
Symfem Documentation

Release 2024.1.1

Matthew Scroggs

Jan 04, 2024

CONTENTS

1 Installing Symfem 3

1.1 Using Symfem 3

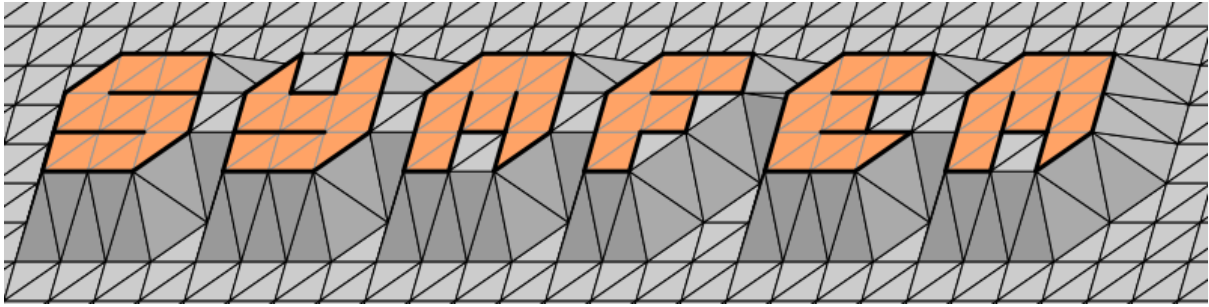
1.2 Documentation index 5

1.3 References 157

Bibliography 159

Python Module Index 161

Index 163



Welcome to the documentation 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.

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), (0.5000000000000000, 0.5000000000000000, 0), (0, 1, 0), (0.5000000000000000, 0.2500000000000000, 0.2500000000000000)]
```

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 vertices 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 documentation¹.

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

<code>ArnoldBoffiFalk</code>	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

<i>AlfeldSorokina</i>	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> (Argyris, Fried, Scharpf, 1968)

Module Contents

Classes

<i>Argyris</i>	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.

These element 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

<code>ArnoldWinther</code>	An Arnold-Winther element.
<code>NonConformingArnoldWinther</code>	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

<i>BDDF</i>	Brezzi-Douglas-Duran-Fortin Hdiv finite element.
-------------	--

Functions

<i>bddf_polyset</i>	(\rightarrow List[symfem.functions.FunctionInp] Create the polynomial basis for a BDDF element.
---------------------	--

`symfem.elements.bddm.bddf_polyset`(*reference*: [symfem.references.Reference](#), *order*: int) \rightarrow
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() \rightarrow 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

<i>BDFM</i>	Brezzi-Douglas-Fortin-Marini Hdiv finite element.
-------------	---

Functions

<i>bdfm_polyset</i>	(\rightarrow List[symfem.functions.FunctionInp] Create the polynomial basis for a BDFM element.
---------------------	--

`symfem.elements.bdfm.bdfm_polyset`(*reference*: `symfem.references.Reference`, *order*: `int`) \rightarrow 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() \rightarrow 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

<i>BDM</i>	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

<i>Bell</i>	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

<i>BernardiRaugel</i>	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

<code>BernsteinFunctional</code>	Functional for a Bernstein element.
<code>Bernstein</code>	Bernstein finite element.

Functions

<code>single_choose</code> (\rightarrow <code>sympy.core.expr.Expr</code>)	Calculate choose function of a set of powers.
<code>choose</code> (\rightarrow <code>sympy.core.expr.Expr</code>)	Calculate choose function of a set of powers.
<code>bernstein_polynomials</code> (\rightarrow <code>List[sympy.core.expr.Expr]</code>)	Return a list of Bernstein polynomials.

`symfem.elements.bernstein.single_choose`(*n*: `int`, *k*: `int`) \rightarrow `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]`) \rightarrow `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*: `sympem.symbols.AxisVariablesNotSingle = x`) \rightarrow `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*: `sympem.references.Reference`, *integral_domain*: `sympem.references.Reference`, *index*: `int`, *degree*: `int`, *entity*: `Tuple[int, int]`)

Bases: `sympem.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

<code>BognerFoxSchmit</code>

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

<code>Bubble</code>	Bubble finite element.
<code>BubbleEnrichedLagrange</code>	Bubble enriched Lagrange element.
<code>BubbleEnrichedVectorLagrange</code>	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:*
sym-
fem.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

<code>CrouzeixRaviart</code>	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

<code>DirectSerendipity</code>	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

<code>DPC</code>	A dPc element.
<code>VectorDPC</code>	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

<code>DualCiarletElement</code>	Abstract barycentric finite element.
<code>Dual</code>	Barycentric dual finite element.
<code>BuffaChristiansen</code>	Buffa-Christiansen barycentric dual finite element.
<code>RotatedBuffaChristiansen</code>	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

<i>EnrichedGalerkin</i>	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

<i>FortinSoulie</i>	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

<i>GuzmanNeilan</i>	Guzman-Neilan Hdiv finite element.
---------------------	------------------------------------

Functions

<code>make_piecewise_lagrange(...)</code>	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

<i>HsiehCloughTocher</i>	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](https://doi.org/10.1016/0045-7825(72)90006-0) (Ciarlet, Raviart, 1972)

Module Contents

Classes

<i>Hermite</i>	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

<i>HellanHerrmannJohnson</i>

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

<i>HuangZhang</i>	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

<i>KongMulderVeldhuizen</i>	Kong-Mulder-Veldhuizen finite element.
-----------------------------	--

Functions

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

<code>Lagrange</code>	Lagrange finite element.
<code>VectorLagrange</code>	Vector Lagrange finite element.
<code>MatrixLagrange</code>	Matrix Lagrange finite element.
<code>SymmetricMatrixLagrange</code>	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

<i>Lagrange</i>	Lagrange finite element.
<i>VectorLagrange</i>	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

<i>Lagrange</i>	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

<i>Morley</i>	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

<i>MorleyWangXu</i>	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

<code>MardalTaiWinther</code>	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

<code>NedelecFirstKind</code>	Nedelec first kind Hcurl finite element.
<code>NedelecSecondKind</code>	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

<i>P1IsoP2Interval</i>	P1-iso-P2 finite element on an interval.
<i>P1IsoP2Tri</i>	P1-iso-P2 finite element on a triangle.
<i>P1IsoP2Quad</i>	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

<code>P1Macro</code>	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

<i>ReducedHsiehCloughTocher</i>	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

<i>RaviartThomas</i>	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

<i>Serendipity</i>	A serendipity element.
<i>SerendipityCurl</i>	A serendipity Hcurl element.
<i>SerendipityDiv</i>	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

<i>Taylor</i>	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

<i>TNT</i>	TiNiest Tensor scalar finite element.
<i>TNTcurl</i>	TiNiest Tensor Hcurl finite element.
<i>TNTdiv</i>	TiNiest Tensor Hdiv finite element.

Functions

<i>p</i> (\rightarrow symfem.functions.ScalarFunction)	Return the kth Legendre polynomial.
<i>b</i> (\rightarrow symfem.functions.ScalarFunction)	Return the function B_k.

```
symfem.elements.tnt.p(k: int, v: sympy.core.symbol.Symbol)  $\rightarrow$  symfem.functions.ScalarFunction
    Return the kth Legendre polynomial.
```

Parameters

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

Returns

The k th 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

<i>Transition</i>	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

<code>TrimmedSerendipityHcurl</code>	Trimmed serendipity Hcurl finite element.
<code>TrimmedSerendipityHdiv</code>	Trimmed serendipity Hdiv finite element.

```
class symfem.elements.trimmed_serendipity.TrimmedSerendipityHcurl(reference: sym-  
fem.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: sym-  
fem.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

<code>Enrichment</code>	An LF enriched Galerkin element.
<code>VectorEnrichedGalerkin</code>	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: sym-
                                                         fem.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

<i>WuXu</i>

Wu-Xu finite element.

Functions

<i>derivatives</i> (\rightarrow List[Tuple[int, Ellipsis]])
--

Return all the orders of a multidimensional derivative.

`symfem.elements.wu_xu.derivatives(dim: int, order: int) \rightarrow 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

<code>l2_dual</code>	$\rightarrow \text{List}[\text{symfem.functions.ScalarFunction}]$	Compute the L2 dual of a set of polynomials.
----------------------	---	--

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

<code>_jrc</code>	$\rightarrow \text{Tuple}[\text{sympy.core.expr.Expr}, \dots]$	Get the Jacobi recurrence relation coefficients.
<code>orthogonal_basis_interval</code>	(\dots)	Create a basis of orthogonal polynomials.
<code>orthogonal_basis_triangle</code>	(\dots)	Create a basis of orthogonal polynomials.
<code>orthogonal_basis_quadrilateral</code>	(\dots)	Create a basis of orthogonal polynomials.
<code>orthogonal_basis_tetrahedron</code>	(\dots)	Create a basis of orthogonal polynomials.
<code>orthogonal_basis_hexahedron</code>	(\dots)	Create a basis of orthogonal polynomials.
<code>orthogonal_basis_prism</code>	(\dots)	Create a basis of orthogonal polynomials.
<code>orthogonal_basis_pyramid</code>	(\dots)	Create a basis of orthogonal polynomials.
<code>orthogonal_basis</code>	(\dots)	Create a basis of orthogonal polynomials.
<code>orthonormal_basis</code>	(\dots)	Create a basis of orthonormal polynomials.

`symfem.polynomials.legendre._jrc`(*a*, *n*) $\rightarrow \text{Tuple}[\text{sympy.core.expr.Expr}, \text{sympy.core.expr.Expr}, \text{sympy.core.expr.Expr}]$

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]]*) \rightarrow *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:  
sym-  
fem.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: sym-  
fem.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: sym-  
fem.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**

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

<code>polynomial_set_1d</code> (→ List[<i>symfem.functions.ScalarFunction</i>])	One dimensional polynomial set.
<code>polynomial_set_vector</code> (...)	Polynomial set.
<code>Hdiv_polynomials</code> (→ List[<i>symfem.functions.VectorFunction</i>])	Hdiv conforming polynomial set.
<code>Hcurl_polynomials</code> (→ List[<i>symfem.functions.VectorFunction</i>])	Hcurl conforming polynomial set.
<code>quolynomial_set_1d</code> (→ List[<i>symfem.functions.ScalarFunction</i>])	One dimensional quolynomial set.
<code>quolynomial_set_vector</code> (...)	Quolynomial set.
<code>Hdiv_quynomials</code> (→ List[<i>symfem.functions.VectorFunction</i>])	Hdiv conforming quolynomial set.
<code>Hcurl_quynomials</code> (→ List[<i>symfem.functions.VectorFunction</i>])	Hcurl conforming quolynomial set.
<code>serendipity_indices</code> (→ List[List[int]])	Get the set indices for a serendipity polynomial set.
<code>serendipity_set_1d</code> (→ List[<i>symfem.functions.ScalarFunction</i>])	One dimensional serendipity set.
<code>serendipity_set_vector</code> (...)	Serendipity set.
<code>Hdiv_serendipity</code> (→ List[<i>symfem.functions.VectorFunction</i>])	Hdiv conforming serendipity set.
<code>Hcurl_serendipity</code> (→ List[<i>symfem.functions.VectorFunction</i>])	Hcurl conforming serendipity set.
<code>prism_polynomial_set_1d</code> (...)	One dimensional polynomial set.
<code>prism_polynomial_set_vector</code> (...)	Polynomial set for a prism.
<code>pyramid_polynomial_set_1d</code> (...)	One dimensional polynomial set.
<code>pyramid_polynomial_set_vector</code> (...)	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_quotionomials(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_quotionomials(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: sym-  
fem.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**

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

<i>BasisFunction</i>	A basis function of a finite element.
<i>SubbedBasisFunction</i>	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 Jacobian

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 lowest 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() \rightarrow `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`) \rightarrow int

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

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

Parameters

cell – The cell

Returns

A version of the function with floats as coefficients

symfem.caching

Functions to cache matrices.

Module Contents

Functions

<code>load_cached_matrix(...)</code>	Load a cached matrix.
<code>save_cached_matrix(matrix_type, cache_id, matrix)</code>	Save a matrix to the cache.
<code>matrix_to_string(\rightarrow str)</code>	Convert a matrix to a string.
<code>matrix_from_string(...)</code>	Convert a string to a matrix.

Attributes

<code>CACHE_DIR</code>
<code>CACHE_FORMAT</code>

`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

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

Attributes

<code>_folder</code>
<code>_elementmap</code>
<code>_elementlist</code>
<code>_fname</code>

`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

<code>FiniteElement</code>	Abstract finite element.
<code>CiarletElement</code>	Finite element defined using the Ciarlet definition.
<code>DirectElement</code>	Finite element defined directly.
<code>EnrichedElement</code>	Finite element defined directly.
<code>ElementBasisFunction</code>	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() \rightarrow *symfem.functions.AnyFunction*

Get the actual basis function.

Returns

The basis function

symfem.functionals

Functionals used to define the dual sets.

Module Contents

Classes

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

Functions

<i>_to_tex</i> (\rightarrow str)	Convert an expression to TeX.
<i>_nth</i> (\rightarrow 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 expression 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: sym-
                                                                fem.functions.FunctionInput,
                                                                direction_in: sym-
                                                                fem.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: sym-
fem.references.Reference,
point_in: sym-
fem.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

<i>AnyFunction</i>	A function.
<i>ScalarFunction</i>	A scalar-valued function.
<i>VectorFunction</i>	A vector-valued function.
<i>MatrixFunction</i>	A matrix-valued function.

Functions

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

Attributes

<i>SingleSympyFormat</i>
<i>SympyFormat</i>
<i>_ValuesToSubstitute</i>
<i>ValuesToSubstitute</i>
<i>FunctionInput</i>

`symfem.functions.SingleSympyFormat`

`symfem.functions.SympyFormat`

`symfem.functions._ValuesToSubstitute`

`symfem.functions._to_sympy_format`(*item*: Any) \rightarrow 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 direction

Returns

The directional derivative

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 lowest 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 determinant

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_()` \rightarrow SympyFormat
Convert to Sympy format.

`__float__()` \rightarrow float
Convert to a float.

`__lt__(other: Any)` \rightarrow bool
Check inequality.

`__le__(other: Any)` \rightarrow bool
Check inequality.

`__gt__(other: Any)` \rightarrow bool
Check inequality.

`__ge__(other: Any)` \rightarrow bool
Check inequality.

`__repr__()` \rightarrow str
Representation.

`__eq__(other: Any)` \rightarrow bool
Check if two functions are equal.

`__ne__(other: Any)` \rightarrow 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)` \rightarrow [`ScalarFunction`](#)
Add.

`__radd__(other: Any)` \rightarrow [`ScalarFunction`](#)
Add.

`__sub__(other: Any)` \rightarrow [`ScalarFunction`](#)
Subtract.

`__rsub__(other: Any)` \rightarrow [`ScalarFunction`](#)
Subtract.

`__truediv__(other: Any)` \rightarrow [`ScalarFunction`](#)
Divide.

`__rtruediv__(other: Any)` \rightarrow [`ScalarFunction`](#)
Divide.

`__mul__(other: Any)` \rightarrow [`ScalarFunction`](#)
Multiply.

`__rmul__(other: Any)` \rightarrow [`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 direction

Returns

The directional derivative

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 lowest 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 derivative

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 lowest 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 column 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 direction

Returns

The directional derivative

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 determinant

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 lowest 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

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

Attributes

<code>PointType</code>
<code>SetOfPoints</code>
<code>PointTypeInput</code>
<code>SetOfPointsInput</code>

`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 format

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
- **triangle** – 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
- **triangle** – 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
- **triangle** – The vertices of the tetrahedron

Returns

Is the point inside the tetrahedron?

`symfem.mappings`

Functions to map functions between cells.

Module Contents

Functions

<code>identity</code> (\rightarrow <code>symfem.functions.AnyFunction</code>)	Map functions.
<code>l2</code> (\rightarrow <code>symfem.functions.AnyFunction</code>)	Map functions, scaling by the determinant of the jacobian.
<code>covariant</code> (\rightarrow <code>symfem.functions.AnyFunction</code>)	Map H(curl) functions.
<code>contravariant</code> (\rightarrow <code>symfem.functions.AnyFunction</code>)	Map H(div) functions.
<code>double_covariant</code> (\rightarrow <code>symfem.functions.MatrixFunction</code>)	Map matrix functions.
<code>double_contravariant</code> (\rightarrow <code>symfem.functions.MatrixFunction</code>)	Map matrix functions.
<code>identity_inverse_transpose</code> (\rightarrow <code>symfem.functions.AnyFunction</code>)	Inverse transpose of <code>identity</code> ().
<code>l2_inverse_transpose</code> (\rightarrow <code>symfem.functions.AnyFunction</code>)	Inverse transpose of <code>l2</code> ().
<code>covariant_inverse_transpose</code> (\rightarrow <code>symfem.functions.AnyFunction</code>)	Inverse transpose of <code>covariant</code> ().
<code>contravariant_inverse_transpose</code> (\rightarrow <code>symfem.functions.AnyFunction</code>)	Inverse transpose of <code>contravariant</code> ().
<code>identity_inverse</code> (\rightarrow <code>symfem.functions.AnyFunction</code>)	Inverse of <code>identity</code> ().
<code>l2_inverse</code> (\rightarrow <code>symfem.functions.AnyFunction</code>)	Inverse of <code>l2</code> ().
<code>covariant_inverse</code> (\rightarrow <code>symfem.functions.AnyFunction</code>)	Inverse of <code>covariant</code> ().
<code>contravariant_inverse</code> (\rightarrow <code>symfem.functions.AnyFunction</code>)	Inverse of <code>contravariant</code> ().
<code>double_covariant_inverse</code> (\rightarrow <code>symfem.functions.AnyFunction</code>)	Inverse of <code>double_covariant</code> ().
<code>double_contravariant_inverse</code> (\rightarrow <code>symfem.functions.AnyFunction</code>)	Inverse of <code>double_contravariant</code> ().
<code>get_mapping</code> (...)	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`) \rightarrow `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`) \rightarrow `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

<code>_extract_moment_data</code>	<code>(→ MomentType)</code>	Get the information for a moment.
<code>make_integral_moment_dofs</code>	<code>(...)</code>	Generate DOFs due to integral moments on sub entities.

Attributes

<code>MomentType</code>
<code>SingleMomentTypeInput</code>
<code>MomentTypeInput</code>

`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 element, 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

<code>PiecewiseFunction</code>	A piecewise function.
--------------------------------	-----------------------

Functions

<code>_piece_reference</code> (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() → sympem.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 direction

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 determinant

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 lowest 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

<i>Colors</i>	Class storing colours used in diagrams.
<i>PictureElement</i>	An element in a picture.
<i>Line</i>	A line.
<i>Bezier</i>	A Bezier curve.
<i>Arrow</i>	An arrow.
<i>NCircle</i>	A circle containing a number.
<i>Fill</i>	A filled polygon.
<i>Math</i>	A mathematical symbol.
<i>Picture</i>	A picture.

Functions

<i>tex_font_size</i> (n)	Convert a font size to a TeX size command.
--------------------------	--

Attributes

<i>PointOrFunction</i>
<i>SetOfPointsOrFunctions</i>
<i>colors</i>

`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*) \rightarrow `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*) \rightarrow `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*) \rightarrow `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

<i>equispaced</i> (\rightarrow Tuple[List[Scalar], List[Scalar]])	Get equispaced points and weights.
<i>lobatto</i> (\rightarrow Tuple[List[Scalar], List[Scalar]])	Get Gauss-Lobatto-Legendre points and weights.
<i>radau</i> (\rightarrow Tuple[List[Scalar], List[Scalar]])	Get Radau points and weights.
<i>legendre</i> (\rightarrow Tuple[List[Scalar], List[Scalar]])	Get Gauss-Legendre points and weights.
<i>get_quadrature</i> (\rightarrow Tuple[List[Scalar], List[Scalar]])	Get quadrature points and weights.

Attributes

<i>Scalar</i>

symfem.quadrature.Scalar

symfem.quadrature.equispaced(*n*: int) \rightarrow 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) \rightarrow 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) \rightarrow 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) \rightarrow 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

<i>Reference</i>	A reference cell on which a finite element can be defined.
<i>Point</i>	A point.
<i>Interval</i>	An interval.
<i>Triangle</i>	A triangle.
<i>Tetrahedron</i>	A tetrahedron.
<i>Quadrilateral</i>	A quadrilateral.
<i>Hexahedron</i>	A hexahedron.
<i>Prism</i>	A (triangular) prism.
<i>Pyramid</i>	A (square-based) pyramid.
<i>DualPolygon</i>	A polygon on a barycentric dual grid.

Functions

<code>_which_side(→ Optional[int])</code>	Check which side of a line or plane a set of points are.
<code>_vsub(→ symfem.geometry.PointType)</code>	Subtract.
<code>_vadd(→ symfem.geometry.PointType)</code>	Add.
<code>_vdot(→ sympy.core.expr.Expr)</code>	Compute the dot product.
<code>_vcross(→ symfem.geometry.PointType)</code>	Compute the cross product.
<code>_vnorm(→ sympy.core.expr.Expr)</code>	Find the norm of a vector.
<code>_vnormalise(→ symfem.geometry.PointType)</code>	Normalise a vector.

Attributes

<i>LatticeWithLines</i>
<i>IntLimits</i>

`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`) \rightarrow `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`) \rightarrow `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`) \rightarrow `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`) \rightarrow `sympy.core.expr.Expr`

Compute the dot product.

Parameters

- **v** – A vector
- **w** – A vector

Returns

The scalar $v \cdot w$

`symfem.references._vcross`(*v_in*: `symfem.geometry.PointTypeInput`, *w_in*: `symfem.geometry.PointTypeInput`) \rightarrow `symfem.geometry.PointType`

Compute the cross product.

Parameters

- **v** – A vector
- **w** – A vector

Returns

The vector $v \times w$

`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 cell

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 cell

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 cell

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() \rightarrow `sympy.core.expr.Expr`

Calculate the volume.

Returns

The volume of the cell

contains(*point*: `symfem.geometry.PointType`) \rightarrow 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() \rightarrow [Reference](#)

Get the default reference for this cell type.

Returns

The default reference

make_lattice(*n*: `int`) \rightarrow `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`) \rightarrow `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() \rightarrow `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`) \rightarrow `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`) \rightarrow `symfem.geometry.PointType`

Get the map from the reference to a cell.

Parameters**vertices** – The vertices of a cell**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 cell

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 cell

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 cell

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 cell

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

<code>allequal</code> (→ 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

<code>add_element(element_class)</code>		Add an element to Symfem.
<code>create_element(→ fem.finite_element.FiniteElement)</code>	sym-	Make a finite element.
<code>create_reference(→ fem.references.Reference)</code>	sym-	Make a reference cell.

Attributes

<code>__version__</code>
<code>__citation__</code>
<code>__github__</code>
<code>__git__</code>

`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](https://doi.org/10.1137/1.9780898719208)

PYTHON MODULE INDEX

S

`symfem`, 10
`symfem.basis_functions`, 68
`symfem.caching`, 73
`symfem.create`, 75
`symfem.elements`, 10
`symfem.elements._guzman_neilan_tetrahedron`, 10
`symfem.elements._guzman_neilan_triangle`, 10
`symfem.elements.abf`, 10
`symfem.elements.ac`, 11
`symfem.elements.alfeld_sorokina`, 11
`symfem.elements.argyris`, 12
`symfem.elements.aw`, 13
`symfem.elements.bddm`, 14
`symfem.elements.bdfm`, 15
`symfem.elements.bdm`, 16
`symfem.elements.bell`, 16
`symfem.elements.bernardi_raugel`, 17
`symfem.elements.bernstein`, 17
`symfem.elements.bfs`, 19
`symfem.elements.bubble`, 20
`symfem.elements.conforming_crouzeix_raviart`, 21
`symfem.elements.crouzeix_raviart`, 22
`symfem.elements.direct_serendipity`, 22
`symfem.elements.dpc`, 23
`symfem.elements.dual`, 24
`symfem.elements.enriched_galerkin`, 26
`symfem.elements.fortin_soulie`, 27
`symfem.elements.guzman_neilan`, 27
`symfem.elements.hct`, 29
`symfem.elements.hermite`, 29
`symfem.elements.hhj`, 30
`symfem.elements.huang_zhang`, 30
`symfem.elements.kmv`, 31
`symfem.elements.lagrange`, 32
`symfem.elements.lagrange_prism`, 34
`symfem.elements.lagrange_pyramid`, 35
`symfem.elements.morley`, 35
`symfem.elements.morley_wang_xu`, 36
`symfem.elements.mtw`, 37
`symfem.elements.nedelec`, 37
`symfem.elements.nedelec_prism`, 38
`symfem.elements.p1_iso_p2`, 39
`symfem.elements.p1_macro`, 40
`symfem.elements.q`, 41
`symfem.elements.rannacher_turek`, 42
`symfem.elements.regge`, 43
`symfem.elements.rhct`, 44
`symfem.elements.rt`, 45
`symfem.elements.serendipity`, 45
`symfem.elements.taylor`, 47
`symfem.elements.tnt`, 47
`symfem.elements.transition`, 49
`symfem.elements.trimmed_serendipity`, 50
`symfem.elements.vector_enriched_galerkin`, 51
`symfem.elements.wu_xu`, 51
`symfem.finite_element`, 76
`symfem.functionals`, 84
`symfem.functions`, 98
`symfem.geometry`, 113
`symfem.mappings`, 115
`symfem.moments`, 121
`symfem.piecewise_functions`, 122
`symfem.plotting`, 126
`symfem.polynomials`, 52
`symfem.polynomials.dual`, 52
`symfem.polynomials.legendre`, 53
`symfem.polynomials.lobatto`, 56
`symfem.polynomials.polysets`, 57
`symfem.quadrature`, 135
`symfem.references`, 136
`symfem.symbols`, 155
`symfem.utils`, 155
`symfem.version`, 156

Symbols

_ValuesToSubstitute (in module *symfem.functions*), 98
 __add__() (symfem.basis_functions.BasisFunction method), 68
 __add__() (symfem.functions.AnyFunction method), 99
 __add__() (symfem.functions.MatrixFunction method), 110
 __add__() (symfem.functions.ScalarFunction method), 103
 __add__() (symfem.functions.VectorFunction method), 106
 __add__() (symfem.piecewise_functions.PiecewiseFunction method), 123
 __build__() (symfem.references.Reference method), 138
 __citation__ (in module *symfem*), 157
 __eq__() (symfem.functions.AnyFunction method), 103
 __eq__() (symfem.piecewise_functions.PiecewiseFunction method), 123
 __eq__() (symfem.references.Reference method), 138
 __float__() (symfem.functions.AnyFunction method), 103
 __ge__() (symfem.functions.AnyFunction method), 103
 __getitem__() (symfem.basis_functions.BasisFunction method), 72
 __getitem__() (symfem.functions.AnyFunction method), 102
 __getitem__() (symfem.functions.MatrixFunction method), 109
 __getitem__() (symfem.functions.VectorFunction method), 106
 __getitem__() (symfem.piecewise_functions.PiecewiseFunction method), 123
 __git__ (in module *symfem*), 157
 __github__ (in module *symfem*), 157
 __gt__() (symfem.functions.AnyFunction method), 103
 __hash__() (symfem.references.Reference method), 139
 __iter__() (symfem.basis_functions.BasisFunction method), 72
 __iter__() (symfem.functions.AnyFunction method), 101
 __iter__() (symfem.functions.VectorFunction method), 109
 __le__() (symfem.functions.AnyFunction method), 103
 __len__() (symfem.basis_functions.BasisFunction method), 72
 __len__() (symfem.functions.AnyFunction method), 102
 __len__() (symfem.functions.VectorFunction method), 106
 __len__() (symfem.piecewise_functions.PiecewiseFunction method), 123
 __lt__() (symfem.functions.AnyFunction method), 103
 __matmul__() (symfem.basis_functions.BasisFunction method), 70
 __matmul__() (symfem.functions.AnyFunction method), 99
 __matmul__() (symfem.functions.MatrixFunction method), 110
 __matmul__() (symfem.functions.ScalarFunction method), 103
 __matmul__() (symfem.functions.VectorFunction method), 107
 __matmul__() (symfem.piecewise_functions.PiecewiseFunction method), 124
 __mul__() (symfem.basis_functions.BasisFunction method), 69
 __mul__() (symfem.functions.AnyFunction method), 99
 __mul__() (symfem.functions.MatrixFunction method), 110
 __mul__() (symfem.functions.ScalarFunction method), 103
 __mul__() (symfem.functions.VectorFunction method), 107
 __mul__() (symfem.piecewise_functions.PiecewiseFunction method), 123
 __ne__() (symfem.functions.AnyFunction method), 103
 __neg__() (symfem.basis_functions.BasisFunction

`method)`, 69
`__neg__()` (*symfem.functions.AnyFunction* *method*), 99
`__neg__()` (*symfem.functions.MatrixFunction* *method*), 110
`__neg__()` (*symfem.functions.ScalarFunction* *method*), 104
`__neg__()` (*symfem.functions.VectorFunction* *method*), 106
`__neg__()` (*symfem.piecewise_functions.PiecewiseFunction* *method*), 124
`__next__()` (*symfem.functions.VectorFunction* *method*), 109
`__pow__()` (*symfem.basis_functions.BasisFunction* *method*), 70
`__pow__()` (*symfem.functions.AnyFunction* *method*), 99
`__pow__()` (*symfem.functions.MatrixFunction* *method*), 110
`__pow__()` (*symfem.functions.ScalarFunction* *method*), 104
`__pow__()` (*symfem.functions.VectorFunction* *method*), 107
`__pow__()` (*symfem.piecewise_functions.PiecewiseFunction* *method*), 124
`__radd__()` (*symfem.basis_functions.BasisFunction* *method*), 68
`__radd__()` (*symfem.functions.AnyFunction* *method*), 99
`__radd__()` (*symfem.functions.MatrixFunction* *method*), 110
`__radd__()` (*symfem.functions.ScalarFunction* *method*), 103
`__radd__()` (*symfem.functions.VectorFunction* *method*), 106
`__radd__()` (*symfem.piecewise_functions.PiecewiseFunction* *method*), 123
`__repr__()` (*symfem.functions.AnyFunction* *method*), 103
`__rmatmul__()` (*symfem.basis_functions.BasisFunction* *method*), 70
`__rmatmul__()` (*symfem.functions.AnyFunction* *method*), 99
`__rmatmul__()` (*symfem.functions.MatrixFunction* *method*), 110
`__rmatmul__()` (*symfem.functions.ScalarFunction* *method*), 103
`__rmatmul__()` (*symfem.functions.VectorFunction* *method*), 107
`__rmatmul__()` (*symfem.piecewise_functions.PiecewiseFunction* *method*), 124
`__rmul__()` (*symfem.basis_functions.BasisFunction* *method*), 69
`__rmul__()` (*symfem.functions.AnyFunction* *method*), 99
`__rmul__()` (*symfem.functions.MatrixFunction* *method*), 110
`__rmul__()` (*symfem.functions.ScalarFunction* *method*), 103
`__rmul__()` (*symfem.functions.VectorFunction* *method*), 107
`__rmul__()` (*symfem.piecewise_functions.PiecewiseFunction* *method*), 123
`__rsub__()` (*symfem.basis_functions.BasisFunction* *method*), 69
`__rsub__()` (*symfem.functions.AnyFunction* *method*), 99
`__rsub__()` (*symfem.functions.MatrixFunction* *method*), 110
`__rsub__()` (*symfem.functions.ScalarFunction* *method*), 103
`__rsub__()` (*symfem.functions.VectorFunction* *method*), 106
`__rsub__()` (*symfem.piecewise_functions.PiecewiseFunction* *method*), 123
`__rtruediv__()` (*symfem.basis_functions.BasisFunction* *method*), 69
`__rtruediv__()` (*symfem.functions.AnyFunction* *method*), 99
`__rtruediv__()` (*symfem.functions.MatrixFunction* *method*), 110
`__rtruediv__()` (*symfem.functions.ScalarFunction* *method*), 103
`__rtruediv__()` (*symfem.functions.VectorFunction* *method*), 107
`__rtruediv__()` (*symfem.piecewise_functions.PiecewiseFunction* *method*), 123
`__sub__()` (*symfem.basis_functions.BasisFunction* *method*), 69
`__sub__()` (*symfem.functions.AnyFunction* *method*), 99
`__sub__()` (*symfem.functions.MatrixFunction* *method*), 110
`__sub__()` (*symfem.functions.ScalarFunction* *method*), 103
`__sub__()` (*symfem.functions.VectorFunction* *method*), 106
`__sub__()` (*symfem.piecewise_functions.PiecewiseFunction* *method*), 123
`__truediv__()` (*symfem.basis_functions.BasisFunction* *method*), 69
`__truediv__()` (*symfem.functions.AnyFunction* *method*), 99
`__truediv__()` (*symfem.functions.MatrixFunction* *method*), 110
`__truediv__()` (*symfem.functions.ScalarFunction* *method*), 103
`__truediv__()` (*symfem.functions.VectorFunction* *method*), 107
`__truediv__()` (*symfem.piecewise_functions.PiecewiseFunction* *method*), 123

- `method`), 123
- `__version__` (in module `symfem`), 157
- `_basis_functions` (symfem.fem.finite_element.DirectElement attribute), 81
- `_basis_functions` (symfem.fem.finite_element.EnrichedElement attribute), 82
- `_check_equal()` (in module `symfem.functions`), 98
- `_compute_inverse_map_to_self()` (symfem.fem.references.DualPolygon method), 154
- `_compute_inverse_map_to_self()` (symfem.fem.references.Hexahedron method), 150
- `_compute_inverse_map_to_self()` (symfem.fem.references.Interval method), 144
- `_compute_inverse_map_to_self()` (symfem.fem.references.Point method), 143
- `_compute_inverse_map_to_self()` (symfem.fem.references.Prism method), 152
- `_compute_inverse_map_to_self()` (symfem.fem.references.Pyramid method), 153
- `_compute_inverse_map_to_self()` (symfem.fem.references.Quadrilateral method), 149
- `_compute_inverse_map_to_self()` (symfem.fem.references.Reference method), 140
- `_compute_inverse_map_to_self()` (symfem.fem.references.Tetrahedron method), 147
- `_compute_inverse_map_to_self()` (symfem.fem.references.Triangle method), 146
- `_compute_map_to_self()` (symfem.fem.references.DualPolygon method), 154
- `_compute_map_to_self()` (symfem.fem.references.Hexahedron method), 150
- `_compute_map_to_self()` (symfem.fem.references.Interval method), 144
- `_compute_map_to_self()` (symfem.fem.references.Point method), 143
- `_compute_map_to_self()` (symfem.fem.references.Prism method), 152
- `_compute_map_to_self()` (symfem.fem.references.Pyramid method), 153
- `_compute_map_to_self()` (symfem.fem.references.Quadrilateral method), 149
- `_compute_map_to_self()` (symfem.fem.references.Reference method), 140
- `_compute_map_to_self()` (symfem.fem.references.Tetrahedron method), 147
- `_compute_map_to_self()` (symfem.fem.references.Triangle method), 146
- `_elementlist` (in module `symfem.create`), 75
- `_elementmap` (in module `symfem.create`), 75
- `_eval_symbolic()` (symfem.fem.elements.bernstein.BernsteinFunctional method), 19
- `_eval_symbolic()` (symfem.fem.functionals.BaseFunctional method), 86
- `_eval_symbolic()` (symfem.fem.functionals.DerivativeIntegralMoment method), 95
- `_eval_symbolic()` (symfem.fem.functionals.DerivativePointEvaluation method), 88
- `_eval_symbolic()` (symfem.fem.functionals.DivergenceIntegralMoment method), 95
- `_eval_symbolic()` (symfem.fem.functionals.DotPointEvaluation method), 91
- `_eval_symbolic()` (symfem.fem.functionals.IntegralAgainst method), 92
- `_eval_symbolic()` (symfem.fem.functionals.IntegralMoment method), 94
- `_eval_symbolic()` (symfem.fem.functionals.IntegralOfDirectionalMultiderivative method), 93
- `_eval_symbolic()` (symfem.fem.functionals.IntegralOfDivergenceAgainst method), 93
- `_eval_symbolic()` (symfem.fem.functionals.PointComponentSecondDerivativeEvaluation method), 90
- `_eval_symbolic()` (symfem.fem.functionals.PointDirectionalDerivativeEvaluation method), 89
- `_eval_symbolic()` (symfem.fem.functionals.PointDivergenceEvaluation method), 91
- `_eval_symbolic()` (symfem.fem.functionals.PointEvaluation method), 87
- `_eval_symbolic()` (symfem.fem.functionals.PointInnerProduct method), 90
- `_eval_symbolic()` (symfem.fem.functionals.WeightedPointEvaluation method), 87
- `_extract_moment_data()` (in module `symfem.moments`), 121
- `_f` (symfem.fem.functions.ScalarFunction attribute), 103
- `_float_basis_functions` (symfem.fem.finite_element.FiniteElement attribute), 77
- `_fname` (in module `symfem.create`), 75
- `_folder` (in module `symfem.create`), 75
- `_get_basis_functions_tensor()` (symfem.fem.finite_element.FiniteElement method), 79
- `_is_close()` (in module `symfem.geometry`), 114
- `_jrc()` (in module `symfem.polynomials.legendre`), 53

- `_make_polyset_tetrahedron()` (symfem.elements.guzman_neilan.GuzmanNeilan method), 28
- `_make_polyset_triangle()` (symfem.elements.guzman_neilan.GuzmanNeilan method), 28
- `_mat` (symfem.functions.MatrixFunction attribute), 109
- `_max_continuity_test_order` (symfem.finite_element.FiniteElement attribute), 77
- `_nth()` (in module symfem.functionals), 85
- `_order_is_allowed()` (in module symfem.create), 76
- `_piece_reference()` (in module symfem.piecewise_functions), 126
- `_pieces` (symfem.piecewise_functions.PiecewiseFunction attribute), 123
- `_sympy_()` (symfem.functions.AnyFunction method), 102
- `_to_sympy_format()` (in module symfem.functions), 98
- `_to_tex()` (in module symfem.functionals), 85
- `_vadd()` (in module symfem.references), 137
- `_value_scale` (symfem.finite_element.FiniteElement attribute), 77
- `_vcross()` (in module symfem.references), 137
- `_vdot()` (in module symfem.geometry), 114
- `_vdot()` (in module symfem.references), 137
- `_vec` (symfem.functions.VectorFunction attribute), 106
- `_vnorm()` (in module symfem.references), 138
- `_vnormalise()` (in module symfem.references), 138
- `_vsub()` (in module symfem.geometry), 114
- `_vsub()` (in module symfem.references), 137
- `_which_side()` (in module symfem.references), 137
- ## A
- `AC` (class in symfem.elements.ac), 11
- `add_arrow()` (symfem.plotting.Picture method), 132
- `add_bezier()` (symfem.plotting.Picture method), 132
- `add_dof_arrow()` (symfem.plotting.Picture method), 133
- `add_dof_marker()` (symfem.plotting.Picture method), 133
- `add_element()` (in module symfem), 156
- `add_element()` (in module symfem.create), 75
- `add_fill()` (symfem.plotting.Picture method), 134
- `add_line()` (symfem.plotting.Picture method), 132
- `add_math()` (symfem.plotting.Picture method), 133
- `add_ncircle()` (symfem.plotting.Picture method), 133
- `adjusted_dof_point()` (symfem.functionals.BaseFunctional method), 86
- `adjusted_dof_point()` (symfem.functionals.DerivativePointEvaluation method), 88
- `adjusted_dof_point()` (symfem.functionals.DotPointEvaluation method), 91
- `adjusted_dof_point()` (symfem.functionals.PointComponentSecondDerivativeEvaluation method), 90
- `adjusted_dof_point()` (symfem.functionals.PointDirectionalDerivativeEvaluation method), 89
- `adjusted_dof_point()` (symfem.functionals.PointDivergenceEvaluation method), 92
- `adjusted_dof_point()` (symfem.functionals.PointEvaluation method), 87
- `adjusted_dof_point()` (symfem.functionals.PointInnerProduct method), 90
- `adjusted_dof_point()` (symfem.functionals.WeightedPointEvaluation method), 87
- `AlfeldSorokina` (class in symfem.elements.alfeld_sorokina), 12
- `allequal()` (in module symfem.utils), 155
- `AnyFunction` (class in symfem.functions), 99
- `Argyris` (class in symfem.elements.argyris), 12
- `ArnoldBoffiFalk` (class in symfem.elements.abf), 10
- `ArnoldWinther` (class in symfem.elements.aw), 13
- `Arrow` (class in symfem.plotting), 130
- `as_png()` (symfem.plotting.Picture method), 134
- `as_svg()` (symfem.plotting.Arrow method), 130
- `as_svg()` (symfem.plotting.Bezier method), 129
- `as_svg()` (symfem.plotting.Fill method), 131
- `as_svg()` (symfem.plotting.Line method), 129
- `as_svg()` (symfem.plotting.Math method), 131
- `as_svg()` (symfem.plotting.NCircle method), 130
- `as_svg()` (symfem.plotting.Picture method), 134
- `as_svg()` (symfem.plotting.PictureElement method), 128
- `as_sympy()` (symfem.basis_functions.BasisFunction method), 70
- `as_sympy()` (symfem.functions.AnyFunction method), 99
- `as_sympy()` (symfem.functions.MatrixFunction method), 110
- `as_sympy()` (symfem.functions.ScalarFunction method), 104
- `as_sympy()` (symfem.functions.VectorFunction method), 107
- `as_sympy()` (symfem.piecewise_functions.PiecewiseFunction method), 123
- `as_tex()` (symfem.basis_functions.BasisFunction method), 70
- `as_tex()` (symfem.functions.AnyFunction method), 99
- `as_tex()` (symfem.functions.MatrixFunction method), 110
- `as_tex()` (symfem.functions.ScalarFunction method), 104
- `as_tex()` (symfem.functions.VectorFunction method), 107
- `as_tex()` (symfem.piecewise_functions.PiecewiseFunction method), 123

- method), 123
- as_tikz() (symfem.plotting.Arrow method), 130
- as_tikz() (symfem.plotting.Bezier method), 129
- as_tikz() (symfem.plotting.Fill method), 131
- as_tikz() (symfem.plotting.Line method), 129
- as_tikz() (symfem.plotting.Math method), 131
- as_tikz() (symfem.plotting.NCircle method), 130
- as_tikz() (symfem.plotting.Picture method), 134
- as_tikz() (symfem.plotting.PictureElement method), 128
- at_vertex() (symfem.references.Reference method), 142
- axes_3d (symfem.plotting.Picture attribute), 132
- AxisVariables (in module symfem.symbols), 155
- AxisVariablesNotSingle (in module symfem.symbols), 155
- ## B
- b() (in module symfem.elements.tnt), 48
- BaseFunctional (class in symfem.functionals), 85
- BasisFunction (class in symfem.basis_functions), 68
- BDDF (class in symfem.elements.bddm), 14
- bddf_polyset() (in module symfem.elements.bddm), 14
- BDFM (class in symfem.elements.bdfm), 15
- bdfm_polyset() (in module symfem.elements.bdfm), 15
- BDM (class in symfem.elements.bdm), 16
- Bell (class in symfem.elements.bell), 16
- BernardiRaugel (class in symfem.elements.bernardi_raugel), 17
- Bernstein (class in symfem.elements.bernstein), 19
- bernstein_polynomials() (in module symfem.elements.bernstein), 18
- BernsteinFunctional (class in symfem.elements.bernstein), 18
- Bezier (class in symfem.plotting), 129
- BLACK (symfem.plotting.Colors attribute), 127
- BLUE (symfem.plotting.Colors attribute), 127
- BognerFoxSchmit (class in symfem.elements.bfs), 19
- Bubble (class in symfem.elements.bubble), 20
- BubbleEnrichedLagrange (class in symfem.elements.bubble), 20
- BubbleEnrichedVectorLagrange (class in symfem.elements.bubble), 21
- BuffaChristiansen (class in symfem.elements.dual), 25
- ## C
- cache (symfem.finite_element.FiniteElement attribute), 77
- CACHE_DIR (in module symfem.caching), 74
- CACHE_FORMAT (in module symfem.caching), 74
- choose() (in module symfem.elements.bernstein), 18
- CiarletElement (class in symfem.finite_element), 79
- clockwise_vertices (symfem.references.Hexahedron property), 149
- clockwise_vertices (symfem.references.Prism property), 151
- clockwise_vertices (symfem.references.Pyramid property), 152
- clockwise_vertices (symfem.references.Quadrilateral property), 148
- clockwise_vertices (symfem.references.Reference property), 138
- clockwise_vertices (symfem.references.Tetrahedron property), 146
- coeffs (in module symfem.elements._guzman_neilan_tetrahedron), 10
- coeffs (in module symfem.elements._guzman_neilan_triangle), 10
- col() (symfem.functions.MatrixFunction method), 110
- Colors (class in symfem.plotting), 127
- colors (in module symfem.plotting), 128
- compute_scale() (symfem.plotting.Picture method), 134
- ConformingCrouzeixRaviart (class in symfem.elements.conforming_crouzeix_raviart), 21
- contains() (symfem.references.DualPolygon method), 154
- contains() (symfem.references.Hexahedron method), 150
- contains() (symfem.references.Interval method), 145
- contains() (symfem.references.Point method), 143
- contains() (symfem.references.Prism method), 152
- contains() (symfem.references.Pyramid method), 153
- contains() (symfem.references.Quadrilateral method), 149
- contains() (symfem.references.Reference method), 142
- contains() (symfem.references.Tetrahedron method), 147
- contains() (symfem.references.Triangle method), 146
- continuity (symfem.elements.abf.ArnoldBoffiFalk attribute), 11
- continuity (symfem.elements.ac.AC attribute), 11
- continuity (symfem.elements.alfeld_sorokina.AlfeldSorokina attribute), 12
- continuity (symfem.elements.argyris.Argyris attribute), 12
- continuity (symfem.elements.aw.ArnoldWinther attribute), 13
- continuity (symfem.elements.aw.NonConformingArnoldWinther attribute), 13
- continuity (symfem.elements.bddm.BDDF attribute), 14
- continuity (symfem.elements.bdfm.BDFM attribute), 15
- continuity (symfem.elements.bdm.BDM attribute), 16

continuity (symfem.elements.bell.Bell attribute), 17
 continuity (symfem.elements.bernardi_raugel.BernardiRaugel attribute), 17
 continuity (symfem.elements.bernstein.Bernstein attribute), 19
 continuity (symfem.elements.bfs.BognerFoxSchmit attribute), 19
 continuity (symfem.elements.bubble.Bubble attribute), 20
 continuity (symfem.elements.bubble.BubbleEnrichedLagrange attribute), 20
 continuity (symfem.elements.bubble.BubbleEnrichedVectorLagrange attribute), 21
 continuity (symfem.elements.conforming_crouzeix_raviart.CrouzeixRaviart attribute), 21
 continuity (symfem.elements.crouzeix_raviart.CrouzeixRaviart attribute), 22
 continuity (symfem.elements.direct_serendipity.DirectSerendipity attribute), 23
 continuity (symfem.elements.dpc.DPC attribute), 23
 continuity (symfem.elements.dpc.VectorDPC attribute), 23
 continuity (symfem.elements.dual.BuffaChristiansen attribute), 26
 continuity (symfem.elements.dual.Dual attribute), 25
 continuity (symfem.elements.dual.RotatedBuffaChristiansen attribute), 26
 continuity (symfem.elements.enriched_galerkin.EnrichedGalerkin attribute), 27
 continuity (symfem.elements.fortin_soulie.FortinSoulie attribute), 27
 continuity (symfem.elements.guzman_neilan.GuzmanNeilan attribute), 28
 continuity (symfem.elements.hct.HsiehCloughTocher attribute), 29
 continuity (symfem.elements.hermite.Hermite attribute), 30
 continuity (symfem.elements.hhj.HellanHerrmannJohnson attribute), 30
 continuity (symfem.elements.huang_zhang.HuangZhang attribute), 31
 continuity (symfem.elements.kmv.KongMulderVeldhuizen attribute), 32
 continuity (symfem.elements.lagrange.Lagrange attribute), 33
 continuity (symfem.elements.lagrange.MatrixLagrange attribute), 33
 continuity (symfem.elements.lagrange.SymmetricMatrixLagrange attribute), 34
 continuity (symfem.elements.lagrange.VectorLagrange attribute), 33
 continuity (symfem.elements.lagrange_prism.Lagrange attribute), 34
 continuity (symfem.elements.lagrange_prism.VectorLagrange attribute), 34
 continuity (symfem.elements.lagrange_pyramid.Lagrange attribute), 35
 continuity (symfem.elements.morley.Morley attribute), 36
 continuity (symfem.elements.morley_wang_xu.MorleyWangXu attribute), 36
 continuity (symfem.elements.mtw.MardalTaiWinther attribute), 37
 continuity (symfem.elements.nedelec.NedelecFirstKind attribute), 38
 continuity (symfem.elements.nedelec.NedelecSecondKind attribute), 38
 continuity (symfem.elements.nedelec_prism.Nedelec attribute), 39
 continuity (symfem.elements.p1_iso_p2.P1IsoP2Interval attribute), 39
 continuity (symfem.elements.p1_iso_p2.P1IsoP2Quad attribute), 40
 continuity (symfem.elements.p1_iso_p2.P1IsoP2Tri attribute), 40
 continuity (symfem.elements.p1_macro.P1Macro attribute), 40
 continuity (symfem.elements.q.Nedelec attribute), 42
 continuity (symfem.elements.q.Q attribute), 41
 continuity (symfem.elements.q.RaviartThomas attribute), 42
 continuity (symfem.elements.q.VectorQ attribute), 41
 continuity (symfem.elements.rannacher_turek.RannacherTurek attribute), 43
 continuity (symfem.elements.regge.Regge attribute), 44
 continuity (symfem.elements.regge.ReggeTP attribute), 44
 continuity (symfem.elements.rhct.ReducedHsiehCloughTocher attribute), 44
 continuity (symfem.elements.rt.RaviartThomas attribute), 45
 continuity (symfem.elements.serendipity.Serendipity attribute), 46
 continuity (symfem.elements.serendipity.SerendipityCurl attribute), 46
 continuity (symfem.elements.serendipity.SerendipityDiv attribute), 46
 continuity (symfem.elements.taylor.Taylor attribute), 47
 continuity (symfem.elements.tnt.TNT attribute), 48
 continuity (symfem.elements.tnt.TNTcurl attribute), 48
 continuity (symfem.elements.tnt.TNTdiv attribute), 49
 continuity (symfem.elements.transition.Transition attribute), 49
 continuity (symfem.elements.trimmed_serendipity.TrimmedSerendipity attribute), 50
 continuity (symfem.elements.trimmed_serendipity.TrimmedSerendipity attribute), 50
 continuity (symfem.elements.vector_enriched_galerkin.Enrichment attribute), 51
 continuity (symfem.elements.vector_enriched_galerkin.VectorEnrichment attribute), 51
 continuity (symfem.elements.wu_xu.WuXu attribute), 51

- tribute), 52
- contravariant() (in module *symfem.mappings*), 117
- contravariant_inverse() (in module *symfem.mappings*), 120
- contravariant_inverse_transpose() (in module *symfem.mappings*), 119
- covariant() (in module *symfem.mappings*), 117
- covariant_inverse() (in module *symfem.mappings*), 119
- covariant_inverse_transpose() (in module *symfem.mappings*), 118
- create_element() (in module *symfem*), 156
- create_element() (in module *symfem.create*), 75
- create_reference() (in module *symfem*), 157
- create_reference() (in module *symfem.create*), 75
- cross() (*symfem.basis_functions.BasisFunction* method), 71
- cross() (*symfem.functions.AnyFunction* method), 100
- cross() (*symfem.functions.MatrixFunction* method), 111
- cross() (*symfem.functions.ScalarFunction* method), 105
- cross() (*symfem.functions.VectorFunction* method), 108
- cross() (*symfem.piecewise_functions.PiecewiseFunction* method), 125
- CrouzeixRaviart (class in *symfem.elements.crouzeix_raviart*), 22
- curl() (*symfem.basis_functions.BasisFunction* method), 71
- curl() (*symfem.functions.AnyFunction* method), 101
- curl() (*symfem.functions.MatrixFunction* method), 112
- curl() (*symfem.functions.ScalarFunction* method), 105
- curl() (*symfem.functions.VectorFunction* method), 108
- curl() (*symfem.piecewise_functions.PiecewiseFunction* method), 125
- ## D
- default_reference() (*symfem.references.DualPolygon* method), 154
- default_reference() (*symfem.references.Hexahedron* method), 149
- default_reference() (*symfem.references.Interval* method), 144
- default_reference() (*symfem.references.Point* method), 142
- default_reference() (*symfem.references.Prism* method), 151
- default_reference() (*symfem.references.Pyramid* method), 152
- default_reference() (*symfem.references.Quadrilateral* method), 148
- default_reference() (*symfem.references.Reference* method), 139
- default_reference() (*symfem.references.Tetrahedron* method), 146
- default_reference() (*symfem.references.Triangle* method), 145
- DerivativeIntegralMoment (class in *symfem.functionals*), 95
- DerivativePointEvaluation (class in *symfem.functionals*), 88
- derivatives() (in module *symfem.elements.wu_xu*), 52
- det() (*symfem.basis_functions.BasisFunction* method), 72
- det() (*symfem.functions.AnyFunction* method), 102
- det() (*symfem.functions.MatrixFunction* method), 112
- det() (*symfem.piecewise_functions.PiecewiseFunction* method), 125
- diff() (*symfem.basis_functions.BasisFunction* method), 70
- diff() (*symfem.functions.AnyFunction* method), 100
- diff() (*symfem.functions.MatrixFunction* method), 111
- diff() (*symfem.functions.ScalarFunction* method), 104
- diff() (*symfem.functions.VectorFunction* method), 107
- diff() (*symfem.piecewise_functions.PiecewiseFunction* method), 124
- DirectElement (class in *symfem.finite_element*), 81
- directional_derivative() (*symfem.basis_functions.BasisFunction* method), 70
- directional_derivative() (*symfem.functions.AnyFunction* method), 100
- directional_derivative() (*symfem.functions.MatrixFunction* method), 111
- directional_derivative() (*symfem.functions.ScalarFunction* method), 104
- directional_derivative() (*symfem.functions.VectorFunction* method), 107
- directional_derivative() (*symfem.piecewise_functions.PiecewiseFunction* method), 124
- DirectSerendipity (class in *symfem.elements.direct_serendipity*), 22
- div() (*symfem.basis_functions.BasisFunction* method), 71
- div() (*symfem.functions.AnyFunction* method), 101
- div() (*symfem.functions.MatrixFunction* method), 111
- div() (*symfem.functions.ScalarFunction* method), 105
- div() (*symfem.functions.VectorFunction* method), 108
- div() (*symfem.piecewise_functions.PiecewiseFunction* method), 125
- DivergenceIntegralMoment (class in *sym-*

- fem.functionals*), 95
- `dof_direction()` (*symfem.functionals.BaseFunctional* method), 86
- `dof_direction()` (*symfem.functionals.DerivativeIntegralMoment* method), 95
- `dof_direction()` (*symfem.functionals.DotPointEvaluation* method), 91
- `dof_direction()` (*symfem.functionals.InnerProductIntegralMoment* method), 97
- `dof_direction()` (*symfem.functionals.IntegralMoment* method), 94
- `dof_direction()` (*symfem.functionals.NormalIntegralMoment* method), 96
- `dof_direction()` (*symfem.functionals.PointDirectionalDerivativeEvaluation* method), 89
- `dof_direction()` (*symfem.functionals.PointDivergenceEvaluation* method), 92
- `dof_direction()` (*symfem.functionals.PointInnerProduct* method), 90
- `dof_direction()` (*symfem.functionals.TangentIntegralMoment* method), 96
- `dof_directions()` (*symfem.elements.dual.DualCiarletElement* method), 25
- `dof_directions()` (*symfem.finite_element.CiarletElement* method), 80
- `dof_directions()` (*symfem.finite_element.DirectElement* method), 82
- `dof_directions()` (*symfem.finite_element.EnrichedElement* method), 83
- `dof_directions()` (*symfem.finite_element.FiniteElement* method), 77
- `dof_entities()` (*symfem.elements.dual.DualCiarletElement* method), 25
- `dof_entities()` (*symfem.finite_element.CiarletElement* method), 80
- `dof_entities()` (*symfem.finite_element.DirectElement* method), 82
- `dof_entities()` (*symfem.finite_element.EnrichedElement* method), 83
- `dof_entities()` (*symfem.finite_element.FiniteElement* method), 77
- `dof_entities()` (*symfem.finite_element.DirectElement* method), 81
- `dof_entities()` (*symfem.finite_element.EnrichedElement* method), 83
- `dof_entities()` (*symfem.finite_element.FiniteElement* method), 77
- `dof_point()` (*symfem.elements.bernstein.BernsteinFunctional* method), 18
- `dof_point()` (*symfem.functionals.BaseFunctional* method), 86
- `dof_point()` (*symfem.functionals.DerivativePointEvaluation* method), 88
- `dof_point()` (*symfem.functionals.DotPointEvaluation* method), 91
- `dof_point()` (*symfem.functionals.IntegralAgainst* method), 92
- `dof_point()` (*symfem.functionals.IntegralMoment* method), 94
- `dof_point()` (*symfem.functionals.IntegralOfDirectionalMultiderivative* method), 93
- `dof_point()` (*symfem.functionals.IntegralOfDivergenceAgainst* method), 93
- `dof_point()` (*symfem.functionals.PointComponentSecondDerivativeEvaluation* method), 90
- `dof_point()` (*symfem.functionals.PointDirectionalDerivativeEvaluation* method), 89
- `dof_point()` (*symfem.functionals.PointDivergenceEvaluation* method), 92
- `dof_point()` (*symfem.functionals.PointEvaluation* method), 87
- `dof_point()` (*symfem.functionals.PointInnerProduct* method), 90
- `dof_point()` (*symfem.functionals.WeightedPointEvaluation* method), 87
- `dot()` (*symfem.basis_functions.BasisFunction* method), 71
- `dot()` (*symfem.functionals.DerivativeIntegralMoment* method), 95
- `dot()` (*symfem.functionals.InnerProductIntegralMoment* method), 97
- `dot()` (*symfem.functionals.IntegralAgainst* method), 92
- `dot()` (*symfem.functionals.IntegralMoment* method), 94
- `dot()` (*symfem.functionals.IntegralOfDivergenceAgainst* method), 93
- `dot()` (*symfem.functions.AnyFunction* method), 100

[dot\(\)](#) (*symfem.functions.MatrixFunction* method), 111
[dot\(\)](#) (*symfem.functions.ScalarFunction* method), 105
[dot\(\)](#) (*symfem.functions.VectorFunction* method), 108
[dot\(\)](#) (*symfem.piecewise_functions.PiecewiseFunction* method), 124
[DotPointEvaluation](#) (class in *symfem.functionals*), 91
[double_contravariant\(\)](#) (in module *symfem.mappings*), 117
[double_contravariant_inverse\(\)](#) (in module *symfem.mappings*), 120
[double_covariant\(\)](#) (in module *symfem.mappings*), 117
[double_covariant_inverse\(\)](#) (in module *symfem.mappings*), 120
[DPC](#) (class in *symfem.elements.dpc*), 23
[Dual](#) (class in *symfem.elements.dual*), 25
[DualCiarletElement](#) (class in *symfem.elements.dual*), 24
[DualPolygon](#) (class in *symfem.references*), 153
E
[ElementBasisFunction](#) (class in *symfem.finite_element*), 83
[EnrichedElement](#) (class in *symfem.finite_element*), 82
[EnrichedGalerkin](#) (class in *symfem.elements.enriched_galerkin*), 26
[Enrichment](#) (class in *symfem.elements.vector_enriched_galerkin*), 51
[entity\(\)](#) (*symfem.plotting.Colors* method), 127
[entity_definition\(\)](#) (*symfem.functionals.BaseFunctional* method), 86
[entity_dim\(\)](#) (*symfem.functionals.BaseFunctional* method), 85
[entity_dofs\(\)](#) (*symfem.elements.dual.DualCiarletElement* method), 24
[entity_dofs\(\)](#) (*symfem.finite_element.CiarletElement* method), 80
[entity_dofs\(\)](#) (*symfem.finite_element.DirectElement* method), 81
[entity_dofs\(\)](#) (*symfem.finite_element.EnrichedElement* method), 82
[entity_dofs\(\)](#) (*symfem.finite_element.FiniteElement* method), 78
[entity_number\(\)](#) (*symfem.functionals.BaseFunctional* method), 85
[entity_tex\(\)](#) (*symfem.functionals.BaseFunctional* method), 85
[equispaced\(\)](#) (in module *symfem.quadrature*), 135
[eval\(\)](#) (*symfem.functionals.BaseFunctional* method), 86

[eval_symbolic\(\)](#) (*symfem.functionals.BaseFunctional* method), 86

F

[f](#) (*symfem.functionals.IntegralAgainst* attribute), 92
[f](#) (*symfem.functionals.IntegralMoment* attribute), 94
[Fill](#) (class in *symfem.plotting*), 131
[FiniteElement](#) (class in *symfem.finite_element*), 77
[FortinSoulie](#) (class in *symfem.elements.fortin_soulie*), 27
[FunctionInput](#) (in module *symfem.functions*), 113

G

[get_basis_function\(\)](#) (*symfem.finite_element.FiniteElement* method), 78
[get_basis_functions\(\)](#) (*symfem.elements.dual.DualCiarletElement* method), 24
[get_basis_functions\(\)](#) (*symfem.finite_element.CiarletElement* method), 80
[get_basis_functions\(\)](#) (*symfem.finite_element.DirectElement* method), 82
[get_basis_functions\(\)](#) (*symfem.finite_element.EnrichedElement* method), 83
[get_basis_functions\(\)](#) (*symfem.finite_element.FiniteElement* method), 78
[get_dual_matrix\(\)](#) (*symfem.elements.dual.DualCiarletElement* method), 24
[get_dual_matrix\(\)](#) (*symfem.finite_element.CiarletElement* method), 80
[get_function\(\)](#) (*symfem.basis_functions.BasisFunction* method), 68
[get_function\(\)](#) (*symfem.basis_functions.SubbedBasisFunction* method), 73
[get_function\(\)](#) (*symfem.finite_element.ElementBasisFunction* method), 83
[get_inverse_map_to\(\)](#) (*symfem.references.DualPolygon* method), 154
[get_inverse_map_to\(\)](#) (*symfem.references.Hexahedron* method), 150
[get_inverse_map_to\(\)](#) (*symfem.references.Interval* method), 144
[get_inverse_map_to\(\)](#) (*symfem.references.Point* method), 143
[get_inverse_map_to\(\)](#) (*symfem.references.Prism* method), 151

`get_inverse_map_to()` (*symfem.references.Pyramid method*), 153

`get_inverse_map_to()` (*symfem.references.Quadrilateral method*), 149

`get_inverse_map_to()` (*symfem.references.Reference method*), 140

`get_inverse_map_to()` (*symfem.references.Tetrahedron method*), 147

`get_inverse_map_to()` (*symfem.references.Triangle method*), 146

`get_inverse_map_to_self()` (*symfem.references.Reference method*), 140

`get_map_to()` (*symfem.references.DualPolygon method*), 154

`get_map_to()` (*symfem.references.Hexahedron method*), 150

`get_map_to()` (*symfem.references.Interval method*), 144

`get_map_to()` (*symfem.references.Point method*), 143

`get_map_to()` (*symfem.references.Prism method*), 151

`get_map_to()` (*symfem.references.Pyramid method*), 153

`get_map_to()` (*symfem.references.Quadrilateral method*), 148

`get_map_to()` (*symfem.references.Reference method*), 140

`get_map_to()` (*symfem.references.Tetrahedron method*), 147

`get_map_to()` (*symfem.references.Triangle method*), 145

`get_map_to_self()` (*symfem.references.Reference method*), 140

`get_mapping()` (*in module symfem.mappings*), 120

`get_piece()` (*symfem.piecewise_functions.PiecewiseFunction method*), 123

`get_point()` (*symfem.references.Reference method*), 140

`get_polynomial_basis()` (*symfem.elements.dual.DualCiarletElement method*), 24

`get_polynomial_basis()` (*symfem.finite_element.CiarletElement method*), 80

`get_polynomial_basis()` (*symfem.finite_element.DirectElement method*), 82

`get_polynomial_basis()` (*symfem.finite_element.EnrichedElement method*), 83

`get_polynomial_basis()` (*symfem.finite_element.FiniteElement method*), 79

`get_quadrature()` (*in module symfem.quadrature*), 135

`get_tensor_factorisation()` (*symfem.elements.q.Q method*), 41

`get_tensor_factorisation()` (*symfem.finite_element.FiniteElement method*), 79

`get_tex()` (*symfem.elements.bernstein.BernsteinFunctional method*), 19

`get_tex()` (*symfem.functionals.BaseFunctional method*), 86

`get_tex()` (*symfem.functionals.DerivativePointEvaluation method*), 88

`get_tex()` (*symfem.functionals.DivergenceIntegralMoment method*), 95

`get_tex()` (*symfem.functionals.DotPointEvaluation method*), 91

`get_tex()` (*symfem.functionals.InnerProductIntegralMoment method*), 97

`get_tex()` (*symfem.functionals.IntegralAgainst method*), 93

`get_tex()` (*symfem.functionals.IntegralMoment method*), 95

`get_tex()` (*symfem.functionals.IntegralOfDirectionalMultiderivative method*), 94

`get_tex()` (*symfem.functionals.IntegralOfDivergenceAgainst method*), 93

`get_tex()` (*symfem.functionals.NormalDerivativeIntegralMoment method*), 96

`get_tex()` (*symfem.functionals.NormalInnerProductIntegralMoment method*), 97

`get_tex()` (*symfem.functionals.NormalIntegralMoment method*), 96

`get_tex()` (*symfem.functionals.PointComponentSecondDerivativeEvaluation method*), 90

`get_tex()` (*symfem.functionals.PointDirectionalDerivativeEvaluation method*), 89

`get_tex()` (*symfem.functionals.PointDivergenceEvaluation method*), 92

`get_tex()` (*symfem.functionals.PointEvaluation method*), 87

`get_tex()` (*symfem.functionals.PointInnerProduct method*), 91

`get_tex()` (*symfem.functionals.PointNormalDerivativeEvaluation method*), 89

`get_tex()` (*symfem.functionals.TangentIntegralMoment method*), 96

`get_tex()` (*symfem.functionals.WeightedPointEvaluation method*), 87

`get_tikz_definitions()` (*symfem.plotting.Colors method*), 128

`get_tikz_name()` (*symfem.plotting.Colors method*), 128

`grad()` (*symfem.basis_functions.BasisFunction method*), 71

`grad()` (*symfem.functions.AnyFunction method*), 101

`grad()` (*symfem.functions.MatrixFunction method*), 112

`grad()` (*symfem.functions.ScalarFunction method*), 105

`grad()` (*symfem.functions.VectorFunction method*), 108

`grad()` (*symfem.piecewise_functions.PiecewiseFunction method*), 101

- method*), 125
- GRAY (*symfem.plotting.Colors* attribute), 127
- GREEN (*symfem.plotting.Colors* attribute), 127
- GuzmanNeilan (class in *sym-fem.elements.guzman_neilan*), 28
- ## H
- Hcurl_polynomials() (in module *sym-fem.polynomials*), 64
- Hcurl_polynomials() (in module *sym-fem.polynomials.polysets*), 58
- Hcurl_quolynomials() (in module *sym-fem.polynomials*), 64
- Hcurl_quolynomials() (in module *sym-fem.polynomials.polysets*), 59
- Hcurl_serendipity() (in module *sym-fem.polynomials*), 64
- Hcurl_serendipity() (in module *sym-fem.polynomials.polysets*), 60
- Hdiv_polynomials() (in module *sym-fem.polynomials*), 64
- Hdiv_polynomials() (in module *sym-fem.polynomials.polysets*), 58
- Hdiv_quolynomials() (in module *sym-fem.polynomials*), 65
- Hdiv_quolynomials() (in module *sym-fem.polynomials.polysets*), 59
- Hdiv_serendipity() (in module *sym-fem.polynomials*), 65
- Hdiv_serendipity() (in module *sym-fem.polynomials.polysets*), 60
- HellanHerrmannJohnson (class in *sym-fem.elements.hhj*), 30
- Hermite (class in *symfem.elements.hermite*), 29
- Hexahedron (class in *symfem.references*), 149
- HsiehCloughTocher (class in *symfem.elements.hct*), 29
- HuangZhang (class in *symfem.elements.huang_zhang*), 31
- ## I
- identity() (in module *symfem.mappings*), 116
- identity_inverse() (in module *symfem.mappings*), 119
- identity_inverse_transpose() (in module *sym-fem.mappings*), 118
- init_kwargs() (*sym-fem.elements.abf.ArnoldBoffiFalk* method), 11
- init_kwargs() (*symfem.elements.ac.AC* method), 11
- init_kwargs() (*symfem.elements.aw.ArnoldWinther* method), 13
- init_kwargs() (*sym-fem.elements.aw.NonConformingArnoldWinther* method), 13
- init_kwargs() (*symfem.elements.bddm.BDDF* method), 14
- init_kwargs() (*symfem.elements.bdfm.BDFM* method), 15
- init_kwargs() (*symfem.elements.bdm.BDM* method), 16
- init_kwargs() (*symfem.elements.bell.Bell* method), 17
- init_kwargs() (*symfem.elements.bubble.Bubble* method), 20
- init_kwargs() (*sym-fem.elements.bubble.BubbleEnrichedLagrange* method), 20
- init_kwargs() (*sym-fem.elements.bubble.BubbleEnrichedVectorLagrange* method), 21
- init_kwargs() (*sym-fem.elements.crouzeix_raviart.CrouzeixRaviart* method), 22
- init_kwargs() (*symfem.elements.dpc.DPC* method), 23
- init_kwargs() (*symfem.elements.dpc.VectorDPC* method), 23
- init_kwargs() (*sym-fem.elements.hhj.HellanHerrmannJohnson* method), 30
- init_kwargs() (*sym-fem.elements.huang_zhang.HuangZhang* method), 31
- init_kwargs() (*symfem.elements.lagrange.Lagrange* method), 33
- init_kwargs() (*sym-fem.elements.lagrange.MatrixLagrange* method), 33
- init_kwargs() (*sym-fem.elements.lagrange.SymmetricMatrixLagrange* method), 34
- init_kwargs() (*sym-fem.elements.lagrange.VectorLagrange* method), 33
- init_kwargs() (*sym-fem.elements.lagrange_prism.Lagrange* method), 34
- init_kwargs() (*sym-fem.elements.lagrange_prism.VectorLagrange* method), 35
- init_kwargs() (*sym-fem.elements.lagrange_pyramid.Lagrange* method), 35
- init_kwargs() (*sym-fem.elements.mtw.MardalTaiWinther* method), 37
- init_kwargs() (*sym-fem.elements.nedelec.NedelecFirstKind* method), 38
- init_kwargs() (*sym-fem.elements.nedelec.NedelecSecondKind* method), 38
- init_kwargs() (*sym-fem.elements.nedelec_prism.Nedelec*

- method*), 39
- `init_kwargs()` (*symfem.elements.q.Nedelec method*), 42
- `init_kwargs()` (*symfem.elements.q.Q method*), 41
- `init_kwargs()` (*symfem.elements.q.RaviartThomas method*), 42
- `init_kwargs()` (*symfem.elements.q.VectorQ method*), 41
- `init_kwargs()` (*symfem.elements.rannacher_turek.RannacherTurek method*), 43
- `init_kwargs()` (*symfem.elements.regge.Regge method*), 44
- `init_kwargs()` (*symfem.elements.regge.ReggeTP method*), 44
- `init_kwargs()` (*symfem.elements.rt.RaviartThomas method*), 45
- `init_kwargs()` (*symfem.elements.serendipity.Serendipity method*), 46
- `init_kwargs()` (*symfem.elements.serendipity.SerendipityCurl method*), 46
- `init_kwargs()` (*symfem.elements.serendipity.SerendipityDiv method*), 46
- `init_kwargs()` (*symfem.elements.tnt.TNT method*), 48
- `init_kwargs()` (*symfem.elements.tnt.TNTcurl method*), 48
- `init_kwargs()` (*symfem.elements.tnt.TNTdiv method*), 49
- `init_kwargs()` (*symfem.elements.transition.Transition method*), 49
- `init_kwargs()` (*symfem.elements.trimmed_serendipity.TrimmedSerendipityHcurl method*), 50
- `init_kwargs()` (*symfem.elements.trimmed_serendipity.TrimmedSerendipityHdiv method*), 50
- `init_kwargs()` (*symfem.finite_element.FiniteElement method*), 79
- `InnerProductIntegralMoment` (*class in symfem.functionals*), 97
- `integral()` (*symfem.basis_functions.BasisFunction method*), 71
- `integral()` (*symfem.functions.AnyFunction method*), 101
- `integral()` (*symfem.functions.MatrixFunction method*), 112
- `integral()` (*symfem.functions.ScalarFunction method*), 105
- `integral()` (*symfem.functions.VectorFunction method*), 108
- `integral()` (*symfem.piecewise_functions.PiecewiseFunction method*), 125
- `IntegralAgainst` (*class in symfem.functionals*), 92
- `IntegralMoment` (*class in symfem.functionals*), 94
- `IntegralOfDirectionalMultiderivative` (*class in symfem.functionals*), 93
- `IntegralOfDivergenceAgainst` (*class in symfem.functionals*), 93
- `integrate()` (*symfem.functions.AnyFunction method*), 101
- `integrate()` (*symfem.functions.ScalarFunction method*), 105
- `integration_limits()` (*symfem.references.DualPolygon method*), 154
- `integration_limits()` (*symfem.references.Hexahedron method*), 150
- `integration_limits()` (*symfem.references.Interval method*), 144
- `integration_limits()` (*symfem.references.Point method*), 143
- `integration_limits()` (*symfem.references.Prism method*), 151
- `integration_limits()` (*symfem.references.Pyramid method*), 153
- `integration_limits()` (*symfem.references.Quadrilateral method*), 148
- `integration_limits()` (*symfem.references.Reference method*), 140
- `integration_limits()` (*symfem.references.Tetrahedron method*), 147
- `integration_limits()` (*symfem.references.Triangle method*), 145
- `intersection()` (*symfem.references.Reference method*), 139
- `Interval` (*class in symfem.references*), 143
- `IntLimits` (*in module symfem.references*), 136
- `Jacobian` (*class in symfem.basis_functions*), 71
- `jacobian()` (*symfem.basis_functions.BasisFunction method*), 71
- `jacobian()` (*symfem.functions.AnyFunction method*), 100
- `jacobian()` (*symfem.functions.MatrixFunction method*), 111
- `jacobian()` (*symfem.functions.ScalarFunction method*), 104
- `jacobian()` (*symfem.functions.VectorFunction method*), 108
- `jacobian()` (*symfem.piecewise_functions.PiecewiseFunction method*), 124
- `jacobian()` (*symfem.references.Reference method*), 141
- `jacobian_component()` (*symfem.basis_functions.BasisFunction method*), 70
- `jacobian_component()` (*symfem.functions.AnyFunction method*), 100
- `jacobian_component()` (*symfem.functions.MatrixFunction method*), 111

- 111
 jacobian_component() (symfem.functions.ScalarFunction method), 104
 jacobian_component() (symfem.functions.VectorFunction method), 107
 jacobian_component() (symfem.piecewise_functions.PiecewiseFunction method), 124
- ## K
- kmv_tet_polyset() (in module symfem.elements.kmv), 32
 kmv_tri_polyset() (in module symfem.elements.kmv), 31
 KongMulderVeldhuizen (class in symfem.elements.kmv), 32
- ## L
- 12() (in module symfem.mappings), 116
 12_dual() (in module symfem.polynomials), 63
 12_dual() (in module symfem.polynomials.dual), 53
 12_inverse() (in module symfem.mappings), 119
 12_inverse_transpose() (in module symfem.mappings), 118
 Lagrange (class in symfem.elements.lagrange), 32
 Lagrange (class in symfem.elements.lagrange_prism), 34
 Lagrange (class in symfem.elements.lagrange_pyramid), 35
 last_updated (symfem.elements.abf.ArnoldBoffiFalk attribute), 11
 last_updated (symfem.elements.ac.AC attribute), 11
 last_updated (symfem.elements.alfeld_sorokina.AlfeldSorokina attribute), 12
 last_updated (symfem.elements.argyris.Argyris attribute), 12
 last_updated (symfem.elements.aw.ArnoldWinther attribute), 13
 last_updated (symfem.elements.aw.NonConformingArnoldWinther attribute), 13
 last_updated (symfem.elements.bddm.BDDF attribute), 14
 last_updated (symfem.elements.bdfm.BDFM attribute), 15
 last_updated (symfem.elements.bdm.BDM attribute), 16
 last_updated (symfem.elements.bell.Bell attribute), 17
 last_updated (symfem.elements.bernardi_raugel.BernardiRaugel attribute), 17
 last_updated (symfem.elements.bernstein.Bernstein attribute), 19
 last_updated (symfem.elements.bfs.BognerFoxSchmit attribute), 20
 last_updated (symfem.elements.bubble.Bubble attribute), 20
 last_updated (symfem.elements.bubble.BubbleEnrichedLagrange attribute), 20
 last_updated (symfem.elements.bubble.BubbleEnrichedVectorLagrange attribute), 21
 last_updated (symfem.elements.conforming_crouzeix_raviart.ConformingCrouzeix attribute), 21
 last_updated (symfem.elements.crouzeix_raviart.CrouzeixRaviart attribute), 22
 last_updated (symfem.elements.direct_serendipity.DirectSerendipity attribute), 23
 last_updated (symfem.elements.dpc.DPC attribute), 23
 last_updated (symfem.elements.dpc.VectorDPC attribute), 23
 last_updated (symfem.elements.dual.BuffaChristiansen attribute), 26
 last_updated (symfem.elements.dual.Dual attribute), 25
 last_updated (symfem.elements.dual.RotatedBuffaChristiansen attribute), 26
 last_updated (symfem.elements.enriched_galerkin.EnrichedGalerkin attribute), 27
 last_updated (symfem.elements.fortin_soulie.FortinSoulie attribute), 27
 last_updated (symfem.elements.guzman_neilan.GuzmanNeilan attribute), 28
 last_updated (symfem.elements.hct.HsiehCloughTocher attribute), 29
 last_updated (symfem.elements.hermite.Hermite attribute), 30
 last_updated (symfem.elements.hhj.HellanHerrmannJohnson attribute), 30
 last_updated (symfem.elements.huang_zhang.HuangZhang attribute), 31
 last_updated (symfem.elements.kmv.KongMulderVeldhuizen attribute), 32
 last_updated (symfem.elements.lagrange.Lagrange attribute), 33
 last_updated (symfem.elements.lagrange.Lagrange attribute), 33

`fem.elements.lagrange.MatrixLagrange` attribute), 33

`last_updated` (sym-
`fem.elements.lagrange.SymmetricMatrixLagrange` attribute), 34

`last_updated` (sym-
`fem.elements.lagrange.VectorLagrange` attribute), 33

`last_updated` (sym-
`fem.elements.lagrange_prism.Lagrange` attribute), 34

`last_updated` (sym-
`fem.elements.lagrange_prism.VectorLagrange` attribute), 34

`last_updated` (sym-
`fem.elements.lagrange_pyramid.Lagrange` attribute), 35

`last_updated` (symfem.elements.morley.Morley attribute), 36

`last_updated` (sym-
`fem.elements.morley_wang_xu.MorleyWangXu` attribute), 36

`last_updated` (sym-
`fem.elements.mtw.MardalTaiWinther` attribute), 37

`last_updated` (sym-
`fem.elements.nedelec.NedelecFirstKind` attribute), 38

`last_updated` (sym-
`fem.elements.nedelec.NedelecSecondKind` attribute), 38

`last_updated` (sym-
`fem.elements.nedelec_prism.Nedelec` attribute), 39

`last_updated` (sym-
`fem.elements.p1_iso_p2.P1IsoP2Interval` attribute), 39

`last_updated` (sym-
`fem.elements.p1_iso_p2.P1IsoP2Quad` attribute), 40

`last_updated` (sym-
`fem.elements.p1_iso_p2.P1IsoP2Tri` attribute), 40

`last_updated` (symfem.elements.p1_macro.P1Macro attribute), 40

`last_updated` (symfem.elements.q.Nedelec attribute), 42

`last_updated` (symfem.elements.q.Q attribute), 41

`last_updated` (symfem.elements.q.RaviartThomas attribute), 42

`last_updated` (symfem.elements.q.VectorQ attribute), 41

`last_updated` (sym-
`fem.elements.rannacher_turek.RannacherTurek` attribute), 43

`last_updated` (symfem.elements.regge.Regge attribute), 44

`last_updated` (symfem.elements.regge.ReggeTP attribute), 44

`last_updated` (sym-
`fem.elements.rhct.ReducedHsiehCloughTocher` attribute), 45

`last_updated` (symfem.elements.rt.RaviartThomas attribute), 45

`last_updated` (sym-
`fem.elements.serendipity.Serendipity` attribute), 46

`last_updated` (sym-
`fem.elements.serendipity.SerendipityCurl` attribute), 46

`last_updated` (sym-
`fem.elements.serendipity.SerendipityDiv` attribute), 46

`last_updated` (symfem.elements.taylor.Taylor attribute), 47

`last_updated` (symfem.elements.tnt.TNT attribute), 48

`last_updated` (symfem.elements.tnt.TNTcurl attribute), 48

`last_updated` (symfem.elements.tnt.TNTdiv attribute), 49

`last_updated` (symfem.elements.transition.Transition attribute), 49

`last_updated` (sym-
`fem.elements.trimmed_serendipity.TrimmedSerendipityHcurl` attribute), 50

`last_updated` (sym-
`fem.elements.trimmed_serendipity.TrimmedSerendipityHdiv` attribute), 50

`last_updated` (sym-
`fem.elements.vector_enriched_galerkin.Enrichment` attribute), 51

`last_updated` (sym-
`fem.elements.vector_enriched_galerkin.VectorEnrichedGalerkin` attribute), 51

`last_updated` (symfem.elements.wu_xu.WuXu attribute), 52

`last_updated` (symfem.finite_element.FiniteElement attribute), 77

`LatticeWithLines` (in module symfem.references), 136

`legendre()` (in module symfem.quadrature), 135

`Line` (class in symfem.plotting), 129

`ListOfFunctionals` (in module symfem.functionals), 97

`load_cached_matrix()` (in module symfem.caching), 74

`lobatto()` (in module symfem.quadrature), 135

`lobatto_basis()` (in module symfem.polynomials), 63

`lobatto_basis()` (in module sym-
`fem.polynomials.lobatto`), 56

`lobatto_basis_interval()` (in module sym-
`fem.polynomials.lobatto`), 56

`lobatto_dual_basis()` (in module sym-
`fem.polynomials`), 63

`lobatto_dual_basis()` (in module `symfem.polynomials.lobatto`), 57
`lobatto_dual_basis_interval()` (in module `symfem.polynomials.lobatto`), 56

M

`make_integral_moment_dofs()` (in module `symfem.moments`), 121
`make_lattice()` (`symfem.references.DualPolygon` method), 155
`make_lattice()` (`symfem.references.Hexahedron` method), 149
`make_lattice()` (`symfem.references.Interval` method), 144
`make_lattice()` (`symfem.references.Point` method), 142
`make_lattice()` (`symfem.references.Prism` method), 151
`make_lattice()` (`symfem.references.Pyramid` method), 152
`make_lattice()` (`symfem.references.Quadrilateral` method), 148
`make_lattice()` (`symfem.references.Reference` method), 139
`make_lattice()` (`symfem.references.Tetrahedron` method), 146
`make_lattice()` (`symfem.references.Triangle` method), 145
`make_lattice_float()` (`symfem.references.Reference` method), 139
`make_lattice_with_lines()` (`symfem.references.DualPolygon` method), 155
`make_lattice_with_lines()` (`symfem.references.Hexahedron` method), 150
`make_lattice_with_lines()` (`symfem.references.Interval` method), 144
`make_lattice_with_lines()` (`symfem.references.Point` method), 142
`make_lattice_with_lines()` (`symfem.references.Prism` method), 151
`make_lattice_with_lines()` (`symfem.references.Pyramid` method), 153
`make_lattice_with_lines()` (`symfem.references.Quadrilateral` method), 148
`make_lattice_with_lines()` (`symfem.references.Reference` method), 139
`make_lattice_with_lines()` (`symfem.references.Tetrahedron` method), 147
`make_lattice_with_lines()` (`symfem.references.Triangle` method), 145
`make_lattice_with_lines_float()` (`symfem.references.Reference` method), 139
`make_piecewise_lagrange()` (in module `symfem.elements.guzman_neilan`), 28
`map_pieces()` (`symfem.piecewise_functions.PiecewiseFunction`

method), 126
`map_polyset_from_default()` (`symfem.references.Reference` method), 142
`map_to_cell()` (`symfem.elements.dual.DualCiarletElement` method), 25
`map_to_cell()` (`symfem.finite_element.CiarletElement` method), 81
`map_to_cell()` (`symfem.finite_element.DirectElement` method), 82
`map_to_cell()` (`symfem.finite_element.EnrichedElement` method), 83
`map_to_cell()` (`symfem.finite_element.FiniteElement` method), 79
`MappingNotImplemented`, 116
`MardalTaiWinther` (class in `symfem.elements.mtw`), 37
`Math` (class in `symfem.plotting`), 131
`matrix_from_string()` (in module `symfem.caching`), 74
`matrix_to_string()` (in module `symfem.caching`), 74
`MatrixFunction` (class in `symfem.functions`), 109
`MatrixLagrange` (class in `symfem.elements.lagrange`), 33
`max_order` (`symfem.elements.alfeld_sorokina.AlfeldSorokina` attribute), 12
`max_order` (`symfem.elements.argyris.Argyris` attribute), 12
`max_order` (`symfem.elements.aw.NonConformingArnoldWinther` attribute), 13
`max_order` (`symfem.elements.bell.Bell` attribute), 17
`max_order` (`symfem.elements.bernardi_raugel.BernardiRaugel` attribute), 17
`max_order` (`symfem.elements.bfs.BognerFoxSchmit` attribute), 19
`max_order` (`symfem.elements.crouzeix_raviart.CrouzeixRaviart` attribute), 22
`max_order` (`symfem.elements.dual.BuffaChristiansen` attribute), 26
`max_order` (`symfem.elements.dual.Dual` attribute), 25
`max_order` (`symfem.elements.dual.RotatedBuffaChristiansen` attribute), 26
`max_order` (`symfem.elements.fortin_soulie.FortinSoulie` attribute), 27
`max_order` (`symfem.elements.guzman_neilan.GuzmanNeilan` attribute), 28
`max_order` (`symfem.elements.hct.HsiehCloughTocher` attribute), 29
`max_order` (`symfem.elements.hermite.Hermite` attribute), 30
`max_order` (`symfem.elements.morley.Morley` attribute), 36
`max_order` (`symfem.elements.morley_wang_xu.MorleyWangXu` attribute), 36
`max_order` (`symfem.elements.mtw.MardalTaiWinther`

`attribute`), 37

`max_order` (`symfem.elements.nedelec_prism.Nedelec attribute`), 39

`max_order` (`symfem.elements.p1_iso_p2.P1IsoP2Interval attribute`), 39

`max_order` (`symfem.elements.p1_iso_p2.P1IsoP2Quad attribute`), 40

`max_order` (`symfem.elements.p1_iso_p2.P1IsoP2Tri attribute`), 40

`max_order` (`symfem.elements.p1_macro.P1Macro attribute`), 40

`max_order` (`symfem.elements.rannacher_turek.RannacherTurek attribute`), 43

`max_order` (`symfem.elements.rhct.ReducedHsiehCloughTocher attribute`), 44

`max_order` (`symfem.elements.vector_enriched_galerkin.FiniteElement attribute`), 51

`max_order` (`symfem.elements.wu_xu.WuXu attribute`), 52

`maximum_degree` (`symfem.elements.dual.DualCiarletElement property`), 24

`maximum_degree` (`symfem.finite_element.CiarletElement property`), 79

`maximum_degree` (`symfem.finite_element.DirectElement property`), 81

`maximum_degree` (`symfem.finite_element.EnrichedElement property`), 82

`maximum_degree` (`symfem.finite_element.FiniteElement property`), 77

`maximum_degree()` (`symfem.basis_functions.BasisFunction method`), 72

`maximum_degree()` (`symfem.basis_functions.SubbedBasisFunction method`), 73

`maximum_degree()` (`symfem.functions.AnyFunction method`), 101

`maximum_degree()` (`symfem.functions.MatrixFunction method`), 112

`maximum_degree()` (`symfem.functions.ScalarFunction method`), 106

`maximum_degree()` (`symfem.functions.VectorFunction method`), 109

`maximum_degree()` (`symfem.piecewise_functions.PiecewiseFunction method`), 126

`maxx()` (`symfem.plotting.PictureElement method`), 128

`maxy()` (`symfem.plotting.PictureElement method`), 129

`midpoint()` (`symfem.references.Reference method`), 141

`min_order` (`symfem.elements.abf.ArnoldBoffiFalk attribute`), 10

`min_order` (`symfem.elements.ac.AC attribute`), 11

`min_order` (`symfem.elements.alfeld_sorokina.AlfeldSorokina attribute`), 12

`min_order` (`symfem.elements.argyris.Argyris attribute`), 12

`min_order` (`symfem.elements.aw.ArnoldWinther attribute`), 13

`min_order` (`symfem.elements.aw.NonConformingArnoldWinther attribute`), 13

`min_order` (`symfem.elements.bddm.BDDF attribute`), 14

`min_order` (`symfem.elements.bdfm.BDFM attribute`), 15

`min_order` (`symfem.elements.bdm.BDM attribute`), 16

`min_order` (`symfem.elements.bell.Bell attribute`), 17

`min_order` (`symfem.elements.bernardi_raugel.BernardiRaugel attribute`), 17

`min_order` (`symfem.elements.bernstein.Bernstein attribute`), 19

`min_order` (`symfem.elements.bfs.BognerFoxSchmit attribute`), 19

`min_order` (`symfem.elements.bubble.Bubble attribute`), 20

`min_order` (`symfem.elements.bubble.BubbleEnrichedLagrange attribute`), 20

`min_order` (`symfem.elements.bubble.BubbleEnrichedVectorLagrange attribute`), 21

`min_order` (`symfem.elements.conforming_crouzeix_raviart.ConformingCrouzeixRaviart attribute`), 21

`min_order` (`symfem.elements.crouzeix_raviart.CrouzeixRaviart attribute`), 22

`min_order` (`symfem.elements.direct_serendipity.DirectSerendipity attribute`), 23

`min_order` (`symfem.elements.dpc.DPC attribute`), 23

`min_order` (`symfem.elements.dpc.VectorDPC attribute`), 23

`min_order` (`symfem.elements.dual.BuffaChristiansen attribute`), 26

`min_order` (`symfem.elements.dual.Dual attribute`), 25

`min_order` (`symfem.elements.dual.RotatedBuffaChristiansen attribute`), 26

`min_order` (`symfem.elements.enriched_galerkin.EnrichedGalerkin attribute`), 26

`min_order` (`symfem.elements.fortin_soulie.FortinSoulie attribute`), 27

`min_order` (`symfem.elements.guzman_neilan.GuzmanNeilan attribute`), 28

`min_order` (`symfem.elements.hct.HsiehCloughTocher attribute`), 29

`min_order` (`symfem.elements.hermite.Hermite attribute`), 30

`min_order` (`symfem.elements.hhj.HellanHerrmannJohnson attribute`), 30

`min_order` (`symfem.elements.huang_zhang.HuangZhang attribute`), 31

`min_order` (`symfem.elements.kmv.KongMulderVeldhuizen attribute`), 31

[attribute](#)), 32
[min_order \(symfem.elements.lagrange.Lagrange attribute\)](#), 32
[min_order \(symfem.elements.lagrange.MatrixLagrange attribute\)](#), 33
[min_order \(symfem.elements.lagrange.SymmetricMatrixLagrange attribute\)](#), 33
[min_order \(symfem.elements.lagrange.VectorLagrange attribute\)](#), 33
[min_order \(symfem.elements.lagrange_prism.Lagrange attribute\)](#), 34
[min_order \(symfem.elements.lagrange_prism.VectorLagrange attribute\)](#), 34
[min_order \(symfem.elements.lagrange_pyramid.Lagrange attribute\)](#), 35
[min_order \(symfem.elements.morley.Morley attribute\)](#), 36
[min_order \(symfem.elements.morley_wang_xu.MorleyWangXu attribute\)](#), 36
[min_order \(symfem.elements.mtw.MardalTaiWinther attribute\)](#), 37
[min_order \(symfem.elements.nedelec.NedelecFirstKind attribute\)](#), 38
[min_order \(symfem.elements.nedelec.NedelecSecondKind attribute\)](#), 38
[min_order \(symfem.elements.nedelec_prism.Nedelec attribute\)](#), 39
[min_order \(symfem.elements.p1_iso_p2.P1IsoP2Interval attribute\)](#), 39
[min_order \(symfem.elements.p1_iso_p2.P1IsoP2Quad attribute\)](#), 40
[min_order \(symfem.elements.p1_iso_p2.P1IsoP2Tri attribute\)](#), 39
[min_order \(symfem.elements.p1_macro.P1Macro attribute\)](#), 40
[min_order \(symfem.elements.q.Nedelec attribute\)](#), 42
[min_order \(symfem.elements.q.Q attribute\)](#), 41
[min_order \(symfem.elements.q.RaviartThomas attribute\)](#), 42
[min_order \(symfem.elements.q.VectorQ attribute\)](#), 41
[min_order \(symfem.elements.rannacher_turek.RannacherTurek attribute\)](#), 43
[min_order \(symfem.elements.regge.Regge attribute\)](#), 43
[min_order \(symfem.elements.regge.ReggeTP attribute\)](#), 44
[min_order \(symfem.elements.rhct.ReducedHsiehCloughTocher attribute\)](#), 44
[min_order \(symfem.elements.rt.RaviartThomas attribute\)](#), 45
[min_order \(symfem.elements.serendipity.Serendipity attribute\)](#), 46
[min_order \(symfem.elements.serendipity.SerendipityCurl attribute\)](#), 46
[min_order \(symfem.elements.serendipity.SerendipityDiv attribute\)](#), 46
[min_order \(symfem.elements.taylor.Taylor attribute\)](#), 47
[min_order \(symfem.elements.tnt.TNT attribute\)](#), 48
[min_order \(symfem.elements.tnt.TNTcurl attribute\)](#), 48
[min_order \(symfem.elements.tnt.TNTdiv attribute\)](#), 49
[min_order \(symfem.elements.transition.Transition attribute\)](#), 49
[min_order \(symfem.elements.trimmed_serendipity.TrimmedSerendipityH attribute\)](#), 50
[min_order \(symfem.elements.trimmed_serendipity.TrimmedSerendipityH attribute\)](#), 50
[min_order \(symfem.elements.vector_enriched_galerkin.Enrichment attribute\)](#), 51
[min_order \(symfem.elements.vector_enriched_galerkin.VectorEnrichment attribute\)](#), 51
[min_order \(symfem.elements.wu_xu.WuXu attribute\)](#), 52
[minx\(\)](#) (symfem.plotting.PictureElement method), 128
[miny\(\)](#) (symfem.plotting.PictureElement method), 128
[module](#)
[symfem](#), 10
[symfem.basis_functions](#), 68
[symfem.caching](#), 73
[symfem.create](#), 75
[symfem.elements](#), 10
[symfem.elements._guzman_neilan_tetrahedron](#), 10
[symfem.elements._guzman_neilan_triangle](#), 10
[symfem.elements.abf](#), 10
[symfem.elements.ac](#), 11
[symfem.elements.alfeld_sorokina](#), 11
[symfem.elements.argyris](#), 12
[symfem.elements.aw](#), 13
[symfem.elements.bddm](#), 14
[symfem.elements.bdfm](#), 15
[symfem.elements.bdm](#), 16
[symfem.elements.bell](#), 16
[symfem.elements.bernardi_raugel](#), 17
[symfem.elements.bernstein](#), 17
[symfem.elements.bfs](#), 19
[symfem.elements.bubble](#), 20
[symfem.elements.conforming_crouzeix_raviart](#), 21
[symfem.elements.crouzeix_raviart](#), 22
[symfem.elements.direct_serendipity](#), 22
[symfem.elements.dpc](#), 23
[symfem.elements.dual](#), 24
[symfem.elements.enriched_galerkin](#), 26
[symfem.elements.fortin_soulie](#), 27
[symfem.elements.guzman_neilan](#), 27
[symfem.elements.hct](#), 29
[symfem.elements.hermite](#), 29
[symfem.elements.hhj](#), 30
[symfem.elements.huang_zhang](#), 30
[symfem.elements.kmv](#), 31
[symfem.elements.lagrange](#), 32
[symfem.elements.lagrange_prism](#), 34
[symfem.elements.lagrange_pyramid](#), 35
[symfem.elements.morley](#), 35

- `symfem.elements.morley_wang_xu`, 36
 - `symfem.elements.mtw`, 37
 - `symfem.elements.nedelec`, 37
 - `symfem.elements.nedelec_prism`, 38
 - `symfem.elements.p1_iso_p2`, 39
 - `symfem.elements.p1_macro`, 40
 - `symfem.elements.q`, 41
 - `symfem.elements.rannacher_turek`, 42
 - `symfem.elements.regge`, 43
 - `symfem.elements.rhct`, 44
 - `symfem.elements.rt`, 45
 - `symfem.elements.serendipity`, 45
 - `symfem.elements.taylor`, 47
 - `symfem.elements.tnt`, 47
 - `symfem.elements.transition`, 49
 - `symfem.elements.trimmed_serendipity`, 50
 - `symfem.elements.vector_enriched_galerkin`, 51
 - `symfem.elements.wu_xu`, 51
 - `symfem.finite_element`, 76
 - `symfem.functionals`, 84
 - `symfem.functions`, 98
 - `symfem.geometry`, 113
 - `symfem.mappings`, 115
 - `symfem.moments`, 121
 - `symfem.piecewise_functions`, 122
 - `symfem.plotting`, 126
 - `symfem.polynomials`, 52
 - `symfem.polynomials.dual`, 52
 - `symfem.polynomials.legendre`, 53
 - `symfem.polynomials.lobatto`, 56
 - `symfem.polynomials.polysets`, 57
 - `symfem.quadrature`, 135
 - `symfem.references`, 136
 - `symfem.symbols`, 155
 - `symfem.utils`, 155
 - `symfem.version`, 156
 - `MomentType` (in module `symfem.moments`), 121
 - `MomentTypeInput` (in module `symfem.moments`), 121
 - `Morley` (class in `symfem.elements.morley`), 36
 - `MorleyWangXu` (class in `symfem.elements.morley_wang_xu`), 36
- ## N
- `name` (`symfem.finite_element.FiniteElement` property), 77
 - `name` (`symfem.functionals.BaseFunctional` attribute), 85
 - `name` (`symfem.functionals.DerivativeIntegralMoment` attribute), 95
 - `name` (`symfem.functionals.DerivativePointEvaluation` attribute), 88
 - `name` (`symfem.functionals.DivergenceIntegralMoment` attribute), 95
 - `name` (`symfem.functionals.DotPointEvaluation` attribute), 91
 - `name` (`symfem.functionals.InnerProductIntegralMoment` attribute), 97
 - `name` (`symfem.functionals.IntegralAgainst` attribute), 92
 - `name` (`symfem.functionals.IntegralMoment` attribute), 94
 - `name` (`symfem.functionals.IntegralOfDirectionalMultiderivative` attribute), 93
 - `name` (`symfem.functionals.IntegralOfDivergenceAgainst` attribute), 93
 - `name` (`symfem.functionals.NormalDerivativeIntegralMoment` attribute), 96
 - `name` (`symfem.functionals.NormalInnerProductIntegralMoment` attribute), 97
 - `name` (`symfem.functionals.NormalIntegralMoment` attribute), 96
 - `name` (`symfem.functionals.PointComponentSecondDerivativeEvaluation` attribute), 90
 - `name` (`symfem.functionals.PointDirectionalDerivativeEvaluation` attribute), 89
 - `name` (`symfem.functionals.PointDivergenceEvaluation` attribute), 91
 - `name` (`symfem.functionals.PointEvaluation` attribute), 87
 - `name` (`symfem.functionals.PointInnerProduct` attribute), 90
 - `name` (`symfem.functionals.PointNormalDerivativeEvaluation` attribute), 89
 - `name` (`symfem.functionals.TangentIntegralMoment` attribute), 96
 - `name` (`symfem.functionals.WeightedPointEvaluation` attribute), 87
 - `names` (`symfem.elements.abf.ArnoldBoffiFalk` attribute), 10
 - `names` (`symfem.elements.ac.AC` attribute), 11
 - `names` (`symfem.elements.alfeld_sorokina.AlfeldSorokina` attribute), 12
 - `names` (`symfem.elements.argyris.Argyris` attribute), 12
 - `names` (`symfem.elements.aw.ArnoldWinther` attribute), 13
 - `names` (`symfem.elements.aw.NonConformingArnoldWinther` attribute), 13
 - `names` (`symfem.elements.bddm.BDDF` attribute), 14
 - `names` (`symfem.elements.bdfm.BDFM` attribute), 15
 - `names` (`symfem.elements.bdm.BDM` attribute), 16
 - `names` (`symfem.elements.bell.Bell` attribute), 16
 - `names` (`symfem.elements.bernardi_raugel.BernardiRaugel` attribute), 17
 - `names` (`symfem.elements.bernstein.Bernstein` attribute), 19
 - `names` (`symfem.elements.bfs.BognerFoxSchmit` attribute), 19
 - `names` (`symfem.elements.bubble.Bubble` attribute), 20
 - `names` (`symfem.elements.bubble.BubbleEnrichedLagrange` attribute), 20
 - `names` (`symfem.elements.bubble.BubbleEnrichedVectorLagrange` attribute), 21
 - `names` (`symfem.elements.conforming_crouzeix_raviart.ConformingCrouzeixRaviart` attribute), 21
 - `names` (`symfem.elements.crouzeix_raviart.CrouzeixRaviart` attribute), 22

`names` (`symfem.elements.direct_serendipity.DirectSerendipity` attribute), 22
`names` (`symfem.elements.dpc.DPC` attribute), 23
`names` (`symfem.elements.dpc.VectorDPC` attribute), 23
`names` (`symfem.elements.dual.BuffaChristiansen` attribute), 26
`names` (`symfem.elements.dual.Dual` attribute), 25
`names` (`symfem.elements.dual.RotatedBuffaChristiansen` attribute), 26
`names` (`symfem.elements.enriched_galerkin.EnrichedGalerkin` attribute), 26
`names` (`symfem.elements.fortin_soulie.FortinSoulie` attribute), 27
`names` (`symfem.elements.guzman_neilan.GuzmanNeilan` attribute), 28
`names` (`symfem.elements.hct.HsiehCloughTocher` attribute), 29
`names` (`symfem.elements.hermite.Hermite` attribute), 29
`names` (`symfem.elements.hhj.HellanHerrmannJohnson` attribute), 30
`names` (`symfem.elements.huang_zhang.HuangZhang` attribute), 31
`names` (`symfem.elements.kmv.KongMulderVeldhuizen` attribute), 32
`names` (`symfem.elements.lagrange.Lagrange` attribute), 32
`names` (`symfem.elements.lagrange.MatrixLagrange` attribute), 33
`names` (`symfem.elements.lagrange.SymmetricMatrixLagrange` attribute), 33
`names` (`symfem.elements.lagrange.VectorLagrange` attribute), 33
`names` (`symfem.elements.lagrange_prism.Lagrange` attribute), 34
`names` (`symfem.elements.lagrange_prism.VectorLagrange` attribute), 34
`names` (`symfem.elements.lagrange_pyramid.Lagrange` attribute), 35
`names` (`symfem.elements.morley.Morley` attribute), 36
`names` (`symfem.elements.morley_wang_xu.MorleyWangXu` attribute), 36
`names` (`symfem.elements.mtw.MardalTaiWinther` attribute), 37
`names` (`symfem.elements.nedelec.NedelecFirstKind` attribute), 38
`names` (`symfem.elements.nedelec.NedelecSecondKind` attribute), 38
`names` (`symfem.elements.nedelec_prism.Nedelec` attribute), 38
`names` (`symfem.elements.p1_iso_p2.P1IsoP2Interval` attribute), 39
`names` (`symfem.elements.p1_iso_p2.P1IsoP2Quad` attribute), 40
`names` (`symfem.elements.p1_iso_p2.P1IsoP2Tri` attribute), 39
`names` (`symfem.elements.p1_macro.P1Macro` attribute), 40
`names` (`symfem.elements.q.Nedelec` attribute), 42
`names` (`symfem.elements.q.Q` attribute), 41
`names` (`symfem.elements.q.RaviartThomas` attribute), 42
`names` (`symfem.elements.q.VectorQ` attribute), 41
`names` (`symfem.elements.rannacher_turek.RannacherTurek` attribute), 43
`names` (`symfem.elements.regge.Regge` attribute), 43
`names` (`symfem.elements.regge.ReggeTP` attribute), 44
`names` (`symfem.elements.rhct.ReducedHsiehCloughTocher` attribute), 44
`names` (`symfem.elements.rt.RaviartThomas` attribute), 45
`names` (`symfem.elements.serendipity.Serendipity` attribute), 46
`names` (`symfem.elements.serendipity.SerendipityCurl` attribute), 46
`names` (`symfem.elements.serendipity.SerendipityDiv` attribute), 46
`names` (`symfem.elements.taylor.Taylor` attribute), 47
`names` (`symfem.elements.tnt.TNT` attribute), 48
`names` (`symfem.elements.tnt.TNTcurl` attribute), 48
`names` (`symfem.elements.tnt.TNTdiv` attribute), 48
`names` (`symfem.elements.transition.Transition` attribute), 49
`names` (`symfem.elements.trimmed_serendipity.TrimmedSerendipityHcurl` attribute), 50
`names` (`symfem.elements.trimmed_serendipity.TrimmedSerendipityHdiv` attribute), 50
`names` (`symfem.elements.vector_enriched_galerkin.Enrichment` attribute), 51
`names` (`symfem.elements.vector_enriched_galerkin.VectorEnrichedGalerkin` attribute), 51
`names` (`symfem.elements.wu_xu.WuXu` attribute), 52
`names` (`symfem.finite_element.FiniteElement` attribute), 77
`NCircle` (class in `symfem.plotting`), 130
`Nedelec` (class in `symfem.elements.nedelec_prism`), 38
`Nedelec` (class in `symfem.elements.q`), 42
`NedelecFirstKind` (class in `symfem.elements.nedelec`), 37
`NedelecSecondKind` (class in `symfem.elements.nedelec`), 38
`NonConformingArnoldWinther` (class in `symfem.elements.aw`), 13
`NonDefaultReferenceError`, 137
`norm()` (`symfem.basis_functions.BasisFunction` method), 71
`norm()` (`symfem.functions.AnyFunction` method), 101
`norm()` (`symfem.functions.MatrixFunction` method), 112
`norm()` (`symfem.functions.ScalarFunction` method), 105
`norm()` (`symfem.functions.VectorFunction` method), 108
`norm()` (`symfem.piecewise_functions.PiecewiseFunction` method), 125
`normal()` (`symfem.references.Reference` method), 141
`NormalDerivativeIntegralMoment` (class in `sym-`

- fem.functionals*), 96
- NormalInnerProductIntegralMoment (class in *symfem.functionals*), 97
- NormalIntegralMoment (class in *symfem.functionals*), 96
- NoTensorProduct, 77
- ## O
- on_edge() (*symfem.references.Reference* method), 142
- on_face() (*symfem.references.Reference* method), 142
- ORANGE (*symfem.plotting.Colors* attribute), 127
- orthogonal_basis() (in module *symfem.polynomials*), 63
- orthogonal_basis() (in module *symfem.polynomials.legendre*), 55
- orthogonal_basis_hexahedron() (in module *symfem.polynomials.legendre*), 54
- orthogonal_basis_interval() (in module *symfem.polynomials.legendre*), 53
- orthogonal_basis_prism() (in module *symfem.polynomials.legendre*), 55
- orthogonal_basis_pyramid() (in module *symfem.polynomials.legendre*), 55
- orthogonal_basis_quadrilateral() (in module *symfem.polynomials.legendre*), 54
- orthogonal_basis_tetrahedron() (in module *symfem.polynomials.legendre*), 54
- orthogonal_basis_triangle() (in module *symfem.polynomials.legendre*), 54
- orthonormal_basis() (in module *symfem.polynomials*), 63
- orthonormal_basis() (in module *symfem.polynomials.legendre*), 55
- ## P
- p() (in module *symfem.elements.tnt*), 47
- P1IsoP2Interval (class in *symfem.elements.p1_iso_p2*), 39
- P1IsoP2Quad (class in *symfem.elements.p1_iso_p2*), 40
- P1IsoP2Tri (class in *symfem.elements.p1_iso_p2*), 39
- P1Macro (class in *symfem.elements.p1_macro*), 40
- parse_function_input() (in module *symfem.functions*), 113
- parse_function_list_input() (in module *symfem.functions*), 113
- parse_point() (*symfem.plotting.Picture* method), 132
- parse_point_input() (in module *symfem.geometry*), 114
- parse_set_of_points_input() (in module *symfem.geometry*), 114
- perform_mapping() (*symfem.functionals.BaseFunctional* method), 85
- perform_mapping() (*symfem.functionals.DerivativePointEvaluation* method), 88
- perform_mapping() (*symfem.functionals.IntegralOfDirectionalMultiderivative* method), 94
- Picture (class in *symfem.plotting*), 131
- PictureElement (class in *symfem.plotting*), 128
- pieces (*symfem.piecewise_functions.PiecewiseFunction* property), 122
- PiecewiseFunction (class in *symfem.piecewise_functions*), 122
- plot() (*symfem.functions.AnyFunction* method), 102
- plot_basis_function() (*symfem.fem.finite_element.CiarletElement* method), 80
- plot_basis_function() (*symfem.fem.finite_element.FiniteElement* method), 78
- plot_dof_diagram() (*symfem.fem.finite_element.FiniteElement* method), 77
- plot_entity_diagrams() (*symfem.references.Reference* method), 142
- plot_values() (*symfem.basis_functions.SubbedBasisFunction* method), 73
- plot_values() (*symfem.functions.AnyFunction* method), 102
- plot_values() (*symfem.functions.ScalarFunction* method), 106
- plot_values() (*symfem.functions.VectorFunction* method), 109
- plot_values() (*symfem.piecewise_functions.PiecewiseFunction* method), 126
- Point (class in *symfem.references*), 142
- point_in_interval() (in module *symfem.geometry*), 114
- point_in_quadrilateral() (in module *symfem.geometry*), 115
- point_in_tetrahedron() (in module *symfem.geometry*), 115
- point_in_triangle() (in module *symfem.geometry*), 115
- PointComponentSecondDerivativeEvaluation (class in *symfem.functionals*), 89
- PointDirectionalDerivativeEvaluation (class in *symfem.functionals*), 88
- PointDivergenceEvaluation (class in *symfem.functionals*), 91
- PointEvaluation (class in *symfem.functionals*), 86
- PointInnerProduct (class in *symfem.functionals*), 90
- PointNormalDerivativeEvaluation (class in *symfem.functionals*), 89
- PointOrFunction (in module *symfem.plotting*), 127
- points (*symfem.plotting.Arrow* property), 130
- points (*symfem.plotting.Bezier* property), 129
- points (*symfem.plotting.Fill* property), 131
- points (*symfem.plotting.Line* property), 129
- points (*symfem.plotting.Math* property), 131

points (*symfem.plotting.NCircle* property), 130

points (*symfem.plotting.PictureElement* property), 128

PointType (*in module symfem.geometry*), 113

PointTypeInput (*in module symfem.geometry*), 114

polynomial_set_1d() (*in module symfem.polynomials*), 65

polynomial_set_1d() (*in module symfem.polynomials.polysets*), 57

polynomial_set_vector() (*in module symfem.polynomials*), 65

polynomial_set_vector() (*in module symfem.polynomials.polysets*), 58

Prism (*class in symfem.references*), 151

prism_polynomial_set_1d() (*in module symfem.polynomials*), 66

prism_polynomial_set_1d() (*in module symfem.polynomials.polysets*), 61

prism_polynomial_set_vector() (*in module symfem.polynomials*), 66

prism_polynomial_set_vector() (*in module symfem.polynomials.polysets*), 61

PURPLE (*symfem.plotting.Colors* attribute), 127

Pyramid (*class in symfem.references*), 152

pyramid_polynomial_set_1d() (*in module symfem.polynomials*), 66

pyramid_polynomial_set_1d() (*in module symfem.polynomials.polysets*), 61

pyramid_polynomial_set_vector() (*in module symfem.polynomials*), 66

pyramid_polynomial_set_vector() (*in module symfem.polynomials.polysets*), 62

Q

Q (*class in symfem.elements.q*), 41

Quadrilateral (*class in symfem.references*), 148

quolynomial_set_1d() (*in module symfem.polynomials*), 67

quolynomial_set_1d() (*in module symfem.polynomials.polysets*), 58

quolynomial_set_vector() (*in module symfem.polynomials*), 67

quolynomial_set_vector() (*in module symfem.polynomials.polysets*), 59

R

radau() (*in module symfem.quadrature*), 135

RannacherTurek (*class in symfem.elements.rannacher_turek*), 43

RaviartThomas (*class in symfem.elements.q*), 42

RaviartThomas (*class in symfem.elements.rt*), 45

ReducedHsiehCloughTocher (*class in symfem.elements.rhct*), 44

Reference (*class in symfem.references*), 138

reference_origin (*symfem.references.DualPolygon* attribute), 154

references (*symfem.elements.abf.ArnoldBoffiFalk* attribute), 10

references (*symfem.elements.ac.AC* attribute), 11

references (*symfem.elements.alfeld_sorokina.AlfeldSorokina* attribute), 12

references (*symfem.elements.argyris.Argyris* attribute), 12

references (*symfem.elements.aw.ArnoldWinther* attribute), 13

references (*symfem.elements.aw.NonConformingArnoldWinther* attribute), 13

references (*symfem.elements.bddm.BDDF* attribute), 14

references (*symfem.elements.bdfm.BDFM* attribute), 15

references (*symfem.elements.bdm.BDM* attribute), 16

references (*symfem.elements.bell.Bell* attribute), 16

references (*symfem.elements.bernardi_raugel.BernardiRaugel* attribute), 17

references (*symfem.elements.bernstein.Bernstein* attribute), 19

references (*symfem.elements.bfs.BognerFoxSchmit* attribute), 19

references (*symfem.elements.bubble.Bubble* attribute), 20

references (*symfem.elements.bubble.BubbleEnrichedLagrange* attribute), 20

references (*symfem.elements.bubble.BubbleEnrichedVectorLagrange* attribute), 21

references (*symfem.elements.conforming_crouzeix_raviart.Conforming* attribute), 21

references (*symfem.elements.crouzeix_raviart.CrouzeixRaviart* attribute), 22

references (*symfem.elements.direct_serendipity.DirectSerendipity* attribute), 23

references (*symfem.elements.dpc.DPC* attribute), 23

references (*symfem.elements.dpc.VectorDPC* attribute), 23

references (*symfem.elements.dual.BuffaChristiansen* attribute), 26

references (*symfem.elements.dual.Dual* attribute), 25

references (*symfem.elements.dual.RotatedBuffaChristiansen* attribute), 26

references (*symfem.elements.enriched_galerkin.EnrichedGalerkin* attribute), 26

references (*symfem.elements.fortin_soulie.FortinSoulie* attribute), 27

references (*symfem.elements.guzman_neilan.GuzmanNeilan* attribute), 28

references (*symfem.elements.hct.HsiehCloughTocher* attribute), 29

references (*symfem.elements.hermite.Hermite* attribute), 29

references (*symfem.elements.hhj.HellanHerrmannJohnson* attribute), 30

references (*symfem.elements.huang_zhang.HuangZhang* attribute), 31

references (*symfem.elements.kmv.KongMulderVeldhuizen* attribute), 32

[references \(symfem.elements.lagrange.Lagrange attribute\), 32](#)
[references \(symfem.elements.lagrange.MatrixLagrange attribute\), 33](#)
[references \(symfem.elements.lagrange.SymmetricMatrixLagrange attribute\), 33](#)
[references \(symfem.elements.lagrange.VectorLagrange attribute\), 33](#)
[references \(symfem.elements.lagrange_prism.Lagrange attribute\), 34](#)
[references \(symfem.elements.lagrange_prism.VectorLagrange attribute\), 34](#)
[references \(symfem.elements.lagrange_pyramid.Lagrange attribute\), 35](#)
[references \(symfem.elements.morley.Morley attribute\), 36](#)
[references \(symfem.elements.morley_wang_xu.MorleyWangXu attribute\), 36](#)
[references \(symfem.elements.mtw.MardalTaiWinther attribute\), 37](#)
[references \(symfem.elements.nedelec.NedelecFirstKind attribute\), 38](#)
[references \(symfem.elements.nedelec.NedelecSecondKind attribute\), 38](#)
[references \(symfem.elements.nedelec_prism.Nedelec attribute\), 39](#)
[references \(symfem.elements.p1_iso_p2.P1IsoP2Interval attribute\), 39](#)
[references \(symfem.elements.p1_iso_p2.P1IsoP2Quad attribute\), 40](#)
[references \(symfem.elements.p1_iso_p2.P1IsoP2Tri attribute\), 39](#)
[references \(symfem.elements.p1_macro.P1Macro attribute\), 40](#)
[references \(symfem.elements.q.Nedelec attribute\), 42](#)
[references \(symfem.elements.q.Q attribute\), 41](#)
[references \(symfem.elements.q.RaviartThomas attribute\), 42](#)
[references \(symfem.elements.q.VectorQ attribute\), 41](#)
[references \(symfem.elements.rannacher_turek.RannacherTurek attribute\), 43](#)
[references \(symfem.elements.regge.Regge attribute\), 43](#)
[references \(symfem.elements.regge.ReggeTP attribute\), 44](#)
[references \(symfem.elements.rhct.ReducedHsiehCloughTocher attribute\), 44](#)
[references \(symfem.elements.rt.RaviartThomas attribute\), 45](#)
[references \(symfem.elements.serendipity.Serendipity attribute\), 46](#)
[references \(symfem.elements.serendipity.SerendipityCurl attribute\), 46](#)
[references \(symfem.elements.serendipity.SerendipityDiv attribute\), 46](#)
[references \(symfem.elements.taylor.Taylor attribute\), 47](#)
[references \(symfem.elements.tnt.TNT attribute\), 48](#)
[references \(symfem.elements.tnt.TNTcurl attribute\), 48](#)
[references \(symfem.elements.tnt.TNTdiv attribute\), 49](#)
[references \(symfem.elements.transition.Transition attribute\), 49](#)
[references \(symfem.elements.trimmed_serendipity.TrimmedSerendipity attribute\), 50](#)
[references \(symfem.elements.trimmed_serendipity.TrimmedSerendipity attribute\), 50](#)
[references \(symfem.elements.vector_enriched_galerkin.Enrichment attribute\), 51](#)
[references \(symfem.elements.vector_enriched_galerkin.VectorEnrichment attribute\), 51](#)
[references \(symfem.elements.wu_xu.WuXu attribute\), 52](#)
[references \(symfem.finite_element.FiniteElement attribute\), 77](#)
[Regge \(class in symfem.elements.regge\), 43](#)
[ReggeTP \(class in symfem.elements.regge\), 44](#)
[RotatedBuffaChristiansen \(class in symfem.elements.dual\), 26](#)
[row\(\) \(symfem.functions.MatrixFunction method\), 109](#)

S

[save\(\) \(symfem.plotting.Picture method\), 134](#)
[save_cached_matrix\(\) \(in module symfem.caching\), 74](#)
[Scalar \(in module symfem.quadrature\), 135](#)
[ScalarFunction \(class in symfem.functions\), 103](#)
[ScalarValueOrFloat \(in module symfem.functionals\), 85](#)
[scaled_axes\(\) \(symfem.references.Reference method\), 141](#)
[Serendipity \(class in symfem.elements.serendipity\), 45](#)
[serendipity_indices\(\) \(in module symfem.polynomials\), 67](#)
[serendipity_indices\(\) \(in module symfem.polynomials.polysets\), 59](#)
[serendipity_set_1d\(\) \(in module symfem.polynomials\), 67](#)
[serendipity_set_1d\(\) \(in module symfem.polynomials.polysets\), 60](#)
[serendipity_set_vector\(\) \(in module symfem.polynomials\), 68](#)
[serendipity_set_vector\(\) \(in module symfem.polynomials.polysets\), 60](#)
[SerendipityCurl \(class in symfem.elements.serendipity\), 46](#)
[SerendipityDiv \(class in symfem.elements.serendipity\), 46](#)
[SetOfPoints \(in module symfem.geometry\), 113](#)
[SetOfPointsInput \(in module symfem.geometry\), 114](#)
[SetOfPointsOrFunctions \(in module symfem.plotting\), 127](#)
[shape \(symfem.basis_functions.SubbedBasisFunction property\), 72](#)

[shape \(symfem.functions.AnyFunction property\)](#), 99
[shape \(symfem.functions.MatrixFunction property\)](#), 109
[shape \(symfem.functions.VectorFunction property\)](#), 106
[shape \(symfem.piecewise_functions.PiecewiseFunction property\)](#), 123
[single_choose\(\)](#) (in module `symfem.elements.bernstein`), 18
[SingleMomentTypeInput](#) (in module `symfem.moments`), 121
[SingleSympyFormat](#) (in module `symfem.functions`), 98
[sub_entities\(\)](#) (`symfem.references.Reference` method), 141
[sub_entity\(\)](#) (`symfem.references.Reference` method), 141
[sub_entity_count\(\)](#) (`symfem.references.Reference` method), 141
[SubbedBasisFunction](#) (class in `symfem.basis_functions`), 72
[subs\(\)](#) (`symfem.basis_functions.BasisFunction` method), 72
[subs\(\)](#) (`symfem.functions.AnyFunction` method), 100
[subs\(\)](#) (`symfem.functions.MatrixFunction` method), 110
[subs\(\)](#) (`symfem.functions.ScalarFunction` method), 104
[subs\(\)](#) (`symfem.functions.VectorFunction` method), 107
[subs\(\)](#) (`symfem.piecewise_functions.PiecewiseFunction` method), 124
[symfem](#)
 module, 10
[symfem.basis_functions](#)
 module, 68
[symfem.caching](#)
 module, 73
[symfem.create](#)
 module, 75
[symfem.elements](#)
 module, 10
[symfem.elements._guzman_neilan_tetrahedron](#)
 module, 10
[symfem.elements._guzman_neilan_triangle](#)
 module, 10
[symfem.elements.abf](#)
 module, 10
[symfem.elements.ac](#)
 module, 11
[symfem.elements.alfeld_sorokina](#)
 module, 11
[symfem.elements.argyris](#)
 module, 12
[symfem.elements.aw](#)
 module, 13
[symfem.elements.bddm](#)
 module, 14
[symfem.elements.bdfm](#)
 module, 15
[symfem.elements.bdm](#)
 module, 16
[symfem.elements.bell](#)
 module, 16
[symfem.elements.bernardi_raugel](#)
 module, 17
[symfem.elements.bernstein](#)
 module, 17
[symfem.elements.bfs](#)
 module, 19
[symfem.elements.bubble](#)
 module, 20
[symfem.elements.conforming_crouzeix_raviart](#)
 module, 21
[symfem.elements.crouzeix_raviart](#)
 module, 22
[symfem.elements.direct_serendipity](#)
 module, 22
[symfem.elements.dpc](#)
 module, 23
[symfem.elements.dual](#)
 module, 24
[symfem.elements.enriched_galerkin](#)
 module, 26
[symfem.elements.fortin_soulie](#)
 module, 27
[symfem.elements.guzman_neilan](#)
 module, 27
[symfem.elements.hct](#)
 module, 29
[symfem.elements.hermite](#)
 module, 29
[symfem.elements.hhj](#)
 module, 30
[symfem.elements.huang_zhang](#)
 module, 30
[symfem.elements.kmv](#)
 module, 31
[symfem.elements.lagrange](#)
 module, 32
[symfem.elements.lagrange_prism](#)
 module, 34
[symfem.elements.lagrange_pyramid](#)
 module, 35
[symfem.elements.morley](#)
 module, 35
[symfem.elements.morley_wang_xu](#)
 module, 36
[symfem.elements.mtw](#)
 module, 37
[symfem.elements.nedelec](#)
 module, 37
[symfem.elements.nedelec_prism](#)
 module, 38
[symfem.elements.p1_iso_p2](#)
 module, 39
[symfem.elements.p1_macro](#)

- module, [40](#)
- `symfem.elements.q`
 - module, [41](#)
- `symfem.elements.rannacher_turek`
 - module, [42](#)
- `symfem.elements.regge`
 - module, [43](#)
- `symfem.elements.rhct`
 - module, [44](#)
- `symfem.elements.rt`
 - module, [45](#)
- `symfem.elements.serendipity`
 - module, [45](#)
- `symfem.elements.taylor`
 - module, [47](#)
- `symfem.elements.tnt`
 - module, [47](#)
- `symfem.elements.transition`
 - module, [49](#)
- `symfem.elements.trimmed_serendipity`
 - module, [50](#)
- `symfem.elements.vector_enriched_galerkin`
 - module, [51](#)
- `symfem.elements.wu_xu`
 - module, [51](#)
- `symfem.finite_element`
 - module, [76](#)
- `symfem.functionals`
 - module, [84](#)
- `symfem.functions`
 - module, [98](#)
- `symfem.geometry`
 - module, [113](#)
- `symfem.mappings`
 - module, [115](#)
- `symfem.moments`
 - module, [121](#)
- `symfem.piecewise_functions`
 - module, [122](#)
- `symfem.plotting`
 - module, [126](#)
- `symfem.polynomials`
 - module, [52](#)
- `symfem.polynomials.dual`
 - module, [52](#)
- `symfem.polynomials.legendre`
 - module, [53](#)
- `symfem.polynomials.lobatto`
 - module, [56](#)
- `symfem.polynomials.polysets`
 - module, [57](#)
- `symfem.quadrature`
 - module, [135](#)
- `symfem.references`
 - module, [136](#)
- `symfem.symbols`
 - module, [155](#)
- `symfem.utils`

- module, [155](#)
- `symfem.version`
 - module, [156](#)
- `SymmetricMatrixLagrange` (class in `symfem.elements.lagrange`), [33](#)
- `SympyFormat` (in module `symfem.functions`), [98](#)

T

- `t` (in module `symfem.symbols`), [155](#)
- `tabulate_basis()` (`symfem.finite_element.FiniteElement` method), [78](#)
- `tabulate_basis_float()` (`symfem.finite_element.FiniteElement` method), [78](#)
- `TabulatedBasis` (in module `symfem.finite_element`), [77](#)
- `tangent()` (`symfem.references.Reference` method), [141](#)
- `TangentIntegralMoment` (class in `symfem.functionals`), [96](#)
- `Taylor` (class in `symfem.elements.taylor`), [47](#)
- `test()` (`symfem.finite_element.CiarletElement` method), [81](#)
- `test()` (`symfem.finite_element.DirectElement` method), [82](#)
- `test()` (`symfem.finite_element.EnrichedElement` method), [83](#)
- `test()` (`symfem.finite_element.FiniteElement` method), [79](#)
- `test_continuity()` (`symfem.finite_element.FiniteElement` method), [79](#)
- `test_dof_points()` (`symfem.finite_element.CiarletElement` method), [81](#)
- `test_functional_entities()` (`symfem.finite_element.CiarletElement` method), [81](#)
- `test_functionals()` (`symfem.finite_element.CiarletElement` method), [81](#)
- `test_independence()` (`symfem.finite_element.DirectElement` method), [82](#)
- `Tetrahedron` (class in `symfem.references`), [146](#)
- `tex_font_size()` (in module `symfem.plotting`), [127](#)
- `TNT` (class in `symfem.elements.tnt`), [48](#)
- `TNTcurl` (class in `symfem.elements.tnt`), [48](#)
- `TNTdiv` (class in `symfem.elements.tnt`), [48](#)
- `to_2d()` (`symfem.plotting.Picture` method), [132](#)
- `Transition` (class in `symfem.elements.transition`), [49](#)
- `transpose()` (`symfem.basis_functions.BasisFunction` method), [72](#)
- `transpose()` (`symfem.functions.AnyFunction` method), [102](#)
- `transpose()` (`symfem.functions.MatrixFunction` method), [112](#)

`transpose()` (*symfem.piecewise_functions.PiecewiseFunction* method), 125

`Triangle` (class in *symfem.references*), 145

`TrimmedSerendipityHcurl` (class in *symfem.elements.trimmed_serendipity*), 50

`TrimmedSerendipityHdiv` (class in *symfem.elements.trimmed_serendipity*), 50

V

`ValuesToSubstitute` (in module *symfem.functions*), 103

`VectorDPC` (class in *symfem.elements.dpc*), 23

`VectorEnrichedGalerkin` (class in *symfem.elements.vector_enriched_galerkin*), 51

`VectorFunction` (class in *symfem.functions*), 106

`VectorLagrange` (class in *symfem.elements.lagrange*), 33

`VectorLagrange` (class in *symfem.elements.lagrange_prism*), 34

`VectorQ` (class in *symfem.elements.q*), 41

`version` (in module *symfem.version*), 156

`volume()` (*symfem.references.DualPolygon* method), 154

`volume()` (*symfem.references.Hexahedron* method), 150

`volume()` (*symfem.references.Interval* method), 145

`volume()` (*symfem.references.Point* method), 143

`volume()` (*symfem.references.Prism* method), 152

`volume()` (*symfem.references.Pyramid* method), 153

`volume()` (*symfem.references.Quadrilateral* method), 149

`volume()` (*symfem.references.Reference* method), 140

`volume()` (*symfem.references.Tetrahedron* method), 147

`volume()` (*symfem.references.Triangle* method), 146

W

`WeightedPointEvaluation` (class in *symfem.functionals*), 87

`WHITE` (*symfem.plotting.Colors* attribute), 127

`with_floats()` (*symfem.basis_functions.BasisFunction* method), 72

`with_floats()` (*symfem.functions.AnyFunction* method), 101

`with_floats()` (*symfem.functions.MatrixFunction* method), 112

`with_floats()` (*symfem.functions.ScalarFunction* method), 106

`with_floats()` (*symfem.functions.VectorFunction* method), 109

`with_floats()` (*symfem.piecewise_functions.PiecewiseFunction* method), 126

`WuXu` (class in *symfem.elements.wu_xu*), 52

`x` (in module *symfem.symbols*), 155

Z

`z()` (*symfem.plotting.Picture* method), 132

`z_ordered_entities()` (*symfem.references.Hexahedron* method), 149

`z_ordered_entities()` (*symfem.references.Prism* method), 151

`z_ordered_entities()` (*symfem.references.Pyramid* method), 152

`z_ordered_entities()` (*symfem.references.Reference* method), 139

`z_ordered_entities()` (*symfem.references.Tetrahedron* method), 146

`z_ordered_entities_extra_dim()` (*symfem.references.DualPolygon* method), 155

`z_ordered_entities_extra_dim()` (*symfem.references.Quadrilateral* method), 148

`z_ordered_entities_extra_dim()` (*symfem.references.Reference* method), 139

`z_ordered_entities_extra_dim()` (*symfem.references.Triangle* method), 145