# Smooth Approximation of Sort Functions Almost Everywhere in Euclidean Space and a Smooth Sort Algorithm

Chenxiao Tian[1]

*Department of mathematics, Peking University, No.5 Summer Palace Road, Haidian District, Beijing, China*

## Abstract

We study a smooth approximation of sort functions almost everywhere in n-dimensional Euclidean Space. A n-dimensional sort function is defined as $f$: $R^n \to R^n$: for each n − dimensional real vector $x \in R^n$: $(x_1, x_2, x_3 \dots x_{n-1}, x_n)$ $is\ translated\ into$ $x' = (x_{(1)}, x_{(2),} x_{(3)} \dots, x_{(k-1)}, x_{(k)})$, where $x_{(k)}\,(1 \le k \le n)$ is one of the components of $x$. At the same time these components have the following order relation, that is, $x_{(1)} \le x_{(2)} \le x_{(3)} \dots x_{(k-1)} \le x_{(k)}$. We will show that, this function can be smoothly approximated almost everywhere in Euclidean space, to be more exactly on a subset $\vartheta \subseteq R^n$,which is $\vartheta \overset{\text{def}}{=} \{ x \in R^n \mid x_{(k)}\text{-}x_{(k-1)} > 0, \text{for each k: } 2 \le k \le n\}$

After that, we will also show how to apply this construction to reorder data points, which can be regarded as a continuous sort algorithm.

**Keywords**: *smooth approximation sort algorithm*

## 1.Introduction

In some optimization and cluster problems, we sometimes need to reorder some data points into a sequence from small to large. There are a number of sorting algorithms such as bubble sort, insertion sort, binary insertion sort, shell sort, selection sort and so on, however, it's not surprised that none of these algorithms can be used to realize a smooth translation on those data for reordering data this operation itself is not a continuous process. But is it possible for us to construct a series of smooth n-dimensional function to approximate the n-dimensional sort function so that we may optimize problems related to sort and reorder by using some continuous optimization methods like gradient descent?

In this paper, we will give a kind of construction to smoothly approximate n-dimensional sort functions in the meaning of almost everywhere, to be more accurately, on a set $\vartheta \overset{\text{def}}{=} \{x \in R^n \mid x_{(k)} - x_{(k-1)} > 0, \text{for each k: } 2 \le k \le n\}$.The structure of this paper is simple, firstly, we will directly come to the n-dimensional cases (n≥2) and give the main construction result in this study, and then we will show how to apply this construction to reorder data points, which can be regarded as a smooth sort algorithm.

[1] Email address: 1700013239@pku.edu.cn
Telephone number: 18813187800

# 2.Smooth Approximation for the n-dimensional sort functions

### Definition 2.1

We define some functions as following:

**(1)** Let $\varphi(x, M_1)$ be a function from $R$ to $R$ with one parameter $M_1$:

$$\varphi(x, M_1)=\frac{1}{\pi}\arctan(M_1 x)+\frac{1}{2}$$

**(2)** Let $h(y, M_2, k)$ also be a function from $R$ to $R$ with two parameters $M_2$ and $k$ $(Here\ k\ is\ a\ positive\ integer)$:

$$h(y,M_2,k)=\frac{1}{1+\{M_2[y-(k-1)]\}^2}$$

Now we give the construction based on the symbols in definition 2.1 and abstract.

### Theorem 2.1

Let n≥ 2, here n is an integer. Also let $f(x)=(x_{(1)},x_{(2)},x_{(3)}\ldots,x_{(n-1)},x_{(n)})$ $(x\in\vartheta)$ be a n-dimensional sort function ,then we have:

$$x_{(k)}(x_1,x_2,x_3\ldots x_{n-1},x_n)$$
$$=\lim_{M_{2\to+\infty}}\lim_{M_{1\to+\infty}}\sum_{i=1}^{n}[h(\sum_{j=1\&j\neq i}^{n}\varphi(xi-xj,M_1),M_2,k)]\,xi$$

***Proof:***

We only need to show that for each k and $x=(x_1,x_2,x_3\ldots,x_{n-1},x_n)\in\vartheta$ :

$$x_{(k)}(x_1,x_2,x_3\ldots,x_{n-1},x_n)=\lim_{M_{2\to\infty}}\lim_{M_{1\to\infty}}\sum_{i=1}^{n}[h(\sum_{j=1\&j\neq i}^{n}\varphi(xi-xj,M_1),M_2,k)]xi$$

Firstly, Let $x_l=x_{(t)}$ $(l,t\in\{1,2,3\ldots n\})$,then we have:

$$\lim_{M_{1\to\infty}}\sum_{j=1\&j\neq i}^{n}\varphi(xi-xj,M_1)=\sum_{j=1\&j\neq i}^{n}\lim_{M1\to\infty}\varphi(xi-xj,M_1)$$

$$=\sum_{k=1\&k\neq t}^{n}\lim_{M1\to\infty}\varphi\left(x_{(t)}-x_{(k)},M_1\right)=\sum_{k=1\&k\neq t}^{n}\lim_{M1\to\infty}\frac{1}{\pi}\arctan\left[M1\left(x_{(t)}-x_{(k)}\right)\right]$$
$$=\sum_{k=1}^{t-1}\lim_{M1\to\infty}\frac{1}{\pi}\arctan\left[M1\left(x_{(t)}-x_{(k)}\right)\right]+\frac{1}{2}$$
$$+\sum_{k=t+1}^{n}\lim_{M1\to\infty}\frac{1}{\pi}\arctan\left[M1\left(x_{(t)}-x_{(k)}\right)\right]+\frac{1}{2}$$

Notice that $x=(x_1,x_2,x_3\ldots,x_{n-1},x_n)\in\vartheta$,so $x_{(t)}-x_{(k)}=\begin{cases}>0\ if\ t>k\\<0\ if\ t<k\end{cases}$

Then we get part of the limitation:

$$\sum_{k=1}^{t-1}\lim_{M1\to+\infty}\frac{1}{\pi}\arctan\left[M1\left(x_{(t)}-x_{(k)}\right)\right]+\frac{1}{2}$$
$$+\sum_{k=t+1}^{n}\lim_{M1\to+\infty}\frac{1}{\pi}\arctan\left[M1\left(x_{(t)}-x_{(k)}\right)\right]+\frac{1}{2}$$

$$=t-1+0$$
$$=t-1$$

Now we consider the whole limitation:

$$\lim_{M_{2\to+\infty}} \lim_{M_{1\to+\infty}} \sum_{i=1}^{n}[h(\sum_{j=1\&j\neq i}^{n} \varphi(xi-xj,M_1),M_2,k)]x_i$$

$$=\lim_{\mathrm{M2}\to+\infty} \sum_{\mathrm{i=1}}^{\mathrm{n}}[h(\mathrm{t}-1,M2,k-1)]x_i$$

$$=\lim_{\mathrm{M2}\to+\infty} \sum_{\mathrm{i=1}}^{\mathrm{n}}[\frac{1}{1+\{M_2[t-1-(k-1)]\}^2}]x_i$$

$$=\lim_{\mathrm{M2}\to+\infty} \sum_{\mathrm{s\neq k}}^{\mathrm{n}}[\frac{1}{1+\{M_2[s-1-(k-1)]\}^2}]\ x_{(s)}+\ x_{(k)}$$

$$=0+x_{(k)}$$

$$=x_{(k)}$$

So we finish the construction.

# 3.A Smooth Sort Algorithm Based on the Construction

Now we apply this construction to reorder data points:

We choose the vector:$x \in R^{300}, x=(x_1,x_2,x_3 \dots x_{299},x_{300})=(1,\sqrt{2},\sqrt{3}\dots\sqrt{299},\sqrt{300})$ and hope to reorder the components of the $x$ from small to large. At the same time, according to the construction in theorem 2.1 , we also use the function $f^{(300)}$ to approximate the accurate reorder function ,which is defined as following:

$f^{(300)}(x_1,x_2,x_3 \dots x_{299},x_{300},k)=\ \sum_{i=1}^{300}[h(\sum_{j=1\&j\neq i}^{300} \varphi(xi-xj,300000000),10000,k)]\ xi$

After using python (***Figure 1***) to calculate $f^{(300)}(x_1,x_2,x_3 \dots x_{299},x_{300},100)$,we get number 10.00003273705686 (***Figure 2***),which is quite close to the real result $x_{(100)}=10$.

```
import math
test=[]
for i in range(1,300):
    test.append(math.sqrt(i))
def f(x):
   a=(1/math.pi)*math.atan(300000000*x)+1/2
   return a
def u(x,k):
    u=1/(1+(1000*(x+1-k))**2)
    return u
def h(form):
    s=[]
    for j in range(0,len(form)):
        w=0
        for i in range(0,j):
            w=w+f(form[j]-form[i])
        for i in range(j+1,len(form)):
            w=w+f(form[j]-form[i])
        s.append(w)
    return s
def order(f,k):
    a=0
    for i in range(0,len(f)):
        a=a+u(h(f)[i],k)*f[i]
    return a
print(order(test,100))
```

***Figure 1***

```
In [1]: runfile('C:/Users/tcx/untitled64.py', wdir='C:/Users/tcx')
10.000003273705686
```

***Figure 2***

# Acknowledgement

This study is mainly guided by my tutor Minghua Deng. And it is also partially supported by the elite undergraduate training program of School of Mathematical Sciences in Peking University.

# References

*[1] Özgür Ergül. Sorting[J]. Guide to Programming & Algorithms Using R, 2013:99-115.*

*[2] Zapata, Guido I. (Guido Ivan). Functional Analysis, Holomorphy, and Approximation Theory[J]. 1984, 466(7303):267-71.*

*[3]Powell, M. J D. Approximation theory and methods[M]. 1981.*

*[4]Martin H. Schultz. L^2 Multivariate Approximation Theory[J]. Siam Journal on Numerical Analysis, 1969, 6(2):184-209.*