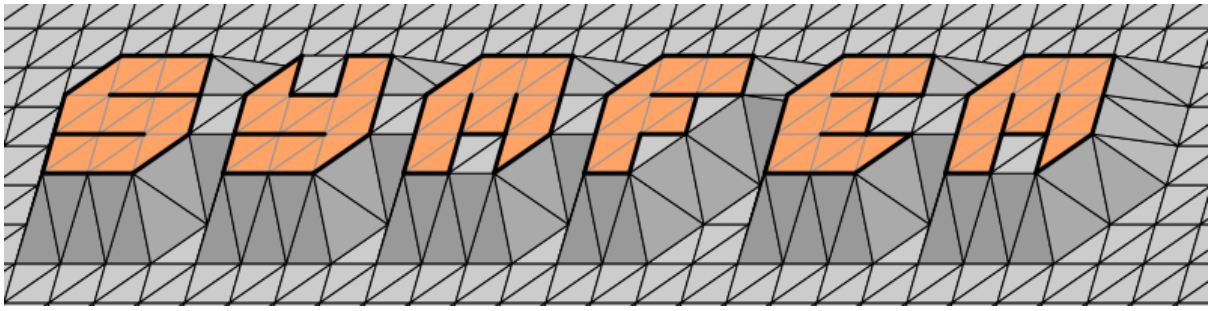
Symfem Documentation

Release 2024.1.1

Matthew Scroggs

Jan 04, 2024

CONTENTS



Welcome to the documention of Symfem: a symbolic finite element definition library.

Symfem can be used to create a very wide range of finite element spaces. The basis functions of these spaces can be computed symbolically, allowing them to easily be further manipulated.

CHAPTER

ONE

INSTALLING SYMFEM

Symfem can be installed from GitHub repo by running:

```
git clone https://github.com/mscroggs/symfem.git
cd symfem
python3 setup.py install
```

Alternatively, the latest release can be installed from PyPI by running:

```
pip3 install symfem
```

... or from Conda by running:

```
conda install symfem
```

1.1 Using Symfem

1.1.1 Finite elements

Finite elements can be created in Symfem using the *symfem.create_element()* function. For example, some elements are created in the following snippet:

```
import symfem

lagrange = symfem.create_element("triangle", "Lagrange", 1)
rt = symfem.create_element("tetrahedron", "Raviart-Thomas", 2)
nedelec = symfem.create_element("triangle", "N2curl", 1)
qcurl = symfem.create_element("quadrilateral", "Qcurl", 2)
```

create_element will create a *symfem.finite_element.FiniteElement* object. The basis functions spanning the finite element space can be obtained, or tabulated at a set of points:

```
import symfem

lagrange = symfem.create_element("triangle", "Lagrange", 1)
print(lagrange.get_basis_functions())

points = [[0, 0], [0.5, 0], [1, 0], [0.25, 0.25]]
print(lagrange.tabulate_basis(points))
```

```
[-x - y + 1, x, y]
[(1, 0, 0), (0.500000000000000, 0.500000000000000, 0), (0, 1, 0), (0.500000000000000,␣
↪0.250000000000000, 0.250000000000000)]
```

Each basis function will be a Sympy symbolic expression.

The majority of the elements in Symfem are defined using Ciarlet's [Ciarlet] definition of a finite element (*symfem.finite_element.CiarletElement*): these elements are using a polynomial set, and a set of functionals. In Symfem, the polynomial set of an element can be obtained by:

```
import symfem

lagrange = symfem.create_element("triangle", "Lagrange", 1)
print(lagrange.get_polynomial_basis())
```

```
[1, x, y]
```

The functionals of the finite element space can be obtained with the following snippet.

```
import symfem

lagrange = symfem.create_element("triangle", "Lagrange", 1)
print(lagrange.dofs)
```

```
[<symfem.functionals.PointEvaluation object at 0x{ADDRESS}>, <symfem.functionals.
↪PointEvaluation object at 0x{ADDRESS}>, <symfem.functionals.PointEvaluation object␣
↪at 0x{ADDRESS}>]
```

Each functional will be a functional defined in *symfem.functionals*.

1.1.2 Reference cells

Reference cells can be obtained from a *symfem.finite_element.FiniteElement*:

```
import symfem

lagrange = symfem.create_element("triangle", "Lagrange", 1)
reference = lagrange.reference
```

Alternatively, reference cells can be created using the *symfem.create_reference()* function. For example:

```
import symfem

triangle = symfem.create_reference("triangle")
interval = symfem.create_reference("interval")
tetrahedron = symfem.create_reference("tetrahedron")
triangle2 = symfem.create_reference("triangle", ((0, 0, 0), (0, 1, 0), (1, 0, 1)))
```

In the final example, the vertices of the reference have been provided, so a reference with these three vertices will be created.

Various information about the reference can be accessed. The reference cell's subentities can be obtained:

```
import symfem

triangle = symfem.create_reference("triangle")
print(triangle.vertices)
print(triangle.edges)
```

```
((0, 0), (1, 0), (0, 1))
((1, 2), (0, 2), (0, 1))
```

The origin and axes of the element can be obtained:

```
import symfem

triangle = symfem.create_reference("triangle")
print(triangle.origin)
print(triangle.axes)
```

```
(0, 0)
((1, 0), (0, 1))
```

The topological and geometric dimensions of the element can be obtained:

```
import symfem

triangle = symfem.create_reference("triangle")
print(triangle.tdim)
print(triangle.gdim)

triangle2 = symfem.create_reference("triangle", ((0, 0, 0), (0, 1, 0), (1, 0, 1)))
print(triangle2.tdim)
print(triangle2.gdim)
```

```
2
2
2
3
```

The reference types of the subentities can be obtained. This can be used to create a reference representing a subentity:

```
import symfem

triangle = symfem.create_reference("triangle")
print(triangle.sub_entity_types)

vertices = []
for i in triangle.edges[0]:
    vertices.append(triangle.vertices[i])
edge0 = symfem.create_reference(
    triangle.sub_entity_types[1], vertices)
print(edge0.vertices)
```

```
['point', 'interval', 'triangle', None]
((1, 0), (0, 1))
```

1.2 Documentation index

1.2.1 Symfem demos

Checking a Lagrange element

This demo shows how Symfem can be used to confirm that when the basis functions of a Lagrange element of a triangle are restricted to an edge, then they are equal to the basis functions of a Lagrange element on that edge.

```
"""Demo showing how Symfem can be used to verify properties of a basis.

The basis functions of a Lagrange element, when restricted to an edge of a cell,
should be equal to the basis functions of a Lagrange space on that edge (or equal to␣
↪0).

In this demo, we verify that this is true for an order 5 Lagrange element on a␣
↪triangle.
"""

import sympy

import symfem
from symfem.symbols import x
from symfem.utils import allequal

element = symfem.create_element("triangle", "Lagrange", 5)
edge_element = symfem.create_element("interval", "Lagrange", 5)

# Define a parameter that will go from 0 to 1 on the chosen edge
a = sympy.Symbol("a")

# Get the basis functions of the Lagrange space and substitute the parameter into the
# functions on the edge
basis = element.get_basis_functions()
edge_basis = [f.subs(x[0], a) for f in edge_element.get_basis_functions()]

# Get the DOFs on edge 0 (from vertex 1 (1,0) to vertex 2 (0,1))
# (1 - a, a) is a parametrisation of this edge
dofs = element.entity_dofs(0, 1) + element.entity_dofs(0, 2) + element.entity_dofs(1,␣
↪0)
# Check that the basis functions on this edge are equal
for d, edge_f in zip(dofs, edge_basis):
    # allequal will simplify the expressions then check that they are equal
    assert allequal(basis[d].subs(x[:2], (1 - a, a)),  edge_f)

# Get the DOFs on edge 1 (from vertex 0 (0,0) to vertex 2 (0,1), parametrised (0, a))
dofs = element.entity_dofs(0, 0) + element.entity_dofs(0, 2) + element.entity_dofs(1,␣
↪1)
for d, edge_f in zip(dofs, edge_basis):
    assert allequal(basis[d].subs(x[:2], (0, a)),  edge_f)

# Get the DOFs on edge 2 (from vertex 0 (0,0) to vertex 1 (1,0), parametrised (a, 0))
dofs = element.entity_dofs(0, 0) + element.entity_dofs(0, 1) + element.entity_dofs(1,␣
↪2)
for d, edge_f in zip(dofs, edge_basis):
    assert allequal(basis[d].subs(x[:2], (a, 0)),  edge_f)
```

Checking a Nedelec element

This demo shows how Symfem can be used to confirm that the polynomial set of a Nedelec first kind element are as expected.

```
"""Demo showing how Symfem can be used to verify properties of a basis.

The polynomial set of a degree k Nedelec first kind space is:
{polynomials of degree < k} UNION {polynomials of degree k such that p DOT x = 0}.

The basis functions of a Nedelec first kind that are associated with the interior of␣
↪the cell
have 0 tangential component on the facets of the cell.

In this demo, we verify that these properties hold for a degree 4 Nedelec first kind
space on a triangle.
"""

import symfem
from symfem.polynomials import polynomial_set_vector
from symfem.symbols import x
from symfem.utils import allequal

element = symfem.create_element("triangle", "Nedelec1", 4)
polys = element.get_polynomial_basis()

# Check that the first 20 polynomials in the polynomial basis are
# the polynomials of degree 3
p3 = polynomial_set_vector(2, 2, 3)
assert len(p3) == 20
for i, j in zip(p3, polys[:20]):
    assert i == j

# Check that the rest of the polynomials in the polynomial basis
# satisfy p DOT x = 0
for p in polys[20:]:
    assert p.dot(x[:2]) == 0

# Get the basis functions associated with the interior of the triangle
basis = element.get_basis_functions()
functions = [basis[d] for d in element.entity_dofs(2, 0)]

# Check that these functions have 0 tangential component on each edge
# allequal will simplify the expressions then check that they are equal
for f in functions:
    assert allequal(f.subs(x[0], 1 - x[1]).dot((1, -1)), 0)
    assert allequal(f.subs(x[0], 0).dot((0, 1)), 0)
    assert allequal(f.subs(x[1], 0).dot((1, 0)), 0)
```

Computing a stiffness matrix

This demo shows how a stiffness matrix over a simple mesh can be computed using Symfem.

```
"""Demo showing how Symfem can be used to compute a stiffness matrix."""

import symfem
from symfem.symbols import x

# Define the vertived and triangles of the mesh
vertices = [(0, 0), (1, 0), (0, 1), (1, 1)]
triangles = [[0, 1, 2], [1, 3, 2]]

# Create a matrix of zeros with the correct shape
matrix = [[0 for i in range(4)] for j in range(4)]

# Create a Lagrange element
element = symfem.create_element("triangle", "Lagrange", 1)

for triangle in triangles:
    # Get the vertices of the triangle
    vs = tuple(vertices[i] for i in triangle)
    # Create a reference cell with these vertices: this will be used
    # to compute the integral over the triangle
    ref = symfem.create_reference("triangle", vertices=vs)
    # Map the basis functions to the cell
    basis = element.map_to_cell(vs)

    for test_i, test_f in zip(triangle, basis):
        for trial_i, trial_f in zip(triangle, basis):
            # Compute the integral of grad(u)-dot-grad(v) for each pair of basis
            # functions u and v. The second input (x) into `ref.integral` tells
            # symfem which variables to use in the integral.
            integrand = test_f.grad(2).dot(trial_f.grad(2))
            print(integrand)
            matrix[test_i][trial_i] += integrand.integral(ref, x)

print(matrix)
```

Defining a custom element

This demo shows how a custom element with custom functionals can be defined in Symfem.

```
"""Demo showing how a custom element can be created in Symfem."""

import sympy

import symfem
from symfem.finite_element import CiarletElement
from symfem.functionals import PointEvaluation
from symfem.symbols import x


class CustomElement(CiarletElement):
    """Custom element on a quadrilateral."""

    def __init__(self, reference, order):
```

(continues on next page)

(continued from previous page)

```
        """Create the element.

        Args:
            reference: the reference element
            order: the polynomial order
        """
        zero = sympy.Integer(0)
        one = sympy.Integer(1)
        half = sympy.Rational(1, 2)

        # The polynomial set contains 1, x and y
        poly_set = [one, x[0], x[1]]

        # The DOFs are point evaluations at vertex 3,
        # and the midpoints of edges 0 and 1
        dofs = [
            PointEvaluation(reference, (one, one), entity=(0, 3)),
            PointEvaluation(reference, (half, zero), entity=(1, 0)),
            PointEvaluation(reference, (zero, half), entity=(1, 1)),
        ]

        super().__init__(reference, order, poly_set, dofs, reference.tdim, 1)

    names = ["custom quad element"]
    references = ["quadrilateral"]
    min_order = 1
    max_order = 1
    continuity = "L2"
    mapping = "identity"


# Add the element to symfem
symfem.add_element(CustomElement)

# Create the element and print its basis functions
element = symfem.create_element("quadrilateral", "custom quad element", 1)
print(element.get_basis_functions())

# Run the Symfem tests on the custom element
element.test()
```

1.2.2 API Reference

This page contains auto-generated API reference documentation[1].

[1] Created with sphinx-autoapi

symfem

Symfem: a symbolic finite element definition library.

Subpackages

symfem.elements

Definitions of symfem elements.

Submodules

symfem.elements._guzman_neilan_tetrahedron

Values for Guzman-Neilan element.

Module Contents

symfem.elements._guzman_neilan_tetrahedron.**coeffs = [None, None, None, None]**

symfem.elements._guzman_neilan_triangle

Values for Guzman-Neilan element.

Module Contents

symfem.elements._guzman_neilan_triangle.**coeffs = [None, None, None]**

symfem.elements.abf

Arnold-Boffi-Falk elements on quadrilaterals.

Thse elements definitions appear in https://dx.doi.org/10.1137/S0036142903431924 (Arnold, Boffi, Falk, 2005)

Module Contents

Classes

ArnoldBoffiFalk	An Arnold-Boffi-Falk element.

class symfem.elements.abf.**ArnoldBoffiFalk**(*reference:* symfem.references.Reference, *order: int*, *variant: str = 'equispaced'*)

Bases: *symfem.finite_element.CiarletElement*

An Arnold-Boffi-Falk element.

names = ['Arnold-Boffi-Falk', 'ABF']

references = ['quadrilateral']

min_order = 0

continuity = 'H(div)'

last_updated = '2023.06'

init_kwargs() → Dict[str, Any]

Return the kwargs used to create this element.

Returns

Keyword argument dictionary

symfem.elements.ac

Arbogast-Correa elements on quadrilaterals.

This element's definition appears in https://dx.doi.org/10.1137/15M1013705 (Arbogast, Correa, 2016)

Module Contents

Classes

AC	Arbogast-Correa Hdiv finite element.

class symfem.elements.ac.**AC**(*reference:* symfem.references.Reference, *order: int*, *variant: str* = *'equispaced'*)

Bases: *symfem.finite_element.CiarletElement*

Arbogast-Correa Hdiv finite element.

names = ['Arbogast-Correa', 'AC', 'AC full', 'Arbogast-Correa full']

references = ['quadrilateral']

min_order = 0

continuity = 'H(div)'

last_updated = '2023.06'

init_kwargs() → Dict[str, Any]

Return the kwargs used to create this element.

Returns

Keyword argument dictionary

symfem.elements.alfeld_sorokina

Alfeld-Sorokina element on a triangle.

This element's definition appears in https://doi.org/10.1007/s10543-015-0557-x (Alfeld, Sorokina, 2015)

Module Contents

Classes

AlfeldSorokina	Alfeld-Sorokina finite element.

class symfem.elements.alfeld_sorokina.**AlfeldSorokina**(*reference:* symfem.references.Reference, *order: int*)

Bases: *symfem.finite_element.CiarletElement*

Alfeld-Sorokina finite element.

names = ['Alfeld-Sorokina', 'AS']

references = ['triangle']

min_order = 2

max_order = 2

continuity = 'C0'

last_updated = '2023.05'

symfem.elements.argyris

Argyris elements on simplices.

This element's definition appears in https://doi.org/10.1017/S000192400008489X (Arygris, Fried, Scharpf, 1968)

Module Contents

Classes

Argyris	Argyris finite element.

class symfem.elements.argyris.**Argyris**(*reference:* symfem.references.Reference, *order: int*)

Bases: *symfem.finite_element.CiarletElement*

Argyris finite element.

names = ['Argyris']

references = ['triangle']

min_order = 5

max_order = 5

continuity = 'L2'

last_updated = '2023.05'

symfem.elements.aw

Arnold-Winther elements on simplices.

Thse elements definitions appear in https://doi.org/10.1007/s002110100348 (Arnold, Winther, 2002) [conforming] and https://doi.org/10.1142/S0218202503002507 (Arnold, Winther, 2003) [nonconforming]

Module Contents

Classes

ArnoldWinther	An Arnold-Winther element.
NonConformingArnoldWinther	A nonconforming Arnold-Winther element.

class symfem.elements.aw.**ArnoldWinther**(*reference:* symfem.references.Reference, *order: int*, *variant: str = 'equispaced'*)

Bases: *symfem.finite_element.CiarletElement*

An Arnold-Winther element.

names = ['Arnold-Winther', 'AW', 'conforming Arnold-Winther']

references = ['triangle']

min_order = 3

continuity = 'integral inner H(div)'

last_updated = '2023.05'

init_kwargs() → Dict[str, Any]

Return the kwargs used to create this element.

Returns
Keyword argument dictionary

class symfem.elements.aw.**NonConformingArnoldWinther**(*reference:* symfem.references.Reference, *order: int*, *variant: str = 'equispaced'*)

Bases: *symfem.finite_element.CiarletElement*

A nonconforming Arnold-Winther element.

names = ['nonconforming Arnold-Winther', 'nonconforming AW']

references = ['triangle']

min_order = 2

max_order = 2

continuity = 'integral inner H(div)'

last_updated = '2023.06'

init_kwargs() → Dict[str, Any]

Return the kwargs used to create this element.

Returns
Keyword argument dictionary

symfem.elements.bddm

Brezzi-Douglas-Duran-Fortin elements.

This element's definition appears in https://doi.org/10.1007/BF01396752 (Brezzi, Douglas, Duran, Fortin, 1987)

Module Contents

Classes

BDDF	Brezzi-Douglas-Duran-Fortin Hdiv finite element.

Functions

bddf_polyset(→ List[symfem.functions.FunctionInp	Create the polynomial basis for a BDDF element.

symfem.elements.bddm.**bddf_polyset**(*reference:* symfem.references.Reference, *order: int*) → List[symfem.functions.FunctionInput]

Create the polynomial basis for a BDDF element.

Parameters

- **reference** – The reference cell
- **order** – The polynomial order

Returns

The polynomial basis

class symfem.elements.bddm.**BDDF**(*reference:* symfem.references.Reference, *order: int*, *variant: str* = *'equispaced'*)

Bases: *symfem.finite_element.CiarletElement*

Brezzi-Douglas-Duran-Fortin Hdiv finite element.

names = ['Brezzi-Douglas-Duran-Fortin', 'BDDF']

references = ['hexahedron']

min_order = 1

continuity = 'H(div)'

last_updated = '2023.06'

init_kwargs() → Dict[str, Any]

Return the kwargs used to create this element.

Returns

Keyword argument dictionary

symfem.elements.bdfm

Brezzi-Douglas-Fortin-Marini elements.

This element's definition appears in https://doi.org/10.1051/m2an/1987210405811 (Brezzi, Douglas, Fortin, Marini, 1987)

Module Contents

Classes

BDFM	Brezzi-Douglas-Fortin-Marini Hdiv finite element.

Functions

bdfm_polyset(→ List[symfem.functions.FunctionInp	Create the polynomial basis for a BDFM element.

symfem.elements.bdfm.**bdfm_polyset**(*reference:* symfem.references.Reference, *order: int*) → List[symfem.functions.FunctionInput]

Create the polynomial basis for a BDFM element.

Parameters

- **reference** – The reference cell
- **order** – The polynomial order

Returns

The polynomial basis

class symfem.elements.bdfm.**BDFM**(*reference:* symfem.references.Reference, *order: int*, *variant: str* = *'equispaced'*)

Bases: *symfem.finite_element.CiarletElement*

Brezzi-Douglas-Fortin-Marini Hdiv finite element.

names = ['Brezzi-Douglas-Fortin-Marini', 'BDFM']

references = ['triangle', 'quadrilateral', 'hexahedron', 'tetrahedron']

min_order = 1

continuity = 'H(div)'

last_updated = '2023.06'

init_kwargs() → Dict[str, Any]

Return the kwargs used to create this element.

Returns

Keyword argument dictionary

symfem.elements.bdm

Brezzi-Douglas-Marini elements on simplices.

This element's definition appears in https://doi.org/10.1007/BF01389710 (Brezzi, Douglas, Marini, 1985)

Module Contents

Classes

BDM	Brezzi-Douglas-Marini Hdiv finite element.

class symfem.elements.bdm.**BDM**(*reference:* symfem.references.Reference, *order: int*, *variant: str* = *'equispaced'*)

Bases: *symfem.finite_element.CiarletElement*

Brezzi-Douglas-Marini Hdiv finite element.

names = ['Brezzi-Douglas-Marini', 'BDM', 'N2div']

references = ['triangle', 'tetrahedron']

min_order = 1

continuity = 'H(div)'

last_updated = '2023.06'

init_kwargs() → Dict[str, Any]

Return the kwargs used to create this element.

Returns
Keyword argument dictionary

symfem.elements.bell

Bell elements on triangle.

This element's definition is given in https://doi.org/10.1002/nme.1620010108 (Bell, 1969)

Module Contents

Classes

Bell	Bell finite element.

class symfem.elements.bell.**Bell**(*reference:* symfem.references.Reference, *order: int*, *variant: str* = *'equispaced'*)

Bases: *symfem.finite_element.CiarletElement*

Bell finite element.

names = ['Bell']

references = ['triangle']

min_order = 5

max_order = 5

continuity = 'C1'

last_updated = '2023.05'

init_kwargs() → Dict[str, Any]

Return the kwargs used to create this element.

Returns

Keyword argument dictionary

symfem.elements.bernardi_raugel

Bernardi-Raugel elements on simplices.

This element's definition appears in https://doi.org/10.2307/2007793 (Bernardi and Raugel, 1985)

Module Contents

Classes

BernardiRaugel	Bernardi-Raugel Hdiv finite element.

class symfem.elements.bernardi_raugel.**BernardiRaugel**(*reference:* symfem.references.Reference, *order: int*)

Bases: *symfem.finite_element.CiarletElement*

Bernardi-Raugel Hdiv finite element.

names = ['Bernardi-Raugel']

references = ['triangle', 'tetrahedron']

min_order = 1

max_order

continuity = 'H(div)'

last_updated = '2023.06'

symfem.elements.bernstein

Bernstein elements on simplices.

This element's definition appears in https://doi.org/10.1007/s00211-010-0327-2 (Kirby, 2011) and https://doi.org/10.1137/11082539X (Ainsworth, Andriamaro, Davydov, 2011)

Module Contents

Classes

BernsteinFunctional	Functional for a Bernstein element.
Bernstein	Bernstein finite element.

Functions

single_choose(→ sympy.core.expr.Expr)	Calculate choose function of a set of powers.
choose(→ sympy.core.expr.Expr)	Calculate choose function of a set of powers.
bernstein_polynomials(→ List[sympy.core.expr.Expr])	Return a list of Bernstein polynomials.

symfem.elements.bernstein.**single_choose**(*n: int*, *k: int*) → sympy.core.expr.Expr

Calculate choose function of a set of powers.

Parameters

- **n** – Number of items
- **k** – Number to select

Returns

Number of ways to pick k items from n items (ie n choose k)

symfem.elements.bernstein.**choose**(*n: int*, *powers: List[int]*) → sympy.core.expr.Expr

Calculate choose function of a set of powers.

Parameters

- **n** – Number of items
- **k** – Numbers to select

Returns

A multichoose function

symfem.elements.bernstein.**bernstein_polynomials**(*n: int*, *d: int*, *vars: symfem.symbols.AxisVariablesNotSingle = x*) → List[sympy.core.expr.Expr]

Return a list of Bernstein polynomials.

Parameters

- **n** – The polynomial order
- **d** – The topological dimension
- **vars** – The variables to use

Returns

Bernstein polynomials

class symfem.elements.bernstein.**BernsteinFunctional**(*reference: symfem.references.Reference*, *integral_domain: symfem.references.Reference*, *index: int*, *degree: int*, *entity: Tuple[int, int]*)

Bases: *symfem.functionals.BaseFunctional*

Functional for a Bernstein element.

dof_point() → symfem.geometry.PointType

Get the location of the DOF in the cell.

Returns
Location of the DOF

_eval_symbolic(*function:* symfem.functions.AnyFunction) → *symfem.functions.AnyFunction*

Apply the functional to a function.

Parameters
function – The function

Returns
Evaluation of the functional

get_tex() → Tuple[str, List[str]]

Get a representation of the functional as TeX, and list of terms involved.

Returns
TeX representation

class symfem.elements.bernstein.**Bernstein**(*reference:* symfem.references.Reference, *order: int*)

Bases: *symfem.finite_element.CiarletElement*

Bernstein finite element.

names = ['Bernstein', 'Bernstein-Bezier']

references = ['interval', 'triangle', 'tetrahedron']

min_order = 0

continuity = 'C0'

last_updated = '2023.05'

symfem.elements.bfs

Bogner-Fox-Schmit elements on tensor products.

This element's definition appears in http://contrails.iit.edu/reports/8569 (Bogner, Fox, Schmit, 1966)

Module Contents

Classes

BognerFoxSchmit	Bogner-Fox-Schmit finite element.

class symfem.elements.bfs.**BognerFoxSchmit**(*reference:* symfem.references.Reference, *order: int*)

Bases: *symfem.finite_element.CiarletElement*

Bogner-Fox-Schmit finite element.

names = ['Bogner-Fox-Schmit', 'BFS']

references = ['quadrilateral']

min_order = 3

max_order = 3

continuity = 'C0'

last_updated = '2023.05'

symfem.elements.bubble

Bubble elements on simplices.

This element's definition appears in https://doi.org/10.1007/978-3-642-23099-8_3 (Kirby, Logg, Rognes, Terrel, 2012)

Module Contents

Classes

Bubble	Bubble finite element.
BubbleEnrichedLagrange	Bubble enriched Lagrange element.
BubbleEnrichedVectorLagrange	Bubble enriched Lagrange element.

class symfem.elements.bubble.**Bubble**(*reference:* symfem.references.Reference, *order: int*, *variant: str = 'equispaced'*)

Bases: *symfem.finite_element.CiarletElement*

Bubble finite element.

names = ['bubble']

references = ['interval', 'triangle', 'tetrahedron', 'quadrilateral', 'hexahedron']

min_order

continuity = 'C0'

last_updated = '2023.09'

init_kwargs() → Dict[str, Any]

Return the kwargs used to create this element.

Returns

Keyword argument dictionary

class symfem.elements.bubble.**BubbleEnrichedLagrange**(*reference:* symfem.references.Reference, *order: int*, *variant: str = 'equispaced'*)

Bases: *symfem.finite_element.CiarletElement*

Bubble enriched Lagrange element.

names = ['bubble enriched Lagrange']

references = ['triangle']

min_order = 1

continuity = 'C0'

last_updated = '2023.09'

init_kwargs() → Dict[str, Any]

Return the kwargs used to create this element.

Returns
Keyword argument dictionary

class symfem.elements.bubble.**BubbleEnrichedVectorLagrange**(*reference: symfem.references.Reference*, *order: int*, *variant: str = 'equispaced'*)

Bases: *symfem.finite_element.CiarletElement*

Bubble enriched Lagrange element.

names = ['bubble enriched vector Lagrange']

references = ['triangle']

min_order = 1

continuity = 'C0'

last_updated = '2023.09'

init_kwargs() → Dict[str, Any]

Return the kwargs used to create this element.

Returns
Keyword argument dictionary

symfem.elements.conforming_crouzeix_raviart

Conforming Crouzeix-Raviart elements on simplices.

This element's definition appears in https://doi.org/10.1051/m2an/197307R300331 (Crouzeix, Raviart, 1973)

Module Contents

Classes

ConformingCrouzeixRaviart	Conforming Crouzeix-Raviart finite element.

class symfem.elements.conforming_crouzeix_raviart.**ConformingCrouzeixRaviart**(*reference: symfem.references.Reference*, *order: int*)

Bases: *symfem.finite_element.CiarletElement*

Conforming Crouzeix-Raviart finite element.

names = ['conforming Crouzeix-Raviart', 'conforming CR']

references = ['triangle']

min_order = 1

continuity = 'L2'

last_updated = '2023.05'

symfem.elements.crouzeix_raviart

Crouzeix-Raviart elements on simplices.

This element's definition appears in https://doi.org/10.1051/m2an/197307R300331 (Crouzeix, Raviart, 1973)

Module Contents

Classes

CrouzeixRaviart	Crouzeix-Raviart finite element.

class symfem.elements.crouzeix_raviart.**CrouzeixRaviart**(*reference: symfem.references.Reference*, *order: int*, *variant: str = 'equispaced'*)

Bases: *symfem.finite_element.CiarletElement*

Crouzeix-Raviart finite element.

names = ['Crouzeix-Raviart', 'CR', 'Crouzeix-Falk', 'CF']

references = ['triangle', 'tetrahedron']

min_order = 1

max_order

continuity = 'L2'

last_updated = '2023.05'

init_kwargs() → Dict[str, Any]

Return the kwargs used to create this element.

Returns

Keyword argument dictionary

symfem.elements.direct_serendipity

Serendipity elements on tensor product cells.

This element's definition appears in https://arxiv.org/abs/1809.02192 (Arbogast, Tao, 2018)

Module Contents

Classes

DirectSerendipity	A direct serendipity element.

class symfem.elements.direct_serendipity.**DirectSerendipity**(*reference: symfem.references.Reference*, *order: int*)

Bases: *symfem.finite_element.DirectElement*

A direct serendipity element.

names = ['direct serendipity']

references = ['quadrilateral']

min_order = 1

continuity = 'C0'

last_updated = '2023.05'

symfem.elements.dpc

DPC elements on tensor product cells.

Module Contents

Classes

DPC	A dPc element.
VectorDPC	Vector dPc finite element.

class symfem.elements.dpc.**DPC**(*reference:* symfem.references.Reference, *order: int*, *variant: str* = *'equispaced'*)

Bases: *symfem.finite_element.CiarletElement*

A dPc element.

names = ['dPc']

references = ['interval', 'quadrilateral', 'hexahedron']

min_order = 0

continuity = 'L2'

last_updated = '2023.07.1'

init_kwargs() → Dict[str, Any]

Return the kwargs used to create this element.

Returns

Keyword argument dictionary

class symfem.elements.dpc.**VectorDPC**(*reference:* symfem.references.Reference, *order: int*, *variant: str* = *'equispaced'*)

Bases: *symfem.finite_element.CiarletElement*

Vector dPc finite element.

names = ['vector dPc']

references = ['quadrilateral', 'hexahedron']

min_order = 0

continuity = 'L2'

last_updated = '2023.07'

init_kwargs() → Dict[str, Any]

Return the kwargs used to create this element.

Returns
Keyword argument dictionary

symfem.elements.dual

Dual elements.

These elements' definitions appear in https://doi.org/10.1016/j.crma.2004.12.022 (Buffa, Christiansen, 2005)

Module Contents

Classes

DualCiarletElement	Abstract barycentric finite element.
Dual	Barycentric dual finite element.
BuffaChristiansen	Buffa-Christiansen barycentric dual finite element.
RotatedBuffaChristiansen	RotatedBuffa-Christiansen barycentric dual finite element.

class symfem.elements.dual.**DualCiarletElement**(*dual_coefficients: List[List[List[int | sympy.core.expr.Expr]]]*, *fine_space: str*, *reference: symfem.references.DualPolygon*, *order: int*, *dof_entities: List[Tuple[int, int]]*, *domain_dim: int*, *range_dim: int*, *range_shape: Tuple[int, Ellipsis] | None = None*, *dof_directions: symfem.geometry.SetOfPoints | None = None*)

Bases: *symfem.finite_element.FiniteElement*

Abstract barycentric finite element.

abstract property maximum_degree: int

Get the maximum degree of this polynomial set for the element.

get_polynomial_basis(*reshape: bool = True*) → List[*symfem.functions.AnyFunction*]

Get the symbolic polynomial basis for the element.

Returns
The polynomial basis

get_dual_matrix() → sympy.matrices.dense.MutableDenseMatrix

Get the dual matrix.

Returns
The dual matrix

get_basis_functions(*use_tensor_factorisation: bool = False*) → List[*symfem.functions.AnyFunction*]

Get the basis functions of the element.

Parameters
use_tensor_factorisation – Should a tensor factorisation be used?

Returns
The basis functions

entity_dofs(*entity_dim: int*, *entity_number: int*) → List[int]

Get the numbers of the DOFs associated with the given entity.

Parameters

- **entity_dim** – The dimension of the entity
- **entity_number** – The number of the entity

Returns

The numbers of the DOFs associated with the entity

dof_plot_positions() → List[symfem.geometry.PointType]

Get the points to plot each DOF at on a DOF diagram.

Returns

The DOF positions

dof_directions() → List[symfem.geometry.PointType | None]

Get the direction associated with each DOF.

Returns

The DOF directions

dof_entities() → List[Tuple[int, int]]

Get the entities that each DOF is associated with.

Returns

The entities

abstract map_to_cell(*vertices_in: symfem.geometry.SetOfPointsInput*, *basis: List[symfem.functions.AnyFunction] | None = None*, *forward_map: symfem.geometry.PointType | None = None*, *inverse_map: symfem.geometry.PointType | None = None*) → List[*symfem.functions.AnyFunction*]

Map the basis onto a cell using the appropriate mapping for the element.

Parameters

- **vertices_in** – The vertices of the cell
- **basis** – The basis functions
- **forward_map** – The map from the reference to the cell
- **inverse_map** – The map to the reference from the cell

Returns

The basis functions mapped to the cell

class symfem.elements.dual.**Dual**(*reference: symfem.references.DualPolygon*, *order: int*)

Bases: *DualCiarletElement*

Barycentric dual finite element.

names = ['dual polynomial', 'dual P', 'dual']

references = ['dual polygon']

min_order = 0

max_order = 1

continuity = 'C0'

last_updated = '2023.05'

class symfem.elements.dual.**BuffaChristiansen**(*reference:* symfem.references.DualPolygon, *order: int*)

Bases: *DualCiarletElement*

Buffa-Christiansen barycentric dual finite element.

names = ['Buffa-Christiansen', 'BC']

references = ['dual polygon']

min_order = 1

max_order = 1

continuity = 'H(div)'

last_updated = '2023.05'

class symfem.elements.dual.**RotatedBuffaChristiansen**(*reference:* symfem.references.DualPolygon, *order: int*)

Bases: *DualCiarletElement*

RotatedBuffa-Christiansen barycentric dual finite element.

names = ['rotated Buffa-Christiansen', 'RBC']

references = ['dual polygon']

min_order = 1

max_order = 1

continuity = 'H(div)'

last_updated = '2023.05'

symfem.elements.enriched_galerkin

Enriched Galerkin elements.

This element's definition appears in https://doi.org/10.1137/080722953 (Sun, Liu, 2009).

Module Contents

Classes

EnrichedGalerkin	An enriched Galerkin element.

class symfem.elements.enriched_galerkin.**EnrichedGalerkin**(*reference:* symfem.references.Reference, *order: int*)

Bases: *symfem.finite_element.EnrichedElement*

An enriched Galerkin element.

names = ['enriched Galerkin', 'EG']

references = ['interval', 'triangle', 'quadrilateral', 'tetrahedron', 'hexahedron']

min_order = 1

continuity = 'C0'

last_updated = '2023.05'

symfem.elements.fortin_soulie

Fortin-Soulie elements on a triangle.

This element's definition appears in https://doi.org/10.1002/nme.1620190405 (Fortin, Soulie, 1973)

Module Contents

Classes

FortinSoulie	Fortin-Soulie finite element.

class symfem.elements.fortin_soulie.**FortinSoulie**(*reference:* symfem.references.Reference, *order:* *int*)

Bases: *symfem.finite_element.CiarletElement*

Fortin-Soulie finite element.

names = ['Fortin-Soulie', 'FS']

references = ['triangle']

min_order = 2

max_order = 2

continuity = 'L2'

last_updated = '2023.05'

symfem.elements.guzman_neilan

Guzman-Neilan elements on simplices.

This element's definition appears in https://doi.org/10.1137/17M1153467 (Guzman and Neilan, 2018)

Module Contents

Classes

GuzmanNeilan	Guzman-Neilan Hdiv finite element.

Functions

make_piecewise_lagrange(...)	Make the basis functions of a piecewise Lagrange space.

class symfem.elements.guzman_neilan.**GuzmanNeilan**(*reference:* symfem.references.Reference, *order: int*)

Bases: *symfem.finite_element.CiarletElement*

Guzman-Neilan Hdiv finite element.

names = ['Guzman-Neilan']

references = ['triangle', 'tetrahedron']

min_order = 1

max_order

continuity = 'H(div)'

last_updated = '2023.06'

_make_polyset_triangle(*reference:* symfem.references.Reference, *order: int*) → List[symfem.functions.FunctionInput]

Make the polyset for a triangle.

Parameters

- **reference** – The reference cell
- **order** – The polynomial order

Returns
The polynomial set

_make_polyset_tetrahedron(*reference:* symfem.references.Reference, *order: int*) → List[symfem.functions.FunctionInput]

Make the polyset for a tetrahedron.

Parameters

- **reference** – The reference cell
- **order** – The polynomial order

Returns
The polynomial set

symfem.elements.guzman_neilan.**make_piecewise_lagrange**(*sub_cells: List[symfem.geometry.SetOfPoints]*, *cell_name*, *order: int*, *zero_on_boundary: bool = False*, *zero_at_centre: bool = False*) → List[*symfem.piecewise_functions.PiecewiseFunction*]

Make the basis functions of a piecewise Lagrange space.

Parameters

- **sub_cells** – A list of vertices of sub cells
- **cell_name** – The cell type of the sub cells
- **order** – The polynomial order

- **zero_in_boundary** – Should the functions be zero on the boundary?
- **zero_at_centre** – Should the functions be zero at the centre?

Returns
The basis functions

symfem.elements.hct

Hsieh-Clough-Tocher elements on simplices.

This element's definition appears in https://doi.org/10.2307/2006147 (Ciarlet, 1978)

Module Contents

Classes

HsiehCloughTocher	Hsieh-Clough-Tocher finite element.

class symfem.elements.hct.**HsiehCloughTocher**(*reference:* symfem.references.Reference, *order: int*)

Bases: *symfem.finite_element.CiarletElement*

Hsieh-Clough-Tocher finite element.

names = ['Hsieh-Clough-Tocher', 'Clough-Tocher', 'HCT', 'CT']

references = ['triangle']

min_order = 3

max_order = 3

continuity = 'C0'

last_updated = '2023.06'

symfem.elements.hermite

Hermite elements on simplices.

This element's definition appears in https://doi.org/10.1016/0045-7825(72)90006-0 (Ciarlet, Raviart, 1972)

Module Contents

Classes

Hermite	Hermite finite element.

class symfem.elements.hermite.**Hermite**(*reference:* symfem.references.Reference, *order: int*)

Bases: *symfem.finite_element.CiarletElement*

Hermite finite element.

names = ['Hermite']

references = ['interval', 'triangle', 'tetrahedron']

min_order = 3

max_order = 3

continuity = 'C0'

last_updated = '2023.05'

symfem.elements.hhj

Hellan-Herrmann-Johnson elements on simplices.

This element's definition appears in https://arxiv.org/abs/1909.09687 (Arnold, Walker, 2020)

For an alternative construction see (Sinwel, 2009) and sections 4.4.2.2 and 4.4.3.2 https://numa.jku.at/media/filer_public/b7/42/b74263c9-f723-4076-b1b2-c2726126bf32/phd-sinwel.pdf

Module Contents

Classes

HellanHerrmannJohnson	A Hellan-Herrmann-Johnson element.

class symfem.elements.hhj.**HellanHerrmannJohnson**(*reference:* symfem.references.Reference, *order: int*, *variant: str = 'equispaced'*)

Bases: *symfem.finite_element.CiarletElement*

A Hellan-Herrmann-Johnson element.

names = ['Hellan-Herrmann-Johnson', 'HHJ']

references = ['triangle', 'tetrahedron']

min_order = 0

continuity = 'inner H(div)'

last_updated = '2023.08'

init_kwargs() → Dict[str, Any]

Return the kwargs used to create this element.

Returns
Keyword argument dictionary

symfem.elements.huang_zhang

Huang-Zhang element on a quadrilateral.

This element's definition appears in https://doi.org/10.1007/s11464-011-0094-0 (Huang, Zhang, 2011) and https://doi.org/10.1137/080728949 (Zhang, 2009)

Module Contents

Classes

HuangZhang	Huang-Zhang finite element.

class symfem.elements.huang_zhang.**HuangZhang**(*reference:* symfem.references.Reference, *order: int*, *variant: str = 'equispaced'*)

Bases: *symfem.finite_element.CiarletElement*

Huang-Zhang finite element.

names = ['Huang-Zhang', 'HZ']

references = ['quadrilateral']

min_order = 2

continuity = 'H(div)'

last_updated = '2023.06'

init_kwargs() → Dict[str, Any]

Return the kwargs used to create this element.

Returns

Keyword argument dictionary

symfem.elements.kmv

Kong-Mulder-Veldhuizen elements on triangle.

This element's definition is given in https://doi.org/10.1023/A:1004420829610 (Chin-Joe-Kong, Mulder, Van Veldhuizen, 1999)

Module Contents

Classes

KongMulderVeldhuizen	Kong-Mulder-Veldhuizen finite element.

Functions

kmv_tri_polyset(→ List[symfem.functions.FunctionInput])	Create the polynomial set for a KMV space on a triangle.
kmv_tet_polyset(→ List[symfem.functions.FunctionInput])	Create the polynomial set for a KMV space on a tetrahedron.

symfem.elements.kmv.**kmv_tri_polyset**(*m: int*, *mf: int*) → List[symfem.functions.FunctionInput]

Create the polynomial set for a KMV space on a triangle.

Parameters

- **m** – The parameter m
- **mf** – The parameter mf

Returns
The polynomial set

symfem.elements.kmv.**kmv_tet_polyset**(*m: int*, *mf: int*, *mi: int*) → List[symfem.functions.FunctionInput]

Create the polynomial set for a KMV space on a tetrahedron.

Parameters

- **m** – The parameter m
- **mf** – The parameter mf
- **mi** – The parameter mi

Returns
The polynomial set

class symfem.elements.kmv.**KongMulderVeldhuizen**(*reference:* symfem.references.Reference, *order: int*)

Bases: *symfem.finite_element.CiarletElement*

Kong-Mulder-Veldhuizen finite element.

names = ['Kong-Mulder-Veldhuizen', 'KMV']

references = ['triangle', 'tetrahedron']

min_order = 1

continuity = 'C0'

last_updated = '2023.05'

symfem.elements.lagrange

Lagrange elements on simplices.

Module Contents

Classes

Lagrange	Lagrange finite element.
VectorLagrange	Vector Lagrange finite element.
MatrixLagrange	Matrix Lagrange finite element.
SymmetricMatrixLagrange	Symmetric matrix Lagrange finite element.

class symfem.elements.lagrange.**Lagrange**(*reference:* symfem.references.Reference, *order: int*, *variant: str = 'equispaced'*)

Bases: *symfem.finite_element.CiarletElement*

Lagrange finite element.

names = ['Lagrange', 'P']

references = ['interval', 'triangle', 'tetrahedron']

min_order = 0

continuity = 'C0'

last_updated = '2023.09'

init_kwargs() → Dict[str, Any]

Return the kwargs used to create this element.

Returns
Keyword argument dictionary

class symfem.elements.lagrange.**VectorLagrange**(*reference:* symfem.references.Reference, *order: int*, *variant: str = 'equispaced'*)

Bases: *symfem.finite_element.CiarletElement*

Vector Lagrange finite element.

names = ['vector Lagrange', 'vP']

references = ['interval', 'triangle', 'tetrahedron']

min_order = 0

continuity = 'C0'

last_updated = '2023.09'

init_kwargs() → Dict[str, Any]

Return the kwargs used to create this element.

Returns
Keyword argument dictionary

class symfem.elements.lagrange.**MatrixLagrange**(*reference:* symfem.references.Reference, *order: int*, *variant: str = 'equispaced'*)

Bases: *symfem.finite_element.CiarletElement*

Matrix Lagrange finite element.

names = ['matrix Lagrange']

references = ['triangle', 'tetrahedron']

min_order = 0

continuity = 'L2'

last_updated = '2023.09'

init_kwargs() → Dict[str, Any]

Return the kwargs used to create this element.

Returns
Keyword argument dictionary

class symfem.elements.lagrange.**SymmetricMatrixLagrange**(*reference:* symfem.references.Reference, *order: int*, *variant: str = 'equispaced'*)

Bases: *symfem.finite_element.CiarletElement*

Symmetric matrix Lagrange finite element.

names = ['symmetric matrix Lagrange']

references = ['triangle', 'tetrahedron']

min_order = 0

continuity = 'L2'

last_updated = '2023.09'

init_kwargs() → Dict[str, Any]

Return the kwargs used to create this element.

Returns

Keyword argument dictionary

symfem.elements.lagrange_prism

Lagrange elements on a prism.

Module Contents

Classes

Lagrange	Lagrange finite element.
VectorLagrange	Vector Lagrange finite element.

class symfem.elements.lagrange_prism.**Lagrange**(*reference:* symfem.references.Reference, *order: int*, *variant: str = 'equispaced'*)

Bases: *symfem.finite_element.CiarletElement*

Lagrange finite element.

names = ['Lagrange', 'P']

references = ['prism']

min_order = 0

continuity = 'C0'

last_updated = '2023.07'

init_kwargs() → Dict[str, Any]

Return the kwargs used to create this element.

Returns

Keyword argument dictionary

class symfem.elements.lagrange_prism.**VectorLagrange**(*reference:* symfem.references.Reference, *order: int*, *variant: str = 'equispaced'*)

Bases: *symfem.finite_element.CiarletElement*

Vector Lagrange finite element.

names: List[str] = []

references = ['prism']

min_order = 0

continuity = 'C0'

last_updated = '2023.05'

init_kwargs() → Dict[str, Any]

Return the kwargs used to create this element.

Returns

Keyword argument dictionary

symfem.elements.lagrange_pyramid

Lagrange elements on a pyramid.

This element's definition appears in https://doi.org/10.1007/s10915-009-9334-9 (Bergot, Cohen, Durufle, 2010)

Module Contents

Classes

Lagrange	Lagrange finite element.

class symfem.elements.lagrange_pyramid.**Lagrange**(*reference:* symfem.references.Reference, *order:* *int*, *variant: str = 'equispaced'*)

Bases: *symfem.finite_element.CiarletElement*

Lagrange finite element.

names = ['Lagrange', 'P']

references = ['pyramid']

min_order = 0

continuity = 'C0'

last_updated = '2023.07'

init_kwargs() → Dict[str, Any]

Return the kwargs used to create this element.

Returns

Keyword argument dictionary

symfem.elements.morley

Morley elements on simplices.

This element's definition appears in https://doi.org/10.1017/S0001925900004546 (Morley, 1968)

Module Contents

Classes

Morley	Morley finite element.

class symfem.elements.morley.**Morley**(*reference:* symfem.references.Reference, *order: int*)

Bases: *symfem.finite_element.CiarletElement*

Morley finite element.

names = ['Morley']

references = ['triangle']

min_order = 2

max_order = 2

continuity = 'L2'

last_updated = '2023.05'

symfem.elements.morley_wang_xu

Morley-Wang-Xu elements on simplices.

This element's definition appears in https://doi.org/10.1090/S0025-5718-2012-02611-1 (Wang, Xu, 2013)

Module Contents

Classes

MorleyWangXu	Morley-Wang-Xu finite element.

class symfem.elements.morley_wang_xu.**MorleyWangXu**(*reference:* symfem.references.Reference, *order: int*)

Bases: *symfem.finite_element.CiarletElement*

Morley-Wang-Xu finite element.

names = ['Morley-Wang-Xu', 'MWX']

references = ['interval', 'triangle', 'tetrahedron']

min_order = 1

max_order

continuity = 'C0'

last_updated = '2023.06'

symfem.elements.mtw

Mardal-Tai-Winther elements on simplices.

This element's definition appears in https://doi.org/10.1137/S0036142901383910 (Mardal, Tai, Winther, 2002) and https://doi.org/10.1007/s10092-006-0124-6 (Tail, Mardal, 2006)

Module Contents

Classes

MardalTaiWinther	Mardal-Tai-Winther Hdiv finite element.

class symfem.elements.mtw.**MardalTaiWinther**(*reference:* symfem.references.Reference, *order: int*, *variant: str = 'equispaced'*)

Bases: *symfem.finite_element.CiarletElement*

Mardal-Tai-Winther Hdiv finite element.

names = ['Mardal-Tai-Winther', 'MTW']

references = ['triangle', 'tetrahedron']

min_order = 3

max_order = 3

continuity = 'H(div)'

last_updated = '2023.06'

init_kwargs() → Dict[str, Any]

Return the kwargs used to create this element.

Returns

Keyword argument dictionary

symfem.elements.nedelec

Nedelec elements on simplices.

These elements' definitions appear in https://doi.org/10.1007/BF01396415 (Nedelec, 1980) and https://doi.org/10.1007/BF01389668 (Nedelec, 1986)

Module Contents

Classes

NedelecFirstKind	Nedelec first kind Hcurl finite element.
NedelecSecondKind	Nedelec second kind Hcurl finite element.

class symfem.elements.nedelec.**NedelecFirstKind**(*reference:* symfem.references.Reference, *order:* *int*, *variant: str = 'equispaced'*)

Bases: *symfem.finite_element.CiarletElement*

Nedelec first kind Hcurl finite element.

names = ['Nedelec', 'Nedelec1', 'N1curl']

references = ['triangle', 'tetrahedron']

min_order = 1

continuity = 'H(curl)'

last_updated = '2023.06'

init_kwargs() → Dict[str, Any]

Return the kwargs used to create this element.

Returns
Keyword argument dictionary

class symfem.elements.nedelec.**NedelecSecondKind**(*reference:* symfem.references.Reference, *order:* *int*, *variant: str = 'equispaced'*)

Bases: *symfem.finite_element.CiarletElement*

Nedelec second kind Hcurl finite element.

names = ['Nedelec2', 'N2curl']

references = ['triangle', 'tetrahedron']

min_order = 1

continuity = 'H(curl)'

last_updated = '2023.06'

init_kwargs() → Dict[str, Any]

Return the kwargs used to create this element.

Returns
Keyword argument dictionary

symfem.elements.nedelec_prism

Nedelec elements on prisms.

Module Contents

Classes

Nedelec	Nedelec Hcurl finite element.

class symfem.elements.nedelec_prism.**Nedelec**(*reference:* symfem.references.Reference, *order: int*, *variant: str = 'equispaced'*)

Bases: *symfem.finite_element.CiarletElement*

Nedelec Hcurl finite element.

names = ['Nedelec', 'Ncurl']

references = ['prism']

min_order = 1

max_order = 2

continuity = 'H(curl)'

last_updated = '2023.06'

init_kwargs() → Dict[str, Any]

Return the kwargs used to create this element.

Returns

Keyword argument dictionary

symfem.elements.p1_iso_p2

P1-iso-P2 elements.

This element's definition appears in https://doi.org/10.1007/BF01399555 (Bercovier, Pironneau, 1979)

Module Contents

Classes

P1IsoP2Interval	P1-iso-P2 finite element on an interval.
P1IsoP2Tri	P1-iso-P2 finite element on a triangle.
P1IsoP2Quad	P1-iso-P2 finite element on a quadrilateral.

class symfem.elements.p1_iso_p2.**P1IsoP2Interval**(*reference:* symfem.references.Reference, *order: int*)

Bases: *symfem.finite_element.CiarletElement*

P1-iso-P2 finite element on an interval.

names = ['P1-iso-P2', 'P2-iso-P1', 'iso-P2 P1']

references = ['interval']

min_order = 1

max_order = 1

continuity = 'C0'

last_updated = '2023.08'

class symfem.elements.p1_iso_p2.**P1IsoP2Tri**(*reference:* symfem.references.Reference, *order: int*)

Bases: *symfem.finite_element.CiarletElement*

P1-iso-P2 finite element on a triangle.

names = ['P1-iso-P2', 'P2-iso-P1', 'iso-P2 P1']

references = ['triangle']

min_order = 1

max_order = 1

continuity = 'C0'

last_updated = '2023.06'

class symfem.elements.p1_iso_p2.**P1IsoP2Quad**(*reference:* symfem.references.Reference, *order: int*)

Bases: *symfem.finite_element.CiarletElement*

P1-iso-P2 finite element on a quadrilateral.

names = ['P1-iso-P2', 'P2-iso-P1', 'iso-P2 P1']

references = ['quadrilateral']

min_order = 1

max_order = 1

continuity = 'C0'

last_updated = '2023.06'

symfem.elements.p1_macro

P1 macro elements.

This element's definition appears in https://doi.org/10.1007/s00211-018-0970-6 (Christiansen, Hu, 2018)

Module Contents

Classes

P1Macro	P1 macro finite element on a triangle.

class symfem.elements.p1_macro.**P1Macro**(*reference:* symfem.references.Reference, *order: int*)

Bases: *symfem.finite_element.CiarletElement*

P1 macro finite element on a triangle.

names = ['P1 macro']

references = ['triangle']

min_order = 1

max_order = 1

continuity = 'C0'

last_updated = '2023.06'

symfem.elements.q

Q elements on tensor product cells.

Module Contents

Classes

Q	A Q element.
VectorQ	A vector Q element.
Nedelec	Nedelec Hcurl finite element.
RaviartThomas	Raviart-Thomas Hdiv finite element.

class symfem.elements.q.**Q**(*reference:* symfem.references.Reference, *order: int*, *variant: str* = *'equispaced'*)

Bases: *symfem.finite_element.CiarletElement*

A Q element.

names = ['Q', 'Lagrange', 'P']

references = ['quadrilateral', 'hexahedron']

min_order = 0

continuity = 'C0'

last_updated = '2023.07.1'

get_tensor_factorisation() → List[Tuple[str, List[*symfem.finite_element.FiniteElement*], List[int]]]

Get the representation of the element as a tensor product.

Returns
The tensor factorisation

init_kwargs() → Dict[str, Any]

Return the kwargs used to create this element.

Returns
Keyword argument dictionary

class symfem.elements.q.**VectorQ**(*reference:* symfem.references.Reference, *order: int*, *variant: str* = *'equispaced'*)

Bases: *symfem.finite_element.CiarletElement*

A vector Q element.

names = ['vector Q', 'vQ']

references = ['quadrilateral', 'hexahedron']

min_order = 0

continuity = 'C0'

last_updated = '2023.06'

init_kwargs() → Dict[str, Any]

Return the kwargs used to create this element.

Returns
Keyword argument dictionary

class symfem.elements.q.**Nedelec**(*reference:* symfem.references.Reference, *order: int*, *variant: str = 'equispaced'*)

Bases: *symfem.finite_element.CiarletElement*

Nedelec Hcurl finite element.

names = ['NCE', 'RTCE', 'Qcurl', 'Nedelec', 'Ncurl']

references = ['quadrilateral', 'hexahedron']

min_order = 1

continuity = 'H(curl)'

last_updated = '2023.06'

init_kwargs() → Dict[str, Any]

Return the kwargs used to create this element.

Returns
Keyword argument dictionary

class symfem.elements.q.**RaviartThomas**(*reference:* symfem.references.Reference, *order: int*, *variant: str = 'equispaced'*)

Bases: *symfem.finite_element.CiarletElement*

Raviart-Thomas Hdiv finite element.

names = ['NCF', 'RTCF', 'Qdiv']

references = ['quadrilateral', 'hexahedron']

min_order = 1

continuity = 'H(div)'

last_updated = '2023.06'

init_kwargs() → Dict[str, Any]

Return the kwargs used to create this element.

Returns
Keyword argument dictionary

symfem.elements.rannacher_turek

Rannacher-Turek elements on tensor product cells.

This element's definition appears in https://doi.org/10.1002/num.1690080202 (Rannacher, Turek, 1992)

Module Contents

Classes

RannacherTurek	Rannacher-Turek finite element.

class symfem.elements.rannacher_turek.**RannacherTurek**(*reference:* symfem.references.Reference, *order: int*)

Bases: *symfem.finite_element.CiarletElement*

Rannacher-Turek finite element.

names = ['Rannacher-Turek']

references = ['quadrilateral', 'hexahedron']

min_order = 1

max_order = 1

continuity = 'L2'

last_updated = '2023.05'

init_kwargs() → Dict[str, Any]

Return the kwargs used to create this element.

Returns

Keyword argument dictionary

symfem.elements.regge

Regge elements on simplices.

This element's definition appears in https://doi.org/10.1007/BF02733251 (Regge, 1961), https://doi.org/10.1007/s00211-011-0394-z (Christiansen, 2011), and http://aurora.asc.tuwien.ac.at/~mneunteu/thesis/doctorthesis_neunteufel.pdf (Neunteufel, 2021)

Module Contents

Classes

Regge	A Regge element on a simplex.
ReggeTP	A Regge element on a tensor product cell.

class symfem.elements.regge.**Regge**(*reference:* symfem.references.Reference, *order: int*, *variant: str = 'point'*)

Bases: *symfem.finite_element.CiarletElement*

A Regge element on a simplex.

names = ['Regge']

references = ['triangle', 'tetrahedron']

min_order = 0

continuity = 'inner H(curl)'

last_updated = '2023.06'

init_kwargs() → Dict[str, Any]

Return the kwargs used to create this element.

Returns
Keyword argument dictionary

class symfem.elements.regge.**ReggeTP**(*reference:* symfem.references.Reference, *order: int*, *variant: str = 'integral'*)

Bases: *symfem.finite_element.CiarletElement*

A Regge element on a tensor product cell.

names = ['Regge']

references = ['quadrilateral', 'hexahedron']

min_order = 0

continuity = 'inner H(curl)'

last_updated = '2023.06'

init_kwargs() → Dict[str, Any]

Return the kwargs used to create this element.

Returns
Keyword argument dictionary

symfem.elements.rhct

Reduced Hsieh-Clough-Tocher elements on simplices.

This element's definition appears in https://doi.org/10.2307/2006147 (Ciarlet, 1978)

Module Contents

Classes

ReducedHsiehCloughTocher	Reduced Hsieh-Clough-Tocher finite element.

class symfem.elements.rhct.**ReducedHsiehCloughTocher**(*reference:* symfem.references.Reference, *order: int*)

Bases: *symfem.finite_element.CiarletElement*

Reduced Hsieh-Clough-Tocher finite element.

names = ['reduced Hsieh-Clough-Tocher', 'rHCT']

references = ['triangle']

min_order = 3

max_order = 3

continuity = 'C0'

last_updated = '2023.06'

symfem.elements.rt

Raviart-Thomas elements on simplices.

This element's definition appears in https://doi.org/10.1007/BF01396415 (Nedelec, 1980)

Module Contents

Classes

RaviartThomas	Raviart-Thomas Hdiv finite element.

class symfem.elements.rt.**RaviartThomas**(*reference:* symfem.references.Reference, *order:* *int*, *variant:* *str = 'equispaced'*)

Bases: *symfem.finite_element.CiarletElement*

Raviart-Thomas Hdiv finite element.

names = ['Raviart-Thomas', 'RT', 'N1div']

references = ['triangle', 'tetrahedron']

min_order = 1

continuity = 'H(div)'

last_updated = '2023.06'

init_kwargs() → Dict[str, Any]

Return the kwargs used to create this element.

Returns
Keyword argument dictionary

symfem.elements.serendipity

Serendipity elements on tensor product cells.

This element's definition appears in https://doi.org/10.1007/s10208-011-9087-3 (Arnold, Awanou, 2011)

Module Contents

Classes

Serendipity	A serendipity element.
SerendipityCurl	A serendipity Hcurl element.
SerendipityDiv	A serendipity Hdiv element.

class symfem.elements.serendipity.**Serendipity**(*reference:* symfem.references.Reference, *order: int*, *variant: str = 'equispaced'*)

Bases: *symfem.finite_element.CiarletElement*

A serendipity element.

names = ['serendipity', 'S']

references = ['interval', 'quadrilateral', 'hexahedron']

min_order = 1

continuity = 'C0'

last_updated = '2023.06'

init_kwargs() → Dict[str, Any]

Return the kwargs used to create this element.

Returns
Keyword argument dictionary

class symfem.elements.serendipity.**SerendipityCurl**(*reference:* symfem.references.Reference, *order: int*, *variant: str = 'equispaced'*)

Bases: *symfem.finite_element.CiarletElement*

A serendipity Hcurl element.

names = ['serendipity Hcurl', 'Scurl', 'BDMCE', 'AAE']

references = ['quadrilateral', 'hexahedron']

min_order = 1

continuity = 'H(curl)'

last_updated = '2023.07'

init_kwargs() → Dict[str, Any]

Return the kwargs used to create this element.

Returns
Keyword argument dictionary

class symfem.elements.serendipity.**SerendipityDiv**(*reference:* symfem.references.Reference, *order: int*, *variant: str = 'equispaced'*)

Bases: *symfem.finite_element.CiarletElement*

A serendipity Hdiv element.

names = ['serendipity Hdiv', 'Sdiv', 'BDMCF', 'AAF']

references = ['quadrilateral', 'hexahedron']

min_order = 1

continuity = 'H(div)'

last_updated = '2023.07'

init_kwargs() → Dict[str, Any]

Return the kwargs used to create this element.

Returns
Keyword argument dictionary

symfem.elements.taylor

Taylor element on an interval, triangle or tetrahedron.

Module Contents

Classes

Taylor	Taylor finite element.

class symfem.elements.taylor.**Taylor**(*reference:* symfem.references.Reference, *order: int*)

Bases: *symfem.finite_element.CiarletElement*

Taylor finite element.

names = ['Taylor', 'discontinuous Taylor']

references = ['interval', 'triangle', 'tetrahedron']

min_order = 0

continuity = 'L2'

last_updated = '2023.06'

symfem.elements.tnt

TiNiest Tensor product (TNT) elements.

These elements' definitions appear in https://doi.org/10.1090/S0025-5718-2013-02729-9 (Cockburn, Qiu, 2013)

Module Contents

Classes

TNT	TiNiest Tensor scalar finite element.
TNTcurl	TiNiest Tensor Hcurl finite element.
TNTdiv	TiNiest Tensor Hdiv finite element.

Functions

p(→ symfem.functions.ScalarFunction)	Return the kth Legendre polynomial.
b(→ symfem.functions.ScalarFunction)	Return the function B_k.

symfem.elements.tnt.**p**(*k: int*, *v: sympy.core.symbol.Symbol*) → *symfem.functions.ScalarFunction*

Return the kth Legendre polynomial.

Parameters

- **k** – k
- **v** – The variable to use

Returns
The kth Legendre polynomial

symfem.elements.tnt.**b**(*k: int*, *v: sympy.core.symbol.Symbol*) → *symfem.functions.ScalarFunction*

Return the function B_k.

This function is defined on page 4 (606) of https://doi.org/10.1090/S0025-5718-2013-02729-9 (Cockburn, Qiu, 2013).

Parameters

- **k** – k
- **v** – The variable to use

Returns
The function B_k

class symfem.elements.tnt.**TNT**(*reference: symfem.references.Reference*, *order: int*, *variant: str = 'equispaced'*)

Bases: *symfem.finite_element.CiarletElement*

TiNiest Tensor scalar finite element.

names = ['tiniest tensor', 'TNT']

references = ['quadrilateral', 'hexahedron']

min_order = 1

continuity = 'C0'

last_updated = '2023.06'

init_kwargs() → Dict[str, Any]

Return the kwargs used to create this element.

Returns
Keyword argument dictionary

class symfem.elements.tnt.**TNTcurl**(*reference: symfem.references.Reference*, *order: int*, *variant: str = 'equispaced'*)

Bases: *symfem.finite_element.CiarletElement*

TiNiest Tensor Hcurl finite element.

names = ['tiniest tensor Hcurl', 'TNTcurl']

references = ['quadrilateral', 'hexahedron']

min_order = 1

continuity = 'H(curl)'

last_updated = '2023.06'

init_kwargs() → Dict[str, Any]

Return the kwargs used to create this element.

Returns
Keyword argument dictionary

class symfem.elements.tnt.**TNTdiv**(*reference: symfem.references.Reference*, *order: int*, *variant: str = 'equispaced'*)

Bases: *symfem.finite_element.CiarletElement*

TiNiest Tensor Hdiv finite element.

names = ['tiniest tensor Hdiv', 'TNTdiv']

references = ['quadrilateral', 'hexahedron']

min_order = 1

continuity = 'H(div)'

last_updated = '2023.06'

init_kwargs() → Dict[str, Any]

Return the kwargs used to create this element.

Returns

Keyword argument dictionary

symfem.elements.transition

Transition elements on simplices.

Module Contents

Classes

Transition	Transition finite element.

class symfem.elements.transition.**Transition**(*reference:* symfem.references.Reference, *order: int*, *edge_orders: List[int] | None = None*, *face_orders: List[int] | None = None*, *variant: str = 'equispaced'*)

Bases: *symfem.finite_element.CiarletElement*

Transition finite element.

names = ['transition']

references = ['triangle', 'tetrahedron']

min_order = 1

continuity = 'C0'

last_updated = '2023.06'

init_kwargs() → Dict[str, Any]

Return the kwargs used to create this element.

Returns

Keyword argument dictionary

symfem.elements.trimmed_serendipity

Trimmed serendipity elements on tensor products.

These elements' definitions appear in https://doi.org/10.1137/16M1073352 (Cockburn, Fu, 2017) and https://doi.org/10.1090/mcom/3354 (Gilette, Kloefkorn, 2018)

Module Contents

Classes

TrimmedSerendipityHcurl	Trimmed serendipity Hcurl finite element.
TrimmedSerendipityHdiv	Trimmed serendipity Hdiv finite element.

class symfem.elements.trimmed_serendipity.**TrimmedSerendipityHcurl**(*reference:* symfem.references.Reference, *order: int*, *variant: str = 'equispaced'*)

Bases: *symfem.finite_element.CiarletElement*

Trimmed serendipity Hcurl finite element.

names = ['trimmed serendipity Hcurl', 'TScurl']

references = ['quadrilateral', 'hexahedron']

min_order = 1

continuity = 'H(curl)'

last_updated = '2023.06'

init_kwargs() → Dict[str, Any]

Return the kwargs used to create this element.

Returns

Keyword argument dictionary

class symfem.elements.trimmed_serendipity.**TrimmedSerendipityHdiv**(*reference:* symfem.references.Reference, *order: int*, *variant: str = 'equispaced'*)

Bases: *symfem.finite_element.CiarletElement*

Trimmed serendipity Hdiv finite element.

names = ['trimmed serendipity Hdiv', 'TSdiv']

references = ['quadrilateral', 'hexahedron']

min_order = 1

continuity = 'H(div)'

last_updated = '2023.06'

init_kwargs() → Dict[str, Any]

Return the kwargs used to create this element.

Returns

Keyword argument dictionary

symfem.elements.vector_enriched_galerkin

Enriched vector Galerkin elements.

This element's definition appears in https://doi.org/10.1016/j.camwa.2022.06.018 (Yi, Hu, Lee, Adler, 2022)

Module Contents

Classes

Enrichment	An LF enriched Galerkin element.
VectorEnrichedGalerkin	An LF enriched Galerkin element.

class symfem.elements.vector_enriched_galerkin.**Enrichment**(*reference:* symfem.references.Reference)

Bases: *symfem.finite_element.CiarletElement*

An LF enriched Galerkin element.

names: List[str] = []

references = ['triangle', 'quadrilateral', 'tetrahedron', 'hexahedron']

min_order = 1

max_order = 1

continuity = 'C0'

last_updated = '2023.05'

class symfem.elements.vector_enriched_galerkin.**VectorEnrichedGalerkin**(*reference:* symfem.references.Reference, *order: int*)

Bases: *symfem.finite_element.EnrichedElement*

An LF enriched Galerkin element.

names = ['enriched vector Galerkin', 'locking-free enriched Galerkin', 'LFEG']

references = ['triangle', 'quadrilateral', 'tetrahedron', 'hexahedron']

min_order = 1

continuity = 'C0'

last_updated = '2023.05'

symfem.elements.wu_xu

Wu-Xu elements on simplices.

This element's definition appears in https://doi.org/10.1090/mcom/3361 (Wu, Xu, 2019)

Module Contents

Classes

WuXu	Wu-Xu finite element.

Functions

derivatives(→ List[Tuple[int, Ellipsis]])	Return all the orders of a multidimensional derivative.

symfem.elements.wu_xu.**derivatives**(*dim: int*, *order: int*) → List[Tuple[int, Ellipsis]]

Return all the orders of a multidimensional derivative.

Parameters

- **dim** – The topological dimension
- **order** – The total derivative order

Returns

List of derivative order tuples

class symfem.elements.wu_xu.**WuXu**(*reference:* symfem.references.Reference, *order: int*)

Bases: *symfem.finite_element.CiarletElement*

Wu-Xu finite element.

names = ['Wu-Xu']

references = ['interval', 'triangle', 'tetrahedron']

min_order

max_order

continuity = 'C0'

last_updated = '2023.06.1'

symfem.polynomials

Polynomials.

Submodules

symfem.polynomials.dual

Dual polynomials.

Module Contents

Functions

l2_dual(→ List[symfem.functions.ScalarFunction])	Compute the L2 dual of a set of polynomials.

symfem.polynomials.dual.**l2_dual**(*cell: str*, *poly: List[*symfem.functions.ScalarFunction*]*) → List[*symfem.functions.ScalarFunction*]

Compute the L2 dual of a set of polynomials.

Parameters

- **cell** – The cell type
- **poly** – The set of polynomial

Returns

The L2 dual polynomials

symfem.polynomials.legendre

Orthogonal (Legendre) polynomials.

Module Contents

Functions

_jrc(→ Tuple[sympy.core.expr.Expr, ...)	Get the Jacobi recurrence relation coefficients.
orthogonal_basis_interval(...)	Create a basis of orthogonal polynomials.
orthogonal_basis_triangle(...)	Create a basis of orthogonal polynomials.
orthogonal_basis_quadrilateral(...)	Create a basis of orthogonal polynomials.
orthogonal_basis_tetrahedron(...)	Create a basis of orthogonal polynomials.
orthogonal_basis_hexahedron(...)	Create a basis of orthogonal polynomials.
orthogonal_basis_prism(...)	Create a basis of orthogonal polynomials.
orthogonal_basis_pyramid(...)	Create a basis of orthogonal polynomials.
orthogonal_basis(...)	Create a basis of orthogonal polynomials.
orthonormal_basis(...)	Create a basis of orthonormal polynomials.

symfem.polynomials.legendre.**_jrc**(*a*, *n*) → Tuple[sympy.core.expr.Expr, sympy.core.expr.Expr, sympy.core.expr.Expr]

Get the Jacobi recurrence relation coefficients.

Parameters

- **a** – The parameter a
- **n** – The parameter n

Returns

The Jacobi coefficients

symfem.polynomials.legendre.**orthogonal_basis_interval**(*order: int*, *derivs: int*, *variables: symfem.symbols.AxisVariablesNotSingle = [x[0]]*) → List[List[*symfem.functions.ScalarFunction*]]

Create a basis of orthogonal polynomials.

Parameters

- **order** – The maximum polynomial degree
- **derivs** – The number of derivatives to include
- **variables** – The variables to use

Returns

A set of orthogonal polynomials

symfem.polynomials.legendre.**orthogonal_basis_triangle**(*order: int*, *derivs: int*, *variables: symfem.symbols.AxisVariablesNotSingle = [x[0], x[1]]*) → List[List[*symfem.functions.ScalarFunction*]]

Create a basis of orthogonal polynomials.

Parameters

- **order** – The maximum polynomial degree
- **derivs** – The number of derivatives to include
- **variables** – The variables to use

Returns

A set of orthogonal polynomials

symfem.polynomials.legendre.**orthogonal_basis_quadrilateral**(*order: int*, *derivs: int*, *variables: symfem.symbols.AxisVariablesNotSingle = [x[0], x[1]]*) → List[List[*symfem.functions.ScalarFunction*]]

Create a basis of orthogonal polynomials.

Parameters

- **order** – The maximum polynomial degree
- **derivs** – The number of derivatives to include
- **variables** – The variables to use

Returns

A set of orthogonal polynomials

symfem.polynomials.legendre.**orthogonal_basis_tetrahedron**(*order: int*, *derivs: int*, *variables: symfem.symbols.AxisVariablesNotSingle = x*) → List[List[*symfem.functions.ScalarFunction*]]

Create a basis of orthogonal polynomials.

Parameters

- **order** – The maximum polynomial degree
- **derivs** – The number of derivatives to include
- **variables** – The variables to use

Returns

A set of orthogonal polynomials

symfem.polynomials.legendre.**orthogonal_basis_hexahedron**(*order: int*, *derivs: int*, *variables: symfem.symbols.AxisVariablesNotSingle = x*) → List[List[*symfem.functions.ScalarFunction*]]

Create a basis of orthogonal polynomials.

Parameters

- **order** – The maximum polynomial degree
- **derivs** – The number of derivatives to include
- **variables** – The variables to use

Returns

A set of orthogonal polynomials

symfem.polynomials.legendre.**orthogonal_basis_prism**(*order: int*, *derivs: int*, *variables: symfem.symbols.AxisVariablesNotSingle = x*) → List[List[*symfem.functions.ScalarFunction*]]

Create a basis of orthogonal polynomials.

Parameters

- **order** – The maximum polynomial degree
- **derivs** – The number of derivatives to include
- **variables** – The variables to use

Returns

A set of orthogonal polynomials

symfem.polynomials.legendre.**orthogonal_basis_pyramid**(*order: int*, *derivs: int*, *variables: symfem.symbols.AxisVariablesNotSingle = x*) → List[List[*symfem.functions.ScalarFunction*]]

Create a basis of orthogonal polynomials.

Parameters

- **order** – The maximum polynomial degree
- **derivs** – The number of derivatives to include
- **variables** – The variables to use

Returns

A set of orthogonal polynomials

symfem.polynomials.legendre.**orthogonal_basis**(*cell: str*, *order: int*, *derivs: int*, *variables: symfem.symbols.AxisVariablesNotSingle | None = None*) → List[List[*symfem.functions.ScalarFunction*]]

Create a basis of orthogonal polynomials.

Parameters

- **cell** – The cell type
- **order** – The maximum polynomial degree
- **derivs** – The number of derivatives to include
- **variables** – The variables to use

Returns

A set of orthogonal polynomials

symfem.polynomials.legendre.**orthonormal_basis**(*cell: str*, *order: int*, *derivs: int*, *variables: symfem.symbols.AxisVariablesNotSingle | None = None*) → List[List[*symfem.functions.ScalarFunction*]]

Create a basis of orthonormal polynomials.

Parameters

- **cell** – The cell type
- **order** – The maximum polynomial degree
- **derivs** – The number of derivatives to include
- **variables** – The variables to use

Returns
A set of orthonormal polynomials

symfem.polynomials.lobatto

Lobatto polynomials.

Module Contents

Functions

lobatto_basis_interval(...)	Get Lobatto polynomials on an interval.
lobatto_dual_basis_interval(...)	Get L2 dual of Lobatto polynomials on an interval.
lobatto_basis(→ List[symfem.functions.ScalarFunc	Get Lobatto polynomials.
lobatto_dual_basis(→ List[symfem.functions.ScalarFunction])	Get L2 dual of Lobatto polynomials.

symfem.polynomials.lobatto.**lobatto_basis_interval**(*order: int*) → List[*symfem.functions.ScalarFunction*]

Get Lobatto polynomials on an interval.

Parameters
order – The maximum polynomial degree

Returns
Lobatto polynomials

symfem.polynomials.lobatto.**lobatto_dual_basis_interval**(*order: int*) → List[*symfem.functions.ScalarFunction*]

Get L2 dual of Lobatto polynomials on an interval.

Parameters
order – The maximum polynomial degree

Returns
Dual Lobatto polynomials

symfem.polynomials.lobatto.**lobatto_basis**(*cell: str*, *order: int*, *include_endpoints: bool = True*) → List[*symfem.functions.ScalarFunction*]

Get Lobatto polynomials.

Parameters

- **cell** – The cell type
- **order** – The maximum polynomial degree
- **include_endpoint** – should polynomials that are non-zero on the boundary be included?

Returns
Lobatto polynomials

symfem.polynomials.lobatto.**lobatto_dual_basis**(*cell: str*, *order: int*, *include_endpoints: bool = True*) → List[*symfem.functions.ScalarFunction*]

Get L2 dual of Lobatto polynomials.

Parameters

- **cell** – The cell type
- **order** – The maximum polynomial degree
- **include_endpoint** – should polynomials that are non-zero on the boundary be included?

Returns
Lobatto polynomials

symfem.polynomials.polysets

Polynomial sets.

Module Contents

Functions

polynomial_set_1d(→ List[symfem.functions.ScalarFunction])	One dimensional polynomial set.
polynomial_set_vector(...)	Polynomial set.
Hdiv_polynomials(→ List[symfem.functions.VectorFunction])	Hdiv conforming polynomial set.
Hcurl_polynomials(→ List[symfem.functions.VectorFunction])	Hcurl conforming polynomial set.
quolynomial_set_1d(→ List[symfem.functions.ScalarFunction])	One dimensional quolynomial set.
quolynomial_set_vector(...)	Quolynomial set.
Hdiv_quolynomials(→ List[symfem.functions.VectorFunction])	Hdiv conforming quolynomial set.
Hcurl_quolynomials(→ List[symfem.functions.VectorFunction])	Hcurl conforming quolynomial set.
serendipity_indices(→ List[List[int]])	Get the set indices for a serendipity polynomial set.
serendipity_set_1d(→ List[symfem.functions.ScalarFunction])	One dimensional serendipity set.
serendipity_set_vector(...)	Serendipity set.
Hdiv_serendipity(→ List[symfem.functions.VectorFunction])	Hdiv conforming serendipity set.
Hcurl_serendipity(→ List[symfem.functions.VectorFunction])	Hcurl conforming serendipity set.
prism_polynomial_set_1d(...)	One dimensional polynomial set.
prism_polynomial_set_vector(...)	Polynomial set for a prism.
pyramid_polynomial_set_1d(...)	One dimensional polynomial set.
pyramid_polynomial_set_vector(...)	Polynomial set for a pyramid.

symfem.polynomials.polysets.**polynomial_set_1d**(*dim: int*, *order: int*, *variables: symfem.symbols.AxisVariablesNotSingle = x*) → List[*symfem.functions.ScalarFunction*]

One dimensional polynomial set.

Parameters

- **dim** – The number of variables
- **order** – The maximum polynomial degree
- **variables** – The variables to use

Returns
A set of polynomials

symfem.polynomials.polysets.**polynomial_set_vector**(*domain_dim: int*, *range_dim: int*, *order: int*, *variables: symfem.symbols.AxisVariablesNotSingle = x*) → List[*symfem.functions.VectorFunction*]

Polynomial set.

Parameters

- **domain_dim** – The number of variables
- **range_dim** – The dimension of the output vector
- **order** – The maximum polynomial degree
- **variables** – The variables to use

Returns
A set of polynomials

symfem.polynomials.polysets.**Hdiv_polynomials**(*domain_dim: int*, *range_dim: int*, *order: int*, *variables: symfem.symbols.AxisVariablesNotSingle = x*) → List[*symfem.functions.VectorFunction*]

Hdiv conforming polynomial set.

Parameters

- **domain_dim** – The number of variables
- **range_dim** – The dimension of the output vector
- **order** – The maximum polynomial degree
- **variables** – The variables to use

Returns
A set of polynomials

symfem.polynomials.polysets.**Hcurl_polynomials**(*domain_dim: int*, *range_dim: int*, *order: int*, *variables: symfem.symbols.AxisVariablesNotSingle = x*) → List[*symfem.functions.VectorFunction*]

Hcurl conforming polynomial set.

Parameters

- **domain_dim** – The number of variables
- **range_dim** – The dimension of the output vector
- **order** – The maximum polynomial degree
- **variables** – The variables to use

Returns
A set of polynomials

symfem.polynomials.polysets.**quolynomial_set_1d**(*dim: int*, *order: int*, *variables: symfem.symbols.AxisVariablesNotSingle = x*) → List[*symfem.functions.ScalarFunction*]

One dimensional quolynomial set.

Parameters

- **dim** – The number of variables
- **order** – The maximum polynomial degree
- **variables** – The variables to use

Returns
A set of polynomials

symfem.polynomials.polysets.**quolynomial_set_vector**(*domain_dim: int*, *range_dim: int*, *order: int*, *variables: symfem.symbols.AxisVariablesNotSingle = x*) → List[*symfem.functions.VectorFunction*]

Quolynomial set.

Parameters

- **domain_dim** – The number of variables
- **range_dim** – The dimension of the output vector
- **order** – The maximum polynomial degree
- **variables** – The variables to use

Returns
A set of polynomials

symfem.polynomials.polysets.**Hdiv_quolynomials**(*domain_dim: int*, *range_dim: int*, *order: int*, *variables: symfem.symbols.AxisVariablesNotSingle = x*) → List[*symfem.functions.VectorFunction*]

Hdiv conforming quolynomial set.

Parameters

- **domain_dim** – The number of variables
- **range_dim** – The dimension of the output vector
- **order** – The maximum polynomial degree
- **variables** – The variables to use

Returns
A set of polynomials

symfem.polynomials.polysets.**Hcurl_quolynomials**(*domain_dim: int*, *range_dim: int*, *order: int*, *variables: symfem.symbols.AxisVariablesNotSingle = x*) → List[*symfem.functions.VectorFunction*]

Hcurl conforming quolynomial set.

Parameters

- **domain_dim** – The number of variables
- **range_dim** – The dimension of the output vector
- **order** – The maximum polynomial degree
- **variables** – The variables to use

Returns
A set of polynomials

symfem.polynomials.polysets.**serendipity_indices**(*total: int*, *linear: int*, *dim: int*, *done: List[int] | None = None*) → List[List[int]]

Get the set indices for a serendipity polynomial set.

Parameters

- **dim** – The number of variables
- **order** – The maximum polynomial degree
- **variables** – The variables to use

Returns
A set of polynomials

symfem.polynomials.polysets.**serendipity_set_1d**(*dim: int*, *order: int*, *variables: symfem.symbols.AxisVariablesNotSingle = x*) → List[*symfem.functions.ScalarFunction*]

One dimensional serendipity set.

Parameters

- **dim** – The number of variables
- **order** – The maximum polynomial degree
- **variables** – The variables to use

Returns
A set of polynomials

symfem.polynomials.polysets.**serendipity_set_vector**(*domain_dim: int*, *range_dim: int*, *order: int*, *variables: symfem.symbols.AxisVariablesNotSingle = x*) → List[*symfem.functions.VectorFunction*]

Serendipity set.

Parameters

- **domain_dim** – The number of variables
- **range_dim** – The dimension of the output vector
- **order** – The maximum polynomial degree
- **variables** – The variables to use

Returns
A set of polynomials

symfem.polynomials.polysets.**Hdiv_serendipity**(*domain_dim: int*, *range_dim: int*, *order: int*, *variables: symfem.symbols.AxisVariablesNotSingle = x*) → List[*symfem.functions.VectorFunction*]

Hdiv conforming serendipity set.

Parameters

- **domain_dim** – The number of variables
- **range_dim** – The dimension of the output vector
- **order** – The maximum polynomial degree
- **variables** – The variables to use

Returns
A set of polynomials

symfem.polynomials.polysets.**Hcurl_serendipity**(*domain_dim: int*, *range_dim: int*, *order: int*, *variables: symfem.symbols.AxisVariablesNotSingle = x*) → List[*symfem.functions.VectorFunction*]

Hcurl conforming serendipity set.

Parameters

- **domain_dim** – The number of variables
- **range_dim** – The dimension of the output vector
- **order** – The maximum polynomial degree
- **variables** – The variables to use

Returns
A set of polynomials

symfem.polynomials.polysets.**prism_polynomial_set_1d**(*dim: int*, *order: int*, *variables: symfem.symbols.AxisVariablesNotSingle = x*) → List[*symfem.functions.ScalarFunction*]

One dimensional polynomial set.

Parameters

- **dim** – The number of variables
- **order** – The maximum polynomial degree
- **variables** – The variables to use

Returns
A set of polynomials

symfem.polynomials.polysets.**prism_polynomial_set_vector**(*domain_dim: int*, *range_dim: int*, *order: int*, *variables: symfem.symbols.AxisVariablesNotSingle = x*) → List[*symfem.functions.VectorFunction*]

Polynomial set for a prism.

Parameters

- **domain_dim** – The number of variables
- **range_dim** – The dimension of the output vector
- **order** – The maximum polynomial degree
- **variables** – The variables to use

Returns
A set of polynomials

symfem.polynomials.polysets.**pyramid_polynomial_set_1d**(*dim: int*, *order: int*, *variables: symfem.symbols.AxisVariablesNotSingle = x*) → List[*symfem.functions.ScalarFunction*]

One dimensional polynomial set.

Parameters

- **dim** – The number of variables
- **order** – The maximum polynomial degree
- **variables** – The variables to use

Returns
A set of polynomials

symfem.polynomials.polysets.**pyramid_polynomial_set_vector**(*domain_dim: int*, *range_dim: int*, *order: int*, *variables: symfem.symbols.AxisVariablesNotSingle = x*) → List[*symfem.functions.VectorFunction*]

Polynomial set for a pyramid.

Parameters

- **domain_dim** – The number of variables
- **range_dim** – The dimension of the output vector
- **order** – The maximum polynomial degree
- **variables** – The variables to use

Returns
A set of polynomials

Package Contents

Functions

l2_dual(→ List[symfem.functions.ScalarFunction])	Compute the L2 dual of a set of polynomials.
orthogonal_basis(...)	Create a basis of orthogonal polynomials.
orthonormal_basis(...)	Create a basis of orthonormal polynomials.
lobatto_basis(→ List[symfem.functions.ScalarFunc	Get Lobatto polynomials.
lobatto_dual_basis(→ List[symfem.functions.ScalarFunction])	Get L2 dual of Lobatto polynomials.
Hcurl_polynomials(→ List[symfem.functions.VectorFunction])	Hcurl conforming polynomial set.
Hcurl_quolynomials(→ List[symfem.functions.VectorFunction])	Hcurl conforming quolynomial set.
Hcurl_serendipity(→ List[symfem.functions.VectorFunction])	Hcurl conforming serendipity set.
Hdiv_polynomials(→ List[symfem.functions.VectorFunction])	Hdiv conforming polynomial set.
Hdiv_quolynomials(→ List[symfem.functions.VectorFunction])	Hdiv conforming quolynomial set.
Hdiv_serendipity(→ List[symfem.functions.VectorFunction])	Hdiv conforming serendipity set.
polynomial_set_1d(→ List[symfem.functions.ScalarFunction])	One dimensional polynomial set.
polynomial_set_vector(...)	Polynomial set.
prism_polynomial_set_1d(...)	One dimensional polynomial set.
prism_polynomial_set_vector(...)	Polynomial set for a prism.
pyramid_polynomial_set_1d(...)	One dimensional polynomial set.
pyramid_polynomial_set_vector(...)	Polynomial set for a pyramid.
quolynomial_set_1d(→ List[symfem.functions.ScalarFunction])	One dimensional quolynomial set.
quolynomial_set_vector(...)	Quolynomial set.
serendipity_indices(→ List[List[int]])	Get the set indices for a serendipity polynomial set.
serendipity_set_1d(→ List[symfem.functions.ScalarFunction])	One dimensional serendipity set.
serendipity_set_vector(...)	Serendipity set.

symfem.polynomials.**l2_dual**(*cell: str*, *poly: List[*symfem.functions.ScalarFunction*]*) → List[*symfem.functions.ScalarFunction*]

Compute the L2 dual of a set of polynomials.

Parameters

- **cell** – The cell type
- **poly** – The set of polynomial

Returns
The L2 dual polynomials

symfem.polynomials.**orthogonal_basis**(*cell: str*, *order: int*, *derivs: int*, *variables: symfem.symbols.AxisVariablesNotSingle | None = None*) → List[List[*symfem.functions.ScalarFunction*]]

Create a basis of orthogonal polynomials.

Parameters

- **cell** – The cell type
- **order** – The maximum polynomial degree
- **derivs** – The number of derivatives to include
- **variables** – The variables to use

Returns
A set of orthogonal polynomials

symfem.polynomials.**orthonormal_basis**(*cell: str*, *order: int*, *derivs: int*, *variables: symfem.symbols.AxisVariablesNotSingle | None = None*) → List[List[*symfem.functions.ScalarFunction*]]

Create a basis of orthonormal polynomials.

Parameters

- **cell** – The cell type
- **order** – The maximum polynomial degree
- **derivs** – The number of derivatives to include
- **variables** – The variables to use

Returns
A set of orthonormal polynomials

symfem.polynomials.**lobatto_basis**(*cell: str*, *order: int*, *include_endpoints: bool = True*) → List[*symfem.functions.ScalarFunction*]

Get Lobatto polynomials.

Parameters

- **cell** – The cell type
- **order** – The maximum polynomial degree
- **include_endpoint** – should polynomials that are non-zero on the boundary be included?

Returns
Lobatto polynomials

symfem.polynomials.**lobatto_dual_basis**(*cell: str*, *order: int*, *include_endpoints: bool = True*) → List[*symfem.functions.ScalarFunction*]

Get L2 dual of Lobatto polynomials.

Parameters

- **cell** – The cell type
- **order** – The maximum polynomial degree
- **include_endpoint** – should polynomials that are non-zero on the boundary be included?

Returns
Lobatto polynomials

symfem.polynomials.**Hcurl_polynomials**(*domain_dim: int*, *range_dim: int*, *order: int*, *variables: symfem.symbols.AxisVariablesNotSingle = x*) → List[*symfem.functions.VectorFunction*]

Hcurl conforming polynomial set.

Parameters

- **domain_dim** – The number of variables
- **range_dim** – The dimension of the output vector
- **order** – The maximum polynomial degree
- **variables** – The variables to use

Returns
A set of polynomials

symfem.polynomials.**Hcurl_quolynomials**(*domain_dim: int*, *range_dim: int*, *order: int*, *variables: symfem.symbols.AxisVariablesNotSingle = x*) → List[*symfem.functions.VectorFunction*]

Hcurl conforming quolynomial set.

Parameters

- **domain_dim** – The number of variables
- **range_dim** – The dimension of the output vector
- **order** – The maximum polynomial degree
- **variables** – The variables to use

Returns
A set of polynomials

symfem.polynomials.**Hcurl_serendipity**(*domain_dim: int*, *range_dim: int*, *order: int*, *variables: symfem.symbols.AxisVariablesNotSingle = x*) → List[*symfem.functions.VectorFunction*]

Hcurl conforming serendipity set.

Parameters

- **domain_dim** – The number of variables
- **range_dim** – The dimension of the output vector
- **order** – The maximum polynomial degree
- **variables** – The variables to use

Returns
A set of polynomials

symfem.polynomials.**Hdiv_polynomials**(*domain_dim: int*, *range_dim: int*, *order: int*, *variables: symfem.symbols.AxisVariablesNotSingle = x*) → List[*symfem.functions.VectorFunction*]

Hdiv conforming polynomial set.

Parameters

- **domain_dim** – The number of variables
- **range_dim** – The dimension of the output vector
- **order** – The maximum polynomial degree
- **variables** – The variables to use

Returns
A set of polynomials

symfem.polynomials.**Hdiv_quolynomials**(*domain_dim: int*, *range_dim: int*, *order: int*, *variables: symfem.symbols.AxisVariablesNotSingle = x*) → List[*symfem.functions.VectorFunction*]

Hdiv conforming quolynomial set.

Parameters

- **domain_dim** – The number of variables
- **range_dim** – The dimension of the output vector
- **order** – The maximum polynomial degree
- **variables** – The variables to use

Returns
A set of polynomials

symfem.polynomials.**Hdiv_serendipity**(*domain_dim: int*, *range_dim: int*, *order: int*, *variables: symfem.symbols.AxisVariablesNotSingle = x*) → List[*symfem.functions.VectorFunction*]

Hdiv conforming serendipity set.

Parameters

- **domain_dim** – The number of variables
- **range_dim** – The dimension of the output vector
- **order** – The maximum polynomial degree
- **variables** – The variables to use

Returns
A set of polynomials

symfem.polynomials.**polynomial_set_1d**(*dim: int*, *order: int*, *variables: symfem.symbols.AxisVariablesNotSingle = x*) → List[*symfem.functions.ScalarFunction*]

One dimensional polynomial set.

Parameters

- **dim** – The number of variables
- **order** – The maximum polynomial degree
- **variables** – The variables to use

Returns
A set of polynomials

symfem.polynomials.**polynomial_set_vector**(*domain_dim: int*, *range_dim: int*, *order: int*, *variables: symfem.symbols.AxisVariablesNotSingle = x*) → List[*symfem.functions.VectorFunction*]

Polynomial set.

Parameters

- **domain_dim** – The number of variables
- **range_dim** – The dimension of the output vector
- **order** – The maximum polynomial degree
- **variables** – The variables to use

Returns
A set of polynomials

symfem.polynomials.**prism_polynomial_set_1d**(*dim: int*, *order: int*, *variables: symfem.symbols.AxisVariablesNotSingle = x*) → List[*symfem.functions.ScalarFunction*]

One dimensional polynomial set.

Parameters

- **dim** – The number of variables
- **order** – The maximum polynomial degree
- **variables** – The variables to use

Returns
A set of polynomials

symfem.polynomials.**prism_polynomial_set_vector**(*domain_dim: int*, *range_dim: int*, *order: int*, *variables: symfem.symbols.AxisVariablesNotSingle = x*) → List[*symfem.functions.VectorFunction*]

Polynomial set for a prism.

Parameters

- **domain_dim** – The number of variables
- **range_dim** – The dimension of the output vector
- **order** – The maximum polynomial degree
- **variables** – The variables to use

Returns
A set of polynomials

symfem.polynomials.**pyramid_polynomial_set_1d**(*dim: int*, *order: int*, *variables: symfem.symbols.AxisVariablesNotSingle = x*) → List[*symfem.functions.ScalarFunction*]

One dimensional polynomial set.

Parameters

- **dim** – The number of variables
- **order** – The maximum polynomial degree
- **variables** – The variables to use

Returns
A set of polynomials

symfem.polynomials.**pyramid_polynomial_set_vector**(*domain_dim: int*, *range_dim: int*, *order: int*, *variables: symfem.symbols.AxisVariablesNotSingle = x*) → List[*symfem.functions.VectorFunction*]

Polynomial set for a pyramid.

Parameters

- **domain_dim** – The number of variables

- **range_dim** – The dimension of the output vector
- **order** – The maximum polynomial degree
- **variables** – The variables to use

Returns
A set of polynomials

symfem.polynomials.**quolynomial_set_1d**(*dim: int*, *order: int*, *variables: symfem.symbols.AxisVariablesNotSingle = x*) → List[*symfem.functions.ScalarFunction*]

One dimensional quolynomial set.

Parameters

- **dim** – The number of variables
- **order** – The maximum polynomial degree
- **variables** – The variables to use

Returns
A set of polynomials

symfem.polynomials.**quolynomial_set_vector**(*domain_dim: int*, *range_dim: int*, *order: int*, *variables: symfem.symbols.AxisVariablesNotSingle = x*) → List[*symfem.functions.VectorFunction*]

Quolynomial set.

Parameters

- **domain_dim** – The number of variables
- **range_dim** – The dimension of the output vector
- **order** – The maximum polynomial degree
- **variables** – The variables to use

Returns
A set of polynomials

symfem.polynomials.**serendipity_indices**(*total: int*, *linear: int*, *dim: int*, *done: List[int] | None = None*) → List[List[int]]

Get the set indices for a serendipity polynomial set.

Parameters

- **dim** – The number of variables
- **order** – The maximum polynomial degree
- **variables** – The variables to use

Returns
A set of polynomials

symfem.polynomials.**serendipity_set_1d**(*dim: int*, *order: int*, *variables: symfem.symbols.AxisVariablesNotSingle = x*) → List[*symfem.functions.ScalarFunction*]

One dimensional serendipity set.

Parameters

- **dim** – The number of variables
- **order** – The maximum polynomial degree
- **variables** – The variables to use

Returns
A set of polynomials

symfem.polynomials.**serendipity_set_vector**(*domain_dim: int*, *range_dim: int*, *order: int*, *variables: symfem.symbols.AxisVariablesNotSingle = x*) → List[*symfem.functions.VectorFunction*]

Serendipity set.

Parameters

- **domain_dim** – The number of variables
- **range_dim** – The dimension of the output vector
- **order** – The maximum polynomial degree
- **variables** – The variables to use

Returns
A set of polynomials

Submodules

symfem.basis_functions

Abstract basis function classes and functions.

Module Contents

Classes

BasisFunction	A basis function of a finite element.
SubbedBasisFunction	A basis function following a substitution.

class symfem.basis_functions.**BasisFunction**(*scalar=False*, *vector=False*, *matrix=False*)

Bases: *symfem.functions.AnyFunction*

A basis function of a finite element.

This basis function can be used before the element's basis functions have been computed. When the explicit basis function is needed, only then will it be computed.

abstract get_function() → *symfem.functions.AnyFunction*

Get the actual basis function.

Returns
The basis function

__add__(*other: Any*) → *symfem.functions.AnyFunction*

Add.

Parameters
other – A function to add to this function.

Returns
The sum of two functions

__radd__(*other: Any*) → *symfem.functions.AnyFunction*

Add.

Parameters
other – A function to add to this function.

Returns
The sum of two functions

__sub__(*other: Any*) → *symfem.functions.AnyFunction*

Subtract.

Parameters
other – A function to subtract this function.

Returns
The difference of two functions

__rsub__(*other: Any*) → *symfem.functions.AnyFunction*

Subtract.

Parameters
other – A function to subtract this function from.

Returns
The difference of two functions

__neg__() → *symfem.functions.AnyFunction*

Negate.

Returns
Negated function

__truediv__(*other: Any*) → *symfem.functions.AnyFunction*

Divide.

Parameters
other – A function to divide this function by.

Returns
The ratio of two functions

__rtruediv__(*other: Any*) → *symfem.functions.AnyFunction*

Divide.

Parameters
other – A function to divide by this function.

Returns
The ratio of two functions

__mul__(*other: Any*) → *symfem.functions.AnyFunction*

Multiply.

Parameters
other – A function to multiply by this function.

Returns
The product of two functions

__rmul__(*other: Any*) → *symfem.functions.AnyFunction*

Multiply.

Parameters
other – A function to multiply by this function.

Returns
The product of two functions

__matmul__(*other: Any*) → *symfem.functions.AnyFunction*

Multiply.

Parameters
other – A function to matrix multiply by this function.

Returns
The product of two matrix functions

__rmatmul__(*other: Any*) → *symfem.functions.AnyFunction*

Multiply.

Parameters
other – A function to matrix multiply by this function.

Returns
The product of two matrix functions

__pow__(*other: Any*) → *symfem.functions.AnyFunction*

Raise to a power.

Parameters
other – A power to raise this function to.

Returns
This function to the power of other

as_sympy() → symfem.functions.SympyFormat

Convert to a sympy expression.

Returns
This function as a Sympy expression

as_tex() → str

Convert to a TeX expression.

Returns
A TeX representation of this function

diff(*variable: sympy.core.symbol.Symbol*) → *symfem.functions.AnyFunction*

Differentiate the function.

Parameters
variable – The variable to differentiate with respect to

Returns
The derivative

directional_derivative(*direction: symfem.geometry.PointType*) → *symfem.functions.AnyFunction*

Compute a directional derivative.

Parameters
direction – The direction of the derivative

Returns
The derivative

jacobian_component(*component: Tuple[int, int]*) → *symfem.functions.AnyFunction*

Compute a component of the jacobian.

Parameters
component – The component to compute

Returns
Component of the Jacobian

jacobian(*dim: int*) → *symfem.functions.AnyFunction*

Compute the jacobian.

Parameters
dim – The dimension of the Jacobian

Returns
The Jabobian

dot(*other_in: symfem.functions.FunctionInput*) → *symfem.functions.AnyFunction*

Compute the dot product with another function.

Parameters
other_in – The function to dot with

Returns
The dot product

cross(*other_in: symfem.functions.FunctionInput*) → *symfem.functions.AnyFunction*

Compute the cross product with another function.

Parameters
other_in – The function to cross with

Returns
The cross product

div() → *symfem.functions.AnyFunction*

Compute the divergence of the function.

Returns
The divergence

grad(*dim: int*) → *symfem.functions.AnyFunction*

Compute the gradient of the function.

Returns
The gradient

curl() → *symfem.functions.AnyFunction*

Compute the curl of the function.

Returns
The curl

norm() → *symfem.functions.ScalarFunction*

Compute the norm of the function.

Returns
The norm

integral(*domain:* symfem.references.Reference, *vars: symfem.symbols.AxisVariablesNotSingle = x*, *dummy_vars: symfem.symbols.AxisVariablesNotSingle = t*) → *symfem.functions.ScalarFunction*

Compute the integral of the function.

Parameters

- **domain** – The domain of the integral
- **vars** – The variables to integrate with respect to
- **dummy_vars** – The dummy variables to use inside the integral

Returns
The integral

subs(*vars: symfem.symbols.AxisVariables*, *values: symfem.functions.ValuesToSubstitute*) → *BasisFunction*

Substitute values into the function.

Parameters

- **vars** – The variable(s) to substitute
- **values** – The value(s) to substitute

Returns
Substituted function

__getitem__(*key*) → *symfem.functions.AnyFunction*

Forward all other function calls to symbolic function.

__len__() → int

Get length.

Returns
The length

det() → *symfem.functions.ScalarFunction*

Compute the determinant.

Returns
The determinant

transpose() → *symfem.functions.ScalarFunction*

Compute the transpose.

Returns
The transpose

with_floats() → *symfem.functions.AnyFunction*

Return a version the function with floats as coefficients.

Returns
The function with floats as coefficients

__iter__() → Iterator[*symfem.functions.AnyFunction*]

Iterate through components of vector function.

maximum_degree(*cell:* symfem.references.Reference) → int

Return the maximum degree of the function on a reference cell.

This function returns the order of the lowerst order Lagrange space on the input cell that includes this function.

Parameters
cell – The cell

Returns
A version the function with floats as coefficients

class symfem.basis_functions.**SubbedBasisFunction**(*f:* BasisFunction, *vars: symfem.symbols.AxisVariables*, *values: symfem.functions.ValuesToSubstitute*)

Bases: *BasisFunction*

A basis function following a substitution.

property shape: Tuple[int, Ellipsis]

Get the value shape of the function.

Returns
shape

get_function() → *symfem.functions.AnyFunction*

Return the symbolic function.

Returns
Symbolic function

plot_values(*reference:* symfem.references.Reference, *img: Any*, *value_scale: sympy.core.expr.Expr = sympy.Integer(1)*, *n: int = 6*)

Plot the function's values.

Parameters

- **reference** – The domain to plot the function on
- **img** – The image to plot the values on
- **value_scale** – The factor to scale values by
- **n** – The number of plotting points

maximum_degree(*cell:* symfem.references.Reference) → int

Return the maximum degree of the function on a reference cell.

This function returns the order of the lowerst order Lagrange space on the input cell that includes this function.

Parameters
cell – The cell

Returns
A version the function with floats as coefficients

symfem.caching

Functions to cache matrices.

Module Contents

Functions

load_cached_matrix(...)	Load a cached matrix.
save_cached_matrix(matrix_type, cache_id, matrix)	Save a matrix to the cache.
matrix_to_string(→ str)	Convert a matrix to a string.
matrix_from_string(...)	Convert a string to a matrix.

Attributes

CACHE_DIR
CACHE_FORMAT

symfem.caching.**CACHE_DIR**

symfem.caching.**CACHE_FORMAT = '1'**

symfem.caching.**load_cached_matrix**(*matrix_type: str*, *cache_id: str*, *size: Tuple[int, int]*) → sympy.matrices.dense.MutableDenseMatrix | None

Load a cached matrix.

Parameters

- **matrix_type** – The type of the matrix. This will be included in the filename.
- **cache_id** – The unique identifier of the matrix within this type

Returns

The matrix

symfem.caching.**save_cached_matrix**(*matrix_type: str*, *cache_id: str*, *matrix: sympy.matrices.dense.MutableDenseMatrix*)

Save a matrix to the cache.

Parameters

- **matrix_type** – The type of the matrix. This will be included in the filename.
- **cache_id** – The unique identifier of the matrix within this type
- **matrix** – The matrix

symfem.caching.**matrix_to_string**(*m: sympy.matrices.dense.MutableDenseMatrix*) → str

Convert a matrix to a string.

Parameters

m – The matrix

Returns

A representation of the matrix as a string

symfem.caching.**matrix_from_string**(*mstr: str*) → sympy.matrices.dense.MutableDenseMatrix

Convert a string to a matrix.

Parameters

mstr – The string in the format output by *matrix_to_string*

Returns

The matrix

symfem.create

Create elements and references.

Module Contents

Functions

add_element(element_class)	Add an element to Symfem.
create_reference(→ sym-fem.references.Reference)	Make a reference cell.
create_element(→ sym-fem.finite_element.FiniteElement)	Make a finite element.
_order_is_allowed(→ bool)	Check that an order is valid for an element.

Attributes

_folder
_elementmap
_elementlist
_fname

symfem.create.**_folder**

symfem.create.**_elementmap: Dict[str, Dict[str, Type]]**

symfem.create.**_elementlist: List[Type] = []**

symfem.create.**add_element**(*element_class: Type*)

Add an element to Symfem.

Parameters

element_class – The class defining the element.

symfem.create.**_fname**

symfem.create.**create_reference**(*cell_type: str*, *vertices: symfem.geometry.SetOfPointsInput | None = None*) → *symfem.references.Reference*

Make a reference cell.

Parameters

- **cell_type** – The reference cell type. Supported values: point, interval, triangle, quadrilateral, tetrahedron, hexahedron, prism, pyramid, dual polygon(number_of_triangles)
- **vertices** – The vertices of the reference.

symfem.create.**create_element**(*cell_type: str*, *element_type: str*, *order: int*, *vertices: symfem.geometry.SetOfPointsInput | None = None*, ***kwargs: Any*) → *symfem.finite_element.FiniteElement*

Make a finite element.

Parameters

- **cell_type** – The reference cell type. Supported values: point, interval, triangle, quadrilateral, tetrahedron, hexahedron, prism, pyramid, dual polygon(number_of_triangles)
- **element_type** – The type of the element. Supported values: Lagrange, P, vector Lagrange, vP, matrix Lagrange, symmetric matrix Lagrange, dPc, vector dPc, Crouzeix-Raviart, CR, Crouzeix-Falk, CF, conforming Crouzeix-Raviart, conforming CR, serendipity, S, serendipity Hcurl, Scurl, BDMCE, AAE, serendipity Hdiv, Sdiv, BDMCF, AAF, direct serendipity, Regge, Nedelec, Nedelec1, N1curl, Ncurl, Nedelec2, N2curl, Raviart-Thomas, RT, N1div, Brezzi-Douglas-Marini, BDM, N2div, Q, vector Q, vQ, NCE, RTCE, Qcurl, NCF, RTCF, Qdiv, Morley, Morley-Wang-Xu, MWX, Hermite, Mardal-Tai-Winther, MTW, Argyris, bubble, dual polynomial, dual P, dual, Buffa-Christiansen, BC, rotated Buffa-Christiansen, RBC, Brezzi-Douglas-Fortin-Marini, BDFM, Brezzi-Douglas-Duran-Fortin, BDDF, Hellan-Herrmann-Johnson, HHJ, Arnold-Winther, AW, conforming Arnold-Winther, Bell, Kong-Mulder-Veldhuizen, KMV, Bernstein, Bernstein-Bezier, Hsieh-Clough-Tocher, Clough-Tocher, HCT, CT, reduced Hsieh-Clough-Tocher, rHCT, Taylor, discontinuous Taylor, bubble enriched Lagrange, bubble enriched vector Lagrange, Bogner-Fox-Schmit, BFS, Fortin-Soulie, FS, Bernardi-Raugel, Wu-Xu, transition, Guzman-Neilan, nonconforming Arnold-Winther, nonconforming AW, TScurl, trimmed serendipity Hcurl, TSdiv, trimmed serendipity Hdiv, TNT, tiniest tensor, TNTcurl, tiniest tensor Hcurl, TNTdiv, tiniest tensor Hdiv, Arnold-Boffi-Falk, ABF, Arbogast-Correa, AC, AC full, Arbogast-Correa full, Rannacher-Turek, P1-iso-P2, P2-iso-P1, iso-P2 P1, Huang-Zhang, HZ, enriched Galerkin, EG, enriched vector Galerkin, locking-free enriched Galerkin, LFEG, P1 macro, Alfeld-Sorokina, AS
- **order** – The order of the element.
- **vertices** – The vertices of the reference.

symfem.create.**_order_is_allowed**(*element_class: Type*, *ref: str*, *order: int*) → bool

Check that an order is valid for an element.

Parameters

- **element_class** – The element class
- **ref** – The reference cell
- **order** – The polynomial order

symfem.finite_element

Abstract finite element classes and functions.

Module Contents

Classes

FiniteElement	Abstract finite element.
CiarletElement	Finite element defined using the Ciarlet definition.
DirectElement	Finite element defined directly.
EnrichedElement	Finite element defined directly.
ElementBasisFunction	A basis function of a finite element.

Attributes

TabulatedBasis

symfem.finite_element.**TabulatedBasis**

exception symfem.finite_element.**NoTensorProduct**

Bases: Exception

Error for element without a tensor representation.

class symfem.finite_element.**FiniteElement**(*reference:* symfem.references.Reference, *order: int*, *space_dim: int*, *domain_dim: int*, *range_dim: int*, *range_shape: Tuple[int, Ellipsis] | None = None*)

Bases: abc.ABC

Abstract finite element.

abstract property maximum_degree: int

Get the maximum degree of this polynomial set for the element.

property name: str

Get the name of the element.

Returns
The name of the element's family

_float_basis_functions: None | List[*symfem.functions.AnyFunction*]

_value_scale: None | sympy.core.expr.Expr

names: List[str] = []

references: List[str] = []

last_updated

cache = True

_max_continuity_test_order = 4

abstract dof_plot_positions() → List[symfem.geometry.PointType]

Get the points to plot each DOF at on a DOF diagram.

Returns
The DOF positions

abstract dof_directions() → List[symfem.geometry.PointType | None]

Get the direction associated with each DOF.

Returns
The DOF directions

abstract dof_entities() → List[Tuple[int, int]]

Get the entities that each DOF is associated with.

Returns
The entities

plot_dof_diagram(*filename: str | List[str]*, *plot_options: Dict[str, Any] = {}*, ***kwargs: Any*)

Plot a diagram showing the DOFs of the element.

Parameters

- **filename** – The file name
- **plot_options** – Options for the plot
- **kwargs** – Keyword arguments

abstract entity_dofs(*entity_dim: int*, *entity_number: int*) → List[int]

Get the numbers of the DOFs associated with the given entity.

Parameters

- **entity_dim** – The dimension of the entity
- **entity_number** – The number of the entity

Returns

The numbers of the DOFs associated with the entity

abstract get_basis_functions(*use_tensor_factorisation: bool = False*) → List[*symfem.functions.AnyFunction*]

Get the basis functions of the element.

Parameters

use_tensor_factorisation – Should a tensor factorisation be used?

Returns

The basis functions

get_basis_function(*n: int*) → *symfem.basis_functions.BasisFunction*

Get a single basis function of the element.

Parameters

n – The number of the basis function

Returns

The basis function

tabulate_basis(*points_in: symfem.geometry.SetOfPointsInput*, *order: str = 'xyzxyz'*) → TabulatedBasis

Evaluate the basis functions of the element at the given points.

Parameters

- **points_in** – The points
- **order** – The order to return the values

Returns

The tabulated basis functions

tabulate_basis_float(*points_in: symfem.geometry.SetOfPointsInput*) → TabulatedBasis

Evaluate the basis functions of the element at the given points in xyz,xyz order.

Parameters

points_in – The points

Returns

The tabulated basis functions

plot_basis_function(*n: int*, *filename: str | List[str]*, *cell:* symfem.references.Reference *| None = None*, ***kwargs: Any*)

Plot a diagram showing a basis function.

Parameters

- **n** – The basis function number
- **filename** – The file name
- **cell** – The cell to push the basis function to and plot on
- **kwargs** – Keyword arguments

abstract map_to_cell(*vertices_in: symfem.geometry.SetOfPointsInput*, *basis: List[*symfem.functions.AnyFunction*] | None = None*, *forward_map: symfem.geometry.PointType | None = None*, *inverse_map: symfem.geometry.PointType | None = None*) → List[*symfem.functions.AnyFunction*]

Map the basis onto a cell using the appropriate mapping for the element.

Parameters

- **vertices_in** – The vertices of the cell
- **basis** – The basis functions
- **forward_map** – The map from the reference to the cell
- **inverse_map** – The map to the reference from the cell

Returns
The basis functions mapped to the cell

abstract get_polynomial_basis() → List[*symfem.functions.AnyFunction*]

Get the symbolic polynomial basis for the element.

Returns
The polynomial basis

test()

Run tests for this element.

test_continuity()

Test that this element has the correct continuity.

get_tensor_factorisation() → List[Tuple[str, List[*FiniteElement*], List[int]]]

Get the representation of the element as a tensor product.

Returns
The tensor factorisation

_get_basis_functions_tensor() → List[*symfem.functions.AnyFunction*]

Compute the basis functions using the space's tensor product factorisation.

Returns
The basis functions

init_kwargs() → Dict[str, Any]

Return the keyword arguments used to create this element.

Returns
Keyword arguments dictionary

class symfem.finite_element.**CiarletElement**(*reference:* symfem.references.Reference, *order: int*, *basis: List[symfem.functions.FunctionInput]*, *dofs: symfem.functionals.ListOfFunctionals*, *domain_dim: int*, *range_dim: int*, *range_shape: Tuple[int, Ellipsis] | None = None*)

Bases: *FiniteElement*

Finite element defined using the Ciarlet definition.

property maximum_degree: int

Get the maximum degree of this polynomial set for the element.

entity_dofs(*entity_dim: int*, *entity_number: int*) → List[int]

Get the numbers of the DOFs associated with the given entity.

Parameters

- **entity_dim** – The dimension of the entity
- **entity_number** – The number of the entity

Returns
The numbers of the DOFs associated with the entity

dof_plot_positions() → List[symfem.geometry.PointType]

Get the points to plot each DOF at on a DOF diagram.

Returns
The DOF positions

dof_directions() → List[symfem.geometry.PointType | None]

Get the direction associated with each DOF.

Returns
The DOF directions

dof_entities() → List[Tuple[int, int]]

Get the entities that each DOF is associated with.

Returns
The entities

get_polynomial_basis() → List[*symfem.functions.AnyFunction*]

Get the symbolic polynomial basis for the element.

Returns
The polynomial basis

get_dual_matrix(*inverse=False*, *caching=True*) → sympy.matrices.dense.MutableDenseMatrix

Get the dual matrix.

Parameters

- **inverse** – Should the dual matrix be inverted?
- **caching** – Should the result be cached

Returns
The dual matrix

get_basis_functions(*use_tensor_factorisation: bool = False*) → List[*symfem.functions.AnyFunction*]

Get the basis functions of the element.

Parameters
use_tensor_factorisation – Should a tensor factorisation be used?

Returns
The basis functions

plot_basis_function(*n: int*, *filename: str | List[str]*, *cell:* symfem.references.Reference | *None = None*, ***kwargs: Any*)

Plot a diagram showing a basis function.

Parameters

- **n** – The basis function number

- **filename** – The file name
- **cell** – The cell to push the basis function to and plot on
- **kwargs** – Keyword arguments

map_to_cell(*vertices_in: symfem.geometry.SetOfPointsInput*, *basis: List[symfem.functions.AnyFunction] | None = None*, *forward_map: symfem.geometry.PointType | None = None*, *inverse_map: symfem.geometry.PointType | None = None*) → List[*symfem.functions.AnyFunction*]

Map the basis onto a cell using the appropriate mapping for the element.

Parameters

- **vertices_in** – The vertices of the cell
- **basis** – The basis functions
- **forward_map** – The map from the reference to the cell
- **inverse_map** – The map to the reference from the cell

Returns

The basis functions mapped to the cell

test()

Run tests for this element.

test_dof_points()

Test that DOF points are valid.

test_functional_entities()

Test that the dof entities are valid and match the references of integrals.

test_functionals()

Test that the functionals are satisfied by the basis functions.

class symfem.finite_element.**DirectElement**(*reference: symfem.references.Reference*, *order: int*, *basis_functions: List[symfem.functions.FunctionInput]*, *basis_entities: List[Tuple[int, int]]*, *domain_dim: int*, *range_dim: int*, *range_shape: Tuple[int, Ellipsis] | None = None*)

Bases: *FiniteElement*

Finite element defined directly.

abstract property maximum_degree: int

Get the maximum degree of this polynomial set for the element.

_basis_functions: List[*symfem.functions.AnyFunction*]

entity_dofs(*entity_dim: int*, *entity_number: int*) → List[int]

Get the numbers of the DOFs associated with the given entity.

Parameters

- **entity_dim** – The dimension of the entity
- **entity_number** – The number of the entity

Returns

The numbers of the DOFs associated with the entity

dof_plot_positions() → List[symfem.geometry.PointType]

Get the points to plot each DOF at on a DOF diagram.

Returns

The DOF positions

dof_directions() → List[symfem.geometry.PointType | None]

Get the direction associated with each DOF.

Returns
The DOF directions

dof_entities() → List[Tuple[int, int]]

Get the entities that each DOF is associated with.

Returns
The entities

get_basis_functions(*use_tensor_factorisation: bool = False*) → List[*symfem.functions.AnyFunction*]

Get the basis functions of the element.

Parameters
use_tensor_factorisation – Should a tensor factorisation be used?

Returns
The basis functions

map_to_cell(*vertices_in: symfem.geometry.SetOfPointsInput*, *basis: List[*symfem.functions.AnyFunction*] | None = None*, *forward_map: symfem.geometry.PointType | None = None*, *inverse_map: symfem.geometry.PointType | None = None*) → List[*symfem.functions.AnyFunction*]

Map the basis onto a cell using the appropriate mapping for the element.

Parameters

- **vertices_in** – The vertices of the cell
- **basis** – The basis functions
- **forward_map** – The map from the reference to the cell
- **inverse_map** – The map to the reference from the cell

Returns
The basis functions mapped to the cell

abstract get_polynomial_basis() → List[*symfem.functions.AnyFunction*]

Get the symbolic polynomial basis for the element.

Returns
The polynomial basis

test()

Run tests for this element.

test_independence()

Test that the basis functions of this element are linearly independent.

class symfem.finite_element.**EnrichedElement**(*subelements: List[*FiniteElement*]*)

Bases: *FiniteElement*

Finite element defined directly.

abstract property maximum_degree: int

Get the maximum degree of this polynomial set for the element.

_basis_functions: List[*symfem.functions.AnyFunction*] | None

entity_dofs(*entity_dim: int*, *entity_number: int*) → List[int]

Get the numbers of the DOFs associated with the given entity.

Parameters

- **entity_dim** – The dimension of the entity
- **entity_number** – The number of the entity

Returns
The numbers of the DOFs associated with the entity

dof_plot_positions() → List[symfem.geometry.PointType]

Get the points to plot each DOF at on a DOF diagram.

Returns
The DOF positions

dof_directions() → List[symfem.geometry.PointType | None]

Get the direction associated with each DOF.

Returns
The DOF directions

dof_entities() → List[Tuple[int, int]]

Get the entities that each DOF is associated with.

Returns
The entities

get_basis_functions(*use_tensor_factorisation: bool = False*) → List[*symfem.functions.AnyFunction*]

Get the basis functions of the element.

Parameters
use_tensor_factorisation – Should a tensor factorisation be used?

Returns
The basis functions

map_to_cell(*vertices_in: symfem.geometry.SetOfPointsInput*, *basis: List[*symfem.functions.AnyFunction*] | None = None*, *forward_map: symfem.geometry.PointType | None = None*, *inverse_map: symfem.geometry.PointType | None = None*) → List[*symfem.functions.AnyFunction*]

Map the basis onto a cell using the appropriate mapping for the element.

Parameters

- **vertices_in** – The vertices of the cell
- **basis** – The basis functions
- **forward_map** – The map from the reference to the cell
- **inverse_map** – The map to the reference from the cell

Returns
The basis functions mapped to the cell

abstract get_polynomial_basis() → List[*symfem.functions.AnyFunction*]

Get the symbolic polynomial basis for the element.

Returns
The polynomial basis

test()

Run tests for this element.

class symfem.finite_element.**ElementBasisFunction**(*element:* FiniteElement, *n: int*)

Bases: *symfem.basis_functions.BasisFunction*

A basis function of a finite element.

get_function() → *symfem.functions.AnyFunction*

Get the actual basis function.

Returns
The basis function

symfem.functionals

Functionals used to define the dual sets.

Module Contents

Classes

BaseFunctional	A functional.
PointEvaluation	A point evaluation.
WeightedPointEvaluation	A point evaluation.
DerivativePointEvaluation	A point evaluation of a given derivative.
PointDirectionalDerivativeEvaluation	A point evaluation of a derivative in a fixed direction.
PointNormalDerivativeEvaluation	A point evaluation of a normal derivative.
PointComponentSecondDerivativeEvaluation	A point evaluation of a component of a second derivative.
PointInnerProduct	An evaluation of an inner product at a point.
DotPointEvaluation	A point evaluation in a given direction.
PointDivergenceEvaluation	A point evaluation of the divergence.
IntegralAgainst	An integral against a function.
IntegralOfDivergenceAgainst	An integral of the divergence against a function.
IntegralOfDirectionalMultiderivative	An integral of a directional derivative of a scalar function.
IntegralMoment	An integral moment.
DerivativeIntegralMoment	An integral moment of the derivative of a scalar function.
DivergenceIntegralMoment	An integral moment of the divergence of a vector function.
TangentIntegralMoment	An integral moment in the tangential direction.
NormalIntegralMoment	An integral moment in the normal direction.
NormalDerivativeIntegralMoment	An integral moment in the normal direction.
InnerProductIntegralMoment	An integral moment of the inner product with a vector.
NormalInnerProductIntegralMoment	An integral moment of the inner product with the normal direction.

Functions

_to_tex(→ str)	Convert an expresson to TeX.
_nth(→ str)	Add th, st or nd to a number.

Attributes

ScalarValueOrFloat
ListOfFunctionals

symfem.functionals.**ScalarValueOrFloat**

symfem.functionals.**_to_tex**(*f: symfem.functions.FunctionInput*, *tfrac: bool = False*) → str

Convert an expresson to TeX.

Parameters

- **f** – the function
- **tfrac** – Should \tfrac be used in the place of \frac?

Returns

The function as TeX

symfem.functionals.**_nth**(*n: int*) → str

Add th, st or nd to a number.

Parameters

n – The number

Returns

The string (n)th, (n)st, (n)rd or (n)nd

class symfem.functionals.**BaseFunctional**(*reference:* symfem.references.Reference, *entity: Tuple[int, int]*, *mapping: str | None*)

Bases: abc.ABC

A functional.

name = 'Base functional'

entity_dim() → int

Get the dimension of the entity this DOF is associated with.

Returns

The dimension of the entity this DOF is associated with

entity_number() → int

Get the number of the entity this DOF is associated with.

Returns

The number of the entity this DOF is associated with

perform_mapping(*fs: List[*symfem.functions.AnyFunction*]*, *map: symfem.geometry.PointType*, *inverse_map: symfem.geometry.PointType*) → List[*symfem.functions.AnyFunction*]

Map functions to a cell.

Parameters

- **fs** – functions
- **map** – Map from the reference cell to a physical cell
- **inverse_map** – Map to the reference cell from a physical cell

Returns

Mapped functions

entity_tex() → str

Get the entity the DOF is associated with in TeX format.

Returns
TeX representation of entity

entity_definition() → str

Get the definition of the entity the DOF is associated with.

Returns
The definition

dof_direction() → symfem.geometry.PointType | None

Get the direction of the DOF.

Returns
The direction

eval(*function:* symfem.functions.AnyFunction, *symbolic: bool = True*) → *symfem.functions.ScalarFunction* | float

Apply to the functional to a function.

Parameters

- **function** – The function
- **symbolic** – Should it be applied symbolically?

Returns
The value of the functional for the function

abstract dof_point() → symfem.geometry.PointType

Get the location of the DOF in the cell.

Returns
The point

adjusted_dof_point() → symfem.geometry.PointType

Get the adjusted position of the DOF in the cell for plotting.

Returns
The point

eval_symbolic(*function:* symfem.functions.AnyFunction) → *symfem.functions.ScalarFunction*

Symbolically apply the functional to a function.

Parameters
function – The function

Returns
The value of the functional for the function

abstract _eval_symbolic(*function:* symfem.functions.AnyFunction) → *symfem.functions.AnyFunction*

Apply to the functional to a function.

Parameters
function – The function

Returns
The value of the functional for the function

abstract get_tex() → Tuple[str, List[str]]

Get a representation of the functional as TeX, and list of terms involved.

Returns
Representation of the functional as TeX, and list of terms involved

class symfem.functionals.**PointEvaluation**(*reference: symfem.references.Reference*, *point_in: symfem.functions.FunctionInput*, *entity: Tuple[int, int]*, *mapping: str | None = 'identity'*)

Bases: *BaseFunctional*

A point evaluation.

name = 'Point evaluation'

_eval_symbolic(*function: symfem.functions.AnyFunction*) → *symfem.functions.AnyFunction*

Apply to the functional to a function.

Parameters
function – The function

Returns
The value of the functional for the function

dof_point() → symfem.geometry.PointType

Get the location of the DOF in the cell.

Returns
The point

adjusted_dof_point() → symfem.geometry.PointType

Get the adjusted position of the DOF in the cell for plotting.

Returns
The point

get_tex() → Tuple[str, List[str]]

Get a representation of the functional as TeX, and list of terms involved.

Returns
Representation of the functional as TeX, and list of terms involved

class symfem.functionals.**WeightedPointEvaluation**(*reference: symfem.references.Reference*, *point_in: symfem.functions.FunctionInput*, *weight: sympy.core.expr.Expr*, *entity: Tuple[int, int]*, *mapping: str | None = 'identity'*)

Bases: *BaseFunctional*

A point evaluation.

name = 'Weighted point evaluation'

_eval_symbolic(*function: symfem.functions.AnyFunction*) → *symfem.functions.AnyFunction*

Apply to the functional to a function.

Parameters
function – The function

Returns
The value of the functional for the function

dof_point() → symfem.geometry.PointType

Get the location of the DOF in the cell.

Returns
The point

adjusted_dof_point() → symfem.geometry.PointType

Get the adjusted position of the DOF in the cell for plotting.

Returns
The point

get_tex() → Tuple[str, List[str]]

Get a representation of the functional as TeX, and list of terms involved.

Returns
Representation of the functional as TeX, and list of terms involved

class symfem.functionals.**DerivativePointEvaluation**(*reference:* symfem.references.Reference, *point_in: symfem.functions.FunctionInput*, *derivative: Tuple[int, Ellipsis]*, *entity: Tuple[int, int]*, *mapping: str | None = None*)

Bases: *BaseFunctional*

A point evaluation of a given derivative.

name = 'Point derivative evaluation'

_eval_symbolic(*function:* symfem.functions.AnyFunction) → *symfem.functions.AnyFunction*

Apply to the functional to a function.

Parameters
function – The function

Returns
The value of the functional for the function

dof_point() → symfem.geometry.PointType

Get the location of the DOF in the cell.

Returns
The point

adjusted_dof_point() → symfem.geometry.PointType

Get the adjusted position of the DOF in the cell for plotting.

Returns
The point

perform_mapping(*fs: List[*symfem.functions.AnyFunction*]*, *map: symfem.geometry.PointType*, *inverse_map: symfem.geometry.PointType*) → List[*symfem.functions.AnyFunction*]

Map functions to a cell.

Parameters

- **fs** – functions
- **map** – Map from the reference cell to a physical cell
- **inverse_map** – Map to the reference cell from a physical cell

Returns
Mapped functions

get_tex() → Tuple[str, List[str]]

Get a representation of the functional as TeX, and list of terms involved.

Returns
Representation of the functional as TeX, and list of terms involved

class symfem.functionals.**PointDirectionalDerivativeEvaluation**(*reference:* symfem.references.Reference, *point_in: symfem.functions.FunctionInput*, *direction_in: symfem.functions.FunctionInput*, *entity: Tuple[int, int]*, *mapping: str | None = 'identity'*)

Bases: *BaseFunctional*

A point evaluation of a derivative in a fixed direction.

name = 'Point evaluation of directional derivative'

_eval_symbolic(*function:* symfem.functions.AnyFunction) → *symfem.functions.AnyFunction*

Apply to the functional to a function.

Parameters
function – The function

Returns
The value of the functional for the function

dof_point() → symfem.geometry.PointType

Get the location of the DOF in the cell.

Returns
The point

dof_direction() → symfem.geometry.PointType | None

Get the direction of the DOF.

Returns
The direction

adjusted_dof_point() → symfem.geometry.PointType

Get the adjusted position of the DOF in the cell for plotting.

Returns
The point

get_tex() → Tuple[str, List[str]]

Get a representation of the functional as TeX, and list of terms involved.

Returns
Representation of the functional as TeX, and list of terms involved

class symfem.functionals.**PointNormalDerivativeEvaluation**(*reference:* symfem.references.Reference, *point_in:* *symfem.functions.FunctionInput*, *edge:* symfem.references.Reference, *entity: Tuple[int, int]*, *mapping: str | None = 'identity'*)

Bases: *PointDirectionalDerivativeEvaluation*

A point evaluation of a normal derivative.

name = 'Point evaluation of normal derivative'

get_tex() → Tuple[str, List[str]]

Get a representation of the functional as TeX, and list of terms involved.

Returns
Representation of the functional as TeX, and list of terms involved

class symfem.functionals.**PointComponentSecondDerivativeEvaluation**(*reference:* symfem.references.Reference, *point_in:* *symfem.functions.FunctionInput*, *component: Tuple[int, int]*, *entity: Tuple[int, int]*, *mapping: str | None = 'identity'*)

Bases: *BaseFunctional*

A point evaluation of a component of a second derivative.

name = 'Point evaluation of Jacobian component'

_eval_symbolic(*function:* symfem.functions.AnyFunction) → *symfem.functions.AnyFunction*

Apply to the functional to a function.

Parameters
function – The function

Returns
The value of the functional for the function

dof_point() → symfem.geometry.PointType

Get the location of the DOF in the cell.

Returns
The point

adjusted_dof_point() → symfem.geometry.PointType

Get the adjusted position of the DOF in the cell for plotting.

Returns
The point

get_tex() → Tuple[str, List[str]]

Get a representation of the functional as TeX, and list of terms involved.

Returns
Representation of the functional as TeX, and list of terms involved

class symfem.functionals.**PointInnerProduct**(*reference:* symfem.references.Reference, *point_in: symfem.functions.FunctionInput*, *lvec: symfem.functions.FunctionInput*, *rvec: symfem.functions.FunctionInput*, *entity: Tuple[int, int]*, *mapping: str | None = 'identity'*)

Bases: *BaseFunctional*

An evaluation of an inner product at a point.

name = 'Point inner product'

_eval_symbolic(*function:* symfem.functions.AnyFunction) → *symfem.functions.AnyFunction*

Apply to the functional to a function.

Parameters
function – The function

Returns
The value of the functional for the function

dof_point() → symfem.geometry.PointType

Get the location of the DOF in the cell.

Returns
The point

dof_direction() → symfem.geometry.PointType | None

Get the direction of the DOF.

Returns
The direction

adjusted_dof_point() → symfem.geometry.PointType

Get the adjusted position of the DOF in the cell for plotting.

Returns
The point

get_tex() → Tuple[str, List[str]]

Get a representation of the functional as TeX, and list of terms involved.

Returns
Representation of the functional as TeX, and list of terms involved

class symfem.functionals.**DotPointEvaluation**(*reference:* symfem.references.Reference, *point_in: symfem.functions.FunctionInput*, *vector_in: symfem.functions.FunctionInput*, *entity: Tuple[int, int]*, *mapping: str | None = 'identity'*)

Bases: *BaseFunctional*

A point evaluation in a given direction.

name = 'Dot point evaluation'

_eval_symbolic(*function:* symfem.functions.AnyFunction) → *symfem.functions.AnyFunction*

Apply to the functional to a function.

Parameters
function – The function

Returns
The value of the functional for the function

dof_point() → symfem.geometry.PointType

Get the location of the DOF in the cell.

Returns
The point

dof_direction() → symfem.geometry.PointType | None

Get the direction of the DOF.

Returns
The direction

adjusted_dof_point() → symfem.geometry.PointType

Get the adjusted position of the DOF in the cell for plotting.

Returns
The point

get_tex() → Tuple[str, List[str]]

Get a representation of the functional as TeX, and list of terms involved.

Returns
Representation of the functional as TeX, and list of terms involved

class symfem.functionals.**PointDivergenceEvaluation**(*reference:* symfem.references.Reference, *point_in: symfem.functions.FunctionInput*, *entity: Tuple[int, int]*, *mapping: str | None = 'identity'*)

Bases: *BaseFunctional*

A point evaluation of the divergence.

name = 'Point evaluation of divergence'

_eval_symbolic(*function:* symfem.functions.AnyFunction) → *symfem.functions.AnyFunction*

Apply to the functional to a function.

Parameters
function – The function

Returns
The value of the functional for the function

dof_point() → symfem.geometry.PointType

Get the location of the DOF in the cell.

Returns
The point

dof_direction() → symfem.geometry.PointType | None

Get the direction of the DOF.

Returns
The direction

adjusted_dof_point() → symfem.geometry.PointType

Get the adjusted position of the DOF in the cell for plotting.

Returns
The point

get_tex() → Tuple[str, List[str]]

Get a representation of the functional as TeX, and list of terms involved.

Returns
Representation of the functional as TeX, and list of terms involved

class symfem.functionals.**IntegralAgainst**(*reference:* symfem.references.Reference, *f_in: symfem.functions.FunctionInput*, *entity: Tuple[int, int]*, *mapping: str | None = 'identity'*)

Bases: *BaseFunctional*

An integral against a function.

f: *symfem.functions.AnyFunction*

name = 'Integral against'

dof_point() → symfem.geometry.PointType

Get the location of the DOF in the cell.

Returns
The point

_eval_symbolic(*function:* symfem.functions.AnyFunction) → *symfem.functions.AnyFunction*

Apply to the functional to a function.

Parameters
function – The function

Returns
The value of the functional for the function

dot(*function:* symfem.functions.AnyFunction) → *symfem.functions.ScalarFunction*

Dot a function with the moment function.

Parameters
function – The function

Returns
The product of the function and the moment function

get_tex() → Tuple[str, List[str]]

Get a representation of the functional as TeX, and list of terms involved.

Returns
Representation of the functional as TeX, and list of terms involved

class symfem.functionals.**IntegralOfDivergenceAgainst**(*reference:* symfem.references.Reference, *f_in: symfem.functions.FunctionInput*, *entity: Tuple[int, int]*, *mapping: str | None = 'identity'*)

Bases: *BaseFunctional*

An integral of the divergence against a function.

name = 'Integral of divergence against'

dof_point() → symfem.geometry.PointType

Get the location of the DOF in the cell.

Returns
The point

_eval_symbolic(*function:* symfem.functions.AnyFunction) → *symfem.functions.AnyFunction*

Apply to the functional to a function.

Parameters
function – The function

Returns
The value of the functional for the function

dot(*function:* symfem.functions.ScalarFunction) → *symfem.functions.ScalarFunction*

Dot a function with the moment function.

Parameters
function – The function

Returns
The product of the function and the moment function

get_tex() → Tuple[str, List[str]]

Get a representation of the functional as TeX, and list of terms involved.

Returns
Representation of the functional as TeX, and list of terms involved

class symfem.functionals.**IntegralOfDirectionalMultiderivative**(*reference:* symfem.references.Reference, *directions: symfem.geometry.SetOfPoints*, *orders: Tuple[int, Ellipsis]*, *entity: Tuple[int, int]*, *scale: int = 1*, *mapping: str | None = 'identity'*)

Bases: *BaseFunctional*

An integral of a directional derivative of a scalar function.

name = 'Integral of a directional derivative'

dof_point() → symfem.geometry.PointType

Get the location of the DOF in the cell.

Returns
The point

_eval_symbolic(*function:* symfem.functions.AnyFunction) → *symfem.functions.AnyFunction*

Apply to the functional to a function.

Parameters
function – The function

Returns
The value of the functional for the function

perform_mapping(*fs: List[*symfem.functions.AnyFunction*]*, *map: symfem.geometry.PointType*, *inverse_map: symfem.geometry.PointType*) → List[*symfem.functions.AnyFunction*]

Map functions to a cell.

Parameters

- **fs** – functions
- **map** – Map from the reference cell to a physical cell
- **inverse_map** – Map to the reference cell from a physical cell

Returns
Mapped functions

get_tex() → Tuple[str, List[str]]

Get a representation of the functional as TeX, and list of terms involved.

Returns
Representation of the functional as TeX, and list of terms involved

class symfem.functionals.**IntegralMoment**(*reference:* symfem.references.Reference, *f_in: symfem.functions.FunctionInput*, *dof:* BaseFunctional, *entity: Tuple[int, int]*, *mapping: str | None = 'identity'*, *map_function: bool = True*)

Bases: *BaseFunctional*

An integral moment.

f: *symfem.functions.AnyFunction*

name = 'Integral moment'

_eval_symbolic(*function:* symfem.functions.AnyFunction) → *symfem.functions.AnyFunction*

Apply to the functional to a function.

Parameters
function – The function

Returns
The value of the functional for the function

dot(*function:* symfem.functions.AnyFunction) → *symfem.functions.ScalarFunction*

Dot a function with the moment function.

Parameters
function – The function

Returns
The product of the function and the moment function

dof_point() → symfem.geometry.PointType

Get the location of the DOF in the cell.

Returns
The point

dof_direction() → symfem.geometry.PointType | None

Get the direction of the DOF.

Returns
The direction

get_tex() → Tuple[str, List[str]]

Get a representation of the functional as TeX, and list of terms involved.

Returns
Representation of the functional as TeX, and list of terms involved

class symfem.functionals.**DerivativeIntegralMoment**(*reference:* symfem.references.Reference, *f:* *symfem.functions.FunctionInput*, *dot_with_in:* *symfem.functions.FunctionInput*, *dof:* BaseFunctional, *entity: Tuple[int, int]*, *mapping: str | None = 'identity'*)

Bases: *IntegralMoment*

An integral moment of the derivative of a scalar function.

name = 'Derivative integral moment'

dot(*function:* symfem.functions.AnyFunction) → *symfem.functions.ScalarFunction*

Dot a function with the moment function.

Parameters
function – The function

Returns
The product of the function and the moment function

dof_direction() → symfem.geometry.PointType | None

Get the direction of the DOF.

Returns
The direction

_eval_symbolic(*function:* symfem.functions.AnyFunction) → *symfem.functions.AnyFunction*

Apply to the functional to a function.

Parameters
function – The function

Returns
The value of the functional for the function

class symfem.functionals.**DivergenceIntegralMoment**(*reference:* symfem.references.Reference, *f_in:* *symfem.functions.FunctionInput*, *dof:* BaseFunctional, *entity: Tuple[int, int]*, *mapping: str | None = 'identity'*)

Bases: *IntegralMoment*

An integral moment of the divergence of a vector function.

name = 'Integral moment of divergence'

_eval_symbolic(*function:* symfem.functions.AnyFunction) → *symfem.functions.AnyFunction*

Apply to the functional to a function.

Parameters
function – The function

Returns
The value of the functional for the function

get_tex() → Tuple[str, List[str]]

Get a representation of the functional as TeX, and list of terms involved.

Returns
Representation of the functional as TeX, and list of terms involved

class symfem.functionals.**TangentIntegralMoment**(*reference:* symfem.references.Reference, *f_in: symfem.functions.FunctionInput*, *dof:* BaseFunctional, *entity: Tuple[int, int]*, *mapping: str | None = 'covariant'*)

Bases: *IntegralMoment*

An integral moment in the tangential direction.

name = 'Tangential integral moment'

dof_direction() → symfem.geometry.PointType | None

Get the direction of the DOF.

Returns
The direction

get_tex() → Tuple[str, List[str]]

Get a representation of the functional as TeX, and list of terms involved.

Returns
Representation of the functional as TeX, and list of terms involved

class symfem.functionals.**NormalIntegralMoment**(*reference:* symfem.references.Reference, *f_in: symfem.functions.FunctionInput*, *dof:* BaseFunctional, *entity: Tuple[int, int]*, *mapping: str | None = 'contravariant'*)

Bases: *IntegralMoment*

An integral moment in the normal direction.

name = 'Normal integral moment'

dof_direction() → symfem.geometry.PointType | None

Get the direction of the DOF.

Returns
The direction

get_tex() → Tuple[str, List[str]]

Get a representation of the functional as TeX, and list of terms involved.

Returns
Representation of the functional as TeX, and list of terms involved

class symfem.functionals.**NormalDerivativeIntegralMoment**(*reference:* symfem.references.Reference, *f_in: symfem.functions.FunctionInput*, *dof:* BaseFunctional, *entity: Tuple[int, int]*, *mapping: str | None = 'identity'*)

Bases: *DerivativeIntegralMoment*

An integral moment in the normal direction.

name = 'Normal derivative integral moment'

get_tex() → Tuple[str, List[str]]

Get a representation of the functional as TeX, and list of terms involved.

Returns
Representation of the functional as TeX, and list of terms involved

class symfem.functionals.**InnerProductIntegralMoment**(*reference:* symfem.references.Reference, *f_in: symfem.functions.FunctionInput*, *inner_with_left_in: symfem.functions.FunctionInput*, *inner_with_right_in: symfem.functions.FunctionInput*, *dof:* BaseFunctional, *entity: Tuple[int, int]*, *mapping: str | None = 'identity'*)

Bases: *IntegralMoment*

An integral moment of the inner product with a vector.

name = 'Inner product integral moment'

dot(*function:* symfem.functions.AnyFunction) → *symfem.functions.ScalarFunction*
Take the inner product of a function with the moment direction.

Parameters
function – The function

Returns
The inner product of the function and the moment direction

dof_direction() → symfem.geometry.PointType | None
Get the direction of the DOF.

Returns
The direction

get_tex() → Tuple[str, List[str]]
Get a representation of the functional as TeX, and list of terms involved.

Returns
Representation of the functional as TeX, and list of terms involved

class symfem.functionals.**NormalInnerProductIntegralMoment**(*reference:* symfem.references.Reference, *f_in: symfem.functions.FunctionInput*, *dof:* BaseFunctional, *entity: Tuple[int, int]*, *mapping: str | None = 'double_contravariant'*)

Bases: *InnerProductIntegralMoment*

An integral moment of the inner product with the normal direction.

name = 'Normal inner product integral moment'

get_tex() → Tuple[str, List[str]]
Get a representation of the functional as TeX, and list of terms involved.

Returns
Representation of the functional as TeX, and list of terms involved

symfem.functionals.**ListOfFunctionals**

symfem.functions

Basis function classes.

Module Contents

Classes

AnyFunction	A function.
ScalarFunction	A scalar-valued function.
VectorFunction	A vector-valued function.
MatrixFunction	A matrix-valued function.

Functions

_to_sympy_format(→ SympyFormat)	Convert to Sympy format used by these functions.
_check_equal(→ bool)	Check if two items are equal.
parse_function_input(→ AnyFunction)	Parse a function.
parse_function_list_input(→ List[AnyFunction])	Parse a list of functions.

Attributes

SingleSympyFormat
SympyFormat
_ValuesToSubstitute
ValuesToSubstitute
FunctionInput

symfem.functions.**SingleSympyFormat**

symfem.functions.**SympyFormat**

symfem.functions.**_ValuesToSubstitute**

symfem.functions.**_to_sympy_format**(*item: Any*) → SympyFormat

Convert to Sympy format used by these functions.

Parameters
item – The input item

Returns
The item in Sympy format expected by functions

symfem.functions.**_check_equal**(*first: SympyFormat*, *second: SympyFormat*) → bool

Check if two items are equal.

Parameters

- **first** – The first item
- **second** – The second item

Returns

Are the two items equal?

class symfem.functions.**AnyFunction**(*scalar: bool = False*, *vector: bool = False*, *matrix: bool = False*)

Bases: abc.ABC

A function.

property shape: Tuple[int, Ellipsis]

Get the value shape of the function.

Returns

The value shape

abstract __add__(*other: Any*)

Add.

abstract __radd__(*other: Any*)

Add.

abstract __sub__(*other: Any*)

Subtract.

abstract __rsub__(*other: Any*)

Subtract.

abstract __neg__()

Negate.

abstract __truediv__(*other: Any*)

Divide.

abstract __rtruediv__(*other: Any*)

Divide.

abstract __mul__(*other: Any*)

Multiply.

abstract __rmul__(*other: Any*)

Multiply.

abstract __matmul__(*other: Any*)

Multiply.

abstract __rmatmul__(*other: Any*)

Multiply.

abstract __pow__(*other: Any*)

Raise to a power.

abstract as_sympy() → SympyFormat

Convert to a Sympy expression.

Returns

A Sympy expression

abstract as_tex() → str

Convert to a TeX expression.

Returns
A TeX string

abstract subs(*vars: symfem.symbols.AxisVariables*, *values:* AnyFunction | *_ValuesToSubstitute*)

Substitute values into the function.

Parameters

- **vars** – The variables to substitute out
- **values** – The value to substitute in

Returns
The substituted function

abstract diff(*variable: sympy.core.symbol.Symbol*)

Differentiate the function.

Parameters
variable – The variable to differentiate with respect to

Returns
The differentiated function

abstract directional_derivative(*direction: symfem.geometry.PointType*)

Compute a directional derivative.

Parameters
direction – The diection

Returns
The directional differentiate

abstract jacobian_component(*component: Tuple[int, int]*)

Compute a component of the jacobian.

Parameters
component – The component

Returns
The component of the jacobian

abstract jacobian(*dim: int*)

Compute the jacobian.

Parameters
dim – The topological dimension of the cell

Returns
The jacobian

abstract dot(*other_in: FunctionInput*)

Compute the dot product with another function.

Parameters
other_in – The function to multiply with

Returns
The product

abstract cross(*other_in: FunctionInput*)

Compute the cross product with another function.

Parameters
other_in – The function to multiply with

Returns
The cross product

abstract div()

Compute the div of the function.

Returns
The divergence

abstract grad(*dim: int*)

Compute the grad of the function.

Returns
The gradient

abstract curl()

Compute the curl of the function.

Returns
The curl

abstract norm()

Compute the norm of the function.

Returns
The norm

abstract integral(*domain:* symfem.references.Reference, *vars:* *symfem.symbols.AxisVariablesNotSingle = x*, *dummy_vars:* *symfem.symbols.AxisVariablesNotSingle = t*) → *ScalarFunction*

Compute the integral of the function.

Parameters

- **domain** – The domain of the integral
- **vars** – The variables to integrate with respect to
- **dummy_vars** – The dummy variables to use inside the integral

Returns
The integral

abstract with_floats() → *AnyFunction*

Return a version the function with floats as coefficients.

Returns
A version the function with floats as coefficients

abstract maximum_degree(*cell:* symfem.references.Reference) → int

Return the maximum degree of the function on a reference cell.

This function returns the order of the lowerst order Lagrange space on the input cell that includes this function.

Parameters
cell – The cell

Returns
A version the function with floats as coefficients

__iter__() → Iterator[*AnyFunction*]

Iterate through components of vector function.

integrate(**limits: Tuple[sympy.core.symbol.Symbol, int | sympy.core.expr.Expr, int | sympy.core.expr.Expr]*)

Integrate the function.

Parameters
limits – The variables and limits

Returns
The integral

det()

Compute the determinant.

Returns
The deteminant

transpose()

Compute the transpose.

Returns
The transpose

plot(*reference: symfem.references.Reference*, *filename: str | List[str]*, *dof_point: symfem.geometry.PointType | None = None*, *dof_direction: symfem.geometry.PointType | None = None*, *dof_entity: Tuple[int, int] | None = None*, *dof_n: int | None = None*, *value_scale: sympy.core.expr.Expr = sympy.Integer(1)*, *plot_options: Dict[str, Any] = {}*, ***kwargs: Any*)

Plot the function.

Parameters

- **reference** – The reference cell
- **filename** – The file name
- **dof_point** – The DOF point
- **dof_direction** – The direction of the DOF
- **dof_entity** – The entity the DOF is associated with
- **dof_n** – The number of the DOF
- **value_scale** – The scale factor for the function values
- **plot_options** – Options for the plot
- **kwargs** – Keyword arguments

plot_values(*reference: symfem.references.Reference*, *img: Any*, *value_scale: sympy.core.expr.Expr = sympy.Integer(1)*, *n: int = 6*)

Plot the function's values.

Parameters

- **reference** – The reference cell
- **img** – The image to plot on
- **value_scale** – The scale factor for the function values
- **n** – The number of points per side for plotting

__len__()

Compute the determinant.

__getitem__(*key*) → *AnyFunction*

Get a component or slice of the function.

sympy() → SympyFormat
Convert to Sympy format.

__float__() → float
Convert to a float.

__lt__(*other: Any*) → bool
Check inequality.

__le__(*other: Any*) → bool
Check inequality.

__gt__(*other: Any*) → bool
Check inequality.

__ge__(*other: Any*) → bool
Check inequality.

__repr__() → str
Representation.

__eq__(*other: Any*) → bool
Check if two functions are equal.

__ne__(*other: Any*) → bool
Check if two functions are not equal.

symfem.functions.**ValuesToSubstitute**

class symfem.functions.**ScalarFunction**(*f: int | sympy.core.expr.Expr*)
Bases: *AnyFunction*

A scalar-valued function.

_f: sympy.core.expr.Expr

__add__(*other: Any*) → *ScalarFunction*
Add.

__radd__(*other: Any*) → *ScalarFunction*
Add.

__sub__(*other: Any*) → *ScalarFunction*
Subtract.

__rsub__(*other: Any*) → *ScalarFunction*
Subtract.

__truediv__(*other: Any*) → *ScalarFunction*
Divide.

__rtruediv__(*other: Any*) → *ScalarFunction*
Divide.

__mul__(*other: Any*) → *ScalarFunction*
Multiply.

__rmul__(*other: Any*) → *ScalarFunction*
Multiply.

__matmul__(*other: Any*)
Multiply.

__rmatmul__(*other: Any*)

Multiply.

__pow__(*other: Any*) → *ScalarFunction*

Raise to a power.

__neg__() → *ScalarFunction*

Negate.

as_sympy() → SympyFormat

Convert to a sympy expression.

Returns
A Sympy expression

as_tex() → str

Convert to a TeX expression.

Returns
A TeX string

subs(*vars: symfem.symbols.AxisVariables*, *values: ValuesToSubstitute*) → *ScalarFunction*

Substitute values into the function.

Parameters

- **vars** – The variables to substitute out
- **values** – The value to substitute in

Returns
The substituted function

diff(*variable: sympy.core.symbol.Symbol*) → *ScalarFunction*

Differentiate the function.

Parameters
variable – The variable to differentiate with respect to

Returns
The differentiated function

directional_derivative(*direction: symfem.geometry.PointType*) → *ScalarFunction*

Compute a directional derivative.

Parameters
direction – The diection

Returns
The directional derivatve

jacobian_component(*component: Tuple[int, int]*) → *ScalarFunction*

Compute a component of the jacobian.

Parameters
component – The component

Returns
The component of the jacobian

jacobian(*dim: int*) → *MatrixFunction*

Compute the jacobian.

Parameters
dim – The topological dimension of the cell

Returns
The jacobian

dot(*other_in: FunctionInput*) → *ScalarFunction*

Compute the dot product with another function.

Parameters
other_in – The function to multiply with

Returns
The product

cross(*other_in: FunctionInput*)

Compute the cross product with another function.

Parameters
other_in – The function to multiply with

Returns
The cross product

div()

Compute the div of the function.

Returns
The divergence

grad(*dim: int*) → *VectorFunction*

Compute the grad of the function.

Returns
The gradient

curl()

Compute the curl of the function.

Returns
The curl

norm() → *ScalarFunction*

Compute the norm of the function.

Returns
The norm

integral(*domain:* symfem.references.Reference, *vars: symfem.symbols.AxisVariablesNotSingle = x*, *dummy_vars: symfem.symbols.AxisVariablesNotSingle = t*) → *ScalarFunction*

Compute the integral of the function.

Parameters

- **domain** – The domain of the integral
- **vars** – The variables to integrate with respect to
- **dummy_vars** – The dummy variables to use inside the integral

Returns
The integral

integrate(**limits: Tuple[sympy.core.symbol.Symbol, int | sympy.core.expr.Expr, int | sympy.core.expr.Expr]*)

Integrate the function.

Parameters
limits – The variables and limits

Returns
The integral

plot_values(*reference:* symfem.references.Reference, *img: Any*, *value_scale: sympy.core.expr.Expr = sympy.Integer(1)*, *n: int = 6*)

Plot the function's values.

Parameters

- **reference** – The reference cell
- **img** – The image to plot on
- **value_scale** – The scale factor for the function values
- **n** – The number of points per side for plotting

with_floats() → *AnyFunction*

Return a version the function with floats as coefficients.

Returns
A version the function with floats as coefficients

maximum_degree(*cell:* symfem.references.Reference) → int

Return the maximum degree of the function on a reference cell.

This function returns the order of the lowerst order Lagrange space on the input cell that includes this function.

Parameters
cell – The cell

Returns
A version the function with floats as coefficients

class symfem.functions.**VectorFunction**(*vec: Tuple[*AnyFunction | *int* | *sympy.core.expr.Expr, Ellipsis]* | *List[*AnyFunction | *int* | *sympy.core.expr.Expr]*)

Bases: *AnyFunction*

A vector-valued function.

property shape: Tuple[int, Ellipsis]

Get the value shape of the function.

Returns
The value shape

_vec: tuple[*ScalarFunction*, Ellipsis]

__len__()

Get the length of the vector.

__getitem__(*key*) → *ScalarFunction* | *VectorFunction*

Get a component or slice of the function.

__add__(*other: Any*) → *VectorFunction*

Add.

__radd__(*other: Any*) → *VectorFunction*

Add.

__sub__(*other: Any*) → *VectorFunction*

Subtract.

__rsub__(*other: Any*) → *VectorFunction*

Subtract.

__neg__() → *VectorFunction*

Negate.

__truediv__(*other: Any*) → *VectorFunction*

Divide.

__rtruediv__(*other: Any*) → *VectorFunction*

Divide.

__mul__(*other: Any*) → *VectorFunction*

Multiply.

__rmul__(*other: Any*) → *VectorFunction*

Multiply.

__matmul__(*other: Any*) → *VectorFunction*

Multiply.

__rmatmul__(*other: Any*) → *VectorFunction*

Multiply.

__pow__(*other: Any*) → *VectorFunction*

Raise to a power.

as_sympy() → SympyFormat

Convert to a sympy expression.

Returns
A Sympy expression

as_tex() → str

Convert to a TeX expression.

Returns
A TeX string

subs(*vars: symfem.symbols.AxisVariables*, *values: ValuesToSubstitute*) → *VectorFunction*

Substitute values into the function.

Parameters

- **vars** – The variables to substitute out
- **values** – The value to substitute in

Returns
The substituted function

diff(*variable: sympy.core.symbol.Symbol*) → *VectorFunction*

Differentiate the function.

Parameters
variable – The variable to differentiate with respect to

Returns
The differentiated function

abstract directional_derivative(*direction: symfem.geometry.PointType*)

Compute a directional derivative.

Parameters
direction – The diection

Returns
The directional derivatve

abstract jacobian_component(*component: Tuple[int, int]*)

Compute a component of the jacobian.

Parameters
component – The component

Returns
The component of the jacobian

abstract jacobian(*dim: int*) → *MatrixFunction*

Compute the jacobian.

Parameters
dim – The topological dimension of the cell

Returns
The jacobian

dot(*other_in: FunctionInput*) → *ScalarFunction*

Compute the dot product with another function.

Parameters
other_in – The function to multiply with

Returns
The product

cross(*other_in: FunctionInput*) → *VectorFunction* | *ScalarFunction*

Compute the cross product with another function.

Parameters
other_in – The function to multiply with

Returns
The cross product

div() → *ScalarFunction*

Compute the div of the function.

Returns
The divergence

grad()

Compute the grad of the function.

Returns
The gradient

curl() → *VectorFunction*

Compute the curl of the function.

Returns
The curl

norm() → *ScalarFunction*

Compute the norm of the function.

Returns
The norm

abstract integral(*domain:* symfem.references.Reference, *vars:*
symfem.symbols.AxisVariablesNotSingle = x, *dummy_vars:*
symfem.symbols.AxisVariablesNotSingle = t) → *ScalarFunction*

Compute the integral of the function.

Parameters

- **domain** – The domain of the integral
- **vars** – The variables to integrate with respect to
- **dummy_vars** – The dummy variables to use inside the integral

Returns
The integral

__iter__()
Get iterable.

__next__()
Get next item.

plot_values(*reference:* symfem.references.Reference, *img: Any*, *value_scale: sympy.core.expr.Expr = sympy.Integer(1)*, *n: int = 6*)
Plot the function's values.

Parameters

- **reference** – The reference cell
- **img** – The image to plot on
- **value_scale** – The scale factor for the function values
- **n** – The number of points per side for plotting

with_floats() → *AnyFunction*
Return a version the function with floats as coefficients.

Returns
A version the function with floats as coefficients

maximum_degree(*cell:* symfem.references.Reference) → int
Return the maximum degree of the function on a reference cell.

This function returns the order of the lowerst order Lagrange space on the input cell that includes this function.

Parameters
cell – The cell

Returns
A version the function with floats as coefficients

class symfem.functions.**MatrixFunction**(*mat: Tuple[Tuple[*AnyFunction | *int* | *sympy.core.expr.Expr, Ellipsis], Ellipsis] | Tuple[List[*AnyFunction | *int* | *sympy.core.expr.Expr], Ellipsis] | List[Tuple[*AnyFunction | *int* | *sympy.core.expr.Expr, Ellipsis]] | List[List[*AnyFunction | *int* | *sympy.core.expr.Expr]] | *sympy.matrices.dense.MutableDenseMatrix*)
Bases: *AnyFunction*

A matrix-valued function.

property shape: Tuple[int, Ellipsis]
Get the value shape of the function.

Returns
The value shape

_mat: Tuple[Tuple[*ScalarFunction*, Ellipsis], Ellipsis]

__getitem__(*key*) → *ScalarFunction* | *VectorFunction*
Get a component or slice of the function.

row(*n: int*) → *VectorFunction*

Get a row of the matrix.

Parameters
n – The row number

Returns
The row of the matrix

col(*n: int*) → *VectorFunction*

Get a colunm of the matrix.

Parameters
n – The column number

Returns
The column of the matrix

__add__(*other: Any*) → *MatrixFunction*

Add.

__radd__(*other: Any*) → *MatrixFunction*

Add.

__sub__(*other: Any*) → *MatrixFunction*

Subtract.

__rsub__(*other: Any*) → *MatrixFunction*

Subtract.

__neg__() → *MatrixFunction*

Negate.

__truediv__(*other: Any*) → *MatrixFunction*

Divide.

__rtruediv__(*other: Any*) → *MatrixFunction*

Divide.

__mul__(*other: Any*) → *MatrixFunction*

Multiply.

__rmul__(*other: Any*) → *MatrixFunction*

Multiply.

__matmul__(*other: Any*) → *MatrixFunction*

Multiply.

__rmatmul__(*other: Any*) → *MatrixFunction*

Multiply.

__pow__(*other: Any*) → *MatrixFunction*

Raise to a power.

as_sympy() → SympyFormat

Convert to a sympy expression.

Returns
A Sympy expression

as_tex() → str

Convert to a TeX expression.

Returns
A TeX string

subs(*vars: symfem.symbols.AxisVariables*, *values: ValuesToSubstitute*) → *MatrixFunction*

Substitute values into the function.

Parameters

- **vars** – The variables to substitute out
- **values** – The value to substitute in

Returns
The substituted function

diff(*variable: sympy.core.symbol.Symbol*) → *MatrixFunction*

Differentiate the function.

Parameters
variable – The variable to differentiate with respect to

Returns
The differentiated function

abstract directional_derivative(*direction: symfem.geometry.PointType*)

Compute a directional derivative.

Parameters
direction – The diection

Returns
The directional derivatve

abstract jacobian_component(*component: Tuple[int, int]*)

Compute a component of the jacobian.

Parameters
component – The component

Returns
The component of the jacobian

abstract jacobian(*dim: int*)

Compute the jacobian.

Parameters
dim – The topological dimension of the cell

Returns
The jacobian

dot(*other_in: FunctionInput*) → *ScalarFunction*

Compute the dot product with another function.

Parameters
other_in – The function to multiply with

Returns
The product

cross(*other_in: FunctionInput*)

Compute the cross product with another function.

Parameters
other_in – The function to multiply with

Returns
The cross product

div()

Compute the div of the function.

Returns
The divergence

grad()

Compute the grad of the function.

Returns
The gradient

curl()

Compute the curl of the function.

Returns
The curl

abstract norm() → *ScalarFunction*

Compute the norm of the function.

Returns
The norm

abstract integral(*domain:* symfem.references.Reference, *vars:* *symfem.symbols.AxisVariablesNotSingle* = *x*, *dummy_vars:* *symfem.symbols.AxisVariablesNotSingle* = *t*) → *ScalarFunction*

Compute the integral of the function.

Parameters

- **domain** – The domain of the integral
- **vars** – The variables to integrate with respect to
- **dummy_vars** – The dummy variables to use inside the integral

Returns
The integral

det() → *ScalarFunction*

Compute the determinant.

Returns
The deteminant

transpose() → *MatrixFunction*

Compute the transpose.

Returns
The transpose

with_floats() → *AnyFunction*

Return a version the function with floats as coefficients.

Returns
A version the function with floats as coefficients

maximum_degree(*cell:* symfem.references.Reference) → int

Return the maximum degree of the function on a reference cell.

This function returns the order of the lowerst order Lagrange space on the input cell that includes this function.

Parameters
cell – The cell

Returns
A version the function with floats as coefficients

symfem.functions.**FunctionInput**

symfem.functions.**parse_function_input**(*f: FunctionInput*) → *AnyFunction*

Parse a function.

Parameters
f – A function

Returns
The function as a Symfem function

symfem.functions.**parse_function_list_input**(*functions: List[FunctionInput] | Tuple[FunctionInput, Ellipsis]*) → List[*AnyFunction*]

Parse a list of functions.

Parameters
functions – The functions

Returns
The functions as Symfem functions

symfem.geometry

Geometry.

Module Contents

Functions

_is_close(→ bool)	Check if a Sympy expression is close to an int.
parse_set_of_points_input(→ SetOfPoints)	Convert an input set of points to the correct format.
parse_point_input(→ PointType)	Convert an input point to the correct format.
_vsub(→ PointType)	Subtract.
_vdot(→ sympy.core.expr.Expr)	Compute dot product.
point_in_interval(→ bool)	Check if a point is inside an interval.
point_in_triangle(→ bool)	Check if a point is inside a triangle.
point_in_quadrilateral(→ bool)	Check if a point is inside a quadrilateral.
point_in_tetrahedron(→ bool)	Check if a point is inside a tetrahedron.

Attributes

PointType
SetOfPoints
PointTypeInput
SetOfPointsInput

symfem.geometry.**PointType**

symfem.geometry.**SetOfPoints**

symfem.geometry.**PointTypeInput**

symfem.geometry.**SetOfPointsInput**

symfem.geometry.**_is_close**(*a: sympy.core.expr.Expr*, *b: int*) → bool

Check if a Sympy expression is close to an int.

Parameters

- **a** – A Sympy expression
- **b** – An integer

Returns

Is the sympy expression close to the integer?

symfem.geometry.**parse_set_of_points_input**(*points: SetOfPointsInput*) → SetOfPoints

Convert an input set of points to the correct format.

Parameters

points – A set of points in some input format

Returns

A set of points

symfem.geometry.**parse_point_input**(*point: PointTypeInput*) → PointType

Convert an input point to the correct format.

Parameters

point – A point in some input fotmat

Returns

A point

symfem.geometry.**_vsub**(*v: PointType*, *w: PointType*) → PointType

Subtract.

Parameters

- **v** – A vector
- **w** – A vector

Returns

The vector v - w

symfem.geometry.**_vdot**(*v: PointType*, *w: PointType*) → sympy.core.expr.Expr

Compute dot product.

Parameters

- **v** – A vector
- **w** – A vector

Returns

The dot product of v and w

symfem.geometry.**point_in_interval**(*point: PointType*, *interval: SetOfPoints*) → bool

Check if a point is inside an interval.

Parameters

- **point** – The point
- **interval** – The vertices of the interval

Returns
Is the point inside the interval?

symfem.geometry.**point_in_triangle**(*point: PointType*, *triangle: SetOfPoints*) → bool

Check if a point is inside a triangle.

Parameters

- **point** – The point
- **traingle** – The vertices of the triangle

Returns
Is the point inside the triangle?

symfem.geometry.**point_in_quadrilateral**(*point: PointType*, *quad: SetOfPoints*) → bool

Check if a point is inside a quadrilateral.

Parameters

- **point** – The point
- **traingle** – The vertices of the quadrilateral

Returns
Is the point inside the quadrilateral?

symfem.geometry.**point_in_tetrahedron**(*point: PointType*, *tetrahedron: SetOfPoints*) → bool

Check if a point is inside a tetrahedron.

Parameters

- **point** – The point
- **traingle** – The vertices of the tetrahedron

Returns
Is the point inside the tetrahedron?

symfem.mappings

Functions to map functions between cells.

Module Contents

Functions

identity(→ symfem.functions.AnyFunction)	Map functions.
l2(→ symfem.functions.AnyFunction)	Map functions, scaling by the determinant of the jacobian.
covariant(→ symfem.functions.AnyFunction)	Map H(curl) functions.
contravariant(→ symfem.functions.AnyFunction)	Map H(div) functions.
double_covariant(→ symfem.functions.MatrixFunction)	Map matrix functions.
double_contravariant(→ symfem.functions.MatrixFunction)	Map matrix functions.
identity_inverse_transpose(→ symfem.functions.AnyFunction)	Inverse transpose of identity().
l2_inverse_transpose(→ symfem.functions.AnyFunction)	Inverse transpose of l2().
covariant_inverse_transpose(→ symfem.functions.AnyFunction)	Inverse transpose of covariant().
contravariant_inverse_transpose(...)	Inverse transpose of contravariant().
identity_inverse(→ symfem.functions.AnyFunction)	Inverse of identity().
l2_inverse(→ symfem.functions.AnyFunction)	Inverse of l2().
covariant_inverse(→ symfem.functions.AnyFunction)	Inverse of covariant().
contravariant_inverse(→ symfem.functions.AnyFunction)	Inverse of contravariant().
double_covariant_inverse(→ symfem.functions.AnyFunction)	Inverse of double_covariant().
double_contravariant_inverse(...)	Inverse of double_contravariant().
get_mapping(...)	Get a mapping.

exception symfem.mappings.**MappingNotImplemented**

Bases: NotImplementedError

Exception thrown when a mapping is not implemented for an element.

symfem.mappings.**identity**(*f_in: symfem.functions.FunctionInput*, *map: symfem.geometry.PointType*, *inverse_map: symfem.geometry.PointType*, *substitute: bool = True*) → *symfem.functions.AnyFunction*

Map functions.

Parameters

- **f_in** – The function
- **map** – The map from the reference cell to the physical cell
- **inverse_map** – The map to the reference cell from the physical cell
- **substitute** – Should the inverse map be substituted in?

Returns

The mapped function

symfem.mappings.**l2**(*f_in: symfem.functions.FunctionInput*, *map: symfem.geometry.PointType*, *inverse_map: symfem.geometry.PointType*, *substitute: bool = True*) → *symfem.functions.AnyFunction*

Map functions, scaling by the determinant of the jacobian.

Parameters

- **f_in** – The function
- **map** – The map from the reference cell to the physical cell
- **inverse_map** – The map to the reference cell from the physical cell
- **substitute** – Should the inverse map be substituted in?

Returns
The mapped function

symfem.mappings.**covariant**(*f_in: symfem.functions.FunctionInput*, *map: symfem.geometry.PointType*, *inverse_map: symfem.geometry.PointType*, *substitute: bool = True*) → *symfem.functions.AnyFunction*

Map H(curl) functions.

Parameters

- **f_in** – The function
- **map** – The map from the reference cell to the physical cell
- **inverse_map** – The map to the reference cell from the physical cell
- **substitute** – Should the inverse map be substituted in?

Returns
The mapped function

symfem.mappings.**contravariant**(*f_in: symfem.functions.FunctionInput*, *map: symfem.geometry.PointType*, *inverse_map: symfem.geometry.PointType*, *substitute: bool = True*) → *symfem.functions.AnyFunction*

Map H(div) functions.

Parameters

- **f_in** – The function
- **map** – The map from the reference cell to the physical cell
- **inverse_map** – The map to the reference cell from the physical cell
- **substitute** – Should the inverse map be substituted in?

Returns
The mapped function

symfem.mappings.**double_covariant**(*f_in: symfem.functions.FunctionInput*, *map: symfem.geometry.PointType*, *inverse_map: symfem.geometry.PointType*, *substitute: bool = True*) → *symfem.functions.MatrixFunction*

Map matrix functions.

Parameters

- **f_in** – The function
- **map** – The map from the reference cell to the physical cell
- **inverse_map** – The map to the reference cell from the physical cell
- **substitute** – Should the inverse map be substituted in?

Returns
The mapped function

symfem.mappings.**double_contravariant**(*f_in: symfem.functions.FunctionInput*, *map: symfem.geometry.PointType*, *inverse_map: symfem.geometry.PointType*, *substitute: bool = True*) → *symfem.functions.MatrixFunction*

Map matrix functions.

Parameters

- **f_in** – The function
- **map** – The map from the reference cell to the physical cell
- **inverse_map** – The map to the reference cell from the physical cell
- **substitute** – Should the inverse map be substituted in?

Returns
The mapped function

symfem.mappings.**identity_inverse_transpose**(*f_in: symfem.functions.FunctionInput*, *map: symfem.geometry.PointType*, *inverse_map: symfem.geometry.PointType*, *substitute: bool = True*) → *symfem.functions.AnyFunction*

Inverse transpose of identity().

Parameters

- **f_in** – The function
- **map** – The map from the reference cell to the physical cell
- **inverse_map** – The map to the reference cell from the physical cell
- **substitute** – Should the inverse map be substituted in?

Returns
The mapped function

symfem.mappings.**l2_inverse_transpose**(*f_in: symfem.functions.FunctionInput*, *map: symfem.geometry.PointType*, *inverse_map: symfem.geometry.PointType*, *substitute: bool = True*) → *symfem.functions.AnyFunction*

Inverse transpose of l2().

Parameters

- **f_in** – The function
- **map** – The map from the reference cell to the physical cell
- **inverse_map** – The map to the reference cell from the physical cell
- **substitute** – Should the inverse map be substituted in?

Returns
The mapped function

symfem.mappings.**covariant_inverse_transpose**(*f_in: symfem.functions.FunctionInput*, *map: symfem.geometry.PointType*, *inverse_map: symfem.geometry.PointType*, *substitute: bool = True*) → *symfem.functions.AnyFunction*

Inverse transpose of covariant().

Parameters

- **f_in** – The function
- **map** – The map from the reference cell to the physical cell
- **inverse_map** – The map to the reference cell from the physical cell
- **substitute** – Should the inverse map be substituted in?

Returns
The mapped function

symfem.mappings.**contravariant_inverse_transpose**(*f_in: symfem.functions.FunctionInput*, *map: symfem.geometry.PointType*, *inverse_map: symfem.geometry.PointType*, *substitute: bool = True*) → *symfem.functions.AnyFunction*

Inverse transpose of contravariant().

Parameters

- **f_in** – The function
- **map** – The map from the reference cell to the physical cell
- **inverse_map** – The map to the reference cell from the physical cell
- **substitute** – Should the inverse map be substituted in?

Returns
The mapped function

symfem.mappings.**identity_inverse**(*f_in: symfem.functions.FunctionInput*, *map: symfem.geometry.PointType*, *inverse_map: symfem.geometry.PointType*, *substitute: bool = True*) → *symfem.functions.AnyFunction*

Inverse of identity().

Parameters

- **f_in** – The function
- **map** – The map from the reference cell to the physical cell
- **inverse_map** – The map to the reference cell from the physical cell
- **substitute** – Should the inverse map be substituted in?

Returns
The mapped function

symfem.mappings.**l2_inverse**(*f_in: symfem.functions.FunctionInput*, *map: symfem.geometry.PointType*, *inverse_map: symfem.geometry.PointType*, *substitute: bool = True*) → *symfem.functions.AnyFunction*

Inverse of l2().

Parameters

- **f_in** – The function
- **map** – The map from the reference cell to the physical cell
- **inverse_map** – The map to the reference cell from the physical cell
- **substitute** – Should the inverse map be substituted in?

Returns
The mapped function

symfem.mappings.**covariant_inverse**(*f_in: symfem.functions.FunctionInput*, *map: symfem.geometry.PointType*, *inverse_map: symfem.geometry.PointType*, *substitute: bool = True*) → *symfem.functions.AnyFunction*

Inverse of covariant().

Parameters

- **f_in** – The function

- **map** – The map from the reference cell to the physical cell
- **inverse_map** – The map to the reference cell from the physical cell
- **substitute** – Should the inverse map be substituted in?

Returns
The mapped function

symfem.mappings.**contravariant_inverse**(*f_in: symfem.functions.FunctionInput*, *map: symfem.geometry.PointType*, *inverse_map: symfem.geometry.PointType*, *substitute: bool = True*) → *symfem.functions.AnyFunction*

Inverse of contravariant().

Parameters

- **f_in** – The function
- **map** – The map from the reference cell to the physical cell
- **inverse_map** – The map to the reference cell from the physical cell
- **substitute** – Should the inverse map be substituted in?

Returns
The mapped function

symfem.mappings.**double_covariant_inverse**(*f_in: symfem.functions.FunctionInput*, *map: symfem.geometry.PointType*, *inverse_map: symfem.geometry.PointType*, *substitute: bool = True*) → *symfem.functions.AnyFunction*

Inverse of double_covariant().

Parameters

- **f_in** – The function
- **map** – The map from the reference cell to the physical cell
- **inverse_map** – The map to the reference cell from the physical cell
- **substitute** – Should the inverse map be substituted in?

Returns
The mapped function

symfem.mappings.**double_contravariant_inverse**(*f_in: symfem.functions.FunctionInput*, *map: symfem.geometry.PointType*, *inverse_map: symfem.geometry.PointType*, *substitute: bool = True*) → *symfem.functions.AnyFunction*

Inverse of double_contravariant().

Parameters

- **f_in** – The function
- **map** – The map from the reference cell to the physical cell
- **inverse_map** – The map to the reference cell from the physical cell
- **substitute** – Should the inverse map be substituted in?

Returns
The mapped function

symfem.mappings.**get_mapping**(*mapname: str*, *inverse: bool = False*, *transpose: bool = False*) → Callable[[symfem.functions.FunctionInput, symfem.geometry.PointType, symfem.geometry.PointType, bool], *symfem.functions.AnyFunction*]

Get a mapping.

Parameters

- **mapname** – The name of the mapping
- **inverse** – Should the map be inverted
- **transpose** – Should the map be transposed

Returns

A function that performs the mapping

symfem.moments

Functions to create integral moments.

Module Contents

Functions

_extract_moment_data(→ MomentType)	Get the information for a moment.
make_integral_moment_dofs(...)	Generate DOFs due to integral moments on sub entities.

Attributes

MomentType
SingleMomentTypeInput
MomentTypeInput

symfem.moments.**MomentType**

symfem.moments.**SingleMomentTypeInput**

symfem.moments.**MomentTypeInput**

symfem.moments.**_extract_moment_data**(*moment_data: MomentTypeInput*, *sub_type: str*) → MomentType

Get the information for a moment.

Parameters

- **moment_data** – The moment data
- **sub_type** – The subentity type

Returns

The moment type, finite elment, order, mapping, and keyword arguments for the moment

symfem.moments.**make_integral_moment_dofs**(*reference:* symfem.references.Reference, *vertices: MomentTypeInput | None = None*, *edges: MomentTypeInput | None = None*, *faces: MomentTypeInput | None = None*, *volumes: MomentTypeInput | None = None*, *cells: MomentTypeInput | None = None*, *facets: MomentTypeInput | None = None*, *ridges: MomentTypeInput | None = None*, *peaks: MomentTypeInput | None = None*) → List[*symfem.functionals.BaseFunctional*]

Generate DOFs due to integral moments on sub entities.

Parameters

- **reference** – The reference cell.
- **vertices** – DOFs on dimension 0 entities.
- **edges** – DOFs on dimension 1 entities.
- **faces** – DOFs on dimension 2 entities.
- **volumes** – DOFs on dimension 3 entities.
- **cells** – DOFs on codimension 0 entities.
- **facets** – DOFs on codimension 1 entities.
- **ridges** – DOFs on codimension 2 entities.
- **peaks** – DOFs on codimension 3 entities.

Returns

A list of DOFs for the element

symfem.piecewise_functions

Piecewise basis function classes.

Module Contents

Classes

PiecewiseFunction	A piecewise function.

Functions

_piece_reference(tdim, shape)	Create a reference element for a single piece.

class symfem.piecewise_functions.**PiecewiseFunction**(*pieces: Dict[symfem.geometry.SetOfPointsInput, symfem.functions.FunctionInput]*, *tdim: int*)

Bases: *symfem.functions.AnyFunction*

A piecewise function.

property pieces: Dict[symfem.geometry.SetOfPoints, *symfem.functions.AnyFunction*]

Get the pieces of the function.

Returns
The function pieces

property shape: Tuple[int, Ellipsis]

Get the value shape of the function.

Returns
The value shape

_pieces: Dict[symfem.geometry.SetOfPoints, *symfem.functions.AnyFunction*]

__len__()

Get the length of the vector.

as_sympy() → symfem.functions.SympyFormat

Convert to a sympy expression.

Returns
A Sympy expression

as_tex() → str

Convert to a TeX expression.

Returns
A TeX string

get_piece(*point: symfem.geometry.PointType*) → *symfem.functions.AnyFunction*

Get a piece of the function.

Parameters
point – The point to get the piece at

Returns
The piece of the function that is valid at that point

__getitem__(*key*) → *PiecewiseFunction*

Get a component or slice of the function.

__eq__(*other: Any*) → bool

Check if two functions are equal.

__add__(*other: Any*) → *PiecewiseFunction*

Add.

__radd__(*other: Any*) → *PiecewiseFunction*

Add.

__sub__(*other: Any*) → *PiecewiseFunction*

Subtract.

__rsub__(*other: Any*) → *PiecewiseFunction*

Subtract.

__truediv__(*other: Any*) → *PiecewiseFunction*

Divide.

__rtruediv__(*other: Any*) → *PiecewiseFunction*

Divide.

__mul__(*other: Any*) → *PiecewiseFunction*

Multiply.

__rmul__(*other: Any*) → *PiecewiseFunction*

Multiply.

__matmul__(*other: Any*) → *PiecewiseFunction*

Multiply.

__rmatmul__(*other: Any*) → *PiecewiseFunction*

Multiply.

__pow__(*other: Any*) → *PiecewiseFunction*

Raise to a power.

__neg__() → *PiecewiseFunction*

Negate.

subs(*vars: symfem.symbols.AxisVariables*, *values: symfem.functions.ValuesToSubstitute*) → *PiecewiseFunction*

Substitute values into the function.

Parameters

- **vars** – The variables to substitute out
- **values** – The value to substitute in

Returns

The substituted function

diff(*variable: sympy.core.symbol.Symbol*) → *PiecewiseFunction*

Differentiate the function.

Parameters

variable – The variable to differentiate with respect to

Returns

The differentiated function

directional_derivative(*direction: symfem.geometry.PointType*) → *PiecewiseFunction*

Compute a directional derivative.

Parameters

direction – The diection

Returns

The directional derivative

jacobian_component(*component: Tuple[int, int]*) → *PiecewiseFunction*

Compute a component of the jacobian.

Parameters

component – The component

Returns

The component of the jacobian

jacobian(*dim: int*) → *PiecewiseFunction*

Compute the jacobian.

Parameters

dim – The topological dimension of the cell

Returns

The jacobian

dot(*other_in: symfem.functions.FunctionInput*) → *PiecewiseFunction*

Compute the dot product with another function.

Parameters
other_in – The function to multiply with

Returns
The product

cross(*other_in: symfem.functions.FunctionInput*) → *PiecewiseFunction*

Compute the cross product with another function.

Parameters
other_in – The function to multiply with

Returns
The cross product

div() → *PiecewiseFunction*

Compute the div of the function.

Returns
The divergence

grad(*dim: int*) → *PiecewiseFunction*

Compute the grad of the function.

Returns
The gradient

curl() → *PiecewiseFunction*

Compute the curl of the function.

Returns
The curl

norm() → *PiecewiseFunction*

Compute the norm of the function.

Returns
The norm

integral(*domain:* symfem.references.Reference, *vars: symfem.symbols.AxisVariablesNotSingle = x*, *dummy_vars: symfem.symbols.AxisVariablesNotSingle = t*) → *symfem.functions.ScalarFunction*

Compute the integral of the function.

Parameters

- **domain** – The domain of the integral
- **vars** – The variables to integrate with respect to
- **dummy_vars** – The dummy variables to use inside the integral

Returns
The integral

det() → *PiecewiseFunction*

Compute the determinant.

Returns
The deteminant

transpose() → *PiecewiseFunction*

Compute the transpose.

Returns
The transpose

map_pieces(*fwd_map: symfem.geometry.PointType*)

Map the function's pieces.

Parameters
fwd_map – The map from the reference cell to a physical cell

Returns
The mapped pieces

plot_values(*reference:* symfem.references.Reference, *img: Any*, *value_scale: sympy.core.expr.Expr = sympy.Integer(1)*, *n: int = 6*)

Plot the function's values.

Parameters

- **reference** – The reference cell
- **img** – The image to plot on
- **value_scale** – The scale factor for the function values
- **n** – The number of points per side for plotting

with_floats() → *symfem.functions.AnyFunction*

Return a version the function with floats as coefficients.

Returns
A version the function with floats as coefficients

abstract maximum_degree(*cell:* symfem.references.Reference) → int

Return the maximum degree of the function on a reference cell.

This function returns the order of the lowerst order Lagrange space on the input cell that includes this function.

Parameters
cell – The cell

Returns
A version the function with floats as coefficients

symfem.piecewise_functions.**_piece_reference**(*tdim*, *shape*)

Create a reference element for a single piece.

symfem.plotting

Plotting.

Module Contents

Classes

Colors	Class storing colours used in diagrams.
PictureElement	An element in a picture.
Line	A line.
Bezier	A Bezier curve.
Arrow	An arrow.
NCircle	A circle containing a number.
Fill	A filled polygon.
Math	A mathematical symbol.
Picture	A picture.

Functions

tex_font_size(n)	Convert a font size to a TeX size command.

Attributes

PointOrFunction
SetOfPointsOrFunctions
colors

symfem.plotting.**PointOrFunction**

symfem.plotting.**SetOfPointsOrFunctions**

symfem.plotting.**tex_font_size**(*n: int*)

Convert a font size to a TeX size command.

Parameters
n – Font size

Returns
TeX size command

class symfem.plotting.**Colors**

Class storing colours used in diagrams.

BLACK = '#000000'

WHITE = '#FFFFFF'

ORANGE = '#FF8800'

BLUE = '#44AAFF'

GREEN = '#55FF00'

PURPLE = '#DD2299'

GRAY = '#AAAAAA'

entity(*n: int*) → str

Get the color used for an entity of a given dimension.

Parameters
n – The dimension of the entity

Returns
The color used for entities of the given dimension

get_tikz_name(*name: str*) → str

Get the name of the colour to be used in Tikz.

Parameters
name – HTML name of the color

Returns
The Tikz name of the color

get_tikz_definitions() → str

Get the definitions of colours used in Tikz.

Returns
Definitions of Tikz colors

symfem.plotting.**colors**

class symfem.plotting.**PictureElement**

Bases: abc.ABC

An element in a picture.

abstract property points: symfem.geometry.SetOfPoints

Get set of points used by this element.

Returns
A set of points

abstract as_svg(*map_pt: Callable[[symfem.geometry.PointType], Tuple[float, float]]*) → str

Return SVG format.

Parameters
map_pt – A function that adjust the origin and scales the picture

Returns
An SVG string

abstract as_tikz(*map_pt: Callable[[symfem.geometry.PointType], Tuple[float, float]]*) → str

Return Tikz format.

Parameters
map_pt – A function that adjust the origin and scales the picture

Returns
A Tikz string

minx() → sympy.core.expr.Expr

Get the minimum x-coordinate.

Returns
The minimum x-coordinate

miny() → sympy.core.expr.Expr

Get the minimum y-coordinate.

Returns
The minimum y-coordinate

maxx() → sympy.core.expr.Expr

Get the maximum x-coordinate.

Returns
The maximum x-coordinate

maxy() → sympy.core.expr.Expr

Get the maximum y-coordinate.

Returns
The maximum y-coordinate

class symfem.plotting.**Line**(*start: symfem.geometry.PointType*, *end: symfem.geometry.PointType*, *color: str*, *width: float*)

Bases: *PictureElement*

A line.

property points: symfem.geometry.SetOfPoints

Get set of points used by this element.

Returns
A set of points

as_svg(*map_pt: Callable[[symfem.geometry.PointType], Tuple[float, float]]*) → str

Return SVG format.

Parameters
map_pt – A function that adjust the origin and scales the picture

Returns
An SVG string

as_tikz(*map_pt: Callable[[symfem.geometry.PointType], Tuple[float, float]]*) → str

Return Tikz format.

Parameters
map_pt – A function that adjust the origin and scales the picture

Returns
A Tikz string

class symfem.plotting.**Bezier**(*start: symfem.geometry.PointType*, *mid1: symfem.geometry.PointType*, *mid2: symfem.geometry.PointType*, *end: symfem.geometry.PointType*, *color: str*, *width: float*)

Bases: *PictureElement*

A Bezier curve.

property points: symfem.geometry.SetOfPoints

Get set of points used by this element.

Returns
A set of points

as_svg(*map_pt: Callable[[symfem.geometry.PointType], Tuple[float, float]]*) → str

Return SVG format.

Parameters
map_pt – A function that adjust the origin and scales the picture

Returns
An SVG string

as_tikz(*map_pt: Callable[[symfem.geometry.PointType], Tuple[float, float]]*) → str

Return Tikz format.

Parameters
map_pt – A function that adjust the origin and scales the picture

Returns
A Tikz string

class symfem.plotting.**Arrow**(*start: symfem.geometry.PointType*, *end: symfem.geometry.PointType*, *color: str*, *width: float*)

Bases: *PictureElement*

An arrow.

property points: symfem.geometry.SetOfPoints

Get set of points used by this element.

Returns
A set of points

as_svg(*map_pt: Callable[[symfem.geometry.PointType], Tuple[float, float]]*) → str

Return SVG format.

Parameters
map_pt – A function that adjust the origin and scales the picture

Returns
An SVG string

as_tikz(*map_pt: Callable[[symfem.geometry.PointType], Tuple[float, float]]*) → str

Return Tikz format.

Parameters
map_pt – A function that adjust the origin and scales the picture

Returns
A Tikz string

class symfem.plotting.**NCircle**(*centre: symfem.geometry.PointType*, *number: int*, *color: str*, *text_color: str*, *fill_color: str*, *radius: float*, *font_size: int | None*, *width: float*, *font: str*)

Bases: *PictureElement*

A circle containing a number.

property points: symfem.geometry.SetOfPoints

Get set of points used by this element.

Returns
A set of points

as_svg(*map_pt: Callable[[symfem.geometry.PointType], Tuple[float, float]]*) → str

Return SVG format.

Parameters
map_pt – A function that adjust the origin and scales the picture

Returns
An SVG string

as_tikz(*map_pt: Callable[[symfem.geometry.PointType], Tuple[float, float]]*) → str

Return Tikz format.

Parameters
map_pt – A function that adjust the origin and scales the picture

Returns
A Tikz string

class symfem.plotting.**Fill**(*vertices: symfem.geometry.SetOfPoints*, *color: str*, *opacity: float*)

Bases: *PictureElement*

A filled polygon.

property **points: symfem.geometry.SetOfPoints**
Get set of points used by this element.

Returns
A set of points

as_svg(*map_pt: Callable[[symfem.geometry.PointType], Tuple[float, float]]*) → str
Return SVG format.

Parameters
map_pt – A function that adjust the origin and scales the picture

Returns
An SVG string

as_tikz(*map_pt: Callable[[symfem.geometry.PointType], Tuple[float, float]]*) → str
Return Tikz format.

Parameters
map_pt – A function that adjust the origin and scales the picture

Returns
A Tikz string

class symfem.plotting.**Math**(*point: symfem.geometry.PointType*, *math: str*, *color: str*, *font_size: int*, *anchor: str*)

Bases: *PictureElement*

A mathematical symbol.

property **points: symfem.geometry.SetOfPoints**
Get set of points used by this element.

Returns
A set of points

as_svg(*map_pt: Callable[[symfem.geometry.PointType], Tuple[float, float]]*) → str
Return SVG format.

Parameters
map_pt – A function that adjust the origin and scales the picture

Returns
An SVG string

as_tikz(*map_pt: Callable[[symfem.geometry.PointType], Tuple[float, float]]*) → str
Return Tikz format.

Parameters
map_pt – A function that adjust the origin and scales the picture

Returns
A Tikz string

class symfem.plotting.**Picture**(*padding: sympy.core.expr.Expr = sympy.Integer(25)*, *scale: int | None = None*, *width: int | None = None*, *height: int | None = None*, *axes_3d: symfem.geometry.SetOfPointsInput | None = None*, *dof_arrow_size: int | sympy.core.expr.Expr = 1*, *title: str | None = None*, *desc: str | None = None*, *svg_metadata: str | None = None*, *tex_comment: str | None = None*)

A picture.

axes_3d: symfem.geometry.SetOfPoints

z(*p_in: PointOrFunction*) → sympy.core.expr.Expr

Get the into/out-of-the-page component of a point.

Parameters
p_in – The point

Returns
The into/out-of-the-page component of the point

to_2d(*p: symfem.geometry.PointType*) → symfem.geometry.PointType

Map a point to 2D.

Parameters
p – The point

Returns
The projection of the point into 2 dimensions

parse_point(*p: PointOrFunction*) → symfem.geometry.PointType

Parse an input point.

Parameters
p – a point or a function

Returns
The point as a tuple of Sympy expressions

add_line(*start: PointOrFunction*, *end: PointOrFunction*, *color: str = colors.BLACK*, *width: float = 4.0*)

Add a line to the picture.

Parameters

- **start** – The start point of the line
- **end** – The end point of the line
- **color** – The color of the line
- **width** – The width of the line

add_bezier(*start: PointOrFunction*, *mid1: PointOrFunction*, *mid2: PointOrFunction*, *end: PointOrFunction*, *color: str = colors.BLACK*, *width: float = 4.0*)

Add a Bezier curve to the picture.

Parameters

- **start** – The start point of the Bezier curve
- **mid1** – The first control point
- **mid2** – The second control point
- **end** – The end point of the Bezier curve
- **color** – The color of the Bezier curve
- **width** – The width of the Bezier curve

add_arrow(*start: PointOrFunction*, *end: PointOrFunction*, *color: str = colors.BLACK*, *width: float = 4.0*)

Add an arrow to the picture.

Parameters

- **start** – The start point of the arrow

- **end** – The end point of the arrow
- **color** – The color of the arrow
- **width** – The width of the arrow

add_dof_marker(*point: PointOrFunction*, *number: int*, *color: str*, *bold: bool = True*)

Add a DOF marker.

Parameters

- **point** – The point
- **number** – The number to put in the marker
- **color** – The color of the marker
- **bold** – Should the marker be bold?

add_dof_arrow(*point: PointOrFunction*, *direction: PointOrFunction*, *number: int*, *color: str = colors.PURPLE*, *shifted: bool = False*, *bold: bool = True*)

Add a DOF arrow.

Parameters

- **point** – The point
- **direction** – The direction of the arrow
- **number** – The number to put in the marker
- **color** – The color of the marker
- **shifted** – Should the marker be shifted?
- **bold** – Should the marker be bold?

add_ncircle(*centre: PointOrFunction*, *number: int*, *color: str = 'red'*, *text_color: str = colors.BLACK*, *fill_color: str = colors.WHITE*, *radius: float = 20.0*, *font_size: int | None = None*, *width: float = 4.0*, *font: str = '"Varela Round",sans-serif'*)

Add a numbered circle to the picture.

Parameters

- **centre** – The centre points
- **number** – The number in the circle
- **color** – The color of the outline
- **text_color** – The color of the test
- **fill_color** – The colour of the background fill
- **radius** – The radius of the circle
- **font_size** – The font size
- **width** – The width of the line
- **font** – The font

add_math(*point: symfem.geometry.PointTypeInput*, *math: str*, *color: str = colors.BLACK*, *font_size: int = 35*, *anchor='center'*)

Create mathematical symbol.

Parameters

- **point** – The point to put the math
- **math** – The math
- **color** – The color of the math

- **font_size** – The font size
- **anchor** – The point on the equation to anchor to

add_fill(*vertices: SetOfPointsOrFunctions*, *color: str = 'red'*, *opacity: float = 1.0*)

Add a filled polygon to the picture.

Parameters

- **vertices** – The vertices of the polygon
- **color** – The color of the polygon
- **opacity** – The opacity of the polygon

compute_scale(*unit: str = 'px'*, *reverse_y: bool = True*) → Tuple[sympy.core.expr.Expr, sympy.core.expr.Expr, sympy.core.expr.Expr, Callable[[symfem.geometry.PointType], Tuple[float, float]]]

Compute the scale and size of the picture.

Parameters

- **unit** – The unit to use. Accepted values: px, cm, mm
- **reverse_y** – Should the y-axis be reversed?

Returns

The scale, height, and width of the image, and a mapping function

as_svg(*filename: str | None = None*) → str

Convert to an SVG.

Parameters

filename – The file name

Returns

The image as an SVG string

as_png(*filename: str*, *png_scale: float | None = None*, *png_width: int | None = None*, *png_height: int | None = None*)

Convert to a PNG.

Parameters

- **filename** – The file name
- **png_scale** – The scale of the png
- **png_width** – The width of the png
- **png_height** – The height of the png

as_tikz(*filename: str | None = None*) → str

Convert to tikz.

Parameters

filename – The file name

Returns

The image as a Tikz string

save(*filename: str | List[str]*, *plot_options: Dict[str, Any] = {}*)

Save the picture as a file.

Parameters

- **filename** – The file name
- **plot_options** – The plotting options

symfem.quadrature

Quadrature definitions.

Module Contents

Functions

equispaced(→ Tuple[List[Scalar], List[Scalar]])	Get equispaced points and weights.
lobatto(→ Tuple[List[Scalar], List[Scalar]])	Get Gauss-Lobatto-Legendre points and weights.
radau(→ Tuple[List[Scalar], List[Scalar]])	Get Radau points and weights.
legendre(→ Tuple[List[Scalar], List[Scalar]])	Get Gauss-Legendre points and weights.
get_quadrature(→ Tuple[List[Scalar], List[Scalar]])	Get quadrature points and weights.

Attributes

Scalar

symfem.quadrature.**Scalar**

symfem.quadrature.**equispaced**(*n: int*) → Tuple[List[Scalar], List[Scalar]]

Get equispaced points and weights.

Parameters
n – Number of points

Returns
Quadrature points and weights

symfem.quadrature.**lobatto**(*n: int*) → Tuple[List[Scalar], List[Scalar]]

Get Gauss-Lobatto-Legendre points and weights.

Parameters
n – Number of points

Returns
Quadrature points and weights

symfem.quadrature.**radau**(*n: int*) → Tuple[List[Scalar], List[Scalar]]

Get Radau points and weights.

Parameters
n – Number of points

Returns
Quadrature points and weights

symfem.quadrature.**legendre**(*n: int*) → Tuple[List[Scalar], List[Scalar]]

Get Gauss-Legendre points and weights.

Parameters
n – Number of points

Returns
Quadrature points and weights

symfem.quadrature.**get_quadrature**(*rule: str*, *n: int*) → Tuple[List[Scalar], List[Scalar]]

Get quadrature points and weights.

Parameters

- **rule** – The quadrature rule. Supported values: equispaced, lobatto, radau, legendre, gll
- **n** – Number of points

Returns

Quadrature points and weights

symfem.references

Reference elements.

Module Contents

Classes

Reference	A reference cell on which a finite element can be defined.
Point	A point.
Interval	An interval.
Triangle	A triangle.
Tetrahedron	A tetrahedron.
Quadrilateral	A quadrilateral.
Hexahedron	A hexahedron.
Prism	A (triangular) prism.
Pyramid	A (square-based) pyramid.
DualPolygon	A polygon on a barycentric dual grid.

Functions

_which_side(→ Optional[int])	Check which side of a line or plane a set of points are.
_vsub(→ symfem.geometry.PointType)	Subtract.
_vadd(→ symfem.geometry.PointType)	Add.
_vdot(→ sympy.core.expr.Expr)	Compute the dot product.
_vcross(→ symfem.geometry.PointType)	Compute the cross product.
_vnorm(→ sympy.core.expr.Expr)	Find the norm of a vector.
_vnormalise(→ symfem.geometry.PointType)	Normalise a vector.

Attributes

LatticeWithLines	
IntLimits	

symfem.references.**LatticeWithLines**

symfem.references.**IntLimits**

exception symfem.references.**NonDefaultReferenceError**

Bases: NotImplementedError

Exception to be thrown when an element can only be created on the default reference.

symfem.references.**_which_side**(*vs: symfem.geometry.SetOfPoints*, *p: symfem.geometry.PointType*, *q: symfem.geometry.PointType*) → int | None

Check which side of a line or plane a set of points are.

Parameters

- **vs** – The set of points
- **p** – A point on the line or plane
- **q** – Another point on the line (2D) or the normal to the plane (3D)

Returns

2 if the points are all to the left, 1 if the points are all to the left or on the line, 0 if the points are all on the line, -1 if the points are all to the right or on the line, -1 if the points are all to the right, None if there are some points on either side.

symfem.references.**_vsub**(*v: symfem.geometry.PointTypeInput*, *w: symfem.geometry.PointTypeInput*) → symfem.geometry.PointType

Subtract.

Parameters

- **v** – A vector
- **w** – A vector

Returns

The vector v - w

symfem.references.**_vadd**(*v: symfem.geometry.PointTypeInput*, *w: symfem.geometry.PointTypeInput*) → symfem.geometry.PointType

Add.

Parameters

- **v** – A vector
- **w** – A vector

Returns

The vector v + w

symfem.references.**_vdot**(*v: symfem.geometry.PointTypeInput*, *w: symfem.geometry.PointTypeInput*) → sympy.core.expr.Expr

Compute the dot product.

Parameters

- **v** – A vector
- **w** – A vector

Returns

The scalar v.w

symfem.references.**_vcross**(*v_in: symfem.geometry.PointTypeInput*, *w_in: symfem.geometry.PointTypeInput*) → symfem.geometry.PointType

Compute the cross product.

Parameters

- **v** – A vector
- **w** – A vector

Returns
The vector v x w

symfem.references._vnorm(*v_in: symfem.geometry.PointTypeInput*) → sympy.core.expr.Expr

Find the norm of a vector.

Parameters
v_in – A vector

Returns
The norm of v_in

symfem.references._vnormalise(*v_in: symfem.geometry.PointTypeInput*) → symfem.geometry.PointType

Normalise a vector.

Parameters
v_in – A vector

Returns
A unit vector pointing in the same direction as v_in

class symfem.references.Reference(*vertices: symfem.geometry.SetOfPointsInput = ()*)

Bases: abc.ABC

A reference cell on which a finite element can be defined.

property clockwise_vertices: symfem.geometry.SetOfPoints

Get list of vertices in clockwise order.

Returns
A list of vertices

__build__(*tdim: int*, *name: str*, *origin: symfem.geometry.PointTypeInput*, *axes: symfem.geometry.SetOfPointsInput*, *reference_vertices: symfem.geometry.SetOfPointsInput*, *vertices: symfem.geometry.SetOfPointsInput*, *edges: Tuple[Tuple[int, int], Ellipsis]*, *faces: Tuple[Tuple[int, Ellipsis], Ellipsis]*, *volumes: Tuple[Tuple[int, Ellipsis], Ellipsis]*, *sub_entity_types: List[List[str] | str | None]*, *simplex: bool = False*, *tp: bool = False*)

Create a reference cell.

Parameters

- **tdim** – The topological dimension of the cell
- **name** – The name of the cell
- **origin** – The coordinates of the origin
- **axes** – Vectors representing the axes of the cell
- **reference_vertices** – The vertices of the default version of this cell
- **vertices** – The vertices of this cell
- **edges** – Pairs of vertex numbers that form the edges of the cell
- **faces** – Tuples of vertex numbers that form the faces of the cell
- **volumes** – Tuples of vertex numbers that form the volumes of the cell
- **sub_entity_types** – The cell types of each sub-entity of the cell
- **simplex** – Is the cell a simplex (interval/triangle/tetrahedron)?
- **tp** – Is the cell a tensor product (interval/quadrilateral/hexahedron)?

__eq__(*other: object*) → bool

Check if two references are equal.

__hash__() → int

Check if two references are equal.

intersection(*other:* Reference) → *Reference* | None

Get the intersection of two references.

Returns
A reference element that is the intersection

abstract default_reference() → *Reference*

Get the default reference for this cell type.

Returns
The default reference

abstract make_lattice(*n: int*) → symfem.geometry.SetOfPoints

Make a lattice of points.

Parameters
n – The number of points along each edge

Returns
A lattice of points offset from the edge of the cell

abstract make_lattice_with_lines(*n: int*) → LatticeWithLines

Make a lattice of points, and a list of lines connecting them.

Parameters
n – The number of points along each edge

Returns
A lattice of points including the edges of the cell Pairs of point numbers that make a mesh of lines across the cell

make_lattice_float(*n: int*) → symfem.geometry.SetOfPoints

Make a lattice of points.

Parameters
n – The number of points along each edge

Returns
A lattice of points offset from the edge of the cell

make_lattice_with_lines_float(*n: int*) → LatticeWithLines

Make a lattice of points, and a list of lines connecting them.

Parameters
n – The number of points along each edge

Returns
A lattice of points including the edges of the cell Pairs of point numbers that make a mesh of lines across the cell

z_ordered_entities() → List[List[Tuple[int, int]]]

Get the subentities of the cell in back-to-front plotting order.

Returns
List of lists of subentity dimensions and numbers

z_ordered_entities_extra_dim() → List[List[Tuple[int, int]]]

Get the subentities in back-to-front plotting order when using an extra dimension.

Returns
List of lists of subentity dimensions and numbers

get_point(*reference_coords: symfem.geometry.PointType*) → Tuple[sympy.core.expr.Expr, Ellipsis]

Get a point in the reference from reference coordinates.

Parameters
reference_coords – The reference coordinates

Returns
A point in the cell

abstract integration_limits(*vars: symfem.symbols.AxisVariablesNotSingle = t*) → IntLimits

Get the limits for an integral over this reference.

Parameters
vars – The variables to use for each direction

Returns
Integration limits that can be passed into sympy.integrate

abstract get_map_to(*vertices: symfem.geometry.SetOfPointsInput*) → symfem.geometry.PointType

Get the map from the reference to a cell.

Parameters
vertices – The vertices of a call

Returns
The map

abstract get_inverse_map_to(*vertices_in: symfem.geometry.SetOfPointsInput*) → symfem.geometry.PointType

Get the inverse map from a cell to the reference.

Parameters
vertices_in – The vertices of a cell

Returns
The inverse map

get_map_to_self() → symfem.geometry.PointType

Get the map from the canonical reference to this reference.

Returns
The map

abstract _compute_map_to_self() → symfem.geometry.PointType

Compute the map from the canonical reference to this reference.

Returns
The map

get_inverse_map_to_self() → symfem.geometry.PointType

Get the inverse map from the canonical reference to this reference.

Returns
The map

abstract _compute_inverse_map_to_self() → symfem.geometry.PointType

Compute the inverse map from the canonical reference to this reference.

Returns
The map

abstract volume() → sympy.core.expr.Expr

Calculate the volume.

Returns
The volume of the cell

midpoint() → symfem.geometry.PointType

Calculate the midpoint.

Returns
The midpoint of the cell

jacobian() → sympy.core.expr.Expr

Calculate the Jacobian.

Returns
The Jacobian

scaled_axes() → symfem.geometry.SetOfPoints

Return the unit axes of the reference.

Returns
The axes

tangent() → symfem.geometry.PointType

Calculate the tangent to the element.

Returns
The tangent

normal() → symfem.geometry.PointType

Calculate the normal to the element.

Returns
The normal

sub_entities(*dim: int | None = None*, *codim: int | None = None*) → Tuple[Tuple[int, Ellipsis], Ellipsis]

Get the sub-entities of a given dimension.

Parameters

- **dim** – The dimension of the sub-entity
- **codim** – The co-dimension of the sub-entity

Returns
A tuple of tuples of vertex numbers

sub_entity_count(*dim: int | None = None*, *codim: int | None = None*) → int

Get the number of sub-entities of a given dimension.

Parameters

- **dim** – the dimension of the sub-entity
- **codim** – the codimension of the sub-entity

Returns
The number of sub-entities

sub_entity(*dim: int*, *n: int*, *reference_vertices: bool = False*) → Any

Get the sub-entity of a given dimension and number.

Parameters

- **dim** – the dimension of the sub-entity
- **n** – The sub-entity number

- **reference_vertices** – Should the reference vertices be used?

Returns
The sub-entity

at_vertex(*point: symfem.geometry.PointType*) → bool
Check if a point is a vertex of the reference.

Parameters
point – The point

Returns
Is the point a vertex?

on_edge(*point_in: symfem.geometry.PointType*) → bool
Check if a point is on an edge of the reference.

Parameters
point_in – The point

Returns
Is the point on an edge?

on_face(*point_in: symfem.geometry.PointType*) → bool
Check if a point is on a face of the reference.

Parameters
point_in – The point

Returns
Is the point on a face?

abstract contains(*point: symfem.geometry.PointType*) → bool
Check if a point is contained in the reference.

Parameters
point – The point

Returns
Is the point contained in the reference?

map_polyset_from_default(*poly: List[symfem.functions.FunctionInput]*) → List[symfem.functions.FunctionInput]
Map the polynomials from the default reference element to this reference.

plot_entity_diagrams(*filename: str | List[str]*, *plot_options: Dict[str, Any] = {}*, ***kwargs: Any*)
Plot diagrams showing the entity numbering of the reference.

class symfem.references.**Point**(*vertices: symfem.geometry.SetOfPointsInput = ((),)*)
Bases: *Reference*

A point.

default_reference() → *Reference*
Get the default reference for this cell type.

Returns
The default reference

abstract make_lattice(*n: int*) → symfem.geometry.SetOfPoints
Make a lattice of points.

Parameters
n – The number of points along each edge

Returns
A lattice of points offset from the edge of the cell

abstract make_lattice_with_lines(*n: int*) → LatticeWithLines

Make a lattice of points, and a list of lines connecting them.

Parameters
n – The number of points along each edge

Returns
A lattice of points including the edges of the cell Pairs of point numbers that make a mesh of lines across the cell

integration_limits(*vars: symfem.symbols.AxisVariablesNotSingle = t*) → IntLimits

Get the limits for an integral over this reference.

Parameters
vars – The variables to use for each direction

Returns
Integration limits that can be passed into sympy.integrate

get_map_to(*vertices: symfem.geometry.SetOfPointsInput*) → symfem.geometry.PointType

Get the map from the reference to a cell.

Parameters
vertices – The vertices of a call

Returns
The map

get_inverse_map_to(*vertices_in: symfem.geometry.SetOfPointsInput*) → symfem.geometry.PointType

Get the inverse map from a cell to the reference.

Parameters
vertices_in – The vertices of a cell

Returns
The inverse map

_compute_map_to_self() → symfem.geometry.PointType

Compute the map from the canonical reference to this reference.

Returns
The map

_compute_inverse_map_to_self() → symfem.geometry.PointType

Compute the inverse map from the canonical reference to this reference.

Returns
The map

volume() → sympy.core.expr.Expr

Calculate the volume.

Returns
The volume of the cell

contains(*point: symfem.geometry.PointType*) → bool

Check if a point is contained in the reference.

Parameters
point – The point

Returns
Is the point contained in the reference?

class symfem.references.**Interval**(*vertices: symfem.geometry.SetOfPointsInput = ((0,), (1,))*)

Bases: *Reference*

An interval.

default_reference() → *Reference*

Get the default reference for this cell type.

Returns
The default reference

make_lattice(*n: int*) → symfem.geometry.SetOfPoints

Make a lattice of points.

Parameters
n – The number of points along each edge

Returns
A lattice of points offset from the edge of the cell

make_lattice_with_lines(*n: int*) → LatticeWithLines

Make a lattice of points, and a list of lines connecting them.

Parameters
n – The number of points along each edge

Returns
A lattice of points including the edges of the cell Pairs of point numbers that make a mesh of lines across the cell

integration_limits(*vars: symfem.symbols.AxisVariablesNotSingle = t*) → IntLimits

Get the limits for an integral over this reference.

Parameters
vars – The variables to use for each direction

Returns
Integration limits that can be passed into sympy.integrate

get_map_to(*vertices: symfem.geometry.SetOfPointsInput*) → symfem.geometry.PointType

Get the map from the reference to a cell.

Parameters
vertices – The vertices of a call

Returns
The map

get_inverse_map_to(*vertices_in: symfem.geometry.SetOfPointsInput*) → symfem.geometry.PointType

Get the inverse map from a cell to the reference.

Parameters
vertices_in – The vertices of a cell

Returns
The inverse map

_compute_map_to_self() → symfem.geometry.PointType

Compute the map from the canonical reference to this reference.

Returns
The map

_compute_inverse_map_to_self() → symfem.geometry.PointType

Compute the inverse map from the canonical reference to this reference.

Returns
The map

volume() → sympy.core.expr.Expr

Calculate the volume.

Returns
The volume of the cell

contains(*point: symfem.geometry.PointType*) → bool

Check if a point is contained in the reference.

Parameters
point – The point

Returns
Is the point contained in the reference?

class symfem.references.**Triangle**(*vertices: symfem.geometry.SetOfPointsInput = ((0, 0), (1, 0), (0, 1))*)

Bases: *Reference*

A triangle.

default_reference() → *Reference*

Get the default reference for this cell type.

Returns
The default reference

make_lattice(*n: int*) → symfem.geometry.SetOfPoints

Make a lattice of points.

Parameters
n – The number of points along each edge

Returns
A lattice of points offset from the edge of the cell

make_lattice_with_lines(*n: int*) → LatticeWithLines

Make a lattice of points, and a list of lines connecting them.

Parameters
n – The number of points along each edge

Returns
A lattice of points including the edges of the cell Pairs of point numbers that make a mesh of lines across the cell

z_ordered_entities_extra_dim() → List[List[Tuple[int, int]]]

Get the subentities in back-to-front plotting order when using an extra dimension.

Returns
List of lists of subentity dimensions and numbers

integration_limits(*vars: symfem.symbols.AxisVariablesNotSingle = t*) → IntLimits

Get the limits for an integral over this reference.

Parameters
vars – The variables to use for each direction

Returns
Integration limits that can be passed into sympy.integrate

get_map_to(*vertices: symfem.geometry.SetOfPointsInput*) → symfem.geometry.PointType

Get the map from the reference to a cell.

Parameters
vertices – The vertices of a call

Returns
The map

get_inverse_map_to(*vertices_in: symfem.geometry.SetOfPointsInput*) → symfem.geometry.PointType

Get the inverse map from a cell to the reference.

Parameters
vertices_in – The vertices of a cell

Returns
The inverse map

_compute_map_to_self() → symfem.geometry.PointType

Compute the map from the canonical reference to this reference.

Returns
The map

_compute_inverse_map_to_self() → symfem.geometry.PointType

Compute the inverse map from the canonical reference to this reference.

Returns
The map

volume() → sympy.core.expr.Expr

Calculate the volume.

Returns
The volume of the cell

contains(*point: symfem.geometry.PointType*) → bool

Check if a point is contained in the reference.

Parameters
point – The point

Returns
Is the point contained in the reference?

class symfem.references.**Tetrahedron**(*vertices: symfem.geometry.SetOfPointsInput = ((0, 0, 0), (1, 0, 0), (0, 1, 0), (0, 0, 1))*)

Bases: *Reference*

A tetrahedron.

property clockwise_vertices: symfem.geometry.SetOfPoints

Get list of vertices in clockwise order.

Returns
A list of vertices

z_ordered_entities() → List[List[Tuple[int, int]]]

Get the subentities of the cell in back-to-front plotting order.

Returns
List of lists of subentity dimensions and numbers

default_reference() → *Reference*

Get the default reference for this cell type.

Returns
The default reference

make_lattice(*n: int*) → symfem.geometry.SetOfPoints

Make a lattice of points.

Parameters
n – The number of points along each edge

Returns
A lattice of points offset from the edge of the cell

abstract make_lattice_with_lines(*n: int*) → LatticeWithLines

Make a lattice of points, and a list of lines connecting them.

Parameters
n – The number of points along each edge

Returns
A lattice of points including the edges of the cell Pairs of point numbers that make a mesh of lines across the cell

integration_limits(*vars: symfem.symbols.AxisVariablesNotSingle = t*) → IntLimits

Get the limits for an integral over this reference.

Parameters
vars – The variables to use for each direction

Returns
Integration limits that can be passed into sympy.integrate

get_map_to(*vertices: symfem.geometry.SetOfPointsInput*) → symfem.geometry.PointType

Get the map from the reference to a cell.

Parameters
vertices – The vertices of a call

Returns
The map

get_inverse_map_to(*vertices_in: symfem.geometry.SetOfPointsInput*) → symfem.geometry.PointType

Get the inverse map from a cell to the reference.

Parameters
vertices_in – The vertices of a cell

Returns
The inverse map

_compute_map_to_self() → symfem.geometry.PointType

Compute the map from the canonical reference to this reference.

Returns
The map

_compute_inverse_map_to_self() → symfem.geometry.PointType

Compute the inverse map from the canonical reference to this reference.

Returns
The map

volume() → sympy.core.expr.Expr

Calculate the volume.

Returns
The volume of the cell

contains(*point: symfem.geometry.PointType*) → bool

Check if a point is contained in the reference.

Parameters
point – The point

Returns
Is the point contained in the reference?

class symfem.references.**Quadrilateral**(*vertices: symfem.geometry.SetOfPointsInput = ((0, 0), (1, 0), (0, 1), (1, 1))*)

Bases: *Reference*

A quadrilateral.

property clockwise_vertices: symfem.geometry.SetOfPoints

Get list of vertices in clockwise order.

Returns
A list of vertices

default_reference() → *Reference*

Get the default reference for this cell type.

Returns
The default reference

make_lattice(*n: int*) → symfem.geometry.SetOfPoints

Make a lattice of points.

Parameters
n – The number of points along each edge

Returns
A lattice of points offset from the edge of the cell

make_lattice_with_lines(*n: int*) → LatticeWithLines

Make a lattice of points, and a list of lines connecting them.

Parameters
n – The number of points along each edge

Returns
A lattice of points including the edges of the cell Pairs of point numbers that make a mesh of lines across the cell

z_ordered_entities_extra_dim() → List[List[Tuple[int, int]]]

Get the subentities in back-to-front plotting order when using an extra dimension.

Returns
List of lists of subentity dimensions and numbers

integration_limits(*vars: symfem.symbols.AxisVariablesNotSingle = t*) → IntLimits

Get the limits for an integral over this reference.

Parameters
vars – The variables to use for each direction

Returns
Integration limits that can be passed into sympy.integrate

get_map_to(*vertices: symfem.geometry.SetOfPointsInput*) → symfem.geometry.PointType

Get the map from the reference to a cell.

Parameters
vertices – The vertices of a call

Returns
The map

`get_inverse_map_to`(*vertices_in: symfem.geometry.SetOfPointsInput*) → symfem.geometry.PointType

Get the inverse map from a cell to the reference.

Parameters
vertices_in – The vertices of a cell

Returns
The inverse map

`_compute_map_to_self`() → symfem.geometry.PointType

Compute the map from the canonical reference to this reference.

Returns
The map

`_compute_inverse_map_to_self`() → symfem.geometry.PointType

Compute the inverse map from the canonical reference to this reference.

Returns
The map

`volume`() → sympy.core.expr.Expr

Calculate the volume.

Returns
The volume of the cell

`contains`(*point: symfem.geometry.PointType*) → bool

Check if a point is contained in the reference.

Parameters
point – The point

Returns
Is the point contained in the reference?

class symfem.references.**Hexahedron**(*vertices: symfem.geometry.SetOfPointsInput = ((0, 0, 0), (1, 0, 0), (0, 1, 0), (1, 1, 0), (0, 0, 1), (1, 0, 1), (0, 1, 1), (1, 1, 1))*)

Bases: *Reference*

A hexahedron.

property **clockwise_vertices: symfem.geometry.SetOfPoints**

Get list of vertices in clockwise order.

Returns
A list of vertices

`z_ordered_entities`() → List[List[Tuple[int, int]]]

Get the subentities of the cell in back-to-front plotting order.

Returns
List of lists of subentity dimensions and numbers

`default_reference`() → *Reference*

Get the default reference for this cell type.

Returns
The default reference

`make_lattice`(*n: int*) → symfem.geometry.SetOfPoints

Make a lattice of points.

Parameters
n – The number of points along each edge

Returns
A lattice of points offset from the edge of the cell

abstract make_lattice_with_lines(*n: int*) → LatticeWithLines

Make a lattice of points, and a list of lines connecting them.

Parameters
n – The number of points along each edge

Returns
A lattice of points including the edges of the cell Pairs of point numbers that make a mesh of lines across the cell

integration_limits(*vars: symfem.symbols.AxisVariablesNotSingle = t*) → IntLimits

Get the limits for an integral over this reference.

Parameters
vars – The variables to use for each direction

Returns
Integration limits that can be passed into sympy.integrate

get_map_to(*vertices: symfem.geometry.SetOfPointsInput*) → symfem.geometry.PointType

Get the map from the reference to a cell.

Parameters
vertices – The vertices of a call

Returns
The map

get_inverse_map_to(*vertices_in: symfem.geometry.SetOfPointsInput*) → symfem.geometry.PointType

Get the inverse map from a cell to the reference.

Parameters
vertices_in – The vertices of a cell

Returns
The inverse map

_compute_map_to_self() → symfem.geometry.PointType

Compute the map from the canonical reference to this reference.

Returns
The map

_compute_inverse_map_to_self() → symfem.geometry.PointType

Compute the inverse map from the canonical reference to this reference.

Returns
The map

volume() → sympy.core.expr.Expr

Calculate the volume.

Returns
The volume of the cell

contains(*point: symfem.geometry.PointType*) → bool

Check if a point is contained in the reference.

Parameters
point – The point

Returns
Is the point contained in the reference?

class symfem.references.**Prism**(*vertices: symfem.geometry.SetOfPointsInput = ((0, 0, 0), (1, 0, 0), (0, 1, 0), (0, 0, 1), (1, 0, 1), (0, 1, 1))*)

Bases: *Reference*

A (triangular) prism.

property clockwise_vertices: symfem.geometry.SetOfPoints
Get list of vertices in clockwise order.

Returns
A list of vertices

z_ordered_entities() → List[List[Tuple[int, int]]]
Get the subentities of the cell in back-to-front plotting order.

Returns
List of lists of subentity dimensions and numbers

default_reference() → *Reference*
Get the default reference for this cell type.

Returns
The default reference

make_lattice(*n: int*) → symfem.geometry.SetOfPoints
Make a lattice of points.

Parameters
n – The number of points along each edge

Returns
A lattice of points offset from the edge of the cell

abstract make_lattice_with_lines(*n: int*) → LatticeWithLines
Make a lattice of points, and a list of lines connecting them.

Parameters
n – The number of points along each edge

Returns
A lattice of points including the edges of the cell Pairs of point numbers that make a mesh of lines across the cell

integration_limits(*vars: symfem.symbols.AxisVariablesNotSingle = t*) → IntLimits
Get the limits for an integral over this reference.

Parameters
vars – The variables to use for each direction

Returns
Integration limits that can be passed into sympy.integrate

get_map_to(*vertices: symfem.geometry.SetOfPointsInput*) → symfem.geometry.PointType
Get the map from the reference to a cell.

Parameters
vertices – The vertices of a call

Returns
The map

get_inverse_map_to(*vertices_in: symfem.geometry.SetOfPointsInput*) → symfem.geometry.PointType

Get the inverse map from a cell to the reference.

Parameters
vertices_in – The vertices of a cell

Returns
The inverse map

_compute_map_to_self() → symfem.geometry.PointType

Compute the map from the canonical reference to this reference.

Returns
The map

_compute_inverse_map_to_self() → symfem.geometry.PointType

Compute the inverse map from the canonical reference to this reference.

Returns
The map

volume() → sympy.core.expr.Expr

Calculate the volume.

Returns
The volume of the cell

contains(*point: symfem.geometry.PointType*) → bool

Check if a point is contained in the reference.

Parameters
point – The point

Returns
Is the point contained in the reference?

class symfem.references.**Pyramid**(*vertices: symfem.geometry.SetOfPointsInput = ((0, 0, 0), (1, 0, 0), (0, 1, 0), (1, 1, 0), (0, 0, 1))*)

Bases: *Reference*

A (square-based) pyramid.

property clockwise_vertices: symfem.geometry.SetOfPoints

Get list of vertices in clockwise order.

Returns
A list of vertices

z_ordered_entities() → List[List[Tuple[int, int]]]

Get the subentities of the cell in back-to-front plotting order.

Returns
List of lists of subentity dimensions and numbers

default_reference() → *Reference*

Get the default reference for this cell type.

Returns
The default reference

abstract make_lattice(*n: int*) → symfem.geometry.SetOfPoints

Make a lattice of points.

Parameters
n – The number of points along each edge

Returns
A lattice of points offset from the edge of the cell

abstract make_lattice_with_lines(*n: int*) → LatticeWithLines

Make a lattice of points, and a list of lines connecting them.

Parameters
n – The number of points along each edge

Returns
A lattice of points including the edges of the cell Pairs of point numbers that make a mesh of lines across the cell

integration_limits(*vars: symfem.symbols.AxisVariablesNotSingle = t*) → IntLimits

Get the limits for an integral over this reference.

Parameters
vars – The variables to use for each direction

Returns
Integration limits that can be passed into sympy.integrate

get_map_to(*vertices: symfem.geometry.SetOfPointsInput*) → symfem.geometry.PointType

Get the map from the reference to a cell.

Parameters
vertices – The vertices of a call

Returns
The map

get_inverse_map_to(*vertices_in: symfem.geometry.SetOfPointsInput*) → symfem.geometry.PointType

Get the inverse map from a cell to the reference.

Parameters
vertices_in – The vertices of a cell

Returns
The inverse map

_compute_map_to_self() → symfem.geometry.PointType

Compute the map from the canonical reference to this reference.

Returns
The map

_compute_inverse_map_to_self() → symfem.geometry.PointType

Compute the inverse map from the canonical reference to this reference.

Returns
The map

volume() → sympy.core.expr.Expr

Calculate the volume.

Returns
The volume of the cell

contains(*point: symfem.geometry.PointType*) → bool

Check if a point is contained in the reference.

Parameters
point – The point

Returns
Is the point contained in the reference?

class symfem.references.**DualPolygon**(*number_of_triangles: int*, *vertices: symfem.geometry.SetOfPointsInput | None = None*)

Bases: *Reference*

A polygon on a barycentric dual grid.

reference_origin: symfem.geometry.PointType

abstract contains(*point: symfem.geometry.PointType*) → bool

Check if a point is contained in the reference.

Parameters
point – The point

Returns
Is the point contained in the reference?

abstract integration_limits(*vars: symfem.symbols.AxisVariablesNotSingle = t*) → IntLimits

Get the limits for an integral over this reference.

Parameters
vars – The variables to use for each direction

Returns
Integration limits that can be passed into sympy.integrate

abstract get_map_to(*vertices: symfem.geometry.SetOfPointsInput*) → symfem.geometry.PointType

Get the map from the reference to a cell.

Parameters
vertices – The vertices of a call

Returns
The map

abstract get_inverse_map_to(*vertices_in: symfem.geometry.SetOfPointsInput*) → symfem.geometry.PointType

Get the inverse map from a cell to the reference.

Parameters
vertices_in – The vertices of a cell

Returns
The inverse map

abstract _compute_map_to_self() → symfem.geometry.PointType

Compute the map from the canonical reference to this reference.

Returns
The map

abstract _compute_inverse_map_to_self() → symfem.geometry.PointType

Compute the inverse map from the canonical reference to this reference.

Returns
The map

abstract volume() → sympy.core.expr.Expr

Calculate the volume.

Returns
The volume of the cell

abstract default_reference() → *Reference*

Get the default reference for this cell type.

Returns
The default reference

make_lattice(*n: int*) → symfem.geometry.SetOfPoints

Make a lattice of points.

Parameters
n – The number of points along each edge

Returns
A lattice of points offset from the edge of the cell

abstract make_lattice_with_lines(*n: int*) → LatticeWithLines

Make a lattice of points, and a list of lines connecting them.

Parameters
n – The number of points along each edge

Returns
A lattice of points including the edges of the cell Pairs of point numbers that make a mesh of lines across the cell

z_ordered_entities_extra_dim() → List[List[Tuple[int, int]]]

Get the subentities in back-to-front plotting order when using an extra dimension.

Returns
List of lists of subentity dimensions and numbers

symfem.symbols

Symbols.

Module Contents

symfem.symbols.**x** = ()

symfem.symbols.**t** = ()

symfem.symbols.**AxisVariablesNotSingle**

symfem.symbols.**AxisVariables**

symfem.utils

Utility functions.

Module Contents

Functions

allequal(→ bool)	Test if two items that may be nested lists/tuples are equal.

symfem.utils.**allequal**(*a: Any*, *b: Any*) → bool

Test if two items that may be nested lists/tuples are equal.

Parameters

- **a** – The first item
- **b** – The second item

Returns

a == b?

symfem.version

Version number.

Module Contents

symfem.version.**version = '2024.1.1'**

Package Contents

Functions

add_element(element_class)	Add an element to Symfem.
create_element(→ sym-fem.finite_element.FiniteElement)	Make a finite element.
create_reference(→ sym-fem.references.Reference)	Make a reference cell.

Attributes

__version__
__citation__
__github__
__git__

symfem.**add_element**(*element_class: Type*)

Add an element to Symfem.

Parameters

element_class – The class defining the element.

symfem.**create_element**(*cell_type: str*, *element_type: str*, *order: int*, *vertices: symfem.geometry.SetOfPointsInput | None = None*, ***kwargs: Any*) → *symfem.finite_element.FiniteElement*

Make a finite element.

Parameters

- **cell_type** – The reference cell type. Supported values: point, interval, triangle, quadrilateral, tetrahedron, hexahedron, prism, pyramid, dual polygon(number_of_triangles)
- **element_type** – The type of the element. Supported values: Lagrange, P, vector Lagrange, vP, matrix Lagrange, symmetric matrix Lagrange, dPc, vector dPc, Crouzeix-Raviart, CR, Crouzeix-Falk, CF, conforming Crouzeix-Raviart, conforming CR, serendipity, S, serendipity Hcurl, Scurl, BDMCE, AAE, serendipity Hdiv, Sdiv, BDMCF, AAF, direct serendipity, Regge, Nedelec, Nedelec1, N1curl, Ncurl, Nedelec2, N2curl, Raviart-Thomas, RT, N1div, Brezzi-Douglas-Marini, BDM, N2div, Q, vector Q, vQ, NCE, RTCE, Qcurl, NCF, RTCF, Qdiv, Morley, Morley-Wang-Xu, MWX, Hermite, Mardal-Tai-Winther, MTW, Argyris, bubble, dual polynomial, dual P, dual, Buffa-Christiansen, BC, rotated Buffa-Christiansen, RBC, Brezzi-Douglas-Fortin-Marini, BDFM, Brezzi-Douglas-Duran-Fortin, BDDF, Hellan-Herrmann-Johnson, HHJ, Arnold-Winther, AW, conforming Arnold-Winther, Bell, Kong-Mulder-Veldhuizen, KMV, Bernstein, Bernstein-Bezier, Hsieh-Clough-Tocher, Clough-Tocher, HCT, CT, reduced Hsieh-Clough-Tocher, rHCT, Taylor, discontinuous Taylor, bubble enriched Lagrange, bubble enriched vector Lagrange, Bogner-Fox-Schmit, BFS, Fortin-Soulie, FS, Bernardi-Raugel, Wu-Xu, transition, Guzman-Neilan, nonconforming Arnold-Winther, nonconforming AW, TScurl, trimmed serendipity Hcurl, TSdiv, trimmed serendipity Hdiv, TNT, tiniest tensor, TNTcurl, tiniest tensor Hcurl, TNTdiv, tiniest tensor Hdiv, Arnold-Boffi-Falk, ABF, Arbogast-Correa, AC, AC full, Arbogast-Correa full, Rannacher-Turek, P1-iso-P2, P2-iso-P1, iso-P2 P1, Huang-Zhang, HZ, enriched Galerkin, EG, enriched vector Galerkin, locking-free enriched Galerkin, LFEG, P1 macro, Alfeld-Sorokina, AS
- **order** – The order of the element.
- **vertices** – The vertices of the reference.

symfem.**create_reference**(*cell_type: str*, *vertices: symfem.geometry.SetOfPointsInput | None = None*) → *symfem.references.Reference*

Make a reference cell.

Parameters

- **cell_type** – The reference cell type. Supported values: point, interval, triangle, quadrilateral, tetrahedron, hexahedron, prism, pyramid, dual polygon(number_of_triangles)
- **vertices** – The vertices of the reference.

symfem.**__version__** = '2024.1.1'

symfem.**__citation__** = 'https://doi.org/10.21105/joss.03556 (Scroggs, 2021)'

symfem.**__github__** = 'https://github.com/mscroggs/symfem'

symfem.**__git__** = 'https://github.com/mscroggs/symfem.git'

1.3 References

BIBLIOGRAPHY

[Ciarlet] Ciarlet, P. G., The Finite Element Method for Elliptic Problems (2002, first published 1978) DOI: 10.1137/1.9780898719208

PYTHON MODULE INDEX

S

INDEX

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

V

W

X

Z