NURBS-Python Documentation

Release

Onur Rauf Bingol

Nov 10, 2017
# Contents

1 Introduction
   1.1 Q&A .................................................. 1

2 Graphical Outputs
   2.1 Curves ................................................ 4
   2.2 Surfaces ............................................. 7

3 Submodules
   3.1 nurbs.Curve module ...................................... 11
   3.2 nurbs.Surface module .................................... 15
   3.3 nurbs.utilities module .................................. 19
   3.4 nurbs.Grid module ...................................... 20

Python Module Index ........................................... 23
This package contains native Python implementations of several The NURBS Book algorithms for generating B-spline / NURBS curves and surfaces. It also provides a data structure for storing elements required for evaluation these curves and surfaces. Please follow the README.md file included in the repository for details on the algorithms.

Some other advantages of this package are:

- Python 2.x and 3.x compatibility
- No external dependencies (such as NumPy)
- Uses Python properties for the data storage access
- A utilities module for auto-generating and normalizing knot vectors
- A Grid class for generating various types of control points grids

The NURBS-Python package follows an object-oriented design as much as possible. However, in order to understand the algorithms, you might need to take a look at The NURBS Book itself.

1.1 Q&A

What is NURBS?

NURBS, namely Non-Uniform Rational Basis Spline, is a mathematical model for generation of curves and surfaces in a flexible way. It is a well-accepted industry standard and used as a basis for nearly all of the 3D modeling and CAD/CAM software packages. Please see the related Wikipedia article or The NURBS Book, a very nice and informative book written by Dr. Piegl and Dr. Tiller.

What is the purpose of this package/library?

Very simple: Implementing the well-known NURBS algorithms in native Python, i.e. without using any converters or wrappers, like SWIG or Boost.Python. This approach comes with some advantages in debugging and implementing new algorithms.

Current version of the library doesn’t require any additional packages, such as NumPy, so that you can run NURBS-Python on a plain Python installation.
The first version of the library was very complicated to use (I developed that version as a class project), so I started developing an alternative, easy-to-use NURBS library with simple data storage functionality, and now, here we are!

**Can I request a new feature?**

Of course you can :-) Please feel free to contact me about the NURBS-Python package anytime you want.

- Github (you can find my email there)
- Twitter: @orbingol
The following 2D and 3D plots are generated using Matplotlib. You can find the scripts generating these graphical outputs in the NURBS-Python Repository.
2.1 Curves
2.1. Curves
2.2 Surfaces
2.2. Surfaces
The `nurbs` package contains `Curve` and `Surface` classes along with the `utilities` module for functions common in both `Curve` and `Surface` classes.

This package also includes a very simple grid generator class, `Grid`, to generate rectangular control point grids for use with the `Surface` class.

### 3.1 `nurbs.Curve` module

`Curve` class provides data storage properties and evaluation functions for NURBS and B-spline curves.

```python
class nurbs.Curve.Curve
    A data storage and evaluation class for B-Spline and NURBS curves.
```

**Data Storage**

`Curve` class implements Python properties using the `@property` decorator. The following properties are present in this class:

- degree
- knotvector
- delta
- ctrlpts
- ctrlptsw
- weights
- curvepts

The functions `read_ctrlpts()` and `read_ctrlptsw()` provide an easy way to read control points from a text file. Additional details for the text format can be found in `FORMATS.md` file.

**Evaluation**

The evaluation methods in the `Curve` class are:
• evaluate()
• evaluate_rational()
• derivatives()
• tangent()

Examples

Please see the examples in the repository named as ex_curveXX.py, where XX is the example number, for details on using the Curve class.

Note: If you update any of the data storage elements after the curve evaluation, the surface points stored in curvepts property will be deleted automatically.

ctrlpts

Control points

Control points of a Curve is stored as a list of (x, y) coordinates

Getter  Gets the control points
Setter  Sets the control points
Type  list

ctrlptsw

Weighted control points

Returns and accepts a tuple containing (x*w, y*w, w) values. The setter method automatically separates the weights vector and compute the unweighted control points.

Getter  Gets the weighted control points
Setter  Sets the weights vector and the control points
Type  list

curvepts

Evaluated curve points

Note: evaluate() or evaluate_rational() should be called first.

Getter  (x, y) coordinates of the evaluated surface points
Type  list

degree

Curve degree

Getter  Gets the curve degree
Setter  Sets the curve degree
Type  integer

delta

Curve evaluation delta
Note: The delta value is 0.01 by default.

**Getter** Gets the delta value

**Setter** Sets the delta value

**Type** float

**derivatives** *(u=-1, order=0)*

Evaluates n-th order curve derivatives at the given u using Algorithm A3.4.

**Parameters**
- **u** *(float)* – knot value
- **order** *(integer)* – derivative order

**Returns** A list containing up to {order}-th derivative of the curve

**Return type** list

**derivatives2** *(u=-1, order=0)*

Evaluates n-th order curve derivatives at the given u using Algorithm A3.2

**Parameters**
- **u** *(float)* – knot value
- **order** *(integer)* – derivative order

**Returns** A list containing up to {order}-th derivative of the curve

**Return type** list

**derivatives_ctrlpts** *(order=0, r1=0, r2=0)*

Computes the control points of all derivative curves up to and including the {degree}-th derivative.

Output is PK[k][i], i-th control point of the k-th derivative curve where 0 <= k <= degree and r1 <= i <= r2-k

**Parameters**
- **order** *(integer)* – derivative order
- **r1** *(integer)* – minimum span
- **r2** *(integer)* – maximum span

**Returns** PK, a 2D list of control points

**Return type** list

**evaluate**

Evaluates the B-Spline curve.

Note: The evaluated surface points are stored in `curvepts`.

**Returns** None
**evaluate_rational()**
Evaluates the NURBS curve.

**Note:** The evaluated surface points are stored in `curvepts`.

**Returns** None

**knotvector**
Knot vector

**Getter** Gets the knot vector

**Setter** Sets the knot vector

**Type** list

**read_ctrlpts(filename='')**
Reads control points from a text file and generates a weight vector composed of 1.0s.

**Note:** The format of the text files is described in `FORMATS.md` file.

**Parameters**
- **filename** (string) – input file name

**Returns** None

**read_ctrlptsw(filename='')**
Reads weighted control points from a text file.

**Note:** The format of the text files is described in `FORMATS.md` file.

**Parameters**
- **filename** (string) – input file name

**Returns** None

**tangent(u=-1)**
Evaluates the surface tangent at the given (u, v) parameter.

**Parameters**
- **u** (float) – knot value

**Returns** A list in the order of “surface point” and “derivative”

**Return type** list

**weights**
Weights vector

**Note:** `ctrlpts` property and `read_ctrlpts()` will automatically generate a weights vector of 1.0s in the size of control points array.

**Getter** Gets the weights vector

**Setter** Sets the weights vector

**Type** list
3.2 nurbs.Surface module

`Surface` class provides data storage properties and evaluation functions for NURBS and B-spline surfaces.

**class nurbs.Surface**

A data storage and evaluation class for B-Spline and NURBS surfaces.

**Data Storage**

`Surface` class implements Python properties using the `@property` decorator. The following properties are present in this class:

- `degree_u`
- `degree_v`
- `knotvector_u`
- `knotvector_v`
- `delta`
- `ctrlpts`
- `ctrlptsw`
- `ctrlpts2D`
- `weights`
- `surfpts`

The functions `read_ctrlpts()` and `read_ctrlptsw()` provide an easy way to read control points from a text file. Additional details for the text format can be found in `FORMATS.md` file.

**Evaluation**

The evaluation methods in the `Surface` class are:

- `evaluate()`
- `evaluate_rational()`
- `derivatives()`
- `tangent()`
- `normal()`

**Examples**

Please see the examples in the repository named as `ex_surfaceXX.py`, where `XX` is the example number, for details on using the `Surface` class.

**Note:** If you update any of the data storage elements after the surface evaluation, the surface points stored in `surfpts` property will be deleted automatically.

**ctrlpts**

Control points

Control points of a `Surface` is stored as a list of `(x, y, z)` coordinates

- **Getter** Gets the control points
- **Setter** Sets the control points
ctrlpts2d
Control points
2D control points in [u][v] format.

*Getter* Gets the control points

*Type* list

ctrlptsw
Weighted control points
This property is a tuple containing (x*w, y*w, z*w, w) values. The setter method automatically separates the weights vector from the input and computes the unweighted control points.

*Getter* Gets the weighted control points

*Setter* Sets the weights vector and the control points

*Type* list

degree_u
Surface degree for U direction

*Getter* Gets the surface degree for U direction

*Setter* Sets the surface degree for U direction

*Type* integer

degree_v
Surface degree for V direction

*Getter* Gets the surface degree V for V direction

*Setter* Sets the surface degree V for V direction

*Type* integer

delta
Surface evaluation delta

*Note:* The delta value is 0.01 by default.

*Getter* Gets the delta value

*Setter* Sets the delta value

*Type* float

derivatives \((u=-1, v=-1, order=0)\)
Evaluates n-th order surface derivatives at the given \((u,v)\) parameter.

- SKL[0][0] will be the surface point itself
- SKL[0][1] will be the 1st derivative w.r.t. \( v \)
- SKL[2][1] will be the 2nd derivative w.r.t. \( u \) and 1st derivative w.r.t. \( v \)

*Parameters*

- \( u (float) \) – parameter in the U direction
• \texttt{v(float)} – parameter in the V direction
• \texttt{order(integer)} – derivative order

\textbf{Returns} A list \texttt{SKL}, where \texttt{SKL[k][l]} is the derivative of the surface \texttt{S(u,v)} w.r.t. \texttt{u k times and v l times}

\textbf{Return type} list

evaluate()
Evaluates the B-Spline surface.

\textbf{Note:} The evaluated surface points are stored in \texttt{surfpts}.

\textbf{Returns} None

evaluate_rational()
Evaluates the NURBS surface.

\textbf{Note:} The evaluated surface points are stored in \texttt{surfpts}.

\textbf{Returns} None

\textbf{knotvector_u}
Knot vector for U direction

\textbf{Getter} Gets the knot vector for \texttt{U direction}
\textbf{Setter} Sets the knot vector for \texttt{U direction}
\textbf{Type} list

\textbf{knotvector_v}
Knot vector for V direction

\textbf{Getter} Gets the knot vector for \texttt{V direction}
\textbf{Setter} Sets the knot vector for \texttt{V direction}
\textbf{Type} list

\textbf{normal}(u=-1, v=-1, \texttt{normalized=True})
Evaluates the surface normal at the given (u, v) parameter.

\textbf{Parameters}
• \texttt{u(float)} – parameter in the \texttt{U direction}
• \texttt{v(float)} – parameter in the \texttt{V direction}
• \texttt{normalized(boolean)} – if True, the returned normal vector is an unit vector

\textbf{Returns} normal vector
\textbf{Return type} list

\textbf{read_ctrlpts}(filename='')
Reads control points from a text file and generates a weight vector composed of 1.0s.
**Parameters**

**filename** (*string*) – input file name

**Returns** None

**read_ctrlptsw** (*filename=''*)

Reads weighted control points from a text file.

**Note:** The format of the text files is described in FORMATS.md file.

**Parameters**

**filename** (*string*) – input file name

**Returns** None

**surfpts**

Evaluated surface points

**Note:** `evaluate()` or `evaluate_rational()` should be called first.

**Getter** (x, y, z) coordinates of the evaluated surface points

**Type** list

**tangent** (*u=-1, v=-1*)

Evaluates the surface tangent at the given (u, v) parameter.

**Parameters**

- **u** (*float*) – parameter in the U direction
- **v** (*float*) – parameter in the V direction

**Returns** A list in the order of “surface point”, “derivative w.r.t. u” and “derivative w.r.t. v”

**Return type** list

**transpose()**

Transposes the surface by swapping U and V directions.

**Returns** None

**weights**

Weights vector

**Note:** `ctrlpts` property and `read_ctrlpts()` will automatically generate a weights vector of 1.0s in the size of control points array.

**Getter** Gets the weights vector

**Setter** Sets the weights vector

**Type** list
3.3 nurbs.utilities module

Utilities module contains common helper functions for B-spline and NURBS curve and surface evaluation.

nurbs.utilities.frange(x, y, step)

An implementation of a range() function which works with decimals.

Reference to this implementation: http://stackoverflow.com/a/7267280

Parameters

• x (integer or float) – start value
• y (integer or float) – end value
• step (integer or float) – increment

Returns float

Return type generator

nurbs.utilities.knotvector_autogen(degree=0, num_ctrlpts=0)

Generates a uniformly-spaced knot vector using the degree and the number of control points.

Parameters

• degree (integer) – degree of the knot vector direction
• num_ctrlpts (integer) – number of control points on that direction

Returns knot vector

Return type list

nurbs.utilities.knotvector_normalize(knotvector=())

Normalizes the input knot vector between 0 and 1.

Parameters knotvector (tuple) – input knot vector

Returns normalized knot vector

Return type list

nurbs.utilities.vector_cross(vect1=(), vect2=())

Computes the cross-product of the input vectors.

Parameters

• vect1 (tuple) – input vector 1
• vect2 (tuple) – input vector 2

Returns result of the cross-product

Return type list

nurbs.utilities.vector_dot(vect1=(), vect2=())

Computes the dot-product of the input vectors.

Parameters

• vect1 (tuple) – input vector 1
• vect2 (tuple) – input vector 2

Returns result of the cross-product

Return type list
Generates a unit vector from the input.

**Parameters**
- `vect (tuple)` – vector to be normalized
**Returns**
the normalized vector (i.e. the unit vector)

**Return type**
list

### 3.4 nurbs.Grid module

**Grid** class provides an easy way to generate control point grids for use with **Surface** module.

This class is designed minimally just to enable more testing cases for the **Surface** module. It is not a fully-featured grid generator which can fit to any purpose, but as always, contributions are welcome!

**class nurbs.Grid.Grid(size_x, size_y)**

Simple 2D grid generator to generate control points grid input for nurbs.Surface module.

**Parameters**
- `size_x (integer or float)` – width of the 2D grid
- `size_y (integer or float)` – height of the 2D grid

**bumps (num_bumps=0, all_positive=False, bump_height=3)**
Generates random bumps (i.e. hills) on the 2D grid.

This method generates hills on the grid defined by the **num_bumps** parameter. The direction of the generated hills are chosen randomly by default, but this behavior can be controlled by **all_positive** parameter. It is also possible to control the z-value using **bump_height** parameter.

Please note that, not all grids can be modified to have **num_bumps** number of bumps. Therefore, this function uses a trial-and-error method to determine whether the bumps can be generateable or not. For instance:

```python
    testgrid = Grid(5, 10)  # generates a 5x10 rectangle
    testgrid.generate(2, 2)  # splits the rectangle into 4 pieces
    testgrid.bumps(100)     # impossible, it will return an error message
    testgrid.bumps(1)       # You will get a bump at the center of the generated grid
```

**Parameters**
- `num_bumps (integer)` – Number of bumps (i.e. hills) to be generated on the 2D grid
- `all_positive (boolean)` – Generate all bumps on the positive z direction
- `bump_height (integer or float)` – z-value of the generated bumps on the grid

**Returns**
None

**generate (num_u, num_v)**
Generates the 2D grid using the input division parameters.

**Parameters**
- `num_u (integer)` – number of divisions in x-direction
- `num_v (integer)` – number of divisions in y-direction

**Returns**
None
**grid()**
Returns the generated grid.

**Note:** The format of the control points grid is described in `FORMATS.md` file.

**Returns** 2D list of points ([x,y,z]) in [u][v] format

**rotate_x (angle=0)**
Rotates the grid about the x-axis.

- **Parameters** `angle` *(integer or float)* – angle of rotation about the x-axis
- **Returns** None

**rotate_y (angle=0)**
Rotates the grid about the y-axis.

- **Parameters** `angle` *(integer or float)* – angle of rotation about the y-axis
- **Returns** None

**rotate_z (angle=0)**
Rotates the grid about the z-axis.

- **Parameters** `angle` *(integer or float)* – angle of rotation about the z-axis
- **Returns** None

**save (file_name='grid.txt')**
Saves the generated grid to a text file.

**Note:** The format of the text files is described in `FORMATS.md` file.

- **Parameters** `file_name` *(string)* – File name to be saved
- **Returns** None

**translate (pt=(0.0, 0.0, 0.0))**
Translates the grid origin to the input position.

Grid origin is (0, 0, 0) at instantiation and always represents the bottom left corner of the 2D grid.

- **Parameters** `pt` *(list)* – new origin point
- **Returns** None

3.4. *nurbs.Grid module*
c
Curve (Unix, Windows), 11

g
Grid (Unix, Windows), 20

n
nurbs, 1
nurbs.Curve, 11
nurbs.Grid, 20
nurbs.Surface, 15
nurbs.utilities, 19

S
Surface (Unix, Windows), 15

U
utilities (Unix, Windows), 19
Index

B
bumps() (nurbs.Grid.Grid method), 20

C
ctrlpts (nurbs.Curve.Curve attribute), 12
ctrlpts (nurbs.Surface.Surface attribute), 15
ctrlpts2d (nurbs.Surface.Surface attribute), 16
ctrlptsw (nurbs.Curve.Curve attribute), 12
ctrlptsw (nurbs.Surface.Surface attribute), 16
Curve (class in nurbs.Curve), 11
Curve (module), 11
curvepts (nurbs.Curve.Curve attribute), 12

d
degree (nurbs.Curve.Curve attribute), 12
degree_u (nurbs.Surface.Surface attribute), 16
degree_v (nurbs.Surface.Surface attribute), 16
delta (nurbs.Curve.Curve attribute), 12
delta (nurbs.Surface.Surface attribute), 16
derivatives() (nurbs.Curve.Curve method), 13
derivatives() (nurbs.Surface.Surface method), 16
derivatives2() (nurbs.Curve.Curve method), 13
derivatives_ctrlpts() (nurbs.Curve.Curve method), 13

E
evaluate() (nurbs.Curve.Curve method), 13
evaluate() (nurbs.Surface.Surface method), 17
evaluate_rational() (nurbs.Curve.Curve method), 13
evaluate_rational() (nurbs.Surface.Surface method), 17

F
frange() (in module nurbs.utilities), 19

G
generate() (nurbs.Grid.Grid method), 20
Grid (class in nurbs.Grid), 20
Grid (module), 20
grid() (nurbs.Grid.Grid method), 20

K
knotvector (nurbs.Curve.Curve attribute), 14
knotvector_autogen() (in module nurbs.utilities), 19
knotvector_normalize() (in module nurbs.utilities), 19
d
knotvector_u (nurbs.Surface.Surface attribute), 17
d
knotvector_v (nurbs.Surface.Surface attribute), 17

N
normal() (nurbs.Surface.Surface method), 17
nurbs (module), 1
nurbs.Curve (module), 11
nurbs.Grid (module), 20
nurbs.Surface (module), 15
nurbs.utilities (module), 19

R
read_ctrlpts() (nurbs.Curve.Curve method), 14
read_ctrlpts() (nurbs.Surface.Surface method), 17
read_ctrlptsw() (nurbs.Curve.Curve method), 14
read_ctrlptsw() (nurbs.Surface.Surface method), 18
rotate_x() (nurbs.Grid.Grid method), 21
rotate_y() (nurbs.Grid.Grid method), 21
rotate_z() (nurbs.Grid.Grid method), 21

S
save() (nurbs.Grid.Grid method), 21
Surface (class in nurbs.Surface), 15
Surface (module), 15
surfpts (nurbs.Surface.Surface attribute), 18

T
tangent() (nurbs.Curve.Curve method), 14
tangent() (nurbs.Surface.Surface method), 18
translate() (nurbs.Grid.Grid method), 21
transpose() (nurbs.Surface.Surface method), 18

U
utilities (module), 19
V

vector_cross() (in module nurbs.utilities), 19
vector_dot() (in module nurbs.utilities), 19
vector_normalize() (in module nurbs.utilities), 19

W

weights (nurbs.Curve.Curve attribute), 14
weights (nurbs.Surface.Surface attribute), 18