Mozart Documentation

Release 0.0

Yoon-gu Hwang, Dong-wook Shin, and Ji-yeon Suh

Apr 17, 2017

Contents

CHAPTER 1

Contents

Get Started

Supports

- Only 64 bit
- `Windows` and `Ubuntu`
- `Python 2.7` and `Python 3.5`

Installation

- Run `pip install git+https://github.com/yoon-gu/Mozart.git`

Test Code

```
>>> import mozart as mz
```

Mesh

Interval Element

`mozart.mesh.interval.interval`(*a*, *b*, *M*, *degree*)

Generates mesh information on an interval [a,b].

Parameters

- `a (float)` : coordinate of left-end point of the interval

- `b` (`float`) : coordinate of right-end point of the interval
- `M` (`int`) : the number of elements
- `degree` (`int`) : polynomial degree for the approximate solution

Returns

- `c4n` (`float array`) : coordinates for nodes
- `n4e` (`int array`) : nodes for elements
- `n4db` (`int array`) : nodes for Dirichlet boundary
- `ind4e` (`int array`) : indices for elements

Example

```
>>> c4n, n4e, n4db, ind4e = interval(0,1,4,2)
>>> c4n
array([ 0.   ,  0.125,  0.25 ,  0.375,  0.5  ,  0.625,  0.75 ,  0.875,  1.   ␣
↪])
>>> n4e
array([[0, 2],
   [2, 4],
   [4, 6],
   [6, 8]])
>>> n4db
array([0, 8])
>>> ind4e
array([[0, 1, 2],
   [2, 3, 4],
   [4, 5, 6],
   [6, 7, 8]])
```

Triangle Element

`mozart.mesh.triangle.`**`compute_e4s`**(*n4e*)

Get a matrix whose each row contains two elements sharing the corresponding side If second column is -1, the corresponding side is on the boundary

Paramters

- `n4e` (`int32 array`) : nodes for elements

Returns

- `e4s` (`int32 array`) : elements for sides

Example

```
>>> n4e = np.array([[1, 3, 0], [3, 1, 2]])
>>> e4s = compute_e4s(n4e)
>>> e4s
array([[ 0,  1],
   [ 0, -1],
   [ 1, -1],
   [ 0, -1],
   [ 1, -1]])
```

`mozart.mesh.triangle.`**`compute_n4s`**(*n4e*)

Get a matrix whose each row contains end points of the corresponding side (or edge)

Paramters

- `n4e` (`int32 array`) : nodes for elements

Returns

- `n4s` (`int32 array`) : nodes for sides

Example

```
>>> n4e = np.array([[1, 3, 0], [3, 1, 2]])
>>> n4s = compute_n4s(n4e)
>>> n4s
array([[1, 3],
   [3, 0],
   [1, 2],
   [0, 1],
   [2, 3]])
```

`mozart.mesh.triangle.`**`compute_s4e`**(*n4e*)

Get a matrix whose each row contains three side numbers of the corresponding element

Paramters

- `n4e` (`int32 array`) : nodes for elements

Returns

- `s4e` (`int32 array`) : sides for elements

Example

```
>>> n4e = np.array([[1, 3, 0], [3, 1, 2]])
>>> s4e = compute_s4e(n4e)
>>> s4e
array([[0, 1, 3],
   [0, 2, 4]])
```

`mozart.mesh.triangle.`**`refineUniformRed`**(*c4n*, *n4e*, *n4Db*, *n4Nb*)

Refine a given mesh uniformly using the red refinement

Paramters

- `c4n` (`float64 array`) : coordinates for elements
- `n4e` (`int32 array`) : nodes for elements
- `n4Db` (`int32 array`) : nodes for Difichlet boundary
- `n4Nb` (`int32 array`) : nodes for Neumann boundary

Returns

- `c4nNew` (`float64 array`) : coordinates for element obtained from red refinement
- `n4eNew` (`int32 array`) : nodes for element obtained from red refinement
- `n4DbNew` (`int32 array`) : nodes for Dirichlet boundary obtained from red refinement
- `n4NbNew` (`int32 array`) : nodes for Neumann boundary obtained from red refinement

Example

```
>>> c4n = np.array([[0., 0.], [1., 0.], [1., 1.], [0., 1.]])
>>> n4e = np.array([[1, 3, 0], [3, 1, 2]])
>>> n4Db = np.array([[0, 1], [1, 2]])
>>> n4Nb = np.array([[2, 3],[3, 0]])
>>> c4nNew, n4eNew, n4DbNew, n4NbNew = refineUniformRed(c4n, n4e, n4Db, n4Nb)
>>> c4nNew
array([[ 0. ,  0. ],
   [ 1. ,  0. ],
   [ 1. ,  1. ],
   [ 0. ,  1. ],
   [ 0.5,  0.5],
   [ 0. ,  0.5],
   [ 1. ,  0.5],
   [ 0.5,  0. ],
   [ 0.5,  1. ]])
>>> n4eNew
array([[1, 4, 7],
   [4, 3, 5],
   [5, 7, 4],
   [7, 5, 0],
   [3, 4, 8],
   [4, 1, 6],
   [6, 8, 4],
   [8, 6, 2]])
>>> n4DbNew
array([[0, 7],
   [7, 1],
   [1, 6],
   [6, 2]])
>>>n4NbNew
array([[2, 8],
   [8, 3],
   [3, 5],
   [5, 0]])
```

Rectangle Element

`mozart.mesh.rectangle.`**`rectangle`**(*x1*, *x2*, *y1*, *y2*, *Mx*, *My*, *degree*)

Generates mesh information on the unit square [x1,x2]x[y1,y2].

Parameters

- `x1` (`float`) : coordinate of left point on the x-axis
- `x2` (`float`) : coordinate of right point on the x-axis
- `y1` (`float`) : coordinate of bottom point on the y-axis
- `y2` (`float`) : coordinate of top point on the y-axis
- `Mx` (`int`) : the number of elements along x-axis
- `My` (`int`) : the number of elements along y-axis
- `degree` (`int`) : polynomial degree for the approximate solution

Returns

- `c4n` (`float array`) : coordinates for nodes

- ind4e (int array) : indices for elements
- n4e (int array) : nodes for elements
- n4Db (int array) : nodes for Dirichlet boundary

Example

```
>>> c4n, ind4e, n4e, n4Db = rectangle(0,1,0,1,2,2,1)
>>> c4n
array([[ 0.   0. ]
       [ 0.5  0. ]
       [ 1.   0. ]
       [ 0.   0.5]
       [ 0.5  0.5]
       [ 1.   0.5]
       [ 0.   1. ]
       [ 0.5  1. ]
       [ 1.   1. ]])
>>> ind4e
array([[0 1 3 4]
       [1 2 4 5]
       [3 4 6 7]
       [4 5 7 8]])
>>> n4e
array([[0 1 4 3]
       [1 2 5 4]
       [3 4 7 6]
       [4 5 8 7]])
>>> n4Db
array([0 1 2 3 5 6 7 8])
```

Cube Element

mozart.mesh.cube.**cube**(*x1*, *x2*, *y1*, *y2*, *z1*, *z2*, *Mx*, *My*, *Mz*, *degree*)

Generates mesh information on the unit square [x1,x2]x[y1,y2].

Parameters

- x1 (float) : coordinate of back point on the x-axis
- x2 (float) : coordinate of front point on the x-axis
- y1 (float) : coordinate of left point on the y-axis
- y2 (float) : coordinate of right point on the y-axis
- z1 (float) : coordinate of bottom point on the z-axis
- z2 (float) : coordinate of top point on the z-axis
- Mx (int) : the number of elements along x-axis
- My (int) : the number of elements along y-axis
- Mz (int) : the number of elements along z-axis
- degree (int) : polynomial degree for the approximate solution

Returns

- c4n (float array) : coordinates for nodes

- ind4e (int array) : indices for elements
- n4e (int array) : nodes for elements
- n4Db (int array) : nodes for Dirichlet boundary

Example

```
>>> c4n, ind4e, n4e, n4Db = cube(0,1,0,1,0,1,2,2,2,1)
>>> c4n
array([[ 0.   0.   0. ], [ 0.5  0.   0. ], [ 1.   0.   0. ], [ 0.   0.5  0. ]
       [ 0.5  0.5  0. ], [ 1.   0.5  0. ], [ 0.   1.   0. ], [ 0.5  1.  
↪0. ]
       [ 1.   1.   0. ], [ 0.   0.   0.5], [ 0.5  0.   0.5], [ 1.   0.  
↪0.5]
       [ 0.   0.5  0.5], [ 0.5  0.5  0.5], [ 1.   0.5  0.5], [ 0.   1.  
↪0.5]
       [ 0.5  1.   0.5], [ 1.   1.   0.5], [ 0.   0.   1. ], [ 0.5  0.  
↪1. ]
       [ 1.   0.   1. ], [ 0.   0.5  1. ], [ 0.5  0.5  1. ], [ 1.   0.5 
↪1. ]
       [ 0.   1.   1. ], [ 0.5  1.   1. ], [ 1.   1.   1. ]])
>>> ind4e
array([[ 0  1  3  4  9 10 12 13]
       [ 1  2  4  5 10 11 13 14]
       [ 3  4  6  7 12 13 15 16]
       [ 4  5  7  8 13 14 16 17]
       [ 9 10 12 13 18 19 21 22]
       [10 11 13 14 19 20 22 23]
       [12 13 15 16 21 22 24 25]
       [13 14 16 17 22 23 25 26]])
>>> n4e
array([[ 0  1  4  3  9 10 13 12]
       [ 1  2  5  4 10 11 14 13]
       [ 3  4  7  6 12 13 16 15]
       [ 4  5  8  7 13 14 17 16]
       [ 9 10 13 12 18 19 22 21]
       [10 11 14 13 19 20 23 22]
       [12 13 16 15 21 22 25 24]
       [13 14 17 16 22 23 26 25]])
>>> n4Db
array([ 0  1  2  3  4  5  6  7  8  9 10 11 12 14 15 16 17 18 19 20 21 22 23 
↪24 25 26])
```

Poisson Equation

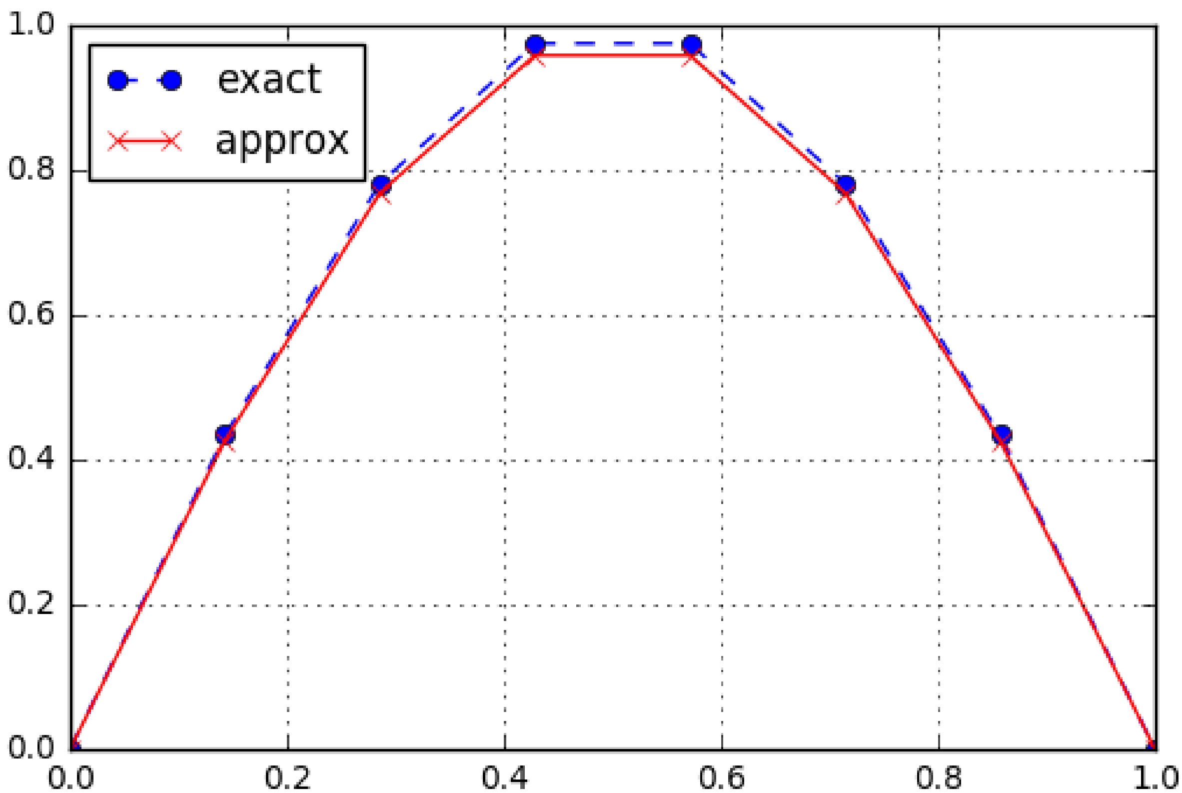
$$\begin{cases} -\nabla^2 u = f(x) & \text{in } \Omega \\ u = 0 & \text{on } \partial\Omega \end{cases}$$

1 Dimensional Case

Example

```
>>> import numpy as np
>>> from mozart.mesh.rectangle import interval
```

```
>>> from mozart.poisson.fem.interval import solve
>>> f = lambda x: np.pi ** 2 * np.sin(np.pi * x)
>>> u_D = lambda x: np.zeros_like(x)
>>> nrElems, degree = (7, 1)
>>> c4n, n4e, n4db, ind4e = interval(0, 1, nrElems, degree)
>>> u = solve(c4n, n4e, n4db, ind4e, f, u_D, degree)
>>> u
array([ 0.        ,  0.42667492,  0.76884164,  0.95872984,  0.95872984,
0.76884164,  0.42667492,  0.        ])
```



mozart.poisson.fem.interval.**computeError**(*c4n, n4e, ind4e, exact_u, exact_ux, approx_u, degree, degree_i*)

Computes L^2-error and semi H^1-error between exact solution and approximate solution.

Parameters

- c4n (float64 array) : coordinates for nodes
- n4e (int32 array) : nodes for elements
- ind4e (int32 array) : indices for elements
- exact_u (lambda) : exact solution
- exact_ux (lambda) : derivative of exact solution
- approx_u (float64 array) : approximate solution

- `degree` (`int32`) : Polynomial degree
- `degree_i` (`int32`) : Polynomial degree for interpolation

Returns

- `L2error` (`float64`) : L^2 error between exact solution and approximate solution.
- `sH1error` (`float64`) : semi H^1 error between exact solution and approximate solution.

Example

```
>>> N = 2
>>> from mozart.mesh.interval import interval
>>> c4n, n4e, n4db, ind4e = interval(0, 1, 4, 2)
>>> f = lambda x: np.pi ** 2 * np.sin(np.pi * x)
>>> u_D = lambda x: np.zeros_like(x)
>>> from mozart.poisson.fem.interval import solve_p
>>> x = solve_p(c4n, n4e, n4db, ind4e, f, u_D, N)
>>> from mozart.poisson.fem.interval import computeError
>>> exact_u = lambda x: np.sin(np.pi * x)
>>> exact_ux = lambda x: np.pi * np.cos(np.pi * x)
>>> L2error, sH1error = computeError(c4n, n4e, ind4e, exact_u, exact_ux, x, N,
↪ N+3)
>>> L2error
0.0020225729623142077
>>> sH1error
0.050627798159754444
```

`mozart.poisson.fem.interval.`**`getMatrix`**(*degree*)

Get FEM matrices on the reference domain I = [-1, 1]

Paramters

- `degree` (`int32`) : degree of polynomial

Returns

- `M_R` (`float64 array`) : Mass matrix on the reference domain
- `S_R` (`float64 array`) : Stiffness matrix on the reference domain
- `D_R` (`float64 array`) : Differentiation matrix on the reference domain

`mozart.poisson.fem.interval.`**`solve`**(*c4n*, *n4e*, *n4db*, *ind4e*, *f*, *u_D*, *degree*)

Computes the coordinates of nodes and elements.

Parameters

- `c4n` (`float64 array`) : coordinates for nodes
- `n4e` (`int32 array`) : nodes for elements
- `n4db` (`int32 array`) : nodes for Dirichlet boundary
- `ind4e` (`int32 array`) : indices for elements
- `f` (`lambda`) : source term
- `u_D` (`lambda`) : Dirichlet boundary condition
- `degree` (`int32`) : Polynomial degree

Returns

- `x` (`float64 array`) : solution

Example

```
>>> N = 2
>>> from mozart.mesh.interval import interval
>>> c4n, n4e, n4db, ind4e = interval(0, 1, 4, 2)
>>> f = lambda x: np.ones_like(x)
>>> u_D = lambda x: np.zeros_like(x)
>>> from mozart.poisson.fem.interval import solve
>>> x = solve(c4n, n4e, n4db, ind4e, f, u_D, N)
>>> x
array([ 0.        ,  0.0546875,  0.09375  ,  0.1171875,  0.125    ,
    0.1171875,  0.09375  ,  0.0546875,  0.        ])
```

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

m

Index

C

G

I

M

R

S