python_lessons Documentation

Release alpha

Ahmed RATNANI

March 12, 2015

CONTENTS

Contents:

CHAPTER

ONE

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 wan 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.recarray
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.dsplit                  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
```

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

```
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 or 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 substract(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

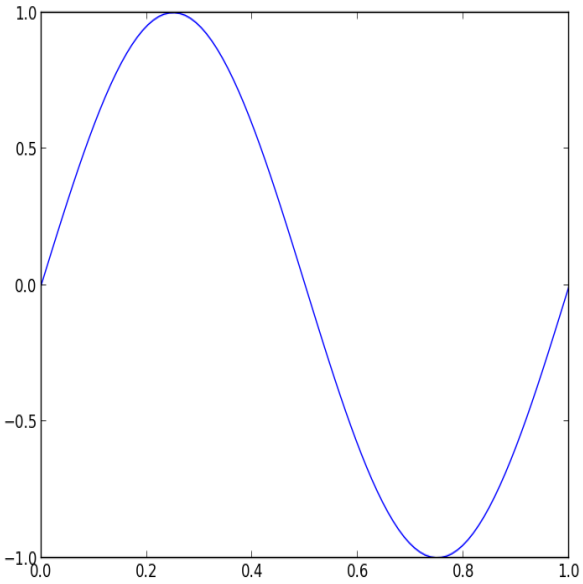
$$f(x) = sin(2\pi x), \forall x \in [0, 1]$$

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

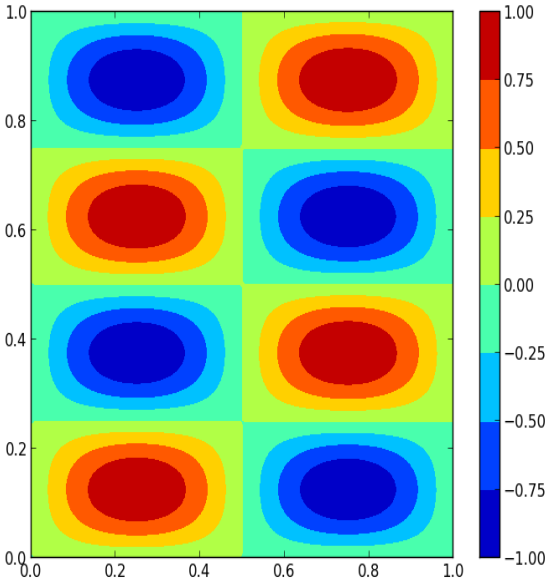
$$f(x, y) = sin(2\pi x)sin(4\pi y), \forall x, y \in [0, 1]$$

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)
```

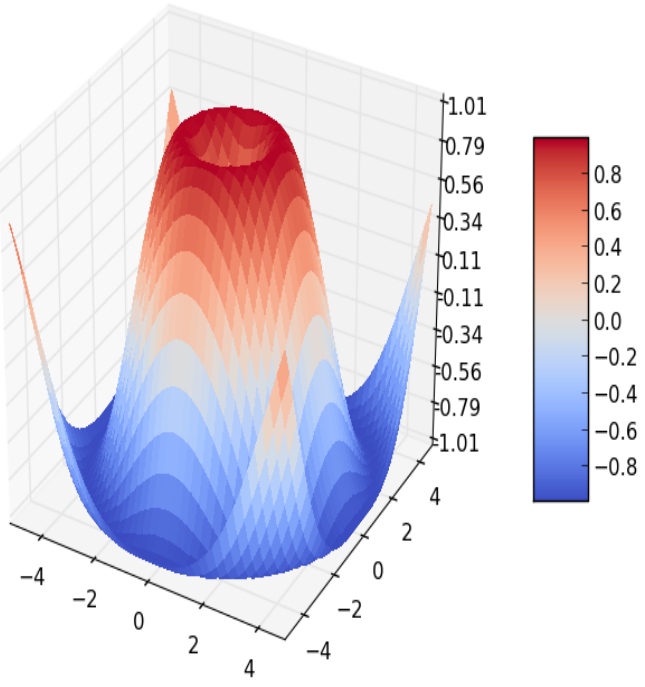
The next example shows how to plot f

```
>>> 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

```
#!/usr/bin/python

from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter
import matplotlib.pyplot as plt
import numpy as np

fig = plt.figure()
ax = fig.gca(projection='3d')
X = np.arange(-5, 5, 0.25)
Y = np.arange(-5, 5, 0.25)
X, Y = np.meshgrid(X, Y)
R = np.sqrt(X**2 + Y**2)
Z = np.sin(R)
surf = ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap=cm.coolwarm,
        linewidth=0, antialiased=False)
ax.set_zlim(-1.01, 1.01)

ax.zaxis.set_major_locator(LinearLocator(10))
ax.zaxis.set_major_formatter(FormatStrFormatter('%.02f'))

fig.colorbar(surf, shrink=0.5, aspect=5)

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,

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

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

$$x_i = i\Delta x, \qquad \forall i \in \{0, ..., N\}$$

1.8.1 Upwind finite difference matrix

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

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

The θ-time scheme can be written as

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

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

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

The finite difference matrix is therefor

$$\begin{aligned} A^n_{i,i} &= \frac{a^n_i}{\Delta x} \\ A^n_{i,i-1} &= -\frac{a^n_{i-1}}{\Delta x} \\ A^n_{i,j} = 0, \quad &\text{if} \quad j \notin \{i-1, i\} \end{aligned}$$

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

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

and the finite difference matrix is

$$\begin{aligned} A^n_{i,i} &= \frac{a^n_i}{\Delta x} \\ A^n_{i,i-1} &= -\frac{a^n_{i-1}}{\Delta x} \\ A^n_{1,0} &:= A^n_{1,N} \\ A^n_{i,j} = 0, \quad &\text{if} \quad j \notin \{i-1, i\} \end{aligned}$$

1. write the python script for these problems
2. study the stability of the θ -scheme
3. study the consistence of the θ -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

$$u^{n+1}_j = u^n_j - \frac{\lambda a}{2}(u^n_{j+1} - u^n_{j-1})$$

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

In the case of an Euler center-explicit scheme, the numerical flux is given by $h^n_{j+\frac{1}{2}} = \frac{a}{2}(u^n_{j+1} + u^n_j)$

Let us consider the exact solution

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

We have

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

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

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

---

Note: The coefficient γ_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 thee 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

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

CHAPTER

TWO

LESSON 2

We consider the finite element discretization of the following problem

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

which can be written in a weak formulation, for any test function ϕ that vanishes on Γ_D,

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

It is assumed that the domain Ω 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:

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

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:*

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

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 $\varphi_i|_{\Gamma_D} = 0, \forall i \in \{1, .., n - n_D\}$. 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 $[g_h]^i = 0, \forall i \in \{1, .., n - n_D\}$ 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

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

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

therefore we get:

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

Finally, let us introduce the matrix:

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

and the vectors :

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

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

this leads to the linear system

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

2.2 Assembling process

We have seen that

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

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

because $\{Q_e, e \in \mathcal{E}\}$ form a partition of the physical domain Ω, 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

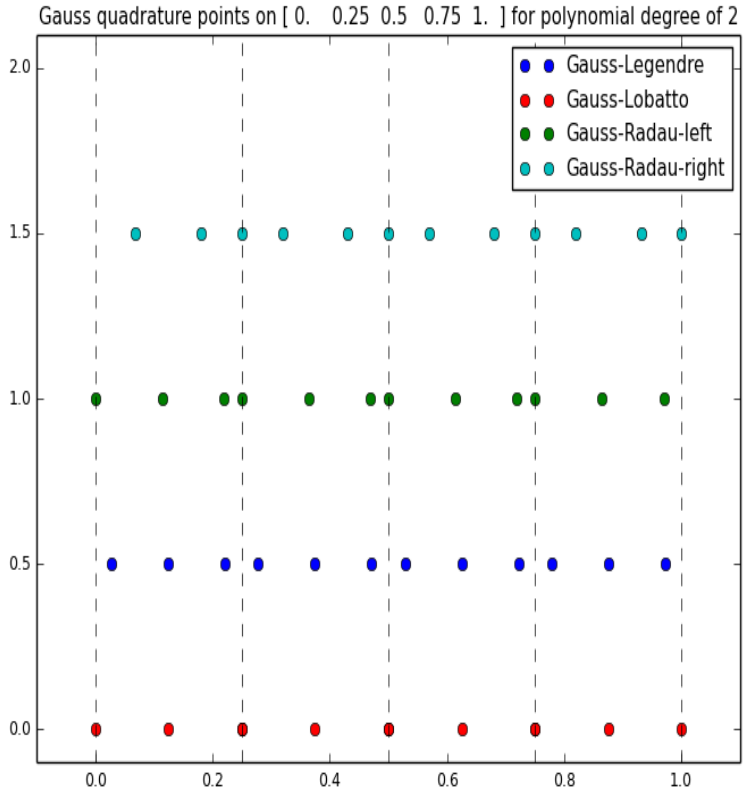
Element contribution relies on computing integrals over a given element. As our basis functions (and all others functions are smooth) we can use a quadrature formulae.

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 qudrature 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*

```
# -*- coding: UTF-8 -*-
#! /usr/bin/python

import numpy as np
from quadratures import *
import matplotlib.pyplot as plt
import matplotlib

matplotlib.rcParams.update({'font.size': 9})

qd = quadratures()

# ... create the 1D mesh
n = 3
x = np.linspace(0.,1., n+2)
# ...

# ... polynomial degree
p = 2
# ...

# ... generate the quadrature points
#     and their corresponding weights
#     on the whole mesh
[x_leg,w_leg] = qd.generate(x, p, "legendre")
```

```
[x_lob,w_lob] = qd.generate(x, p, "lobatto")
[x_rdl,w_rdl] = qd.generate(x, p, "radau_left")
[x_rdr,w_rdr] = qd.generate(x, p, "radau_right")
# ...

# ...
x_leg = np.ravel(x_leg)
x_lob = np.ravel(x_lob)
x_rdl = np.ravel(x_rdl)
x_rdr = np.ravel(x_rdr)

y_lob = 0.*np.ones_like(x_lob)
y_leg = 0.5*np.ones_like(x_leg)
y_rdl = 1.*np.ones_like(x_rdl)
y_rdr = 1.5*np.ones_like(x_rdr)

plt.plot(x_leg, y_leg, 'ob', label="Gauss-Legendre")
plt.plot(x_lob, y_lob, 'or', label="Gauss-Lobatto")
plt.plot(x_rdl, y_rdl, 'og', label="Gauss-Radau-left")
plt.plot(x_rdr, y_rdr, 'oc', label="Gauss-Radau-right")

for xi in x:
    plt.axvline(x=xi, ymin=0., ymax = 1., linewidth=0.5, linestyle='--', color='k')

plt.legend()

plt.xlim(-0.1, 1.1)
plt.ylim(-0.1, 2.1)

plt.title("Gauss quadrature points on " + str(x) + " for polynomial degree of " + str(p) )

plt.savefig("ex4_quadrature_1d.png")
# ...
```

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** refeers to the element, while the second one refeers to the quadrature point.

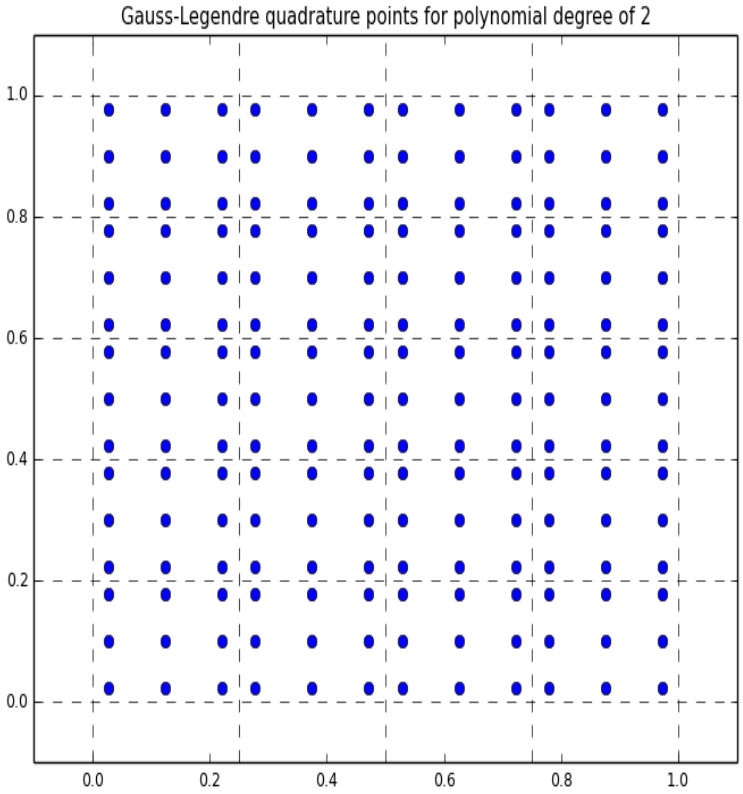
Quadrature formulae in 2D

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

```
# -*- coding: UTF-8 -*-
#! /usr/bin/python

import numpy as np
from quadratures import *
import matplotlib.pyplot as plt
import matplotlib

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 adressed in the next lesson.

---

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

Todo

to finish

6. **matrix.py** defines the discretization matrix Σ.

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

CHAPTER

THREE

LESSON 3

3.1 Splines in CAD

3.1.1 Modeling a curve

In the sequel, we will explain how we can model curves, beginning from the simplest one (the p-form) to textit{B-splines, NURBS} curves. We will explain the advantages of each method. Most of the results presented in this section were taken from [Piegl_Book1997]. We will denote by $\mathcal{C}(\mathbf{x}(t))$ a parametric curve, where each point is defined by the value of the parameter t.

The p-form

The first way to model a curve would be to consider the following form :

$$\mathcal{C}(\mathbf{x}(t)) = \sum_{i=0}^{n} t^i \mathbf{P}_i \tag{3.1}$$

Let us denote $\mathbf{x}(t) = \begin{pmatrix} x(t) \\ y(t) \\ z(t) \end{pmatrix}$, and $\mathbf{P}_i = \begin{pmatrix} P_i^x \\ P_i^y \\ P_i^z \end{pmatrix}$. The relation ((3.1)) can be written,

$$\begin{cases} x(t) = \sum_{i=0}^{n} t^i P_i^x \\ y(t) = \sum_{i=0}^{n} t^i P_i^y \\ z(t) = \sum_{i=0}^{n} t^i P_i^z \end{cases}$$

A simple computation leads to $\mathbf{P}_i = \frac{1}{i!}\frac{d^{(i)}}{dt}\mathcal{C}(\mathbf{x}(t))|_{t=0}$, for each $i \in \{0, \cdots, n\}$. Even if the p-form is a natural description for curves, it presents some disadvantages :

- the curve is not necessary regular everywhere. So a regular description of the curve, may lead to non-efficient approximation,
- the points $(\mathbf{P}_i)_{0 \leq i \leq n}$ do not have any geometric interpretation,
- numerical evaluation of such description, needs the use of Horner algorithm, which is unstable.

The Bezier-form

Rather than use $\{1, t, \cdots, t^n\}$ as a basis of $\Pi_{<n+1}$, we can take Bernstein polynomials; this leads to the Bezier-form. Therefore, it is equivalent to the p-form and writes :

$$\mathcal{C}(\mathbf{x}(t)) = \sum_{i=0}^{n} B_i^n(t)\mathbf{P}_i, \quad \text{for each } 0 \leq t \leq 1 \tag{3.2}$$

where B_i^n denote Bernstein polynomials :

$$B_i^n(t) = \begin{pmatrix} n \\ i \end{pmatrix} t^i(1-t)^{n-i} = \frac{n!}{i!(n-i)!} t^i(1-t)^{n-i}, \quad \text{for each } 0 \leq t \leq 1$$

The sequence $(\mathbf{P}_i)_{0 \leq i \leq n}$ is called control points.

Examples

- n = 1 :

$\mathcal{C}(\mathbf{x}(t)) = (1-t)\mathbf{P}_0 + t\mathbf{P}_1$, for each :math:‘ 0 leq t leq 1‘.

This is just a description of a segment (figure ref{bezier_curve_p1}).

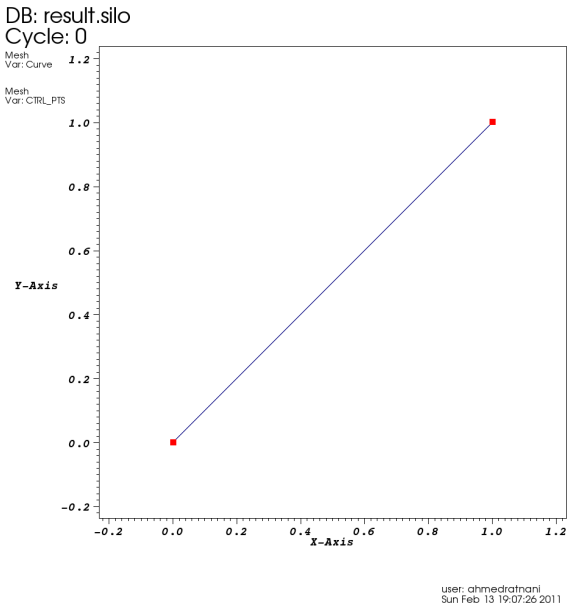


Figure 3.1: A Bezier-curve of degree 1 and its control points

- n = 2 :

$\mathcal{C}(\mathbf{x}(t)) = (1-t)^2\mathbf{P}_0 + 2t(1-t)\mathbf{P}_1 + t^2\mathbf{P}_2$, for each $\leq t \leq 1$.

This forms a parabolic arc (figure ref{bezier_curve_p2}). We have the following properties:

1. $\{\mathbf{P}_0, \mathbf{P}_1, \mathbf{P}_2\}$ form a polygon: this is the control polygon,
2. $\mathbf{P}_0 = \mathcal{C}(\mathbf{x}(0))$, and $\mathbf{P}_2 = \mathcal{C}(\mathbf{x}(1))$,

3. $\mathcal{C}'(\mathbf{x}(0))$ is parallel to $\mathbf{P}_1 - \mathbf{P}_0$, and $\mathcal{C}'(\mathbf{x}(1))$ is parallel to $\mathbf{P}_2 - \mathbf{P}_1$,
4. the curve is contained in the triangle $\mathbf{P}_0\mathbf{P}_1\mathbf{P}_2$,
5. the control points textit{approach the behavior} of the curve.

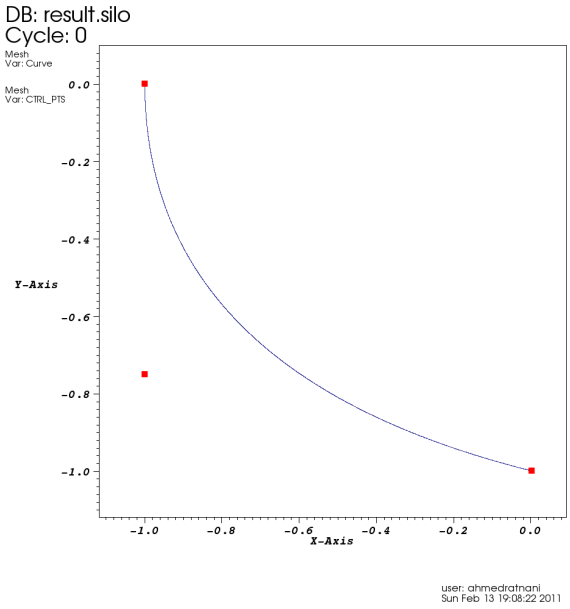


Figure 3.2: A Bezier-curve of degree 2 and its control points

- n = 3 :

$\mathcal{C}(\mathbf{x}(t)) = (1-t)^3\mathbf{P}_0 + 3t^2(1-t)\mathbf{P}_1 + 3t^2(1-t)\mathbf{P}_2 + t^3\mathbf{P}_3$, for each $0 \leq t \leq 1$. We give an example of such a curve (figure ref{bezier_curve_p3})

We have the following properties:

1. $\{\mathbf{P}_0, \mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3\}$ form a polygon: this is the control polygon,
2. $\mathbf{P}_0 = \mathcal{C}(\mathbf{x}(0))$, and $\mathbf{P}_3 = \mathcal{C}(\mathbf{x}(1))$,
3. $\mathcal{C}'(\mathbf{x}(0))$ is parallel to $\mathbf{P}_1 - \mathbf{P}_0$, and $\mathcal{C}'(\mathbf{x}(1))$ is parallel to $\mathbf{P}_3 - \mathbf{P}_2$,
4. the control points textit{approach the behavior} of the curve.
5. convex-hull : the curve is contained in the convex-hull associated to the control points,
6. variation diminution : there is no line that intersects the control polygon in more points than the curve,
7. it associates an implicit direction to the curve.

Bezier-curves properties

We summarize the properties revealed in the previous examples:

- Invariance under some transformations : rotation, translation, scaling; it is sufficient to transform the control points,
- $B_i^n(t) \geq 0, \forall\ 0 \leq t \leq 1$
- partition of unity : $\sum_{i=0}^{n} B_i^n(t) = 1, \forall\ 0 \leq t \leq 1$

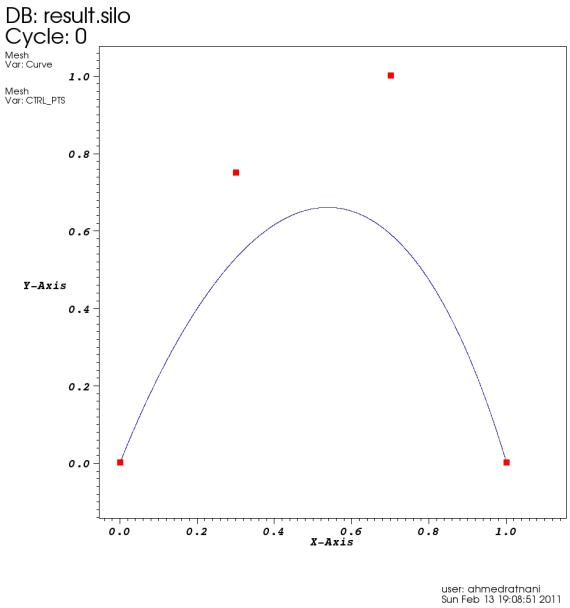


Figure 3.3: A Bezier-curve of degree 3 and its control points.

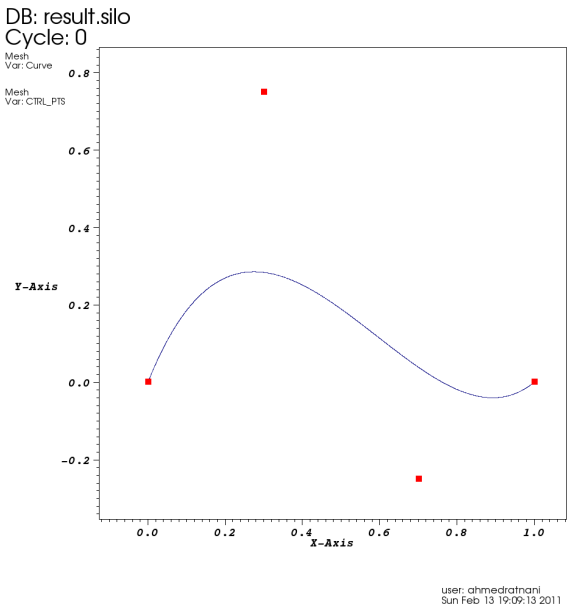


Figure 3.4: A Bezier-curve of degree 3 and its control points after moving one control point.

- $B_0^n(0) = B_n^n(1) = 1$
- each B_i^n has exactly one maximum in $[0, 1]$, at $\frac{i}{n}$
- B_i^n are symmetric with respect to $\frac{1}{2}$
- recursive property : $B_i^n(t) = (1-t)B_i^{n-1}(t) + tB_{i-1}^{n-1}(t)$, and $B_i^n(t) = 0, \text{ if } i < 0 \text{ or, } i > n$
- derivation property: $B_i^{n\prime}(t) = n\{B_{i-1}^{n-1}(t) - B_i^{n-1}(t)\}$, and $B_{-1}^{n-1}(t) \equiv B_n^{n-1}(t) \equiv 0$
- deriving a curve : $\mathcal{C}'(t) = n\{\sum_{i=0}^{n-1} B_i^{n-1}(t)\,(\mathbf{P}_{i+1} - \mathbf{P}_i)\}$

hence, we have :

$$\mathcal{C}'(0) = n\,(\mathbf{P}_1 - \mathbf{P}_0) \quad \mathcal{C}'(1) = n\,(\mathbf{P}_n - \mathbf{P}_{n-1})$$

and,

$$\mathcal{C}''(0) = n(n-1)\,(\mathbf{P}_0 - 2\mathbf{P}_1 + \mathbf{P}_2) \quad \mathcal{C}''(1) = n\,(\mathbf{P}_n - 2\mathbf{P}_{n-1} + \mathbf{P}_{n-2})$$

this shows the interest of Bezier curves.

- DeCasteljau algorithm:

$$\mathcal{C}^n(t; \mathbf{P}_0, \cdots, \mathbf{P}_n) = (1-t)\mathcal{C}^{n-1}(t; \mathbf{P}_0, \cdots, \mathbf{P}_{n-1}) + t\mathcal{C}^{n-1}(t; \mathbf{P}_1, \cdots, \mathbf{P}_n)$$

The spline-form

Even with the advantages of the Bezier-form, still the problem of the regularity of the curve. We need to use a piecewise-polynomial form. For this purpose, we use textit{B-splines}. As seen before, they form a basis for the Schoenberg space.

Let $(\mathbf{P}_i)_{1 \le i \le N} \in \mathbf{R}^d$ be a sequence of control points, forming a control polygon.

B-Spline curve

The B-spline curve in $\mathbf{R}^d$ associated to $T = (t_i)_{1 \le i \le N+k}$ and $(\mathbf{P}_i)_{1 \le i \le N}$ is defined by : $\mathcal{C}(t) = \sum_{i=1}^{N} N_i^k(t)\mathbf{P}_i$

We have the following properties for a textit{B-spline} curve:

- If $N = k$, then $\mathcal{C}$ is just a B'ezier-curve,
- $\mathcal{C}$ is a piecewise polynomial curve,
- The curve interpolates its extremas if the associated multiplicity of the first and the last knot are maximum (textit{i.e.} equal to k),
- Invariance with respect to affine transformations,
- strong convex-hull property:

if $t_i \le t \le t_{i+1}$, then $\mathcal{C}(t)$ is inside the convex-hull associated to the control points $\mathbf{P}_{i-p}, \cdots, \mathbf{P}_i$,

- local modification : moving $\mathbf{P}_i$ affects $\mathcal{C}(t)$, only in the interval $[t_i, t_{i+k}]$,
- the control polygon approaches the behavior of the curve

Note: we can use multiple control points : $\mathbf{P}_i = \mathbf{P}_{i+1}$.

CHAPTER

FOUR

INDICES AND TABLES

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