETRY POLAR

```

idefix.ini

```

1 [Grid]
2 X1-grid 1 0.4 128 l 2.5
3 X2-grid 1 0.0 256 u 6.283185307179586
4 X3-grid 1 -1 1 u 1
5
6
7
8
9
10
11
12 [Hydro]
13 solver hllc
14 csiso userdef
15
16
17
18
19
20
21
22
23 [Gravity]
24 potential central planet
25 Mcentral 1.0
26
27 [Planet]
28 integrator analytical
29 planetToPrimary 1.0e-3
30 initialDistance 1.0
31 feelDisk false
32 feelPlanets false
33 smoothing plummer 0.03 0.0
34
35
36
37
38
39
40
41
42
43 [Setup]
44 h0 0.05
45
46
47
48
49

```

# EXAMPLE OF SETUP

## Using Idefix

<https://idefix.readthedocs.io/latest/index.html>

setup.cpp

```

1 #include <algorithm>
2 #include "idefix.hpp"
3 #include "setup.hpp"
4
5 namespace SetupVariables {
6 real h0;
7 }
8
9 void MySoundSpeed(DataBlock &data, const real t, IdefixArray3D<real> &cs) {
10   IdefixArray3D<real> x1 = data.x[IDIR];
11   real h0 = SetupVariables::h0;
12   idefix_for("MyCS", 0, data.np_tot[KDIR], 0, data.np_tot[JDIR], 0, data.np_tot[IDIR],
13     KOKKOS_LAMBDA (int k, int j, int i) {
14     real R = x1(i);
15     cs(k,j,1) = h0/sqrt(R);
16   });
17 }
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71

```

```

73 // Default constructor
74 Setup::Setup(Input &input, Grid &grid, DataBlock &data, Output &output)
75 {
76
77
78   data.hydro->EnrollIsoSoundSpeed(&MySoundSpeed);
79   SetupVariables::h0 = input.Get<real>("Setup", "h0", 0);
80 }
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137

```

definitions.hpp

```

1 #define COMPONENTS 2
2 #define DIMENSIONS 2
3
4 #define ISOTHERMAL
5
6 #define GEOMETRY POLAR

```

idefix.in

```

1 [Grid]
2 X1-grid 1 0.4 128 l 2.5
3 X2-grid 1 0.0 256 u 6.283185307179586
4 X3-grid 1 -1 1 u 1
5
6
7
8
9
10
11
12 [Hydro]
13 solver hllc
14 csiso userdef
15
16
17
18
19
20
21
22
23 [Gravity]
24 potential central planet
25 Mcentral 1.0
26
27 [Planet]
28 integrator analytical
29 planetToPrimary 1.0e-3
30 initialDistance 1.0
31 feelDisk false
32 feelPlanets false
33 smoothing plummer 0.03 0.0
34
35
36
37
38
39
40
41
42
43 [Setup]
44 h0 0.05
45
46
47
48
49

```

# EXAMPLE OF SETUP

## Using Idefix

<https://idefix.readthedocs.io/latest/index.html>

setup.cpp

```

1 #include <algorithm>
2 #include "idefix.hpp"
3 #include "setup.hpp"
4
5 namespace SetupVariables {
6 real h0;
7 }
8
9 void MySoundSpeed(DataBlock &data, const real t, IdefixArray3D<real> &cs) {
10 IdefixArray3D<real> x1 = data.x[IDIR];
11 real h0 = SetupVariables::h0;
12 idefix_for("MyCS", 0, data.np_tot[KDIR], 0, data.np_tot[JDIR], 0, data.np_tot[IDIR],
13 KOKKOS_LAMBDA(int k, int j, int i) {
14     real R = x1(i);
15     cs(k,j,i) = h0/sqrt(R);
16 });
17 }
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71

```

```

73 // Default constructor
74 Setup::Setup(Input &input, Grid &grid, DataBlock &data, Output &output)
75 {
76
77
78     data.hydro->EnrollIsoSoundSpeed(&MySoundSpeed);
79     SetupVariables::h0 = input.Get<real>("Setup", "h0", 0);
80 }
81
82 void Setup::InitFlow(DataBlock &data) {
83     DataBlockHost d(data);
84     real h0 = SetupVariables::h0;
85
86     for(int k = 0; k < d.np_tot[KDIR]; k++) {
87         for(int j = 0; j < d.np_tot[JDIR]; j++) {
88             for(int i = 0; i < d.np_tot[IDIR]; i++) {
89                 real R=d.x[IDIR](i);
90                 real phi = d.x[JDIR](j);
91                 real Vk=1.0/sqrt(R);
92                 real cs2=(h0*Vk)*(h0*Vk);
93
94                 // Gas initial conditions
95                 d.Vc(RHO,k,j,i) = 100/R;
96                 d.Vc(VX1,k,j,i) = 0.0;
97                 d.Vc(VX2,k,j,i) = Vk*sqrt(1 - 2.0*h0*h0);
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137 }
138 }
139 d.SyncToDevice();

```

definitions.hpp

```

1 #define COMPONENTS 2
2 #define DIMENSIONS 2
3
4 #define ISOTHERMAL
5
6 #define GEOMETRY POLAR

```

idefix.in

```

1 [Grid]
2 X1-grid 1 0.4 128 l 2.5
3 X2-grid 1 0.0 256 u 6.283185307179586
4 X3-grid 1 -1 1 u 1
5
6
7
8
9
10
11
12 [Hydro]
13 solver hllc
14 csiso userdef
15
16
17
18
19
20
21
22
23 [Gravity]
24 potential central planet
25 Mcentral 1.0
26
27 [Planet]
28 integrator analytical
29 planetToPrimary 1.0e-3
30 initialDistance 1.0
31 feelDisk false
32 feelPlanets false
33 smoothing plummer 0.03 0.0
34
35
36
37
38
39
40
41
42
43 [Setup]
44 h0 0.05
45
46
47
48
49

```



# EXAMPLE OF SETUP

## Using Idefix

<https://idefix.readthedocs.io/latest/index.html>

setup.cpp

```

1 #include <algorithm>
2 #include "idefix.hpp"
3 #include "setup.hpp"
4
5 namespace SetupVariables {
6 real h0;
7 }
8
9 void MySoundSpeed(DataBlock &data, const real t, IdefixArray3D<real> &cs) {
10 IdefixArray3D<real> x1 = data.x[IDIR];
11 real h0 = SetupVariables::h0;
12 idefix_for("MyCS", 0, data.np_tot[KDIR], 0, data.np_tot[JDIR], 0, data.np_tot[IDIR],
13 KOKKOS_LAMBDA(int k, int j, int i) {
14     real R = x1(i);
15     cs(k,j,i) = h0/sqrt(R);
16 });
17 }
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73 // Default constructor
74 Setup::Setup(Input &input, Grid &grid, DataBlock &data, Output &output)
75 {
76
77
78     data.hydro->EnrollIsoSoundSpeed(&MySoundSpeed);
79     SetupVariables::h0 = input.Get<real>("Setup", "h0", 0);
80 }
81
82 void Setup::InitFlow(DataBlock &data) {
83     DataBlockHost d(data);
84     real h0 = SetupVariables::h0;
85
86     for(int k = 0; k < d.np_tot[KDIR]; k++) {
87         for(int j = 0; j < d.np_tot[JDIR]; j++) {
88             for(int i = 0; i < d.np_tot[IDIR]; i++) {
89                 real R=d.x[IDIR](i);
90                 real phi = d.x[JDIR](j);
91                 real Vk=1.0/sqrt(R);
92                 real cs2=(h0*Vk)*(h0*Vk);
93
94                 // Gas initial conditions
95                 d.Vc(RHO,k,j,i) = 100/R;
96                 d.Vc(VX1,k,j,i) = 0.0;
97                 d.Vc(VX2,k,j,i) = Vk*sqrt(1 - 2.0*h0*h0);
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137 }
138 }
139
140 d.SyncToDevice();

```

definitions.hpp

```

1 #define COMPONENTS 2
2 #define DIMENSIONS 2
3
4 #define ISOTHERMAL
5
6 #define GEOMETRY POLAR

```

idefix.in

```

1 [Grid]
2 X1-grid 1 0.4 128 l 2.5
3 X2-grid 1 0.0 256 u 6.283185307179586
4 X3-grid 1 -1 1 u 1
5
6
7
8
9
10
11
12 [Hydro]
13 solver hllc
14 csiso userdef
15
16
17 [Dust]
18 nSpecies 1
19 drag tau 1.0
20
21
22
23 [Gravity]
24 potential central planet
25 Mcentral 1.0
26
27 [Planet]
28 integrator analytical
29 planetToPrimary 1.0e-3
30 initialDistance 1.0
31 feelDisk false
32 feelPlanets false
33 smoothing plummer 0.03 0.0
34
35
36
37
38
39
40
41
42
43 [Setup]
44 h0 0.05
45
46
47
48
49

```

# EXAMPLE OF SETUP

## Using Idefix

<https://idefix.readthedocs.io/latest/index.html>

setup.cpp

```

1 #include <algorithm>
2 #include "idefix.hpp"
3 #include "setup.hpp"
4
5 namespace SetupVariables {
6 real h0;
7 }
8
9 void MySoundSpeed(DataBlock &data, const real t, IdefixArray3D<real> &cs) {
10 IdefixArray3D<real> x1 = data.x[IDIR];
11 real h0 = SetupVariables::h0;
12 idefix_for("MyCS", 0, data.np_tot[KDIR], 0, data.np_tot[JDIR], 0, data.np_tot[IDIR],
13 KOKKOS_LAMBDA(int k, int j, int i) {
14     real R = x1(i);
15     cs(k,j,i) = h0/sqrt(R);
16 });
17 }
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72 // Default constructor
73 Setup::Setup(Input &input, Grid &grid, DataBlock &data, Output &output)
74 {
75
76
77     data.hydro->EnrollIsoSoundSpeed(&MySoundSpeed);
78     SetupVariables::h0 = input.Get<real>("Setup", "h0", 0);
79
80 }
81
82 void Setup::InitFlow(DataBlock &data) {
83     DataBlockHost d(data);
84     real h0 = SetupVariables::h0;
85
86     for(int k = 0; k < d.np_tot[KDIR]; k++) {
87         for(int j = 0; j < d.np_tot[JDIR]; j++) {
88             for(int i = 0; i < d.np_tot[IDIR]; i++) {
89                 real R=d.x[IDIR](i);
90                 real phi = d.x[JDIR](j);
91                 real Vk=1.0/sqrt(R);
92                 real cs2=(h0*Vk)+(h0*Vk);
93
94                 // Gas initial conditions
95                 d.Vc(RHO,k,j,i) = 100/R;
96                 d.Vc(VX1,k,j,i) = 0.0;
97                 d.Vc(VX2,k,j,i) = Vk*sqrt(1 - 2.0*h0*h0);
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137 }
138 }
139 d.SyncToDevice();

```

definitions.hpp

```

1 #define COMPONENTS 2
2 #define DIMENSIONS 2
3
4 #define ISOTHERMAL
5
6 #define GEOMETRY POLAR

```

idefix.in

```

1 [Grid]
2 X1-grid 1 0.4 128 l 2.5
3 X2-grid 1 0.0 256 u 6.283185307179586
4 X3-grid 1 -1 1 u 1
5
6
7
8
9
10
11
12 [Hydro]
13 solver hllc
14 csiso userdef
15
16
17 [Dust] ⚠ drag_feedback = true
18 nSpecies 1 by default
19 drag tau 1.0
20
21 → see Thomas' presentation
22
23 [Gravity]
24 potential central planet
25 Mcentral 1.0
26
27 [Planet]
28 integrator analytical
29 planetToPrimary 1.0e-3
30 initialDistance 1.0
31 feelDisk false
32 feelPlanets false
33 smoothing plummer 0.03 0.0
34
35
36
37
38
39
40
41
42
43 [Setup]
44 h0 0.05
45
46
47
48
49

```

# EXAMPLE OF SETUP

*Using Idefix*

## Dust parameters

<https://idefix.readthedocs.io/latest/modules/dust.html>

The dust module can be enabled adding a block `[Dust]` in your input `.ini` file. The parameters are as follow:

Entry name	Parameter type	Comment
nSpecies	integer	Number of dust species to solve
drag	string, float, ...	<p>The first parameter describe the drag type. Possible values are: <code>gamma</code>, <code>tau</code>, <code>size</code> and <code>userdef</code>.</p> <p>The remaining parameters gives the drag parameter <math>\beta_i</math> for each dust specie. (see below). <i>Idefix</i> expect to have as many drag parameters as there are dust species.</p>
drag_feedback	bool	(optionnal) whether the gas feedback is enabled (default true).

# EXAMPLE OF SETUP

## Using Idefix

<https://idefix.readthedocs.io/latest/index.html>

setup.cpp

```

1 #include <algorithm>
2 #include "idefix.hpp"
3 #include "setup.hpp"
4
5 namespace SetupVariables {
6 real h0;
7 }
8
9 void MySoundSpeed(DataBlock &data, const real t, IdefixArray3D<real> &cs) {
10 IdefixArray3D<real> x1 = data.x[IDIR];
11 real h0 = SetupVariables::h0;
12 idefix_for("MyCS", 0, data.np_tot[KDIR], 0, data.np_tot[JDIR], 0, data.np_tot[IDIR],
13 KOKKOS_LAMBDA(int k, int j, int i) {
14     real R = x1(i);
15     cs(k,j,i) = h0/sqrt(R);
16 });
17 }
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72 // Default constructor
73 Setup::Setup(Input &input, Grid &grid, DataBlock &data, Output &output)
74 {
75
76
77     data.hydro->EnrollIsoSoundSpeed(&MySoundSpeed);
78     SetupVariables::h0 = input.Get<real>("Setup", "h0", 0);
79
80 }
81
82 void Setup::InitFlow(DataBlock &data) {
83     DataBlockHost d(data);
84     real h0 = SetupVariables::h0;
85
86     for(int k = 0; k < d.np_tot[KDIR]; k++) {
87         for(int j = 0; j < d.np_tot[JDIR]; j++) {
88             for(int i = 0; i < d.np_tot[IDIR]; i++) {
89                 real R=d.x[IDIR](i);
90                 real phi = d.x[JDIR](j);
91                 real Vk=1.0/sqrt(R);
92                 real cs2=(h0*Vk)*(h0*Vk);
93
94                 // Gas initial conditions
95                 d.Vc(RHO,k,j,i) = 100/R;
96                 d.Vc(VX1,k,j,i) = 0.0;
97                 d.Vc(VX2,k,j,i) = Vk*sqrt(1 - 2.0*h0*h0);
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137 }
138 }
139 }
140 d.SyncToDevice();

```

definitions.hpp

```

1 #define COMPONENTS 2
2 #define DIMENSIONS 2
3
4 #define ISOTHERMAL
5
6 #define GEOMETRY POLAR

```

idefix.in

```

1 [Grid]
2 X1-grid 1 0.4 128 l 2.5
3 X2-grid 1 0.0 256 u 6.283185307179586
4 X3-grid 1 -1 1 u 1
5
6
7
8
9
10
11
12 [Hydro]
13 solver hllc
14 csiso userdef
15
16
17 [Dust]
18 nSpecies 1
19 drag tau 1.0
20
21
22
23 [Gravity]
24 potential central planet
25 Mcentral 1.0
26
27 [Planet]
28 integrator analytical
29 planetToPrimary 1.0e-3
30 initialDistance 1.0
31 feelDisk false
32 feelPlanets false
33 smoothing plummer 0.03 0.0
34
35
36
37
38
39
40
41
42
43 [Setup]
44 h0 0.05
45
46
47
48
49

```

# EXAMPLE OF SETUP

## Using Idefix

<https://idefix.readthedocs.io/latest/index.html>

setup.cpp

```

1 #include <algorithm>
2 #include "idefix.hpp"
3 #include "setup.hpp"
4
5 namespace SetupVariables {
6 real h0;
7 }
8
9 void MySoundSpeed(DataBlock &data, const real t, IdefixArray3D<real> &cs) {
10 IdefixArray3D<real> x1 = data.x[IDIR];
11 real h0 = SetupVariables::h0;
12 idefix_for("MyCS", 0, data.np_tot[KDIR], 0, data.np_tot[JDIR], 0, data.np_tot[IDIR],
13 KOKKOS_LAMBDA(int k, int j, int i) {
14     real R = x1(i);
15     cs(k,j,i) = h0/sqrt(R);
16 });
17 }
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73 // Default constructor
74 Setup::Setup(Input &input, Grid &grid, DataBlock &data, Output &output)
75 {
76
77
78     data.hydro->EnrollIsoSoundSpeed(&MySoundSpeed);
79     SetupVariables::h0 = input.Get<real>("h0", 0);
80 }
81
82 void Setup::InitFlow(DataBlock &data) {
83     DataBlockHost d(data);
84     real h0 = SetupVariables::h0;
85
86     for(int k = 0; k < d.np_tot[KDIR]; k++) {
87         for(int j = 0; j < d.np_tot[JDIR]; j++) {
88             for(int i = 0; i < d.np_tot[IDIR]; i++) {
89                 real R=d.x[IDIR](i);
90                 real phi = d.x[JDIR](j);
91                 real Vk=1.0/sqrt(R);
92                 real cs2=(h0*Vk)+(h0*Vk);
93
94                 // Gas initial conditions
95                 d.Vc(RHO,k,j,i) = 100/R;
96                 d.Vc(VX1,k,j,i) = 0.0;
97                 d.Vc(VX2,k,j,i) = Vk*sqrt(1 - 2.0*h0*h0);
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137 }
138 }
139 }
140 d.SyncToDevice();

```

definitions.hpp

```

1 #define COMPONENTS 2
2 #define DIMENSIONS 2
3
4 #define ISOTHERMAL
5
6 #define GEOMETRY POLAR

```

idefix.in

```

1 [Grid]
2 X1-grid 1 0.4 128 l 2.5
3 X2-grid 1 0.0 256 u 6.283185307179586
4 X3-grid 1 -1 1 u 1
5
6
7
8
9
10
11
12 [Hydro]
13 solver hllc
14 csiso userdef
15
16
17 [Dust]
18 nSpecies 1
19 drag tau 1.0
20
21
22
23 [Gravity]
24 potential central planet
25 Mcentral 1.0
26
27 [Planet]
28 integrator analytical
29 planetToPrimary 1.0e-3
30 initialDistance 1.0
31 feelDisk false
32 feelPlanets false
33 smoothing plummer 0.03 0.0
34
35
36
37
38
39
40
41
42
43 [Setup]
44 h0 0.05
45
46
47
48
49

```

# EXAMPLE OF SETUP

## Using Idefix

<https://idefix.readthedocs.io/latest/index.html>

setup.cpp

```

1 #include <algorithm>
2 #include "idefix.hpp"
3 #include "setup.hpp"
4
5 namespace SetupVariables {
6 real h0;
7 }
8
9 void MySoundSpeed(DataBlock &data, const real t, IdefixArray3D<real> &cs) {
10 IdefixArray3D<real> x1 = data.x[IDIR];
11 real h0 = SetupVariables::h0;
12 idefix_for("MyCS", 0, data.np_tot[KDIR], 0, data.np_tot[JDIR], 0, data.np_tot[IDIR],
13 KOKKOS_LAMBDA(int k, int j, int i) {
14     real R = x1(i);
15     cs(k,j,i) = h0/sqrt(R);
16 });
17 }
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71

```

```

73 // Default constructor
74 Setup::Setup(Input &input, Grid &grid, DataBlock &data, Output &output)
75 {
76
77
78     data.hydro->EnrollIsoSoundSpeed(&MySoundSpeed);
79     SetupVariables::h0 = input.Get<real>("Setup", "h0", 0);
80 }
81
82 void Setup::InitFlow(DataBlock &data) {
83     DataBlockHost d(data);
84     real h0 = SetupVariables::h0;
85
86     for(int k = 0; k < d.np_tot[KDIR]; k++) {
87         for(int j = 0; j < d.np_tot[JDIR]; j++) {
88             for(int i = 0; i < d.np_tot[IDIR]; i++) {
89                 real R=d.x[IDIR](i);
90                 real phi = d.x[JDIR](j);
91                 real Vk=1.0/sqrt(R);
92                 real cs2=(h0*Vk)+(h0*Vk);
93
94                 // Gas initial conditions
95                 d.Vc(RHO,k,j,i) = 100/R;
96                 d.Vc(VX1,k,j,i) = 0.0;
97                 d.Vc(VX2,k,j,i) = Vk*sqrt(1 - 2.0*h0*h0);
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137 }
138 }
139
140 d.SyncToDevice();

```

definitions.hpp

```

1 #define COMPONENTS 2
2 #define DIMENSIONS 2
3
4 #define ISOTHERMAL
5
6 #define GEOMETRY POLAR

```

idefix.in

```

1 [Grid]
2 X1-grid 1 0.4 128 l 2.5
3 X2-grid 1 0.0 256 u 6.283185307179586
4 X3-grid 1 -1 1 u 1
5
6
7
8
9
10
11
12 [Hydro]
13 solver hllc
14 csiso userdef
15
16
17 [Dust]
18 nSpecies 1
19 drag tau 1.0
20
21
22
23 [Gravity]
24 potential central planet
25 Mcentral 1.0
26
27 [Planet]
28 integrator analytical
29 planetToPrimary 1.0e-3
30 initialDistance 1.0
31 feelDisk false
32 feelPlanets false
33 smoothing plummer 0.03 0.0
34
35 [Boundary]
36 X1-beg userdef
37 X1-end userdef
38 X2-beg periodic
39 X2-end periodic
40 X3-beg outflow
41 X3-end outflow
42
43 [Setup]
44 h0 0.05
45
46
47
48
49

```

# EXAMPLE OF SETUP

## Using Idefix

<https://idefix.readthedocs.io/latest/index.html>

setup.cpp

```

1 #include <algorithm>
2 #include "idefix.hpp"
3 #include "setup.hpp"
4
5 namespace SetupVariables {
6 real h0;
7 }
8
9 void MySoundSpeed(DataBlock &data, const real t, IdefixArray3D<real> &cs) {
10 IdefixArray1D<real> x1 = data.x[IDIR];
11 real h0 = SetupVariables::h0;
12 idefix_for("MyCS", 0, data.np_tot[KDIR], 0, data.np_tot[JDIR], 0, data.np_tot[IDIR],
13 KOKKOS_LAMBDA (int k, int j, int i) {
14     real R = x1(i);
15     cs(k,j,i) = h0/sqrt(R);
16 });
17 }
18
19 // Gas User-defined boundaries
20 void BCGas(Fluid<DefaultPhysics> *hydro, int dir, BoundarySide side, real t) {
21 idfx::pushRegion("UserDefinedBoundary");
22 if(dir==IDIR) {
23     IdefixArray4D<real> Vc = hydro->Vc;
24     IdefixArray1D<real> x1 = hydro->data->x[IDIR];
25     int iref;
26     if(side == left) {
27         iref = hydro->data->beg[IDIR];
28     } else if (side==right) {
29         iref = hydro->data->end[IDIR]-1;
30     }
31     hydro->boundary->BoundaryFor("UserDefBoundary_X1", dir, side,
32 KOKKOS_LAMBDA (int k, int j, int i) {
33         real R=x1(i);
34         real Vk = 1.0/sqrt(R);
35
36         Vc(RHO,k,j,i) = Vc(RHO,k,j,iref);
37         Vc(VX1,k,j,i) = Vc(VX1,k,j,iref);
38         Vc(VX2,k,j,i) = Vk;
39
40     });
41 }
42 }
43 idfx::popRegion();
44 }
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71

```

```

73 // Default constructor
74 Setup::Setup(Input &input, Grid &grid, DataBlock &data, Output &output)
75 {
76
77     data.hydro->EnrollIsoSoundSpeed(&MySoundSpeed);
78     SetupVariables::h0 = input.Get<real>("Setup", "h0", 0);
79 }
80
81 void Setup::InitFlow(DataBlock &data) {
82     DataBlockHost d(data);
83     real h0 = SetupVariables::h0;
84
85     for(int k = 0; k < d.np_tot[KDIR]; k++) {
86         for(int j = 0; j < d.np_tot[JDIR]; j++) {
87             for(int i = 0; i < d.np_tot[IDIR]; i++) {
88                 real R=d.x[IDIR](i);
89                 real phi = d.x[JDIR](j);
90                 real Vk=1.0/sqrt(R);
91                 real cs2=(h0*Vk)*(h0*Vk);
92
93                 // Gas initial conditions
94                 d.Vc(RHO,k,j,i) = 100/R;
95                 d.Vc(VX1,k,j,i) = 0.0;
96                 d.Vc(VX2,k,j,i) = Vk*sqrt(1 - 2.0*h0*h0);
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137 }
138 }
139 d.SyncToDevice();

```

definitions.hpp

```

1 #define COMPONENTS 2
2 #define DIMENSIONS 2
3
4 #define ISOTHERMAL
5
6 #define GEOMETRY POLAR

```

idefix.in

```

1 [Grid]
2 X1-grid 1 0.4 128 l 2.5
3 X2-grid 1 0.0 256 u 6.283185307179586
4 X3-grid 1 -1 1 u 1
5
6
7
8
9
10
11
12 [Hydro]
13 solver hllc
14 csiso userdef
15
16
17 [Dust]
18 nSpecies 1
19 drag tau 1.0
20
21
22
23 [Gravity]
24 potential central planet
25 Mcentral 1.0
26
27 [Planet]
28 integrator analytical
29 planetToPrimary 1.0e-3
30 initialDistance 1.0
31 feelDisk false
32 feelPlanets false
33 smoothing plummer 0.03 0.0
34
35 [Boundary]
36 X1-beg userdef
37 X1-end userdef
38 X2-beg periodic
39 X2-end periodic
40 X3-beg outflow
41 X3-end outflow
42
43 [Setup]
44 h0 0.05
45
46
47
48
49

```

# EXAMPLE OF SETUP

## Using Idefix

<https://idefix.readthedocs.io/latest/index.html>

setup.cpp

```

1 #include <algorithm>
2 #include "idefix.hpp"
3 #include "setup.hpp"
4
5 namespace SetupVariables {
6 real h0;
7 }
8
9 void MySoundSpeed(DataBlock &data, const real t, IdefixArray3D<real> &cs) {
10 IdefixArray1D<real> x1 = data.x[IDIR];
11 real h0 = SetupVariables::h0;
12 idefix_for("MyCS", 0, data.np_tot[KDIR], 0, data.np_tot[JDIR], 0, data.np_tot[IDIR],
13 KOKKOS_LAMBDA (int k, int j, int i) {
14     real R = x1(i);
15     cs(k,j,i) = h0/sqrt(R);
16 });
17 }
18
19 // Gas User-defined boundaries
20 void BCGas(Fluid<DefaultPhysics> *hydro, int dir, BoundarySide side, real t) {
21 idfx::pushRegion("UserDefinedBoundary");
22 if(dir==IDIR) {
23     IdefixArray4D<real> Vc = hydro->Vc;
24     IdefixArray1D<real> x1 = hydro->data->x[IDIR];
25     int iref;
26     if(side == left) {
27         iref = hydro->data->beg[IDIR];
28     } else if (side==right) {
29         iref = hydro->data->end[IDIR]-1;
30     }
31     hydro->boundary->BoundaryFor("UserDefBoundary_X1", dir, side,
32 KOKKOS_LAMBDA (int k, int j, int i) {
33         real R=x1(i);
34         real Vk = 1.0/sqrt(R);
35
36         Vc(RHO,k,j,i) = Vc(RHO,k,j,iref);
37         Vc(VX1,k,j,i) = Vc(VX1,k,j,iref);
38         Vc(VX2,k,j,i) = Vk;
39
40     });
41 }
42 idfx::popRegion();
43 }
44
45
46 // Dust initial conditions
47 d.dustVc[0](RHO,k,j,i) = d.Vc(RHO,k,j,i) / 100.0;
48 d.dustVc[0](VX1,k,j,i) = 0.0;
49 d.dustVc[0](VX2,k,j,i) = Vk;
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71

```

```

73 // Default constructor
74 Setup::Setup(Input &input, Grid &grid, DataBlock &data, Output &output)
75 {
76     data.hydro->EnrollUserDefBoundary(&BCGas);
77     data.hydro->EnrollIsoSoundSpeed(&MySoundSpeed);
78     SetupVariables::h0 = input.Get<real>("Setup", "h0", 0);
79 }
80
81 void Setup::InitFlow(DataBlock &data) {
82     DataBlockHost d(data);
83     real h0 = SetupVariables::h0;
84
85     for(int k = 0; k < d.np_tot[KDIR]; k++) {
86         for(int j = 0; j < d.np_tot[JDIR]; j++) {
87             for(int i = 0; i < d.np_tot[IDIR]; i++) {
88                 real R=d.x[IDIR](i);
89                 real phi = d.x[JDIR](j);
90                 real Vk=1.0/sqrt(R);
91                 real cs2=(h0*Vk)*(h0*Vk);
92
93                 // Gas initial conditions
94                 d.Vc(RHO,k,j,i) = 100/R;
95                 d.Vc(VX1,k,j,i) = 0.0;
96                 d.Vc(VX2,k,j,i) = Vk*sqrt(1 - 2.0*h0*h0);
97
98             }
99
100         }
101     }
102
103     }
104
105     }
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137 }
d.SyncToDevice();

```

definitions.hpp

```

1 #define COMPONENTS 2
2 #define DIMENSIONS 2
3
4 #define ISOTHERMAL
5
6 #define GEOMETRY POLAR

```

idefix.in

```

1 [Grid]
2 X1-grid 1 0.4 128 l 2.5
3 X2-grid 1 0.0 256 u 6.283185307179586
4 X3-grid 1 -1 1 u 1
5
6
7
8
9
10
11
12 [Hydro]
13 solver hllc
14 csiso userdef
15
16
17 [Dust]
18 nSpecies 1
19 drag tau 1.0
20
21
22
23 [Gravity]
24 potential central planet
25 Mcentral 1.0
26
27 [Planet]
28 integrator analytical
29 planetToPrimary 1.0e-3
30 initialDistance 1.0
31 feelDisk false
32 feelPlanets false
33 smoothing plummer 0.03 0.0
34
35 [Boundary]
36 X1-beg userdef
37 X1-end userdef
38 X2-beg periodic
39 X2-end periodic
40 X3-beg outflow
41 X3-end outflow
42
43 [Setup]
44 h0 0.05
45
46
47
48
49

```



# EXAMPLE OF SETUP

## Using Idefix

<https://idefix.readthedocs.io/latest/index.html>

setup.cpp

```

1 #include <algorithm>
2 #include "idefix.hpp"
3 #include "setup.hpp"
4
5 namespace SetupVariables {
6 real h0;
7 }
8
9 void MySoundSpeed(DataBlock &data, const real t, IdefixArray3D<real> &cs) {
10 IdefixArray1D<real> x1 = data.x[IDIR];
11 real h0 = SetupVariables::h0;
12 idfix_for("MyCS", 0, data.np_tot[KDIR], 0, data.np_tot[JDIR], 0, data.np_tot[IDIR],
13 KOKKOS_LAMBDA (int k, int j, int i) {
14     real R = x1(i);
15     cs(k,j,i) = h0/sqrt(R);
16 });
17 }
18
19 // Gas User-defined boundaries
20 void BCGas(Fluid<DefaultPhysics> *hydro, int dir, BoundarySide side, real t) {
21 idfx::pushRegion("UserDefinedBoundary");
22 if (dir == IDIR) {
23     IdefixArray4D<real> Vc = hydro->Vc;
24     IdefixArray1D<real> x1 = hydro->data->x[IDIR];
25     int iref;
26     if (side == left) {
27         iref = hydro->data->beg[IDIR];
28     } else if (side == right) {
29         iref = hydro->data->end[IDIR]-1;
30     }
31     hydro->boundary->BoundaryFor("UserDefBoundary_X1", dir, side,
32 KOKKOS_LAMBDA (int k, int j, int i) {
33         real R=x1(i);
34         real Vk = 1.0/sqrt(R);
35
36         Vc(RHO, k, j, i) = Vc(RHO, k, j, iref);
37         Vc(VX1, k, j, i) = Vc(VX1, k, j, iref);
38         Vc(VX2, k, j, i) = Vk;
39
40     });
41 }
42 }
43 idfx::popRegion();
44 }
45
46 // Dust User-defined boundaries
47 void BCDust(Fluid<DustPhysics> *dust, int dir, BoundarySide side, real t) {
48 idfx::pushRegion("UserDefinedBoundary");
49 if (dir == IDIR) {
50     IdefixArray4D<real> Vc = dust->Vc;
51     IdefixArray1D<real> x1 = dust->data->x[IDIR];
52     int iref;
53     if (side == left) {
54         iref = dust->data->beg[IDIR];
55     } else if (side == right) {
56         iref = dust->data->end[IDIR]-1;
57     }
58     dust->boundary->BoundaryFor("UserDefBoundary_X1Dust", dir, side,
59 KOKKOS_LAMBDA (int k, int j, int i) {
60         real R=x1(i);
61         real Vk = 1.0/sqrt(R);
62
63         Vc(RHO, k, j, i) = Vc(RHO, k, j, iref);
64         Vc(VX1, k, j, i) = Vc(VX1, k, j, iref);
65         Vc(VX2, k, j, i) = Vk;
66
67     });
68 }
69 }
70 idfx::popRegion();
71 }

```

```

73 // Default constructor
74 Setup::Setup(Input &input, Grid &grid, DataBlock &data, Output &output)
75 {
76     data.hydro->EnrollUserDefBoundary(&BCGas);
77     data.hydro->EnrollIsoSoundSpeed(&MySoundSpeed);
78     SetupVariables::h0 = input.def<real>("Setup", "h0", 0);
79 }
80
81 void Setup::InitFlow(DataBlock &data) {
82     DataBlockHost d(data);
83     real h0 = SetupVariables::h0;
84
85     for (int k = 0; k < d.np_tot[KDIR]; k++) {
86         for (int j = 0; j < d.np_tot[JDIR]; j++) {
87             for (int i = 0; i < d.np_tot[IDIR]; i++) {
88                 real R=d.x[IDIR](i);
89                 real phi = d.x[JDIR](j);
90                 real Vk=1.0/sqrt(R);
91                 real cs2=(h0*Vk)*(h0*Vk);
92
93                 // Gas initial conditions
94                 d.Vc(RHO, k, j, i) = 100/R;
95                 d.Vc(VX1, k, j, i) = 0.0;
96                 d.Vc(VX2, k, j, i) = Vk*sqrt(1 - 2.0*h0*h0);
97
98             }
99
100         }
101     }
102
103     // Dust initial conditions
104     d.dustVc[0](RHO, k, j, i) = d.Vc(RHO, k, j, i) / 100.0;
105     d.dustVc[0](VX1, k, j, i) = 0.0;
106     d.dustVc[0](VX2, k, j, i) = Vk;
107
108 }
109
110 void SyncToDevice() {
111 }
112
113 }

```

definitions.hpp

```

1 #define COMPONENTS 2
2 #define DIMENSIONS 2
3
4 #define ISOTHERMAL
5
6 #define GEOMETRY POLAR

```

idefix.in

```

1 [Grid]
2 X1-grid 1 0.4 128 l 2.5
3 X2-grid 1 0.0 256 u 6.283185307179586
4 X3-grid 1 -1 1 u 1
5
6
7
8
9
10
11
12 [Hydro]
13 solver hllc
14 csiso userdef
15
16
17 [Dust]
18 nSpecies 1
19 drag tau 1.0
20
21
22
23 [Gravity]
24 potential central planet
25 Mcentral 1.0
26
27 [Planet]
28 integrator analytical
29 planetToPrimary 1.0e-3
30 initialDistance 1.0
31 feelDisk false
32 feelPlanets false
33 smoothing plummer 0.03 0.0
34
35 [Boundary]
36 X1-beg userdef
37 X1-end userdef
38 X2-beg periodic
39 X2-end periodic
40 X3-beg outflow
41 X3-end outflow
42
43 [Setup]
44 h0 0.05
45
46
47
48
49

```

# EXAMPLE OF SETUP

## Using Idefix

<https://idefix.readthedocs.io/latest/index.html>

setup.cpp

```

1 #include <algorithm>
2 #include "idefix.hpp"
3 #include "setup.hpp"
4
5 namespace SetupVariables {
6 real h0;
7
8
9 void MySoundSpeed(DataBlock &data, const real t, IdefixArray3D<real> &cs) {
10 IdefixArray1D<real> x1 = data.x[IDIR];
11 real h0 = SetupVariables::h0;
12 idfix_for("MyCS", 0, data.np_tot[KDIR], 0, data.np_tot[JDIR], 0, data.np_tot[IDIR],
13 KOKKOS_LAMBDA (int k, int j, int i) {
14     real R = x1(i);
15     cs(k,j,i) = h0/sqrt(R);
16 });
17 }
18
19 // Gas User-defined boundaries
20 void BCGas(Fluid<DefaultPhysics> *hydro, int dir, BoundarySide side, real t) {
21 idfx::pushRegion("UserDefinedBoundary");
22 if (dir == IDIR) {
23     IdefixArray4D<real> Vc = hydro->Vc;
24     IdefixArray1D<real> x1 = hydro->data->x[IDIR];
25     int iref;
26     if (side == left) {
27         iref = hydro->data->beg[IDIR];
28     } else if (side == right) {
29         iref = hydro->data->end[IDIR]-1;
30     }
31     hydro->boundary->BoundaryFor("UserDefBoundary_X1", dir, side,
32 KOKKOS_LAMBDA (int k, int j, int i) {
33         real R=x1(i);
34         real Vk = 1.0/sqrt(R);
35
36         Vc(RHO, k, j, i) = Vc(RHO, k, j, iref);
37         Vc(VX1, k, j, i) = Vc(VX1, k, j, iref);
38         Vc(VX2, k, j, i) = Vk;
39
40     });
41 }
42 }
43 idfx::popRegion();
44 }
45
46 // Dust User-defined boundaries
47 void BCDust(Fluid<DustPhysics> *dust, int dir, BoundarySide side, real t) {
48 idfx::pushRegion("UserDefinedBoundary");
49 if (dir == IDIR) {
50     IdefixArray4D<real> Vc = dust->Vc;
51     IdefixArray1D<real> x1 = dust->data->x[IDIR];
52     int iref;
53     if (side == left) {
54         iref = dust->data->beg[IDIR];
55     } else if (side == right) {
56         iref = dust->data->end[IDIR]-1;
57     }
58     dust->boundary->BoundaryFor("UserDefBoundary_X1Dust", dir, side,
59 KOKKOS_LAMBDA (int k, int j, int i) {
60         real R=x1(i);
61         real Vk = 1.0/sqrt(R);
62
63         Vc(RHO, k, j, i) = Vc(RHO, k, j, iref);
64         Vc(VX1, k, j, i) = Vc(VX1, k, j, iref);
65         Vc(VX2, k, j, i) = Vk;
66
67     });
68 }
69 }
70 idfx::popRegion();
71 }

```

```

73 // Default constructor
74 Setup::Setup(Input &input, Grid &grid, DataBlock &data, Output &output)
75 {
76     data.hydro->EnrollUserDefBoundary(&BCGas);
77     data.dust[0]->EnrollUserDefBoundary(&BCDust);
78     data.hydro->EnrollIsoSoundSpeed(&MySoundSpeed);
79     SetupVariables::h0 = input.Get<real>("Setup", "h0", 0);
80 }
81
82 void Setup::InitFlow(DataBlock &data) {
83     DataBlockHost d(data);
84     real h0 = SetupVariables::h0;
85
86     for (int k = 0; k < d.np_tot[KDIR]; k++) {
87         for (int j = 0; j < d.np_tot[JDIR]; j++) {
88             for (int i = 0; i < d.np_tot[IDIR]; i++) {
89                 real R=d.x[IDIR](i);
90                 real phi = d.x[JDIR](j);
91                 real Vk=1.0/sqrt(R);
92                 real cs2=(h0*Vk)*(h0*Vk);
93
94                 // Gas initial conditions
95                 d.Vc(RHO, k, j, i) = 100/R;
96                 d.Vc(VX1, k, j, i) = 0.0;
97                 d.Vc(VX2, k, j, i) = Vk*sqrt(1 - 2.0*h0*h0);
98
99             }
100         }
101     }
102
103     // Dust initial conditions
104     d.dustVc[0](RHO, k, j, i) = d.Vc(RHO, k, j, i) / 100.0;
105     d.dustVc[0](VX1, k, j, i) = 0.0;
106     d.dustVc[0](VX2, k, j, i) = Vk;
107
108     }
109     }
110     }
111     }
112     }
113     }
114     }
115     }
116     }
117     }
118     }
119     }
120     }
121     }
122     }
123     }
124     }
125     }
126     }
127     }
128     }
129     }
130     }
131     }
132     }
133     }
134     }
135     }
136     }
137 }

```

definitions.hpp

```

1 #define COMPONENTS 2
2 #define DIMENSIONS 2
3
4 #define ISOTHERMAL
5
6 #define GEOMETRY POLAR

```

idefix.in

```

1 [Grid]
2 X1-grid 1 0.4 128 l 2.5
3 X2-grid 1 0.0 256 u 6.283185307179586
4 X3-grid 1 -1 1 u 1
5
6
7
8
9
10
11
12 [Hydro]
13 solver hllc
14 csiso userdef
15
16
17 [Dust]
18 nSpecies 1
19 drag tau 1.0
20
21
22
23 [Gravity]
24 potential central planet
25 Mcentral 1.0
26
27 [Planet]
28 integrator analytical
29 planetToPrimary 1.0e-3
30 initialDistance 1.0
31 feelDisk false
32 feelPlanets false
33 smoothing plummer 0.03 0.0
34
35 [Boundary]
36 X1-beg userdef
37 X1-end userdef
38 X2-beg periodic
39 X2-end periodic
40 X3-beg outflow
41 X3-end outflow
42
43 [Setup]
44 h0 0.05
45
46
47
48
49

```

# EXAMPLE OF SETUP

## Using Idefix

<https://idefix.readthedocs.io/latest/index.html>

setup.cpp

```

1 #include <algorithm>
2 #include "idefix.hpp"
3 #include "setup.hpp"
4
5 namespace SetupVariables {
6 real h0;
7
8
9 void MySoundSpeed(DataBlock &data, const real t, IdefixArray3D<real> &cs) {
10 IdefixArray1D<real> x1 = data.x[IDIR];
11 real h0 = SetupVariables::h0;
12 idfix_for("MyCS", 0, data.np_tot[KDIR], 0, data.np_tot[JDIR], 0, data.np_tot[IDIR],
13 KOKKOS_LAMBDA (int k, int j, int i) {
14     real R = x1(i);
15     cs(k,j,i) = h0/sqrt(R);
16 });
17 }
18
19 // Gas User-defined boundaries
20 void BCGas(Fluid<DefaultPhysics> *hydro, int dir, BoundarySide side, real t) {
21 idfx::pushRegion("UserDefinedBoundary");
22 if (dir==IDIR) {
23     IdefixArray4D<real> Vc = hydro->Vc;
24     IdefixArray1D<real> x1 = hydro->data->x[IDIR];
25     int iref;
26     if (side == left) {
27         iref = hydro->data->beg[IDIR];
28     } else if (side==right) {
29         iref = hydro->data->end[IDIR]-1;
30     }
31     hydro->boundary->BoundaryFor("UserDefBoundary_X1", dir, side,
32 KOKKOS_LAMBDA (int k, int j, int i) {
33         real R=x1(i);
34         real Vc = 1.0/sqrt(R);
35
36         Vc(RHO, k, j, i) = Vc(RHO, k, j, iref);
37         Vc(VX1, k, j, i) = Vc(VX1, k, j, iref);
38         Vc(VX2, k, j, i) = Vc(VX2, k, j, i);
39
40     });
41 }
42 }
43 idfx::popRegion();
44 }
45
46 // Dust User-defined boundaries
47 void BCDust(Fluid<DustPhysics> *dust, int dir, BoundarySide side, real t) {
48 idfx::pushRegion("UserDefinedBoundary");
49 if (dir==IDIR) {
50     IdefixArray4D<real> Vc = dust->Vc;
51     IdefixArray1D<real> x1 = dust->data->x[IDIR];
52     int iref;
53     if (side == left) {
54         iref = dust->data->beg[IDIR];
55     } else if (side==right) {
56         iref = dust->data->end[IDIR]-1;
57     }
58     dust->boundary->BoundaryFor("UserDefBoundary_X1Dust", dir, side,
59 KOKKOS_LAMBDA (int k, int j, int i) {
60         real R=x1(i);
61         real Vc = 1.0/sqrt(R);
62
63         Vc(RHO, k, j, i) = Vc(RHO, k, j, iref);
64         Vc(VX1, k, j, i) = Vc(VX1, k, j, iref);
65         Vc(VX2, k, j, i) = Vc(VX2, k, j, i);
66
67     });
68 }
69 }
70 idfx::popRegion();
71 }

```

```

73 // Default constructor
74 Setup::Setup(Input &input, Grid &grid, DataBlock &data, Output &output)
75 {
76     data.hydro->EnrollUserDefBoundary(&BCGas);
77     data.dust[0]->EnrollUserDefBoundary(&BCDust);
78     data.hydro->EnrollIsoSoundSpeed(&MySoundSpeed);
79     SetupVariables::h0 = input.Get<real>("Setup", "h0", 0);
80 }
81
82 void Setup::InitFlow(DataBlock &data) {
83     DataBlockHost d(data);
84     real h0 = SetupVariables::h0;
85
86     for (int k = 0; k < d.np_tot[KDIR]; k++) {
87         for (int j = 0; j < d.np_tot[JDIR]; j++) {
88             for (int i = 0; i < d.np_tot[IDIR]; i++) {
89                 real R=d.x[IDIR](i);
90                 real phi = d.x[JDIR](j);
91                 real Vc=1.0/sqrt(R);
92                 real cs2=(h0*Vc)*(h0*Vc);
93
94                 // Gas initial conditions
95                 d.Vc(RHO, k, j, i) = 100/R;
96                 d.Vc(VX1, k, j, i) = 0.0;
97                 d.Vc(VX2, k, j, i) = Vc*sqrt(1 - 2.0*h0*h0);
98
99             }
100         }
101     }
102
103     // Dust initial conditions
104     d.dustVc[0](RHO, k, j, i) = d.Vc(RHO, k, j, i) / 100.0;
105     d.dustVc[0](VX1, k, j, i) = 0.0;
106     d.dustVc[0](VX2, k, j, i) = Vc;
107
108     }
109     }
110     }
111     }
112     }
113     }
114     }
115     }
116     }
117     }
118     }
119     }
120     }
121     }
122     }
123     }
124     }
125     }
126     }
127     }
128     }
129     }
130     }
131     }
132     }
133     }
134     }
135     }
136     }
137     }

```

definitions.hpp

```

1 #define COMPONENTS 2
2 #define DIMENSIONS 2
3
4 #define ISOTHERMAL
5
6 #define GEOMETRY POLAR

```

idefix.in

```

1 [Grid]
2 X1-grid 1 0.4 128 l 2.5
3 X2-grid 1 0.0 256 u 6.283185307179586
4 X3-grid 1 -1 1 u 1
5
6
7
8
9
10
11
12 [Hydro]
13 solver hllc
14 csiso userdef
15 tracer 2
16
17 [Dust]
18 nSpecies 1
19 drag tau 1.0
20
21
22
23 [Gravity]
24 potential central planet
25 Mcentral 1.0
26
27 [Planet]
28 integrator analytical
29 planetToPrimary 1.0e-3
30 initialDistance 1.0
31 feelDisk false
32 feelPlanets false
33 smoothing plummer 0.03 0.0
34
35 [Boundary]
36 X1-beg userdef
37 X1-end userdef
38 X2-beg periodic
39 X2-end periodic
40 X3-beg outflow
41 X3-end outflow
42
43 [Setup]
44 h0 0.05
45
46
47
48
49

```

# EXAMPLE OF SETUP

## Using Idefix

<https://idefix.readthedocs.io/latest/index.html>

setup.cpp

```

1 #include <algorithm>
2 #include "idefix.hpp"
3 #include "setup.hpp"
4
5 namespace SetupVariables {
6 real h0;
7
8
9 void MySoundSpeed(DataBlock &data, const real t, IdefixArray3D<real> &cs) {
10 IdefixArray1D<real> x1 = data.x[IDIR];
11 real h0 = SetupVariables::h0;
12 idfix_for("MyCS", 0, data.np_tot[KDIR], 0, data.np_tot[JDIR], 0, data.np_tot[IDIR],
13 KOKKOS_LAMBDA (int k, int j, int i) {
14     real R = x1(i);
15     cs(k,j,i) = h0/sqrt(R);
16 });
17 }
18
19 // Gas User-defined boundaries
20 void BCGas(Fluid<DefaultPhysics> *hydro, int dir, BoundarySide side, real t) {
21 idfx::pushRegion("UserDefinedBoundary");
22 if(dir==IDIR) {
23     IdefixArray4D<real> Vc = hydro->Vc;
24     IdefixArray1D<real> x1 = hydro->data->x[IDIR];
25     int iref;
26     if(side == left) {
27         iref = hydro->data->beg[IDIR];
28     } else if (side==right) {
29         iref = hydro->data->end[IDIR]-1;
30     }
31     hydro->boundary->BoundaryFor("UserDefBoundary_X1", dir, side,
32 KOKKOS_LAMBDA (int k, int j, int i) {
33         real R=x1(i);
34         real Vk = 1.0/sqrt(R);
35
36         Vc(RHO,k,j,i) = Vc(RHO,k,j,iref);
37         Vc(VX1,k,j,i) = Vc(VX1,k,j,iref);
38         Vc(VX2,k,j,i) = Vk;
39
40     });
41 }
42 }
43 idfx::popRegion();
44 }
45
46 // Dust User-defined boundaries
47 void BCDust(Fluid<DustPhysics> *dust, int dir, BoundarySide side, real t) {
48 idfx::pushRegion("UserDefinedBoundary");
49 if(dir==IDIR) {
50     IdefixArray4D<real> Vc = dust->Vc;
51     IdefixArray1D<real> x1 = dust->data->x[IDIR];
52     int iref;
53     if(side == left) {
54         iref = dust->data->beg[IDIR];
55     } else if (side==right) {
56         iref = dust->data->end[IDIR]-1;
57     }
58     dust->boundary->BoundaryFor("UserDefBoundary_X1Dust", dir, side,
59 KOKKOS_LAMBDA (int k, int j, int i) {
60         real R=x1(i);
61         real Vk = 1.0/sqrt(R);
62
63         Vc(RHO,k,j,i) = Vc(RHO,k,j,iref);
64         Vc(VX1,k,j,i) = Vc(VX1,k,j,iref);
65         Vc(VX2,k,j,i) = Vk;
66
67     });
68 }
69 }
70 idfx::popRegion();
71 }

```

```

73 // Default constructor
74 Setup::Setup(Input &input, Grid &grid, DataBlock &data, Output &output)
75 {
76     data.hydro->EnrollUserDefBoundary(&BCGas);
77     data.dust[0]->EnrollUserDefBoundary(&BCDust);
78     data.hydro->EnrollIsoSoundSpeed(&MySoundSpeed);
79     SetupVariables::h0 = input.Get<real>("Setup", "h0", 0);
80 }
81
82 void Setup::InitFlow(DataBlock &data) {
83     DataBlockHost d(data);
84     real h0 = SetupVariables::h0;
85
86     for(int k = 0; k < d.np_tot[KDIR]; k++) {
87         for(int i = 0; i < d.np_tot[JDIR]; i++) {
88             for(int j = 0; j < d.np_tot[IDIR]; j++) {
89                 real R=d.x[IDIR](i);
90                 real phi = d.x[JDIR](j);
91                 real Vk=1.0/sqrt(R);
92                 real cs2=(h0*Vk)*(h0*Vk);
93
94                 // Gas initial conditions
95                 d.Vc(RHO,k,j,i) = 100/R;
96                 d.Vc(VX1,k,j,i) = 0.0;
97                 d.Vc(VX2,k,j,i) = Vk*sqrt(1 - 2.0*h0*h0);
98
99                 // Tracers
100                 if(R>1.0) {
101                     d.Vc(TRG,k,j,i) = 1.0;
102                 } else {
103                     d.Vc(TRG,k,j,i) = 0.0;
104                 }
105
106                 if(R>1.05) {
107                     d.Vc(TRG+1,k,j,i) = 0.0;
108                 } else if(R<0.95) {
109                     d.Vc(TRG+1,k,j,i) = 0.0;
110                 } else {
111                     d.Vc(TRG+1,k,j,i) = 1.0;
112                 }
113
114                 // Dust initial conditions
115                 d.dustVc[0](RHO,k,j,i) = d.Vc(RHO,k,j,i) / 100.0;
116                 d.dustVc[0](VX1,k,j,i) = 0.0;
117                 d.dustVc[0](VX2,k,j,i) = Vk;
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137 }
138 }
139 }
140 }
141 }
142 }
143 }
144 }
145 }
146 }
147 }
148 }
149 }
150 }
151 }
152 }
153 }
154 }
155 }
156 }
157 }
158 }
159 }
160 }
161 }
162 }
163 }
164 }
165 }
166 }
167 }
168 }
169 }
170 }
171 }
172 }
173 }
174 }
175 }
176 }
177 }
178 }
179 }
180 }
181 }
182 }
183 }
184 }
185 }
186 }
187 }
188 }
189 }
190 }
191 }
192 }
193 }
194 }
195 }
196 }
197 }
198 }
199 }
200 }
201 }
202 }
203 }
204 }
205 }
206 }
207 }
208 }
209 }
210 }
211 }
212 }
213 }
214 }
215 }
216 }
217 }
218 }
219 }
220 }
221 }
222 }
223 }
224 }
225 }
226 }
227 }
228 }
229 }
230 }
231 }
232 }
233 }
234 }
235 }
236 }
237 }
238 }
239 }
240 }
241 }
242 }
243 }
244 }
245 }
246 }
247 }
248 }
249 }
250 }
251 }
252 }
253 }
254 }
255 }
256 }
257 }
258 }
259 }
260 }
261 }
262 }
263 }
264 }
265 }
266 }
267 }
268 }
269 }
270 }
271 }
272 }
273 }
274 }
275 }
276 }
277 }
278 }
279 }
280 }
281 }
282 }
283 }
284 }
285 }
286 }
287 }
288 }
289 }
290 }
291 }
292 }
293 }
294 }
295 }
296 }
297 }
298 }
299 }
300 }
301 }
302 }
303 }
304 }
305 }
306 }
307 }
308 }
309 }
310 }
311 }
312 }
313 }
314 }
315 }
316 }
317 }
318 }
319 }
320 }
321 }
322 }
323 }
324 }
325 }
326 }
327 }
328 }
329 }
330 }
331 }
332 }
333 }
334 }
335 }
336 }
337 }
338 }
339 }
340 }
341 }
342 }
343 }
344 }
345 }
346 }
347 }
348 }
349 }
350 }
351 }
352 }
353 }
354 }
355 }
356 }
357 }
358 }
359 }
360 }
361 }
362 }
363 }
364 }
365 }
366 }
367 }
368 }
369 }
370 }
371 }
372 }
373 }
374 }
375 }
376 }
377 }
378 }
379 }
380 }
381 }
382 }
383 }
384 }
385 }
386 }
387 }
388 }
389 }
390 }
391 }
392 }
393 }
394 }
395 }
396 }
397 }
398 }
399 }
400 }
401 }
402 }
403 }
404 }
405 }
406 }
407 }
408 }
409 }
410 }
411 }
412 }
413 }
414 }
415 }
416 }
417 }
418 }
419 }
420 }
421 }
422 }
423 }
424 }
425 }
426 }
427 }
428 }
429 }
430 }
431 }
432 }
433 }
434 }
435 }
436 }
437 }
438 }
439 }
440 }
441 }
442 }
443 }
444 }
445 }
446 }
447 }
448 }
449 }
450 }
451 }
452 }
453 }
454 }
455 }
456 }
457 }
458 }
459 }
460 }
461 }
462 }
463 }
464 }
465 }
466 }
467 }
468 }
469 }
470 }
471 }
472 }
473 }
474 }
475 }
476 }
477 }
478 }
479 }
480 }
481 }
482 }
483 }
484 }
485 }
486 }
487 }
488 }
489 }
490 }
491 }
492 }
493 }
494 }
495 }
496 }
497 }
498 }
499 }
500 }
501 }
502 }
503 }
504 }
505 }
506 }
507 }
508 }
509 }
510 }
511 }
512 }
513 }
514 }
515 }
516 }
517 }
518 }
519 }
520 }
521 }
522 }
523 }
524 }
525 }
526 }
527 }
528 }
529 }
530 }
531 }
532 }
533 }
534 }
535 }
536 }
537 }
538 }
539 }
540 }
541 }
542 }
543 }
544 }
545 }
546 }
547 }
548 }
549 }
550 }
551 }
552 }
553 }
554 }
555 }
556 }
557 }
558 }
559 }
560 }
561 }
562 }
563 }
564 }
565 }
566 }
567 }
568 }
569 }
570 }
571 }
572 }
573 }
574 }
575 }
576 }
577 }
578 }
579 }
580 }
581 }
582 }
583 }
584 }
585 }
586 }
587 }
588 }
589 }
590 }
591 }
592 }
593 }
594 }
595 }
596 }
597 }
598 }
599 }
600 }
601 }
602 }
603 }
604 }
605 }
606 }
607 }
608 }
609 }
610 }
611 }
612 }
613 }
614 }
615 }
616 }
617 }
618 }
619 }
620 }
621 }
622 }
623 }
624 }
625 }
626 }
627 }
628 }
629 }
630 }
631 }
632 }
633 }
634 }
635 }
636 }
637 }
638 }
639 }
640 }
641 }
642 }
643 }
644 }
645 }
646 }
647 }
648 }
649 }
650 }
651 }
652 }
653 }
654 }
655 }
656 }
657 }
658 }
659 }
660 }
661 }
662 }
663 }
664 }
665 }
666 }
667 }
668 }
669 }
670 }
671 }
672 }
673 }
674 }
675 }
676 }
677 }
678 }
679 }
680 }
681 }
682 }
683 }
684 }
685 }
686 }
687 }
688 }
689 }
690 }
691 }
692 }
693 }
694 }
695 }
696 }
697 }
698 }
699 }
700 }
701 }
702 }
703 }
704 }
705 }
706 }
707 }
708 }
709 }
710 }
711 }
712 }
713 }
714 }
715 }
716 }
717 }
718 }
719 }
720 }
721 }
722 }
723 }
724 }
725 }
726 }
727 }
728 }
729 }
730 }
731 }
732 }
733 }
734 }
735 }
736 }
737 }
738 }
739 }
740 }
741 }
742 }
743 }
744 }
745 }
746 }
747 }
748 }
749 }
750 }
751 }
752 }
753 }
754 }
755 }
756 }
757 }
758 }
759 }
760 }
761 }
762 }
763 }
764 }
765 }
766 }
767 }
768 }
769 }
770 }
771 }
772 }
773 }
774 }
775 }
776 }
777 }
778 }
779 }
780 }
781 }
782 }
783 }
784 }
785 }
786 }
787 }
788 }
789 }
790 }
791 }
792 }
793 }
794 }
795 }
796 }
797 }
798 }
799 }
800 }
801 }
802 }
803 }
804 }
805 }
806 }
807 }
808 }
809 }
810 }
811 }
812 }
813 }
814 }
815 }
816 }
817 }
818 }
819 }
820 }
821 }
822 }
823 }
824 }
825 }
826 }
827 }
828 }
829 }
830 }
831 }
832 }
833 }
834 }
835 }
836 }
837 }
838 }
839 }
840 }
841 }
842 }
843 }
844 }
845 }
846 }
847 }
848 }
849 }
850 }
851 }
852 }
853 }
854 }
855 }
856 }
857 }
858 }
859 }
860 }
861 }
862 }
863 }
864 }
865 }
866 }
867 }
868 }
869 }
870 }
871 }
872 }
873 }
874 }
875 }
876 }
877 }
878 }
879 }
880 }
881 }
882 }
883 }
884 }
885 }
886 }
887 }
888 }
889 }
890 }
891 }
892 }
893 }
894 }
895 }
896 }
897 }
898 }
899 }
900 }
901 }
902 }
903 }
904 }
905 }
906 }
907 }
908 }
909 }
910 }
911 }
912 }
913 }
914 }
915 }
916 }
917 }
918 }
919 }
920 }
921 }
922 }
923 }
924 }
925 }
926 }
927 }
928 }
929 }
930 }
931 }
932 }
933 }
934 }
935 }
936 }
937 }
938 }
939 }
940 }
941 }
942 }
943 }
944 }
945 }
946 }
947 }
948 }
949 }
950 }
951 }
952 }
953 }
954 }
955 }
956 }
957 }
958 }
959 }
960 }
961 }
962 }
963 }
964 }
965 }
966 }
967 }
968 }
969 }
970 }
971 }
972 }
973 }
974 }
975 }
976 }
977 }
978 }
979 }
980 }
981 }
982 }
983 }
984 }
985 }
986 }
987 }
988 }
989 }
990 }
991 }
992 }
993 }
994 }
995 }
996 }
997 }
998 }
999 }
1000 }

```

definitions.hpp

```

1 #define COMPONENTS 2
2 #define DIMENSIONS 2
3
4 #define ISOTHERMAL
5
6 #define GEOMETRY POLAR

```

idefix.in

```

1 [Grid]
2 X1-grid 1 0.4 128 l 2.5
3 X2-grid 1 0.0 256 u 6.283185307179586
4 X3-grid 1 -1 1 u 1
5
6
7
8
9
10
11
12 [Hydro]
13 solver hllc
14 csiso userdef
15 tracer 2
16
17 [Dust]
18 nSpecies 1
19 drag tau 1.0
20
21
22
23 [Gravity]
24 potential central planet
25 Mcentral 1.0
26
27 [Planet]
28 integrator analytical
29 planetToPrimary 1.0e-3
30 initialDistance 1.0
31 feelDisk false
32 feelPlanets false
33 smoothing plummer 0.03 0.0
34
35 [Boundary]
36 X1-beg userdef
37 X1-end userdef
38 X2-beg periodic
39 X2-end periodic
40 X3-beg outflow
41 X3-end outflow
42
43 [Setup]
44 h0 0.05
45
46
47
48
49

```

# EXAMPLE OF SETUP

## Using Idefix

<https://idefix.readthedocs.io/latest/index.html>

setup.cpp

```

1 #include <algorithm>
2 #include "idefix.hpp"
3 #include "setup.hpp"
4
5 namespace SetupVariables {
6 real h0;
7
8
9 void MySoundSpeed(DataBlock &data, const real t, IdefixArray3D<real> &cs) {
10 IdefixArray1D<real> x1 = data.x[IDIR];
11 real h0 = SetupVariables::h0;
12 idfix_for("MyCS", 0, data.np_tot[KDIR], 0, data.np_tot[JDIR], 0, data.np_tot[IDIR],
13 KOKKOS_LAMBDA (int k, int j, int i) {
14     real R = x1(i);
15     cs(k,j,i) = h0/sqrt(R);
16 });
17 }
18
19 // Gas User-defined boundaries
20 void BCGas(Fluid<DefaultPhysics> *hydro, int dir, BoundarySide side, real t) {
21 idfx::pushRegion("UserDefinedBoundary");
22 if(dir==IDIR) {
23     IdefixArray4D<real> Vc = hydro->Vc;
24     IdefixArray1D<real> x1 = hydro->data->x[IDIR];
25     int iref;
26     if(side == left) {
27         iref = hydro->data->beg[IDIR];
28     } else if (side==right) {
29         iref = hydro->data->end[IDIR]-1;
30     }
31     hydro->boundary->BoundaryFor("UserDefBoundary_X1", dir, side,
32 KOKKOS_LAMBDA (int k, int j, int i) {
33         real R=x1(i);
34         real Vc = 1.0/sqrt(R);
35
36         Vc(RHO, k, j, i) = Vc(RHO, k, j, iref);
37         Vc(VX1, k, j, i) = Vc(VX1, k, j, iref);
38         Vc(VX2, k, j, i) = Vc(VX2, k, j, iref);
39         Vc(TRG, k, j, i) = Vc(TRG, k, j, iref);
40         Vc(TRG+1, k, j, i) = Vc(TRG+1, k, j, iref);
41     });
42 }
43 idfx::popRegion();
44 }
45
46 // Dust User-defined boundaries
47 void BCDust(Fluid<DustPhysics> *dust, int dir, BoundarySide side, real t) {
48 idfx::pushRegion("UserDefinedBoundary");
49 if(dir==IDIR) {
50     IdefixArray4D<real> Vc = dust->Vc;
51     IdefixArray1D<real> x1 = dust->data->x[IDIR];
52     int iref;
53     if(side == left) {
54         iref = dust->data->beg[IDIR];
55     } else if (side==right) {
56         iref = dust->data->end[IDIR]-1;
57     }
58     dust->boundary->BoundaryFor("UserDefBoundary_X1Dust", dir, side,
59 KOKKOS_LAMBDA (int k, int j, int i) {
60         real R=x1(i);
61         real Vc = 1.0/sqrt(R);
62
63         Vc(RHO, k, j, i) = Vc(RHO, k, j, iref);
64         Vc(VX1, k, j, i) = Vc(VX1, k, j, iref);
65         Vc(VX2, k, j, i) = Vc(VX2, k, j, iref);
66
67     });
68 }
69 idfx::popRegion();
70 }
71 }

```

```

73 // Default constructor
74 Setup::Setup(Input &input, Grid &grid, DataBlock &data, Output &output)
75 {
76     data.hydro->EnrollUserDefBoundary(&BCGas);
77     data.dust[0]->EnrollUserDefBoundary(&BCDust);
78     data.hydro->EnrollIsoSoundSpeed(&MySoundSpeed);
79     SetupVariables::h0 = input.Get<real>("Setup", "h0", 0);
80 }
81
82 void Setup::InitFlow(DataBlock &data) {
83     DataBlockHost d(data);
84     real h0 = SetupVariables::h0;
85
86     for(int k = 0; k < d.np_tot[KDIR]; k++) {
87         for(int i = 0; i < d.np_tot[JDIR]; i++) {
88             for(int j = 0; j < d.np_tot[IDIR]; j++) {
89                 real R=d.x[IDIR](i);
90                 real phi = d.x[JDIR](j);
91                 real Vc=1.0/sqrt(R);
92                 real cs2=(h0*Vc)*(h0*Vc);
93
94                 // Gas initial conditions
95                 d.Vc(RHO, k, j, i) = 100/R;
96                 d.Vc(VX1, k, j, i) = 0.0;
97                 d.Vc(VX2, k, j, i) = Vc*sqrt(1 - 2.0*h0*h0);
98
99                 // Tracers
100                 if(R<1.0) {
101                     d.Vc(TRG, k, j, i) = 1.0;
102                 } else {
103                     d.Vc(TRG, k, j, i) = 0.0;
104                 }
105
106                 if(R<1.05) {
107                     d.Vc(TRG+1, k, j, i) = 0.0;
108                 } else if(R<0.95) {
109                     d.Vc(TRG+1, k, j, i) = 0.0;
110                 } else {
111                     d.Vc(TRG+1, k, j, i) = 1.0;
112                 }
113
114                 // Dust initial conditions
115                 d.dustVc[0](RHO, k, j, i) = d.Vc(RHO, k, j, i) / 100.0;
116                 d.dustVc[0](VX1, k, j, i) = 0.0;
117                 d.dustVc[0](VX2, k, j, i) = Vc;
118
119             }
120         }
121     }
122     d.SyncToDevice();
123 }

```

definitions.hpp

```

1 #define COMPONENTS 2
2 #define DIMENSIONS 2
3
4 #define ISOTHERMAL
5
6 #define GEOMETRY POLAR

```

idefix.in

```

1 [Grid]
2 X1-grid 1 0.4 128 l 2.5
3 X2-grid 1 0.0 256 u 6.283185307179586
4 X3-grid 1 -1 1 u 1
5
6
7
8
9
10
11
12 [Hydro]
13 solver hllc
14 csiso userdef
15 tracer 2
16
17 [Dust]
18 nSpecies 1
19 drag tau 1.0
20
21
22
23 [Gravity]
24 potential central planet
25 Mcentral 1.0
26
27 [Planet]
28 integrator analytical
29 planetToPrimary 1.0e-3
30 initialDistance 1.0
31 feelDisk false
32 feelPlanets false
33 smoothing plummer 0.03 0.0
34
35 [Boundary]
36 X1-beg userdef
37 X1-end userdef
38 X2-beg periodic
39 X2-end periodic
40 X3-beg outflow
41 X3-end outflow
42
43 [Setup]
44 h0 0.05
45
46
47
48
49

```

# EXAMPLE OF SETUP

## Using Idefix

<https://idefix.readthedocs.io/latest/index.html>

setup.cpp

```

1 #include <algorithm>
2 #include "idefix.hpp"
3 #include "setup.hpp"
4
5 namespace SetupVariables {
6 real h0;
7
8
9 void MySoundSpeed(DataBlock &data, const real t, IdefixArray3D<real> &cs) {
10 IdefixArray1D<real> x1 = data.x[IDIR];
11 real h0 = SetupVariables::h0;
12 idfix_for("MyCS", 0, data.np_tot[KDIR], 0, data.np_tot[JDIR], 0, data.np_tot[IDIR],
13 KOKKOS_LAMBDA (int k, int j, int i) {
14     real R = x1(i);
15     cs(k,j,i) = h0/sqrt(R);
16 });
17 }
18
19 // Gas User-defined boundaries
20 void BCGas(Fluid<DefaultPhysics> *hydro, int dir, BoundarySide side, real t) {
21 idfx::pushRegion("UserDefinedBoundary");
22 if(dir==IDIR) {
23     IdefixArray4D<real> Vc = hydro->Vc;
24     IdefixArray1D<real> x1 = hydro->data->x[IDIR];
25     int iref;
26     if(side == left) {
27         iref = hydro->data->beg[IDIR];
28     } else if (side==right) {
29         iref = hydro->data->end[IDIR]-1;
30     }
31     hydro->boundary->BoundaryFor("UserDefBoundary_X1", dir, side,
32     KOKKOS_LAMBDA (int k, int j, int i) {
33         real R=x1(i);
34         real Vc = 1.0/sqrt(R);
35
36         Vc(RHO, k, j, i) = Vc(RHO, k, j, iref);
37         Vc(VX1, k, j, i) = Vc(VX1, k, j, iref);
38         Vc(VX2, k, j, i) = Vc(VX2, k, j, i);
39         Vc(TRG, k, j, i) = Vc(TRG, k, j, iref);
40         Vc(TRG+1, k, j, i) = Vc(TRG+1, k, j, iref);
41     });
42 }
43 idfx::popRegion();
44 }
45
46 // Dust User-defined boundaries
47 void BCDust(Fluid<DustPhysics> *dust, int dir, BoundarySide side, real t) {
48 idfx::pushRegion("UserDefinedBoundary");
49 if(dir==IDIR) {
50     IdefixArray4D<real> Vc = dust->Vc;
51     IdefixArray1D<real> x1 = dust->data->x[IDIR];
52     int iref;
53     if(side == left) {
54         iref = dust->data->beg[IDIR];
55     } else if (side==right) {
56         iref = dust->data->end[IDIR]-1;
57     }
58     dust->boundary->BoundaryFor("UserDefBoundary_X1Dust", dir, side,
59     KOKKOS_LAMBDA (int k, int j, int i) {
60         real R=x1(i);
61         real Vc = 1.0/sqrt(R);
62
63         Vc(RHO, k, j, i) = Vc(RHO, k, j, iref);
64         Vc(VX1, k, j, i) = Vc(VX1, k, j, iref);
65         Vc(VX2, k, j, i) = Vc(VX2, k, j, i);
66
67     });
68 }
69 idfx::popRegion();
70 }
71 }

```

```

73 // Default constructor
74 Setup::Setup(Input &input, Grid &grid, DataBlock &data, Output &output)
75 {
76     data.hydro->EnrollUserDefBoundary(&BCGas);
77     data.dust[0]->EnrollUserDefBoundary(&BCDust);
78     data.hydro->EnrollIsoSoundSpeed(&MySoundSpeed);
79     SetupVariables::h0 = input.Get<real>("Setup", "h0", 0);
80 }
81
82 void Setup::InitFlow(DataBlock &data) {
83     DataBlockHost d(data);
84     real h0 = SetupVariables::h0;
85
86     for(int k = 0; k < d.np_tot[KDIR]; k++) {
87         for(int j = 0; j < d.np_tot[JDIR]; j++) {
88             for(int i = 0; i < d.np_tot[IDIR]; i++) {
89                 real R=d.x[IDIR](i);
90                 real phi = d.x[JDIR](j);
91                 real Vc=1.0/sqrt(R);
92                 real cs2=(h0*Vc)*(h0*Vc);
93
94                 // Gas initial conditions
95                 d.Vc(RHO, k, j, i) = 100/R;
96                 d.Vc(VX1, k, j, i) = 0.0;
97                 d.Vc(VX2, k, j, i) = Vc*sqrt(1 - 2.0*h0*h0);
98
99                 // Tracers
100                 if(R>1.0) {
101                     d.Vc(TRG, k, j, i) = 1.0;
102                 } else {
103                     d.Vc(TRG, k, j, i) = 0.0;
104                 }
105
106                 if(R>1.05) {
107                     d.Vc(TRG+1, k, j, i) = 0.0;
108                 } else if(R<0.95) {
109                     d.Vc(TRG+1, k, j, i) = 0.0;
110                 } else {
111                     d.Vc(TRG+1, k, j, i) = 1.0;
112                 }
113
114                 // Dust initial conditions
115                 d.dustVc[0](RHO, k, j, i) = d.Vc(RHO, k, j, i) / 100.0;
116                 d.dustVc[0](VX1, k, j, i) = 0.0;
117                 d.dustVc[0](VX2, k, j, i) = Vc;
118
119             }
120         }
121     }
122     d.SyncToDevice();
123 }

```

definitions.hpp

```

1 #define COMPONENTS 2
2 #define DIMENSIONS 2
3
4 #define ISOTHERMAL
5
6 #define GEOMETRY POLAR

```

idefix.in

```

1 [Grid]
2 X1-grid 1 0.4 128 l 2.5
3 X2-grid 1 0.0 256 u 6.283185307179586
4 X3-grid 1 -1 1 u 1
5
6
7
8
9
10
11
12 [Hydro]
13 solver hllc
14 csiso userdef
15 tracer 2
16
17 [Dust]
18 nSpecies 1
19 drag tau 1.0
20 tracer 2
21
22
23 [Gravity]
24 potential central planet
25 Mcentral 1.0
26
27 [Planet]
28 integrator analytical
29 planetToPrimary 1.0e-3
30 initialDistance 1.0
31 feelDisk false
32 feelPlanets false
33 smoothing plummer 0.03 0.0
34
35 [Boundary]
36 X1-beg userdef
37 X1-end userdef
38 X2-beg periodic
39 X2-end periodic
40 X3-beg outflow
41 X3-end outflow
42
43 [Setup]
44 h0 0.05
45
46
47
48
49

```

# EXAMPLE OF SETUP

## Using Idefix

<https://idefix.readthedocs.io/latest/index.html>

setup.cpp

```

1 #include <algorithm>
2 #include "idefix.hpp"
3 #include "setup.hpp"
4
5 namespace SetupVariables {
6 real h0;
7
8
9 void MySoundSpeed(DataBlock &data, const real t, IdefixArray3D<real> &cs) {
10 IdefixArray4D<real> x1 = data.x[IDIR];
11 real h0 = SetupVariables::h0;
12 idfix_for("MyCS", 0, data.np_tot[KDIR], 0, data.np_tot[JDIR], 0, data.np_tot[IDIR],
13 KOKKOS_LAMBDA (int k, int j, int i) {
14     real R = x1(i);
15     cs(k,j,i) = h0/sqrt(R);
16 });
17 }
18
19 // Gas User-defined boundaries
20 void BCGas(Fluid<DefaultPhysics> *hydro, int dir, BoundarySide side, real t) {
21 idfx::pushRegion("UserDefinedBoundary");
22 if(dir==IDIR) {
23     IdefixArray4D<real> Vc = hydro->Vc;
24     IdefixArray1D<real> x1 = hydro->data->x[IDIR];
25     int iref;
26     if(side == left) {
27         iref = hydro->data->beg[IDIR];
28     } else if (side==right) {
29         iref = hydro->data->end[IDIR]-1;
30     }
31     hydro->boundary->BoundaryFor("UserDefBoundary_X1", dir, side,
32     KOKKOS_LAMBDA (int k, int j, int i) {
33         real R=x1(i);
34         real Vc = 1.0/sqrt(R);
35
36         Vc(RHO,k,j,i) = Vc(RHO,k,j,iref);
37         Vc(VX1,k,j,i) = Vc(VX1,k,j,iref);
38         Vc(VX2,k,j,i) = Vc(VX2,k,j,iref);
39         Vc(TRG,k,j,i) = Vc(TRG,k,j,iref);
40         Vc(TRG+1,k,j,i) = Vc(TRG+1,k,j,iref);
41     });
42 }
43 idfx::popRegion();
44 }
45
46 // Dust User-defined boundaries
47 void BCDust(Fluid<DustPhysics> *dust, int dir, BoundarySide side, real t) {
48 idfx::pushRegion("UserDefinedBoundary");
49 if(dir==IDIR) {
50     IdefixArray4D<real> Vc = dust->Vc;
51     IdefixArray1D<real> x1 = dust->data->x[IDIR];
52     int iref;
53     if(side == left) {
54         iref = dust->data->beg[IDIR];
55     } else if (side==right) {
56         iref = dust->data->end[IDIR]-1;
57     }
58     dust->boundary->BoundaryFor("UserDefBoundary_X1Dust", dir, side,
59     KOKKOS_LAMBDA (int k, int j, int i) {
60         real R=x1(i);
61         real Vc = 1.0/sqrt(R);
62
63         Vc(RHO,k,j,i) = Vc(RHO,k,j,iref);
64         Vc(VX1,k,j,i) = Vc(VX1,k,j,iref);
65         Vc(VX2,k,j,i) = Vc(VX2,k,j,iref);
66
67     });
68 }
69 idfx::popRegion();
70 }
71 }

```

```

73 // Default constructor
74 Setup::Setup(Input &input, Grid &grid, DataBlock &data, Output &output)
75 {
76     data.hydro->EnrollUserDefBoundary(&BCGas);
77     data.dust[0]->EnrollUserDefBoundary(&BCDust);
78     data.hydro->EnrollIsoSoundSpeed(&MySoundSpeed);
79     SetupVariables::h0 = input.Get<real>("Setup", "h0", 0);
80 }
81
82 void Setup::InitFlow(DataBlock &data) {
83     DataBlockHost d(data);
84     real h0 = SetupVariables::h0;
85
86     for(int k = 0; k < d.np_tot[KDIR]; k++) {
87         for(int j = 0; j < d.np_tot[JDIR]; j++) {
88             for(int i = 0; i < d.np_tot[IDIR]; i++) {
89                 real R=d.x[IDIR](i);
90                 real phi = d.x[JDIR](j);
91                 real Vc=1.0/sqrt(R);
92                 real cs2=(h0*Vc)*(h0*Vc);
93
94                 // Gas initial conditions
95                 d.Vc(RHO,k,j,i) = 100/R;
96                 d.Vc(VX1,k,j,i) = 0.0;
97                 d.Vc(VX2,k,j,i) = Vc*sqrt(1 - 2.0*h0*h0);
98
99                 // Tracers
100                 if(R>1.0) {
101                     d.Vc(TRG,k,j,i) = 1.0;
102                 } else {
103                     d.Vc(TRG,k,j,i) = 0.0;
104                 }
105
106                 if(R>1.05) {
107                     d.Vc(TRG+1,k,j,i) = 0.0;
108                 } else if(R<0.95) {
109                     d.Vc(TRG+1,k,j,i) = 0.0;
110                 } else {
111                     d.Vc(TRG+1,k,j,i) = 1.0;
112                 }
113
114                 // Dust initial conditions
115                 d.dustVc[0](RHO,k,j,i) = d.Vc(RHO,k,j,i) / 100.0;
116                 d.dustVc[0](VX1,k,j,i) = 0.0;
117                 d.dustVc[0](VX2,k,j,i) = Vc;
118
119                 // Dust Tracers
120                 if(R>1) {
121                     d.dustVc[0](TRD,k,j,i) = 0.0;
122                 } else {
123                     d.dustVc[0](TRD,k,j,i) = 1.0;
124                 }
125
126                 if(phi>0.5) {
127                     d.dustVc[0](TRD+1,k,j,i) = 0.0;
128                 } else if(phi<-0.5) {
129                     d.dustVc[0](TRD+1,k,j,i) = 0.0;
130                 } else {
131                     d.dustVc[0](TRD+1,k,j,i) = 1.0;
132                 }
133
134             }
135         }
136     }
137     d.SyncToDevice();

```

definitions.hpp

```

1 #define COMPONENTS 2
2 #define DIMENSIONS 2
3
4 #define ISOTHERMAL
5
6 #define GEOMETRY POLAR

```

idefix.in

```

1 [Grid]
2 X1-grid 1 0.4 128 l 2.5
3 X2-grid 1 0.0 256 u 6.283185307179586
4 X3-grid 1 -1 1 u 1
5
6
7
8
9
10
11
12 [Hydro]
13 solver hllc
14 csiso userdef
15 tracer 2
16
17 [Dust]
18 nSpecies 1
19 drag tau 1.0
20 tracer 2
21
22
23 [Gravity]
24 potential central planet
25 nCentral 1.0
26
27 [Planet]
28 integrator analytical
29 planetToPrimary 1.0e-3
30 initialDistance 1.0
31 feelDisk false
32 feelPlanets false
33 smoothing plummer 0.03 0.0
34
35 [Boundary]
36 X1-beg userdef
37 X1-end userdef
38 X2-beg periodic
39 X2-end periodic
40 X3-beg outflow
41 X3-end outflow
42
43 [Setup]
44 h0 0.05
45
46
47
48
49

```

# EXAMPLE OF SETUP

## Using Idefix

<https://idefix.readthedocs.io/latest/index.html>

setup.cpp

```

1 #include <algorithm>
2 #include "idefix.hpp"
3 #include "setup.hpp"
4
5 namespace SetupVariables {
6 real h0;
7
8 void MySoundSpeed(DataBlock &data, const real t, IdefixArray3D<real> &cs) {
9     IdefixArray1D<real> x1 = data.x[IDIR];
10     real h0 = SetupVariables::h0;
11     idfix_for("MyCS", 0, data.np_tot[IDIR], 0, data.np_tot[JDIR], 0, data.np_tot[IDIR],
12             KOKKOS_LAMBDA (int k, int j, int i) {
13         real R = x1(i);
14         cs(k,j,i) = h0/sqrt(R);
15     });
16 }
17 }
18
19 // Gas User-defined boundaries
20 void BCGas(Fluid<DefaultPhysics> *hydro, int dir, BoundarySide side, real t) {
21     idfx::pushRegion("UserDefinedBoundary");
22     if(dir==IDIR) {
23         IdefixArray4D<real> Vc = hydro->Vc;
24         IdefixArray1D<real> x1 = hydro->data->x[IDIR];
25         int iref;
26         if(side == left) {
27             iref = hydro->data->beg[IDIR];
28         } else if (side==right) {
29             iref = hydro->data->end[IDIR]-1;
30         }
31         hydro->boundary->BoundaryFor("UserDefBoundary_X1", dir, side,
32                                     KOKKOS_LAMBDA (int k, int j, int i) {
33             real R=x1(i);
34             real Vc = 1.0/sqrt(R);
35
36             Vc(RHO,k,j,i) = Vc(RHO,k,j,iref);
37             Vc(VX1,k,j,i) = Vc(VX1,k,j,iref);
38             Vc(VX2,k,j,i) = Vc(VX2,k,j,iref);
39             Vc(TRG,k,j,i) = Vc(TRG,k,j,iref);
40             Vc(TRG+1,k,j,i) = Vc(TRG+1,k,j,iref);
41         });
42     }
43     idfx::popRegion();
44 }
45
46 // Dust User-defined boundaries
47 void BCDust(Fluid<DustPhysics> *dust, int dir, BoundarySide side, real t) {
48     idfx::pushRegion("UserDefinedBoundary");
49     if(dir==IDIR) {
50         IdefixArray4D<real> Vc = dust->Vc;
51         IdefixArray1D<real> x1 = dust->data->x[IDIR];
52         int iref;
53         if(side == left) {
54             iref = dust->data->beg[IDIR];
55         } else if (side==right) {
56             iref = dust->data->end[IDIR]-1;
57         }
58         dust->boundary->BoundaryFor("UserDefBoundary_X1Dust", dir, side,
59                                     KOKKOS_LAMBDA (int k, int j, int i) {
60             real R=x1(i);
61             real Vc = 1.0/sqrt(R);
62
63             Vc(RHO,k,j,i) = Vc(RHO,k,j,iref);
64             Vc(VX1,k,j,i) = Vc(VX1,k,j,iref);
65             Vc(VX2,k,j,i) = Vc(VX2,k,j,iref);
66             Vc(TRD,k,j,i) = Vc(TRD,k,j,iref);
67             Vc(TRD+1,k,j,i) = Vc(TRD+1,k,j,iref);
68         });
69     }
70     idfx::popRegion();
71 }
72
73 // Default constructor
74 Setup::Setup(Input &input, Grid &grid, DataBlock &data, Output &output)
75 {
76     data.hydro->EnrollUserDefBoundary(&BCGas);
77     data.dust[0]->EnrollUserDefBoundary(&BCDust);
78     data.hydro->EnrollIsoSoundSpeed(&MySoundSpeed);
79     SetupVariables::h0 = input.Get<real>("Setup", "h0", 0);
80 }
81
82 void Setup::InitFlow(DataBlock &data) {
83     DataBlockHost d(data);
84     real h0 = SetupVariables::h0;
85
86     for(int k = 0; k < d.np_tot[KDIR]; k++) {
87         for(int j = 0; j < d.np_tot[JDIR]; j++) {
88             for(int i = 0; i < d.np_tot[IDIR]; i++) {
89                 real R=d.x[IDIR](i);
90                 real phi = d.x[JDIR](j);
91                 real Vc=1.0/sqrt(R);
92                 real cs2=(h0*Vc)*(h0*Vc);
93
94                 // Gas initial conditions
95                 d.Vc(RHO,k,j,i) = 100/R;
96                 d.Vc(VX1,k,j,i) = 0.0;
97                 d.Vc(VX2,k,j,i) = Vc*sqrt(1 - 2.0*h0*h0);
98
99                 // Tracers
100                 if(R>1.0) {
101                     d.Vc(TRG,k,j,i) = 1.0;
102                 } else {
103                     d.Vc(TRG,k,j,i) = 0.0;
104                 }
105
106                 if(R>1.05) {
107                     d.Vc(TRG+1,k,j,i) = 0.0;
108                 } else if(R<0.95) {
109                     d.Vc(TRG+1,k,j,i) = 0.0;
110                 } else {
111                     d.Vc(TRG+1,k,j,i) = 1.0;
112                 }
113
114                 // Dust initial conditions
115                 d.dustVc[0](RHO,k,j,i) = d.Vc(RHO,k,j,i) / 100.0;
116                 d.dustVc[0](VX1,k,j,i) = 0.0;
117                 d.dustVc[0](VX2,k,j,i) = Vc;
118
119                 // Dust Tracers
120                 if(R>1) {
121                     d.dustVc[0](TRD,k,j,i) = 0.0;
122                 } else {
123                     d.dustVc[0](TRD,k,j,i) = 1.0;
124                 }
125
126                 if(phi>0.5) {
127                     d.dustVc[0](TRD+1,k,j,i) = 0.0;
128                 } else if(phi<-0.5) {
129                     d.dustVc[0](TRD+1,k,j,i) = 0.0;
130                 } else {
131                     d.dustVc[0](TRD+1,k,j,i) = 1.0;
132                 }
133             }
134         }
135     }
136     d.SyncToDevice();
137 }

```

definitions.hpp

```

1 #define COMPONENTS 2
2 #define DIMENSIONS 2
3
4 #define ISOTHERMAL
5
6 #define GEOMETRY POLAR

```

idefix.in

```

1 [Grid]
2 X1-grid 1 0.4 128 l 2.5
3 X2-grid 1 0.0 256 u 6.283185307179586
4 X3-grid 1 -1 1 u 1
5
6
7
8
9
10
11
12 [Hydro]
13 solver hllc
14 csiso userdef
15 tracer 2
16
17 [Dust]
18 nSpecies 1
19 drag tau 1.0
20 tracer 2
21
22
23 [Gravity]
24 potential central planet
25 Mcentral 1.0
26
27 [Planet]
28 integrator analytical
29 planetToPrimary 1.0e-3
30 initialDistance 1.0
31 feelDisk false
32 feelPlanets false
33 smoothing plummer 0.03 0.0
34
35 [Boundary]
36 X1-beg userdef
37 X1-end userdef
38 X2-beg periodic
39 X2-end periodic
40 X3-beg outflow
41 X3-end outflow
42
43 [Setup]
44 h0 0.05
45
46
47
48
49

```



# EXAMPLE OF SETUP

## Using Idefix

<https://idefix.readthedocs.io/latest/index.html>

setup.cpp

```

1 #include <algorithm>
2 #include "idefix.hpp"
3 #include "setup.hpp"
4
5 namespace SetupVariables {
6 real h0;
7
8
9 void MySoundSpeed(DataBlock &data, const real t, IdefixArray3D<real> &cs) {
10 IdefixArray4D<real> x1 = data.x[IDIR];
11 real h0 = SetupVariables::h0;
12 Idefix_for("MyCS", 0, data.np_tot[KDIR], 0, data.np_tot[JDIR], 0, data.np_tot[IDIR],
13 KOKKOS_LAMBDA (int k, int j, int i) {
14     real R = x1(i);
15     cs(k,j,i) = h0/sqrt(R);
16 });
17 }
18
19 // Gas User-defined boundaries
20 void BCGas(Fluid<DefaultPhysics> *hydro, int dir, BoundarySide side, real t) {
21 idfx::pushRegion("UserDefinedBoundary");
22 if(dir==IDIR) {
23     IdefixArray4D<real> Vc = hydro->Vc;
24     IdefixArray1D<real> x1 = hydro->data->x[IDIR];
25     int iref;
26     if(side == left) {
27         iref = hydro->data->beg[IDIR];
28     } else if (side==right) {
29         iref = hydro->data->end[IDIR]-1;
30     }
31     hydro->boundary->BoundaryFor("UserDefBoundary_X1", dir, side,
32 KOKKOS_LAMBDA (int k, int j, int i) {
33         real R=x1(i);
34         real Vk = 1.0/sqrt(R);
35
36         Vc(RHO,k,j,i) = Vc(RHO,k,j,iref);
37         Vc(VX1,k,j,i) = Vc(VX1,k,j,iref);
38         Vc(VX2,k,j,i) = Vk;
39         Vc(TRG,k,j,i) = Vc(TRG,k,j,iref);
40         Vc(TRG+1,k,j,i) = Vc(TRG+1,k,j,iref);
41     });
42 }
43 idfx::popRegion();
44 }
45
46 // Dust User-defined boundaries
47 void BCDust(Fluid<DustPhysics> *dust, int dir, BoundarySide side, real t) {
48 idfx::pushRegion("UserDefinedBoundary");
49 if(dir==IDIR) {
50     IdefixArray4D<real> Vc = dust->Vc;
51     IdefixArray1D<real> x1 = dust->data->x[IDIR];
52     int iref;
53     if(side == left) {
54         iref = dust->data->beg[IDIR];
55     } else if (side==right) {
56         iref = dust->data->end[IDIR]-1;
57     }
58     dust->boundary->BoundaryFor("UserDefBoundary_X1Dust", dir, side,
59 KOKKOS_LAMBDA (int k, int j, int i) {
60         real R=x1(i);
61         real Vk = 1.0/sqrt(R);
62
63         Vc(RHO,k,j,i) = Vc(RHO,k,j,iref);
64         Vc(VX1,k,j,i) = Vc(VX1,k,j,iref);
65         Vc(VX2,k,j,i) = Vk;
66         Vc(TRD,k,j,i) = Vc(TRD,k,j,iref);
67         Vc(TRD+1,k,j,i) = Vc(TRD+1,k,j,iref);
68     });
69 }
70 idfx::popRegion();
71 }

```

```

73 // Default constructor
74 Setup::Setup(Input &input, Grid &grid, DataBlock &data, Output &output)
75 {
76     data.hydro->EnrollUserDefBoundary(&BCGas);
77     data.dust[0]->EnrollUserDefBoundary(&BCDust);
78     data.hydro->EnrollIsoSoundSpeed(&MySoundSpeed);
79     SetupVariables::h0 = input.Get<real>("Setup", "h0", 0);
80 }
81
82 void Setup::InitFlow(DataBlock &data) {
83     DataBlockHost d(data);
84     real h0 = SetupVariables::h0;
85
86     for(int k = 0; k < d.np_tot[KDIR]; k++) {
87         for(int j = 0; j < d.np_tot[JDIR]; j++) {
88             for(int i = 0; i < d.np_tot[IDIR]; i++) {
89                 real R=d.x[IDIR](i);
90                 real phi = d.x[JDIR](j);
91                 real Vk=1.0/sqrt(R);
92                 real cs2=(h0*Vk)*(h0*Vk);
93
94                 // Gas initial conditions
95                 d.Vc(RHO,k,j,i) = 100/R;
96                 d.Vc(VX1,k,j,i) = 0.0;
97                 d.Vc(VX2,k,j,i) = Vk*sqrt(1 - 2.0*h0*h0);
98
99                 // Tracers
100                 if(R>1.0) {
101                     d.Vc(TRG,k,j,i) = 1.0;
102                 } else {
103                     d.Vc(TRG,k,j,i) = 0.0;
104                 }
105
106                 if(R>1.05) {
107                     d.Vc(TRG+1,k,j,i) = 0.0;
108                 } else if(R<0.95) {
109                     d.Vc(TRG+1,k,j,i) = 0.0;
110                 } else {
111                     d.Vc(TRG+1,k,j,i) = 1.0;
112                 }
113
114                 // Dust initial conditions
115                 d.dustVc[0](RHO,k,j,i) = d.Vc(RHO,k,j,i) / 100.0;
116                 d.dustVc[0](VX1,k,j,i) = 0.0;
117                 d.dustVc[0](VX2,k,j,i) = Vk;
118
119                 // Dust Tracers
120                 if(R>1) {
121                     d.dustVc[0](TRD,k,j,i) = 0.0;
122                 } else {
123                     d.dustVc[0](TRD,k,j,i) = 1.0;
124                 }
125
126                 if(phi>0.5) {
127                     d.dustVc[0](TRD+1,k,j,i) = 0.0;
128                 } else if(phi<-0.5) {
129                     d.dustVc[0](TRD+1,k,j,i) = 0.0;
130                 } else {
131                     d.dustVc[0](TRD+1,k,j,i) = 1.0;
132                 }
133             }
134         }
135     }
136     d.SyncToDevice();
137 }

```

definitions.hpp

```

1 #define COMPONENTS 2
2 #define DIMENSIONS 2
3
4 #define ISOTHERMAL
5
6 #define GEOMETRY POLAR

```

idefix.in

```

1 [Grid]
2 X1-grid 1 0.4 128 l 2.5
3 X2-grid 1 0.0 256 u 6.283185307179586
4 X3-grid 1 -1 1 u 1
5
6 [TimeIntegrator]
7 CFL 0.5
8 tstop 37.69911184307752 # 6 orbital periods
9 first_dt 1.e-3
10 nstages 2
11
12 [Hydro]
13 solver hllc
14 csiso userdef
15 tracer 2
16
17 [Dust]
18 nSpecies 1
19 drag tau 1.0
20 tracer 2
21
22
23 [Gravity]
24 potential central planet
25 Mcentral 1.0
26
27 [Planet]
28 integrator analytical
29 planetToPrimary 1.0e-3
30 initialDistance 1.0
31 feelDisk false
32 feelPlanets false
33 smoothing plummer 0.03 0.0
34
35 [Boundary]
36 X1-beg userdef
37 X1-end userdef
38 X2-beg periodic
39 X2-end periodic
40 X3-beg outflow
41 X3-end outflow
42
43 [Setup]
44 h0 0.05
45
46
47
48
49

```

# EXAMPLE OF SETUP

## Using Idefix

<https://idefix.readthedocs.io/latest/index.html>

setup.cpp

```

1 #include <algorithm>
2 #include "idefix.hpp"
3 #include "setup.hpp"
4
5 namespace SetupVariables {
6 real h0;
7
8
9 void MySoundSpeed(DataBlock &data, const real t, IdefixArray3D<real> &cs) {
10 IdefixArray4D<real> x1 = data.x[IDIR];
11 real h0 = SetupVariables::h0;
12 Idefix_for("MyCS", 0, data.np_tot[KDIR], 0, data.np_tot[JDIR], 0, data.np_tot[IDIR],
13 KOKKOS_LAMBDA (int k, int j, int i) {
14     real R = x1(i);
15     cs(k,j,i) = h0/sqrt(R);
16 });
17 }
18
19 // Gas User-defined boundaries
20 void BCGas(Fluid<DefaultPhysics> *hydro, int dir, BoundarySide side, real t) {
21 idfx::pushRegion("UserDefinedBoundary");
22 if(dir==IDIR) {
23     IdefixArray4D<real> Vc = hydro->Vc;
24     IdefixArray1D<real> x1 = hydro->data->x[IDIR];
25     int iref;
26     if(side == left) {
27         iref = hydro->data->beg[IDIR];
28     } else if (side==right) {
29         iref = hydro->data->end[IDIR]-1;
30     }
31     hydro->boundary->BoundaryFor("UserDefBoundary_X1", dir, side,
32 KOKKOS_LAMBDA (int k, int j, int i) {
33         real R=x1(i);
34         real Vk = 1.0/sqrt(R);
35
36         Vc(RHO,k,j,i) = Vc(RHO,k,j,iref);
37         Vc(VX1,k,j,i) = Vc(VX1,k,j,iref);
38         Vc(VX2,k,j,i) = Vk;
39         Vc(TRG,k,j,i) = Vc(TRG,k,j,iref);
40         Vc(TRG+1,k,j,i) = Vc(TRG+1,k,j,iref);
41     });
42 }
43 idfx::popRegion();
44 }
45
46 // Dust User-defined boundaries
47 void BCDust(Fluid<DustPhysics> *dust, int dir, BoundarySide side, real t) {
48 idfx::pushRegion("UserDefinedBoundary");
49 if(dir==IDIR) {
50     IdefixArray4D<real> Vc = dust->Vc;
51     IdefixArray1D<real> x1 = dust->data->x[IDIR];
52     int iref;
53     if(side == left) {
54         iref = dust->data->beg[IDIR];
55     } else if (side==right) {
56         iref = dust->data->end[IDIR]-1;
57     }
58     dust->boundary->BoundaryFor("UserDefBoundary_X1Dust", dir, side,
59 KOKKOS_LAMBDA (int k, int j, int i) {
60         real R=x1(i);
61         real Vk = 1.0/sqrt(R);
62
63         Vc(RHO,k,j,i) = Vc(RHO,k,j,iref);
64         Vc(VX1,k,j,i) = Vc(VX1,k,j,iref);
65         Vc(VX2,k,j,i) = Vk;
66         Vc(TRD,k,j,i) = Vc(TRD,k,j,iref);
67         Vc(TRD+1,k,j,i) = Vc(TRD+1,k,j,iref);
68     });
69 }
70 idfx::popRegion();
71 }

```

```

73 // Default constructor
74 Setup::Setup(Input &input, Grid &grid, DataBlock &data, Output &output)
75 {
76     data.hydro->EnrollUserDefBoundary(&BCGas);
77     data.dust[0]->EnrollUserDefBoundary(&BCDust);
78     data.hydro->EnrollIsoSoundSpeed(&MySoundSpeed);
79     SetupVariables::h0 = input.Get<real>("Setup", "h0", 0);
80 }
81
82 void Setup::InitFlow(DataBlock &data) {
83     DataBlockHost d(data);
84     real h0 = SetupVariables::h0;
85
86     for(int k = 0; k < d.np_tot[KDIR]; k++) {
87         for(int j = 0; j < d.np_tot[JDIR]; j++) {
88             for(int i = 0; i < d.np_tot[IDIR]; i++) {
89                 real R=d.x[IDIR](i);
90                 real phi = d.x[JDIR](j);
91                 real Vk=1.0/sqrt(R);
92                 real cs2=(h0*Vk)*(h0*Vk);
93
94                 // Gas initial conditions
95                 d.Vc(RHO,k,j,i) = 100/R;
96                 d.Vc(VX1,k,j,i) = 0.0;
97                 d.Vc(VX2,k,j,i) = Vk*sqrt(1 - 2.0*h0*h0);
98
99                 // Tracers
100                 if(R>1.0) {
101                     d.Vc(TRG,k,j,i) = 1.0;
102                 } else {
103                     d.Vc(TRG,k,j,i) = 0.0;
104                 }
105
106                 if(R>1.05) {
107                     d.Vc(TRG+1,k,j,i) = 0.0;
108                 } else if(R<0.95) {
109                     d.Vc(TRG+1,k,j,i) = 0.0;
110                 } else {
111                     d.Vc(TRG+1,k,j,i) = 1.0;
112                 }
113
114                 // Dust initial conditions
115                 d.dustVc[0](RHO,k,j,i) = d.Vc(RHO,k,j,i) / 100.0;
116                 d.dustVc[0](VX1,k,j,i) = 0.0;
117                 d.dustVc[0](VX2,k,j,i) = Vk;
118
119                 // Dust Tracers
120                 if(R>1) {
121                     d.dustVc[0](TRD,k,j,i) = 0.0;
122                 } else {
123                     d.dustVc[0](TRD,k,j,i) = 1.0;
124                 }
125
126                 if(phi>0.5) {
127                     d.dustVc[0](TRD+1,k,j,i) = 0.0;
128                 } else if(phi<-0.5) {
129                     d.dustVc[0](TRD+1,k,j,i) = 0.0;
130                 } else {
131                     d.dustVc[0](TRD+1,k,j,i) = 1.0;
132                 }
133
134             }
135         }
136     }
137     d.SyncToDevice();

```

definitions.hpp

```

1 #define COMPONENTS 2
2 #define DIMENSIONS 2
3
4 #define ISOTHERMAL
5
6 #define GEOMETRY POLAR

```

idefix.in

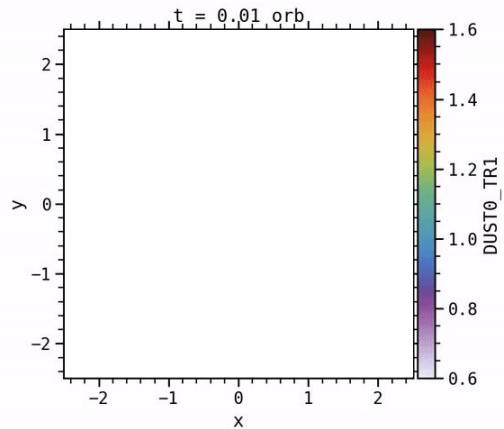
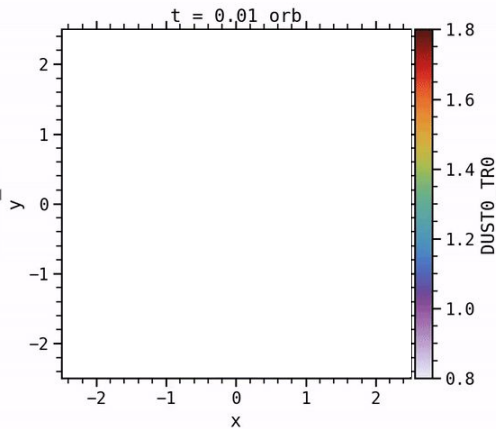
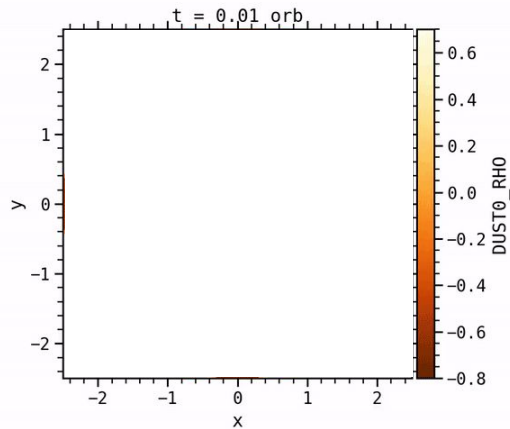
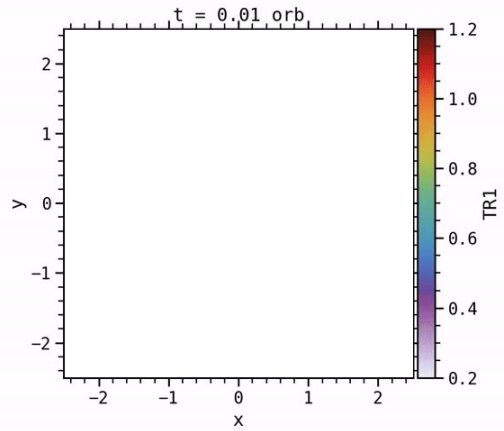
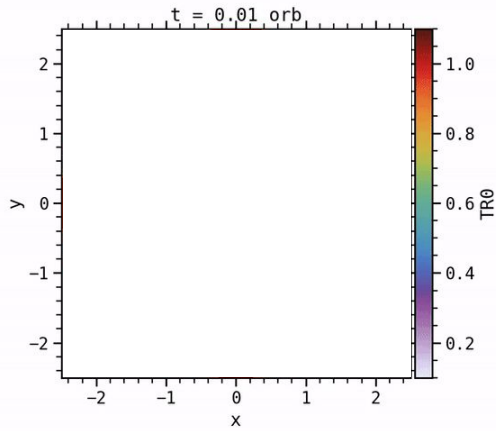
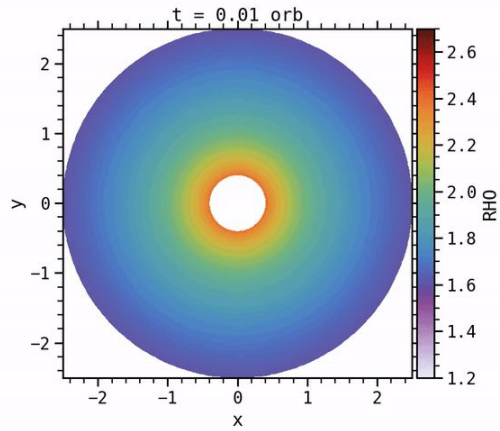
```

1 [Grid]
2 X1-grid 1 0.4 128 l 2.5
3 X2-grid 1 0.0 256 u 6.283185307179586
4 X3-grid 1 -1 1 u 1
5
6 [TimeIntegrator]
7 CFL 0.5
8 tstop 37.69911184307752 # 6 orbital periods
9 first_dt 1.e-3
10 nstages 2
11
12 [Hydro]
13 solver hllc
14 csiso userdef
15 tracer 2
16
17 [Dust]
18 nSpecies 1
19 drag tau 1.0
20 tracer 2
21
22
23 [Gravity]
24 potential central planet
25 Mcentral 1.0
26
27 [Planet]
28 integrator analytical
29 planetToPrimary 1.0e-3
30 initialDistance 1.0
31 feelDisk false
32 feelPlanets false
33 smoothing plummer 0.03 0.0
34
35 [Boundary]
36 X1-beg userdef
37 X1-end userdef
38 X2-beg periodic
39 X2-end periodic
40 X3-beg outflow
41 X3-end outflow
42
43 [Setup]
44 h0 0.05
45
46 [Output]
47 vtk 0.06283185307179586
48 dmp 6.283185307179586
49 log 100

```

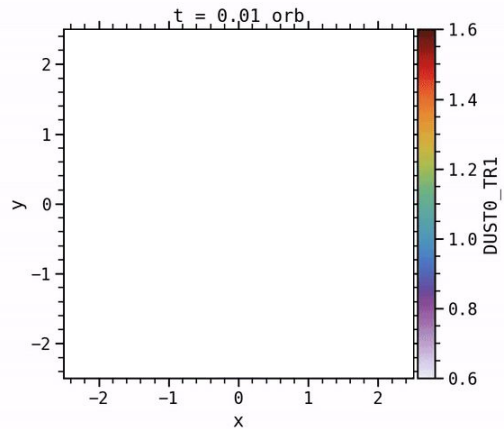
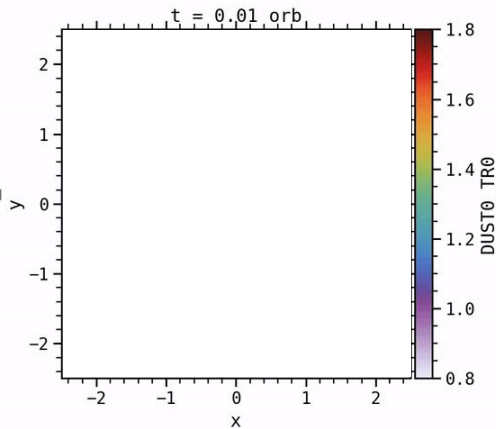
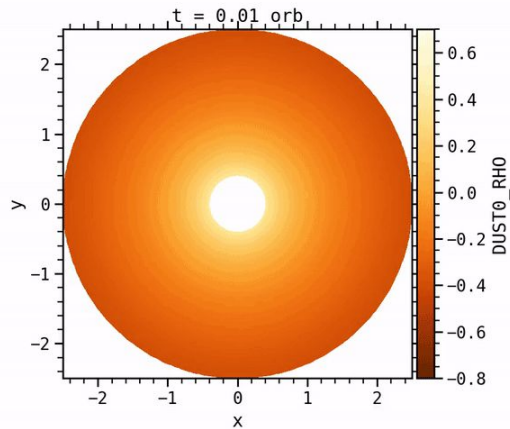
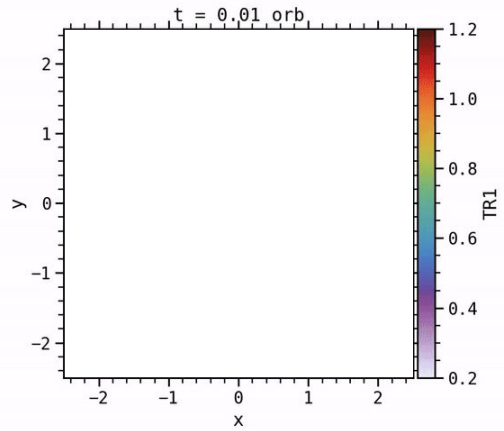
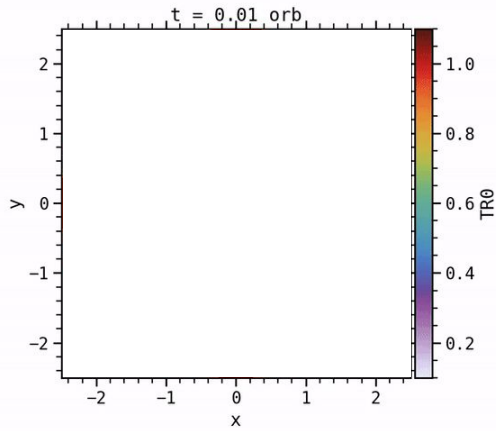
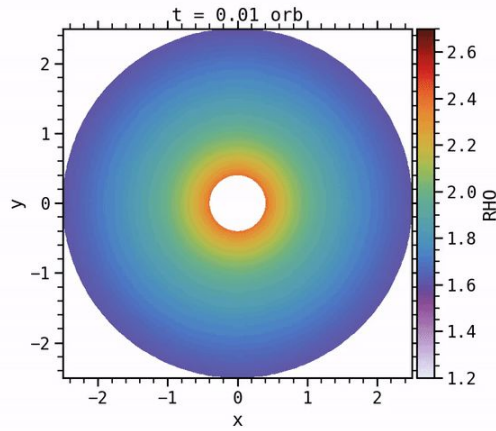
# EXAMPLE OF SETUP

## *Visualization*



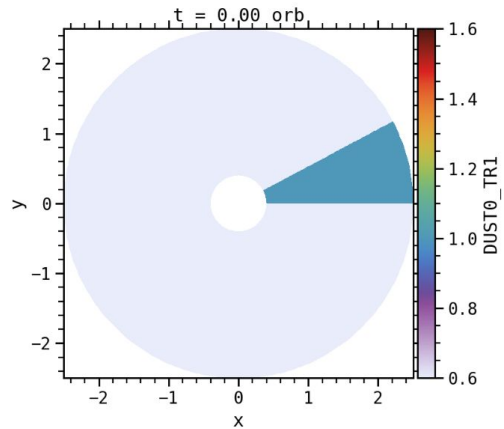
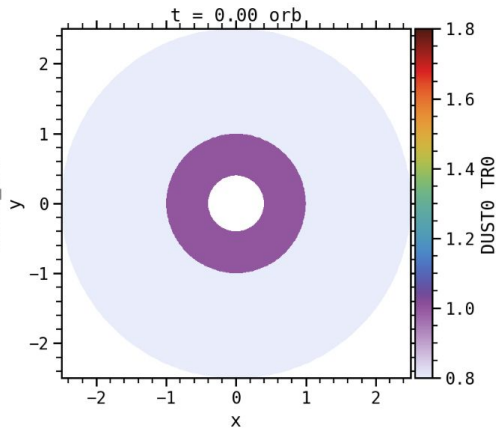
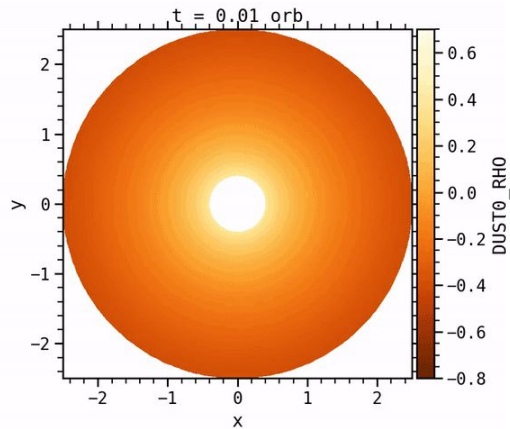
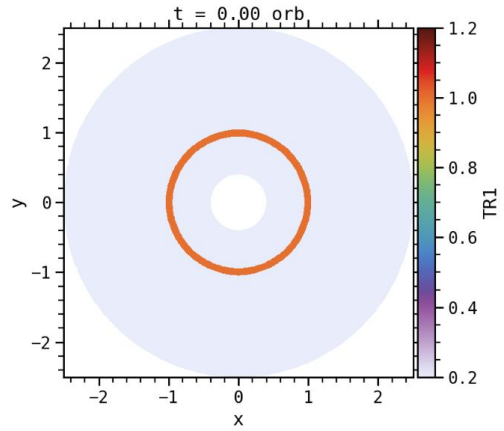
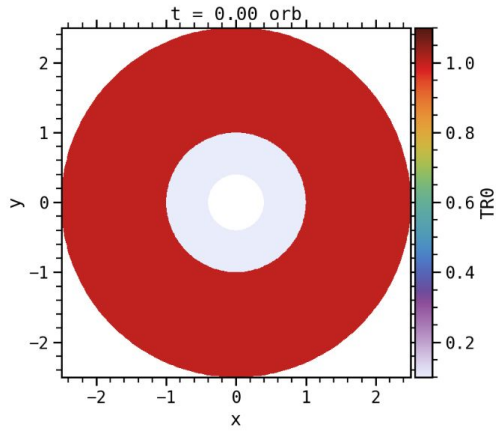
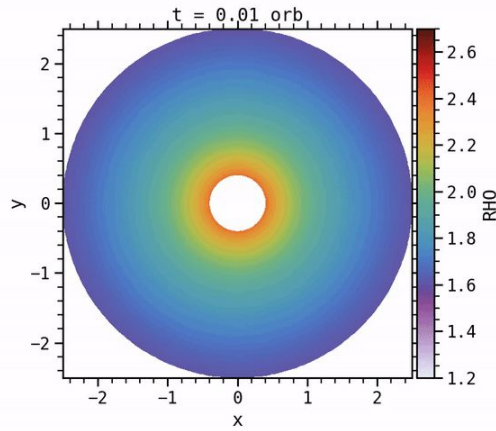
# EXAMPLE OF SETUP

## *Visualization*



# EXAMPLE OF SETUP

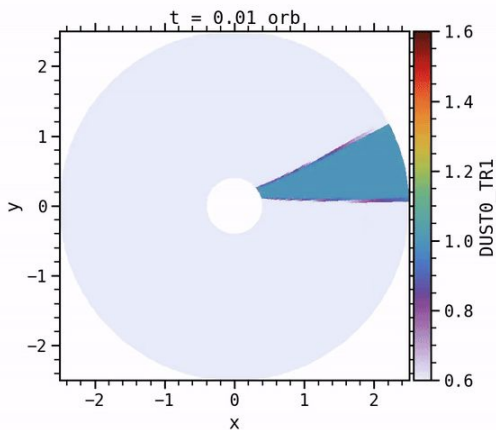
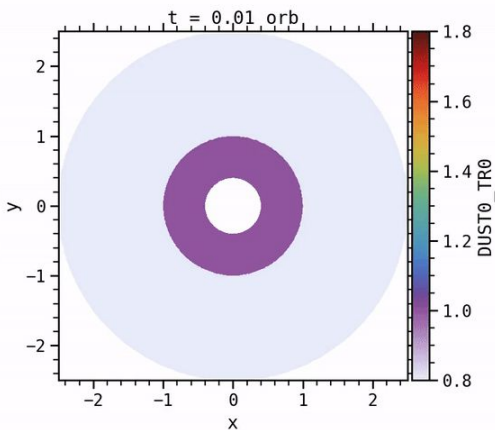
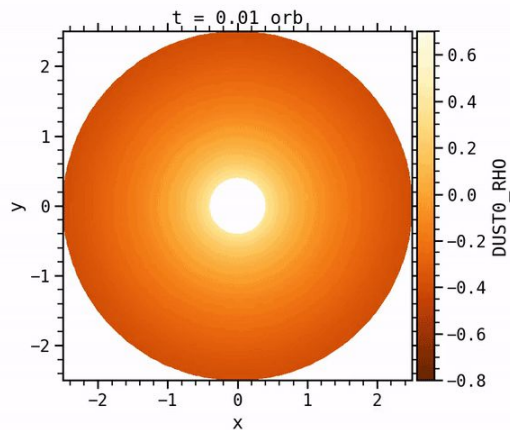
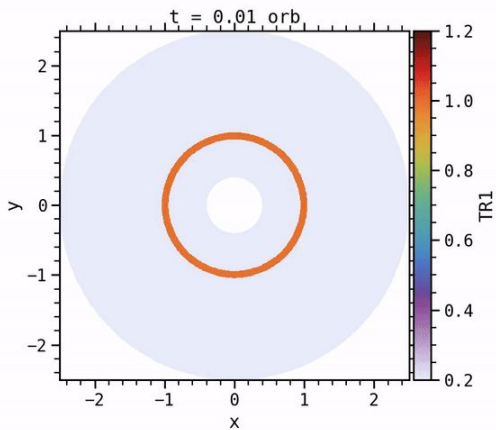
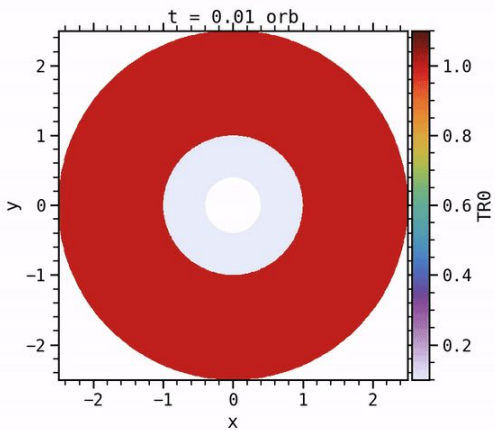
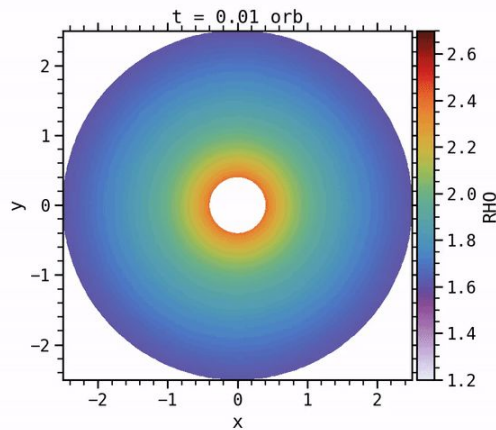
*Visualization*



# EXAMPLE OF SETUP

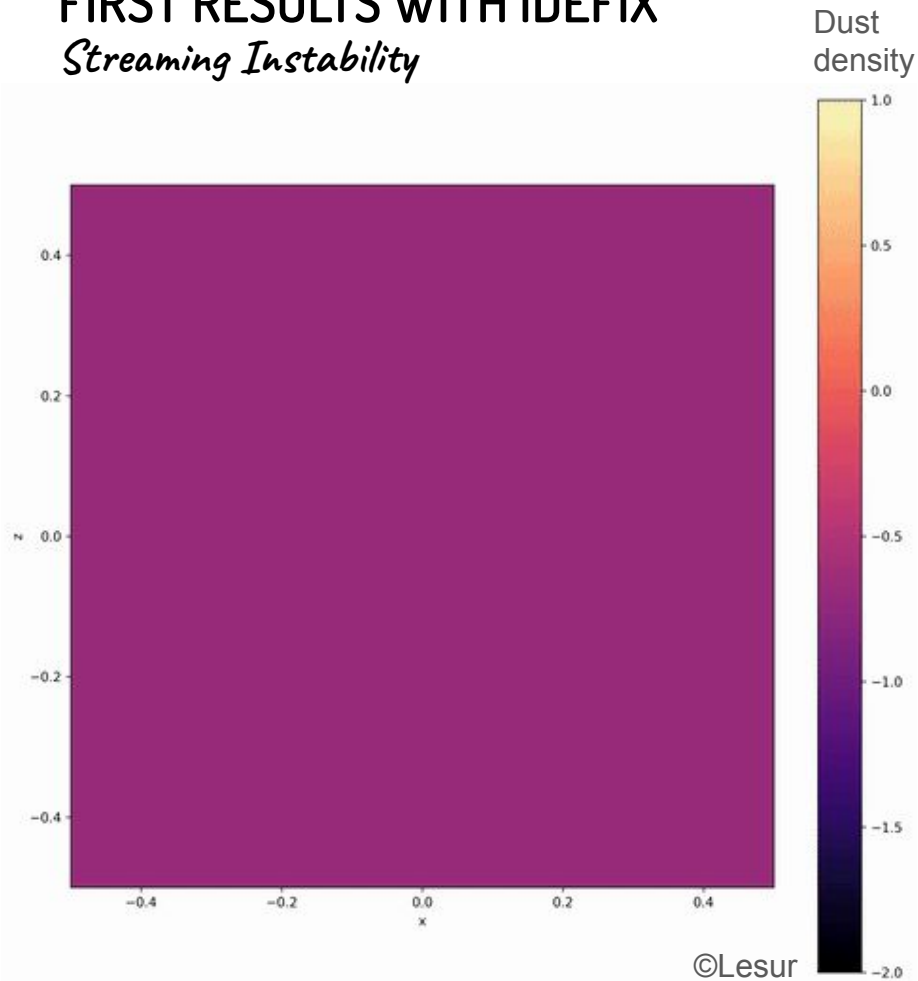
*Visualization*

$$t_d \simeq 0.8 t_g$$



# FIRST RESULTS WITH IDEFIX

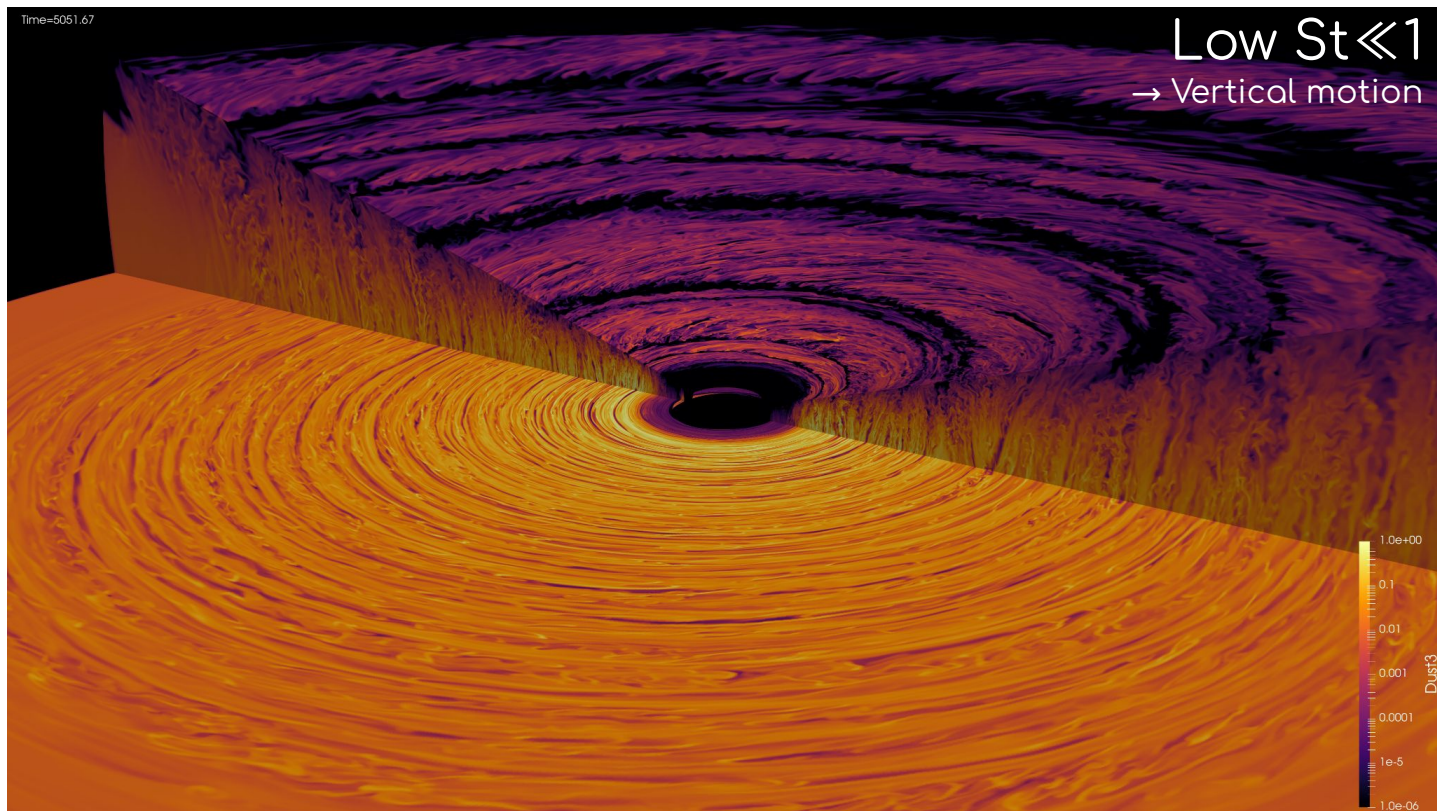
## *Streaming Instability*



See also the low-resolution test in  
`$IDEFIX_DIR/test/Dust/StreamingInstability`

# FIRST RESULTS WITH IDEFIX

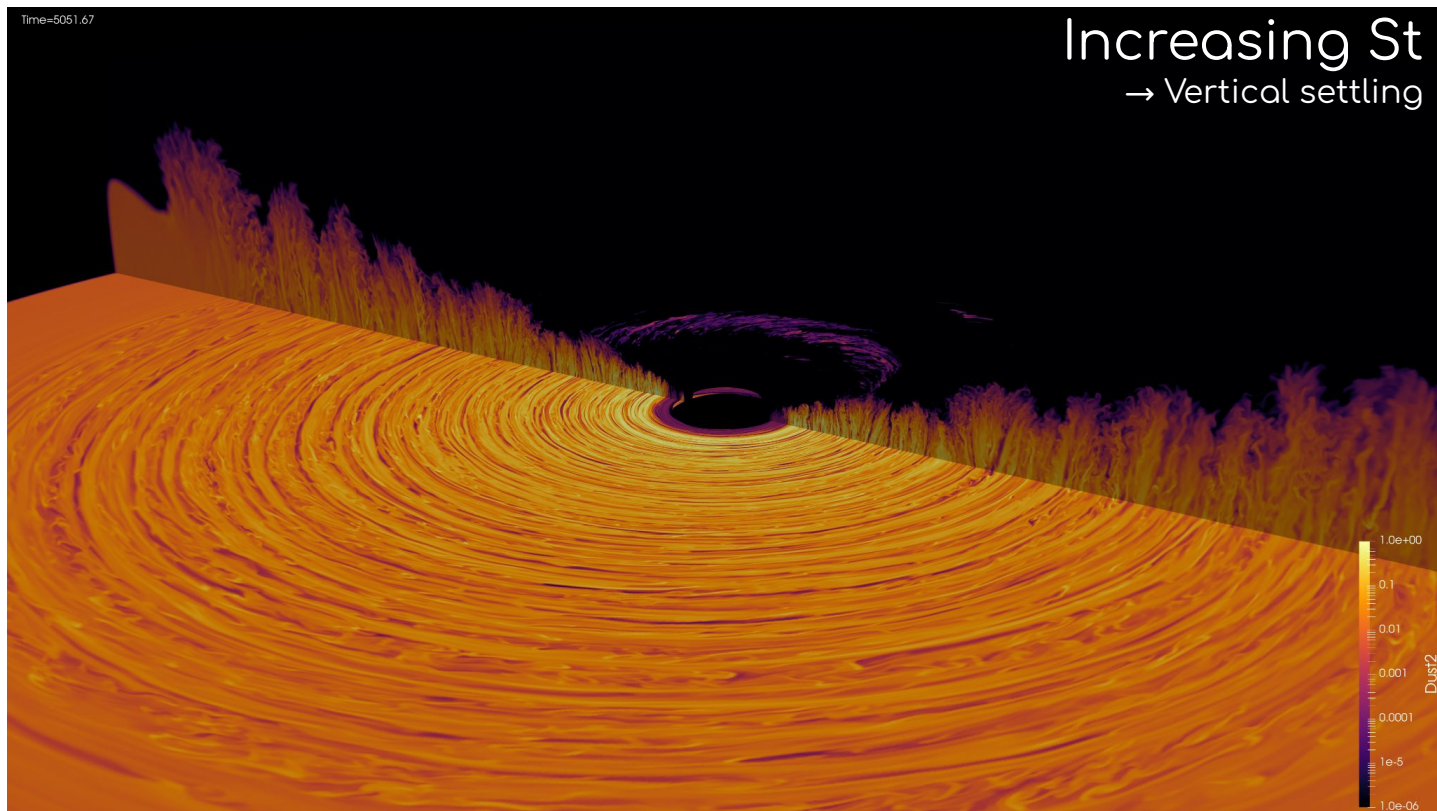
*Vertical Shear Instability with dust*





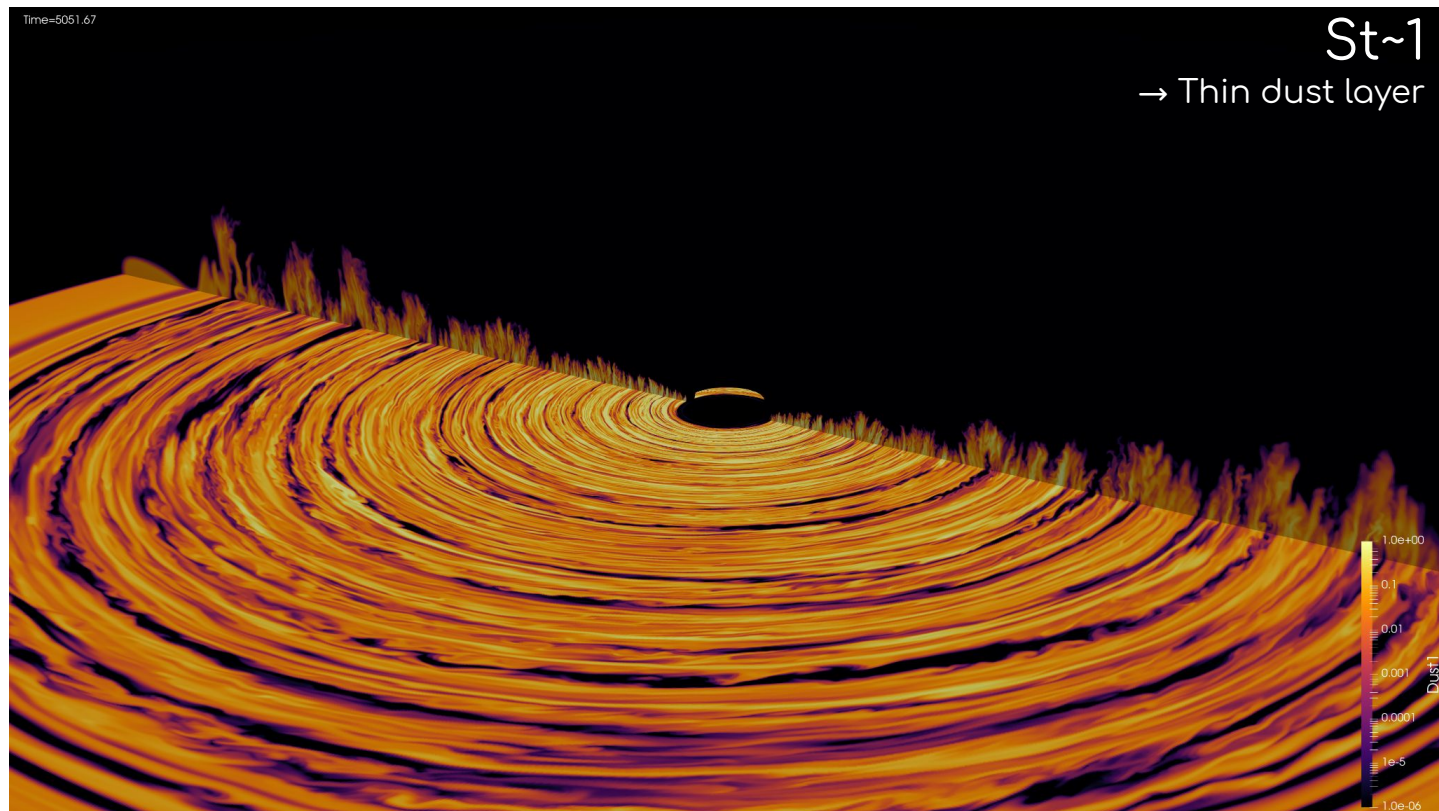
# FIRST RESULTS WITH IDEFIX

*Vertical Shear Instability with dust*



# FIRST RESULTS WITH IDEFIX

*Vertical Shear Instability with dust*





# ROUND TABLE

## *Prep*

Victor

- userdef resistive simulations + userdef braginskii
- wish: variable gamma OR modify the current Braginskii implementation

Jean

- 3D spherical compressible MHD + Braginskii, varying the thermal diffusivity
- wish: go to the pole ?

François

- ecology class, import matrix for interactions from python package to idfix

Geoff

- eurohpc-ju.europa.eu
- nonos on 2.2 To vtk file ?

GWF/Thomas

- wish: implicit scheme for the drag ?

Mario

- wish: multiple star system, go to the pole ?

Hossam

- wish: dust solver → crashes : add a flat reconstruction when dust\_density < 0
- wish: dust diffusion, dust particles ?

# ROUND TABLE

*Prep*

Thomas

- wish: more dust tests, what are tests and what are true setups, vtk slices per field

Geoff

- wish: priorities (dev ? doc ? formation ?), AMR, GR, PIC, incompressible method (spectral) ?

GWF: test → vtk\_slices (readable with nonos ?) + dust with MHD wind + LookupTable/DumpImage  
for restarting with .npy file could be possible

# Parameters in a dusty run

In your idefix.ini

[Dust]

nSpecies

drag

drag\_feedback

# Parameters in a dusty run

In your idfix.ini

[Dust]

nSpecies

*integer n, number of dust species*

drag

drag\_feedback

# Parameters in a dusty run

In your idfix.ini

[Dust]

nSpecies

*integer n, number of dust species*

drag

*gas→dust drag law*

drag\_feedback



# Parameters in a dusty run

In your idfix.ini

[Dust]

nSpecies

*integer n, number of dust species*

drag

*gas→dust drag law*

drag\_feedback

*is there dust→gas drag ?*

# Parameters in a dusty run

In your idfix.ini

[Dust]

nSpecies

*integer n, number of dust species*

drag

*gas→dust drag law*

drag\_feedback

*is there dust→gas drag ? yes or no*

# Drag laws

$$\frac{\partial(\rho_{d_i})}{\partial t} + \vec{\nabla} \cdot (\rho_{d_i} \vec{v}_{d_i}) = 0$$

i integer,  $1 \leq i \leq n$

$$\frac{\partial(\rho_{d_i} \vec{v}_{d_i})}{\partial t} + \vec{\nabla} \cdot (\rho_{d_i} \vec{v}_{d_i} \otimes \vec{v}_{d_i}) = \rho_{d_i} \vec{g} + \vec{f}_{g \rightarrow d_i}$$

# Drag laws

$$\frac{\partial(\rho_{d_i})}{\partial t} + \vec{\nabla} \cdot (\rho_{d_i} \vec{v}_{d_i}) = 0$$

i integer,  $1 \leq i \leq n$

$$\frac{\partial(\rho_{d_i} \vec{v}_{d_i})}{\partial t} + \vec{\nabla} \cdot (\rho_{d_i} \vec{v}_{d_i} \otimes \vec{v}_{d_i}) = \rho_{d_i} \vec{g} + \vec{f}_{g \rightarrow d_i}$$

Where  $\vec{f}_{g \rightarrow d_i} = \gamma_i \rho_{d_i} \rho (\vec{v}_g - \vec{v}_{d_i})$

Four possible drag laws: **gamma** fixes  $\gamma_i$

fixed drag parameter



# Drag laws

$$\frac{\partial(\rho_{d_i})}{\partial t} + \vec{\nabla} \cdot (\rho_{d_i} \vec{v}_{d_i}) = 0$$

i integer,  $1 \leq i \leq n$

$$\frac{\partial(\rho_{d_i} \vec{v}_{d_i})}{\partial t} + \vec{\nabla} \cdot (\rho_{d_i} \vec{v}_{d_i} \otimes \vec{v}_{d_i}) = \rho_{d_i} \vec{g} + \vec{f}_{g \rightarrow d_i}$$

$$\gamma_i = \frac{1}{\rho t_i}$$

Where  $\vec{f}_{g \rightarrow d_i} = \gamma_i \rho_{d_i} \rho (\vec{v}_g - \vec{v}_{d_i}) = \frac{\rho_i}{t_i} (\vec{v}_g - \vec{v}_{d_i})$

Four possible drag laws: **gamma** fixes  $\gamma_i$ , **tau** fixes  $t_i$

fixed drag parameter

fixed stopping time

# Drag laws

$$\frac{\partial(\rho_{d_i})}{\partial t} + \vec{\nabla} \cdot (\rho_{d_i} \vec{v}_{d_i}) = 0$$

i integer,  $1 \leq i \leq n$

$$\frac{\partial(\rho_{d_i} \vec{v}_{d_i})}{\partial t} + \vec{\nabla} \cdot (\rho_{d_i} \vec{v}_{d_i} \otimes \vec{v}_{d_i}) = \rho_{d_i} \vec{g} + \vec{f}_{g \rightarrow d_i}$$

Where  $\vec{f}_{g \rightarrow d_i} = \gamma_i \rho_{d_i} \rho (\vec{v}_g - \vec{v}_{d_i}) = \frac{\rho_i}{t_i} (\vec{v}_g - \vec{v}_{d_i}) = \frac{c_s \rho_{d_i} \rho}{\beta_i} (\vec{v}_g - \vec{v}_{d_i})$

$$\gamma_i = \frac{1}{\rho t_i}$$

$$t_i = \frac{\beta_i}{\rho c_s}$$

Four possible drag laws: **gamma** fixes  $\gamma_i$ , **tau** fixes  $t_i$ , **size** fixes  $\beta_i$

fixed drag parameter

fixed stopping time

Epstein or Stokes drag law with fixed:

- Dust density  $\rho_s$
- Dust size  $a$

Epstein:  $\beta_i = (\rho_s a)_i$

# Drag laws

$$\frac{\partial(\rho_{d_i})}{\partial t} + \vec{\nabla} \cdot (\rho_{d_i} \vec{v}_{d_i}) = 0$$

i integer,  $1 \leq i \leq n$

$$\frac{\partial(\rho_{d_i} \vec{v}_{d_i})}{\partial t} + \vec{\nabla} \cdot (\rho_{d_i} \vec{v}_{d_i} \otimes \vec{v}_{d_i}) = \rho_{d_i} \vec{g} + \vec{f}_{g \rightarrow d_i}$$

Where  $\vec{f}_{g \rightarrow d_i} = \gamma_i \rho_{d_i} \rho (\vec{v}_g - \vec{v}_{d_i}) = \frac{\rho_i}{t_i} (\vec{v}_g - \vec{v}_{d_i}) = \frac{c_s \rho_{d_i} \rho}{\beta_i} (\vec{v}_g - \vec{v}_{d_i})$

$$\gamma_i = \frac{1}{\rho t_i}$$

$$t_i = \frac{\beta_i}{\rho c_s}$$

Four possible drag laws: **gamma** fixes  $\gamma_i$ , **tau** fixes  $t_i$ , **size** fixes  $\beta_i$

fixed drag parameter

fixed stopping time

Epstein or Stokes drag law with fixed:

- Dust density  $\rho_s$
- Dust size  $a$

Epstein:  $\beta_i = (\rho_s a)_i$

See Thomas' presentation  
this afternoon!

# Drag laws

$$\frac{\partial(\rho_{d_i})}{\partial t} + \vec{\nabla} \cdot (\rho_{d_i} \vec{v}_{d_i}) = 0$$

i integer,  $1 \leq i \leq n$

$$\frac{\partial(\rho_{d_i} \vec{v}_{d_i})}{\partial t} + \vec{\nabla} \cdot (\rho_{d_i} \vec{v}_{d_i} \otimes \vec{v}_{d_i}) = \rho_{d_i} \vec{g} + \vec{f}_{g \rightarrow d_i}$$

Where  $\vec{f}_{g \rightarrow d_i} = \gamma_i \rho_{d_i} \rho (\vec{v}_g - \vec{v}_{d_i}) = \frac{\rho_i}{t_i} (\vec{v}_g - \vec{v}_{d_i}) = \frac{c_s \rho_{d_i} \rho}{\beta_i} (\vec{v}_g - \vec{v}_{d_i})$

$$\gamma_i = \frac{1}{\rho t_i}$$

$$t_i = \frac{\beta_i}{\rho c_s}$$

Four possible drag laws: **gamma** fixes  $\gamma_i$ , **tau** fixes  $t_i$ , **size** fixes  $\beta_i$  and **userdef** is whatever you like

fixed drag parameter

fixed stopping time

Epstein or Stokes drag law with fixed:

- Dust density  $\rho_s$
- Dust size  $a$

Epstein:  $\beta_i = (\rho_s a)_i$

See Thomas' presentation this afternoon!



# Drag laws

$$\frac{\partial(\rho_{d_i})}{\partial t} + \vec{\nabla} \cdot (\rho_{d_i} \vec{v}_{d_i}) = 0$$

i integer,  $1 \leq i \leq n$

$$\frac{\partial(\rho_{d_i} \vec{v}_{d_i})}{\partial t} + \vec{\nabla} \cdot (\rho_{d_i} \vec{v}_{d_i} \otimes \vec{v}_{d_i}) = \rho_{d_i} \vec{g} + \vec{f}_{g \rightarrow d_i}$$

Where  $\vec{f}_{g \rightarrow d_i} = \gamma_i \rho_{d_i} \rho (\vec{v}_g - \vec{v}_{d_i}) = \frac{\rho_i}{t_i} (\vec{v}_g - \vec{v}_{d_i}) = \frac{c_s \rho_{d_i} \rho}{\beta_i} (\vec{v}_g - \vec{v}_{d_i})$

$$\gamma_i = \frac{1}{\rho t_i}$$

$$t_i = \frac{\beta_i}{\rho c_s}$$

Four possible drag laws: **gamma** fixes  $\gamma_i$ , **tau** fixes  $t_i$ , **size** fixes  $\beta_i$  and **userdef** is whatever you like

fixed drag parameter

fixed stopping time

Epstein or Stokes drag law with fixed:

- Dust density  $\rho_s$
- Dust size  $a$

Epstein:  $\beta_i = (\rho_s a)_i$

See Thomas' presentation this afternoon!

Example for **gamma**

[Dust]

nSpecies

3

drag

gamma 1.0. 2.0 3.0

sets three dust species ( $1 \leq i \leq 3$ ) with drag laws  $\vec{f}_{g \rightarrow d_i} = i \rho_{d_i} \rho (\vec{v}_g - \vec{v}_{d_i})$

drag\_feedback

yes

# Example for userdef

In your setup.cpp

```
void MyDrag(DataBlock *data, real beta, IdefixArray3D<real> &gamma) {  
  
    //Compute the drag coefficient gamma from the input beta  
  
    idfix_for("MyDrag",0,data->np_tot[KDIR],0,data->np_tot[JDIR],0,data->np_tot[IDIR],  
    KOKKOS_LAMBDA (int k, int j, int i) {  
        gamma(k,j,i) = 1/(beta*data->hydro->Vc(RHO,k,j,i));  
    });  
}
```

# Example for userdef

In your setup.cpp

```
void MyDrag(DataBlock *data, real beta, IdefixArray3D<real> &gamma) {  
  
    //Compute the drag coefficient gamma from the input beta  
  
    idefix_for("MyDrag",0,data->np_tot[KDIR],0,data->np_tot[JDIR],0,data->np_tot[IDIR],  
        KOKKOS_LAMBDA (int k, int j, int i) {  
            gamma(k,j,i) = 1/(beta*data->hydro->Vc(RHO,k,j,i));  
        });  
}
```

Don't forget to enroll it !

```
Setup::Setup(Input &input, Grid &grid, DataBlock &data, Output &output){  
    //(...)  
    if(data.haveDust) {  
        int nSpecies = data.dust.size();  
        for(int n = 0 ; n < nSpecies ; n++) {  
            data.dust[n]->drag->EnrollUserDrag(&MyDrag);  
        }  
    }  
}
```

# Example for userdef

In your setup.cpp

```
void MyDrag(DataBlock *data, real beta, IdefixArray3D<real> &gamma) {  
  
    //Compute the drag coefficient gamma from the input beta  
  
    idefix_for("MyDrag",0,data->np_tot[KDIR],0,data->np_tot[JDIR],0,data->np_tot[IDIR],  
        KOKKOS_LAMBDA (int k, int j, int i) {  
        gamma(k,j,i) = 1/(beta*data->hydro->Vc(RHO,k,j,i));  
        });  
}
```

Don't forget to enroll it !

```
Setup::Setup(Input &input, Grid &grid, DataBlock &data, Output &output){  
    //(...)  
    if(data.haveDust) {  
        int nSpecies = data.dust.size();  
        for(int n = 0 ; n < nSpecies ; n++) {  
            data.dust[n]->drag->EnrollUserDrag(&MyDrag);  
        }  
    }  
}
```

With that drag

```
[Dust]  
nSpecies          1  
drag              userdef  1.0  
drag_feedback    yes
```

==

```
[Dust]  
nSpecies          1  
drag              tau   1.0  
drag_feedback    yes
```

# Drag feedback and CFL condition

Dust equation of motion:  $\frac{\partial(\rho_{d_i}\vec{v}_{d_i})}{\partial t} + \vec{\nabla} \cdot (\rho_{d_i}\vec{v}_{d_i} \otimes \vec{v}_{d_i}) = \rho_{d_i}\vec{g} + \vec{f}_{g \rightarrow d_i}$

# Drag feedback and CFL condition

Dust equation of motion: 
$$\frac{\partial(\rho_{d_i}\vec{v}_{d_i})}{\partial t} + \vec{\nabla} \cdot (\rho_{d_i}\vec{v}_{d_i} \otimes \vec{v}_{d_i}) = \rho_{d_i}\vec{g} + \vec{f}_{g \rightarrow d_i}$$

With feedback, additional forces on the gas:

$$\vec{f}_{d_i \rightarrow g} = -\vec{f}_{g \rightarrow d_i} = -\gamma_i \rho_{d_i} \rho (\vec{v}_g - \vec{v}_{d_i}) \text{ for each dust species } i$$

# Drag feedback and CFL condition

Dust equation of motion:  $\frac{\partial(\rho_{d_i}\vec{v}_{d_i})}{\partial t} + \vec{\nabla} \cdot (\rho_{d_i}\vec{v}_{d_i} \otimes \vec{v}_{d_i}) = \rho_{d_i}\vec{g} + \vec{f}_{g \rightarrow d_i}$

With feedback, additional forces on the gas:

$$\vec{f}_{d_i \rightarrow g} = -\vec{f}_{g \rightarrow d_i} = -\gamma_i \rho_{d_i} \rho (\vec{v}_g - \vec{v}_{d_i}) \text{ for each dust species } i$$

Without dust or feedback

$$dt = \sigma_{\text{CFL}} \left( \max_{\mathcal{V}} \left[ \sum_d \frac{c_{\text{max},d}}{d\ell_d} + \frac{2\eta_{\text{max}}}{d\ell_d^2} \right] \right)^{-1}$$

*Lesur+(2021)*

# Drag feedback and CFL condition

Dust equation of motion:  $\frac{\partial(\rho_{d_i}\vec{v}_{d_i})}{\partial t} + \vec{\nabla} \cdot (\rho_{d_i}\vec{v}_{d_i} \otimes \vec{v}_{d_i}) = \rho_{d_i}\vec{g} + \vec{f}_{g \rightarrow d_i}$

With feedback, additional forces on the gas:

$$\vec{f}_{d_i \rightarrow g} = -\vec{f}_{g \rightarrow d_i} = -\gamma_i \rho_{d_i} \rho (\vec{v}_g - \vec{v}_{d_i}) \text{ for each dust species } i$$

Without dust or feedback

$$dt = \sigma_{\text{CFL}} \left( \max_{\mathcal{V}} \left[ \sum_d \frac{c_{\text{max},d}}{d\ell_d} + \frac{2\eta_{\text{max}}}{d\ell_d^2} \right] \right)^{-1}$$

*Lesur+(2021)*

With feedback

$$dt < \min \left( \frac{1}{\sum_i \gamma_i (\rho_i + \rho)} \right)$$

*Idefix userguide*



**Thank you**