INTER-DISCIPLINARY PROJECT

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

Advisor: **Valeriy Khakhutskyy**

Examiner: **Univ.-Prof. Dr. Hans-Joachim Bungartz**

May 2015

Contents

1 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 [4]

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

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 [6]

2 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 [5]. 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.

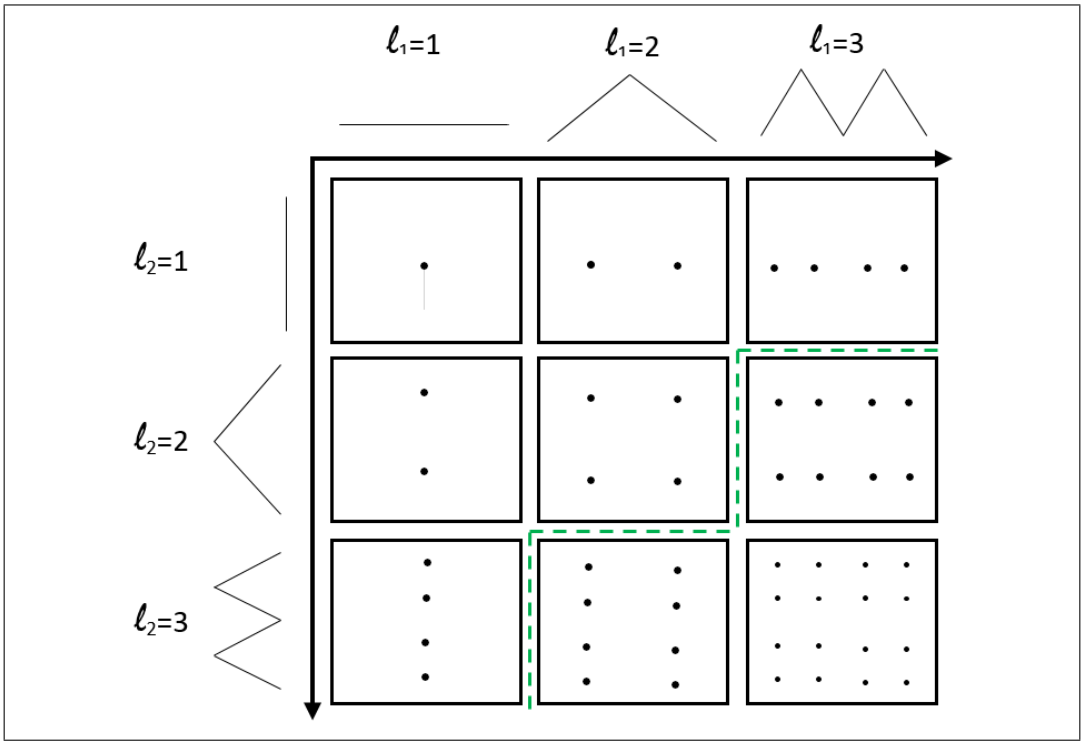


Figure 1: A two Dimensional Full Grid with modified linear basis on Level 3 with the dashed line indicating the sparse grid

3 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.

4 Algorithms

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

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

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

Algorithm 1: Determining the Co-efficients of 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* > ε *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 2: Update 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 3: 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

5 Experiments

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

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

5.1 Comparision of results:

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

Note: The mean of the densities across the data points estimated using Sparse Grids may not match with the Kernel Density Estimates, however the shape of the function is similar in both, which is evident on the following plots.

Toy DataSet (1D):

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

Mean : 0.5
Variance : 0.1

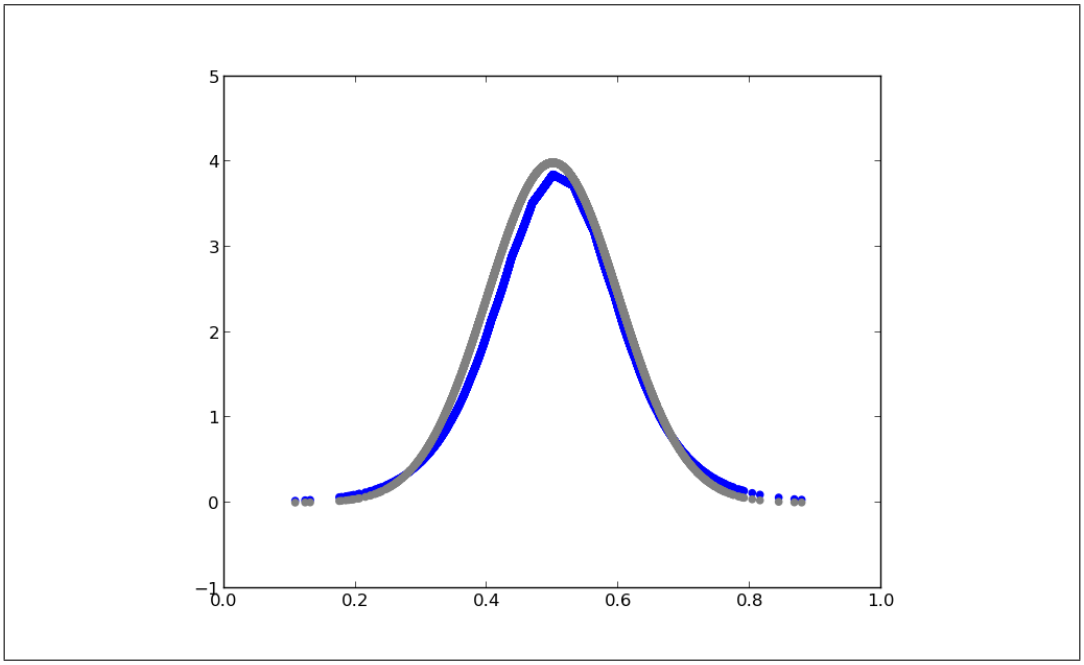


Figure 2: Probability Density Distribution using Sparse Grids

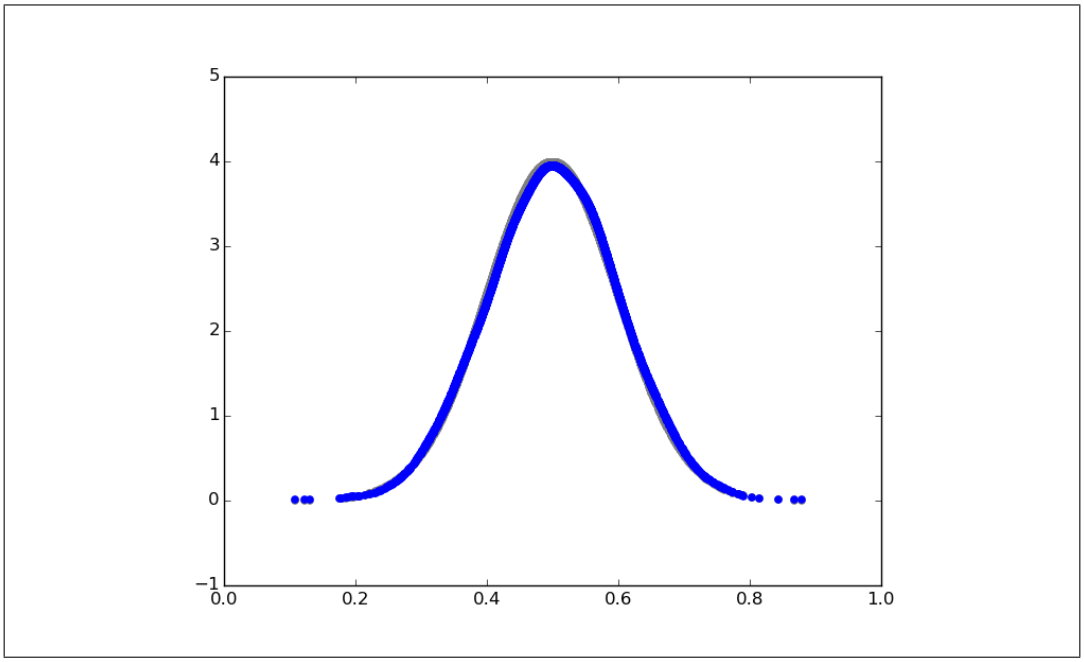


Figure 3: Probability Density Distribution using SciKit KDE

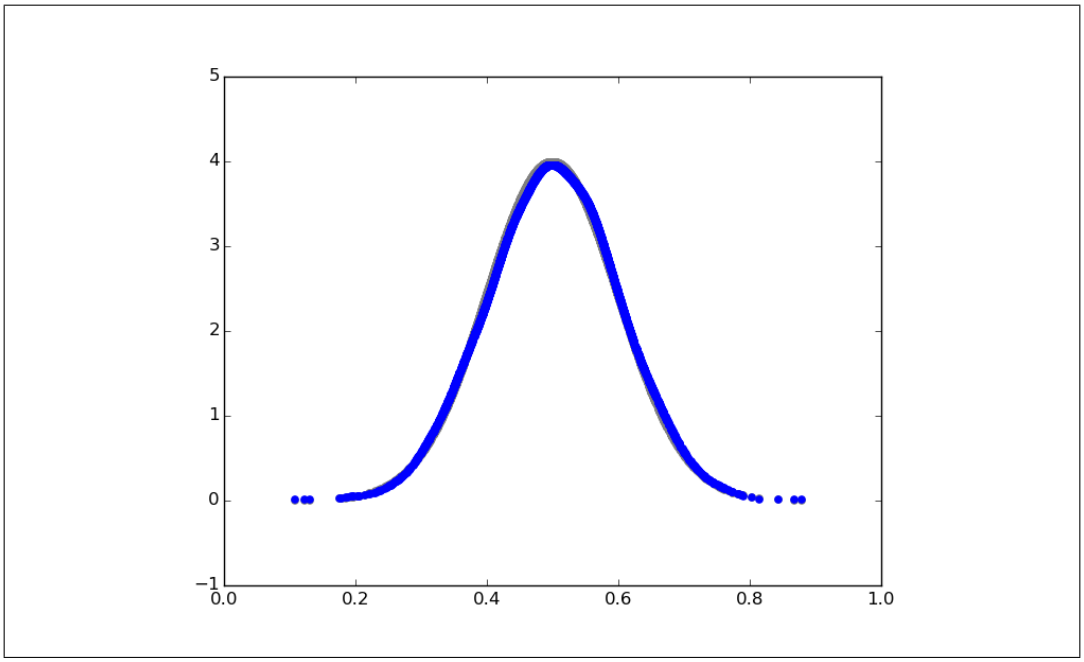


Figure 4: Probability Density Distribution using Scipy KDE

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 $\begin{pmatrix} 0.01 & 0 \\ 0 & 0.01 \end{pmatrix}$

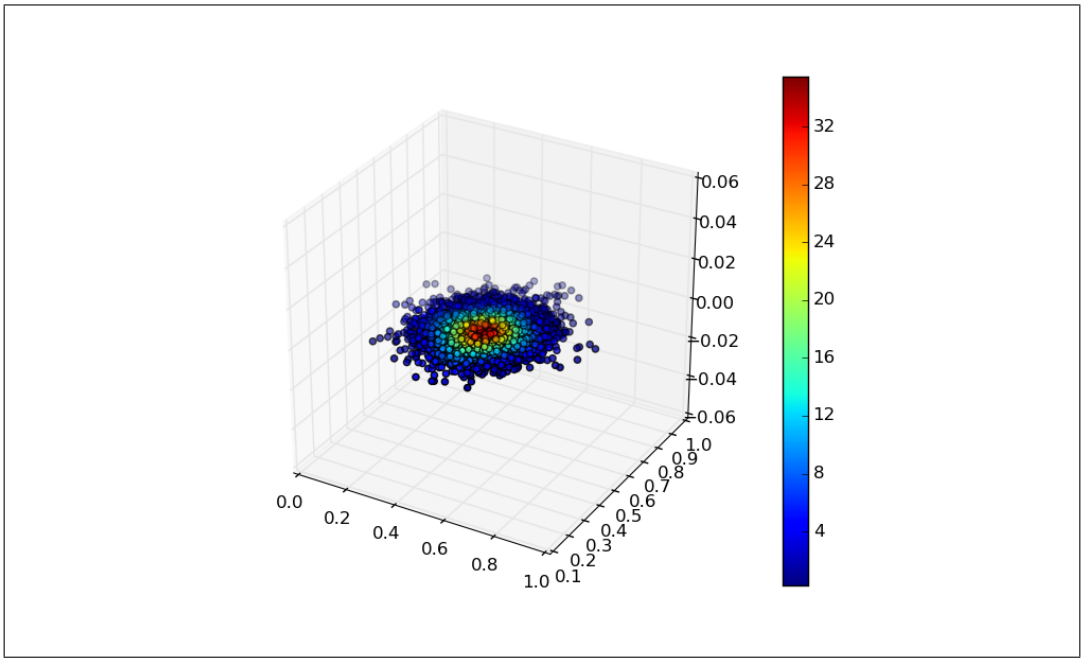


Figure 5: Probability Density Distribution using Sparse Grids

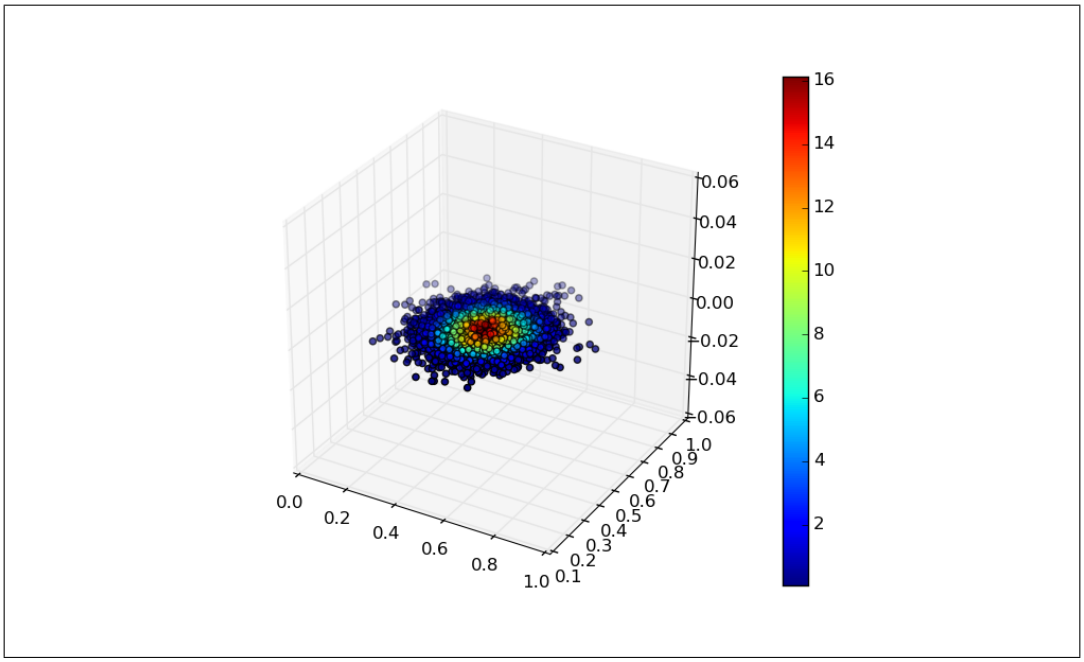


Figure 6: Probability Density Distribution using SciKit KDE

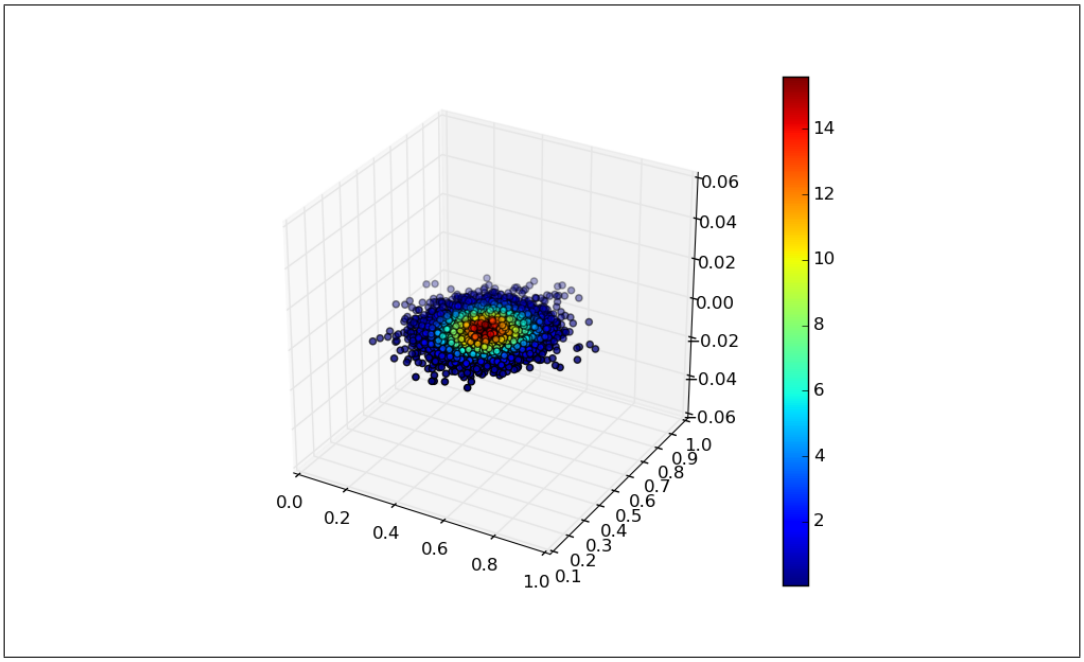


Figure 7: Probability Density Distribution using Scipy KDE

Toy DataSet (3D):

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

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

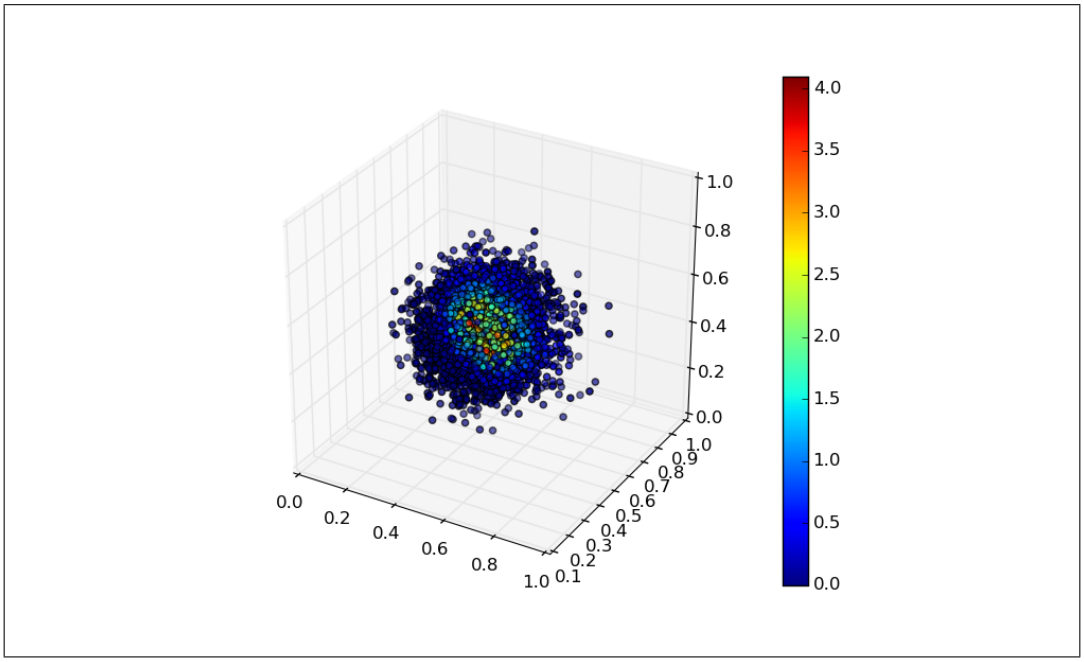


Figure 8: Probability Density Distribution using Sparse Grids

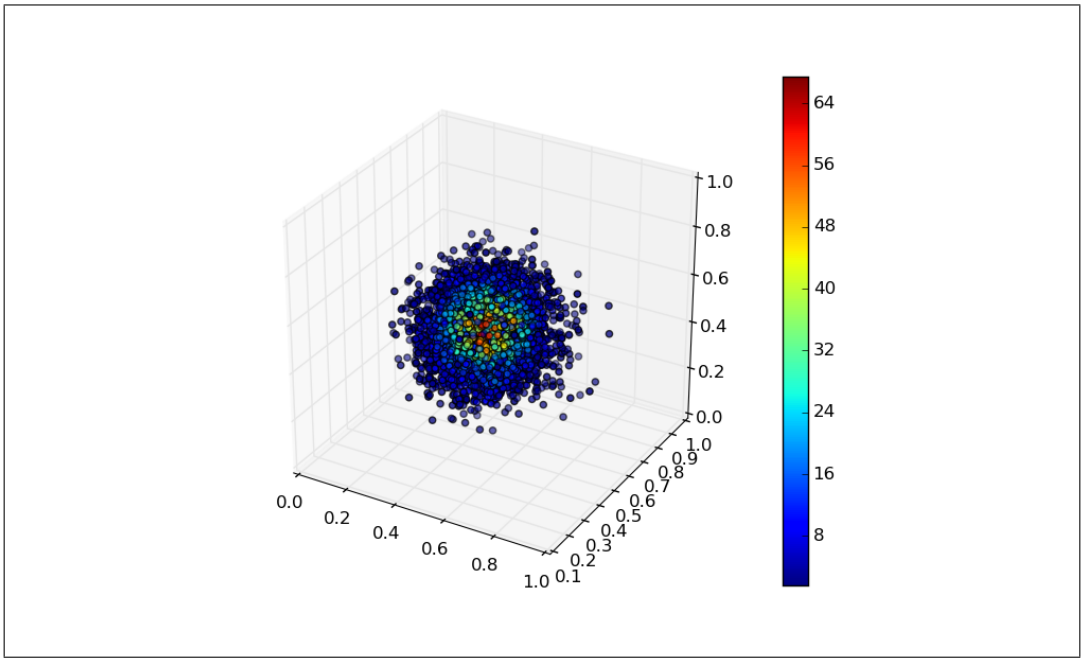


Figure 9: Probability Density Distribution using SciKit KDE

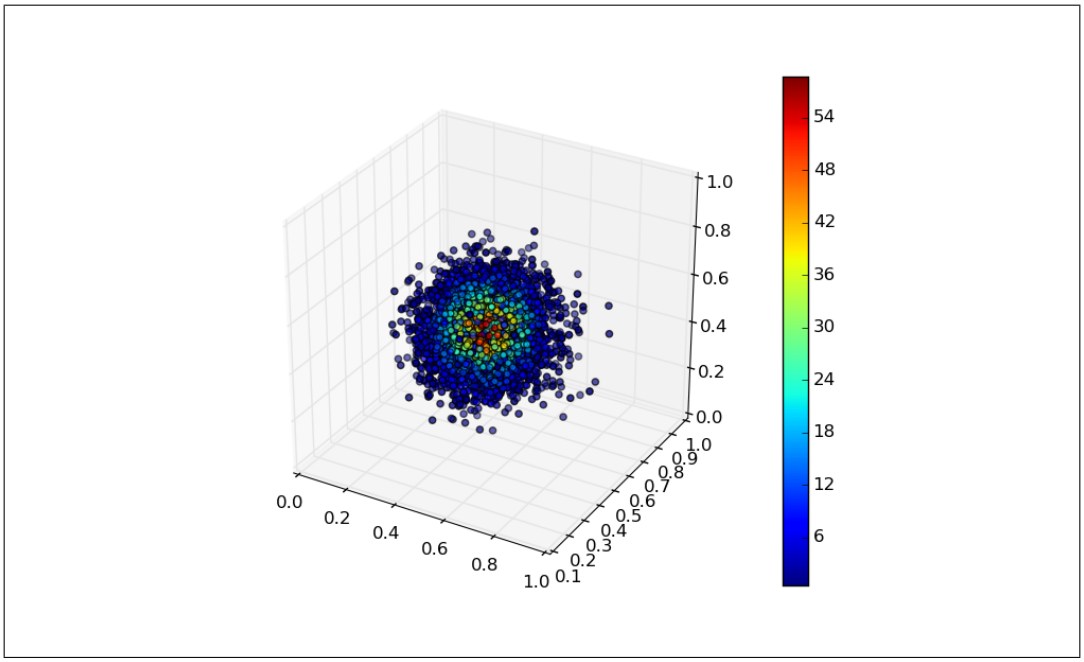


Figure 10: Probability Density Distribution using Scipy KDE

Ripley Garcke DataSet:

Normalised two dimensional data set.

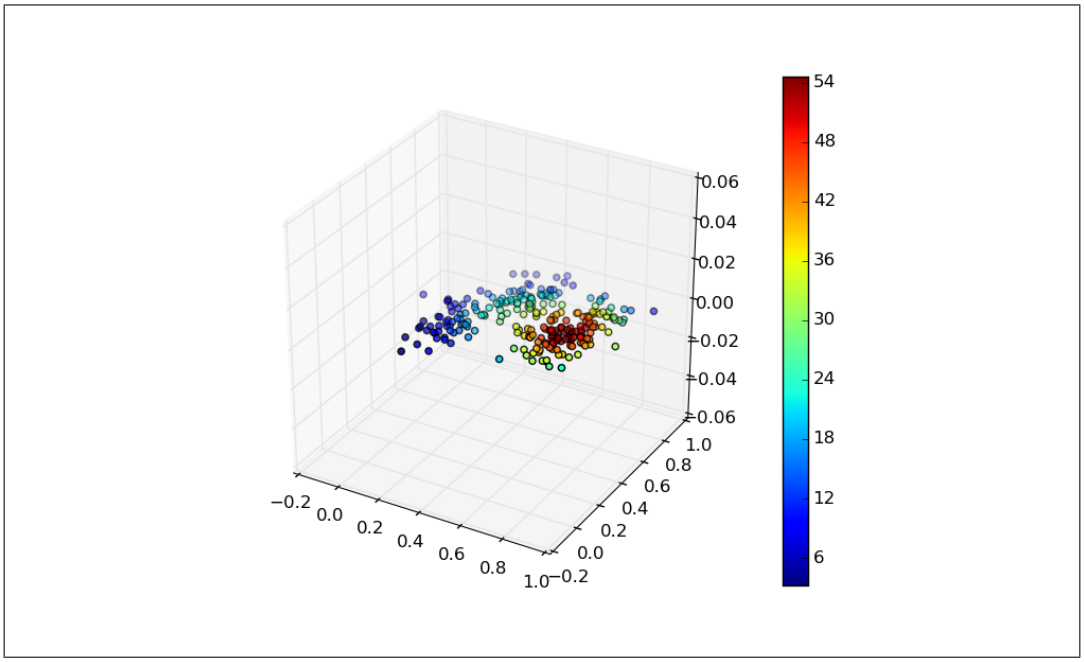


Figure 11: Probability Density Distribution using Sparse Grids

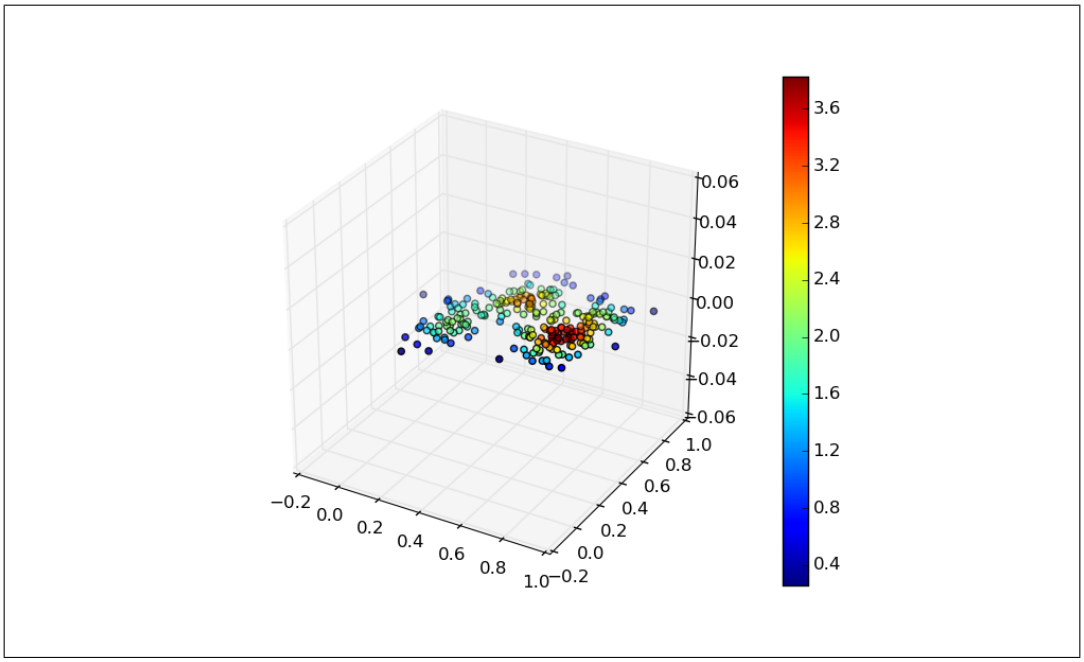


Figure 12: Probability Density Distribution using SciKit KDE

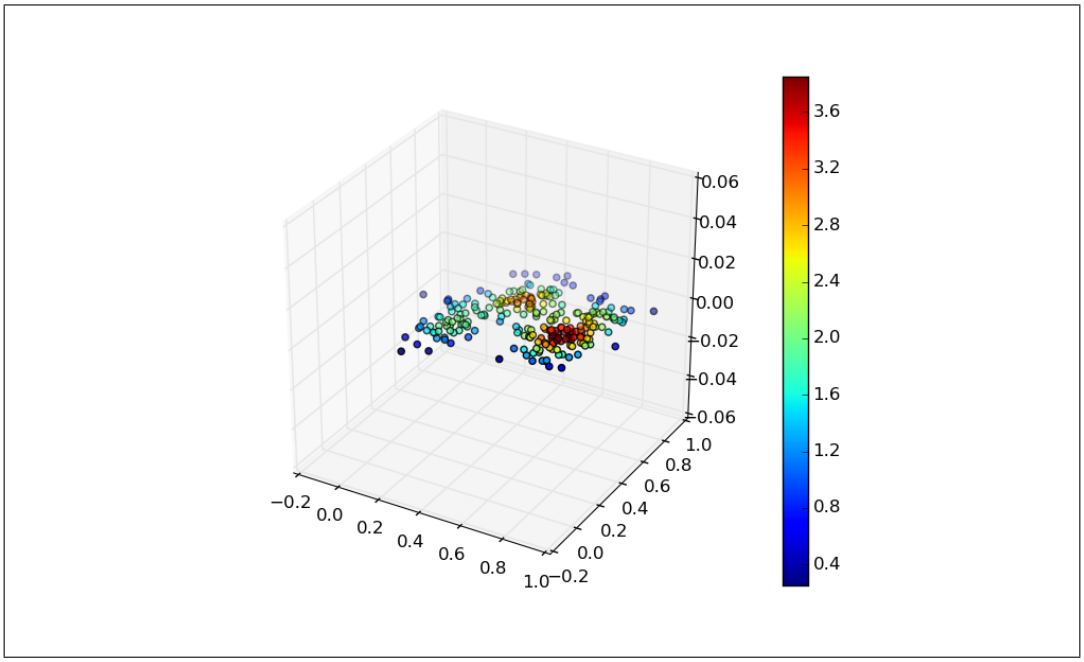


Figure 13: Probability Density Distribution using Scipy KDE

5.2 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.

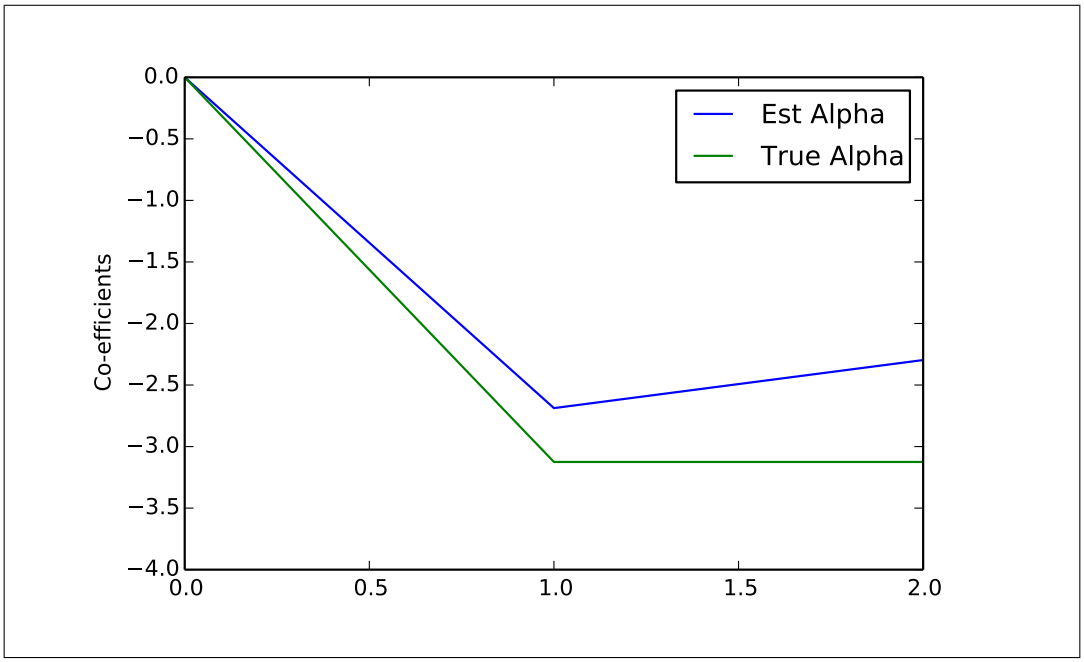


Figure 14: Ture and Estimated values of the Co-efficients of 1D Toy DataSet

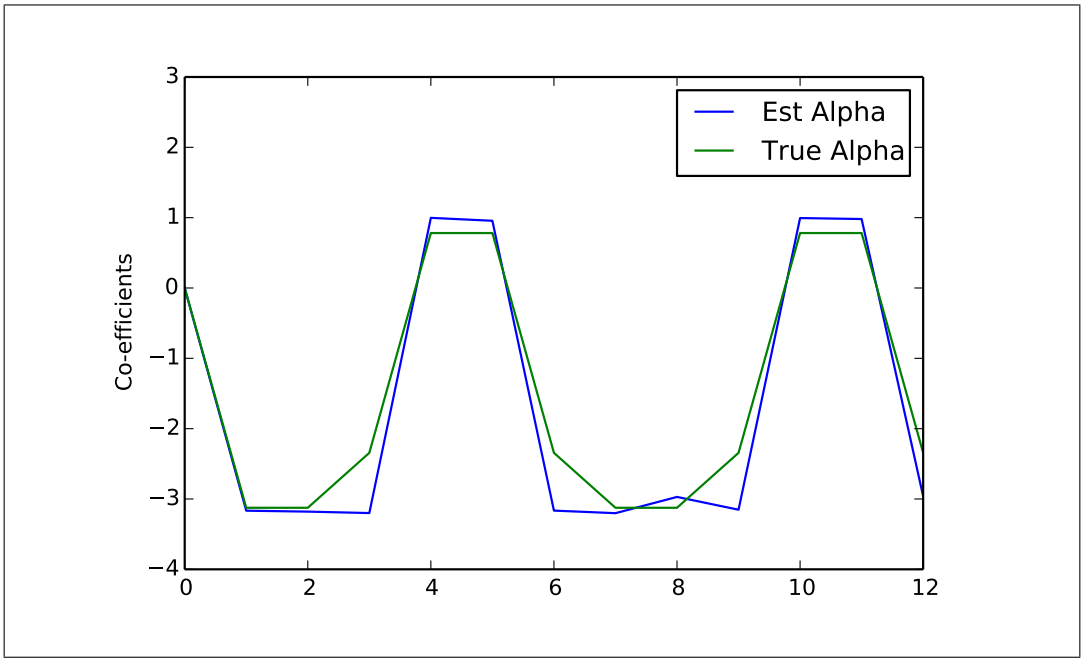


Figure 15: Ture and Estimated values of the Co-efficients of 2D Toy DataSet

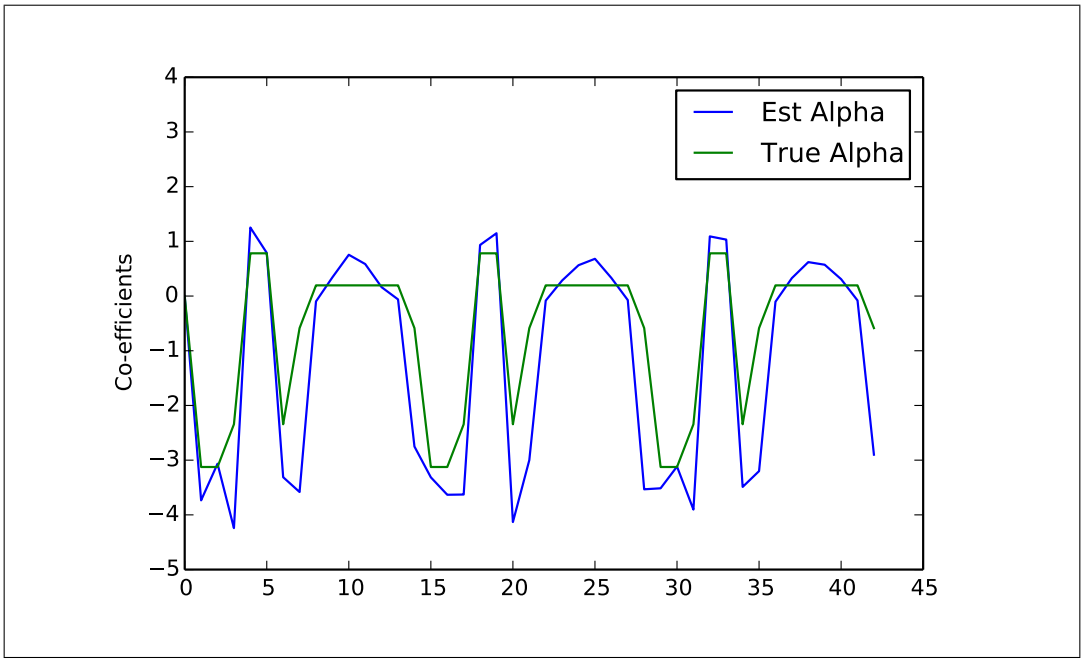


Figure 16: Ture and Estimated values of the Co-efficients of 3D Toy DataSet

Following figure shows the euclidean distance between the means of the true and estimated co-efficient values.

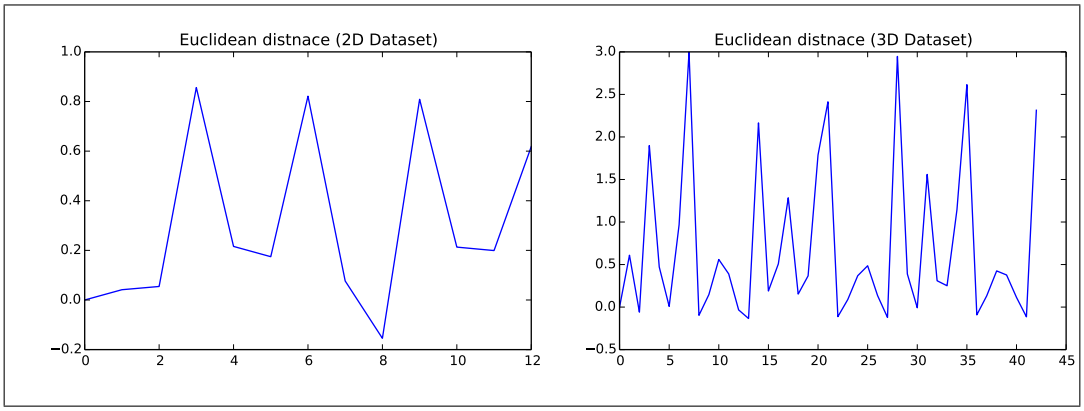


Figure 17: Euclidean distance between the means of the true and estimated co-efficient values.

6 Performance Enhancement

6.1 Learning Rate of Co-efficient Estimation

Even though having an assumed and fixed learning rate for the estimation of co-efficients gave us good results, it would not suit for all kinds of problems and would take more time to converge. In order make the algorithm more efficient and reliable, we employ Armijo type line search to determine the learning rate at every iteration.

6.2 Armijo Line Search

The problem that we are trying to solve is a minimization problem where we are trying to minimize the objective function which is the negative log likelihood of the given dataset. We employ Newton-Raphson procedure for minimizing the objective function. This involves determining the descent direction and the step size of the descent. Once we obtain the descent direction for our objective function, we need to pick the right step size. Taking a step too large might result in the function value being greater than the current value or if the step size is too small it might take forever to converge. Armijo's condition basically suggests that the 'right' step size is such that we have a sufficient decrease in the function value at the new point. This is mathematically represented as follows [3]:
$f(x_k + \alpha p_k) \leq f(x_k) + c_1 \alpha \nabla f_k^T p_k$
where p_k is the descent direction at x_k and $c_1 \in (0,1)$

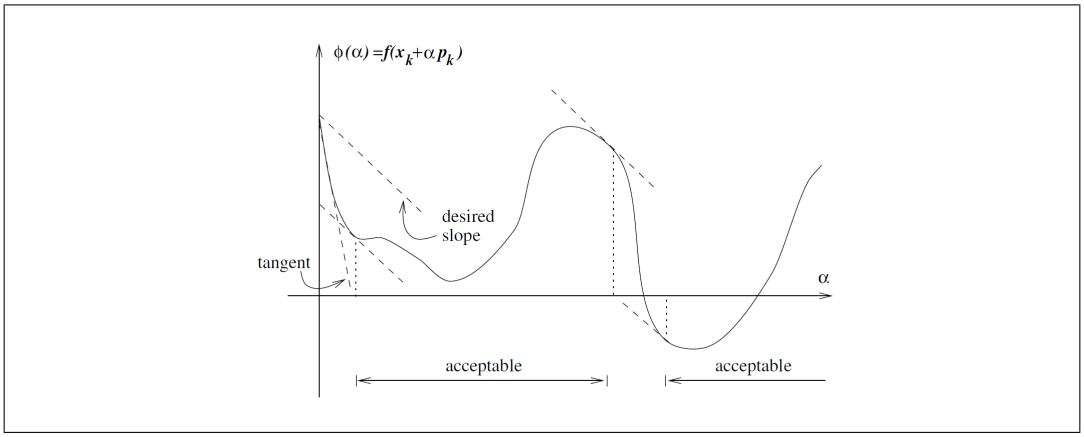


Figure 18: Armijo Condition for choosing the right step size in the descent direction. [3]

Algorithm 4: Updated Algorithm with Dynamic Learning Rate - 'Newton-Raphson' procedure for penalized maximum likelihood

Data: ModLineaerGrid, Co-effecients, Factor graph, Set of DataPoints $X = x_1, .., x_M$
Result: Co-efficients corresponding with each Basis Function
Choose Parameters $\varepsilon > 0$ *and* $i_{max} = Grid\ Size$;
Calculate A;
Compute empirical expected values - E_empir;
Calculate model expected values - E_model $\Phi(\alpha^{initial})$;
Compute $b = E\_empir - E\_model - A.\alpha$;
while *likelihood_grad_norm* > ε *and* i <= i_{max} **do**
    Estimate alpha_direction by solving $A = \tilde{\alpha}b$;
    Calculate learning_rate using Armijo line search;
    $\alpha = \alpha + learning\_rate * alpha\_direction$;
    Calculate model expected values - E_model $\Phi(\alpha^{i+1})$;
    Compute $b = E\_empir - E\_model - A.\alpha$;
    $likelihood\_grad\_norm = \|b\|$;
    Update Factor Graph;
    Coarsen the Grid;
    if *Grid updated* **then**
        Update Co-efficient vector - retaining only non zero values;
        Calculate A;
        Compute empirical expected values - E_empir;
        Calculate model expected values - E_model $\Phi(\alpha^{initial})$;
        Compute $b = E\_empir - E\_model - A.\alpha$;
    end
    $i = i + 1$;
end

6.3 Validation of updated algorithm:

In order to validate the results of the updated algorithm, the estimated co-efficients are again compared with the true co-efficients.
Note: Following experiments were run using factor graph (ANOVA components) with interacting factors of order 2. This results in faster convergence, however the results are not as good as with full factor graphs.

Toy DataSet (4D):

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

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

Parameter	Reg Strategy	Reg Param	Grid Level
Values	laplace	0.1	5

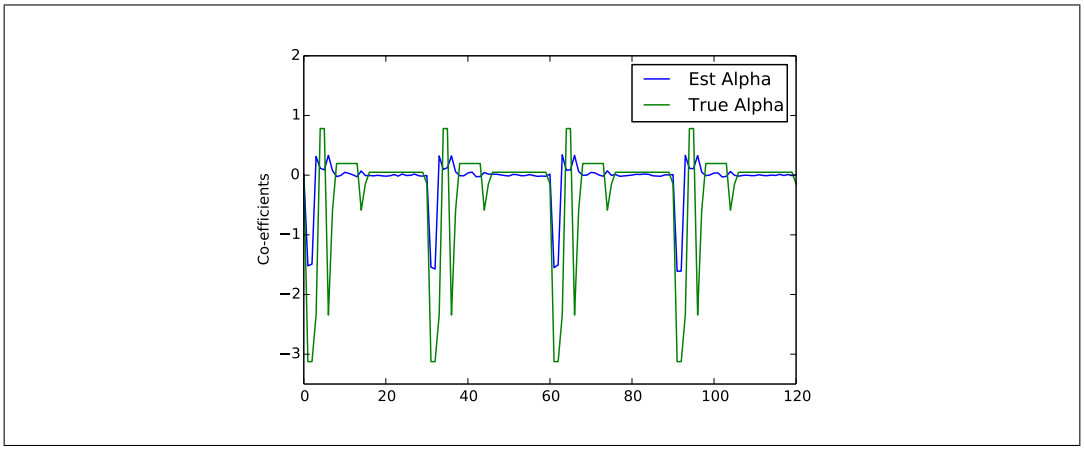


Figure 19: Ture and Estimated values of the Co-efficients of 4D Toy DataSet

Toy DataSet (4D):

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

$$\text{Mean} \begin{pmatrix} 0.5 \\ 0.5 \\ 0.5 \\ 0.5 \end{pmatrix} \text{Co-variance Matrix} \begin{pmatrix} 0.02 & 0 & 0 & 0 \\ 0 & 0.02 & 0 & 0 \\ 0 & 0 & 0.02 & 0 \\ 0 & 0 & 0 & 0.02 \end{pmatrix}$$

Parameter	Reg Strategy	Reg Param	Grid Level
Values	laplace	0.1	5

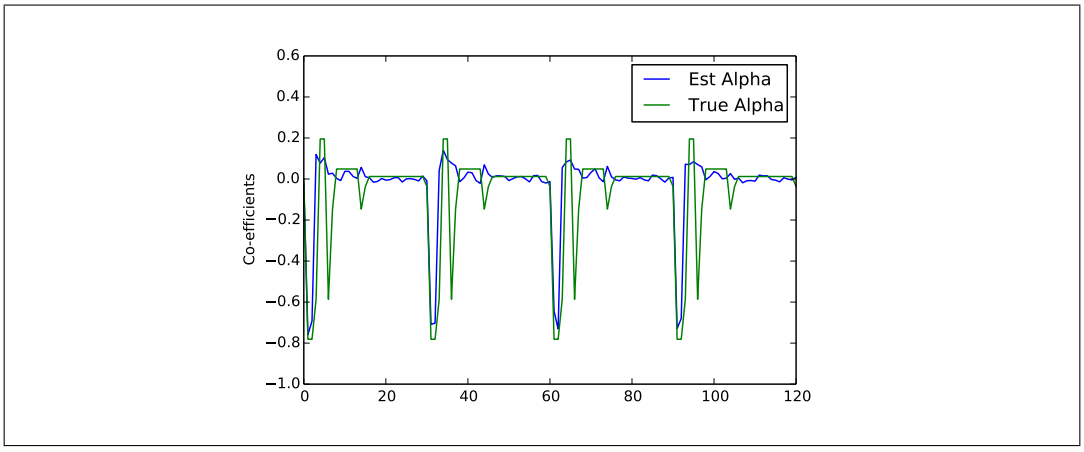


Figure 20: Ture and Estimated values of the Co-efficients of 4D Toy DataSet

Minimizing the objective function

One more way to validate our approach is to verify the reduction in the objective function value. Since our goal is to minimize the objective function, with estimation of the co-efficients at every iteration the likelihood gradient norm should reduce and ultimately reach the minimum. Following experiments confirm the same.

Ripley-Garcke Dataset (2D):

Parameter	Reg Strategy	Reg Param	Grid Level
Values	laplace	0.1	3

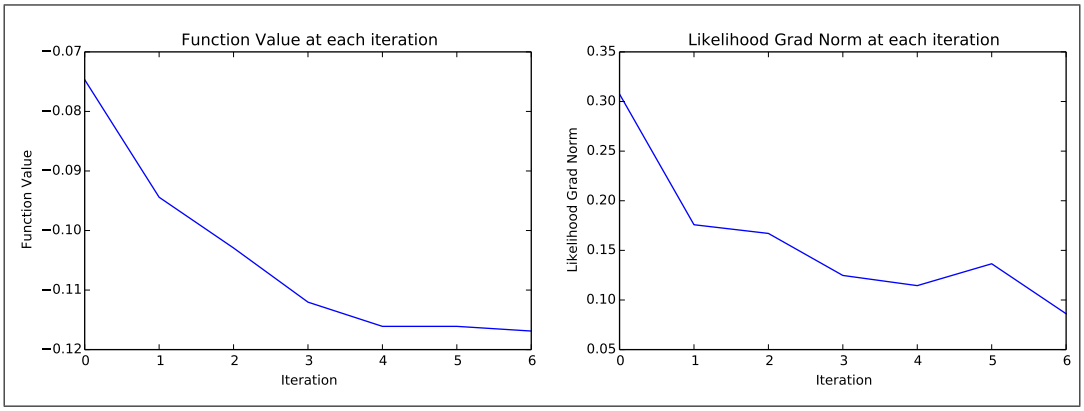


Figure 21: Function Value and Likelihood Gradient Norm per iteration

Bupa Liver Dataset (6D):

Parameter	Reg Strategy	Reg Param	Grid Level
Values	laplace	0.1	3

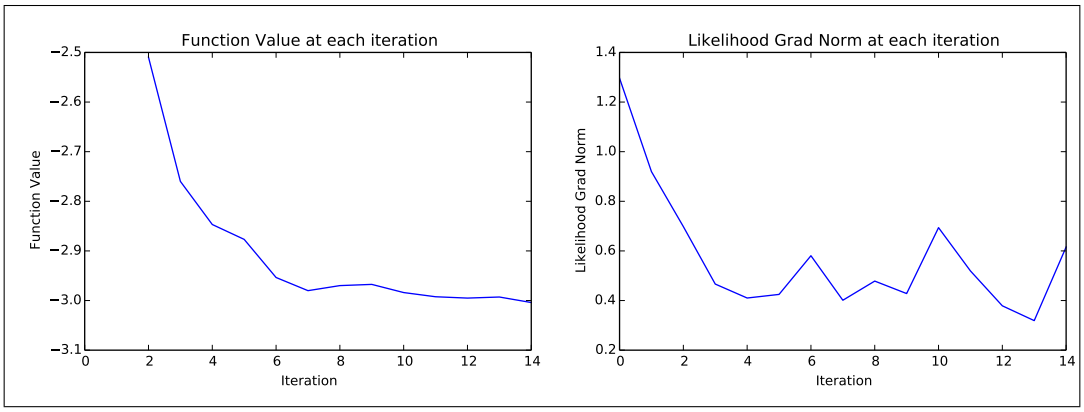


Figure 22: Function Value and Likelihood Gradient Norm per iteration

7 Dependency Modeling and Structure Learning

We use factor graph to model the dependencies among the variables (dimensions). Initially we model the dependency 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-efficients. Since we derive the underlying structure of the dataset, we perform few experiments to validate the performance of our algorithm in structure learning. Different strategies for identifying the unwanted or less important factors are employed and evaluated as follows:

7.1 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 **chosen 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.

7.2 Choosing Co-efficient Threshold

Choosing the threshold as decision measure for throwing away dependencies from the model is non-trivial. However, experiments with few artificial datasets with known co-variance helped in determining a strategy to calculate the threshold. Mean of the absolute values of all the co-efficients, estimated in the very first execution of our algorithm turned to be a good choice. Following are the results of these experiments which clearly demonstrate that the model is capable of determining the dependencies and therefore retain or throw away the factors accordingly.

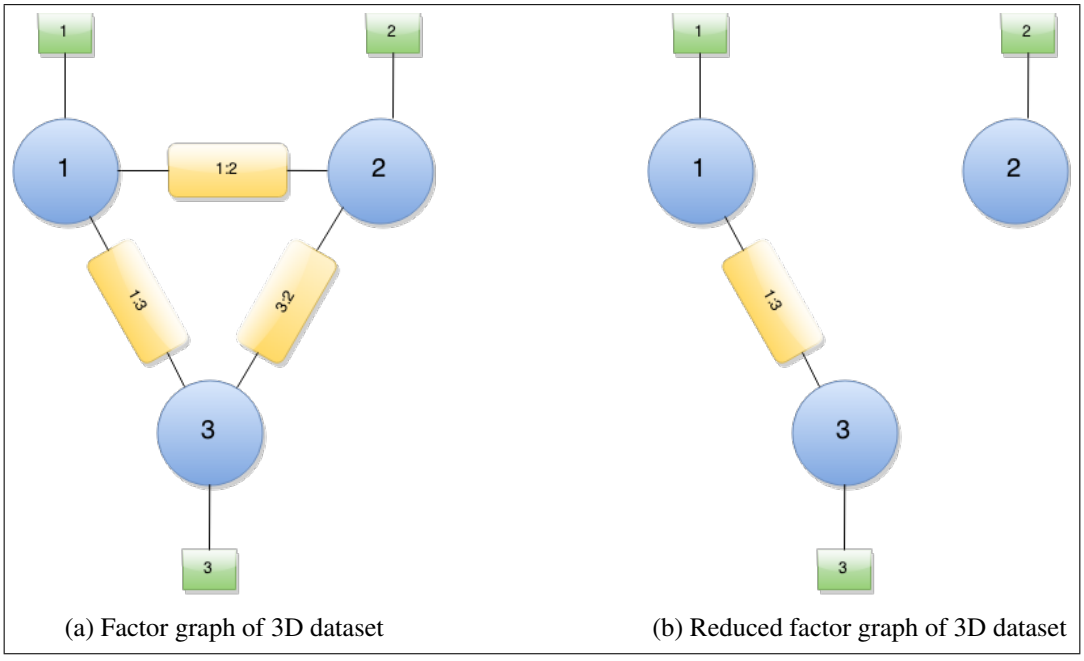
Toy DataSet with Dependencies (3D):

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

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

We run our estimation algorithm with factor of order 2. The following figures show the factor graphs before and after the run. The Co-variance matrix clearly shows that there is a dependency between dimension one and three and the result of the experiment is consistent with this fact.

Parameter	Reg Strategy	Reg Param	Grid Level
Values	laplace	0.1	4



(a) Factor graph of 3D dataset

(b) Reduced factor graph of 3D dataset

Toy DataSet with Dependencies (4D):

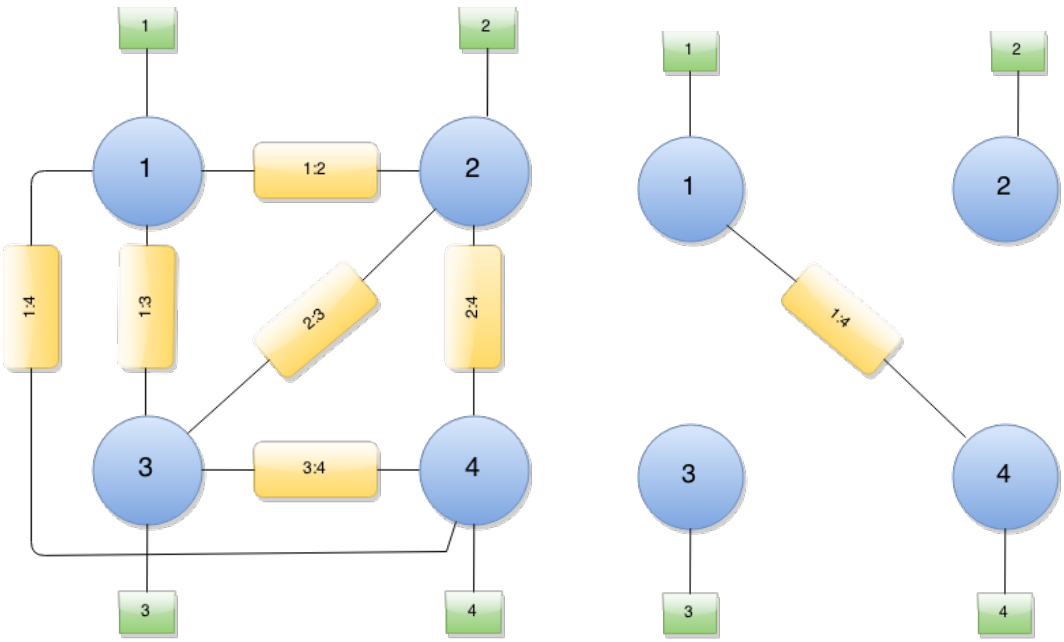
A Toy Dataset generated using Numpy. Normalized four Dimensional Dataset with:

$$\text{Mean} \begin{pmatrix} 0.5 \\ 0.5 \\ 0.5 \\ 0.5 \end{pmatrix} \text{Co-variance Matrix} \begin{pmatrix} 0.02 & 0 & 0 & 0.02 \\ 0 & 0.02 & 0 & 0 \\ 0 & 0 & 0.02 & 0 \\ 0.02 & 0 & 0 & 0.02 \end{pmatrix}$$

We run our estimation algorithm with factor or order 2. The following figures show the factor graphs before and after the run. The Co-variance matrix clearly shows that there is a

dependency between dimension one and four and the result of the experiment is consistent with this fact.

Parameter	Reg Strategy	Reg Param	Grid Level
Values	laplace	0.1	5



(a) Factor graph of 4D dataset (b) Reduced factor graph of 4D dataset

Drawbacks

This strategy has a drawback which is made evident by the following experiment. The model is able to determine only the strongest co-relation but not the weaker ones.

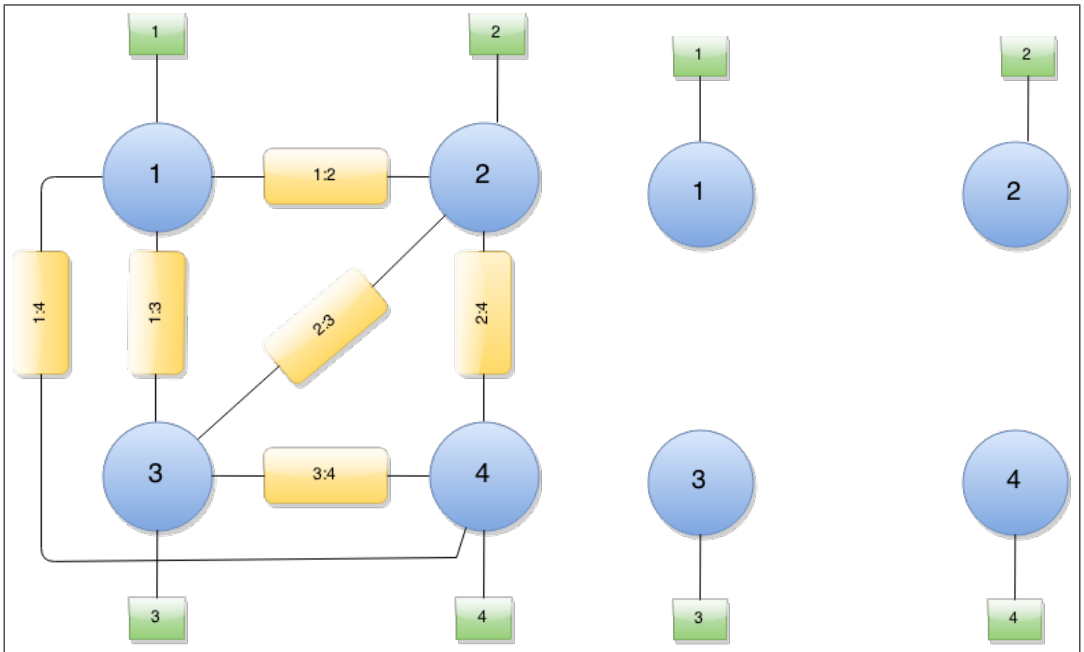
Toy DataSet with Dependencies (4D):

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

$$\text{Mean} \begin{pmatrix} 0.5 \\ 0.5 \\ 0.5 \\ 0.5 \end{pmatrix} \text{Co-variance Matrix} \begin{pmatrix} 0.02 & 0 & 0 & 0.01 \\ 0 & 0.02 & 0 & 0 \\ 0 & 0 & 0.02 & 0 \\ 0.01 & 0 & 0 & 0.02 \end{pmatrix}$$

We run our estimation algorithm using factor graph with interacting factors of order 2. The following figures show the factor graphs before and after the run. The Co-variance matrix clearly shows that there is a dependency between dimension one and four and the result of the experiment is not consistent with this fact since the co-relation is weaker.

Parameter	Reg Strategy	Reg Param	Grid Level
Values	laplace	0.1	5



(a) Factor graph of 4D dataset (b) Reduced factor graph of 4D dataset

Note: One more drawback of this approach is that the algorithm is not able to remove the higher order ANOVA components (greater than 2). Hence the experiments have been run using the factor graph with interacting factors of order 2.

7.3 Updating the Grid

As and when we throw away components from the factor graph / ANOVA components and drive the corresponding co-efficient values to zero, we also need to update the grid accordingly by deleting the corresponding grid points. This reduction in the grid contributes to the improvement in the performance of the algorithm. Entire logic which controls the deletion of the components in the model is laid out in the form of a factor graph and driven using the co-efficient values which corresponds to the components in factor graph. Hence it is possible to make sure that crucial grid points are not deleted when there are higher level grid points present in the grid; which otherwise might lead to inconsistency in the grid.

8 Conclusion and Future Work

Estimation of probability density using Sparse Grids and modeling the dependencies of dimensions using a factor graph which corresponds to the ANOVA decomposition, proved to be successful. There are three main outcomes of this project. First, improvement in the efficiency of the density estimation algorithm by deleting the unwanted ANOVA components from the model without much effect on the accuracy of results. Second, successful use of Markov Chain Monte Carlo sampling in estimating the expected values of sufficient statistics of the model. Third, learning the underlying structure of the dataset.

Next step would be to focus on improving the algorithm to identify the weaker components in the model even when there are higher order ANOVA components (more than two interacting factors). Also, to device a better method to calculate the co-efficient threshold used in co-efficient thresholding process. One more important factor is the regularization factor which had a significant effect on the performance of the algorithm and hence needs to be experimented with.

References

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

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

[3] Stephen J. Wright Jorge Nocedal. Numerical optimization. 1999.

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

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

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