PySAGAS

Release 0.1.0

Kieran Mackle, Ingo Jahn

Feb 27, 2024

DOCUMENTATION

PySAGAS is a Python package for the generation of **S**ensitivity **A**pproximations for **G**eometric and **A**erodynamic **S**urface properties. It provides a computationally-efficient method for generating surface sensitivity approximations from existing flow solutions, to use in aerodynamic shape optimisation studies. The GIF below is an example of this, where a hypersonic waverider was optimised for maximum L/D at Mach 6.

To learn more about the theory behind *PySAGAS*, see the *theory documentation*. Otherwise, see the *Getting Started* guide, or have a look at the *examples*.

CHAPTER

ONE

GETTING STARTED

1.1 Prerequisites

1.1.1 ParaView

If you plan on using *PySAGAS* with Cart3D solutions, you should also install ParaView, or at least the ParaView Python bindings. If you are using an Anaconda environment, you can install the ParaView Python packages via Conda Forge using the command below:

```
conda install -c conda-forge paraview
```

If you already have ParaView installed, you can append the path to the binaries to the Python path using the snippet below.

```
import sys
sys.path.insert(0, "/opt/ParaView-5.6.2-MPI-Linux-64bit/bin")

# Now the import should work
import paraview
```

For more information on ParaView's Python packages, see the ParaView Wiki.

1.1.2 pyOptSparse

PySAGAS shape optimisation modules wrap around pyOptSparse to converge on optimal geometries. Follow the installation instructions, noting that special optimisers require custom builds.

If using an Anaconda environment, you can also install PyOptSparse from Conda forge:

```
conda install -c conda-forge pyoptsparse
```

1.1.3 PyMesh

Having PyMesh installed can greatly enhance the capabilities offered by PySAGAS. However, it can be difficult to install. Troubleshooting guide coming soon.

1.2 Installation

After installing the dependencies above, clone this repo to your machine.

```
git clone https://github.com/kieran-mackle/pysagas
```

Next, use pip to install the pysagas package from the repo you just cloned.

```
cd pysagas
python3 -m pip install .
```

CHAPTER

TWO

USING PYSAGAS

PySAGAS uses low-order methods to approximate sensitivities on the surface of aerodynamic geometries. The user must provide a nominal condition of the flow properties on the surface, along with the sensitivity of the geometric vertices to the design parameters. From here, one of *PySAGAS* sensitivity calculators can be used.

CHAPTER

THREE

PYSAGAS DEFINITIONS

This page defines all symbols, terms and data types used by *PySAGAS*.

3.1 Nomenclature

Some of the nomenclature used by *PySAGAS* is defined in the table below.

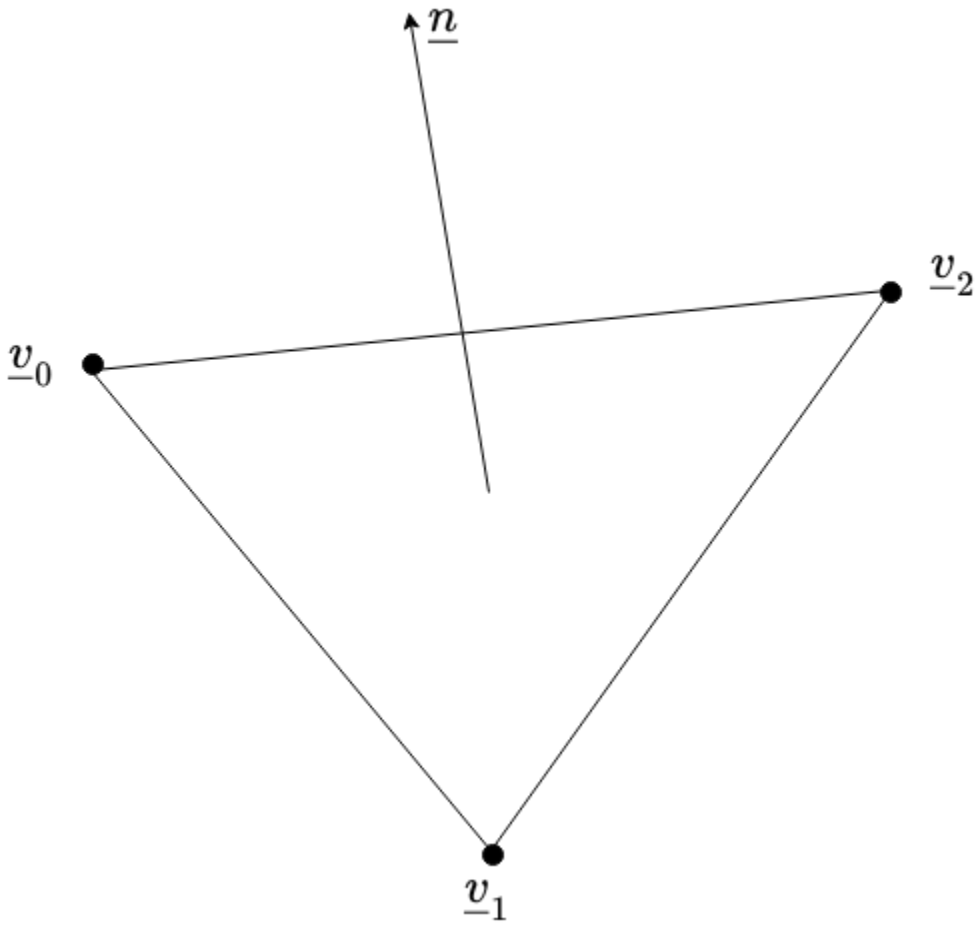
Symbol	Description
$\underline{\underline{\sqsubseteq}}$	Vertex vector
$\underline{\underline{\sqsubseteq}}$	Normal vector
$\underline{\theta}$	Design parameters vector
A	Area

3.2 Cell Definition

A `Cell` in *PySAGAS* is a triangular element, defined by three unique vectors pointing to the cell's vertices, $\underline{v}_0, \underline{v}_1$ and $\underline{v}_2$. These vertices form the corners of the triangular cell, and also form the face of the cell. This face has both an area A and a normal vector $\underline{n}$ associated with it. These properties are defined in the figure below.

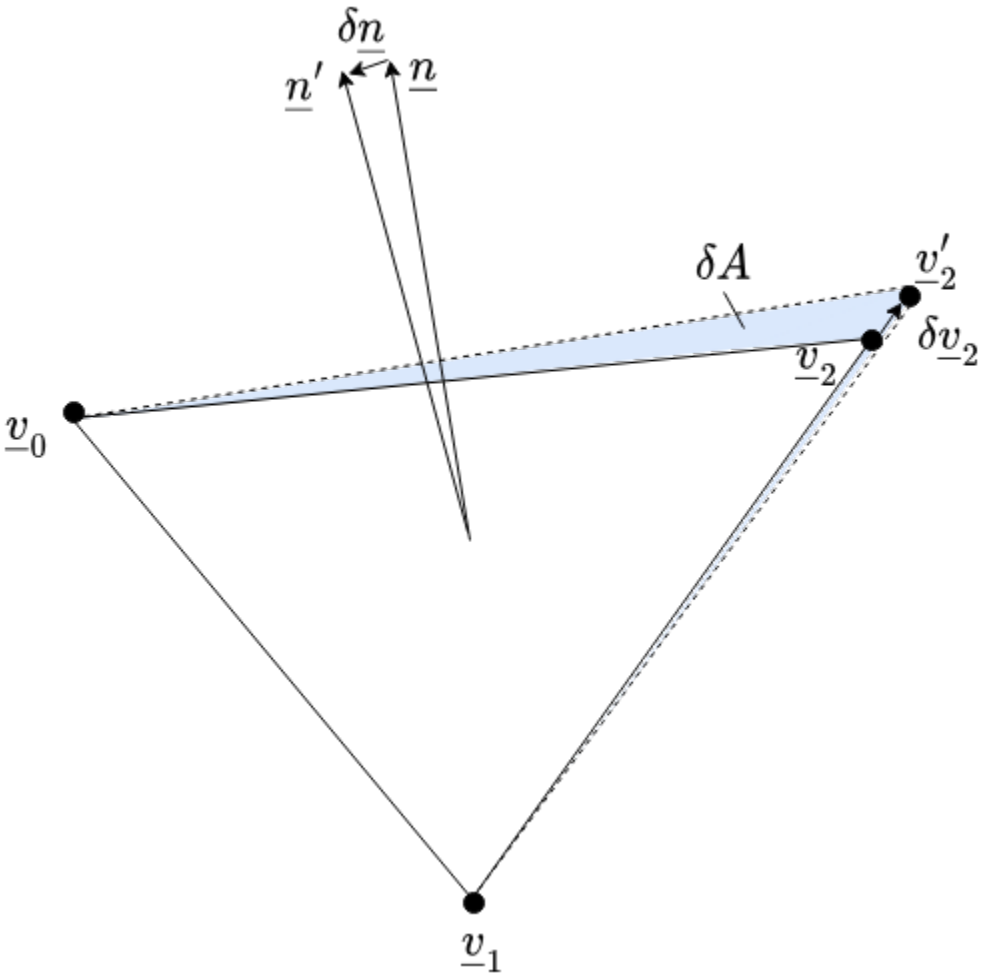
See also:

The Cell definition shown below is consistent with the `Cell` object.



3.3 Cell Normal and Area Sensitivities

Given the definition of a cell above, the sensitivities of the cell normal and the cell area to the cell's vertices can be determined. That is, $\frac{\partial \underline{n}}{\partial \underline{v}}$ and $\frac{dA}{\partial \underline{v}}$, respectively. The figure below exemplifies how a cell's normal vector and area changes with variations in one of its vertices, $\underline{v}_2$ to $\underline{v}_2'$.



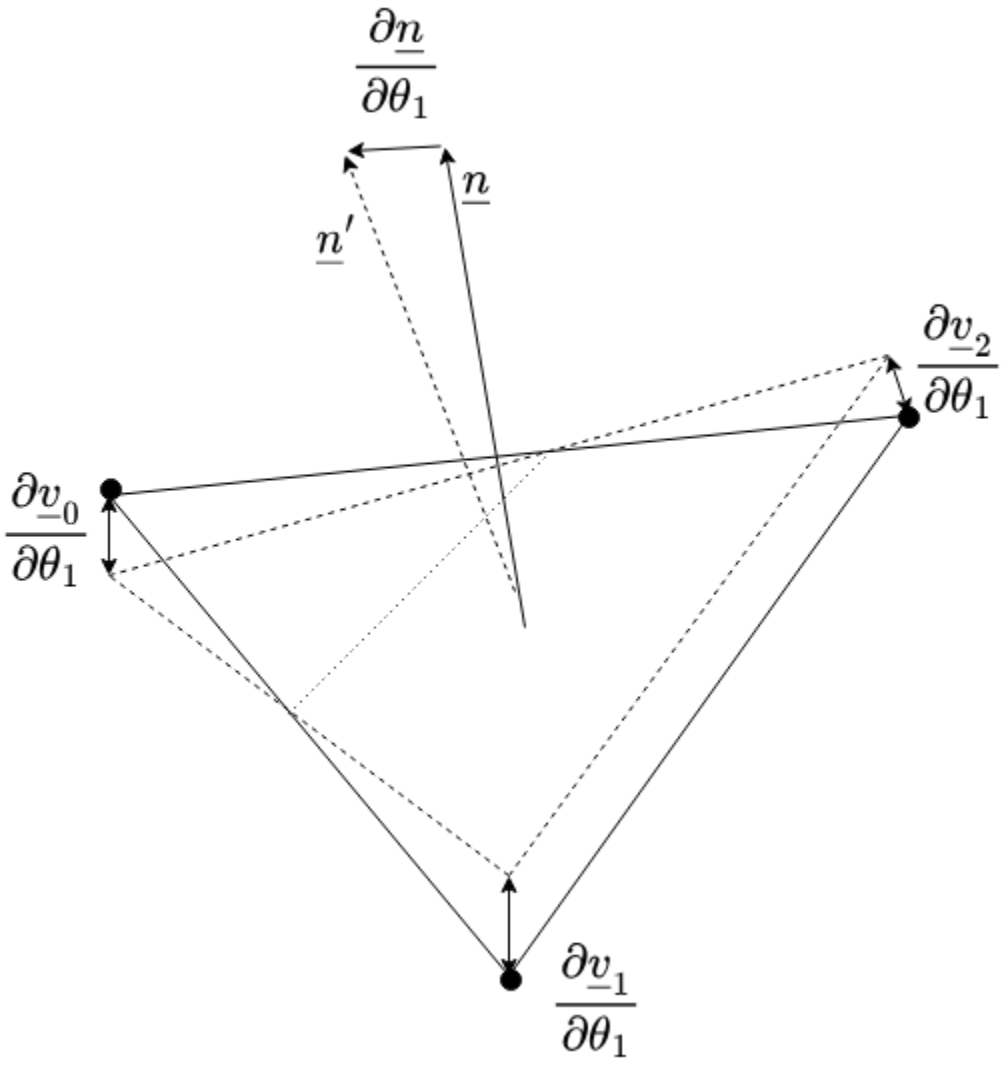
These sensitivities can be calculated using analytical derivitives.

See also:

The calculations of $\frac{\partial \underline{n}}{\partial \underline{v}}$ and $\frac{dA}{\partial \underline{v}}$ are implemented in *n_sensitivity()* and *A_sensitivity()*, respectively.

3.4 Geometric Parameter Sensitivities

Although users of *PySAGAS* are required to provide their own geometric parameter sensitivities, the figure below may be insightful. To be clear, a user must provide the sensitivity of each vertex defining a geometry to the design parameters, that is, $\frac{\partial \underline{v}}{\partial \underline{\theta}}$. The diagram in the figure below illustrates a cell's vertices changing as a result of a change in a parameter θ_1.



Given $\frac{\partial \underline{v}}{\partial \underline{\theta}}$, the sensitivity of both the cell normals and cell areas to the deisgn parameters can be calculated using the chain rule, as per the equations below.

$$\frac{\partial \underline{n}}{\partial \underline{\theta}} = \frac{\partial \underline{n}}{\partial \underline{v}} \frac{\partial \underline{v}}{\partial \underline{\theta}}$$

$$\frac{dA}{\partial \underline{\theta}} = \frac{dA}{\partial \underline{v}} \frac{\partial \underline{v}}{\partial \underline{\theta}}$$

CHAPTER

FOUR

APPROXIMATING SENSITIVITIES USING LOW-ORDER MODELS

The core idea underpinning *PySAGAS* is to use low-order, differentiable fluid models to evaluate the sensitivities of surface pressures to geometric parameters.

4.1 High-Level Overview of Approach

Tip: Refer to the *nomenclature* for definitions of symbols used here.

4.1.1 Vertex-Parameter Sensitivities

To make the link between *cell* data and geometric parameters, *PySAGAS* requires the user to provide the *vertex-parameter sensitivities*, $\partial \underline{v}/\partial \underline{\theta}$. In simple cases, such as the *inclined ramp example*, $\partial \underline{v}/\partial \underline{\theta}$ can be found manually. However, for more *complex geometries*, this is not possible. For this reason, a parametric geometry generation tool offering geometric differentiation is recommended. One such tool is *hypervehicle*.

4.1.2 Nominal Surface Properties

PySAGAS also requires the nominal surface properties of the geometry being considered. Namely, the local Mach number, pressure and temperature. This information can come from any source, making *PySAGAS* CFD-solver agnostic; whether it be *Oblique Shock Theory*, or *Cart3D*.

4.1.3 Pressure Sensitivities

Given the nominal surface properties, *PySAGAS* can calculate the sensitivity of local pressure to changes in the normal vector of a *cell*. Once these sensitivities are known, they can be linked to the geometric design parameters via application of the chain rule.

Let's start with *piston theory*. Surface pressure is given by the following expression, relative to the freestream properties, denoted by the ∞ subscript.

$$P = P_\infty \left(1 + \frac{\gamma - 1}{2} \frac{W}{a_\infty}\right)^{\frac{2\gamma}{\gamma - 1}}$$

In this expression, W is the downwash speed, a_∞ is the freestream speed of sound, and γ is the ratio of specific heats. By following the approach presented in Zhan (2009)[1], keeping only first-order terms and decomposing the downwash

[1] Zhan, W.-W., Ye, Z.-Y., and Zhang, C.-A., "Supersonic Flutter Analysis Based on a Local Piston Theory," AIAA Journal, Vol. 47, No. 10, 2009.

velocity, we obtain "local piston theory":

$$P = P_l + \rho_l \, a_l \, W$$

$$W = \underline{v}_l \cdot \delta\underline{n} + \underline{V}_b \cdot \underline{n}$$

$$\delta\underline{n} = \underline{n}_0 - \underline{n}$$

Here, the subscript l refers to the local surface conditions - i.e. the *nominal surface properties* as described above. In this model, we are only interested in the first term, $\underline{v}_l \cdot \delta\underline{n}$, as this relates to geometric deformation. The second term, $\underline{V}_b \cdot \underline{n}$, relates to vibration, and can therefore be discarded. We therefore reach an expression for the local pressure, relative to a cell's normal vector.

$$P = P_l + \rho_l \, a_l \, \underline{v}_l \cdot (\underline{n}_0 - \underline{n})$$

This expression can be differentiated with respect to the geometric parameters $\underline{p}$, yielding the sensitivity of pressure to parameters $\underline{p}$:

$$\frac{\partial P}{\partial \underline{\theta}} = \rho_l \, a_l \, \underline{v}_l \cdot \left(-\frac{\partial \underline{n}}{\partial \underline{\theta}} \right)$$

Of course, this expression requires $d\underline{n}/d\underline{p}$. Conveniently, this can be evaluated using the *vertex parameter sensitivities* and the *normal vector vertex sensitivities* as follows:

$$\frac{\partial \underline{n}}{\partial \underline{\theta}} = \frac{\partial \underline{n}}{\partial \underline{v}} \frac{\partial \underline{v}}{\partial \underline{\theta}}$$

4.1.4 Aerodynamic Sensitivities

With the sensitivity of pressure to parameters, $\partial P / \partial \underline{\theta}$, *PySAGAS* is able to calculate the sensitivity of the aerodynamic surface properties to the geometric design parameters (i.e. $\frac{\partial C_X}{\partial \underline{\theta}}$). First, note that the force acting on a given cell is the product of the pressure acting on that cell, and the cell's area. The total force acting on the geometry is therefore the sum of cell forces:

$$\underline{F}_{\text{i}} = P_i(\underline{\theta}) \, A_i(\underline{\theta}) \, \underline{n}_i(\underline{\theta})$$

$$\underline{F}_{\text{net}} = \sum_{i=0}^{N} \underline{F}_{\text{i}}$$

This expression can be differentiated with respect to the geometric parameters to yield the following.

$$\frac{\partial \underline{F}_{\text{net}}}{\partial \underline{\theta}_j} = \sum_{i=0}^{N} \left[\underbrace{\frac{\partial P_i}{\partial \underline{\theta}_j} A_i(\underline{\theta}_j) \underline{n}_i(\underline{\theta}_j)}_{\text{pressure sensitivity}} + \underbrace{P_i(\underline{\theta}_j) \frac{\partial A_i}{\partial \underline{\theta}_j} \underline{n}_i(\underline{\theta}_j)}_{\text{area sensitivity}} + \underbrace{P_i(\underline{\theta}_j) A_i(\underline{\theta}_j) \frac{\partial \underline{n}_i}{\partial \underline{\theta}_j}}_{\text{normal sensitivity}} \right]$$

These sensitivities can also be converted into non-dimensional force coefficients by dividing through by $\frac{1}{2} \, \rho_\infty \, V_\infty^2 \, A_{ref}$.

CHAPTER

FIVE

PUBLICATIONS OF PYSAGAS

5.1 2024 Publications

5.1.1 Developing a Co-Design Framework for Hypersonic Vehicle Aerodynamics and Trajectory

The extreme conditions at which hypersonic vehicles are required to operate necessitate a novel method of design, capable of producing vehicles that can perform across their entire mission trajectory. Traditional methods of design using trade studies of design parameters are incapable of capturing complex and non-linear subsystem interactions, which dominate hypersonic flight vehicles. Further challenges arise from the many competing design requirements, including packaging constraints, aerodynamic requirements, and flight path objectives. While traditional multi-objective design optimization methods attempt to address these challenges, they fail to account for the entirety of a mission's flight trajectory, and how the design parameters impact the objective at a system-level. This paper presents a novel and computationally tractable approach to co-design a vehicle's geometry and flight trajectory simultaneously, in order to obtain an optimal vehicle shape and flight path for a specified mission objective. That is, the vehicle shape is and trajectory are optimized in the same phase to obtain a the highest performing system solution in regards mission-level objectives. Importantly, the computational cost of the proposed method scales favourably with the number of vehicle design parameters, and is designed to reduce the number of computational fluid dynamic simulations. The approach is demonstrated by optimizing a hypersonic glider (waverider) parametrically defined using 16 variables to attain maximum range while also obeying an internal volume constraint. The optimal vehicle configuration obtained through the proposed co-designed framework exhibits an 11.6 % improvement over the nominal configuration.

5.1.2 Efficient and Flexible Methodology for the Aerodynamic Shape Optimization of Hypersonic Vehicle Concepts in a High-Dimensional Design Space

Aerodynamic shape of optimization of hypersonic vehicles parameterized by a large number of design variables (e.g. more than 10) is challenging due to the high computational cost of CFD evaluations. Gradient-based optimization methods are commonly used for this reason, along with the adjoint method applied to the governing equations of the flow to provide the optimizer with a search direction. Downsides of this approach include that it requires specialized CFD solvers, highly converged CFD simulations, and it can still be costly when optimizing against multiple performance objectives. Exploration studies in a high-dimensional design space are particularly beneficial during the conceptual design phases, and thus an optimization approach that is easy to implement, flexible, and quick to run is essential. In this paper, we develop an alternative approach to obtain the search direction for gradient-based aerodynamic shape optimization. The proposed approach is efficient, scalable, and CFD solver-agnostic. An approximate Jacobian is obtained by applying lower order aerodynamic models (such as Piston theory or Van Dyke's second order theory) locally, thus allowing pressure sensitivities to design parameters to be calculated without the need for further costly CFD simulations. The accuracy of this approach is demonstrated by estimating pressure sensitivities for a canonical shape, and the results are verified by comparison to finite difference of CFD solutions. We then demonstrate the

approach as a suitable means for shape optimization by optimizing a generic hypersonic waverider for maximum Lift-to-Drag ratio (/), whilst also respecting an internal volume constraint. Using an 8 CPU workstation, the optimized configuration (parameterized by 16 design variables) is obtained in little over 4 hours, with a 33% increase in /.

CHAPTER

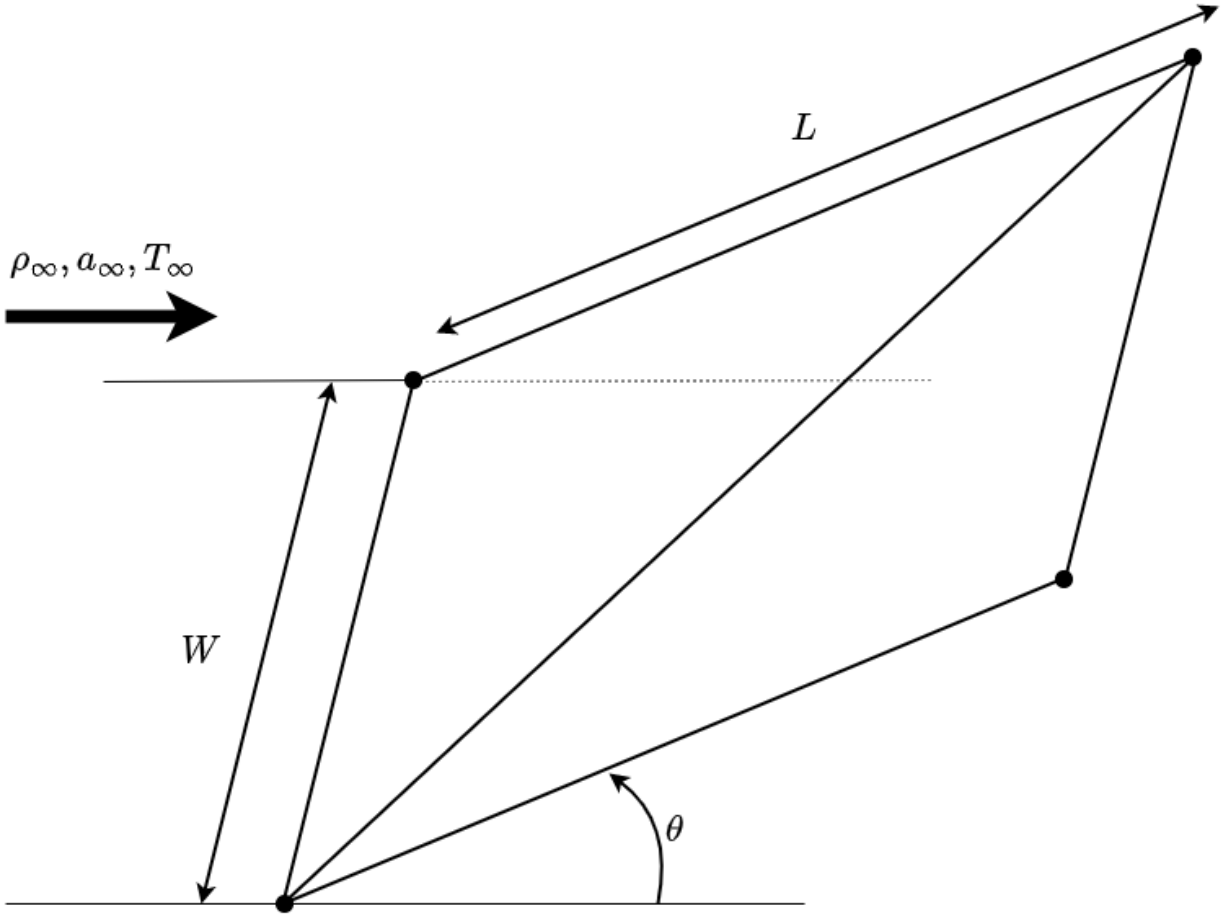
SIX

INCLINED RAMP EXAMPLE

This page covers a simple test case of an inclined ramp.

6.1 Problem Definition

The parameters of this problem are $\underline{\theta} = [\theta,\ L,\ W]$, shown diagramatically in the figure below. The geometry is built from two *cells*, sharing a common edge.



6.1.1 Free Stream Conditions

The freestream conditions are defined below.

$$\gamma = 1.4$$

$$Mach_{freestream} = 6$$

$$P_{freestream} = 700Pa$$

$$T_{freestream} = 70K$$

6.2 Analytical Solution

This problem can be solved analytically using simple isentropic flow relations and shock expansion theory. This solution will later serve as validation for the results obtained using *PySAGAS*.

6.2.1 Oblique Shock Relations

The nominal ramp conditions are solved using oblique shock wave theory.

$$\frac{p_2}{p_1} = 1 + \frac{2\gamma}{\gamma+1}(M_1^2 \sin^2\beta - 1)$$

$$\frac{\rho_2}{\rho_1} = \frac{(\gamma+1)M_1^2 \sin^2\beta}{(\gamma-1)M_1^2 \sin^2\beta + 2}$$

$$\frac{T_2}{T_1} = \frac{p_2}{p_1}\frac{\rho_1}{\rho_2}$$

$$M_2 = \frac{1}{\sin(\beta-\theta)}\sqrt{\frac{1+\frac{\gamma-1}{2}M_1^2\sin^2\beta}{\gamma M_1^2 \sin^2\beta - \frac{\gamma-1}{2}}}$$

The oblique shock angle is found by solving the $\theta - \beta - M$ equation, shown below for reference.

$$\tan\theta = 2\cot\beta\frac{M_1^2\sin^2\beta - 1}{M_1^2(\gamma + \cos 2\beta) + 2}$$

6.2.2 Calculating Ramp Conditions

Using the oblique shock relations presented above, the following conditions on the ramp surface can be calculated.

Property	Ramp Value
M_{ramp}	4.65
P_{ramp}	2567.41 Pa
T_{ramp}	107.89 K
ρ_{ramp}	0.0829 kg/m^3

6.2.3 Sensitivity Results

The force sensitivities for each ramp design parameter can be calculated using the method of finite differencing. Performing such a study produces the results shown in the table below.

$$F = P \times A \times \underline{n}_{ramp} \cdot \underline{u}$$

Parameter	dF_x/dp	dF_y/dp	dF_z/dp
θ	1.07	-3.11	0
L	8.92	-50.57	0
W	4.46	-25.28	0

6.3 PySAGAS Solution

Given the surface properties on the ramp calculated using the analytical solution, the parameter sensitivities can be approximated using *PySAGAS*. Note, the error of each sensitivitiy, as calculated using the analytical solution for reference, is shown in brackets.

Parameter	dF_x/dp	dF_y/dp	dF_z/dp
θ	1.09 (-1.6%)	-3.20 (-3.1%)	0 (0%)
L	8.92 (0%)	-50.57 (0%)	0 (0%)
W	4.46 (0%)	-25.28 (0%)	0 (0%)

CHAPTER

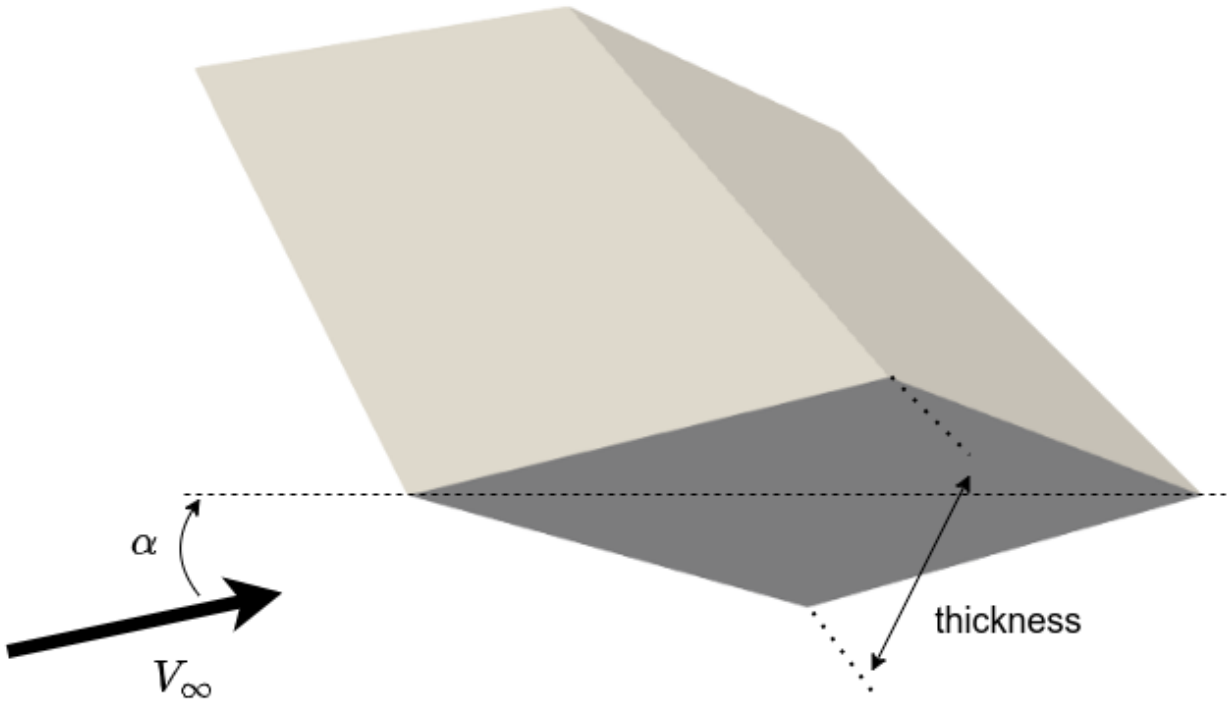
SEVEN

CART3D SHARP WEDGE STUDY

This example highlights the use of *PySAGAS* on a CFD solution. Here, a diamond wedge geometry has been simulated in Cart3D, though the flow solution from any solver could be used instead of Cart3D.

7.1 Problem Defintion

The geometry for this example was generated using the parameteric geometry generation tool HyperVehicle. This tool provides the capability of generating cell vertex sensitivities to geometric parameters.

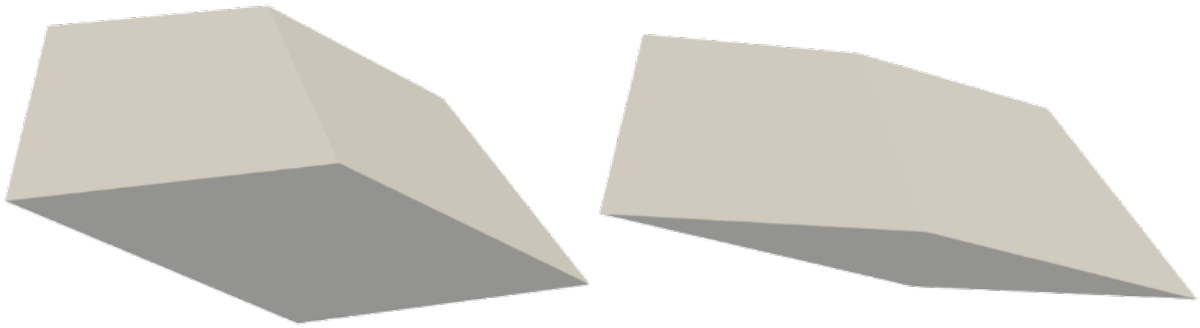


7.1.1 Free Stream Conditions

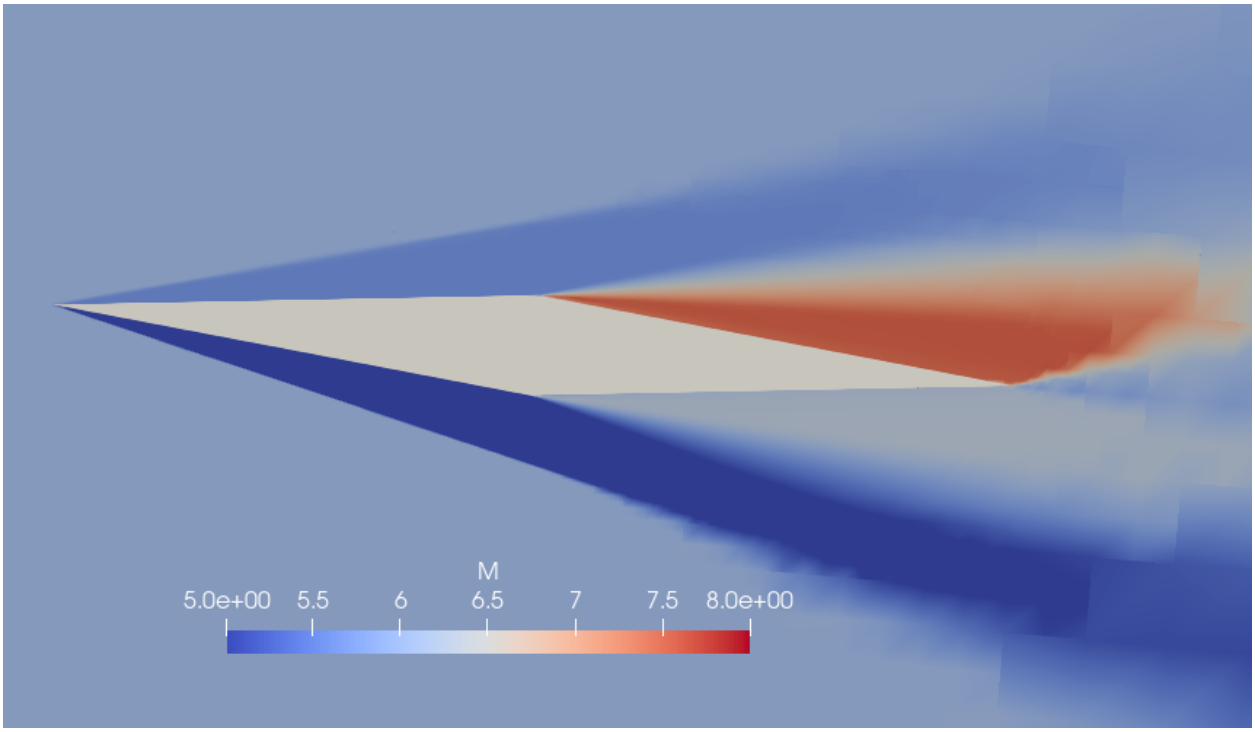
The wedge is simulated at Mach 6, with a 3-degree angle of attack. Since Cart3D is an inviscid flow solver, this is all that is required to define the non-dimensional flow state.

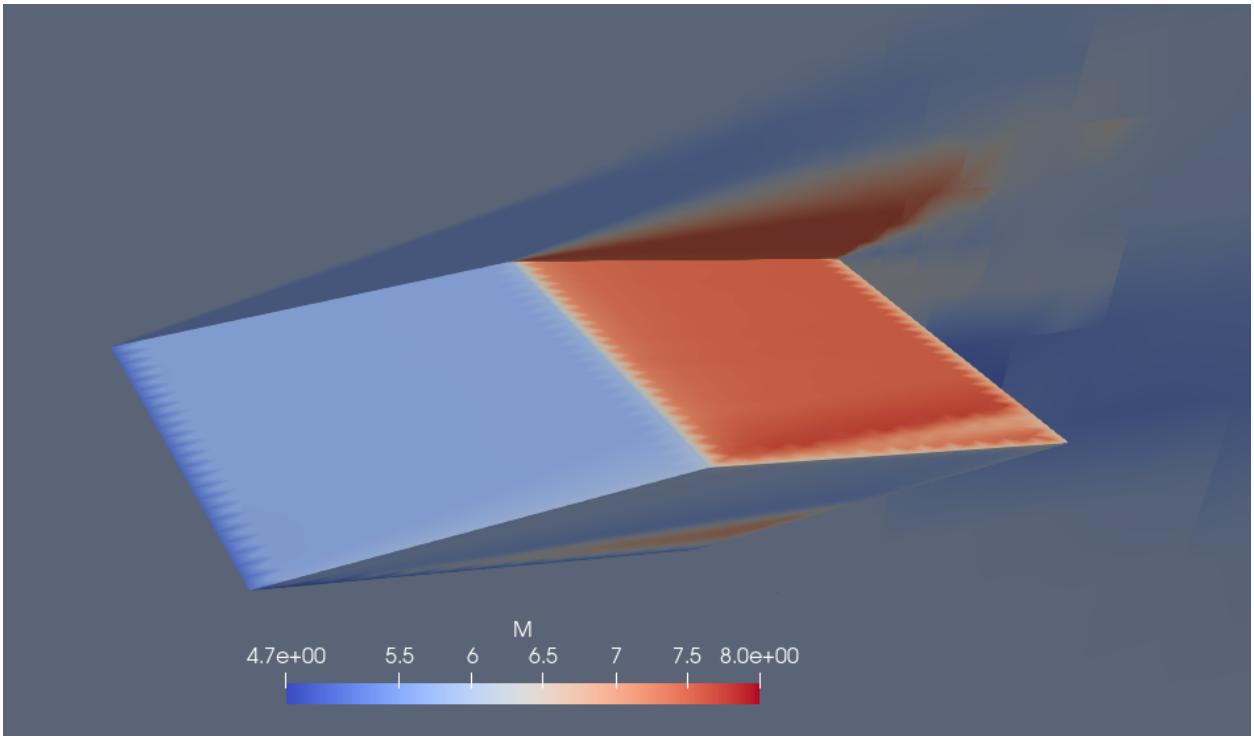
7.1.2 Parameterisation

To simplify this case study, a single parameter of wedge thickness is used to alter the wedge geometry.



7.2 Cart3D Solution





7.2.1 Parameter Sensitivities via Finite Differencing

Running a series of simulations in Cart3D for geometric perturbations of the wedge thickness about the nominal value, the following sensitiviites can be generated using finite differencing. Note that the values are reported for sensitivities in the non-dimensional coefficients C_x, C_y and C_z.

Parameter	$\partial C_x/\partial\theta$	$\partial C_y/\partial\theta$	$\partial C_z/\partial\theta$
Thickness	0.14517	0.126153	0.0000

7.3 PySAGAS Solution

Using the nominal geometry's Cart3D solution, the following sensitivities can be generated using *PySAGAS*. Note, the error of each sensitivitiy, as calculated using the Cart3D solution for reference, is shown in brackets.

Parameter	$\partial C_x/\partial\theta$	$\partial C_y/\partial\theta$	$\partial C_z/\partial\theta$
Thickness	0.15496 (5.0%)	0.11912 (-5.6%)	0.00312 (-)

7.4 Special Considerations

- Having a coarse geometry mesh will impact the accuracy of the Cart3D solution. In terms of computational expense, there is little reason to use a coarse geometry mesh, since this is loaded by Cart3D just once. A coarse geometry mesh may also not accurately capture the features of the geometry.
- The computational expense of *PySAGAS* scales with the number of cells which must be transcribed from the Cart3D solution, so the resolution of the geometry mesh should not be excessive.

CHAPTER

EIGHT

NOSE CONE SHAPE OPTIMISATION

This page illustrates how *pysagas* can be used for design optimisation. The example is applied to a nose cone at an operating point of Mach 6, 0 degrees angle-of-attack.

8.1 Literature Review

A review of literature reveals that the optimal shape for an axisymmetric nose at supersonic speeds which minimises drag for a given length and base diameter is given by the exponential function:

$$y = R(x/L)^n$$

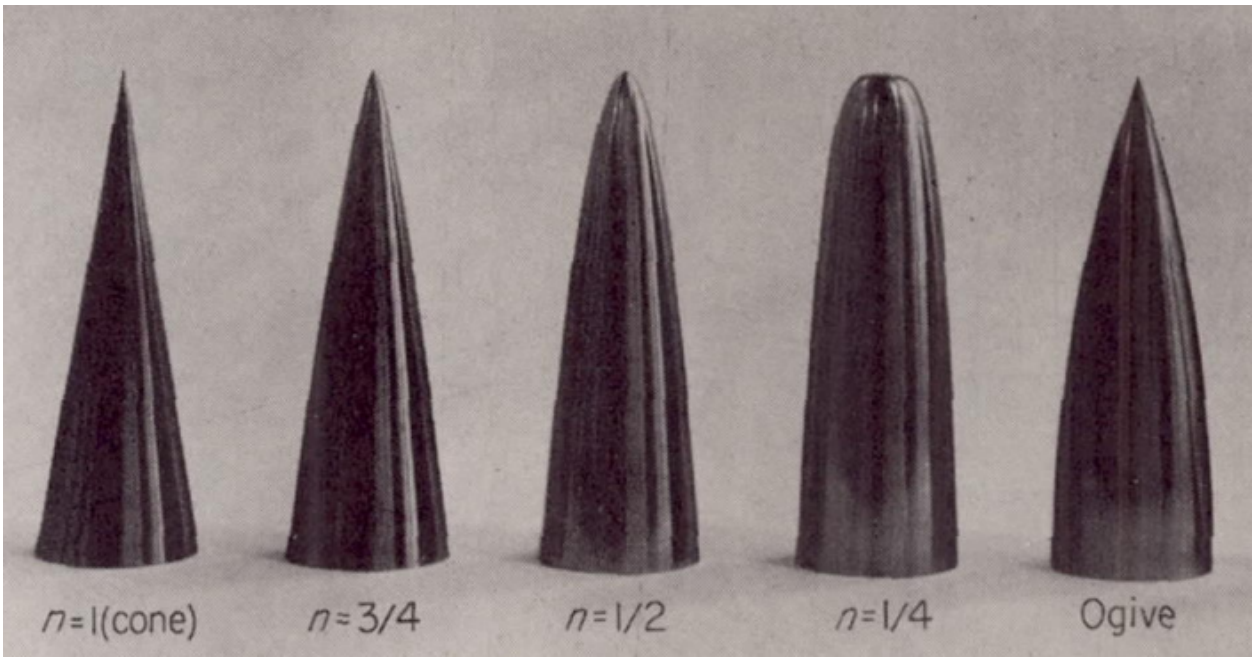
The exponent n depends weakly on the Mach number: for Mach 2-4, n should be approximately 0.68; for Mach >4, $n \approx 0.69$[1]. When $n = 1$, a sharp cone is produced.

The difference in drag for these exponnents is very slim, so previous experimental studies have made approximations with $n = 0.75$[2],[3].

[1] W. H. Mason and Jaewoo Lee. "Minimum-drag axisymmetric bodies in the supersonic/hypersonic flow regimes". In: Journal of Spacecraft and Rockets 31.3 (1994)

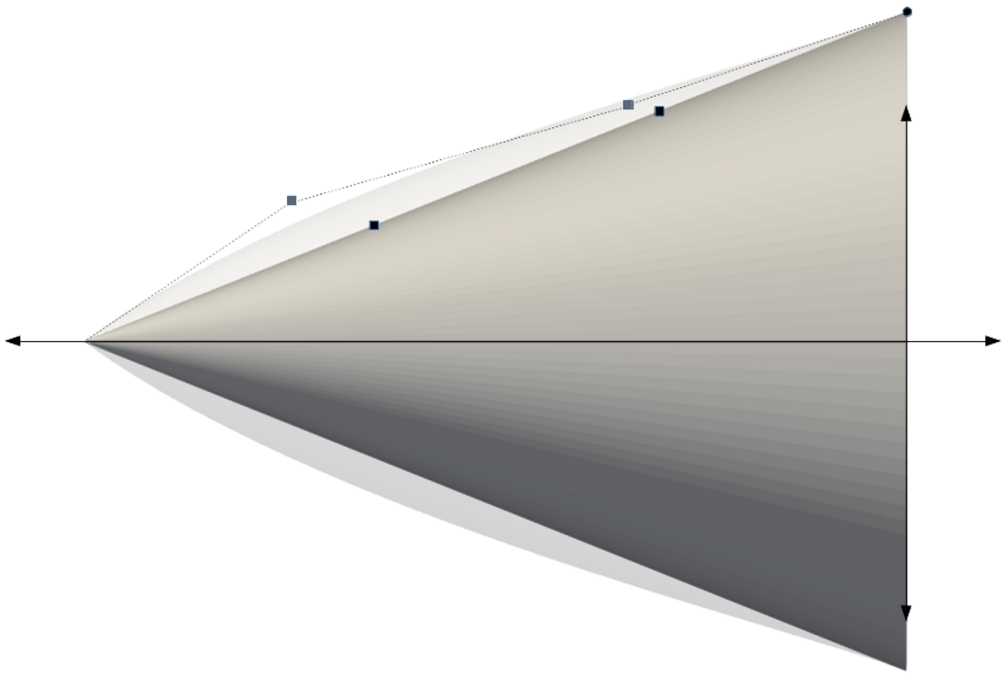
[2] Perkins, E.W., Jorgensen, L. H., and Sommer, S. C., "Investigation of the Drag of Various Axially Symmetric Nose Shapes of Fineness Ratio 3 for Mach Numbers from 1.24 to 7.4," NACA-TR-1386, 1958.

[3] Eggers, A. J. Jr., Resnikoff, M. M., and Dennis, D. H., "Bodies of Revolution Having Minimum Drag at High Supersonic Airspeeds," NACA-TR-1306, 1957.



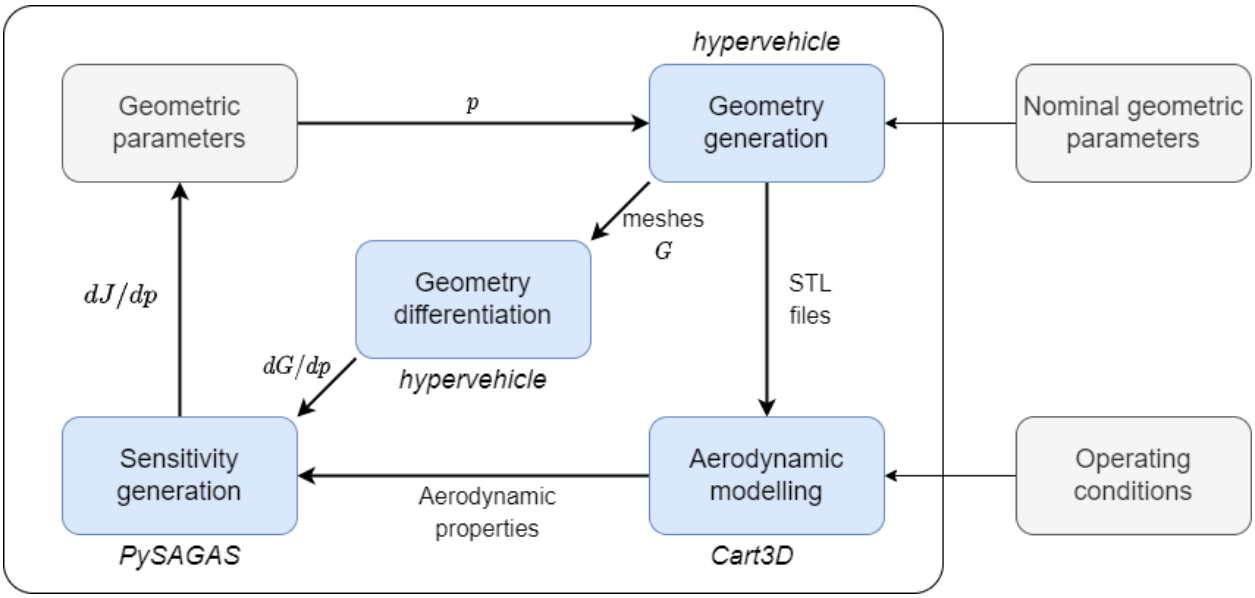
8.2 Geometry Definition

The nose geometry is defined by a Bezier curve with two internal control points, as shown in the figure below. These control points can move in both the horizontal and vertical direction. Therefore, there are 4 design parameters in this problem.



8.3 Optimisation Process

The optimisation process follows the schematic in the figure below. Starting with a nominal set of geometric parameters, *HyperVehicle* is used to generate STL files. These files are passed into an aerodynamic modelling package, in order to obtain the nominal flow solution for that geometry. *HyperVehicle* is also used to generate the geometry sensitivities, dv/dp, which are used in conjunction with the flow solution to generate the design parameter sensitivities via *pysagas*. Finally, these sensitivities can be passed into an optimisation algorithm to update the geometric parameters, and perform another update iteration. The loop will continue until some convergence tolerance is satisfied, at which point the optimal geometry will have been reached. This loop has been implemented in `pysagas.optimisation.cartd.ShapeOpt`.



In this example, the objective function is the drag coefficient of the nose cone, neglecting the base drag (as per the *prior art*).

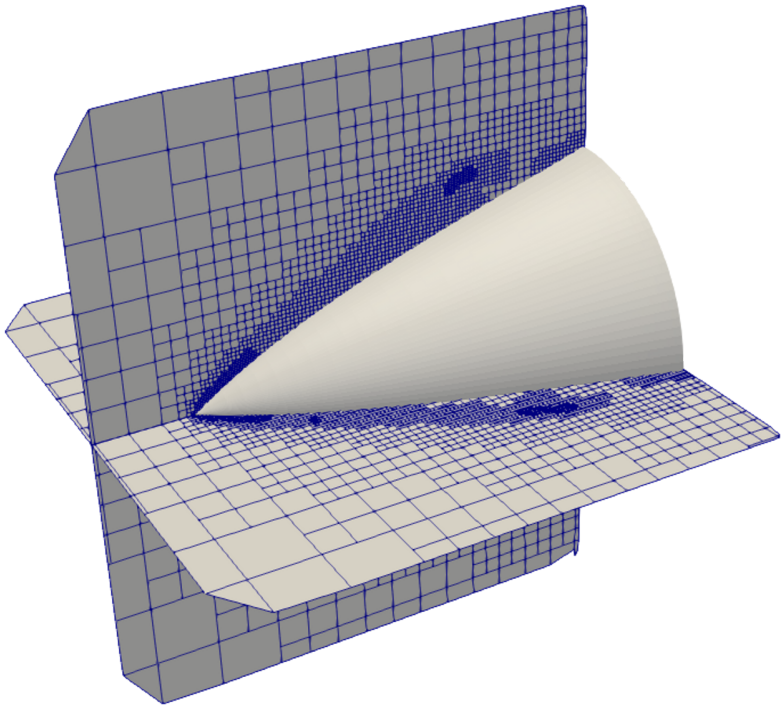
$$J = C_D$$

8.3.1 Geometry Generation and Differentiation

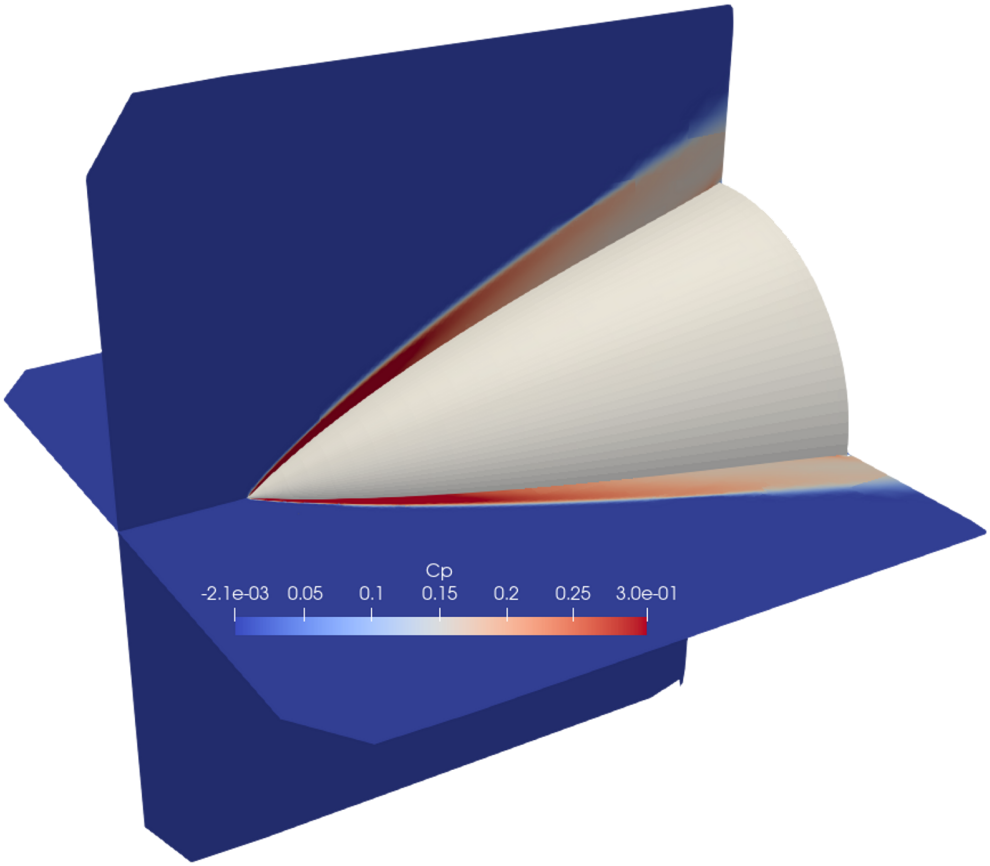
As mentioned above, the geometry and geometry-parameter sensitivities, dv/dp, will be generated using the parametric geometry generation package, *HyperVehicle*. This package allows a user to parametrically define a vehicle and output STL files for aerodynamic modelling.

8.3.2 Aerodynamic Modelling

Given the STL files from the geometry generation module, the geometry will be simulated in the inviscid CFD solver, *Cart3D*. The figure below shows an example mesh produced by Cart3D.

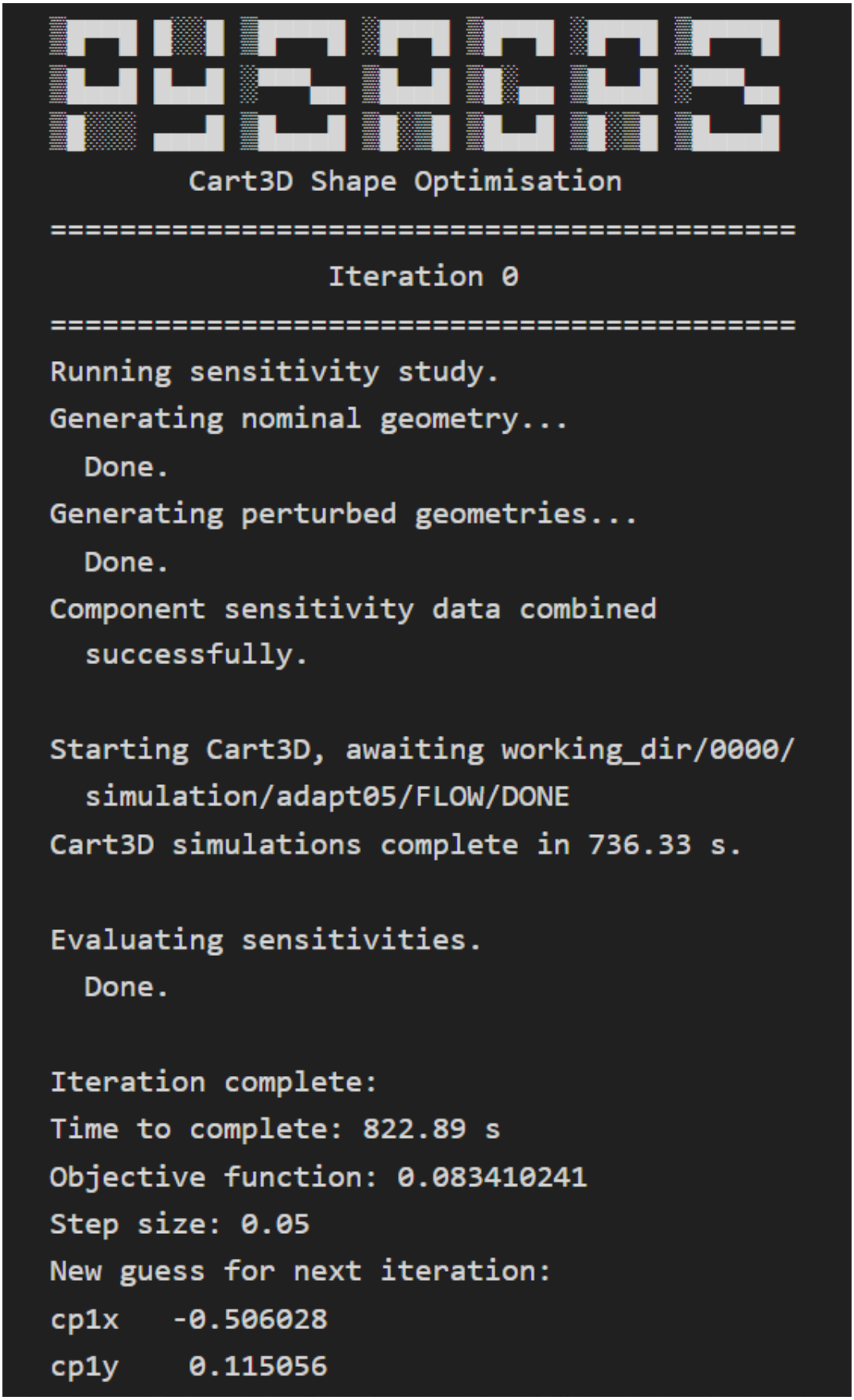


The corresponding flow solution for this mesh is visualised in the figure below.

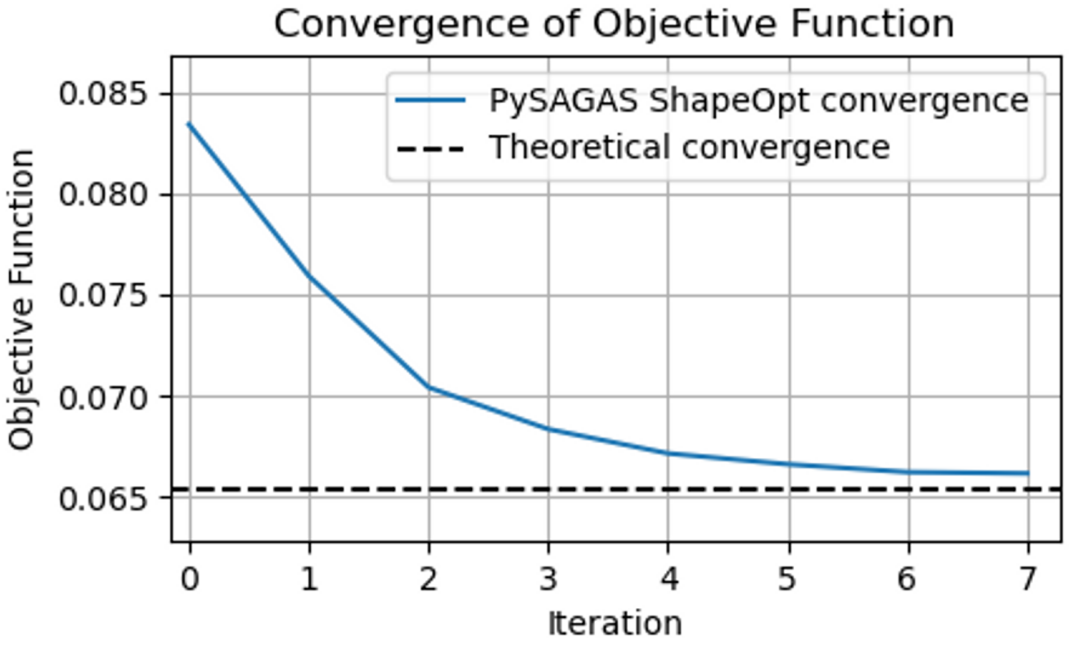


8.4 Results

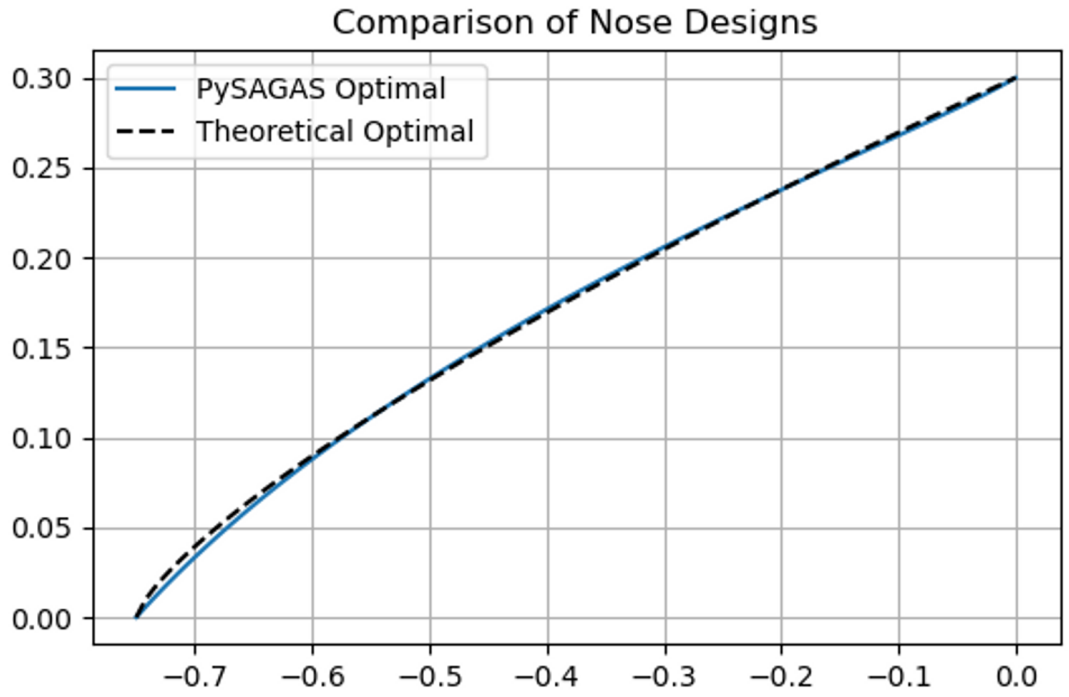
With the modules described above in place, the following output will be displayed when running `ShapeOpt`.


Cart3D Shape Optimisation
==========
Iteration 0
==========
Running sensitivity study.
Generating nominal geometry...
Done.
Generating perturbed geometries...
Done.
Component sensitivity data combined
successfully.
Starting Cart3D, awaiting working_dir/0000/
simulation/adapt05/FLOW/DONE
Cart3D simulations complete in 736.33 s.
Evaluating sensitivities.
Done.
Iteration complete:
Time to complete: 822.89 s
Objective function: 0.083410241
Step size: 0.05
New guess for next iteration:
cp1x -0.506028
cp1y 0.115056

After 7 iterations, the following convergence plot can be created. Also shown on this plot is the 'theoretical convergence', which refers to the drag coefficient of the nose defined by the *exponential function* above, when simulated in the same manner used in the optimisation.



Finally, the nose geometry produced by *pysagas* can be compared to the theoretical optimal nose geometry, defined by the *exponential function* above. This comparison is shown in the figure below.



CHAPTER

NINE

PYSAGAS GEOMETRY MODULES

9.1 PySAGAS Cell

class pysagas.geometry.cell.**Cell**

A triangular cell object.

p0

The first vertex of the cell.

Type
Vector

p1

The second vertex of the cell.

Type
Vector

p2

The third vertex of the cell.

Type
Vector

A

The cell face area.

Type
float

n

The cell normal.

Type
Vector

c

The cell centroid.

Type
Vector

dndv

The sensitivity of the cell's normal vector to each vertex.

Type
np.array

dAdv

The sensitivity of the cell's area to each vertex.

Type
np.array

dvdp

The sensitivity of the cell's vertices to each geometric parameter.

Type
np.array

dndp

The sensitivity of the cell's normal vector to each geometric parameter.

Type
np.array

dAdp

The sensitivity of the cell's area to each geometric parameter.

Type
np.array

dcdp

The sensitivity of the centroid to each geometric parameter.

Type
np.array

flowstate

The flowstate associated with the cell.

Type
FlowState

sensitivities

An array containing the [x,y,z] force sensitivities of the cell.

Type
np.array

static A_sensitivity(*p0*, *p1*, *p2*)

Calculates the sensitivity of a cell's area to the points defining the cell analytically.

Parameters

- **p0** (Vector) – The first point defining the cell.
- **p1** (Vector) – The second point defining the cell.
- **p2** (Vector) – The third point defining the cell.

Returns
sensitivity – The sensitivity matrix with size m x n. Rows m refer to the vertices, columns n refer to the vertex coordinates.

Return type
np.array

__init__(*p0*, *p1*, *p2*, *face_ids=None*)

Constructs a cell, defined by three points.

Parameters

- **p0** (Vector) – The first point defining the cell.
- **p1** (Vector) – The second point defining the cell.
- **p2** (Vector) – The third point defining the cell.
- **face_ids** (*list[int] | None*) –

static c_sensitivity(*p0*, *p1*, *p2*)

Calculates the sensitivity of a cell's centroid to the points defining the cell.

Parameters

- **p0** (Vector) – The first point defining the cell.
- **p1** (Vector) – The second point defining the cell.
- **p2** (Vector) – The third point defining the cell.

Returns

sensitivity – The sensitivity matrix with size m x n x p. Rows m refer to the vertices, columns n refer to the vertex coordinates, and slices p refer to the components of the centroid point.

Return type

np.array

static calc_area(*p0*, *p1*, *p2*)

Calculates the area of a cell defined by three points.

Parameters

- **p0** (Vector) – The first point defining the cell.
- **p1** (Vector) – The second point defining the cell.
- **p2** (Vector) – The third point defining the cell.

Returns

area – The area of the cell defined by the points.

Return type

float

References

https://en.wikipedia.org/wiki/Cross_product

static calc_centroid(*p0*, *p1*, *p2*)

Calculates the centroid of a cell defined by three points.

Parameters

- **p0** (Vector) – The first point defining the cell.
- **p1** (Vector) – The second point defining the cell.
- **p2** (Vector) – The third point defining the cell.

Returns

c – The centroid of the cell defined by the points.

Return type
Vector

References

https://en.wikipedia.org/wiki/Centroid

`static calc_normal`(*p0*, *p1*, *p2*)

Calculates the normal vector of a cell defined by three points.

Parameters

- **p0** (Vector) – The first point defining the cell.
- **p1** (Vector) – The second point defining the cell.
- **p2** (Vector) – The third point defining the cell.

Returns
normal – The unit normal vector of the cell defined by the points.

Return type
Vector

References

https://www.khronos.org/opengl/wiki/Calculating_a_Surface_Normal

`classmethod from_points`(*points*, ***kwargs*)

Constructs a Vector object from an array of coordinates.

Parameters
points (*Union[List[Vector], np.array[Vector]]*) – The points defining the cell.

Return type
Cell

`static n_sensitivity`(*p0*, *p1*, *p2*)

Calculates the sensitivity of a cell's normal vector to the points defining the cell analytically.

Parameters

- **p0** (Vector) – The first point defining the cell.
- **p1** (Vector) – The second point defining the cell.
- **p2** (Vector) – The third point defining the cell.

Returns
sensitivity – The sensitivity matrix with size m x n x p. Rows m refer to the vertices, columns n refer to the vertex coordinates, and slices p refer to the components of the normal vector.

Return type
np.array

`to_dict`()

Returns the Cell as a dictionary.

`property vertices`

The cell vertices.

exception pysagas.geometry.cell.**DegenerateCell**

Exception raised for degenerate cells.

__init__(*message='Degenerate cell'*)

9.2 PySAGAS Vector

class pysagas.geometry.vector.**Vector**

An N-dimensional vector.

__init__(*x=None*, *y=None*, *z=None*)

Define a new vector.

Parameters

- **x** (*float, optional*) – The x-component of the vector.
- **y** (*float, optional*) – The y-component of the vector.
- **z** (*float, optional*) – The z-component of the vector.

classmethod from_coordinates(*coordinates*)

Constructs a Vector object from an array of coordinates.

Parameters
coordinates (*np.array*) – The coordinates of the vector.

Return type
Vector

Examples

```
>>> Vector.from_coordinates([1,2,3])
Vector(1, 2, 3)
```

property norm: *Vector*

The norm associated with the Vector.

property unit: *Vector*

The unit vector associated with the Vector.

property vec: **array**

The vector represented as a Numpy array.

9.3 PySAGAS Geometry Parsers

CHAPTER

TEN

FLOW MODULE

class pysagas.flow.**FlowState**

An ideal gas state defined by Mach number, pressure and temperature, with a flow direction.

__init__(*mach*, *pressure*, *temperature*, *direction=None*, *aoa=0.0*, *gamma=1.4*)

Define a new flow state.

Parameters

- **mach** (*float*) – The flow Mach number.
- **pressure** (*float*) – The flow pressure (Pa).
- **temperature** (*float*) – The flow temperature (K).
- **direction** (Vector, *optional*) – The direction vector of the flow. The default is Vector(1,0,0).
- **aoa** (*float, optional*) – The angle of attack of the flow. The default is 0.0 (specified in degrees).
- **gamma** (*float, optional*) – The ratio of specific heats. The default is 1.4.

Return type
None

class pysagas.flow.**GasState**

An ideal gas state defined by Mach number, pressure and temperature.

__init__(*mach*, *pressure*, *temperature*, *gamma=1.4*)

Define a new gas state.

Parameters

- **mach** (*float*) – The flow Mach number.
- **pressure** (*float*) – The flow pressure (Pa).
- **temperature** (*float*) – The flow temperature (K).
- **gamma** (*float, optional*) – The ratio of specific heats. The default is 1.4.

Return type
None

CHAPTER

ELEVEN

PYSAGAS UTILITIES

pysagas.utilities.**add_sens_data**(*cells*, *data*, *verbosity=1*, *match_tolerance=1e-05*, *rounding_tolerance=1e-08*, *force=False*)

Appends shape sensitivity data to a list of Cell objects.

Parameters

- **cells** (*List[Cell]*) – A list of cell objects.
- **data** (*pd.DataFrame*) – The sensitivity data to be transcribed onto the cells.
- **verbosity** (*int, optional*) – The verbosity. The default is 1.
- **match_tolerance** (*float, optional*) – The precision tolerance for matching point coordinates. The default is 1e-5.
- **rounding_tolerance** (*float, optional*) – The tolerance to round data off to. The default is 1e-8.
- **force** (*bool, optional*) – Force the sensitivity data to be added, even if a cell already has sensitivity data. This can be used if new data is being used, or if the append is being repeated with a different matching tolerance. The default is False.

Returns

match_fraction – The fraction of points matched.

Return type

float

CHAPTER

TWELVE

PYSAGAS SENSITIVITY CALCULATORS

class pysagas.sensitivity.calculator.**AbstractSensitivityCalculator**

Abstract sensitivity calculator class defining the interface.

abstract __init__(***kwargs*)

Return type
None

abstract _extract_parameters()

Extract the geometry parameters.

Returns
parameters – A list of the parameter names.

Return type
List[str]

abstract _transcribe_cells(*parameters*)

Transcribes the unified Cells from the solver data.

Returns
cells – A list of the Cells from the flow solution.

Return type
List[*Cell*]

Parameters
parameters (*List*) –

abstract to_csv()

Dumps the sensitivity data to CSV file.

class pysagas.sensitivity.calculator.**GenericSensitivityCalculator**

__init__(*cells*, *sensitivity_filepath=None*, *cells_have_sens_data=False*, *verbosity=1*, ***kwargs*)

Instantiate a generic PySAGAS sensitivity calculator.

Parameters

- **cells** (*list[Cell]*) – A list of transcribed Cell objects, containing the nominal flow solution.
- **sensitivity_filepath** (*str*) – The filepath to the geometry sensitivities. This must be provided, if the cells do not have geometric sensitivity information attached. This must also be provided anyway, to determine the geometric design parameters.

- **cells_have_sens_data** (*bool, optional*) – The cells already have geometric sensitivity data matched to them. When this is True, the sensitivity_filepath argument (if provided) is ignored. The default is False.
- **verbosity** (*int, optional*) – The verbosity of the code. The defualt is 1.

Return type
None

`_transcribe_cells`(***kwargs*)

This is a dummy method to satisfy the abstract base class. Transcribed cells are provided upon instantiation of the sensitivity calculator.

Return type
List[Cell]

`class pysagas.sensitivity.calculator.SensitivityCalculator`

SensitivityCalculator base class.

`__init__`(***kwargs*)

Return type
None

`_extract_parameters`()

Extract the geometry parameters.

Returns
parameters – A list of the parameter names.

Return type
List[str]

`calculate`(*sensitivity_model='van_dyke'*, *cog=Vector(0, 0, 0)*, *flowstate=None*, ***kwargs*)

Calculate the force sensitivities of the surface to the parameters.

Parameters

- **sensitivity_model** (*Callable | Literal["piston", "van_dyke", "isentropic"], optional*) – The model used to calculate the pressure/parameter sensitivities. The default is Van Dyke's second-order theory model.
- **cog** (Vector, *optional*) – The centre of gravity. The default is Vector(0, 0, 0).
- **flowstate** (FlowState, *optional*) – The flowstate associated with this result. The default is None.

Returns
SensitivityResults

Return type
the sensitivity results object.

`static cell_sensitivity`(*cell*, *sensitivity_function*, *cog=Vector(0, 0, 0)*, ***kwargs*)

Calculates force and moment sensitivities for a single cell.

Parameters

- **cell** (Cell) – The cell.
- **sensitivity_function** (*Callable*) – The function to use when calculating the pressure sensitivities.

- **cog** (Vector, *optional*) – The reference centre of gravity, used in calculating the moment sensitivities. The defualt is Vector(0,0,0).

Returns
sensitivities – An array of shape n x 3, for a 3-dimensional cell with n parameters.

Return type
np.array

See also:

all_dfdp
a wrapper to calculate force sensitivities for many cells

static net_sensitivity(*cells*, *sensitivity_model='van_dyke'*, *cog=Vector(0, 0, 0)*, ***kwargs*)

Calcualtes the net force and moment sensitivities for a list of Cells.

Parameters

- **cells** (*list[Cell]*) – The cells to be analysed.
- **sensitivity_model** (*Callable | Literal["piston", "van_dyke", "isentropic"], optional*) – The model used to calculate the pressure/parameter sensitivities. The default is Van Dyke's second-order theory model.
- **cog** (Vector, *optional*) – The reference centre of gravity, used in calculating the moment sensitivities. The defualt is Vector(0,0,0).

Returns

- **dFdp** (*np.array*) – The force sensitivity matrix with respect to the parameters.
- **dMdp** (*np.array*) – The moment sensitivity matrix with respect to the parameters.

Return type
Tuple[*array*, *array*]

to_csv()

Dumps the sensitivity data to CSV file.

12.1 Supported CFD Wrappers

The following CFD sensitivity wrappers have been implemented into *PySAGAS*.

12.1.1 Cart3D PySAGAS Wrapper

The *PySAGAS* wrapper for Cart3D includes a main wrapper module and a utilities module.

class pysagas.sensitivity.cart3d.cart3d.**Cart3DSensitivityCalculator**

PySAGAS Cart3D flow sensitivity calculator.

__init__(*freestream*, *sensitivity_filepath*, *components_filepath=None*, *pointdata=None*, *celldata=None*, *write_data=False*, *verbosity=1*, ***kwargs*)

A PySAGAS sensitivity calculator for Cart3D.

Parameters

- **freestream** (FlowState) – The flow state of the freestream.

- **sensitivity_filepath** (*str*) – The filepath to the geometry sensitivities.
- **components_filepath** (*str, optional*) – The filepath to the Components.i.plt file to be processed. The default is None.
- **pointdata** (*pd.DataFrame, optional*) – The point data. Must be supplied with celldata.
- **celldata** (*pd.DataFrame, optional*) – The cell data. Must be supplied with pointdata.
- **write_data** (*bool, optional*) – Write the flow data to CSV files. The default is True.
- **verbosity** (*int, optional*) – The verbosity of the code. The defualt is 1.

Return type
None

_transcribe_cells(*parameters*)

Transcribes the cells from Components.i.plt files into PySAGAS Cell objects.

Parameters
parameters (*List[str]*) – A list of the geometric design parameters.

Returns
cells – A list of all transcribed cells.

Return type
List[*Cell*]

See also:

`pysagas.geometry.Cell`

pysagas.sensitivity.cart3d.utilities.**process_components_file**(*a_inf*, *rho_inf*, *filepath='Components.i.plt'*, *write_data=True*, *verbosity=1*)

A ParaView script to process Components.i.plt to extract points and cells with data attached.

Parameters

- **a_inf** (*float*) – The freestream speed of sound (m/s).
- **rho_inf** (*float*) – The freestream density (kg/m^3).
- **filepath** (*str, optional*) – The filepath to the Components.i.plt file to be processed. The default is Components.i.plt.
- **write_data** (*bool, optional*) – Write the flow data to CSV files. The default is True.
- **verbosity** (*int, optional*) – The verbosity of the code. The defualt is 1.

Returns

- **points** (*pd.DataFrame*) – A DataFrame of the point data.
- **cells** (*pd.DataFrame*) – A DataFrame of the cell data.

Return type
Tuple[*DataFrame*, *DataFrame*]

CHAPTER

THIRTEEN

PYSAGAS CFD MODULE

13.1 Oblique / Prandtl-Meyer Flow Solver

class pysagas.cfd.oblique_prandtl_meyer.**OPM**

Oblique shock thoery and Prandtl-Meyer expansion theory flow solver.

This implementation uses oblique shock theory for flow-facing cell elements, and Prandtl-Meyer expansion theory for rearward-facing elements.

13.1.1 Extended Summary

Data attribute 'method' refers to which method was used for a particular cell, according to:

- -1 : invalid / skipped (eg. 90 degree face)
- 0 : parallel face, do nothing
- 1 : Prandlt-Meyer
- 2 : normal shock
- 3 : oblique shock

static _load_sens_data(*sensitivity_filepath*)

Extracts design parameters from the sensitivity data.

Parameters
sensitivity_filepath (*str*) –

Return type
Tuple[*DataFrame*, list[set]]

static _solve_normal(*M1*, *p1=1.0*, *T1=1.0*, *gamma=1.4*)

Solves the flow using normal shock theory.

Parameters

- **M1** (*float*) – The pre-shock Mach number.
- **p1** (*float, optional*) – The pre-shock pressure (Pa). The default is 1.0.
- **T1** (*float, optional*) – The pre-expansion temperature (K). The default is 1.0.
- **gamma** (*float, optional*) – The ratio of specific heats. The default is 1.4.

Returns

- **M2** (*float*) – The post-shock Mach number.

- **p2** (*float*) – The post-shock pressure (Pa).
- **T2** (*float*) – The post-shock temperature (K).

static _solve_oblique(*theta*, *M1*, *p1=1.0*, *T1=1.0*, *gamma=1.4*)

Solves the flow using oblique shock theory.

Parameters

- **theta** (*float*) – The flow deflection angle, specified in radians.
- **M1** (*float*) – The pre-shock Mach number.
- **p1** (*float, optional*) – The pre-shock pressure (Pa). The default is 1.0.
- **T1** (*float, optional*) – The pre-expansion temperature (K). The default is 1.0.
- **gamma** (*float, optional*) – The ratio of specific heats. The default is 1.4.

Returns

- **M2** (*float*) – The post-shock Mach number.
- **p2** (*float*) – The post-shock pressure (Pa).
- **T2** (*float*) – The post-shock temperature (K).

static _solve_pm(*theta*, *M1*, *p1=1.0*, *T1=1.0*, *gamma=1.4*)

Solves for the Mach number, pressure and temperature after a Prandtl-Meyer expansion fan.

Parameters

- **theta** (*float*) – The deflection angle, specified in radians.
- **M1** (*float*) – The pre-expansion Mach number.
- **p1** (*float, optional*) – The pre-expansion pressure (Pa). The default is 1.0.
- **T1** (*float, optional*) – The pre-expansion temperature (K). The default is 1.0.
- **gamma** (*float, optional*) – The ratio of specific heats. The default is 1.4.

Returns

- **M2** (*float*) – The post-expansion Mach number.
- **p2** (*float*) – The post-expansion pressure (Pa).
- **T2** (*float*) – The post-expansion temperature (K).

static beta_max(*M*, *gamma=1.4*)

Returns the maximum shock angle for a given Mach number.

Parameters

- **M** (*float*) –
- **gamma** (*float*) –

static dp_dtheta_obl(*theta*, *M1*, *nominal_flowstate*, *p1=1.0*, *T1=1.0*, *gamma=1.4*, *perturbation=0.001*)

Solves for the sensitivities at the given point using oblique shock theory.

Parameters

- **theta** (*float*) – The deflection angle, specified in radians.
- **M1** (*float*) – The pre-expansion Mach number.
- **nominal_flowstate** (FlowState) – The nominal flowstate at this point.

- **p1** (*float, optional*) – The pre-expansion pressure (Pa). The default is 1.0.
- **T1** (*float, optional*) – The pre-expansion temperature (K). The default is 1.0.
- **gamma** (*float, optional*) – The ratio of specific heats. The default is 1.4.
- **perturbation** (*float, optional*) – The perturbation amount. Perturbations are calculated according to a multiple of (1 +/- perturbation). The default is 1e-3.

Returns

- **dM_dtheta** (*float*) – The sensitivity of the post-expansion Mach number with respect to the deflection angle theta.
- **dP_dtheta** (*float*) – The sensitivity of the post-expansion pressure (Pa) with respect to the deflection angle theta.
- **dT_dtheta** (*float*) – The sensitivity of the post-expansion temperature (K) with respect to the deflection angle theta.

static dp_dtheta_pm(*theta*, *M1*, *nominal_flowstate*, *p1=1.0*, *T1=1.0*, *gamma=1.4*, *perturbation=0.001*)

Solves for the sensitivities at the given point using Prandtl-Meyer theory.

Parameters

- **theta** (*float*) – The deflection angle, specified in radians.
- **M1** (*float*) – The pre-expansion Mach number.
- **nominal_flowstate** (FlowState) – The nominal flowstate at this point.
- **p1** (*float, optional*) – The pre-expansion pressure (Pa). The default is 1.0.
- **T1** (*float, optional*) – The pre-expansion temperature (K). The default is 1.0.
- **gamma** (*float, optional*) – The ratio of specific heats. The default is 1.4.
- **perturbation** (*float, optional*) – The perturbation amount. Perturbations are calculated according to a multiple of (1 +/- perturbation). The default is 1e-3.

Returns

- **dM_dtheta** (*float*) – The sensitivity of the post-expansion Mach number with respect to the deflection angle theta.
- **dP_dtheta** (*float*) – The sensitivity of the post-expansion pressure (Pa) with respect to the deflection angle theta.
- **dT_dtheta** (*float*) – The sensitivity of the post-expansion temperature (K) with respect to the deflection angle theta.

static inv_pm(*angle*, *gamma=1.4*)

Solves the inverse Prandtl-Meyer function using a bisection algorithm.

Parameters

- **angle** (*float*) –
- **gamma** (*float*) –

static oblique_M2(*M1*, *beta*, *theta*, *gamma=1.4*)

Calculates the Mach number following an oblique shock.

Parameters

- **M1** (*float*) – The pre-expansion Mach number.

- **beta** (*float*) – The shock angle specified in radians.
- **theta** (*float*) – The deflection angle, specified in radians.
- **gamma** (*float, optional*) – The ratio of specific heats. The default is 1.4.

Returns
M2 – The post-shock Mach number.

Return type
float

References

Peter Jacobs

oblique_T2_T1(*beta*, *gamma=1.4*)

Returns the temperature ratio T2/T1 across an oblique shock.

Parameters

- **M1** (*float*) – The pre-shock Mach number.
- **beta** (*float*) – The shock angle specified in radians.
- **gamma** (*float, optional*) – The ratio of specific heats. The default is 1.4.

Returns
T2/T1 – The temperature ratio across the shock.

Return type
float

References

Peter Jacobs

static oblique_beta(*M1*, *theta*, *gamma=1.4*, *tolerance=1e-06*)

Calculates the oblique shock angle using a recursive algorithm.

Parameters

- **M1** (*float*) – The pre-shock Mach number.
- **theta** (*float*) – The flow deflection angle, specified in radians.
- **gamma** (*float, optional*) – The ratio of specific heats. The default is 1.4.
- **tolerance** (*float*) –

References

Peter Jacobs

static oblique_p2_p1(*M1*, *beta*, *gamma=1.4*)

Returns the pressure ratio p2/p1 across an oblique shock.

Parameters

- **M1** (*float*) – The pre-shock Mach number.
- **beta** (*float*) – The shock angle specified in radians.

- **gamma** (*float*) –

Returns
p2/p1 – The pressure ratio across the shock.

Return type
float

References

Peter Jacobs

static oblique_rho2_rho1(*M1*, *beta*, *gamma=1.4*)

Returns the density ratio rho2/rho1 across an oblique shock.

Parameters

- **M1** (*float*) – The pre-shock Mach number.
- **beta** (*float*) – The shock angle specified in radians.
- **gamma** (*float, optional*) – The ratio of specific heats. The default is 1.4.

Returns
T2/T1 – The temperature ratio across the shock.

Return type
float

References

Peter Jacobs

static pm(*M*, *gamma=1.4*)

Solves the Prandtl-Meyer function and returns the Prandtl-Meyer angle in radians.

Parameters

- **M** (*float*) –
- **gamma** (*float*) –

save(*name*)

Save the solution to VTK file format. Note that the geometry mesh must have been loaded from a parser which includes connectivity information (eg. PyMesh or TRI).

Parameters

- **name** (*str*) – The filename prefix of the desired output file.
- **attributes** (*Dict[str, list]*) – The attributes dictionary used to initialise the file writer. Note that this is not required when calling *solve* from a child FlowSolver.

solve(*freestream=None*, *mach=None*, *aoa=None*, *cog=Vector(0, 0, 0)*)

Run the flow solver.

Parameters

- **freestream** (*Flowstate, optional*) – The free-stream flow state. The default is the freestream provided upon instantiation of the solver.

- **mach** (*float, optional*) – The free-stream Mach number. The default is that specified in the freestream flow state.
- **aoa** (*float, optional*) – The free-stream angle of attack. The default is that specified in the freestream flow state.
- **cog** (Vector) –

Returns
result – The flow results.

Return type
FlowResults

:raises Exception : when no freestream can be found.:

solve_sens(*sensitivity_filepath=None*, *parameters=None*, *cells_have_sens_data=False*, *freestream=None*, *mach=None*, *aoa=None*, *cog=Vector(0, 0, 0)*, *perturbation=0.001*)

Run the flow solver to obtain sensitivity information.

Parameters

- **freestream** (*Flowstate, optional*) – The free-stream flow state. The default is the freestream provided upon instantiation of the solver.
- **Mach** (*float, optional*) – The free-stream Mach number. The default is that specified in the freestream flow state.
- **aoa** (*float, optional*) – The free-stream angle of attack. The default is that specified in the freestream flow state.
- **sensitivity_filepath** (*str | None*) –
- **parameters** (*list[str] | None*) –
- **cells_have_sens_data** (*bool | None*) –
- **mach** (*float | None*) –
- **cog** (Vector) –
- **perturbation** (*float*) –

Return type
SensitivityResults

:raises Exception : when no freestream can be found.:

static theta_from_beta(*M1*, *beta*, *gamma=1.4*)

Calculates the flow deflection angle from the oblique shock angle.

Parameters

- **M1** (*float*) – The pre-expansion Mach number.
- **beta** (*float*) – The shock angle specified in radians.
- **gamma** (*float, optional*) – The ratio of specific heats. The default is 1.4.

Returns
theta – The deflection angle, specified in radians.

Return type
float

References

Peter Jacobs

13.2 Cart3D CFD Interface

13.3 PySAGAS CFD Utilities

class pysagas.cfd.deck.**AbstractDeck**

abstract __init__()

Return type
None

abstract from_csv(***kwargs*)

Load the deck from csv.

abstract insert()

Insert data into the deck.

abstract to_csv()

Write the deck to csv.

class pysagas.cfd.deck.**AeroDeck**

Aerodynamic coefficient deck.

__init__(*inputs*, *columns=['CL', 'CD', 'Cm']*, *a_ref=1*, *c_ref=1*)

Instantiate a new aerodeck.

Parameters

- **inputs** (*list[str]*) – A list of the inputs to this aerodeck. For example, ["aoa", "mach"].
- **columns** (*list[str], optional*) – A list of column names to use for this deck. The default is ["CL", "CD", "Cm"].
- **a_ref** (*float, optional*) – The reference area. The default is 1.
- **c_ref** (*float, optional*) – The reference length. The default is 1.

Return type
None

classmethod from_csv(*filepath*, *inputs*, *a_ref=1*, *c_ref=1*)

Create an Aerodeck from a CSV file.

Parameters

- **filepath** (*str*) – The filepath to the csv file containing aerodeck data.
- **inputs** (*list[str]*) – A list of the inputs to this aerodeck. For example, ["aoa", "mach"].
- **a_ref** (*float, optional*) – The reference area. The default is 1.
- **c_ref** (*float, optional*) – The reference length. The default is 1.

insert(*result*, ***kwargs*)

Insert aerodynamic coefficients into the aerodeck.

Parameters

result (*FlowResults | dict*) – The aerodynamic coefficients to be inserted, either as a PySAGAS native FlowResults object, or a dictionary with keys matching the data columns specified on instantiation of the deck.

See also:

FlowResults

class pysagas.cfd.deck.**Deck**

__init__(*inputs*, *columns*, ***kwargs*)

Instantiate a new deck.

Parameters

- **inputs** (*list[str]*) – A list of the inputs to this aerodeck. For example, ["aoa", "mach"].
- **columns** (*list[str]*) – A list of column names to use for this deck. For example, ["CL", "CD", "Cm"].

Return type
None

to_csv(*file_prefix=None*)

Save the deck to CSV file.

Parameters

file_prefix (*str, optional*) – The CSV file name prefix. If None is provided, the deck __repr__ will be used. The default is None.

class pysagas.cfd.deck.**MultiDeck**

Collection of multiple Deck objects.

__init__(*inputs*, *parameters*, *base_deck*, ***kwargs*)

Instantiate a new deck collection.

Parameters

- **inputs** (*list[str]*) – A list of the inputs to this deck. For example, ["aoa", "mach"].
- **parameters** (*list[str]*) – A list of the parameters to this deck. For example, ["wingspan", "length"].
- **base_deck** (*Deck*) – The base deck to initalise self._decks with.

Return type
None

class pysagas.cfd.deck.**SensDeck**

__init__(*inputs*, *parameters*, *a_ref=1*, *c_ref=1*, ***kwargs*)

Instantiate a new sensitivity deck.

This object is a collection of `AeroDeck`s, containing aerodynamic sensitivity information.

Parameters

- **inputs** (*list[str]*) – A list of the inputs to this deck. For example, ["aoa", "mach"].

- **parameters** (*list[str]*) – A list of the parameters to this deck. For example, ["wingspan", "length"].
- **a_ref** (*float, optional*) – The reference area. The default is 1.
- **c_ref** (*float, optional*) – The reference length. The default is 1.

Return type
None

See also:

AeroDeck

classmethod from_csv(*param_filepaths*, *inputs*, *a_ref=1*, *c_ref=1*)

Create a SensDeck from a collection of CSV files.

Parameters

- **param_filepaths** (*dict[str, str]*) – A dictionary of filepaths, keyed by the associated parameter.
- **inputs** (*list[str]*) – A list of the inputs to this aerodeck. For example, ["aoa", "mach"].
- **a_ref** (*float, optional*) – The reference area. The default is 1.
- **c_ref** (*float, optional*) – The reference length. The default is 1.

insert(*result*, ***kwargs*)

Insert data into the deck.

Parameters
result (SensitivityResults) –

to_csv(*file_prefix=None*)

Save the decks to CSV file.

Parameters
file_prefix (*str, optional*) – The CSV file name prefix. If None is provided, the deck __repr__ will be used. The default is None.

class pysagas.cfd.solver.**AbstractFlowSolver**

Abstract interface for flow solvers.

abstract __init__(***kwargs*)

Return type
None

class pysagas.cfd.solver.**FlowResults**

A class containing the aerodynamic force and moment information with respect to design parameters.

freestream

The freestream flow state.

Type
FlowState

net_force

The net force in cartesian coordinate frame (x,y,z).

Type
pd.DataFrame

m_sense

The net moment in cartesian coordinate frame (x,y,z).

Type
pd.DataFrame

__init__(*freestream*, *net_force*, *net_moment*)

Parameters

- **freestream** (FlowState) –
- **net_force** (Vector) –
- **net_moment** (Vector) –

Return type
None

coefficients(*A_ref=1.0*, *c_ref=1.0*)

Calculate the aerodynamic coefficients CL, CD and Cm.

Parameters

- **A_ref** (*float, optional*) – The reference area (m^2). The default is 1 m^2.
- **c_ref** (*float, optional*) – The reference length (m). The default is 1 m.

Return type
C_L, C_D, C_m

class pysagas.cfd.solver.**FlowSolver**

Base class for CFD flow solver.

__init__(*cells*, *freestream=None*, *verbosity=1*)

Instantiates the flow solver.

Parameters

- **cells** (*list[Cell]*) – A list of the cells describing the geometry.
- **freestream** (*Flowstate, optional*) – The free-stream flow state. The default is the freestream provided upon instantiation of the solver.
- **verbosity** (*int, optional*) – The verbosity of the solver. The default is 1.

Return type
None

static body_to_wind(*v*, *aoa*)

Converts a vector from body axes to wind axes.

Parameters

- **v** (Vector) – The result to transform.
- **aoa** (*float*) – The angle of attack, specified in degrees.

Returns
r – The transformed result.

Return type
Vector

save(*name*, *attributes*)

Save the solution to VTK file format. Note that the geometry mesh must have been loaded from a parser which includes connectivity information (eg. PyMesh or TRI).

Parameters

- **name** (*str*) – The filename prefix of the desired output file.
- **attributes** (*Dict[str, list]*) – The attributes dictionary used to initialise the file writer. Note that this is not required when calling *solve* from a child FlowSolver.

solve(*freestream=None*, *mach=None*, *aoa=None*)

Run the flow solver.

Parameters

- **freestream** (*Flowstate, optional*) – The free-stream flow state. The default is the freestream provided upon instantiation of the solver.
- **mach** (*float, optional*) – The free-stream Mach number. The default is that specified in the freestream flow state.
- **aoa** (*float, optional*) – The free-stream angle of attack. The default is that specified in the freestream flow state.

Returns
result – The flow results.

Return type
FlowResults

:raises Exception : when no freestream can be found.:

solve_sens(*freestream=None*, *mach=None*, *aoa=None*)

Run the flow solver to obtain sensitivity information.

Parameters

- **freestream** (*Flowstate, optional*) – The free-stream flow state. The default is the freestream provided upon instantiation of the solver.
- **Mach** (*float, optional*) – The free-stream Mach number. The default is that specified in the freestream flow state.
- **aoa** (*float, optional*) – The free-stream angle of attack. The default is that specified in the freestream flow state.
- **mach** (*float | None*) –

Return type
SensitivityResults

:raises Exception : when no freestream can be found.:

sweep(*aoa_range*, *mach_range*)

Evaluate the aerodynamic coefficients over a sweep of angle of attack and mach numbers.

Parameters

- **aoa_range** (*list[float]*) – The angles of attack to evaluate at, specified in degrees.
- **mach_range** (*list[float]*) – The Mach numbers to evaluate at.

Returns
results – A Pandas DataFrame of the aerodynamic coefficients at each angle of attack and Mach number combination.

Return type
pd.DataFrame

class pysagas.cfd.solver.**SensitivityResults**

A class containing the aerodynamic force and moment sensitivity information with respect to design parameters.

freestream

The freestream flow state.

Type
FlowState

f_sense

The force sensitivities in cartesian coordinate frame (x,y,z).

Type
pd.DataFrame

m_sense

The moment sensitivities in cartesian coordinate frame (x,y,z).

Type
pd.DataFrame

__init__(*f_sens*, *m_sens*, *freestream=None*)

Parameters

- **f_sens** (*DataFrame*) –
- **m_sens** (*DataFrame*) –
- **freestream** (FlowState | *None*) –

Return type
None

coefficients(*A_ref=1.0*, *c_ref=1.0*)

Calculate the aerodynamic coefficients CL, CD and Cm.

Parameters

- **A_ref** (*float, optional*) – The reference area (m^2). The default is 1 m^2.
- **c_ref** (*float, optional*) – The reference length (m). The default is 1 m.

Return type
A tuple (cf_sens, cm_sens) containing the force and moment coefficient sensitivities.

CHAPTER

FOURTEEN

PYSAGAS OPTIMISATION WRAPPERS

14.1 Supported Optimisation Wrappers

The following `ShapeOpt` wrappers have been implemented into *PySAGAS*.

14.1.1 Cart3D ShapeOpt Wrapper

CHAPTER

FIFTEEN

PYSAGAS CONTRIBUTION GUIDE

15.1 Contribution Guidelines

To contribute to `pysagas`, please carefully read the instructions below.

15.1.1 Seek Feedback Early

Please open an issue before a pull request to discuss any changes you wish to make.

15.1.2 Setting up for Development

1. Create a new Python virtual environment to isolate the package and its dependencies. You can do so using Python's `venv` or anaconda.
2. Install the code in editable mode using the command below (run from inside the `pysagas` root directory). Also install all dependencies using the `[all]` command, which includes the developer dependencies.

```
pip install -e .[all]
```

3. Install the pre-commit hooks. This will check that your changes are formatted correctly before you make any commits.

```
pre-commit install
```

4. Start developing! After following the steps above, you are ready to start developing the code. Make sure to follow the guidelines below.

15.1.3 Developing PySAGAS

- Before making any changes, fork this repository and clone it to your machine.
- Run black on any code you modify. This formats it according to PEP8 standards.
- Document as you go: use numpy style docstrings, and add to the docs where relevant.
- Write unit tests for the code you add, and include them in `tests/`. This project uses pytest.
- Commit code regularly to avoid large commits with many unrelated changes.

- Write meaningful commit messages, following the Conventional Commits standard. This is used for the purpose of maintaining semantic versioning and automated *changelogs*. The Python package `commitizen` is a great tool to help with this, and is already configured for this repo. Simply stage changed code, then use the `cz c` command to make a commit. If you do not wish to do this, your commits will be squashed before being merged into `main`.
- Open a Pull Request when your changes are complete and ready to be merged.

15.1.4 Building the Docs

PySAGAS is documented using Sphinx. To build the documentation, run the commands below from the root directory.

```
cd docs/
make html
xdg-open build/html/index.html
```

If you are actively developing the docs, consider using sphinx-autobuild. This will continuosly update the docs for you to see any changes live, rather than re-building repeatadly. From the root directory, run the following command:

```
sphinx-autobuild docs/source/ docs/build/html --open-browser --watch pysagas/
```

15.2 Implementing a New Senstivity Calculater

PySAGAS sensitivity calculators for different CFD solvers all inherit from the *SensitivityCalculator* class.

CHAPTER

SIXTEEN

CHANGELOG

16.1 v0.14.0 (2024-02-27)

16.1.1 Feat

- use meshio instead of pymesh when saving FlowSolver results
- **parsers.py**: added meshio parser

16.2 v0.13.0 (2024-01-02)

16.2.1 Feat

- **ShapeOpt**: added passing of cog data

16.2.2 Fix

- **optimisation**: fixed stale import
- **OPMShapeOpt**: use generic sensitivity solver instead of finite diff
- **Cart3DShapeOpt**: add sim_success flag to recover from bad sim
- **van_dyke_dPdp**: remove temporary method
- **Cell**: fix face_ids type hint

16.2.3 Refactor

- **opm**: convert opm module name to lowercase for PEP8
- renamed dPdp methods to sensitivity for clarity
- refactored sensitivity models into senstivity module
- **wrapper**: refactored wrapper module into sensitivity module
- **utilities.py**: integrate piston sens with mach_limited_piston function
- **utilities.py**: move test helper functions into relevant test module
- **cart3d.py**: update import to avoid pymesh dependency error

16.3 v0.12.1 (2023-11-06)

16.3.1 Refactor

- **optimisation.cartd.init.py**: remove imports to simplify

16.4 v0.12.0 (2023-10-31)

16.4.1 Feat

- **Parser**: STL load_from_file also appends sensitivity data
- **OPM**: added count of bad cells for diagnosing bad solve
- **OPM**: optionally provide parameters to avoid sens extraction
- **FlowState**: add Vector property
- **Cart3DShapeOpt**: provide kwargs for wrapper.calculate

16.4.2 Fix

- **Wrapper**: _extract_parameters preserves parameter ordering
- **van_dyke_dPdp_ingo**: take kwargs

16.5 v0.11.0 (2023-06-18)

16.5.1 Feat

- **deck.py**: added multideck object to tidy sensitivity deck
- **SensitivityResults**: translate f_sens to aero frame for coefficients
- **Deck**: added from_csv methods
- **deck.py**: added aerodeck object
- **Wrapper**: return SensitivityResult from calculate
- **Wrapper**: add verbosity to calculate method
- **PyMesh-Parser**: optionally provide geom sens data to append to cells on load
- **GenericWrapper**: specify that cells already have geom sens data to prevent transcription
- **wrappers**: added generic wrapper to calculate sensitivities from provided cells

16.5.2 Refactor

- **Sensdeck**: change type to sensdeck

16.6 v0.10.0 (2023-06-13)

16.6.1 Feat

- **C3DPrep**: improved docstrings
- **C3DPrep**: allow passing args to autoinputs
- **utilities.py**: implemented van dykes second order theory for sensitivities
- **Cart3DMooShapeOpt**: added multi-objective cart3d shape optimiser
- **combine_sense_data**: added verbosity control
- **Cart3D**: solve_sens method implemented
- **Cart3D**: improved running function
- **Cart3D**: added status and stop functionality
- **Cart3D**: added cfd wrapper for cart3d
- **FlowState**: define direction by aoa
- **OPMShapeOpt**: added verbosity
- **OPMShapeOpt**: pass parameters to objective callback
- **add_sens_data**: improved reporting
- **OPM**: added perturbation amount argument for sens
- **add_sens_data**: return point match fraction metric
- **FlowSolver**: added sweep method to generate aerodeck
- **OPM**: calculate net moment
- first implementation of OPM optimisation wrapper
- **FlowResults**: improved aero coefficient helper method
- **FlowSolver**: added aero coefficient helper
- **PyMesh**: implemented multiprocessing for parser
- **FlowSolver**: check if a flow state has already been solved before proceeding
- **OPM**: implemented sensitivity solver
- **add_sens_data**: function to append sensitivity data to cells
- **FlowSolver**: added method to convert body to wind axes
- **GasState**: added default flow direction and q property
- **FlowSolver**: added convenience inputs of aoa and mach for solver instance
- **OPM**: added finite difference sensitivities for PM and oblique
- **OPM**: use normal shock relations for theta > theta_max

- **TRI-Parser**: now includes connectivity information
- **FlowSolver**: added vtk file save method
- **OPM**: first implementation running to produce net force vector
- **Parser**: implemented stl and tri parsers
- **OPM**: added oblique theory methods

16.6.2 Fix

- **van_dyke_dPdp**: skip subsonic cells
- **Cart3DMooShapeOpt**: combine sens data for each sim point
- **Cart3D**: path management
- **OPMShapeOpt**: force reloading cells and instantiating solver
- **OPMShapeOpt**: re-instantiate cells and solver on each evaluation
- **OPM-ShapeOpt**: chain rule theta through parameter
- **OPMShapeOpt**: try load with pymesh, except use STL
- **add_sens_data**: removed print line interfering with progress bar
- **OPM**: fixed sensitivity evaluation
- **OPM**: allow solving sens with cells that already have geom sens data
- **GasState**: dynamically return properties to allow updating state
- **OPM**: threshold expansion cells
- **FlowResults**: **str** return type
- **Cell**: ambiguity of numpy array
- **geometry**: fixed namespace generation

16.6.3 Refactor

- **Cart3DWrapper**: pass flowstate instead of individual properties
- **combine_sense_data**: moved to pysagas.optimisation.cartd.utilities
- **geometry**: tidied geometry module into submodule
- **FlowSolver**: started implemented prelimary flow solvers

16.6.4 Perf

- **Cell**: delay calculation of geometric sensitivities until required

16.7 v0.9.0 (2023-04-15)

16.7.1 Feat

- **C3DPrep**: add option to not write config.xml file
- **Cart3DShapeOpt**: pass user-defined properties and sensitivities to callback functions
- **Cart3DShapeOpt**: convert tri files to dat
- **Cart3DShapeOpt**: option to copy components file to observe evolution
- **ShapeOpt**: check bounds on variables when added
- **ShapeOpt**: implemented constraint interface
- **ShapeOpt**: implemented hypercube bounds
- **parse_tri_file**: added progress bar and verbosity control
- **ShapeOpt**: read c3d commands from warmstart dir
- **ShapeOpt**: prepare new simulation with warmstart flag
- **utilities**: implemented Newtonian impact solver

16.7.2 Fix

- **C3DPrep**: config.xml writing option
- paraview and vtk compatibility
- **Cart3DShapeOpt**: revert file copying without .dat conversion
- **ShapeOpt**: pass variable bounds
- **Cart3DShapeOpt**: working directory cleaning logic
- **ShapeOpt**: pass parameters to obj_jac_cb
- **ShapeOpt**: handle when a geom has no properties data
- **ShapeOpt**: raise exception if Cart3D commands cannot be found
- **ShapeOpt**: added verbosity to logging
- **ShapeOpt**: append cart3d stdout to log file
- **ShapeOpt**: open/close file when running main cmd with stdout/err pipe
- **ShapeOpt**: mgPrep filename for warmstarted iterations
- **ShapeOpt**: run c3d commands using subprocess.run
- **ShapeOpt**: use os.path.join on conditional
- **ShapeOpt**: rename checkpoint file when copying warmstart files
- **ShapeOpt**: distinguish preparaton warmstart flag from simulation warmstart flag
- **ShapeOpt**: handle FileNotFoundErrors in _infer_adapt method
- **ShapeOpt**: remove hardcoded run command
- **ShapeOpt**: allow restarting warmstarted iteration
- **ShapeOpt**: return line in _find_in_file method

- **ShapeOpt**: move *.tri files into simulation directory on warmstarts
- **ShapeOpt**: override warmstart flag on restarts
- **ShapeOpt**: warmstart bool ambiguity
- **ShapeOpt**: falsify warmstart flag on initial iteration
- **ShapeOpt**: pass warmstart flag to run_simulation method
- **ShapeOpt**: allow specifying max_iterations
- **newtonian_impact_solver**: return type hint

16.7.3 Refactor

- **Cart3DShapeOpt**: tidied repeated code
- **Cart3DShapeOpt**: clean working directory before running any operations
- **Cart3DShapeOpt**: minimal working example complete
- **Cart3DShapeOpt**: moved methods to functions
- **Cart3DShapeOpt**: improved flow of evaluate_objective method
- **Cart3DShapeOpt**: refactored into unified interface
- tidied up cart3d optimisation methods
- **C3DPrep**: moved into utilities module for C3D optimisation
- set up inheritance structure

16.8 v0.8.0 (2023-03-01)

16.8.1 Feat

- **ShapeOpt**: handle bad combined sens data
- **ShapeOpt**: limit number of C3D restarts
- **ShapeOpt**: added general C3D handling for 'ERROR'
- **C3DPrep**: optionally specify number of rotation attempts before quitting
- **ShapeOpt**: objective and jacobian provided by user callback function

16.8.2 Fix

- **ShapeOpt**: explicitly specify sensitivity files when combining
- **Cart3DWrapper**: raise exception when len(sensdata) != len(pointdata)
- **C3DPrep**: apply reverse rotations upon intersection
- **ShapeOpt**: added max_adapt arg back in after merge deletion

16.8.3 Refactor

- **ShapeOpt**: attempt component intersection multiple times
- **ShapeOpt**: retrieve vol and mass from properties dir

16.9 v0.7.0 (2023-02-05)

16.9.1 Feat

- implemented moment sensitivities
- **_C3DPrep**: overwrite Config.xml with tri files prefix
- **ShapeOpt**: implemented c3d adapt cycle scheduling
- **ShapeOpt**: added cubes failure to c3d errors to catch
- **cell_dfdp**: added mechanism to calculate moment sensitivities
- **ShapeOpt**: optionally provide theoretical convergence limit to post process
- **_C3DPrep**: added control over jitter
- **ShapeOpt**: implemented max step size argument
- **ShapeOpt**: added C3D error to recovery
- **ShapeOpt**: added error handling for C3D crashes
- **ShapeOpt**: added iterative matching tol reduction
- **ShapeOpt**: catch zero norm jacobian and bail
- **Cart3D**: improved verbosity output
- **DegenerateCell**: implemented exception for degenerate cells
- option to show plot in post for c3d shapeopt
- **ShapeOpt**: implemented automated step size
- **optimisation**: implemented basic shape optimisation wrapper for cart3d

16.9.2 Fix

- **cell_dfdp**: include dr/dp in moment sensitivity calculation
- **ShapeOpt**: updated output of wrapper.calculate and sensitivity notation
- **ShapeOpt**: warmstart continuity of x_older and jac_older
- **process_components**: enforce paraview to point data in spreadsheet view
- **Cart3DWrapper**: improved parameter extraction

16.9.3 Refactor

- **Wrapper**: allow passing c.o.g. as arg to calculate

16.10 v0.6.0 (2023-01-18)

16.10.1 Feat

- **Wrapper**: return sensitivities as dataframe

16.11 v0.5.0 (2022-12-20)

16.11.1 Feat

- **Cart3DWrapper**: allow directly providing point and cell data frames
- added sensitivities to transcribed cells
- modularised dPdp method

16.11.2 Fix

- **test_cart3d.py**: name of test function
- **isentropic_dPdp**: calculation of pressure sensitivity

16.11.3 Refactor

- improved flow of cart3d wrapper
- improved wrapper interface

16.11.4 Perf

- **Wrapper**: check if cells have already been transcribed

16.12 v0.4.0 (2022-12-08)

16.12.1 Feat

- **utilities.py**: modularised dPdp method

16.12.2 Fix

- merge conflicts

16.12.3 Refactor

- **wrappers**: added wrappers submodule heirarchy

16.13 v0.3.1 (2022-12-08)

16.13.1 Fix

- **pyproject.toml**: rename to pysagas

16.13.2 Refactor

- rename package to pysagas

16.14 v0.3.0 (2022-12-06)

16.14.1 Feat

- **init.py**: expose key classes to top namespace
- **test_cart3d.py**: code running to completion
- assign FlowState to Cells
- **flow.py**: created GasState and FlowState classes
- **Vector**: added unit vector property
- **utilities.py**: added ParaView macro to process components.i.plt
- **geometry.py**: added class methods for Vector and Cell objects

16.14.2 Fix

- **flow.py**: circular import
- fixed circular import error
- **utilities.py**: fixed dimensionality for parameters

16.15 v0.2.0 (2022-11-30)

16.15.1 Feat

- implemented use of Cells
- **Cell**: added sensitivity methods to cell
- **geometry.py**: implemented area and normal methods for Cell
- **geometry.py**: implemented operation methods for Vector
- **utilities.py**: added force sensitivity utilities
- **geometry.py**: added vec property to Vector object
- **geometry.py**: added geometry module
- added some utilities
- **flow.py**: added flow state object

16.15.2 Refactor

- code tidy up

CHAPTER

SEVENTEEN

CITING PYSAGAS

Information coming soon!

PYTHON MODULE INDEX

p

INDEX

Symbols

A

B

C

D

F

G

I

M

N

O

P

S

T

U

V