Initiator Documentation

Release 3.0

/ELSA/MU-10019/V3.0

Dec 05, 2019

CONTENTS

CHAPTER

ONE

PREAMBLE

Initiator module works on arrays (as defined in Converter) or on CGNS/python trees (pyTrees) containing grid information (coordinates must be defined).

This module is part of Cassiopee, a free open-source pre- and post-processor for CFD simulations.

For use with the array interface, you have to import Initiator module:

```
import Initiator as I
```

For use with the pyTree interface:

```
import Initiator.PyTree as I
```

CHAPTER

TWO

LIST OF FUNCTIONS

– CFD field initialisations

initConst(a[, adim, MInf, alphaZ, alphaY, ReInf])	Init a by a constant field.
initLamb(a[, position, Gamma, MInf])	Init a with a Lamb vortex of intensity Gamma and position (x0,y0).
initVisbal(a[, position, Gamma, MInf])	Init a with a Visbal vortex of intensity Gamma and position (x0,y0).
initScully(a[, position, Gamma, coreRadius, ...])	Init a with a Scully vortex of intensity Gamma, core radius coreRadius and position (x0,y0).
initYee(a[, position, Gamma, MInf])	Init a with a Yee vortex of intensity Gamma and position (x0,y0).
overlayField(a1, a2[, MInf])	Overlay the field of a1 and a2.

– Adimensioning

Adim.adim1([MInf, alphaZ, alphaY, ReInf, ...])	Return a state corresponding to adimensioning by density, sound speed and temperature.
Adim.adim2([MInf, alphaZ, alphaY, ReInf, ...])	Return a state corresponding to adimensioning by density, velocity and temperature.
Adim.adim3([MInf, alphaZ, alphaY, ReInf, ...])	Return a state corresponding to adimensioning by density, sound speed and temperature.
Adim.dim1([UInf, TInf, PInf, LInf, alphaZ, ...])	Return a dimensioned state specifying velocity, temperature and pressure.
Adim.dim2([UInf, TInf, RoInf, LInf, alphaZ, ...])	Return a dimensioned state specifying velocity, temperature and density.
Adim.dim3([UInf, PInf, RoInf, LInf, alphaZ, ...])	Return a dimensioned state specifying velocity, pressure and density.

CHAPTER

THREE

CONTENTS

3.1 CFD field initialisations

The input data defines a grid or a set of grids on which the solution has to be initialized.

The created field variables are the variables defined in the input data. If the five conservative variables are not present, then the default output variables are the coordinates and the conservative variables.

`Initiator.`**`initConst`**(*a*, *adim='adim1'*, *MInf=None*, *alphaZ=0.*, *alphaY=0.*, *ReInf=1.e8*, *loc='nodes'*)

Initialization by a constant field, given Mach number, incident flow angles and Reynolds number.

Exists also as in place version (_initConst) that modifies a and returns None.

Parameters

- **a** (`[array, list of arrays] or [pyTree, base, zone, list of zones]`) – input data
- **adim** (`string`) – Name of adim ('adim1', 'adim2', 'dim1', ...) - see Adimensioning section
- **MInf** (`float`) – freestream Mach number
- **alphaZ** (`float`) – Angle with respect to (0,Z) axe (in degrees)
- **alphaY** (`float`) – Angle with respect to (0,Y) axe (in degrees)
- **ReInf** (`float`) – Reynolds Number
- **loc** (`string`) – created field localisation ('nodes' or 'centers') - only for pyTree interface

Return type Identical to a

Example of use:

- Constant field initialization (array):

```
# - initConst (array) -
import Converter as C
import Generator as G
import Initiator as I

NI = 100; NJ = 100
HI = 50./(NI-1); HJ = 50./(NJ-1)
a = G.cart((0.,0.,0.), (HI,HJ,1.), (NI,NJ,2))
a = I.initConst(a, MInf=0.8)
C.convertArrays2File([a], 'out.plt')
```

- Constant field initialization (pyTree):

```
# - initConst (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Initiator.PyTree as I

NI = 100; NJ = 100
HI = 50./(NI-1); HJ = 50./(NJ-1)
a = G.cart((0.,0.,0.), (HI,HJ,1.), (NI,NJ,2))
a = I.initConst(a, MInf=0.8, loc='centers')
C.convertPyTree2File(a, 'out.cgns')
```

Initiator.**initLamb**(*a*, *(x0, y0)*, *Gamma=2.*, *MInf=0.5*, *loc='nodes'*)

Initialization of conservative variables by a 2D Lamb vortex at position (x0,y0), intensity Gamma and infinite Mach number MInf.

Exists also as in place version (_initLamb) that modifies a and returns None.

Parameters

- **a** (`[array, list of arrays] or [pyTree, base, zone, list of zones]`) – Input data
- **Gamma** (`float`) – Intensity of vortex
- **MInf** (`float`) – Mach number
- **loc** – created field localisation ('nodes' or 'centers') - only for pyTree interface

Return type Identical to a

Example of use:

- Field initialization with a lamb vortex (array):

```
# - initLamb (array) -
import Converter as C
import Generator as G
import Initiator as I

NI = 100; NJ = 100
HI = 50./(NI-1); HJ = 50./(NJ-1)
a = G.cart((0.,0.,0.), (HI,HJ,1.), (NI,NJ,2))
a = I.initLamb(a, position=(25.,25.), Gamma=2., MInf=0.8)
C.convertArrays2File(a, 'out.plt')
```

- Field initialization with a lamb vortex (pyTree):

```
# - initLamb (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Initiator.PyTree as I

NI = 100; NJ = 100
HI = 50./(NI-1); HJ = 50./(NJ-1)
a = G.cart((0.,0.,0.), (HI,HJ,1.), (NI,NJ,2))
a = I.initLamb(a, position=(7.,7.), Gamma=2., MInf=0.8, loc='centers')
C.convertPyTree2File(a, "out.cgns")
```

Initiator.**initVisbal**(*a*, *(x0, y0)*, *Gamma=2.*, *MInf=0.5*, *loc='nodes'*)

Initialization of conservative variables by a 2D Visbal vortex at position (x0,y0), intensity Gamma and infinite Mach number MInf.

Exists also as in place version (_initVisbal) that modifies a and returns None.

Parameters

- **a** ([array, list of arrays] or [pyTree, base, zone, list of zones]) – Input data
- **Gamma** (float) – Intensity of vortex
- **MInf** (float) – Mach number
- **loc** – field localisation ('nodes' or 'centers') - only for pyTree interface

Return type Identical to a

Example of use:

- Field initialization with a visbal vortex (array):

```
# - initVisbal (array) -
import Converter as C
import Generator as G
import Initiator as I

NI = 100; NJ = 100
HI = 50./(NI-1); HJ = 50./(NJ-1)
a = G.cart( (0.,0.,0.), (HI,HJ,1.), (NI,NJ,2))
a = I.initVisbal(a,position=(25.,25.), Gamma=2., MInf=0.8)
C.convertArrays2File([a], 'out.plt')
```

- Field initialization with a visbal vortex (pyTree):

```
# - initVisbal (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Initiator.PyTree as I

NI = 100; NJ = 100
HI = 50./(NI-1); HJ = 50./(NJ-1)
MachInf = 0.8

a = G.cart((0.,0.,0.), (HI,HJ,1.), (NI,NJ,2))
z = I.initVisbal(a, (3.,3.), 2., MachInf, loc='centers')
C.convertPyTree2File(z, 'out.cgns')
```

Initiator.**initScully**(*a, (x0, y0), Gamma=2., coreRadius=1., MInf=0.5, loc='nodes'*)

Initialization of conservative variables by a 2D Scully vortex at position (x0,y0), intensity Gamma and infinite Mach number MInf.

Exists also as in place version (_initScully) that modifies a and returns None.

Parameters

- **a** (`[array, list of arrays] or [pyTree, base, zone, list of zones]`) – Input data
- **Gamma** (`float`) – Intensity of vortex
- **coreRadius** (`float`) – radius of vortex core
- **MInf** (`float`) – Mach number
- **loc** – field localisation ('nodes' or 'centers') - only for pyTree interface

Return type Identical to a

Example of use:

- Field initialization with a scully vortex (array):

```
# - initScully (array) -
import Generator as G
import Converter as C
import Initiator as I

NI = 200; NJ = 200
HI = 1./(NI-1); HJ = 1./(NJ-1)
a = G.cart((0.,0.,0.), (HI,HJ,1.), (NI,NJ,2))
a = I.initScully(a, (0.5,0.5), -0.2, 0.05, 0.8, 0)
C.convertArrays2File([a], "out.plt")
```

- Field initialization with a scully vortex (pyTree):

```
# - initScully (pyTree) -
import Generator.PyTree as G
import Converter.PyTree as C
import Initiator.PyTree as I

NI = 200; NJ = 200
HI = 1./(NI-1); HJ = 1./(NJ-1)
a = G.cart((0.,0.,0.), (HI,HJ,1.), (NI,NJ,2))
a = I.initScully(a, (0.5,0.5), -0.2, 0.05, 0.8, 0, loc='centers')
C.convertPyTree2File(a, 'out.cgns')
```

Initiator.**initYee**(*a*, *(x0, y0)*, *Gamma=2.*, *MInf=0.5*, *loc='nodes'*)

Initialization of conservative variables by a 2D Yee vortex at position (x0,y0), intensity Gamma and infinite Mach number MInf.

Exists also as in place version (_initYee) that modifies a and returns None.

Parameters

- **a** ([array, list of arrays] or [pyTree, base, zone, list of zones]) – Input data
- **Gamma** (float) – Intensity of vortex
- **MInf** (float) – Mach number
- **loc** – field localisation ('nodes' or 'centers') - only for pyTree interface

Return type Identical to a

Example of use:

- Field initialization with a Yee vortex (array):

```
# - initYee (array) -
import Converter as C
import Generator as G
import Initiator as I

NI = 100; NJ = 100
HI = 50./(NI-1); HJ = 50./(NJ-1)
a = G.cart( (0.,0.,0.), (HI,HJ,1.), (NI,NJ,2))
a = I.initYee(a, position=(25.,25.), Gamma=2., MInf=0.8)
C.convertArrays2File([a], 'out.plt')
```

- Field initialization with a Yee vortex (pyTree):

```
# - initYee (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Initiator.PyTree as I

NI = 100; NJ = 100
HI = 50./(NI-1); HJ = 50./(NJ-1)
MachInf = 0.8

a = G.cart((0.,0.,0.), (HI,HJ,1.), (NI,NJ,2))
z = I.initYee(a, (3.,3.), 2., MachInf, loc='centers')
C.convertPyTree2File(z, 'out.cgns')
```

Initiator.**overlayField**(*a*, *b*, *MInf=0.5*, *loc='nodes'*)

Overlay the field of two solutions defined on the same grid (defined in a and b). Only density, velocity and energy stagnation density are overlaid. Both fields must use the same adimensioning, which corresponds to the same infinite Mach number MInf.

Exists also as in place version (_overlayField) that modifies a and returns None.

Parameters

- **a** ([array, list of arrays] or [pyTree, base, zone, list of zones]) – First input data
- **b** ([array, list of arrays] or [pyTree, base, zone, list of zones]) – Second input data
- **MInf** (float) – Mach number
- **loc** – field localisation ('nodes' or 'centers') - only for pyTree interface

Return type Identical to a

Example of use:

- Field initialization by overlaying two solutions (array):

```
# - overlayField (array) -
import Converter as C
import Generator as G
import Initiator as I

NI = 100; NJ = 100
HI = 50./(NI-1); HJ = 50./(NJ-1)
MachInf = 0.8

a = G.cart((0.,0.,0.), (HI,HJ,1.), (NI,NJ,2))
ac = I.initVisbal(a, (3.,3.), 2., MachInf)
ac2 = I.initVisbal(a, (20.,3.), 2., MachInf)
an = I.overlayField(ac, ac2, MachInf)
C.convertArrays2File([an], "out.plt")
```

- Field initialization by overlaying two solutions (pyTree):

```
# - overlayField (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Initiator.PyTree as I

NI = 100; NJ = 100
HI = 50./(NI-1); HJ = 50./(NJ-1)
MachInf = 0.8
a = G.cart((0.,0.,0.), (HI,HJ,1.), (NI,NJ,2))
z1 = I.initVisbal(a, (3.,3.), 2., MachInf, loc='centers'); z1[0]='cart1'
z2 = I.initVisbal(a, (20.,3.), 2., MachInf, loc='centers'); z2[0]='cart2'
zn = I.overlayField(z1, z2, MachInf, loc='centers')
C.convertPyTree2File(zn, 'out.cgns')
```

3.2 Adimensioning

Initiator.Adim.**adim1**(*MInf=0.5*, *alphaZ=0.*, *alphaY=0.*, *ReInf=1.e8*, *MutSMuInf=0.2*, *TurbLevelInf=1.e-4*)

Return a reference state corresponding to an adimensioning by density, sound speed and temperature. Returned state is adimensioned. When using this state, the mesh must also be adimensioned.

Parameters

- **MInf** (float) – Mach number
- **alphaZ** (float) – Angle with respect to (0,Z) axe (in degrees)
- **alphaY** (float) – Angle with respect to (0,Y) axe (in degrees)
- **ReInf** (float) – Reynolds Number
- **MutSMuInf** (float) – ratio of mut/mu (turbulence viscosity/molecular viscosity) at infinity
- **TurbLevelInf** – turbulence level at infinity

Returns [RoInf, RouInf, RovInf, RowInf, RoEInf, PInf, TInf, cvInf, MInf, ReInf, Cs, Gamma, RokInf, RoomegaInf, RonutildeInf, Mus, Cs, Ts, Pr]

Return type list

Example of use:

- Get adimensioned state:

```
# - adim1 -
import Initiator.Adim as Adim
state = Adim.adim1(MInf=0.8, alphaZ=1., ReInf=1.e6); print(state)
#>> [1.0, 0.799878156125113, 0.01396192514982681, 0.0, 2.1057142857142863,
#>> 0.7142857142857143, 1.0, 1.7857142857142863, 0.8, 1000000.0,
#>> 0.3831337844872463, 1.4, 8e-09, 0.049999999999999996,
#>> 1.6000000000000003e-07, 8.000000000000001e-07, 0.3831337844872463, 1.0, 0.
↪70951]
```

Note: New in version 2.5

Initiator.Adim.**adim2**(*MInf=0.5*, *alphaZ=0.*, *alphaY=0.*, *ReInf=1.e8*, *MutSMuInf=0.2*, *TurbLevelInf=1.e-4*)

Return a reference state corresponding to an adimensioning by density, fluid velocity and temperature. Returned state is adimensioned. When using this state, the mesh must also be adimensioned.

Parameters

- **MInf** (float) – Mach number
- **alphaZ** (float) – Angle with respect to (0,Z) axe (in degrees)
- **alphaY** (float) – Angle with respect to (0,Y) axe (in degrees)
- **ReInf** (float) – Reynolds Number

- **MutSMuInf** (float) – ratio of mut/mu (turbulence viscosity/molecular viscosity) at infinity
- **TurbLevelInf** – turbulence level at infinity

Return type [RoInf, RouInf, RovInf, RowInf, RoEInf, PInf, TInf, cvInf, MInf, ReInf, Cs, Gamma, RokInf, RoomegaInf, RonutildeInf, Mus, Cs, Ts, Pr]

Example of use:

- Get adimensioned state 2:

```
# - adim2 -
import Initiator.Adim as Adim
state = Adim.adim2(MInf=0.8, alphaZ=1., ReInf=1.e6); print(state)
#>> [1.0, 0.9998476951563913, 0.01745240643728351, 0.0, 3.290178571428572,
#>> 1.1160714285714286, 1.0, 2.790178571428572, 0.8, 1000000.0,
#>> 0.3831337844872463, 1.4, 1e-08, 0.05, 2e-07, 1e-06, 0.3831337844872463,
#>> 1.0, 0.70951]
```

Note: New in version 2.5

Initiator.Adim.**adim3**(*MInf=0.5*, *alphaZ=0.*, *alphaY=0.*, *ReInf=1.e8*, *LInf=1.*, *MutSMuInf=0.2*, *TurbLevelInf=1.e-4*)

Return a reference state corresponding to an adimensioning by density, sound speed and temperature. Returned state is adimensioned. When using this state, the mesh doesn't need to be adimensioned and has a characteritic length of LInf.

Parameters

- **MInf** (float) – Mach number
- **alphaZ** (float) – Angle with respect to (0,Z) axe (in degrees)
- **alphaY** (float) – Angle with respect to (0,Y) axe (in degrees)
- **ReInf** (float) – Reynolds Number
- **LInf** (float) – Characteristic length of mesh (on which ReInf is based)
- **MutSMuInf** (float) – ratio of mut/mu (turbulence viscosity/molecular viscosity) at infinity
- **TurbLevelInf** – turbulence level at infinity

Return type [RoInf, RouInf, RovInf, RowInf, RoEInf, PInf, TInf, cvInf, MInf, ReInf, Cs, Gamma, RokInf, RoomegaInf, RonutildeInf, Mus, Cs, Ts, Pr]

Example of use:

- Get adimensioned state 3:

```
# - adim3 -
import Initiator.Adim as Adim
state = Adim.adim3(MInf=0.8, alphaZ=1., LInf=2.5, ReInf=1.e6); print(state)
#>> [1.0, 0.799878156125113, 0.01396192514982681, 0.0, 2.1057142857142863,
#>> 0.7142857142857143, 1.0, 1.7857142857142863, 0.8, 1000000.0,
#>> 0.3831337844872463, 1.4, 8e-09, 0.02, 4e-07, 2e-06, 0.3831337844872463,
#>> 1.0, 0.70951]
```

Note: New in version 2.5

Initiator.Adim.**dim1**(*UInf=2.7777*, *TInf=298.15*, *PInf=101325.*, *LInf=1.*, *alphaZ=0.*, *alphaY=0.*, *MutSMuInf=0.2*, *TurbLevelInf=1.e-4*)

Return a dimensioned reference state corresponding to dry air, considered as a perfect gaz. Returned a dimensioned state (USI).

Parameters

- **UInf** (float) – Fluid velocity in m/s
- **TInf** (float) – Temperature in K (0 degree=273.15K)
- **PInf** (float) – Pressure in Pa
- **LInf** (float) – Reference length (m). Usefull to compute Reynolds.
- **MutSMuInf** (float) – ratio of mut/mu (turbulence viscosity/molecular viscosity) at infinity
- **TurbLevelInf** – turbulence level at infinity

Return type [RoInf, RouInf, RovInf, RowInf, RoEInf, PInf, TInf, cvInf, MInf, ReInf, Cs, Gamma, RokInf, RoomegaInf, RonutildeInf, Mus, Cs, Ts, Pr]

Example of use:

- Get dimensioned state:

```
# - dim1 -
import Initiator.Adim as Adim
state = Adim.dim1(UInf=2.8, TInf=298., PInf=101325, LInf=12., alphaZ=1.);␣
↪print(state)
#>> [1.1845087092749074, 3.3161192480113266, 0.05788307678374978, 0.0,
#>> 253317.14327414043, 101325, 298.0, 717.6325, 0.00809104909572454,
#>> 2167112.2969719185, 110.4, 1.4, 3.316624385969741e-08, 0.009029634570716328,
↪
#>> 3.673043864616362e-06, 1.78938e-05, 110.4, 288.15, 0.7084595175093612]
```

Note: New in version 2.5

Initiator.Adim.**dim2**(*UInf=2.7777*, *TInf=298.15*, *RoInf=1.225*, *LInf=1.*, *alphaZ=0.*, *alphaY=0.*, *MutSMuInf=0.2*, *TurbLevelInf=1.e-4*)

Return a dimensioned reference state corresponding to dry air, considered as a perfect gaz. Only the input is different from dim1. Returned a dimensioned state (USI).

Parameters

- **UInf** (float) – Fluid velocity in m/s
- **TInf** (float) – Temperature in K (0 degree=273.15K)
- **RoInf** (float) – Input density (kg/m3)
- **LInf** (float) – Reference length (m). Usefull to compute Reynolds.
- **MutSMuInf** (float) – ratio of mut/mu (turbulence viscosity/molecular viscosity) at infinity
- **TurbLevelInf** – turbulence level at infinity

Return type [RoInf, RouInf, RovInf, RowInf, RoEInf, PInf, TInf, cvInf, MInf, ReInf, Cs, Gamma, RokInf, RoomegaInf, RonutildeInf, Mus, Cs, Ts, Pr]

Example of use:

- Get dimensioned state 2:

```
# - dim2 -
import Initiator.Adim as Adim
state = Adim.dim2(UInf=2.8, TInf=298., RoInf=1.2, LInf=12., alphaZ=1.);␣
↪print(state)
#>> [1.2, 3.3594882557254744, 0.058640085629272594, 0.0, 256630.08600000004,
#>> 102650.1528, 298.0, 717.6325000000002, 0.00809104909572454,
#>> 2195454.314521849, 110.4, 1.4, 3.3599999999999996e-08, 0.009147726310507706,
↪
#>> 3.673043864616362e-06, 1.78938e-05, 110.4, 288.15, 0.7084595175093613]
```

Note: New in version 2.5

Initiator.Adim.**dim3**(*UInf=2.7777*, *PInf=101325.*, *RoInf=1.225*, *LInf=1.*, *alphaZ=0.*, *alphaY=0.*, *MutSMuInf=0.2*, *TurbLevelInf=1.e-4*)

Return a dimensioned reference state corresponding to dry air, considered as a perfect gaz. Only the input is different from dim1. Returned a dimensioned state (USI).

Parameters

- **UInf** (float) – Fluid velocity in m/s
- **PInf** (float) – Pressure in Pa
- **RoInf** (float) – Input density (kg/m3)
- **LInf** (float) – Reference length (m). Usefull to compute Reynolds.
- **MutSMuInf** (float) – ratio of mut/mu (turbulence viscosity/molecular viscosity) at infinity
- **TurbLevelInf** – turbulence level at infinity

Return type [RoInf, RouInf, RovInf, RowInf, RoEInf, PInf, TInf, cvInf, MInf, ReInf, Cs, Gamma, RokInf, RoomegaInf, RonutildeInf, Mus, Cs, Ts, Pr]

Example of use:

- Get dimensioned state 3:

```
# - dim3 -
import Initiator.Adim as Adim
state = Adim.dim3(UInf=2.8, PInf=101325., RoInf=1.2, LInf=12., alphaZ=1.);␣
↪print(state)
#>> [1.1999999999999997, 3.3594882557254735, 0.05864008562927258, 0.0,
#>> 253317.20400000006, 101325.0, 294.152996136602, 717.6325000000002,
#>> 0.0081437855769486, 2217576.314799729, 110.4, 1.4, 3.359999999999999e-08,
#>> 0.009239901311665535, 3.6364024751627385e-06, 1.78938e-05, 110.4,
#>> 288.15, 0.7094696135544664]
```

Note: New in version 2.5

CHAPTER

FOUR

INDEX

- genindex
- modindex
- search