Standardisation Guide for "rbf-tools"

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

December 14, 2018

Abstract

The purpose of this guide is to collect any standardisation I have come up with over time. On top of that, I mention other guidelines, like naming conventions, structure conventions and more.

Contents

1. Purpose of the module

The purpose of the module is to collect most of the things I have programmed in the past 18 months with regard to radial basis functions. This collection is supposed to be handed over, eventually, without losing any reusability possibilities–i.e. I should not be the only one who understands this.

Resuability is a driving force of most of the modules. Many features have to be used in almost every script; for instance, building a kernel matrix. I got sick of doing it from scratch everytime, hence I started this collection.

2. Hierarchy

The hierarchy is supposed to be as flat as possible. The only things that are supposed to be in directories are figures and pointsets; see Figure 1.

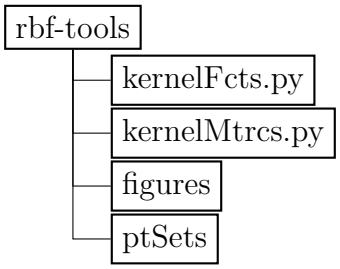

Figure 1: File structure of `rbf-tools`

3. Naming and coding conventions

I follow naming conventions with two purposes in mind:

1. Good programming practice
2. Unification

3.1. Good naming practice

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

1. **Variable naming:**
   - **Descriptive naming:** do not use `x`, `N` or `K`, but `pt`, `numPts` or `kernelMtrx`
   - **Short names:** do not use `standardKernelMatrixWithMaternKernel`, but `kernelMtrx`
   - No underscores (privilege of python)
   - No all-uppercase variables (privilege of python)
   - Indicate new "term" with a single uppercase letter: `kernelFct`, `kernelMtrx`, `ptSet`
2. **Commenting:** As long as the variables are named well, I do not need comments except for very few occasions
3. **Function naming:** verb-noun scheme, i.e. `buildKernelMtrx`, `getPtSet`, ...
4. **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 me

An exemplary header is the following, taken from `'interpolMatern1d.py'`:

```
# NAME: 'interpolMatern1d.py'
#
# PURPOSE: Basic 1-dimensional interpolation using Matern functions
#
# DESCRIPTION: I solve a system involving a Matern-kernel matrix
# where the Matern kernel is based on scipy.special's functions
# and plot the 10-th Lagrange function.
#
# AUTHOR: NK, kraemer(at)ins.uni-bonn.de
```

3.2. Unification

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

1. **Kernel functions:** I refer to kernel functions and kernel matrices using `kernel`, not `kern` nor `cov`
2. **Common Abbreviations:** I use as common abbreviations:
   - Indices: `idx`, `jdx`, `kdx`, ...
   - Point: `pt`
   - Pointset: `ptSet`
   - Numer of points: `numPts`
   - Matrix: `mtrx`, matrices: `mtrcs`
   - Length of a vector called `vecAbc`: `lenVecAbc`
   - Pointset for evaluation (plotting): `evalPtSet`
   - Number of evaluation points: `numEvalPts`
   - Lebesgue constant: `lebCnst`
   - Gaussian: `gauss` (as in `gaussKernel` instead of `gaussianKernel`)

3.3. Other good practices

1. **Functions:**
   - Each function should serve **a single** purpose which should be clear from the naming
   - Each function should be deterministic, i.e. two runs with the same input give the same output; see next point

2. **Seeds for random numbers:** Each file should always give the same result as long as nothing is changed. Hence, start everything that includes random numbers with `np.random.seed(15051994)`

3. Readability of a program often trumps performance

4. Files

In the following I describe some files and their standardisations.

4.1. Kernel functions

I 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 gaussKernel(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 I can construct kernel matrices in a very easy manner.

4.2. Kernel matrices

I 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] = kernelFct(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