Standardisation Guide
`rbf-tools`

N.K.
kraemer@ins.uni-bonn.de

December 14, 2018

Contents

1. Purpose

The purpose of this guide is to standardise anything that needs standardising, like naming conventions, structure conventions and more. The purpose of the entire module is to collect most of the things I have programmed in the past 18 months with regard to radial basis functions. This collection has the purpose of handing it over, eventually, without losing any reusability possibilities–i.e. I should not be the only one who understands this.

2. Hierarchy

The hierarchy is supposed to be as flat as possible. The only things that are supposed to be in directories are

1. Figures
2. Pointsets

3. Naming conventions

We follow naming conventions with two purposes in mind:

1. Good programming practice
2. We want as much unification as possible

3.1. Good practice

The following is a list of most naming conventions regarding good practices:

1. **Variable naming:**
   - No underscores (privilege of python)
   - No all-uppercase variables (privilege of python)
   - Indicate new "term" with a single uppercase letter: `kernelFct`, `kernelMtrx`, `ptSet`
2. **Function naming:** verb-noun scheme, i.e. `buildKernelMtrx`, `getPtSet`, ...
3. **File naming:** Each file has to include the following information:
   (a) **Name:** e.g. `'interpolation.py'`
   (b) **Purpose:** Describe the purpose of the file in a single sentence (if that is not possible, think again about starting this file at all)
   (c) **Description:** Describe the method in two or three sentences giving the main keywords
   (d) **Author:** Usually `N.K. - kraemer@ins.uni-bonn.de`

3.2. Unification

The following is a list of most naming conventions regarding a unified system:

1. **Kernel functions:** We refer to kernel functions and kernel matrices using `kernel`, not `kern` nor `cov`
2. **Common Abbreviations:** We use as common abbreviations:
   - Point: `pt`
   - Pointset: `ptSet`
   - Matrix: `mtrx`
   - Length of a vector called `vecAbc`: `lenVecAbc`
   - Pointset for evaluation (plotting): `evalPtSet`
   - Lebesgue constant: `lebCnst`

4. Files

In the following I describe some files and their standardisations.

4.1. Kernel functions

We collect kernel functions in the file `kernelFcts.py`. They all take two points as inputs and give out a scalar. As an example, the Gaussian:

```
def gaussianKernel(x,y, lengthScale = 1.0):
    return np.exp(-np.linalg.norm(x-y)**2/(2*lengthScale**2))
```

The distance of the two inputs, x and y, is computed inside the function. The purpose of this is that we can construct kernel matrices in a very easy manner.

4.2. Kernel matrices

We collect kernel matrices in the file `kernelMtrcs.py`. The all take two pointsets as inputs and return a matrix. As an example, the standard kernel matrix:

```
def getKernelMtrx(ptSetOne, ptSetTwo, kernelFct):
    lenPtSetOne = len(ptSetOne)
    lenPtSetTwo = len(ptSetTwo)
    kernelMtrx = np.zeros((lenPtSetOne, lenPtSetTwo))
    for idx in range(lenPtSetOne):
        for jdx in range(lenPtSetOne):
            kernelMtrx[idx,jdx] = kernel(ptSetOne[idx,:], ptSetTwo[jdx,:])
    return kernelMtrx
```

The input pointsets need to have the same dimension, but do not need to match in size. The kernel function `kernelFct` needs to be of the form I described in section 4.1