

---

# **python\_lessons Documentation**

*Release alpha*

**Ahmed RATNANI**

November 05, 2014



# CONTENTS

<b>1</b>	<b>Lesson 1</b>	<b>3</b>
1.1	Numbers . . . . .	3
1.2	strings . . . . .	3
1.3	lists . . . . .	4
1.4	numpy . . . . .	5
1.5	From <b>Matlab/Octave</b> to <b>numpy</b> . . . . .	11
1.6	Matplotlib . . . . .	11
1.7	scipy . . . . .	14
1.8	Exercise . . . . .	15
<b>2</b>	<b>Lesson 2</b>	<b>19</b>
2.1	Galerkin-Ritz approximation . . . . .	20
2.2	Assembling process . . . . .	21
2.3	Implementing the Finite Element Method . . . . .	25
<b>3</b>	<b>Indices and tables</b>	<b>27</b>



Contents:



# LESSON 1

In this lesson, you will write your first python code.

## 1.1 Numbers

```
>>> 2 + 2
4
>>> 50 - 5*6
20
>>> (50 - 5.0*6) / 4
5.0
>>> 8 / 5.0
1.6

>>> 17 / 3 # int / int -> int
5
>>> 17 / 3.0 # int / float -> float
5.666666666666667
>>> 17 // 3.0 # explicit floor division discards the fractional part
5.0
>>> 17 % 3 # the % operator returns the remainder of the division
2
>>> 5 * 3 + 2 # result * divisor + remainder
17
```

The equal sign (=) is used to assign a value to a variable. No result is displayed

```
>>> width = 20
>>> height = 5 * 9
>>> width * height
900
```

## 1.2 strings

```
>>> 'spam eggs' # single quotes
'spam eggs'
>>> 'doesn\'t' # use \' to escape the single quote...
"doesn't"
>>> "doesn't" # ...or use double quotes instead
"doesn't"
```

```
>>> ' "Yes," he said.'
' "Yes," he said.'
>>> "\"Yes,\" he said."
' "Yes," he said.'
>>> ' "Isn\'t," she said.'
' "Isn\'t," she said.'

>>> ' "Isn\'t," she said.'
' "Isn\'t," she said.'
>>> print ' "Isn\'t," she said.'
' "Isn\'t," she said.'
>>> s = 'First line.\nSecond line.' # \n means newline
>>> s # without print(), \n is included in the output
'First line.\nSecond line.'
>>> print s # with print, \n produces a new line
First line.
Second line.
```

(+) operator is available for strings

```
>>> s = "Hello world."
>>> t = " Said Python"
>>> s+t
'Hello world. Said Python'
```

In fact, a string is a list object.

## 1.3 lists

Rather than using arrays of a given dimension and size, with **list** object, you can insert new elements or remove them, iterate, ...

```
>>> L = [2, 3, 4, -5, 9, 10]
>>> L[0]
2
>>> L[:]
[2, 3, 4, -5, 9, 10]
>>> L[2:-2]
[4, -5]
```

Lists also supports operations like concatenation:

```
>>> A = [21, 22, 23, 24]
>>> L+A
[2, 3, 4, -5, 9, 10, 21, 22, 23, 24]
```

Insertion can be done using the **append** method

```
>>> L.append(100)
>>> print L
[2, 3, 4, -5, 9, 10, 100]
```

Getting the index of an element can be done using the **index** method

```
>>> L.index(10)
5
```

Removing an element can be done using the **pop** or **remove** methods



```
>>> L.pop(L.index(10))
>>> L
[2, 3, 4, -5, 9, 100]
>>> L.remove(4)
>>> L
[2, 3, -5, 9, 100]
```

The list's length can be accessed using

```
>>> len(L)
5
```

## 1.4 numpy

The first thing to do is to import the **numpy** package:

```
import numpy as np
```

If you want to see what are the functions and modules available in **numpy**, write **np.** and press on **TAB**. The result is:

```
>>> np.
np.ALLOW_THREADS          np.iscomplex
np.BUFSIZE                np.iscomplexobj
np.CLIP                   np.isfinite
np.ComplexWarning        np.isfortran
np.DataSource             np.isinf
np.ERR_CALL               np.isnan
np.ERR_DEFAULT            np.isneginf
np.ERR_DEFAULT2           np.isposinf
np.ERR_IGNORE             np.isreal
np.ERR_LOG                np.isrealobj
np.ERR_PRINT              np.isscalar
np.ERR_RAISE              np.issctype
np.ERR_WARN               np.issubclass_
np.FLOATING_POINT_SUPPORT np.issubdtype
np.FPE_DIVIDEBYZERO      np.issubsctype
np.FPE_INVALID            np.iterable
np.FPE_OVERFLOW           np.ix_
np.FPE_UNDERFLOW         np.kaiser
np.False_                 np.kron
np.Inf                    np.ldexp
np.Infinity               np.left_shift
np.MAXDIMS                np.less
np.MachAr                 np.less_equal
np.NAN                    np.lexsort
np.NINF                   np.lib
np.NZERO                  np.linalg
np.NaN                    np.linspace
np.PINF                   np.little_endian
np.PZERO                  np.load
np.PackageLoader         np.loads
np.RAISE                  np.loadtxt
np.RankWarning           np.log
np.SHIFT_DIVIDEBYZERO    np.log10
np.SHIFT_INVALID         np.log1p
np.SHIFT_OVERFLOW        np.log2
np.SHIFT_UNDERFLOW      np.logaddexp
```

np.ScalarType	np.logaddexp2
np.Tester	np.logical_and
np.True_	np.logical_not
np.UFUNC_BUFSIZE_DEFAULT	np.logical_or
np.UFUNC_PYVALS_NAME	np.logical_xor
np.WRAP	np.logspace
np.abs	np.long
np.absolute	np.longcomplex
np.add	np.longdouble
np.add_docstring	np.longfloat
np.add_newdoc	np.longlong
np.add_newdocs	np.lookfor
np.alen	np.ma
np.all	np.mafromtxt
np.allclose	np.mask_indices
np.alltrue	np.mat
np.alterdot	np.math
np.amax	np.matrix
np.amin	np.matrixlib
np.angle	np.max
np.any	np.maximum
np.append	np.maximum_sctype
np.apply_along_axis	np.may_share_memory
np.apply_over_axes	np.mean
np.arange	np.median
np.arccos	np.memmap
np.arccosh	np.meshgrid
np.arcsin	np.mgrid
np.arcsinh	np.min
np.arctan	np.min_scalar_type
np.arctan2	np.minimum
np.arctanh	np.mintypecode
np.argmax	np.mirr
np.argmin	np.mod
np.argsort	np.modf
np.argwhere	np.msort
np.around	np.multiply
np.array	np.nan
np.array2string	np.nan_to_num
np.array_equal	np.nanargmax
np.array_equiv	np.nanargmin
np.array_repr	np.nanmax
np.array_split	np.nanmin
np.array_str	np.nansum
np.asanyarray	np.nbytes
np.asarray	np.ndarray
np.asarray_chkfinite	np.ndenumerate
np.ascontiguousarray	np.ndfromtxt
np.asfarray	np.ndim
np.asfortranarray	np.ndindex
np.asmatrix	np.nditer
np.asscalar	np.negative
np.atleast_1d	np.nested_iters
np.atleast_2d	np.newaxis
np.atleast_3d	np.newbuffer
np.average	np.nextafter
np.bartlett	np.nonzero
np.base_repr	np.not_equal

---

np.bench	np.nper
np.binary_repr	np.npv
np.bincount	np.number
np.bitwise_and	np.obj2sctype
np.bitwise_not	np.object
np.bitwise_or	np.object0
np.bitwise_xor	np.object_
np.blackman	np.ogrid
np.bmat	np.ones
np.bool	np.ones_like
np.bool8	np.outer
np.bool_	np.packbits
np.broadcast	np.percentile
np.broadcast_arrays	np.pi
np.byte	np.piecewise
np.byte_bounds	np.pkgload
np.bytes_	np.place
np.c_	np.pmt
np.can_cast	np.poly
np.cast	np.poly1d
np.cdouble	np.polyadd
np.ceil	np.polyder
np.cfloat	np.polydiv
np.char	np.polyfit
np.character	np.polyint
np.chararray	np.polymul
np.choose	np.polynomial
np.clip	np.polysub
np.clongdouble	np.polyval
np.clongfloat	np.power
np.column_stack	np.ppmt
np.common_type	np.prod
np.compare_chararrays	np.product
np.compat	np.promote_types
np.complex	np.ptp
np.complex128	np.put
np.complex256	np.putmask
np.complex64	np.pv
np.complex_	np.r_
np.complexfloating	np.rad2deg
np.compress	np.radians
np.concatenate	np.random
np.conj	np.rank
np.conjugate	np.rate
np.convolve	np.ravel
np.copy	np.ravel_multi_index
np.copysign	np.real
np.core	np.real_if_close
np.corrcoef	np.rec
np.correlate	np.reccarray
np.cos	np.recfromcsv
np.cosh	np.recfromtxt
np.count_nonzero	np.reciprocal
np.cov	np.record
np.cross	np.remainder
np.csingle	np.repeat
np.ctypeslib	np.require
np.cumprod	np.reshape

np.cumproduct	np.resize
np.cumsum	np.restoredot
np.datetime64	np.result_type
np.datetime_	np.right_shift
np.datetime_data	np rint
np.deg2rad	np.roll
np.degrees	np.rollaxis
np.delete	np.roots
np.deprecate	np.rot90
np.deprecate_with_doc	np.round
np.diag	np.round_
np.diag_indices	np.row_stack
np.diag_indices_from	np.s_
np.diagflat	np.safe_eval
np.diagonal	np.save
np.diff	np.savetxt
np.digitize	np.savez
np.disp	np.savez_compressed
np.divide	np.sctype2char
np.dot	np.sctypeDict
np.double	np.sctypeNA
np.dsplitt	np.sctypes
np.dstack	np.searchsorted
np.dtype	np.select
np.e	np.set_numeric_ops
np.ediff1d	np.set_printoptions
np.einsum	np.set_string_function
np.emath	np.setbufsize
np.empty	np.setdiff1d
np.empty_like	np.seterr
np.equal	np.seterrcall
np.errstate	np.seterrobj
np.exp	np.setxor1d
np.exp2	np.shape
np.expand_dims	np.short
np.expm1	np.show_config
np.extract	np.sign
np.eye	np.signbit
np.fabs	np.signedinteger
np.fastCopyAndTranspose	np.sin
np.fft	np.sinc
np.fill_diagonal	np.single
np.find_common_type	np.singlecomplex
np.finfo	np.sinh
np.fix	np.size
np.flatiter	np.sometrue
np.flatnonzero	np.sort
np.flexible	np.sort_complex
np.fliplr	np.source
np.flipud	np.spacing
np.float	np.split
np.float128	np.sqrt
np.float16	np.square
np.float32	np.squeeze
np.float64	np.std
np.float_	np.str
np.floating	np.str_
np.floor	np.string0

---

<code>np.floor_divide</code>	<code>np.string_</code>
<code>np.fmax</code>	<code>np.subtract</code>
<code>np.fmin</code>	<code>np.sum</code>
<code>np.fmod</code>	<code>np.swapaxes</code>
<code>np.format_parser</code>	<code>np.take</code>
<code>np.frexp</code>	<code>np.tan</code>
<code>np.frombuffer</code>	<code>np.tanh</code>
<code>np.fromfile</code>	<code>np.tensordot</code>
<code>np.fromfunction</code>	<code>np.test</code>
<code>np.fromiter</code>	<code>np.testing</code>
<code>np.frompyfunc</code>	<code>np.tile</code>
<code>np.fromregex</code>	<code>np.timedelta64</code>
<code>np.fromstring</code>	<code>np.timedelta_</code>
<code>np.fv</code>	<code>np.timeinteger</code>
<code>np.generic</code>	<code>np.trace</code>
<code>np.genfromtxt</code>	<code>np.transpose</code>
<code>np.get_array_wrap</code>	<code>np.trapz</code>
<code>np.get_include</code>	<code>np.tri</code>
<code>np.get_numarray_include</code>	<code>np.tril</code>
<code>np.get_printoptions</code>	<code>np.tril_indices</code>
<code>np.getbuffer</code>	<code>np.tril_indices_from</code>
<code>np.getbufsize</code>	<code>np.trim_zeros</code>
<code>np.geterr</code>	<code>np.triu</code>
<code>np.geterrcall</code>	<code>np.triu_indices</code>
<code>np.geterrobj</code>	<code>np.triu_indices_from</code>
<code>np.gradient</code>	<code>np.true_divide</code>
<code>np.greater</code>	<code>np.trunc</code>
<code>np.greater_equal</code>	<code>np.typeDict</code>
<code>np.half</code>	<code>np.typeNA</code>
<code>np.hamming</code>	<code>np.typecodes</code>
<code>np.hanning</code>	<code>np.typeName</code>
<code>np.histogram</code>	<code>np.ubyte</code>
<code>np.histogram2d</code>	<code>np.ufunc</code>
<code>np.histogramdd</code>	<code>np.uint</code>
<code>np.hsplit</code>	<code>np.uint0</code>
<code>np.hstack</code>	<code>np.uint16</code>
<code>np.hypot</code>	<code>np.uint32</code>
<code>np.i0</code>	<code>np.uint64</code>
<code>np.identity</code>	<code>np.uint8</code>
<code>np.iinfo</code>	<code>np.uintc</code>
<code>np.imag</code>	<code>np.uintp</code>
<code>np.in1d</code>	<code>np.ulonglong</code>
<code>np.index_exp</code>	<code>np.unicode</code>
<code>np.indices</code>	<code>np.unicode0</code>
<code>np.inexact</code>	<code>np.unicode_</code>
<code>np.inf</code>	<code>np.union1d</code>
<code>np.info</code>	<code>np.unique</code>
<code>np.infty</code>	<code>np.unpackbits</code>
<code>np.inner</code>	<code>np.unravel_index</code>
<code>np.insert</code>	<code>np.unsignedinteger</code>
<code>np.int</code>	<code>np.unwrap</code>
<code>np.int0</code>	<code>np.ushort</code>
<code>np.int16</code>	<code>np.vander</code>
<code>np.int32</code>	<code>np.var</code>
<code>np.int64</code>	<code>np.vdot</code>
<code>np.int8</code>	<code>np.vectorize</code>
<code>np.int_</code>	<code>np.version</code>
<code>np.int_asbuffer</code>	<code>np.void</code>

```
np.intc                np.void0
np.integer            np.vsplit
np.interp            np.vstack
np.intersect1d       np.where
np.intp              np.who
np.invert            np.zeros
np.ipmt              np.zeros_like
np.irr
```

Let's take the last list and convert it to a numpy array:

```
>>> x = np.asarray(L)
>>> print x
array([ 2,  3, -5,  9, 100])
```

You notice that the result is no longer a list, but an **array**

Arrays can be reshaped using:

```
>>> L = range(0,12) # create a list or integers from 0 to 11
>>> x = np.asarray(L)
>>> print x
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
>>> print x.shape # access the shape of an array
(12,)
>>> y = x.reshape((3,4)) # reshape
>>> print y
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
>>> print y.shape
(3, 4)
>>> x.ndim, y.ndim
(1,2)
>>> x.dtype.name
'int64'
>>> x.itemsize, y.itemsize
(8, 8)
>>> x.size, y.size
(12,12)
>>> type(x)
numpy.ndarray

>>> a = np.array(1,2,3,4) # WRONG
>>> a = np.array([1,2,3,4]) # RIGHT
```

You can specify the type at the creation, or convert your array:

```
>>> x = np.array( [ [1,2], [3,4] ], dtype=np.float )
>>> c = np.array( x, dtype=complex )
array([[ 1.+0.j,  2.+0.j],
       [ 3.+0.j,  4.+0.j]])
```

Loop over an array (or list) can be done in different ways:

```
>>> L = range(0,12) # create a list or integers from 0 to 11
>>> X = np.asarray(L)
>>> # method 1: simple loop
>>> n = X.shape[0]
>>> for i in range(0,n):
```

```

>>> x = X[i]
>>> print x
>>> # method 2: using enumerate
>>> for i,x in enumerate(X):
>>>     x = X[i]
>>>     print x
>>> # method 3: using intrinsic iterator
>>> for x in X:
>>>     print x

```

Sometimes you need to loop over two lists (or more), this can be done using **zip** function:

```

>>> L = range(0,12) # create a list of integers from 0 to 11
>>> X = np.asarray(L)
>>> Y = X**2 # taking the square of X
>>> for (x,y) in zip(X,Y):
>>>     print x**2-y

```

Another way of computing the square of X is to use a **list**:

```

>>> Z = np.asarray([x**2 for x in X if np.mod(x,2)==0]) # compute squares of even values

```

## 1.5 From Matlab/Octave to numpy

### 1.5.1 Arithmetic operators

Description	Matlab/Octave	Python
Assignment; defining a number	a=1; b=2;	a=1; b=1
Addition	a+b;	a+b or add(a,b)
Substruction	a-b;	a-b or subtract(a,b)
Multiplication	a*b;	a*b or multiply(a,b)
Division	a/b;	a/b or devide(a,b)
Power	a.^b;	a**b or pow(a,b)
Remainder	rem(a,b);	a%b or remainder(a,b)
In place operation	a+=1;	a+=b or add(a,b,a)
Factorial $n!$	factorial(a);	

More informations can be found at <http://mathesaurus.sourceforge.net/matlab-numpy.html>

## 1.6 Matplotlib

Let us consider the function

$$f(x) = \sin(2\pi x), \forall x \in [0, 1]$$

$f$  can be defined in python using:

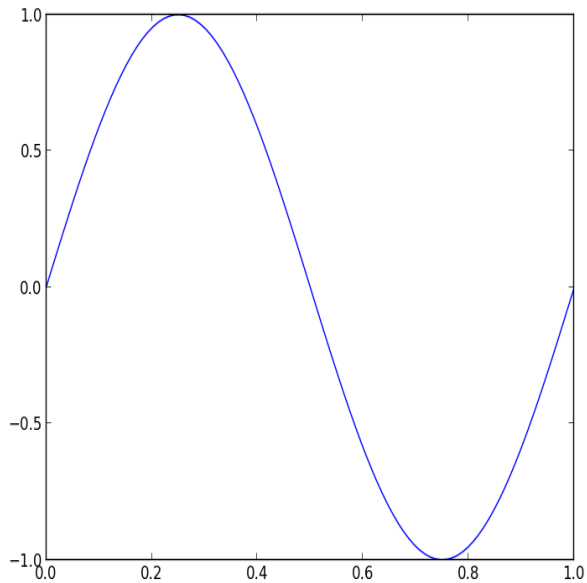
```

>>> from numpy import pi, sin, cos
>>> # method 1
>>> def f(x):
>>>     return sin(2*pi*x)
>>> # method 2
>>> f = lambda x: sin(2*pi*x)

```

The next example shows how to plot  $f$

```
>>> x = np.linspace(0., 1., 100)
>>> y = f(x)
>>> import matplotlib.pyplot as plt
>>> plt.plot(x, y)
>>> plt.show()
```



Let's now take a **2D** function

$$f(x, y) = \sin(2\pi x)\sin(4\pi y), \forall x, y \in [0, 1]$$

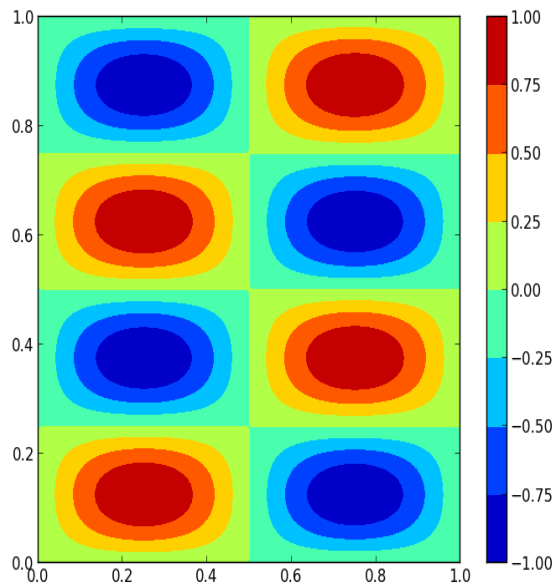
$f$  can be defined in python using:

```
>>> # method 1
>>> def f(x, y):
>>>     return sin(2*pi*x) * sin(4*pi*y)
>>> # method 2
>>> f = lambda x, y: sin(2*pi*x) * sin(4*pi*y)
```

The next example shows how to plot  $f$

```
>>> x = np.linspace(0., 1., 100)
>>> y = np.linspace(0., 1., 100)
>>> X, Y = np.meshgrid(x, y)
>>> Z = f(X, Y)
>>> plt.contourf(X, Y, Z)
>>> plt.colorbar()
>>> plt.show()
```



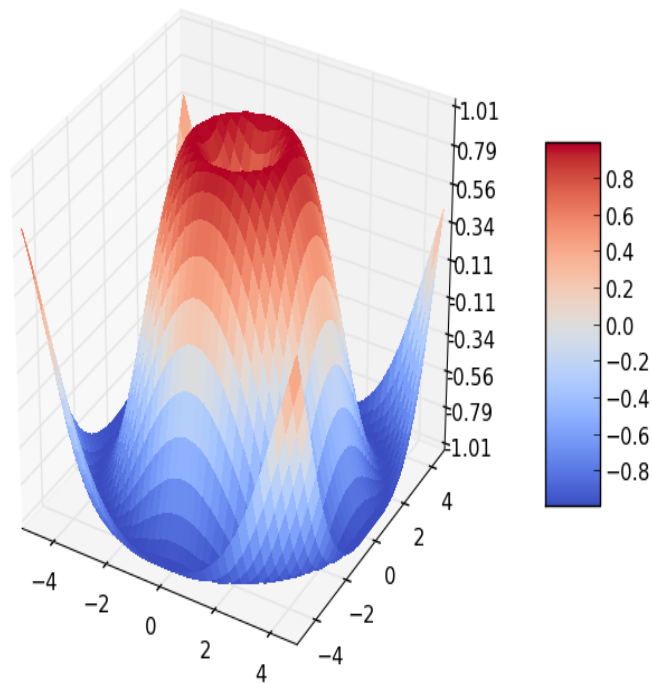


The following example shows how to plot **3D** surface

```

1  #!/usr/bin/python
2
3  from mpl_toolkits.mplot3d import Axes3D
4  from matplotlib import cm
5  from matplotlib.ticker import LinearLocator, FormatStrFormatter
6  import matplotlib.pyplot as plt
7  import numpy as np
8
9  fig = plt.figure()
10 ax = fig.gca(projection='3d')
11 X = np.arange(-5, 5, 0.25)
12 Y = np.arange(-5, 5, 0.25)
13 X, Y = np.meshgrid(X, Y)
14 R = np.sqrt(X**2 + Y**2)
15 Z = np.sin(R)
16 surf = ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap=cm.coolwarm,
17     linewidth=0, antialiased=False)
18 ax.set_zlim(-1.01, 1.01)
19
20 ax.zaxis.set_major_locator(LinearLocator(10))
21 ax.zaxis.set_major_formatter(FormatStrFormatter('%.02f'))
22
23 fig.colorbar(surf, shrink=0.5, aspect=5)
24
25 plt.show()

```



## 1.7 scipy

SciPy (pronounced “Sigh Pie”) is open-source software for mathematics, science, and engineering.

Here is a list of some **scipy** modules

- Discrete Fourier transforms (`scipy.fftpack`)
- Integration and ODEs (`scipy.integrate`)
- Interpolation (`scipy.interpolate`)
- Input and output (`scipy.io`)
- Linear algebra (`scipy.linalg`)
- Optimization and root finding (`scipy.optimize`)
- Sparse matrices (`scipy.sparse`)
- Sparse linear algebra (`scipy.sparse.linalg`)
- Compressed Sparse Graph Routines (`scipy.sparse.csgraph`)
- Spatial algorithms and data structures (`scipy.spatial`)
- Special functions (`scipy.special`)

We will focus on the use of *sparse matrices* and their linear solvers.

## 1.7.1 scipy.sparse

SciPy 2-D sparse matrix package for numeric data.

Available constructors are

- `bsr_matrix`(arg1[, shape, dtype, copy, blocksize]) Block Sparse Row matrix
- `coo_matrix`(arg1[, shape, dtype, copy]) A sparse matrix in COOrdinate format.
- `csc_matrix`(arg1[, shape, dtype, copy]) Compressed Sparse Column matrix
- `csr_matrix`(arg1[, shape, dtype, copy]) Compressed Sparse Row matrix
- `dia_matrix`(arg1[, shape, dtype, copy]) Sparse matrix with DIAGONAL storage
- `dok_matrix`(arg1[, shape, dtype, copy]) Dictionary Of Keys based sparse matrix.
- `lil_matrix`(arg1[, shape, dtype, copy]) Row-based linked list sparse matrix

```
>>> import numpy as np
>>> from scipy.sparse import csr_matrix
>>> A = csr_matrix([[1, 2, 0], [0, 0, 3], [4, 0, 5]])
>>> v = np.array([1, 0, -1])
>>> A.dot(v)
array([ 1, -3, -1], dtype=int64)
```

## 1.8 Exercise

Let us consider the following advection problem,

$$\begin{aligned}\partial_t u + \mathbf{div}(au) &= 0, \quad x \in \Omega = (0, 2\pi) \\ u(t=0, x) &= u_0(x), \quad \forall x \in \Omega\end{aligned}$$

where  $a$  is a constant or any real function.

Let us denote  $A^n$  the resulting finite difference matrix at time  $n$ . The spatial grid is defined by

$$x_i = i\Delta x, \quad \forall i \in \{0, \dots, N\}$$

### 1.8.1 Upwind finite difference matrix

We consider an upwind finite difference method for the spatial discretization.

$$\frac{d}{dt} \mathbf{u} = A(t)\mathbf{u}, \quad \forall t > 0$$

The  $\theta$ -time scheme can be written as

$$\frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{\Delta t} = \theta A^{n+1} \mathbf{u}^{n+1} + (1 - \theta) A^n \mathbf{u}^n, \quad \forall t > 0$$

Reordering the last equation, leads to

$$\{1 - \theta \Delta t A^{n+1}\} \mathbf{u}^{n+1} = \{1 + (1 - \theta) \Delta t A^n\} \mathbf{u}^n$$

In the case of dirichlet boundary, the vector  $\mathbf{u}^n$  is

$$\mathbf{u}^n = \{u_1, u_2, \dots, u_{N-1}\}$$

The finite difference matrix is therefor

$$\begin{aligned} A_{i,i}^n &= \frac{a_i^n}{\Delta x} \\ A_{i,i-1}^n &= -\frac{a_{i-1}^n}{\Delta x} \\ A_{i,j}^n &= 0, \quad \text{if } j \notin \{i-1, i\} \end{aligned}$$

In the case of periodic boundary, the vector  $\mathbf{u}^n$  is

$$\mathbf{u}^n = \{u_1, u_2, \dots, u_{N-1}, u_N\}$$

and the finite difference matrix is

$$\begin{aligned} A_{i,i}^n &= \frac{a_i^n}{\Delta x} \\ A_{i,i-1}^n &= -\frac{a_{i-1}^n}{\Delta x} \\ A_{1,0}^n &:= A_{1,N}^n \\ A_{i,j}^n &= 0, \quad \text{if } j \notin \{i-1, i\} \end{aligned}$$

1. write the python script for these problems
2. study the stability of the  $\theta$ -scheme
3. study the consistence of the  $\theta$ -scheme
4. check the expected analytical CFL number, for the periodic case

## 1.8.2 Dissipation and dispersion

We consider the periodic boundary condition and we assume that  $a$  is a constant. We also consider the Euler centered-explicit scheme, which can be written as

$$u_j^{n+1} = u_j^n - \frac{\lambda a}{2}(u_{j+1}^n - u_{j-1}^n)$$

---

**Note:** any finite difference scheme can be written in the form  $u_j^{n+1} = u_j^n - \lambda(h_{j+\frac{1}{2}}^n - h_{j-\frac{1}{2}}^n)$ . Where  $h_{j+\frac{1}{2}}^n = h(u_j^n, u_{j+1}^n)$ ,  $h(\cdot, \cdot)$  is the numerical flux.

---

In the case of an Euler center-explicit scheme, the numerical flux is given by  $h_{j+\frac{1}{2}}^n = \frac{a}{2}(u_{j+1}^n + u_j^n)$

Let us consider the exact solution

$$u(t^n, x) = u_0(x - an\Delta t), \quad \forall n \geq 0, \quad \forall x \in \Omega$$

We have

$$u(t^n, x_j) = \sum_k \alpha_k e^{ikj\Delta x} g_k^n, \quad \text{where } g_k = e^{-iak\Delta t}$$

Let us define  $\phi_k = k\Delta x$  and  $\lambda = \frac{\Delta t}{\Delta x}$

$\phi_k$  is the  $k$ -harmonic phase.

1. Show that  $\forall n \geq 0$ , we have

$$u_j^n = \sum_k = \alpha_k e^{ikjh} \gamma_k^n$$

where  $\gamma_k = 1 - \frac{a\Delta t}{\Delta x} i \sin(k\Delta x)$  is the amplification coefficient for the  $k$ -harmonic

**Note:** The coefficient  $\gamma_k$  is the analogue of  $g_k$  for the numerical scheme.

Notice that  $|g_k| = 1$  while  $|\gamma_k| \leq 1$ .

The amplification error for the  $k$ -harmonic is defined by  $\epsilon_a(k) = \frac{|\gamma_k|}{|g_k|}$

2. plot the amplification error for the Euler centered explicit scheme

Let us define the propagation velocity  $\frac{\omega}{k}$  as

$$\gamma_k = |\gamma_k| e^{-i\omega\Delta t} = |\gamma_k| e^{-i\frac{\omega}{k}\lambda\phi_k}$$

The dispersion error is defined by

$$\epsilon_d(k) = \frac{\omega}{ka} = \frac{\omega\Delta x}{\phi_k a}$$

3. Plot the dispersion error for the Euler centered explicit scheme
4. Do the same study for the following schemes

### 1.8.3 Lax-Friedrichs

$$u_j^{n+1} = \frac{1}{2}(u_{j+1}^n + u_{j-1}^n) - \frac{\lambda}{2}(u_{j+1}^n - u_{j-1}^n)$$

can also be defined using the numerical flux

$$h_{j+\frac{1}{2}}^n = \frac{1}{2}\{a(u_{j+1}^n + u_j^n) - \lambda^{-1}(u_{j+1}^n - u_j^n)\}$$

### 1.8.4 Lax-Wendroff

$$h_{j+\frac{1}{2}}^n = \frac{1}{2}\{a(u_{j+1}^n + u_j^n) - \lambda a^2(u_{j+1}^n - u_j^n)\}$$

### 1.8.5 Decentered Explicit Euler

$$h_{j+\frac{1}{2}}^n = \frac{1}{2}\{a(u_{j+1}^n + u_j^n) - |a|(u_{j+1}^n - u_j^n)\}$$



## LESSON 2

We consider the finite element discretization of the following problem

$$\begin{aligned} -\nabla \cdot (A\nabla u) + (\mathbf{v} - \mathbf{w}) \cdot \nabla u + bu &= f, & \Omega \\ u &= g, & \Gamma_D \\ A\nabla u \cdot \mathbf{n} &= k, & \Gamma_N \end{aligned}$$

which can be written in a weak formulation, for any test function  $\phi$  that vanishes on  $\Gamma_D$ ,

$$\begin{aligned} \int_{\Omega} A\nabla u \cdot \nabla \phi \, d\Omega + \int_{\Omega} (\mathbf{v} \cdot \nabla u) \phi \, d\Omega + \int_{\Omega} u \mathbf{w} \cdot \nabla \phi \, d\Omega + \int_{\Omega} (b + \nabla \cdot \mathbf{w}) u \phi \, d\Omega = \\ \int_{\Omega} f \phi \, d\Omega + \int_{\Gamma_N} k \phi \, d\Gamma_N + \int_{\Gamma_D} u \mathbf{w} \cdot \mathbf{n} \phi \, d\Gamma_D \end{aligned}$$

It is assumed that the domain  $\Omega$  has a smooth boundary  $\partial\Omega$  such that

$$\begin{aligned} \Gamma_D \cup \Gamma_N &= \partial\Omega \\ \Gamma_D \cap \Gamma_N &= \emptyset \end{aligned}$$

We also assume the existence of a mapping  $\mathbf{F}$  such that  $\Omega = \mathbf{F}(\mathcal{P})$  where  $\mathcal{P}$  is the unit line/square.  $\mathcal{P}$  is called a **patch, logical domain or reference element**.

*for the moment, we will focus on the simple case where : math :  $\mathbf{F}$  is the identity mapping.*

The weak formulation can be written as

$$a(u, \phi) = l(\phi)$$

where

$$a(u, \phi) = \int_{\Omega} \{A\nabla u \cdot \nabla \phi + (\mathbf{v} \cdot \nabla u) \phi + u \mathbf{w} \cdot \nabla \phi + (b + \nabla \cdot \mathbf{w}) u \phi\} \, d\Omega$$

and

$$l(\phi) = \int_{\Omega} f \phi \, d\Omega + \int_{\Gamma_N} k \phi \, d\Gamma_N + \int_{\Gamma_D} g \mathbf{w} \cdot \mathbf{n} \phi \, d\Gamma_D$$

Let us denote  $\mathcal{D}$  the following bilinear differential operator

$$\mathcal{D}(u, \phi) = A\nabla u \cdot \nabla \phi + (\mathbf{v} \cdot \nabla u) \phi + u \mathbf{w} \cdot \nabla \phi + (b + \nabla \cdot \mathbf{w}) u \phi$$

## 2.1 Galerkin-Ritz approximation

Let us consider the following sets:

$$\begin{aligned}\mathcal{X} &= H^1(\Omega) \\ \mathcal{S} &= \{v/v \in \mathcal{X}, v|_{\Gamma_D} = g\} \\ \mathcal{V} &= \{v/v \in \mathcal{X}, v|_{\Gamma_D} = 0\}\end{aligned}$$

Let  $\mathcal{X}_h, \mathcal{V}_h$  be finite dimensional subspaces of  $\mathcal{X}, \mathcal{V}$ , where  $h$  is a parameter intended to tend to zero. Let  $\mathcal{S}_h$  be a finite dimensional approximation of  $\mathcal{S}$ . Remark that the last set is not necessarily a vector space.

The discrete Galerkin formulation writes:

Find  $u_h = u_h^0 + g_h \in \mathcal{S}_h$  such that:

$$a(u_h, \phi_h) = l(\phi_h), \quad \forall \phi_h \in \mathcal{V}_h$$

where here  $g_h, k$  are given functions, with  $g_h \in \mathcal{S}_h$ .

We suppose a rearrangement of the basis functions such that  $\varphi_i|_{\Gamma_D} = 0, \forall i \in \{1, \dots, n - n_D\}$ . This helps us to construct a basis of  $\mathcal{V}_h$  s.t :

$$\forall \phi_h \in \mathcal{V}_h, \phi_h = \sum_{i=1}^{n-n_D} [\phi_h]^i \varphi_i$$

$g^h \in \mathcal{S}_h$  is such that  $[g^h]^i = 0, \forall i \in \{1, \dots, n - n_D\}$  and so,  $g^h = \sum_{i=n-n_D+1}^n [g^h]^i \varphi_i$  Therefore,  $u_h \in \mathcal{S}_h$  writes

$$u_h = \sum_{i=1}^{n-n_D} [u_h]^i \varphi_i + \sum_{i=n-n_D+1}^n [g^h]^i \varphi_i .$$

---

**Note:** From now on, we will drop the subscript  $h$ .

---

therefore

$$\sum_{j=1}^{n-n_D} [u]^j a(\varphi_i, \varphi_j) = l(\varphi_i) - \sum_{m=n-n_D+1}^n [g]^m a(\varphi_i, \varphi_m), \quad \forall i \in \{1, \dots, n - n_D\}$$

For  $i, j \in \{1, \dots, n - n_D\}$ , we denote  $\Sigma_{i,j} = a(\varphi_i, \varphi_j)$  and  $L_i = l(\varphi_i) - a(\varphi_i, g_h)$

therefore we get:

$$\sum_{j=1}^{n-n_D} \Sigma_{i,j} [u]^j = L_i, \quad \forall i \in \{1, \dots, n - n_D\}$$

Finally, let us introduce the matrix:

$$\Sigma = (\Sigma_{i,j})_{1 \leq i, j \leq n-n_D}$$

and the vectors :

$$L = (L_i)_{1 \leq i \leq n-n_D}^T$$

$$[u] = ([u]^i)_{1 \leq i \leq n-n_D}^T$$

this leads to the linear system

$$\Sigma [u] = L .$$



## 2.2 Assembling process

We have seen that

$$\sum_{j=1}^{n-n_D} \Sigma_{i,j} [u]^j = L_i, \quad \forall i \in \{1, \dots, n - n_D\}$$

$$\begin{aligned} \Sigma_{i,j} &= a(\varphi_i, \varphi_j) = \int_{\Omega} \mathcal{D}(\varphi_i, \varphi_j) d\Omega \\ &= \sum_{e \in \mathcal{E}} \int_{Q_e} \mathcal{D}(\varphi_i, \varphi_j) d\Omega \\ &= \sum_{e \in \mathcal{E}} a_e(\varphi_i, \varphi_j) \end{aligned}$$

because  $\{Q_e, e \in \mathcal{E}\}$  form a partition of the physical domain  $\Omega$ , where  $\mathcal{E}$  is the set of all elements

The next step is to compute each of the element contribution  $a_e$

### 2.2.1 Element contribution

Element contribution relies on computing integrals over a given element. As our basis functions (and all others functions are smooth) we can use a quadrature formulae.

---

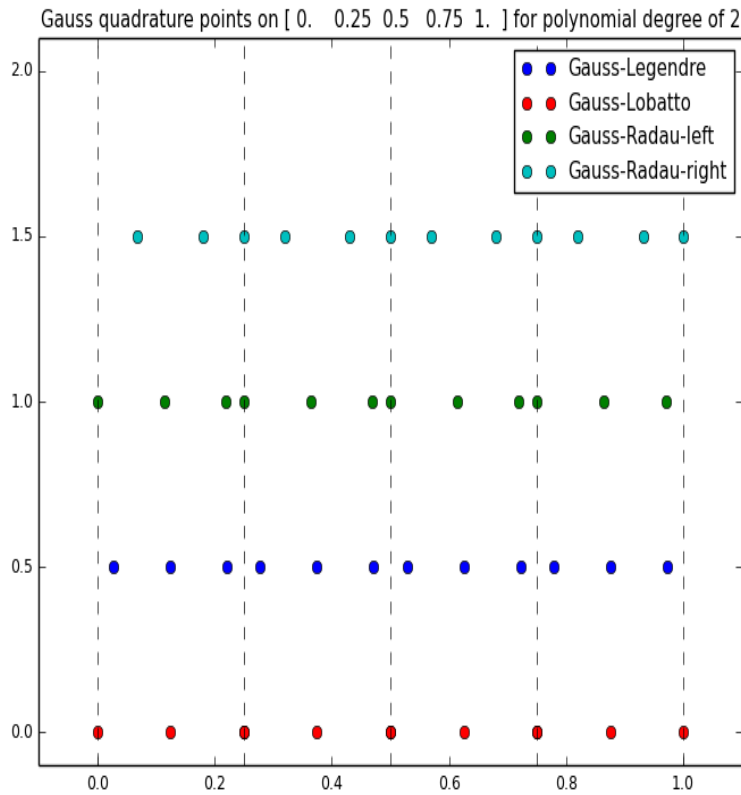
**Note:** The choice of a quadrature formulae depends on the basis functions we want to integrate. Generally, when their derivatives presents a discontinuity, a natural choice is the use of the Gauss-Legendre points.

---

#### Quadrature formulae in 1D

You can find, here `the complete script` that generates quadratures points on a given line.

The next plot shows the quadrature points formulae for a mesh  $\{x_0 = 0, x_1 = \frac{1}{4}, x_2 = \frac{1}{2}, x_3 = \frac{3}{4}, x_4 = 1\}$  for a cubic polynomial degree.



The following example shows how to use the module *quadratures*

```

1  # -*- coding: UTF-8 -*-
2  #! /usr/bin/python
3
4  import numpy as np
5  from quadratures import *
6  import matplotlib.pyplot as plt
7  import matplotlib
8
9  matplotlib.rcParams.update({'font.size': 9})
10
11  qd = quadratures()
12
13  # ... create the 1D mesh
14  n = 3
15  x = np.linspace(0., 1., n+2)
16  # ...
17
18  # ... polynomial degree
19  p = 2
20  # ...
21
22  # ... generate the quadrature points
23  #     and their corresponding weights
24  #     on the whole mesh
25  [x_leg, w_leg] = qd.generate(x, p, "legendre")

```

```

26 [x_lob,w_lob] = qd.generate(x, p, "lobatto")
27 [x_rdl,w_rdl] = qd.generate(x, p, "radau_left")
28 [x_rdr,w_rdr] = qd.generate(x, p, "radau_right")
29 # ...
30
31 # ...
32 x_leg = np.ravel(x_leg)
33 x_lob = np.ravel(x_lob)
34 x_rdl = np.ravel(x_rdl)
35 x_rdr = np.ravel(x_rdr)
36
37 y_lob = 0.*np.ones_like(x_lob)
38 y_leg = 0.5*np.ones_like(x_leg)
39 y_rdl = 1.*np.ones_like(x_rdl)
40 y_rdr = 1.5*np.ones_like(x_rdr)
41
42 plt.plot(x_leg, y_leg, 'ob', label="Gauss-Legendre")
43 plt.plot(x_lob, y_lob, 'or', label="Gauss-Lobatto")
44 plt.plot(x_rdl, y_rdl, 'og', label="Gauss-Radau-left")
45 plt.plot(x_rdr, y_rdr, 'oc', label="Gauss-Radau-right")
46
47 for xi in x:
48     plt.axvline(x=xi, ymin=0., ymax = 1., linewidth=0.5, linestyle='--', color='k')
49
50 plt.legend()
51
52 plt.xlim(-0.1, 1.1)
53 plt.ylim(-0.1, 2.1)
54
55 plt.title("Gauss quadrature points on " + str(x) + " for polynomial degree of " + str(p) )
56
57 plt.savefig("ex4_quadrature_1d.png")
58 # ...

```

Note the shape of the arrays `x_leg` and `w_leg`

```

>>> x_leg.shape
(4, 3)
>>> w_leg.shape
(4, 3)

```

In fact the first dimension of `x_leg` and `w_leg` refers to the element, while the second one refers to the quadrature point.

## Quadrature formulae in 2D

The following script shows how to construct a 2D grid by calling two time the *generate* method and using the *numpy-meshgrid* function.

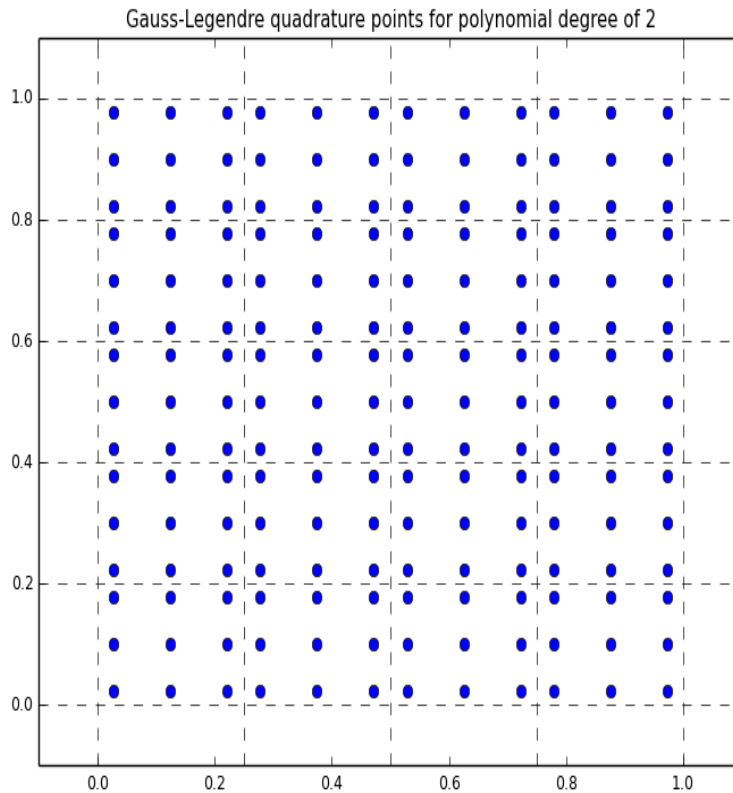
```

1 # -*- coding: UTF-8 -*-
2 #! /usr/bin/python
3
4 import numpy as np
5 from quadratures import *
6 import matplotlib.pyplot as plt
7 import matplotlib
8
9 matplotlib.rcParams.update({'font.size': 9})

```

```
10
11 qd = quadratures()
12
13 # ... create the 1D meshes
14 nx = 3 ; ny = 4
15 x = np.linspace(0.,1., nx+2)
16 y = np.linspace(0.,1., ny+2)
17 # ...
18
19 # ... polynomial degree
20 p = 2
21 # ...
22
23 # ... generate the quadrature points
24 #     and their corresponding weights
25 #     on the whole mesh
26 [x_leg,wx_leg] = qd.generate(x, p, "legendre")
27 [y_leg,wy_leg] = qd.generate(y, p, "legendre")
28 # ...
29
30 # ...
31 x_leg = np.ravel(x_leg)
32 y_leg = np.ravel(y_leg)
33
34 X,Y = np.meshgrid(x_leg, y_leg)
35
36 plt.plot(X, Y, 'ob')
37
38 for xi in x:
39     plt.axvline(x=xi, ymin=0., ymax = 1., linewidth=0.5, linestyle='--', color='k')
40 for yi in y:
41     plt.axhline(y=yi, xmin=0., xmax = 1., linewidth=0.5, linestyle='--', color='k')
42
43 plt.xlim(-0.1, 1.1)
44 plt.ylim(-0.1, 1.1)
45
46 plt.title("Gauss-Legendre quadrature points for polynomial degree of " + str(p) )
47
48 plt.savefig("ex4_quadrature_2d.png")
49 # ...
```

The resulting grid is shown in the next figure



## 2.3 Implementing the Finite Element Method

You can find the `compressed project directory` [here](#). This project consists of the following files

1. **quadratures.py** includes different quadrature formulae, as described in the previous section.
2. **grid.py** defines *1D* and *2D* grids.
3. **connectivity.py** defines *1D* and *2D* connectivities depending on the number of elements, polynomial degrees and regularity.

---

### Todo

Initialize the arrays `ID`, `LM`, `IEN` depending on the number of elements, polynomial degree and regularity. You can start by considering a periodic or homogeneous Dirichlet boundary condition

4. **basis.py** defines *1D* and *2D* basis functions. Bernstein polynomials are provided.

---

### Todo

add B-Splines using Bezier extraction matrices. This will be addressed in the next lesson.

5. **space.py** defines the vectorial space  $\mathcal{V}_h$ .

---

**Todo**

to finish

---

6. **matrix.py** defines the discretization matrix  $\Sigma$ .

---

**Todo**

to finish

---

7. **field.py** defines both the right-hand-side and the unknown.

---

**Todo**

to finish

---

8. **pde.py** this is the main class of our project. A PDE object contains the assembly procedure. You must provide the *element\_contribution* as an argument.

---

**Todo**

to finish

---

# INDICES AND TABLES

- *genindex*
- *modindex*
- *search*