Dynamic Data Summary Structures and Computation of the Color Plots for the Temporal Analyses of Nutritive Containment

Universität Zürich, HS 2012
Author: Louis-Marie LOE

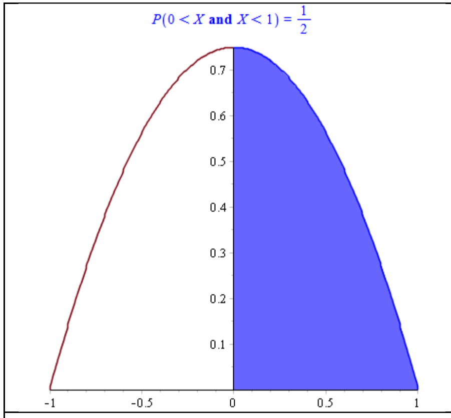
I. Introduction

I.1. Utility of density estimation

A basic characteristic of the behavior of a random variable X is its probability density function. Knowing the density function can help us in many ways. For example, if we want to know how many observations fall within the interval (a,b) we compute with the density f the probability as an integral over (a,b).

$P(a < X < b) = \int_a^b f(x)dx$. For example given the density function $f(x) = \begin{cases} \frac{3}{4}(1-x^2) \text{ if } |x| \le 1 \\ 0 \text{ otherwise} \end{cases}$

If we want to know how many observations fall within the interval (0,1) we compute with the density f the probability as an integral over (0,1) $P(0 < X < 1) = \int_0^1 \frac{3}{4}(1-x^2) = \frac{1}{2}$ which is represented by the shaded area in plot1 here below.



Plot1: The density $P(0 < X < 1) = \int_0^1 \frac{3}{4}(1-x^2) = \frac{1}{2}$ as an integral over (0, $\frac{1}{2}$)

If this value is high for a certain interval compared with the value for other intervals we would predict that the observations accumulate in (a,b). Thus the density tells us where the observations cluster and occur more frequently. Thanks to the graph of the density function we also may say the distribution of X is skewed or is multimodal. Hence the estimation of the unknown density f helps us see many structural elements, behavior and features of X. In this study we aim to use the density estimation in the computation of the color plots of nutrients in the Swiss Fed Database.

I.2 The Feed data in the Swiss Fed Database and the color plots

The goal of this project is to investigate the Kernel Density Estimation and the kernel regression for the computation of the color plots in the Swiss Fed Database. In this report we limit our investigation to that of the Kernel Density Estimation.

I.2.1 The Feed data

The Feed data is made up of several categories of data:

Feed Group: It is used to group different feeds of the same category. An example is Straw.

Feed: It is an agricultural foodstuff that is mostly used to feed animals. An example is Maize starch.

Species: It is used to identify a group of animals. An example is Poultry.

Nutrient: It is mostly an ingredient of a feed. An example is Calcium

Nutrient group: It identifies a group of nutrients according to their biological properties. An example is carbohydrates.

A feed sample: It identifies a sample on which a particular laboratory carries on measurements so as to determine the quantity of different nutrients present in the sample.

Origin: It represents the location from where the sample was taken.

Time: It identifies the moment when a sample was taken

Measurement: It identifies the quantity of a certain nutrient in a particular sample.

I.2.2 The colors plot

The color plots represent the density estimate for each nutrient of the different nutrients for which measurements have been taken across different regions of Switzerland. We do not have the direct density function of the nutrients involved in the study; instead we are given a set of n measurements with the unknown density f . For example given the nutrient calcium, there are 2000 measurements of this nutrient from 555 different locations. The colors plot employs hot and cold colors to represent high and low concentrations of calcium. This way it is easy to compare the concentration of calcium in different regions of Switzerland. The two color plots in **Figure1** and **Figure2** below show the density of the calcium and Magnesium nutrients across various regions of Switzerland. It can be drawn from these plots that in the lower region of Chur for example there is a greater concentration of Calcium in comparison to Magnesium.

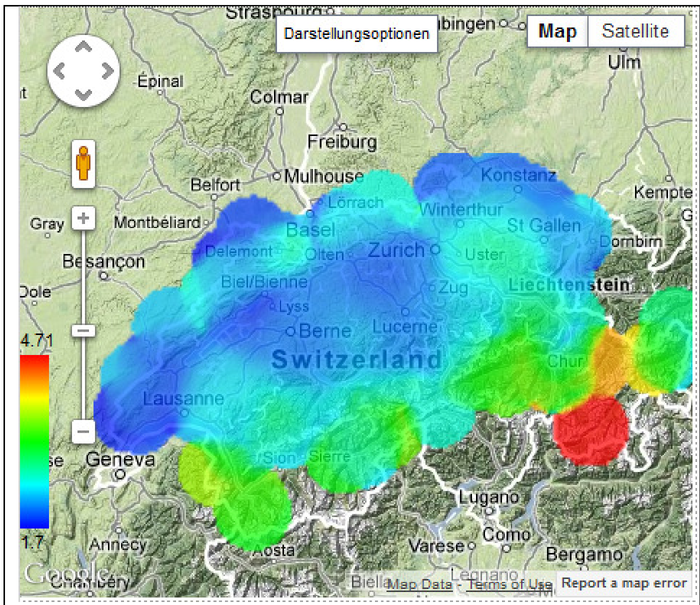
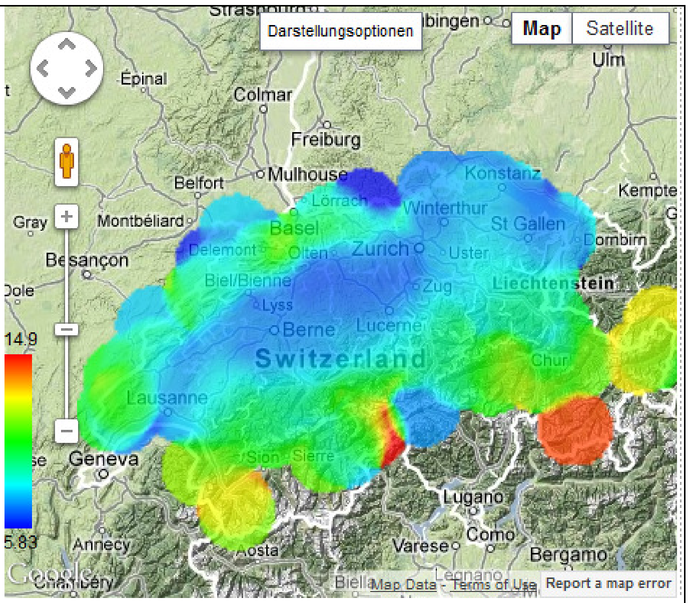
	

Figure1: Calcium densities across regions	**Figure2:** Magnesium densities across regions

Computation of the colors plot is done with the help of Kernel Density Estimation and Kernel Regression. These methods take into account all measurements of the query results and provide the expected nutritive value even in the areas that are between origins of the feed sample.

It is our goal in the present study to investigate the estimation of the density $\hat{f}$ of f given a set of n measurements of which we assume that they are realizations of independent, identically distributed bivariate random variables X $\{X_{i1}, X_{i2}\}_{i=1}^{n}$ using the Kernel Density Estimation approach.

II. Description of the kernel methods and the algorithms

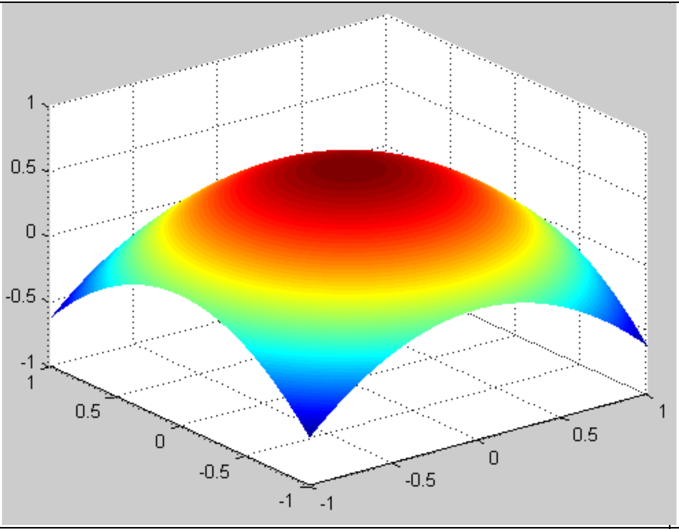
II.1. Description of the Epanechnikov and Gaussian kernel methods

Let $X_1,\ldots,X_n$ be a given bivariate data set where each data point $X_1,\ldots,X_n$ is defined in a 2-dimensional space. For this study, $X_1,\ldots,X_n$ is a data set of measurements of different nutrients taken across different regions of Switzerland. The multivariate kernel density estimator with kernel K and window width h at a grid point x with coordinates $x=(x_1,..,x_d)$

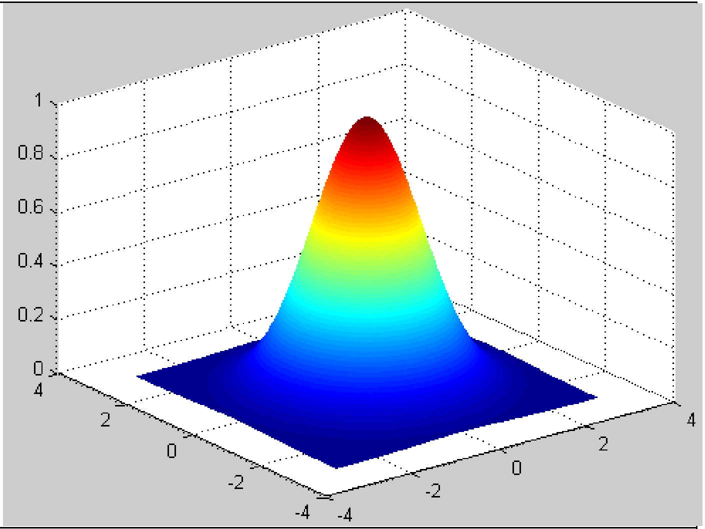
where d is the dimension is defined by $\hat{f}(x)=\frac{1}{nh^{d}}\sum_{i=1}^{n}K\left\{\frac{1}{h}(x-X_i)\right\}$. For our study we deal with a bivariate dataset and hence d=2. Thus the bivariate kernel density estimator with kernel K, d=2 and window h is defined by $\hat{f}(x_1,x_2)=\frac{1}{nh^{2}}\sum_{i=1}^{n}K\left\{\frac{x_1-X_{i1}}{h},\frac{x_2-X_{i2}}{h}\right\}$. The kernel function K(x) is now a function, defined for 2-dimensional x, satisfying $\int_{R^2}K(x)dx=1$. K(x) is defined as follows:

Epanechnikov kernel	Gaussian kernel
$K(x_1,x_2)=\begin{cases}\frac{2}{\pi}(1-x_1^2-x_2^2) \text{ if } (x_1^2+x_2^2)<1 \\ 0 \qquad \text{otherwise}\end{cases}$	$K(x_1,x_2)=\frac{1}{2\pi}e^{-\frac{x_1^2+x_2^2}{2}}$

The shapes of both kernels are given in the Plot2 and Plot3 below:



Plot2: Epanechnikov kernel shape



Plot3: Gaussian kernel shape

II.2. Description of the Epanechnikov and Gaussian kernel density computation algorithms

Input:

no_datapoints =number of datapoints in the given data set
dataX1= 1D array of datapoints along X1 dimension
dataX2=1D array of datapoints along X2 dimension
bandwidth_X1= optimal bandwidth for the datapoints along the X1 dimension
bandwidth_X2= optimal bandwidth for the datapoints along the X2 dimension
no_gridpoints= no of gridpoints in a 2-D grid

Processing: Using a nested for loop, we iterate over all grid points of a given 2-D grid. At each grid point we use the chosen kernel method to compute the density estimate at that point.

Output: kernel_density[i][j]= 2D array of kernel density values. Each array value represents the Epanechnikov kernel or the Gaussian kernel density estimate at the corresponding (I,j) grid coordinate.

II.2.1 Algorithms listing

```
1:.  compute_kernel_density(String kernel_name)
2:   compute_bandwidth(kernel_name)
3:   Kernel_distance=0
4:   Kernel_value=0
5:   Kernel_estimator=0
6:   For i=0 to no_gridpoints
7:       For j=0 to no_gridpoints
8:           For k=0 to no_dapoints
9:               Kernel_var_X1[k]=(i-dataX1[k])/bandwidth_X1
10:              Kernel_var_X2[k]=(j-dataX2[k])/bandwidth_X2
11:              Kernel_distance=compute_kernel_distance(kernel_var_X1[k], kernel_var_X2[k])
12:              kernel_value=compute_kernel(kernel_name, kernel_distance)
13:              kernel_estimator=kernel_estimator+kernel_value
14:          endFor
15:          kernel_density[i][j]=kernel_estimator/(no_datapoints*bandwith_X1*bandwith_X2)
16:      endFor
17   endFor
```

On line 1, the *compute_kernel_density(String kernel_name)* initiates either the Gaussian kernel density computation when *(kernel_name="gaussian")* or the epanechnikov kernel density computation when *(kernel_name="epanechnikov").*
On line 2, the *compute_bandwidth(kernel_name)* method uses the *compute_variance()* method to compute the optimal bandwidth for the specified kernel density using the formula :

(1) Optimal bandwidth $=\dfrac{\sigma * \mathrm{A(K)}}{n^{\frac{1}{6}}}$ where $\sigma=\sqrt{\dfrac{\sum_{i=1}^{n}\left(X_{1i}-\dfrac{\sum_{i=1}^{n} X_{1i}}{n}\right)^{2}}{n}}$ is the standard deviation computed as the square root of the variance of X_i.

$A(K)=0.96$ for the Gaussian kernel

$A(K)=1.77$ for the Epanechnikov kernel

We chose to compute the optimal bandwidth along both the X1 and X2 dimensions yielding the bandwidth_X2 and bandwidth_X2 values. This approach limits the impact of the variance of the input data along one dimension to the bandwidth of that dimension.

Lines 3 to 5 initialize the values of the variables *Kernel_distance*, *Kernel_value* and *Kernel_estimator* to 0.

On line 6 to 10 for each grid point (I,j) and all datapoint (dataX1[k], dataX2[k]) we compute the difference $(\frac{i}{bandwidth\_X1},\frac{j}{bandwidth\_X2})-(\frac{\text{dataX1[k]}}{bandwidth\_X1},\frac{\text{dataX2[k]}}{bandwidth\_X2})$

and store it as $\text{kernel\_var\_X1}=\frac{i-\text{dataX1[k]}}{bandwidth\_X1}$ and $\text{kernel\_var\_X2}=\frac{j-\text{dataX2[k]}}{bandwidth\_X2}$

Line 11 *Kernel_distance=compute_kernel_distance(kernel_var_X1[k], kernel_var_X2[k])* computes

$$\text{kernel\_distance}=\left(\frac{i-\text{dataX1[k]}}{bandwidth\_X1}\right)^2+\left(\frac{j-\text{dataX2[k]}}{bandwidth\_X2}\right)^2$$

Line 12 ***kernel_value=compute_kernel(kernel_name, kernel_distance)*** computes

$$\text{kernel\_value}=\frac{2}{\pi}\left(1-\left(\frac{i-\text{dataX1[k]}}{bandwidth\_X1}\right)^2-\left(\frac{j-\text{dataX2[k]}}{bandwidth\_X2}\right)^2\right)$$ for the Epanechnikov

$$\text{kernel\_value}=\frac{1}{2\pi}e^{-\frac{\left(\frac{i-dataX1[k]}{bandwidth\_X1}\right)^2+\left(\frac{j-dataX2[k]}{bandwidth\_X2}\right)^2}{2}}$$ for the Gaussian kernel.

Line 13 *kernel_estimator=kernel_estimator+kernel_value* sums all values of the kernel_value variable and when k=no_datapoints the variable kernel_estimator contains the sum of the kernel_values for all tuples in the dataset.

Line 14 *kernel_density[i][j]=kernel_estimator/(no_datapoints*bandwith_X1*bandwith_X2)* computes the kernel_density as $\text{kernel\_density}=\frac{\text{kernel\_estimator}}{(n*bandwidth\_X*bandwidth\_X2)}$ for each grid point (i,j).

II.2.2 Optimal bandwidth computation

As already mentioned the optimal bandwidth is computed using the following formula

(1) Optimal bandwidth $=\frac{\sigma*\text{A(K)}}{n^{\frac{1}{6}}}$ where $\sigma=\sqrt{\frac{\sum_{i=1}^{n}\left(X_{1i}-\frac{\sum_{i=1}^{n}X_{1i}}{n}\right)^2}{n}}$ is the standard deviation computed as the square root of the variance of X_i

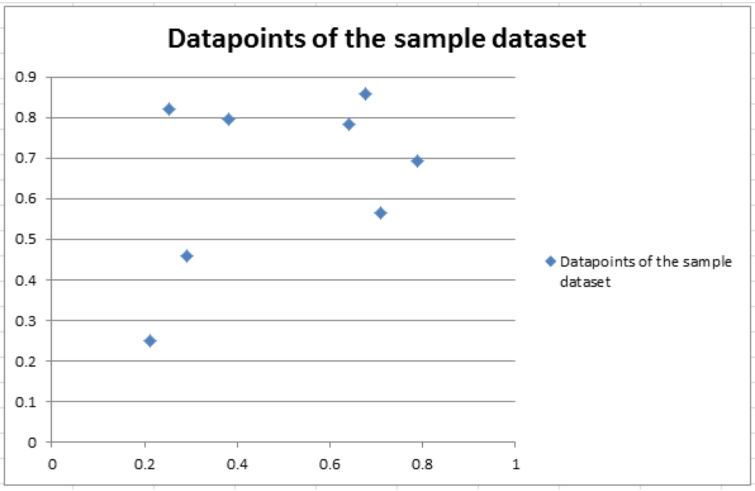
$A(K)=0.96$ for the Gaussian kernel

$A(K)=1.77$ for the Epanechnikov kernel

II.3 Examples of the Gaussian and Epanechnikov kernel densities computation

Tuples of the sample dataset (a)
(0.2123,0.2512) (0.2912,0.4578) (0.7112,0.5637) (0.789,0.6935) (0.3813,0.7945) (0.6418,0.7845) (0.2523,0.8199) (0.6781,0.8569)
Grid points at which the density will be estimated
(0,0) (0,0.5) (0,1) (0.5,0.5) (0.5,1) (1,0) (1,0.5) (1,1) on a (3x3) Grid

II.3.1 Epanechnikov kernel density computation for sample dataset (a)



$$\hat{f}(x_1,x_2)=\frac{1}{nh^2}\sum_{i=1}^{n}K\left\{\frac{x_1-X_{i1}}{h},\frac{x_2-X_{i2}}{h}\right\}$$

$$K(x_1,x_2)=\begin{cases}\frac{2}{\pi}(1-x_1^2-x_2^2) \text{ if } (x_1^2+x_2^2)<1\\ 0 \qquad\qquad\qquad \text{otherwise}\end{cases}$$

$$\hat{f}(0,0)=\frac{1}{8*bandwidth\_X1*bandwidth\_X2}\sum_{i=1}^{8}\frac{2}{\pi}\left(1-\left(\frac{0-X_{i1}}{bandwidth\_X1}\right)^2-\left(\frac{0-X_{i2}}{bandwidth\_X2}\right)^2\right)$$

$$\hat{f}(0,0.5)=\frac{1}{8*bandwidth\_X1*bandwidth\_X2}\sum_{i=1}^{8}\frac{2}{\pi}\left(1-\left(\frac{0-X_{i1}}{bandwidth\_X1}\right)^2-\left(\frac{0.5-X_{i2}}{bandwidth\_X2}\right)^2\right)$$

$$\hat{f}(0,1)=\frac{1}{8*bandwidth\_X1*bandwidth\_X2}\sum_{i=1}^{8}\frac{2}{\pi}\left(1-\left(\frac{0-X_{i1}}{bandwidth\_X1}\right)^2-\left(\frac{1-X_{i2}}{bandwidth\_X2}\right)^2\right)$$

$$\hat{f}(0.5,0)=\frac{1}{8*bandwidth\_X1*bandwidth\_X2}\sum_{i=1}^{8}\frac{2}{\pi}\left(1-\left(\frac{0.5-X_{i1}}{bandwidth\_X1}\right)^2-\left(\frac{0-X_{i2}}{bandwidth\_X2}\right)^2\right)$$

$$\hat{f}(0.5,0.5)=\frac{1}{8*bandwidth\_X1*bandwidth\_X2}\sum_{i=1}^{8}\frac{2}{\pi}\left(1-\left(\frac{0.5-X_{i1}}{bandwidth\_X1}\right)^2-\left(\frac{0.5-X_{i2}}{bandwidth\_X2}\right)^2\right)$$

$$\hat{f}(0.5,1)=\frac{1}{8*bandwidth\_X1*bandwidth\_X2}\sum_{i=1}^{8}\frac{2}{\pi}\left(1-\left(\frac{0.5-X_{i1}}{bandwidth\_X1}\right)^2-\left(\frac{1-X_{i2}}{bandwidth\_X2}\right)^2\right)$$

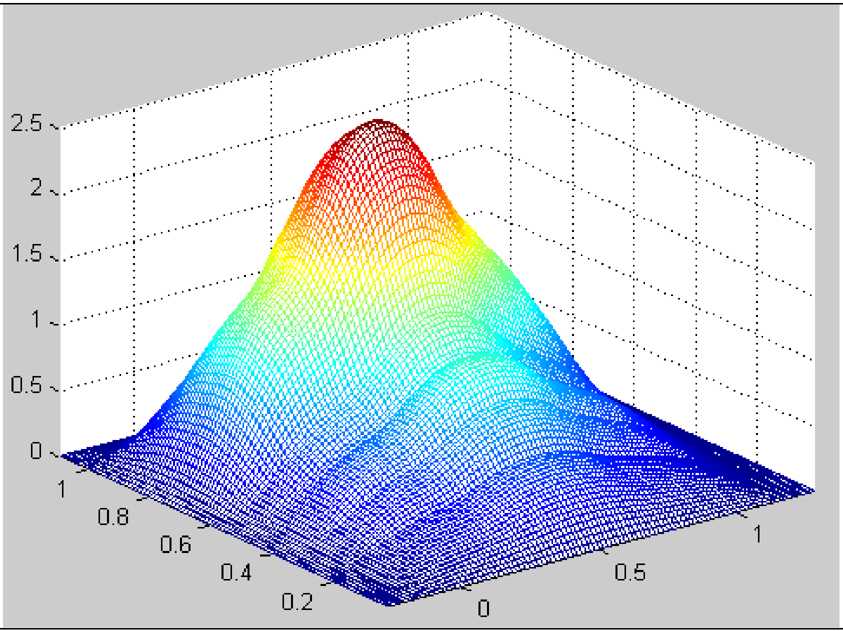
$$\hat{f}(1,0)=\frac{1}{8*bandwidth\_X1*bandwidth\_X2}\sum_{i=1}^{8}\frac{2}{\pi}\left(1-\left(\frac{1-X_{i1}}{bandwidth\_X1}\right)^2-\left(\frac{0-X_{i2}}{bandwidth\_X2}\right)^2\right)$$

$$\hat{f}(1,0.5)=\frac{1}{8*bandwidth\_X1*bandwidth\_X2}\sum_{i=1}^{8}\frac{2}{\pi}\left(1-\left(\frac{1-X_{i1}}{bandwidth\_X1}\right)^2-\left(\frac{0.5-X_{i2}}{bandwidth\_X2}\right)^2\right)$$

$$\hat{f}(1,1)=\frac{1}{8*bandwidth\_X1*bandwidth\_X2}\sum_{i=1}^{8}\frac{2}{\pi}\left(1-\left(\frac{1-X_{i1}}{bandwidth\_X1}\right)^2-\left(\frac{1-X_{i2}}{bandwidth\_X2}\right)^2\right)$$

Epanechnikov kernel density values for sample dataset (a)

$\hat{f}(0,0) =$ **0**	$\hat{f}(0.5,0) =$ **0**	$\hat{f}(1,0) =$ **0**
$\hat{f}(0,0.50) =$ **0**	$\hat{f}(0.5,0.5) =$ **0.723199**	$\hat{f}(1,0.5) =$ **0.064937**
$\hat{f}(0,1) =$ **0**	$\hat{f}(0.5,1) =$ **1.000574**	$\hat{f}(1,1) =$ **0**



Epanechnikov kernel density estimation for the sample data (a)

II.3.2 Walkthrough the Epanechnikov kernel density computation of $\hat{f}(0.5,0.5) =$ 0.723199

$$\hat{f}(0.5,0.5) = \frac{1}{8*bandwidth\_X1*bandwidth\_X2}\sum_{i=1}^{8}\frac{2}{\pi}\left(1-\left(\frac{0.5-X_{i1}}{bandwidth\_X1}\right)^2-\left(\frac{0.5-X_{i2}}{bandwidth\_X2}\right)^2\right)$$

1:

compute_kernel_density("epanechnikov")

2: ***compute_bandwidth("epanechnikov")***

kernel_constant=1.77

compute_variance()

variance1+=Math.pow((dataX1[i]-meanX1),2)= 0.38153802

variance2+=Math.pow((dataX2[i]-meanX2),2)= 0.5815029

variance_X1=Math.pow(variance1/no_datapoints, 0.5)= 0.218385559

variance_X2=Math.pow(variance2/no_datapoints, 0.5)= 0.269606867

optimal_bandwidth_X1=

```
     (variance_X1*kernel_constant)/Math.pow(no_datapoints,(1/6))= 0.27332678

     optimal_bandwidth_X2=

     (variance_X2*kernel_constant)/Math.pow(no_datapoints,(1/6))= 0.337434293
3:   Kernel_distance=0
4:   Kernel_value=0
5:   Kernel_estimator=0
6:
     For i=0 to no_gridpoints i=0.5
7:
     For j=0 to no_gridpoints j=0.5
8:
     For k=0 to no_dapoints k=0, 1, 2, 3, 4, 5, 6, 7

9:12 k=0
     Kernel_var_X1[0]=(0.5-dataX1[0])/bandwidth_X1=1.05258621
     Kernel_var_X2[0]=0.5j-dataX2[0])/bandwidth_X2=0.737328733
     Kernel_distance=compute_kernel_distance(kernel_var_X1[0], kernel_var_X2[0])= 1.651591391
     kernel_value =compute_kernel(kernel_name, kernel_distance)=0

     k=1
     Kernel_var_X1[1]=(0.5-dataX1[1])/bandwidth_X1=0.763920753
     Kernel_var_X2[1]=0.5j-dataX2[1])/bandwidth_X2=0.125061385
     Kernel_distance=compute_kernel_distance(kernel_var_X1[1], kernel_var_X2[1])= 0.599215267
     kernel_value =compute_kernel(kernel_name, kernel_distance)= 0.2552769

     k=2
     Kernel_var_X1[2]=(0.5-dataX1[2])/bandwidth_X1=-0.772701451
     Kernel_var_X2[2]=0.5j-dataX2[2])/bandwidth_X2=-0.188777493
     Kernel_distance=compute_kernel_distance(kernel_var_X1[2], kernel_var_X2[2])= 0.632704475
     kernel_value =compute_kernel(kernel_name, kernel_distance)= 0.233946194

     k=3
     Kernel_var_X1[3]=(0.5-dataX1[3])/bandwidth_X1=-1.057342422
     Kernel_var_X2[3]=0.5j-dataX2[3])/bandwidth_X2=-0.573444976
     Kernel_distance=compute_kernel_distance(kernel_var_X1[3], kernel_var_X2[3])= 1.446812137
     kernel_value =compute_kernel(kernel_name, kernel_distance)=0

     k=4
     Kernel_var_X1[4]=(0.5-dataX1[4])/bandwidth_X1=0.434278704
     Kernel_var_X2[4]=0.5j-dataX2[4])/bandwidth_X2=-0.872762508
     Kernel_distance=compute_kernel_distance(kernel_var_X1[4], kernel_var_X2[4])= 0.950312388
     kernel_value =compute_kernel(kernel_name, kernel_distance)= 0.03164816

     k=5
     Kernel_var_X1[5]=(0.5-dataX1[5])/bandwidth_X1=-0.518792925
     Kernel_var_X2[5]=0.5j-dataX2[5])/bandwidth_X2=-0.843127109
     Kernel_distance=compute_kernel_distance(kernel_var_X1[5], kernel_var_X2[5])= 0.980009421
     kernel_value =compute_kernel(kernel_name, kernel_distance)= 0.012732853
```

k=6
Kernel_var_X1[6]=(0.5-dataX1[6])/bandwidth_X1=0.906241238
Kernel_var_X2[6]=0.5j-dataX2[6])/bandwidth_X2=-0.948036422
Kernel_distance=compute_kernel_distance(kernel_var_X1[6], kernel_var_X2[6])= 1.72004624
kernel_value =compute_kernel(kernel_name, kernel_distance)= 0

k=7
Kernel_var_X1[7]=(0.5-dataX1[7])/bandwidth_X1=-0.651600987
Kernel_var_X2[7]=0.5j-dataX2[7])/bandwidth_X2=-1.057687399
Kernel_distance=compute_kernel_distance(kernel_var_X1[7], kernel_var_X2[7])= 1.543286481
kernel_value =compute_kernel(kernel_name, kernel_distance)= 0

13:
Kernel_estimator+=kernel_value = 0.533604108

15:
Kernel_density[0.5][0.5]=kernel_estimator)/(no_datapoints*optimal_bandwidth_X1*optimal_bandwidth_X2) = 0.723199

II.3.3 Gaussian kernel density computation for sample dataset (a)

$$\hat{f}(x_1, x_2) = \frac{1}{nh^2}\sum_{i=1}^{n} K\left\{\frac{x_1 - X_{i1}}{h}, \frac{x_2 - X_{i2}}{h}\right\} \quad K(x_1, x_2) = \frac{1}{2\pi} e^{-\frac{x_1^2 + x_2^2}{2}}$$

$$\hat{f}(0,0) = \frac{1}{8 * bandwidth\_X1 * bandwidth\_X2}\sum_{i=1}^{8}\frac{1}{2\pi} e^{-\frac{\left(\frac{0-X_{i1}}{bandwidth\_X1}\right)^2 + \left(\frac{0-X_{i2}}{bandwidth\_X2}\right)^2}{2}}$$

$$\hat{f}(0,0.5) = \frac{1}{8 * ban2width\_X1 * bandwidth\_X2}\sum_{i=1}^{8}\frac{1}{2\pi} e^{-\frac{\left(\frac{0-X_{i1}}{bandwidth\_X1}\right)^2 + \left(\frac{0.5-X_{i2}}{bandwidth\_X2}\right)^2}{2}}$$

$$\hat{f}(0,1) = \frac{1}{8 * bandwidth\_X1 * bandwidth\_X2}\sum_{i=1}^{8}\frac{1}{2\pi} e^{-\frac{\left(\frac{0-X_{i1}}{bandwidth\_X1}\right)^2 + \left(\frac{1-X_{i2}}{bandwidth\_X2}\right)^2}{2}}$$

$$\hat{f}(0.5,0) = \frac{1}{8 * bandwidth\_X1 * bandwidth\_X2}\sum_{i=1}^{8}\frac{1}{2\pi} e^{-\frac{\left(\frac{0.5-X_{i1}}{bandwidth\_X1}\right)^2 + \left(\frac{0-X_{i2}}{bandwidth\_X2}\right)^2}{2}}$$

$$\hat{f}(0.5,0.5) = \frac{1}{8 * bandwidth\_X1 * bandwidth\_X2}\sum_{i=1}^{8}\frac{1}{2\pi} e^{-\frac{\left(\frac{0.5-X_{i1}}{bandwidth\_X1}\right)^2 + \left(\frac{0.5-X_{i2}}{bandwidth\_X2}\right)^2}{2}}$$

$$\hat{f}(0.5,1) = \frac{1}{8 * bandwidth\_X1 * bandwidth\_X2}\sum_{i=1}^{8}\frac{1}{2\pi} e^{-\frac{\left(\frac{0.5-X_{i1}}{bandwidth\_X1}\right)^2 + \left(\frac{1-X_{i2}}{bandwidth\_X2}\right)^2}{2}}$$

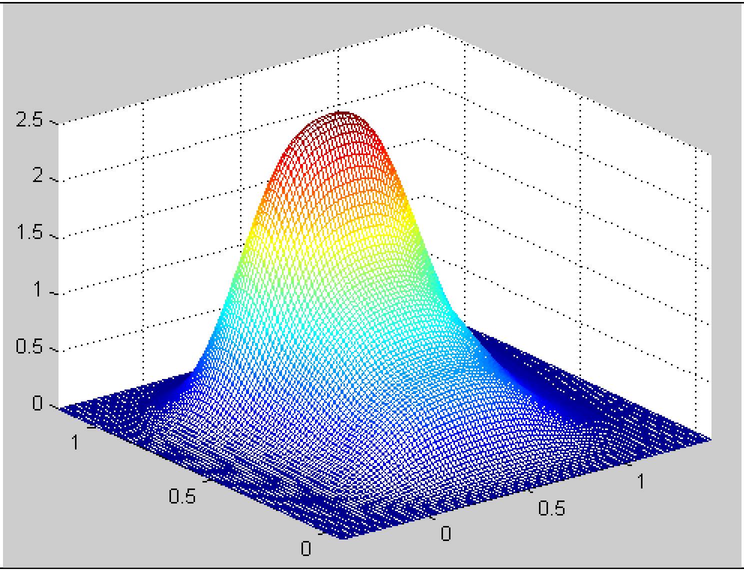
$$\hat{f}(1,0)=\frac{1}{8*bandwidth\_X1*bandwidth\_X2}\sum_{i=1}^{8}\frac{1}{2\pi}e^{-\frac{\left(\frac{1-X_{i1}}{bandwidth\_X1}\right)^2+\left(\frac{0-X_{i2}}{bandwidth\_X2}\right)^2}{2}}$$

$$\hat{f}(1,0.5)=\frac{1}{bandwidth\_X1*bandwidth\_X2}\sum_{i=1}^{8}\frac{1}{2\pi}e^{-\frac{\left(\frac{1-X_{i1}}{bandwidth\_X1}\right)^2+\left(\frac{0.5-X_{i2}}{bandwidth\_X2}\right)^2}{2}}$$

$$\hat{f}(1,1)=\frac{1}{8*bandwidth\_X1*bandwidth\_X2}\sum_{i=1}^{8}\frac{1}{2\pi}e^{-\frac{\left(\frac{1-X_{i1}}{bandwidth\_X1}\right)^2+\left(\frac{1-X_{i2}}{bandwidth\_X2}\right)^2}{2}}$$

Gaussian kernel density values for sample dataset (a)

$\hat{f}(0,0)=$ **0.107252**	$\hat{f}(0.5,0)=$ **0.057921**	$\hat{f}(1,0)=$ **0.001167**
$\hat{f}(0,0.50)=$ **0.253002**	$\hat{f}(0.5,0.5)=$ **0.999501**	$\hat{f}(1,0.5)=$ **0.278119**
$\hat{f}(0,1)=$ **0.121951**	$\hat{f}(0.5,1)=$ **0.936032**	$\hat{f}(1,1)=$ **0.14298**



Gaussian kernel density estimation for the sample data (a)

Note: $dataX1[k]=X_{i1}$ and $dataX2[k]=X_{i2}$

II.3.4 Walkthrough the Gaussian kernel density computation of $\hat{f}(0.5,0.5)=$ 0.999501

$$\hat{f}(0.5,0.5)=\frac{1}{8*bandwidth\_X1*bandwidth\_X2}\sum_{i=1}^{8}\frac{1}{2\pi}e^{-\frac{\left(\frac{0.5-X_{i1}}{bandwidth\_X1}\right)^2+\left(\frac{0.5-X_{i2}}{bandwidth\_X2}\right)^2}{2}}$$

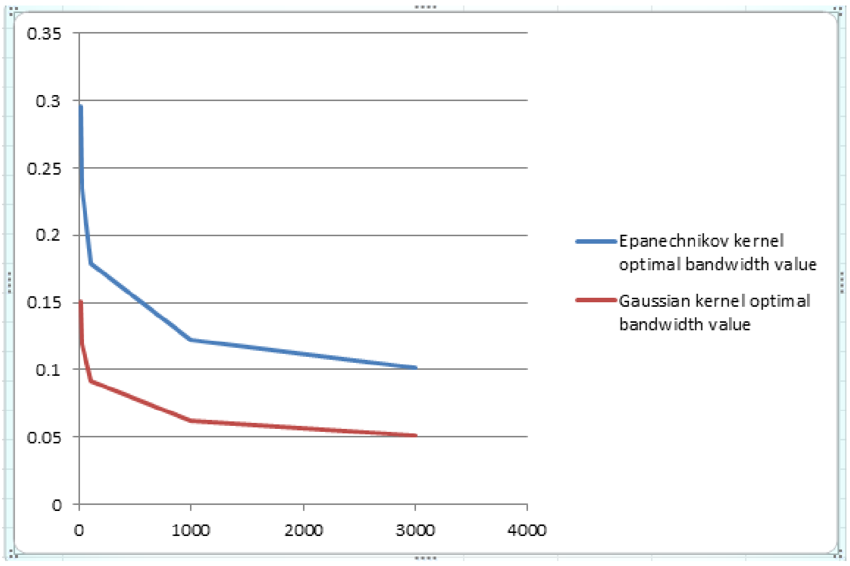
```
1:      compute_kernel_density(“gaussian“)
2:      compute_bandwidth(“gaussian”)
        kernel_constant=0.96
        compute_variance()
        variance1+=Math.pow((dataX1[i]-meanX1),2)= 0.38153802
        variance2+=Math.pow((dataX2[i]-meanX2),2)= 0.5815029
        variance_X1=Math.pow(variance1/no_datapoints, 0.5)= 0.218385559
        variance_X2=Math.pow(variance2/no_datapoints, 0.5)= 0.269606867
        optimal_bandwidth_X1=
        (variance_X1*kernel_constant)/Math.pow(no_datapoints,(1/6))= 0.27332678
        optimal_bandwidth_X2=
        (variance_X2*kernel_constant)/Math.pow(no_datapoints,(1/6))= 0.337434293
3:      Kernel_distance=0
4:      Kernel_value=0
5:      Kernel_estimator=0
6:      For i=0 to no_gridpoints i=0.5
7:      For j=0 to no_gridpoints j=0.5
8:      For k=0 to no_dapoints k=0, 1, 2, 3, 4, 5, 6, 7
9:12    k=0
        Kernel_var_X1[0]=(0.5-dataX1[0])/bandwidth_X1=1.940705825
        Kernel_var_X2[0]=0.5j-dataX2[0])/bandwidth_X2=1.359449852
        Kernel_distance=compute_kernel_distance(kernel_var_X1[0], kernel_var_X2[0])= 5.614443
        kernel_value =compute_kernel(kernel_name, kernel_distance)= 0.009613456

        k=1
        Kernel_var_X1[1]=(0.5-dataX1[1])/bandwidth_X1=1.408478889
        Kernel_var_X2[1]=0.5j-dataX2[1])/bandwidth_X2=0.230581928
        Kernel_distance=compute_kernel_distance(kernel_var_X1[1], kernel_var_X2[1])= 2.036980805
        kernel_value =compute_kernel(kernel_name, kernel_distance)= 0.057506322

        k=2
        Kernel_var_X1[2]=(0.5-dataX1[2])/bandwidth_X1=-1.424668301
        Kernel_var_X2[2]=0.5j-dataX2[2])/bandwidth_X2=-0.348058503
        Kernel_distance=compute_kernel_distance(kernel_var_X1[2], kernel_var_X2[2])= 2.15082449
        kernel_value =compute_kernel(kernel_name, kernel_distance)= 0.054324377

        k=3
        Kernel_var_X1[3]=(0.5-dataX1[3])/bandwidth_X1=-1.94947509
        Kernel_var_X2[3]=0.5j-dataX2[3])/bandwidth_X2=-1.057289174
        Kernel_distance=compute_kernel_distance(kernel_var_X1[3], kernel_var_X2[3])= 4.918313524
        kernel_value =compute_kernel(kernel_name, kernel_distance)= 0.013615768

        k=4
        Kernel_var_X1[4]=(0.5-dataX1[4])/bandwidth_X1=0.800701361
        Kernel_var_X2[4]=0.5j-dataX2[4])/bandwidth_X2=-1.609155874
        Kernel_distance=compute_kernel_distance(kernel_var_X1[4], kernel_var_X2[4])= 3.230505296
        kernel_value =compute_kernel(kernel_name, kernel_distance)= 0.031662488

        k=5
        Kernel_var_X1[5]=(0.5-dataX1[5])/bandwidth_X1=-0.956524456
        Kernel_var_X2[5]=0.5j-dataX2[5])/bandwidth_X2=-1.554515607
        Kernel_distance=compute_kernel_distance(kernel_var_X1[5], kernel_var_X2[5])= 3.331457806
        kernel_value =compute_kernel(kernel_name, kernel_distance)= 0.03010395

        k=6
        Kernel_var_X1[6]=(0.5-dataX1[6])/bandwidth_X1=1.670882283
        Kernel_var_X2[6]=0.5j-dataX2[6])/bandwidth_X2=-1.747942153
        Kernel_distance=compute_kernel_distance(kernel_var_X1[6], kernel_var_X2[6])= 5.847149375
        kernel_value =compute_kernel(kernel_name, kernel_distance)= 0.008557522

        k=7
        Kernel_var_X1[7]=(0.5-dataX1[7])/bandwidth_X1=-1.20138932
        Kernel_var_X2[7]=0.5j-dataX2[7])/bandwidth_X2=-1.950111143
        Kernel_distance=compute_kernel_distance(kernel_var_X1[7], kernel_var_X2[7])= 5.246269767
        kernel_value =compute_kernel(kernel_name, kernel_distance)= 0.011556527
13:     Kernel_estimator+=kernel_value = 0.216940411

15:     Kernel_density[0.5][0.5]=kernel_estimator)/(no_datapoints*optimal_bandwidth_X1*optimal_bandwidth_X2)
        = 0.999501
```

III. Experimental investigation of the Epanechnikov and Gaussian Kernel density estimation

III.1. Data size versus the optimal bandwidth

In the following experiment we vary the data size from 3000 tuples to 5 tuples and measure the impact of that variation on the optimal bandwidth for the Epanechnikov and Gaussian kernels. The aim is to derive the impact of the data size on the optimal bandwidth.

Data size	Epanechnikov kernel optimal bandwidth value	Gaussian kernel optimal bandwidth value
3000	0.101783405	0.051786999
1000	0.122235452	0.062192921
100	0.179417107	0.091286725
20	0.234616662	0.119372043
5	0.295598471	0.150399349

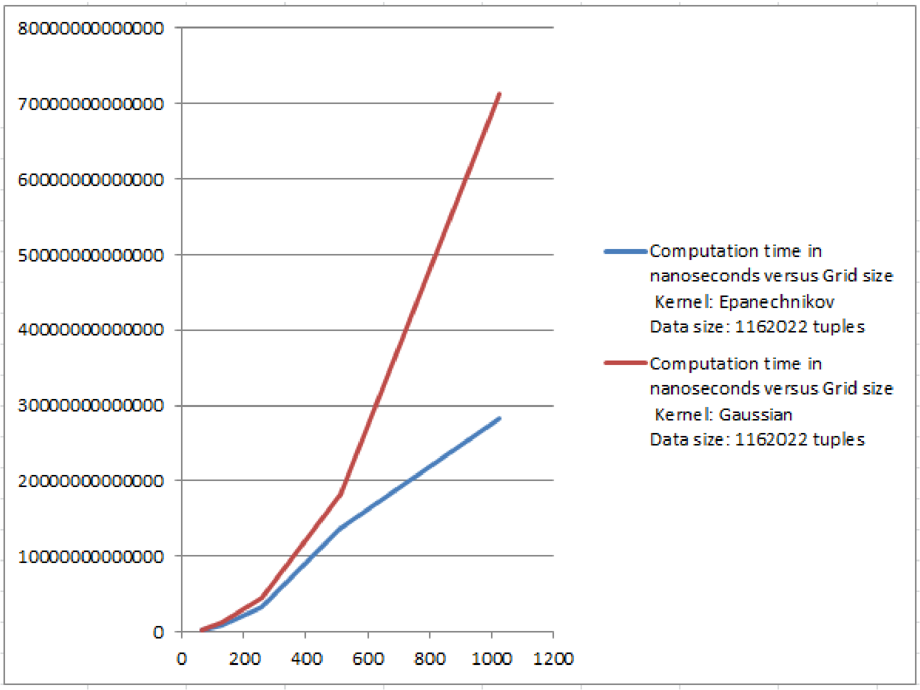


Conclusion: Given both kernel methods, the optimal bandwidth varies proportionally to the standard deviation. As the data size increases the standard deviation decreases and the bandwidth decreases towards its optimal value. A very small data size produces a noisy density estimate while an extremely high data size produces a flat density estimate.

III.2. Size of the grid versus computation time

In the following experiment we vary the Grid size from 1024x1024 to 64x64 and measure the Epanechnikov and Gaussian kernel density computation time in nanoseconds. This aims to derive the complexity of both kernel density computation methods with respect to the grid size.

Grid size **Datasize= 1162022 tuples**	**Computation time in nanoseconds using the Epanechnikov kernel**	**Computation time in nanoseconds using the Gaussian kernel**
1024x1024	28315171956752	71334280770701
512x512	13702809041226	18192756745384
256x256	3377400673769	4600724715457
128x128	857880447366	1148827387954
64x64	216239146320	302001790901

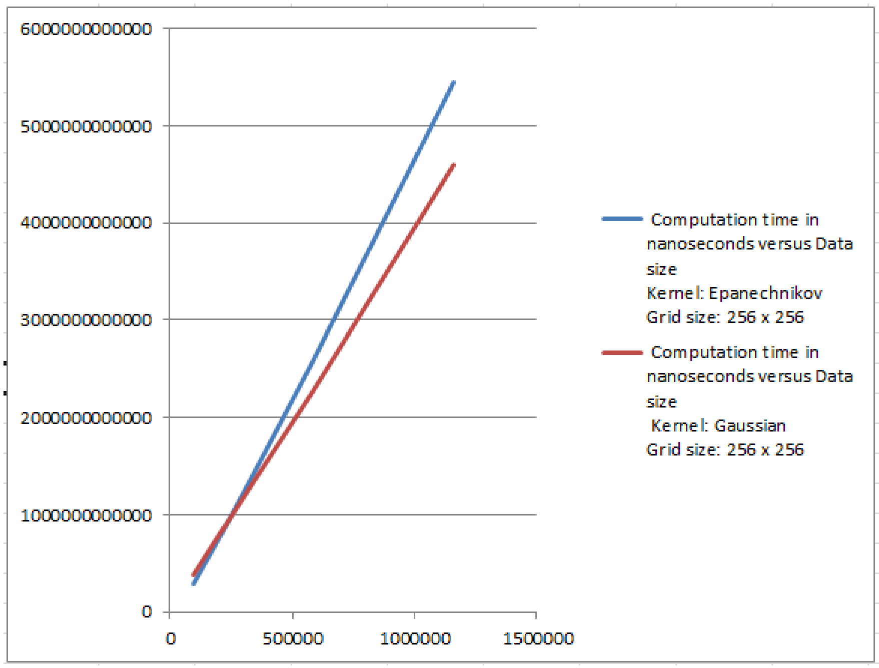


Conclusion: Given the same data size it takes longer to compute the Gaussian kernel density values as the Grid size increases. But the complexity of both algorithms grow as O(n x m^2) where n is the data size and m is the grid size.

III.3. Size of the data versus computation time

In this experiment we vary the data size from 1162022 tuples to 93961 tuples and measure the Epanechnikov and Gaussian kernel density computation in nanoseconds. This aims to derive the complexity of both kernel density computation methods with respect to the data size.

Data size in number of tuples Grid size = 256x256	**Computation time in nanoseconds for the Epanechnikov kernel**	**Computation time in nanoseconds for the Gaussian kernel**
1162022	5444841324650	4600724715457
581011	2572178380390	2266619038455
337404	1402105543582	1320084227244
182237	673827997578	713229588863
93961	286008646613	374953785523



Conclusion: Given the same Grid size it takes longer to compute the Epanechnikov kernel density values as the data size increases. But the complexity of both algorithms grow as O(n x m^2) where n is the data size and m is the grid size.

Way forward

As a way forward following the investigations in this study, the naïve algorithms implemented herein will need to be improved, expanded using both the kernel density methods and the kernel regression methods and implemented on the Swiss Feed production data so as to address all other outstanding issues related to this project.

References

1. **Scott, D.W. (1992). Multivariate Density Estimation: Theory, Pratice, and Visualization. John Wiley&Sons, New York.**
2. **Silverman, B.W. (1986). Density Estimation for Statistics and Data Analysis. Chapman and Hall, London.**