Estimation of Probability Density Functions and Graphical Models using regularized Sparse Grids.

Karthikeya Sampa Subbarao

Technical University of Munich, Department of Informatics
Boltzmannstr. 3, D-85748 Garching bei Munich, Germany
karthikeya108@gmail.com

Introduction

Density estimation reconstructs based on observed data, an unobservable underlying probability density function. The Curse of Dimensionality is frequently encountered in the process of density estimation in high-dimensional spaces. In this project we employ model order reduction with sparse grid techniques to deal with the curse of dimensionality. L2 regularised density estimation model has been developed using the Sparse Grid techniques which has proven to be effective in dealing with the curse of dimensionality [3]

$$\widetilde{P} = argmin_{f \in V} \int_{\Omega} (f(x) - p_{\varepsilon}(x))^2 dx + \lambda || \wedge f ||_{L^2}^2$$

In this project we estimate $\widetilde{P}$ defined as the Probability density function of an exponential family. In which, the required function is approximated using sparse grid discretization technique. In other terms MAP (Maximum A Posteriori) approach with sparse grid discretization. Minimization is performed using variational methods: The solution of the underlying

$$\sum_{j=1}^{N} \alpha_j^{i+1} a(\varphi_k, \varphi_j) = \frac{1}{n} \sum_{i=1}^{N} \varphi_k(x_i) - \int \varphi_k(x) \frac{exp(\sum_{j=1}^{N} \alpha_j^i \varphi_j(x))}{\int exp(\sum_{j=1}^{N} \alpha_j^i \varphi_j(z)) dz} dx$$

In the matrix form we have

$$A\alpha^{i+1} = q - \phi(\alpha^i)$$

The challenge lies in solving the following Non-Linear part

$$\phi(\alpha^i) = \int \varphi_k(x) \frac{exp(\sum_{j=1}^{N} \alpha_j^i \varphi_j(x))}{\int exp(\sum_{j=1}^{N} \alpha_j^i \varphi_j(z)) dz} dx$$

Graphical Models provide a principled approach to deal with uncertainty through the use of probability theory, and an effective approach to cope with the complexity through the use of graph theory. Density estimation and Knowledge discovery are the two main applications of graphical models and in this project we work on deriving a graphical model which will facilitate us to get rid of unimportant dependencies among the variables (dimensions) and hence improve the efficiency of the model.

We employ Markov Chain Monte Carlo sampling for computing the Non-linear term mentioned above. Different approaches to solve the Non-linear term, like Monte Carlo Integration has been evaluated in [5]

Sparse Grid with Modified Linear Basis

Estimation of density using Grid based methods is computationally infeasible for higher dimensional problems. However if we switch to a hierarchical grid instead of an equidistant grid it is shown that we can construct a grid which retains the high accuracy of the full mesh grid with much less grid points [4]. Such grids are termed as Sparse Grids. In depth details of the topic can be found in [1].

The existence of a relation between sparse grids and dimension decomposition has been studied and sparse grid approach to dimension decomposition which works similar to ANOVA decompositions has been shown in [2]. In this project we employ sparse grid with modified basis which corresponds to ANOVA like decomposition and further associate the components with the factors of a factor graph. Thus we formulate an approach to compute, analyze and visualize ANOVA components which will help us to identify unwanted or less important components and remove them in order to reduce the complexity of the problem and hence improving the efficiency of computation.

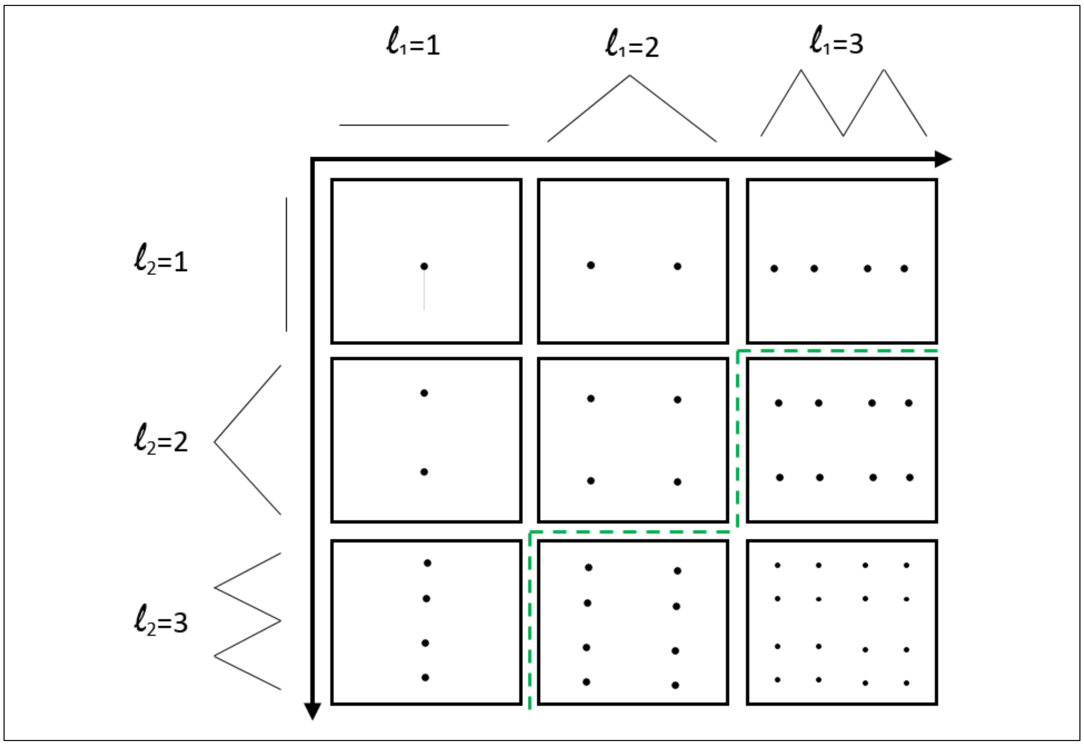

Figure 1: A two Dimensional Full Grid with modified linear basis on Level 3 with the dashed line indicating the sparse grid

Modeling the Dependecies

We use factor graph to model the dependencies among the variables (dimensions). Initially we model the denpendency of all the variables being dependent on one another by creating a fully connected factor graph. Then gradually based on the estimation of the co-efficients, unwanted factors are removed from the factor graph and the grid is coarsened accordingly. This will reduce the computation complexity of estimating the co-efficents. Different strategies for identifying the unwanted or less important factors are employed and evaluated as follows:

Mean Co-efficient Thresholding

First step is to estimate the co-efficients of all the grid points. Then we fetch and store the grid point index and corresponding factors that it is involved with. For each such factors we list all the co-efficients. If the mean of the co-efficients is less than some threshold value then the factor is marked to be deleted. Once we have evaluated all the factors we delete the factors that have been marked to be deleted, if there are no higher order factors that have this factor as a subset.

Computation of Non-Linear term using MCMC

We need to compute this Non-Linear term.

$$\phi(\alpha^i) = \int \varphi_k(x) \frac{exp(\sum_{j=1}^{N} \alpha_j^i \varphi_j(x))}{\int exp(\sum_{j=1}^{N} \alpha_j^i \varphi_j(z))dz} dx$$

We can write the above in as follows:

$\phi(\alpha^i) = \int \varphi_k(x)p(x)dx$

Further

$\phi(\alpha^i) = \int f(x)p(x)dx$

Where $p(x) = \frac{exp(\sum_{j=1}^{N} \alpha_j^i \varphi_j(x))}{\int exp(\sum_{j=1}^{N} \alpha_j^i \varphi_j(z))dz}$

$p(x)$ gives the probability with which we need sample the input values 'x' for the function f(x).

We can compute $\phi(\alpha^i)$ in two ways:

1. We can compute the denominator of $p(x)$ which is $\int exp(\sum_{j=1}^{N} \alpha_j^i \varphi_j(z))dz$ using Monte Carlo Integration and use the value in computing $\phi(\alpha^i)$ which is also computed with Monte Carlo Integration. This method resulted in large error when used for evaluating the integral.

2. Since we do not have any proper method to compute $\int exp(\sum_{j=1}^{N} \alpha_j^i \varphi_j(z))dz$ we can compute $p(x)$ using Markov Chain Monte Carlo (MCMC) by creating a model based on the Sparse Grid structure and assuming particular distribution (preferable Uniform) for the random variables. Once we sample the values for the random variables (x) using MCMC, we can evaluate the function f(x) on these values and compute the mean which will be the expected value of the function f(x).

$E[f(x)] = \int f(x)p(x)dx$

For computing the expected value of the function iteratively based on the updated model, we can use the sample values obtained in the previous sampling run. We can either store the sampled values of the random variables and use them to initiate the sampling process which should result in faster convergence of the sampling run, or we can sample few new points and calculate the expected value by reusing the output of the function values.

Algorithms

Determining the Co-efficients of the Basis Functions and Updating the Grid:

```
**Data**: ModLineaerGrid, Co-effecients, Factor graph, Set of DataPoints $X = x_1, .., x_M$
**Result**: Co-efficients corresponding with each Basis Function
Choose Parameters $\omega > 0$, $\varepsilon > 0\, and\, i_{max} = Grid\ Size$;
Calculate $q$;
Calculate $A$;
Calculate $\Phi(\alpha^{initial})$;
**while** *residual* > $\varepsilon$ *and* $i <= i_{max}$ **do**
    $b = q - \Phi(\alpha^i)$;
    Solve $A = \tilde{\alpha} b$;
    $\alpha^{i+1} = \alpha^i + \omega\tilde{\alpha}$;
    Calculate $\Phi(\alpha^{i+1})$;
    residual = $||A\alpha^{i+1} - q + \Phi(\alpha^{i+1})||$;
    $i = i + 1$;
    *Update Factor Graph*;
    *Coarsen the Grid*;
    **if** *Grid updated* **then**
        Update Co-efficient vector - retaining only non zero values;
        Calculate $q$;
        Calculate $A$;
        Calculate $\Phi(\alpha^i)$;
    **end**
**end**
```

Algorithm 1: Determining the Co-efficients of Basis Functions and updating the Grid

Where

$$\Phi(\alpha^i) = \int \varphi_k(x) \frac{exp(\sum_{j=1}^{N} \alpha_j^i \varphi_j(x))}{\int exp(\sum_{j=1}^{N} \alpha_j^i \varphi_j(z))dz} dx$$

$$A = a(\varphi_i, \varphi_j) = \lambda \int \bigtriangledown \varphi_i(x) \bigtriangledown \varphi_i(x) dx$$

$$q = \frac{1}{n} \sum_{i=1}^{n} \varphi_k(x_i)$$

Updating Factor Graph through Co-efficient Thresholding:

```
Data: ModLineaerGrid, Co-effecients, Factor graph
Result: Updated Factor Graph
Choose Parameters: Co-efficient Threshold Value;
for Grid Points in Grid do
    Fetch and Store the grid point index and corresponding tuple of interacting
    factors;
    Delete all the higher order interacting factors in the Factor Graph which are
    higher than the maximum length of the interacting factors obtained in the
    previuos step;
    Fetch and Store the interacting factors and all the corresponding co-efficient
    values contributing to each interacting factor;
    if average absolute values of the co-efficients corresponding to a interacting
    factor < Co-efficient Threshold then
        Add the interacting factor tuple to the delete list;
    end
end
for factor tuple in delete list do
    if factor tuple is not contained in any higher order interacting factors tuple then
        Delete the factor tuple from the Factor Graph
    end
end
```

Algorithm 2: Update Factor Graph through Co-efficient Thresholding

Coarsening the Grid based on the Factor Graph:

```
Data: ModLineaerGrid, Factor graph
Result: Coarsened Grid
for Grid Points in Grid do
    Fetch and Store the grid point index and corresponding tuple of interacting
    factors;
    if factors not in factor graph then
        set the corresponding co-efficient to zero
    end
    Delete the Grid Points whose corresponding co-efficients are zero
end
```

Algorithm 3: Coarsening the Grid based on the Factor Graph

Results:

Upon obtaining the α and the *Grid*, I estimate the density of the input data points using:

$$f(x) = \exp\left(\sum_{i=1}^{n} \alpha_i \varphi_i(x)\right)$$

I compare the results with the Kernel Density Estimation results implemented in ScitKit and Scipy Python packages. Scikit requires *Bandwidth* to be provided explicitly where as Scipy computes the *Bandwidth.* Scipy Uses *Gaussian* kernel by default. For Scikit KDE, I set the kernel as *Gaussian* and *Bandwidth* as 0.2 Following table provides the *mean* of the densities across the data points.

DataSet (Dimension)	SG DE	Scikit KDE	Scipy KDE
Toy(1)	2.45	2.8127	2.815
Toy(2)	16.434	7.963	7.867
Toy(3)	1.156	23.129	21.366
Ripley Garcke(2)	29.144	2.1477	2.1465

Toy DataSet (1D):

A Toy Dataset generated using Numpy. Normalized One Dimensional Dataset with:

Mean : 0.5
Variance : 0.1

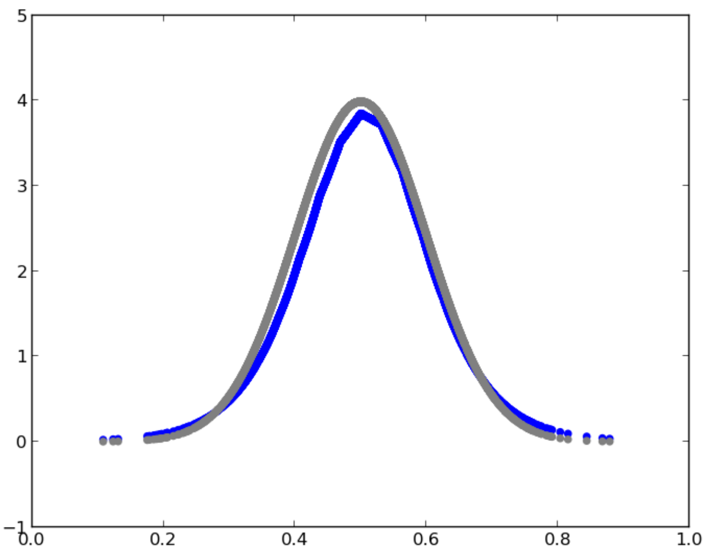

Figure 2: Probability Density Distribution using Sparse Grids

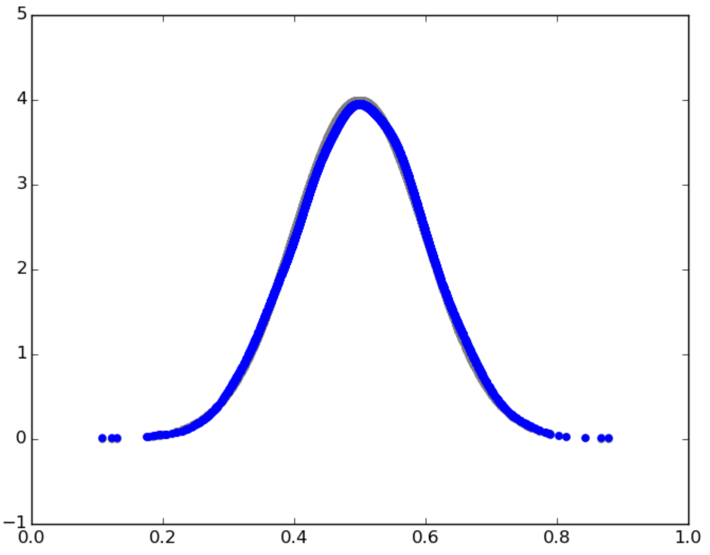

Figure 3: Probability Density Distribution using SciKit KDE

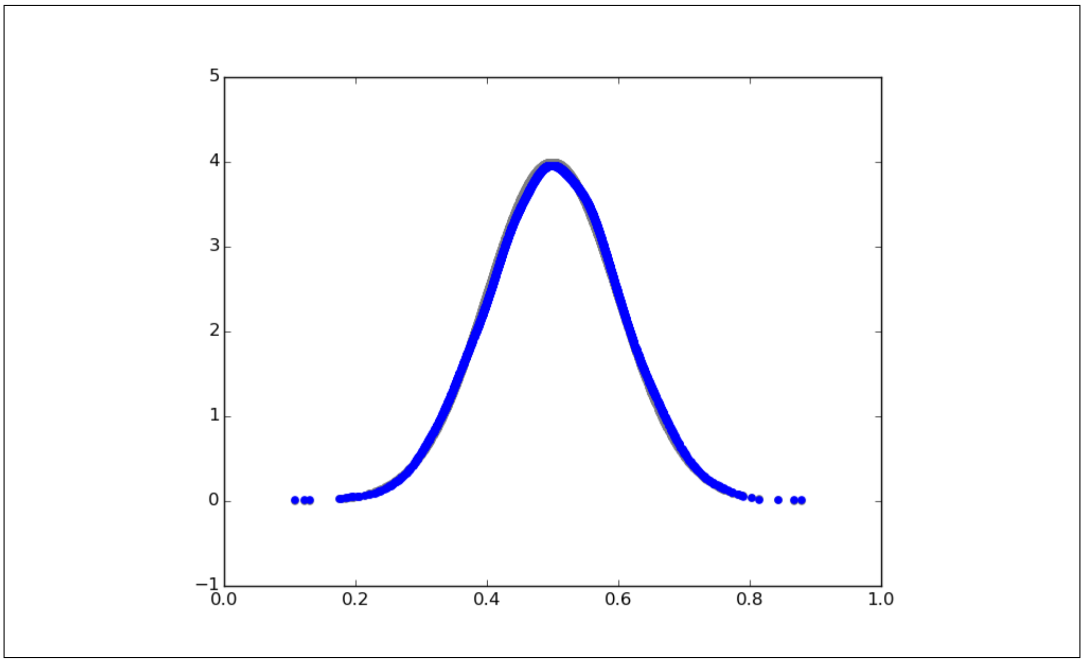

Figure 4: Probability Density Distribution using Scipy KDE

Toy DataSet (2D):

A Toy Dataset generated using Numpy. Normalized Two Dimensional Dataset with:

Mean $\begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}$ Co-variance Matrix $\begin{pmatrix} 0.01 & 0 \\ 0 & 0.01 \end{pmatrix}$

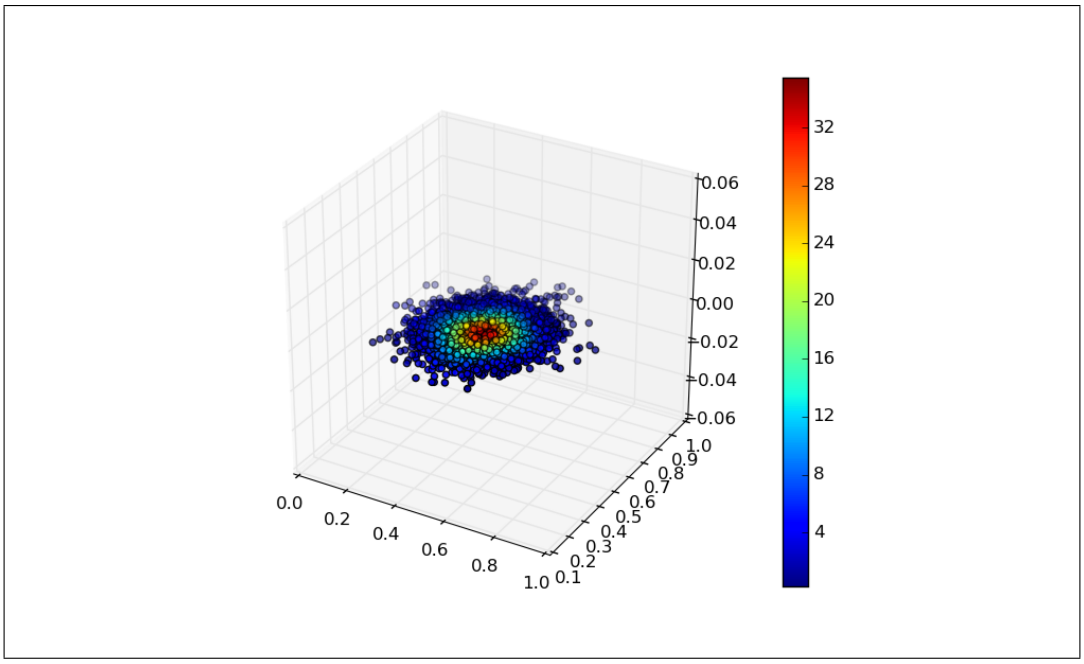

Figure 5: Probability Density Distribution using Sparse Grids

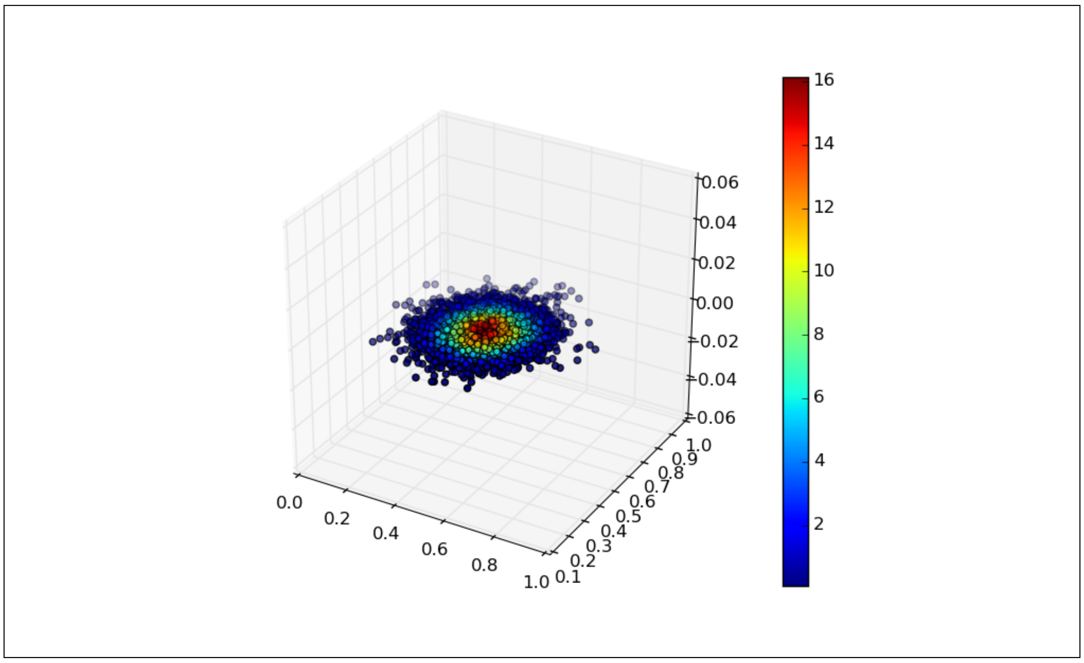

Figure 6: Probability Density Distribution using SciKit KDE

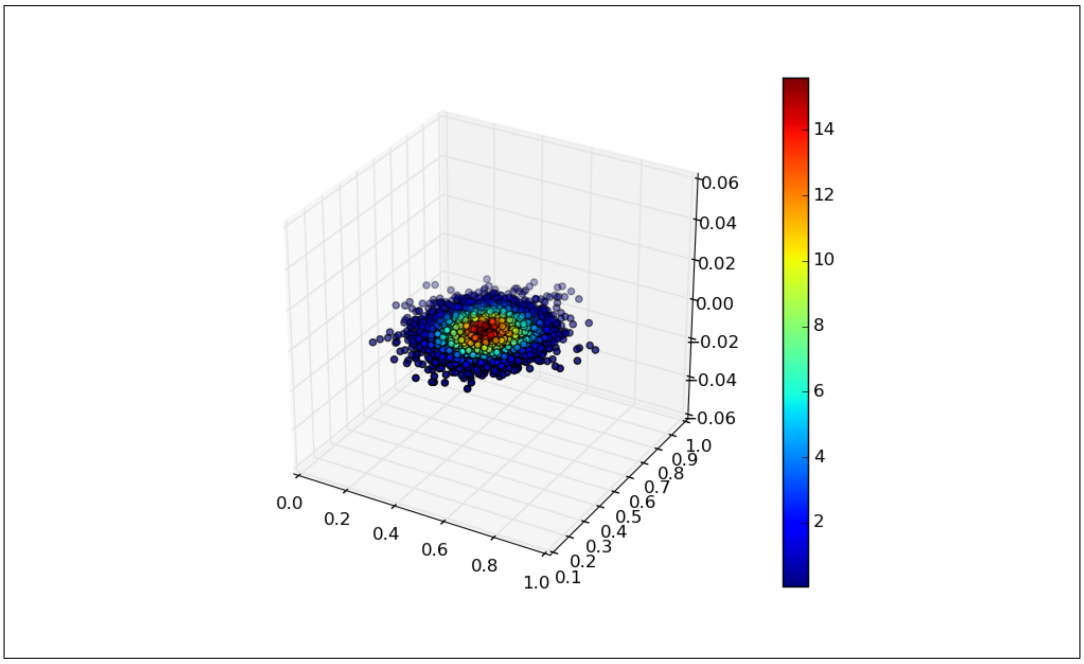

Figure 7: Probability Density Distribution using Scipy KDE

Toy DataSet (3D):

A Toy Dataset generated using Numpy. Normalized Three Dimensional Dataset with:

$$\text{Mean} \begin{pmatrix} 0.5 \\ 0.5 \\ 0.5 \end{pmatrix} \text{Co-variance Matrix} \begin{pmatrix} 0.01 & 0 & 0 \\ 0 & 0.01 & 0 \\ 0 & 0 & 0.01 \end{pmatrix}$$

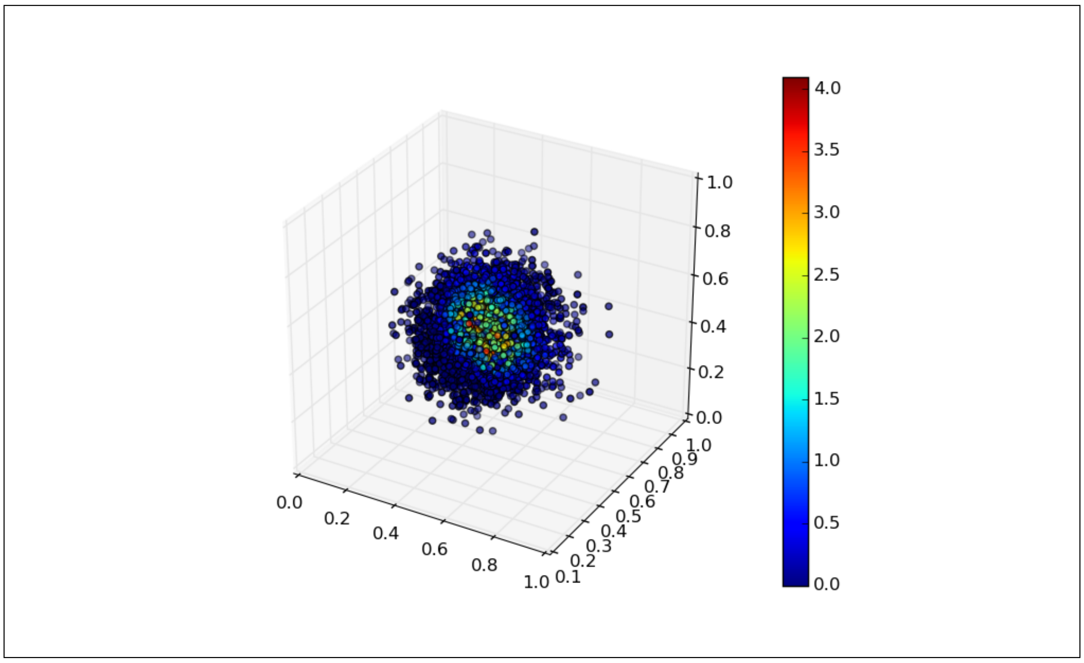

Figure 8: Probability Density Distribution using Sparse Grids

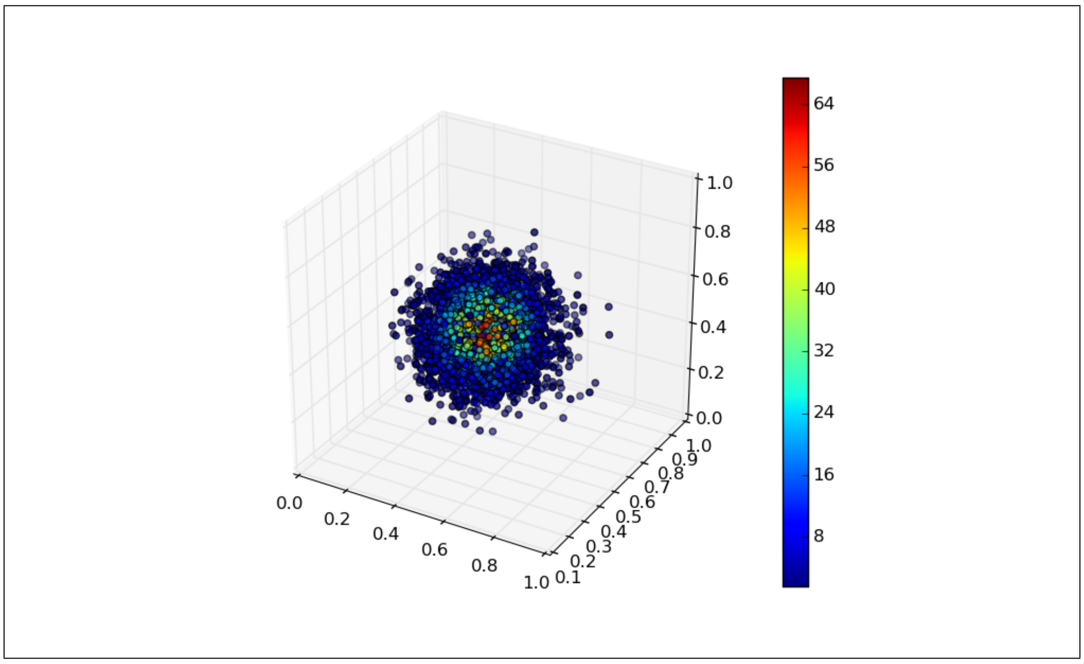

Figure 9: Probability Density Distribution using SciKit KDE

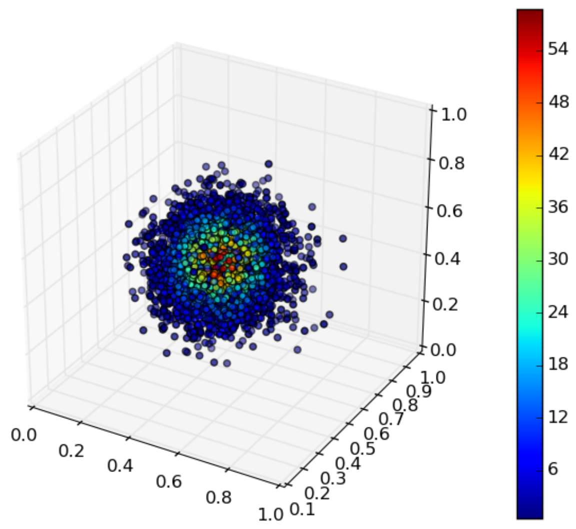

Figure 10: Probability Density Distribution using Scipy KDE

Validation of results:

In order to validate the estimated values of the co-efficients, I compute the true values of the co-efficients and compare them with the estimated values.

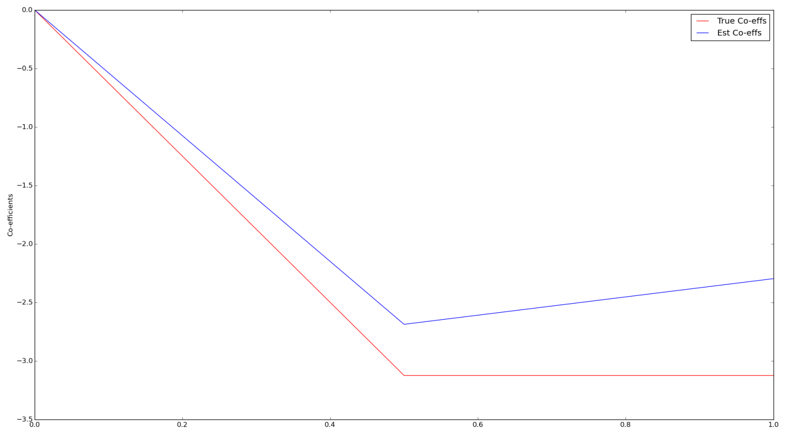

Figure 11: Ture and Estimated values of the Co-efficients of 1D Toy DataSet

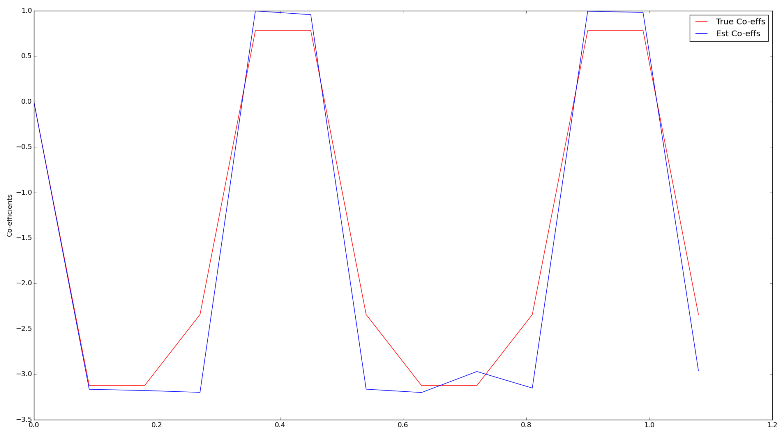

Figure 12: Ture and Estimated values of the Co-efficients of 2D Toy DataSet

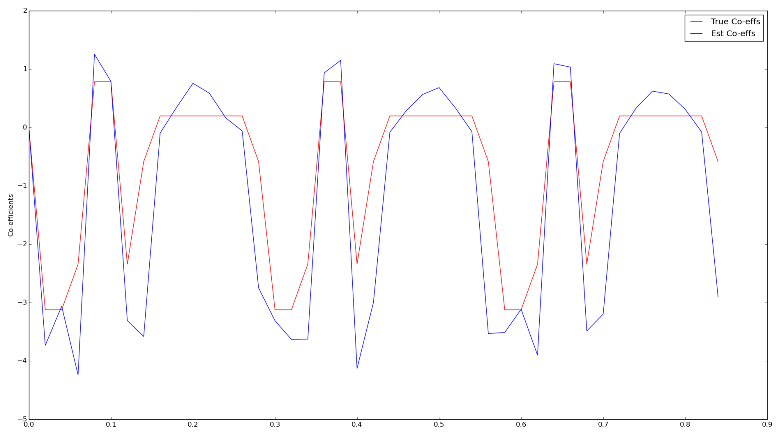

Figure 13: Ture and Estimated values of the Co-efficients of 3D Toy DataSet

Ripley Garcke DataSet:

Normalised two dimensional data set.

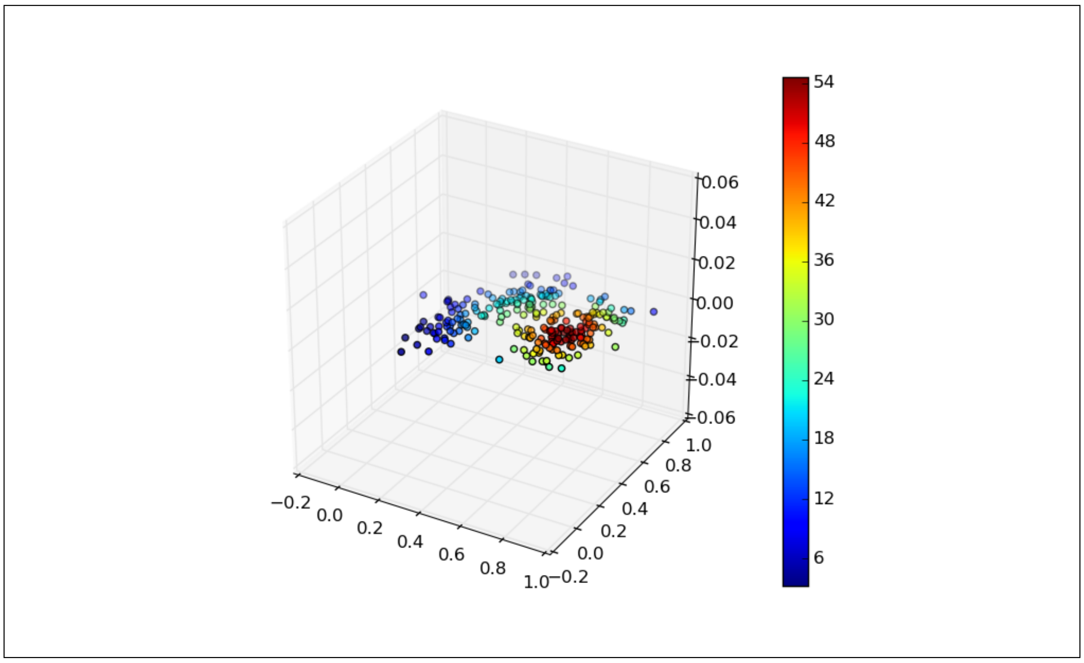

Figure 14: Probability Density Distribution using Sparse Grids

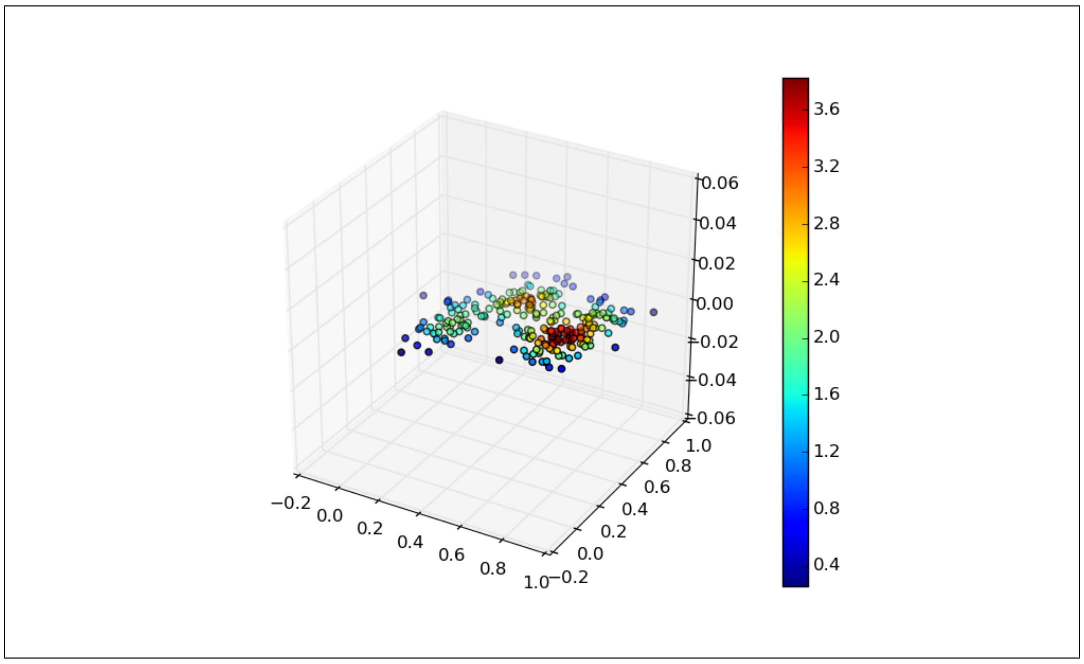

Figure 15: Probability Density Distribution using SciKit KDE

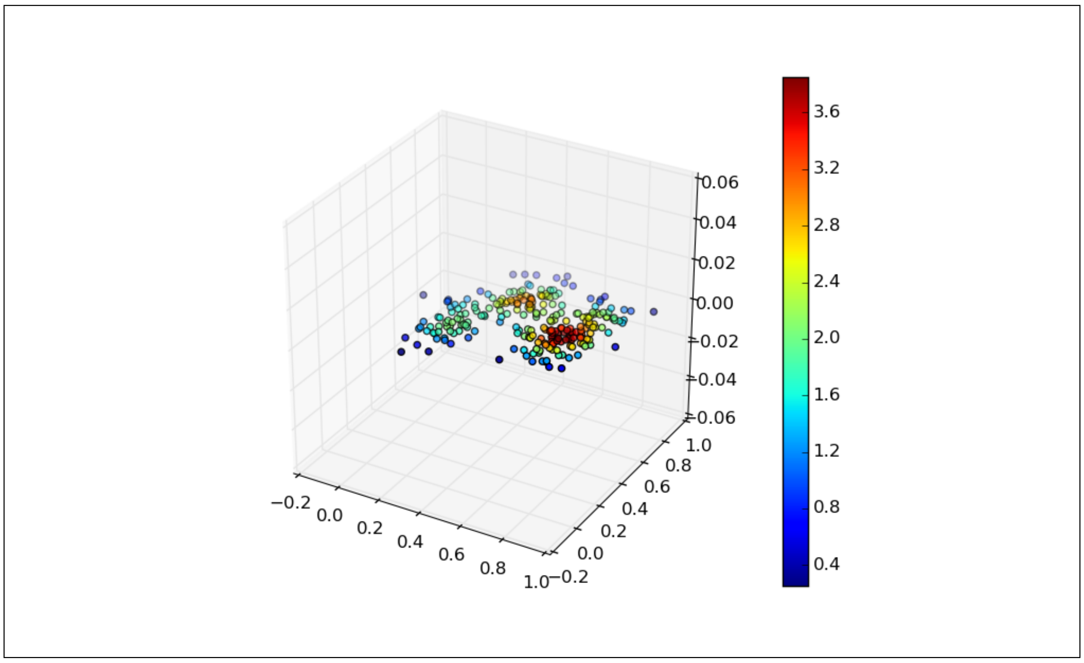

Figure 16: Probability Density Distribution using Scipy KDE

References

[1] Michael Griebel Hans-Joachim Bungartz. Sparse grids. 2004.

[2] M. Hegland. Adaptive sparse grids. 2003.

[3] Benjamin Peherstorfer. Model order reduction of parameterized systems with sparse grid learning techniques. 2013.

[4] S. A. Smolyak. Quadrature and interpolation formulas for tensor products of certain classes of functions. *Dokl. Akad. Nauk SSSR*, 4, 1963.

[5] Sebastian Soyer. Nonlinear density estimation with applications in astronomy. 2014.