matrixutils documentation

Release 0.0.3b0

OpenGeophysics Developers

Apr 03, 2018

Contents

CHAPTER 1

Why

matrixutils is a python package that includes utilities for working with matrices as linear operators in python. It includes utilities for wrapping and unwrapping matrices and vectors, tools for creating matrices, and operators *Zero* and *Identity* which improve code efficiency without sacrificing readability.

It is used by discretize and SimPEG.

CHAPTER 2

Installation

```
pip install matrixutils
```

To install as a developer

```
git clone https://github.com/opengeophysics/matrixutils.git
cd matrixutils
python setup.py install
```

or to build the installation in-place:

```
git clone https://github.com/opengeophysics/matrixutils.git
cd matrixutils
python setup.py install build_ext --inplace
```

CHAPTER 3

Links

Website: http://simpeg.xyz

Documentation: http://matrixutils.readthedocs.io

Code: https://github.com/opengeophysics/matrixutils

Tests: Coming soon...

Bugs & Issues: https://github.com/opengeophysics/matrixutils/issues

Questions: https://groups.google.com/forum/#!forum/simpeg

Chat: http://slack.simpeg.xyz/

3.1 Examples

3.1.1 Examples

3.2 API

3.2.1 Matrix Utilities

matrixutils.matutils.**mkvc**(*x*, *numDims=1*)

Creates a vector with the number of dimension specified

e.g.:

```
a = np.array([1, 2, 3])

mkvc(a, 1).shape
    > (3, )

mkvc(a, 2).shape
```

```
    > (3, 1)

mkvc(a, 3).shape
    > (3, 1, 1)
```

matrixutils.matutils.**sdiag**(*h*)
Sparse diagonal matrix

matrixutils.matutils.**sdInv**(*M*)
Inverse of a sparse diagonal matrix

matrixutils.matutils.**speye**(*n*)
Sparse identity

matrixutils.matutils.**kron3**(*A*, *B*, *C*)
Three kron prods

matrixutils.matutils.**spzeros**(*n1*, *n2*)
a sparse matrix of zeros

matrixutils.matutils.**ddx**(*n*)
Define 1D derivatives, inner, this means we go from n+1 to n

matrixutils.matutils.**av**(*n*)
Define 1D averaging operator from nodes to cell-centers.

matrixutils.matutils.**av_extrap**(*n*)
Define 1D averaging operator from cell-centers to nodes.

matrixutils.matutils.**ndgrid**(**args*, ***kwargs*)
Form tensorial grid for 1, 2, or 3 dimensions.

Returns as column vectors by default.

To return as matrix input:

ndgrid(. . . , vector=False)

The inputs can be a list or separate arguments.

e.g.:

```
a = np.array([1, 2, 3])
b = np.array([1, 2])

XY = ndgrid(a, b)
    > [[1 1]
       [2 1]
       [3 1]
       [1 2]
       [2 2]
       [3 2]]

X, Y = ndgrid(a, b, vector=False)
    > X = [[1 1]
           [2 2]
           [3 3]]
    > Y = [[1 2]
           [1 2]
           [1 2]]
```

matrixutils.matutils.**ind2sub**(*shape*, *inds*)
From the given shape, returns the subscripts of the given index

matrixutils.matutils.**sub2ind**(*shape*, *subs*)
From the given shape, returns the index of the given subscript

matrixutils.matutils.**getSubArray**(*A*, *ind*)
subArray

matrixutils.matutils.**inv3X3BlockDiagonal**(*a11*, *a12*, *a13*, *a21*, *a22*, *a23*, *a31*, *a32*, *a33*, *returnMatrix=True*)
B = inv3X3BlockDiagonal(a11, a12, a13, a21, a22, a23, a31, a32, a33)

inverts a stack of 3x3 matrices

Input: A - a11, a12, a13, a21, a22, a23, a31, a32, a33

Output: B - inverse

matrixutils.matutils.**inv2X2BlockDiagonal**(*a11*, *a12*, *a21*, *a22*, *returnMatrix=True*)
B = inv2X2BlockDiagonal(a11, a12, a21, a22)

Inverts a stack of 2x2 matrices by using the inversion formula

inv(A) = (1/det(A)) * cof(A)^T

Input: A - a11, a12, a21, a22

Output: B - inverse

class matrixutils.matutils.**Zero**
An efficient zero object.

transpose()

T

class matrixutils.matutils.**Identity**(*positive=True*)
An efficient identity object.

T

transpose()

3.2.2 Curv Utilities

matrixutils.curvutils.**volTetra**(*xyz*, *A*, *B*, *C*, *D*)
Returns the volume for tetrahedras volume specified by the indexes A to D.

Parameters

- **xyz** (*numpy.ndarray*) – X,Y,Z vertex vector
- **A,B,C,D** (*numpy.ndarray*) – vert index of the tetrahedra

Return type numpy.ndarray

Returns V, volume of the tetrahedra

Algorithm https://en.wikipedia.org/wiki/Tetrahedron#Volume

$$V = \frac{1}{3}Ah$$

$$V = \frac{1}{6}|(a-d)\cdot((b-d)(c-d))|$$

matrixutils.curvutils.**indexCube**(*nodes*, *gridSize*, *n=None*)

Returns the index of nodes on the mesh.

Input: nodes - string of which nodes to return. e.g. 'ABCD' gridSize - size of the nodal grid n - number of nodes each i,j,k direction: [ni,nj,nk]

Output: index - index in the order asked e.g. 'ABCD' –> (A,B,C,D)

TWO DIMENSIONS:

```
node(i,j)           node(i,j+1)
     A --------------- B
     |                 |
     |    cell(i,j)    |
     |        I        |
     |                 |
    D --------------- C
node(i+1,j)         node(i+1,j+1)
```

THREE DIMENSIONS:

```
      node(i,j,k+1)        node(i,j+1,k+1)
          E ---------------- F
         /|                 / |
        / |                /  |
       /  |               /   |
node(i,j,k)           node(i,j+1,k)
     A --------------- B      |
     |    H ----------|---- G
     |   /cell(i,j)   |    /
     |  /      I      |   /
     | /              |  /
     D --------------- C
node(i+1,j,k)       node(i+1,j+1,k)
```

matrixutils.curvutils.**faceInfo**(*xyz*, *A*, *B*, *C*, *D*, *average=True*, *normalizeNormals=True*)

function [N] = faceInfo(y,A,B,C,D)

Returns the averaged normal, area, and edge lengths for a given set of faces.

If average option is FALSE then N is a cell array {nA,nB,nC,nD}

Input: xyz - X,Y,Z vertex vector A,B,C,D - vert index of the face (counter clockwize)

Options: average - [true]/false, toggles returning all normals or the average

Output: N - average face normal or {nA,nB,nC,nD} if average = false area - average face area edgeLengths - exact edge Lengths, 4 column vector [AB, BC, CD, DA]

see also testFaceNormal testFaceArea

@author Rowan Cockett

Last modified on: 2013/07/26

3.2.3 Mesh Utilities

matrixutils.meshutils.**meshTensor**(*value*)

meshTensor takes a list of numbers and tuples that have the form:

```
mT = [ float, (cellSize, numCell), (cellSize, numCell, factor) ]
```

For example, a time domain mesh code needs many time steps at one time:

```
[(1e-5, 30), (1e-4, 30), 1e-3]
```

Means take 30 steps at 1e-5 and then 30 more at 1e-4, and then one step of 1e-3.

Tensor meshes can also be created by increase factors:

```
[(10.0, 5, -1.3), (10.0, 50), (10.0, 5, 1.3)]
```

When there is a third number in the tuple, it refers to the increase factor, if this number is negative this section of the tensor is flipped right-to-left.

3.2.4 Interpolation Utilities

matrixutils.interputils.**interpmat**(*locs*, *x*, *y=None*, *z=None*)

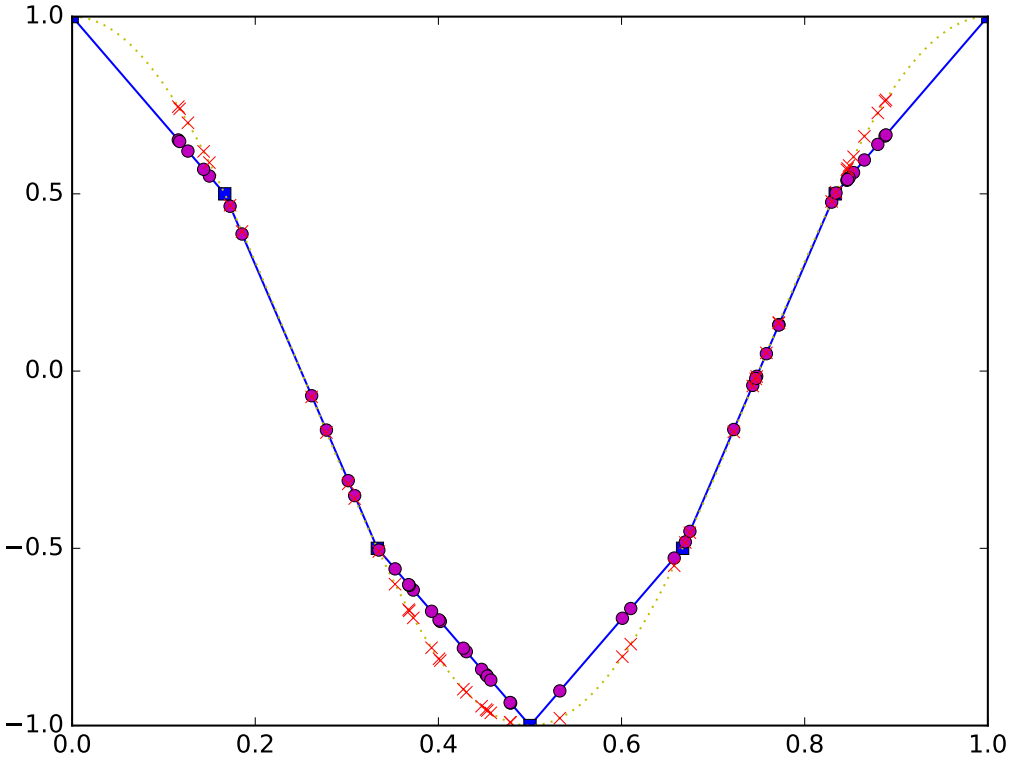
Local interpolation computed for each receiver point in turn

Parameters

- **loc** (*numpy.ndarray*) – Location of points to interpolate to
- **x** (*numpy.ndarray*) – Tensor of 1st dimension of grid.
- **y** (*numpy.ndarray*) – Tensor of 2nd dimension of grid. None by default.
- **z** (*numpy.ndarray*) – Tensor of 3rd dimension of grid. None by default.

Return type scipy.sparse.csr_matrix

Returns Interpolation matrix



3.3 Project Index & Search

- genindex
- modindex
- search

Python Module Index

m

Index