The "Combining Coefficient" for Anisotropic Sparse Grids

John Burkardt
Interdisciplinary Center for Applied Mathematics
& Information Technology Department
Virginia Tech
.....
https://people.sc.fsu.edu/~jburkardt/presentations/sgmga_coefficient.pdf

February 14, 2024

Abstract

The anisotropic sparse grid is a natural extension of the isotropic sparse grid, adapted for situations in which the behavior of the data varies with respect to particular spatial dimensions. An appropriately configured anisotropic sparse grid can achieve accuracy comparable to that of an isotropic sparse grid, while further compounding the reduction in the number of function evaluations that an isotropic sparse grid offers compared to a product rule. To modify an isotropic sparse grid algorithm into an anisotropic one requires changes to the selection criterion and the combining coefficients used in connection with the component product rules. This article discusses these changes and their implementation in a particular computer code.

1 Introduction

Sparse grids are tools for sampling data from a space of high dimension **M**. for the purpose of constructing models, such as a function which interpolates the data, or extracting information, such as estimating the integral of the sampled function, or estimating the argument at which the sampled function attains a minimum value.

Assuming we are free to choose the locations at which the function is sampled, these tasks could be carried out in a natural way by choosing sample points on a product grid, and working with the corresponding interpolant function.

Both the accuracy and the cost of a product rule approach depend on the number of sample points. Unfortunately, the accuracy is related to the accuracy of the 1D factors, while the cost is related to the *product* of the factor costs. Thus, roughly speaking, doubling the accuracy of a product rule increases the cost by the factor 2^M.

In any numerical procedure, it is desirable to have a sequence of approximants of increasing accuracy. However, for a product rule approach to data in a space of even moderate dimension, the cost of successively more accurate product rules explodes exponentially with respect to the dimension.

Sparse grids can be viewed as a way of reorganizing the product rule approach, so that fewer points are used, but the desired accuracies can be reached. A sparse grid achieves this by combining certain low-order product rules that concentrate more efficiently on capturing just the information that is needed to achieve a particular level of accuracy.

An *isotropic* sparse grid is one which is most suitable for problems in which there is no strong dimension-dependent variation in the data; if, on the other hand, a problem is known to exhibit dimension-dependent variation, it is usually desirable, again for reasons of efficiency, to employ an *anisotropic* version of the sparse grid approach.

The formula for the construction of isotropic sparse grids is well known [4]. When moving to an anisotropic formulation, there are many possible approaches. The method employed by Nobile, Tempone, and Webster [2] is framed in such a way that the form of the isotropic construction is preserved, while certain indices and coefficients have been adjusted.

It is the purpose of this note to lay out the isotropic formulations, discuss the changes necessary when moving to the anisotropic version, to compare Clayton Webster's writeup of the anisotropic version of the algorithm (referred to hereafter as "CGW"), against an implementation of that algorithm in a program library called "SGMGA"[1], and to work through a simple example.

2 Isotropic Sparse Grids

We suppose we are interested in computing quadrature rules for integrand functions in a spatial dimension **M**.

We assume that we have an indexed family of 1D quadrature rules $\mathcal{U}^i$. These quadrature rules are indexed by the variable i, which is called the *1D level*. The index i begins at 1, and the corresponding first 1D quadrature rule is often taken to be the midpoint rule. It is understood that as the index i increases, so do both the order (number of points) and the polynomial accuracy of the rule; the exact nature of these relationships is not necessary to specify yet; it is enough to understand that accuracy increases with i.

We can specify a product rule for an **M**-dimensional space by creating a *level vector*, that is, an **M**-vector **i**, each of whose entries is the 1D level which indexes the rule to be used in that spatial coordinate. The resulting product rule may be written as $\mathcal{U}^{i_1} \otimes \cdots \otimes \mathcal{U}^{i_M}$. We say that this product rule has a *product level* of $|\mathbf{i}| = \sum_{k=1}^{M} i_k$. Since the lowest value of a 1D level is taken to be 1, the lowest value of $|\mathbf{i}|$ is M. Thus, the first product rule in **M** dimensions has a product level of **M**, and is formed by **M** factors of the midpoint rule, each having 1D level of 1.

A product rule is a method for approximating an integral; a sparse grid is a refined method for approximating integrals that uses weighted combinations of product rules. A sparse grid rule can be indexed by a variable called the *sparse grid level*, here symbolized by **L**. The lowest value of **L** is taken to be M. The sparse grid is formed by weighted combinations of those product rules whose product level $|\mathbf{i}|$ falls between $L - M + 1$ and L.

The formula for the isotropic sparse grid has the form:

$$\mathcal{A}(L, M) = \sum_{L-M+1 \le |\mathbf{i}| \le L} (-1)^{L-|\mathbf{i}|} \left(\begin{array}{c} M-1 \\ L-|\mathbf{i}| \end{array} \right) (\mathcal{U}^{i_1} \otimes \cdots \otimes \mathcal{U}^{i_M})$$

The condition under the summation sign can be rewritten as

$$0 \le L - |\mathbf{i}| \le M - 1$$

so we select just those **i** for which the combinatorial coefficient makes sense.

The first sparse grid, of sparse grid level $L = M$, will be formed of all product grids with product levels between 1 and M. The lowest possible product level is actually M, and is attained only by the **M**-dimensional product of the midpoint rule (or whatever the first 1D quadrature rule is). So this one-point rule will be the first sparse grid in the series.

Each isotropic sparse grid rule $\mathcal{A}(L, M)$ is a weighted sum of the selected product rules. The selection is based on the product levels, and each weighting coefficient is a power of -1 times a combinatorial coefficient.

If we wish to adapt the isotropic sparse grid formulation so that it can handle anisotropy, the necessary changes can be grouped into two categories, identified by the algorithmic factors that must be modified:

1.) the *selection criterion* for product rules to be combined;

2.) the *combining coefficient* for each selected product rule.

Item 1, the selection criterion, is expressed by the index range in the summation, which is a condition on the product levels of those product rules accepted for the sparse grid. Since we will be discussing this criterion and its variations, let us extract it from the isotropic sparse grid formula:

$$L - M + 1 \leq |\mathbf{i}| \leq L \tag{1}$$

Item 2, the combining coefficient for the isotropic case is a signed combinatorial coefficient; this must be replaced by a new combining coefficient that is consistent with the anisotropic approach.

3 Anisotropic Sparse Grids

In order to develop the anisotropic algorithm, we need to choose some notational conventions. According to the conventions of the "CGW" formulation, we are working in a space of dimension d and we have a set of anisotropy coefficients α which measure in some way the relative importance of the various dimensions. When we wish to describe a sparse grid, we use its sparse grid level index, typically denoted by q. Since the space has dimension d, the lowest order sparse grid rule will have sparse grid index $q = d$. We may find it convenient to consider the *0-based sparse grid index*, which is simply the usual sparse grid index shifted down by d. The 0-based sparse grid index is typically denoted by w; using this convention, the lowest order sparse grid rule, in any dimension, has $w = 0$. The two varieties of sparse grid index, q and w, are simply related by $q = w + d$.

An anisotropic sparse grid is formed as the weighted combination of product grids. Any particular product grid can be represented by $\mathbf{i}$, its d-dimensional *level vector*; that is, for the dimension n, the level vector entry i_n is an index whose values range from 1, 2, ... The value selects the particular quadrature rule $\mathcal{U}^{i_n}$ to be used in the n-th dimension when forming the given product rule. We write the corresponding product rule as

$$\mathcal{U}^{\mathbf{i}} = \mathcal{U}^{i_1} \otimes \cdots \otimes \mathcal{U}^{i_M}$$

The anisotropic sparse grid of 0-based index w, spatial dimension d and anisotropy vector α, is denoted by $\mathcal{A}(w, d, \alpha)$, and has the form:

$$\mathcal{A}(w, d, \alpha) = \sum_{i \in Y_\alpha(w,d)} c_\alpha(\mathbf{i})(\mathcal{U}^{i_1} \otimes \cdots \otimes \mathcal{U}^{i_M})$$

with the combining cofficient defined by

$$c_\alpha(\mathbf{i}) = \sum_{\substack{\mathbf{j} \in \{0,1\}^d \\ \mathbf{i}+\mathbf{j} \in X_\alpha(w,d)}} (-1)^{|\mathbf{j}|}$$

and the selection region of acceptable product rules defined by

$$Y_\alpha(w, d) = X_\alpha(w, d) \backslash X_\alpha(w - \frac{|\alpha|}{\underline{\alpha}}, d)$$

where

$$X_\alpha(w, d) = \{\mathbf{i} \in \mathbb{N}_+^d, \mathbf{i} \geq 1 : \sum_{n=1}^{d} (i_n - 1)\alpha_n \leq w\underline{\alpha}\}$$

with $\underline{\alpha} = \min_{1 \leq n \leq d} \alpha_n$ and $|\alpha| = \sum_{n=1}^{d} \alpha_n$.

(Note that the vector α may be allowed to have 0 entries, but that such values are placeholders, indicating that the corresponding dimension is always to be assigned the lowest possible quadrature rule. Therefore, the computation of $\underline{\alpha}$ should take the minimum only over all *strictly positive* entries of α; we will guarantee that there will always be at least one such strictly positive entry.)

The *selection criterion* given above may be rewritten in a form that is easier to explain. For a product rule to be included in the sparse grid of sparse grid level w, the product rule's level vector $\mathbf{i}$ must satisfy the following condition:

$$w\underline{\alpha} - |\alpha| < \sum_{n=1}^{d} (i_n - 1)\alpha_n \leq w\underline{\alpha} \tag{2}$$

Note here that w is the 0-based sparse grid level, that i_n is a 1-based level index, and that no assumptions are made about the α vector except that all entries are nonnegative and at least one entry is strictly positive.

4 The Combining Coefficient for a Product Rule

When moving from the isotropic to anisotropic formulation, the combining coefficient that multiplies each component product grid is computed in a new way. Instead of being a signed combinatorial coefficient, the combining coefficient for a product grid with level vector $\mathbf{i}$ has the form

$$c_\alpha(\mathbf{i}) = \sum_{\substack{\mathbf{j} \in \{0,1\}^d \\ \mathbf{i}+\mathbf{j} \in X_\alpha(w,d)}} (-1)^{|\mathbf{j}|} \tag{3}$$

where

$$X_\alpha(w,d) = \{\mathbf{i} \in \mathbb{N}_+^d, \mathbf{i} \geq 1 : \sum_{n=1}^{d} (i_n - 1)\alpha_n \leq w\underline{\alpha}\} \tag{4}$$

with $\underline{\alpha} = \min_{1 \leq n \leq d} \alpha_n$ and $|\alpha| = \sum_{n=1}^{d} \alpha_n$.

It turns out that this expression has a fairly straightforward meaning. The combining coefficient $c_\alpha(\mathbf{i})$ is formed by a sum. The sum is over a set of perturbations of the fundamental level vector $\mathbf{i}$. The perturbed level vectors have the form $\mathbf{i}+\mathbf{j}$, where the entries of the $\mathbf{j}$ vector are 0 or 1. A perturbed level vector $\mathbf{i}+\mathbf{j}$ contributes to the sum only if it satisfies the same sparse grid selection constraint that $\mathbf{i}$ does, that is, only if $\mathbf{i}+\mathbf{j}$ is also a component of the sparse grid. In that case, a contribution of $(-1)^{|\mathbf{j}|}$ is added to $c_\alpha(\mathbf{i})$. So generating the combining coefficient is simply a matter of generating all the perturbed vectors, checking whether they satisfy the criterion, and if so incrementing the sum with +1 or -1.

This means, in particular, that the combining coefficient for any product rule is always an integer value. It is quite possible for this value to be exactly zero. As a matter of efficiency, product rules that pass the selection criterion but are then found to have a combining coefficient of zero can be entirely dropped from the combination used to form the sparse grid.

Finally, consider the naive approach to computing the combining coefficient of a single product rule whose level vector is $\mathbf{i}$. Formally, we should generate every perturbed vector $\mathbf{i}+\mathbf{j}$ and determine if it too satisfies the selection criterion. However, in a space of dimension $\mathbf{M}$, there will be 2^M such perturbed vectors to generate and check. For a 10 dimension space, this is about 1,000, for a 20 dimensional space it is about 1,000,000. For higher dimensions, the cost of merely *generating* the perturbed vectors associated with a single combining coefficient will exceed whatever computer time we have available! Therefore, when actually implementing the algorithm, the logically simple approach will have to be replaced by a more complicated approach that avoids this exponential explosion of work.

5 The SGMGA Implementation

The SGMGA program is an attempt to implement the anisotropic sparse grid formulation described in [3]. Because it is a computer program, it can be difficult to identify the correspondence between names and formulas in the references versus the variables and assignment statements in the program. We attempt here to restate some of the previous discussion in such a way as to emphasize the points where the the program

is setting up the selection criterion and the combining coefficient, so that we can examine the details of the implementation.

Again, we begin with some notation, but now we will be using names for things as they are described in the computer program. We assume we are working in a space of **dim_num** dimensions. We allow the user to express the anisotropy of the problem in terms of a vector called **importance**. Presumably, it is more natural to assign a high importance to dimensions in which more variation is going on. Only the relative sizes of the entries of this vector are meaningful. As we will see for the example of Rosenbrock problem, if we wish to indicate that dimension 1 is "twice as important" as dimension 2, we can do so by specifying an **importance** vector of (2,1) ... or (10,5) or any similar set of values in the ratio of 2 to 1.

Note that it might be useful to be able to say that a particular dimension has "0 importance". This would indicate that, while the context of the problem description requires that this dimension be included, the lowest possible quadrature rule can be assigned to this dimension, and never increased. This is another advantage of allowing the user to describe the anisotropy in terms of importance: it is easy to understand what a dimension of 0 importance means.

Internally, the program converts the importance vector to a set of weights for the 1D levels. At the moment, this is done simply by inverting all the nonzero importance entries, and "remembering" that any 0 entries indicate the special case of 0 importance.

The information in the user's **importance** vector will define the anisotropy coefficients, or "level weights" of the level orders. Variables with high importance need a correspondingly small level weight. The user-friendly **importance** vector is therefore converted to a program-friendly vector called **level_weight** by inversion. However, any zero importance entries result in corresponding zero entries in the **level_weight** vector. The implications of zero values in this vector will be handled at the appropriate time.

```
if ( importance}(i) == 0.0 )
  level_weight(i) = 0.0
else
  level_weight(i) = 1.0 / importance(i)
```

The **level_weight** vector may be normalized, if desired, so that the smallest nonzero entry is 1, or so that the entries sum up to **M**. However the program does not assume that any particular normalization has been carried out, and again, it is the relative sizes of the weights that determine the program's actions. The **level_weight** vector in the SGMGA program plays the role of the **alpha** anisotropy vector in the CGW formulation.

It seems more natural to use 0-based indexing for the 1D levels. For instance, that in equation (CGW-17), each 1D level index must be decremented by 1 to form the constraint. Therefore, in the SGMGA program, the 1D level indices begin at 0 rather than 1. When forming a product grid by gathering together a 1D level vector of such indices, the resulting vector is stored in the array called **level_1d**.

To define our families of 1D quadrature rules, we include an implicit assumption about the relationship between the 1D level and the order (number of points) in the corresponding quadrature rule. For many of the rules we consider, this *growth rule* has the form:

$$\mathbf{order\_1d} = 2 * \mathbf{level\_1d} + 1$$

Again, because of 0-based indexing, this means that the rules of levels 0, 1 and 2 will have orders of 1, 3, and 5 respectively. The order of a product rule is then easily determined from the orders of the 1D rules. For example, a 1D level vector of (2,5) corresponds to a 1D order vector of (5,11) which specifies a product rule of order 5 * 11 = 55 points.

(Note that other growth rules are possible. In particular, it is possible to form families of 1D quadrature rules that are *nested*, so that each successive rule includes all the points used by the previous one. Nesting offers some efficiency, because old function values are reused; the drawback is that the growth rule is typically exponential. For example, the Clenshaw Curtis rule begins with a level 0 rule of order 1 (of course), and then for levels 1 and beyond, follows the growth rule:

$$\mathbf{order\_1d} = 2^{\mathbf{level\_1d}} + 1$$

The typical Gauss-Patterson family follows a similar exponential growth rule.)

Now we turn to the formation of a particular sparse grid. A family of sparse grids for dimension **dim_num** will be indexed by a variable which stores the sparse grid level. This variable, elsewhere called **L** or **w**, is called **level_max** in the SGMGA program.

The selection criterion used to determine which product rules will be combined to form a particular sparse grid involves a pair of inequality constraints controlled by the values of two variables called **q_min** and **q_max**. For a candidate product rule, we compute its **q** value as

$$q = \sum_{i=1}^{\text{dim\_num}} \text{level\_weight}(i) * \text{level\_1d}(i) \tag{5}$$

The product grid will be selected to be included in the combination for level **level_max** if its **q** value satisfies

$$\text{q\_min} < q \leq \text{q\_max} \tag{6}$$

(Notice that we must use strict inequality on the left to guarantee a kind of nesting of the selection regions.)

The limits **q_min** and **q_max** are determined by the formulas:

$$\begin{aligned} \text{level\_weight\_min\_pos} = & \min_{0<\text{level\_weight}_i} \text{level\_weight}_i \\ \text{q\_min} = & \text{level\_max} * \text{level\_weight\_min\_pos} - \text{sum}(\text{level\_weight}(:)) \\ \text{q\_max} = & \text{level\_max} * \text{level\_weight\_min\_pos} \end{aligned}$$

Notice that we have preserved the form of the condition used in the isotropic formulation. After all, for the isotropic case, the **importance** vector entries would be equal, which implies that after inversion, we would arrive at a **level_weight** vector whose entries are also equal. Then our anisotropic selection condition, (Equation-6), can be expanded to read

$$\begin{aligned} & \text{level\_max} * \text{level\_weight\_min\_pos} - \text{sum}(\text{level\_weight}(:)) \\ < & \sum_{i=1}^{\text{dim\_num}} \text{level\_1d}(i) * \text{level\_weight}(i) \\ \leq & \text{level\_max} * \text{level\_weight\_min\_pos} \end{aligned}$$

The entries of **level_weight** are all equal, and hence equal to **level_weight_min_pos**. Hence, we can divide through by the common value of **level_weight** to get:

$$\text{level\_max} - \text{dim\_num} < \sum_{i=1}^{\text{dim\_num}} \text{level\_1d}(i) \leq \text{level\_max}$$

Since the sum now only involves integers, we can increment the lower limit by 1, and replace the less-than sign by a less-than-or-equal-to sign. Therefore, if the anisotropic selection criterion is given data that is actually isotropic, the criterion is equivalent to:

$$\text{level\_max} - \text{dim\_num} + 1 \leq \sum_{i=1}^{\text{dim\_num}} \text{level\_1d}(i) \leq \text{level\_max}$$

which, after replacing corresponding variable names, is identical to the isotropic selection criterion, (Equation-1).

6 Computational Details

In this section, we sketch in moderate detail the procedure by which a sparse grid is built up. We are primarily interested in presenting the flow of the code, and pointing out where the selection criterion is made and how the combining coefficient is constructed.

Recall that in the SGMGA program, the spatial dimension is denoted by **dim_num**, the anisotropic coefficients by **level_weight**[], and the sparse grid level by **level_max**.

Based on this input information, the values of **q_min** and **q_max** are determined, as well as the vector **level_1d_max**[], which simply represents the largest value a given component of **level_1d**[] can attain (by forcing all the other entries to 0).

There follows a *for* loop to search for and process all the product rules that participate in the given sparse grid. The search is carried out inside a simplex defined by **level_1d_max**[], and applies the selection criterion on the corresponding value **q**. If the search returns an acceptable level vector, the combining coefficient is computed, and then the contribution of this product grid to the sparse grid can be determined.

In the following pseudocode, there are calls to several functions:

- **r8vec_min_pos()** returns the minimum positive value in a vector;
- **r8vec_sum()** returns the sum of the entries of a vector;
- **sgmga_vcn_ordered()** returns, one at a time, level vectors that satisfy the selection criterion;
- **sgmga_vcn_coef()** computes the combining coefficient;

We have simplified the code and replaced certain items by simple text remarks. However, the general flow of the code is correct. Versions of this procedure, modified for the particular operation being carried out, can be found in the SGMGA functions **sgmga_index**, **sgmga_size**, **sgmga_size_total**, **sgmga_unique_index**, and **sgmga_weight**.

```
//
//  Begin CODE TO GENERATE AN ANISOTROPIC SPARSE GRID.
//
  level_weight_min_pos = r8vec_min_pos ( dim_num, level_weight );
  q_min = level_max * level_weight_min_pos
    - r8vec_sum ( dim_num, level_weight );
  q_max = level_max * level_weight_min_pos;
//
//  LEVEL_1D_MAX defines the simplex of acceptable level vectors.
//
  for ( dim = 0; dim < dim_num; dim++ )
  {
    if ( 0 < level_weight[dim] )
    {
      level_1d_max[dim] = floor ( q_max / level_weight[dim] );
    }
    else
    {
      level_1d_max[dim] = 0;
    }
  }

  Sparse grid is initially empty;
```

```
  for ( ;; )
  {
//
//  Find another level vector in the simplex.
//  The selection criterion is imposed inside this function.
//
    sgmga_vcn_ordered ( );
    if ( no more grids ) break;
//
//  Compute the combining coefficient.
//  If it's 0, we can ignore this level vector.
//
    coef = sgmga_vcn_coef ( )
    if ( coef == 0.0 ) continue;
//
//  The growth rule transforms the level vector to an order vector.
//
    order_nd = level_to_order ( )
//
//  From the order vector, we can compute the product rule.
//
    Compute points and weights of product rule of given order.
//
//  Add the product rule points to the sparse grid point set;
//  Some points might duplicate previous ones.
//
    Add (unique) points to sparse grid points.
//
//  Multiply the product rule weights by the combining coefficient,
//  and associate them with the product rule points you just included.
//
    Add ( coef * product weight ) to sparse grid weights.
  }
//
//  End CODE TO GENERATE AN ANISOTROPIC SPARSE GRID.
//
```

7 Sample Calculations of Combining Coefficients

It is helpful to work through some sample calculations of the combining coefficients for a simple case. In this section, we display the results of computing the combining coefficients for sparse grids of sparse grid levels 0 through 4, in a space of dimension 2. The **importance** vector is taken as (2.0,1.0) and so the **level_weight** vector can be taken to be (1.0,2.0). The linear growth rule is used, so that a level vector (2,1) corresponds to an order vector of (5,3) and a product rule order of 15.

Sparse Grid Level	Level vector	Order vector	Product order	q_min < q ≤ q_max	Combining Coefficient
0				q_min = 0.00	
	(0, 0)	(1, 1)	1	3.00	+1.0
				q_max = +3.00	
1				q_min = 1.00	
	(0, 0)	(1, 1)	(1)	3.00	0.0
	(1, 0)	(3, 1)	3	4.00	+1.0
				q_max = +4.00	
2				q_min = +2.00	
	(0, 0)	(1, 1)	1	3.00	-1.0
	(1, 0)	(3, 1)	(3)	4.00	0.0
	(0, 1)	(1, 3)	3	5.00	+1.0
	(2, 0)	(5, 1)	5	5.00	+1.0
				q_max = +5.00	
3				q_min = +3.00	
	(1, 0)	(3, 1)	3	4.00	-1.0
	(0, 1)	(1, 3)	(3)	5.00	0.0
	(2, 0)	(5, 1)	(5)	5.00	0.0
	(1, 1)	(3, 3)	9	6.00	+1.0
	(3, 0)	(7, 1)	7	6.00	+1.0
				q_max = +6.00	
4				q_min = +4.00	
	(0, 1)	(1, 3)	3	5.00	-1.0
	(2, 0)	(5, 1)	5	5.00	-1.0
	(1, 1)	(3, 3)	(9)	6.00	0.0
	(3, 0)	(7, 1)	(7)	6.00	0.0
	(0, 2)	(1, 5)	5	7.00	+1.0
	(2, 1)	(5, 3)	15	7.00	+1.0
	(4, 0)	(9, 1)	9	7.00	+1.0
				q_max = +7.00	

The constraint has the form $q\_min < q \leq q\_max$. In the table, we include the values of q_min and q_max, along with the value of

$$\mathbf{q} = \sum_{i=1}^{\mathbf{dim\_num}} \mathbf{level\_weight}(i) * \mathbf{level\_1d}(i)$$

for each level vector.

In the final column, we show the combining coefficient, that is, the equivalent, in the anisotropic case, of the signed combinatorial coefficient used in the isotropic case as a weight applied to each component product grid. For the anisotropic case, it is possible for this combining coefficient to come out 0. In that case, the product grid doesn't actually participate at all in the sparse grid, and an efficient algorithm will drop it

from the procedure. For that reason, whenever a product grid has a zero combining coefficient, we have also parenthesized the corresponding product order, to emphasize that these points aren't actually going to be used at all.

The results from the SGMGA code are identical to these computations from the CGW formulas, except for some notational conventions. In particular, in the SGMGA code, the 1D level vectors are 0-based, so that each entry of the CGW 1D level vectors must be reduced by 1 to describe the corresponding SGMGA 1D level vectors. However, the 1D order vectors and the product orders are the same.

The following sequence of "box plots" suggests the way in which the 2D sparse grids for the example problem are constructed. The coordinate system is based on the 1D level vectors. The sparse grid of level 1, for instance, includes just the two vectors (0,0) and (1,0) and you can see just those two highlighted in the corresponding plot. Moreover, as we move from one sparse grid level to the next, some of the level vectors drop out, some are reused, and some new ones are added.

In these plots, the new level vectors have combining coefficient +1, and show up in red. Old level vectors have a combining coefficient of 0 and are shown in yellow, or a combining coefficient of -1 and are shown in blue. For instance, in moving from level 2 to level 3, the blue things disappear, the yellows become blue, the reds become yellow, and we add two new level vectors, (1,1) and (3,0) which show up in red. The progression of these plots also suggest the way in which the user's 2-to-1 importance vector affects the strength of the sparse grid in each dimension.

8 Model Problem

As an example of how an anisotropic sparse grid calculation might be carried out, we consider as a model problem the Rosenbrock function in 2D:

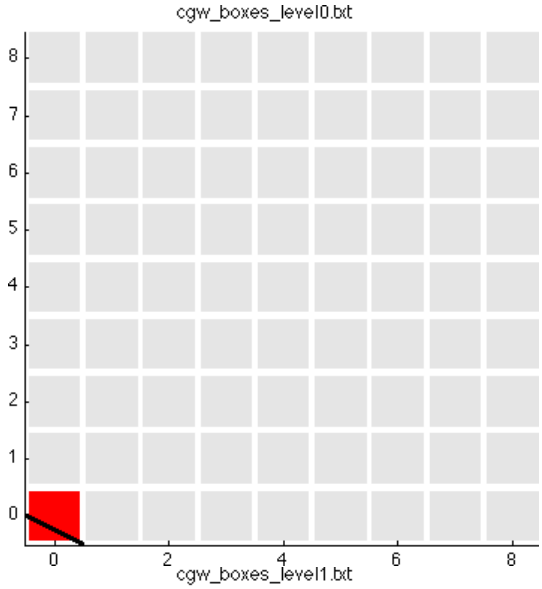
$$f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

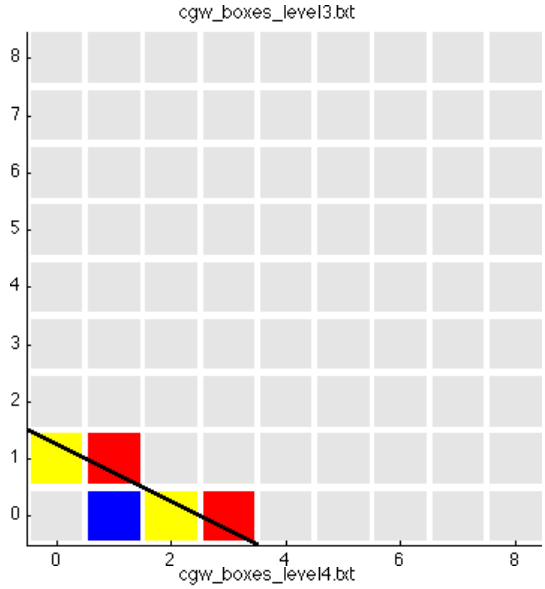
We suppose we arbitrarily set the "importance" of x_1 to be twice that of x_2, since x_1 appears quartically, and x_1 only quadratically.

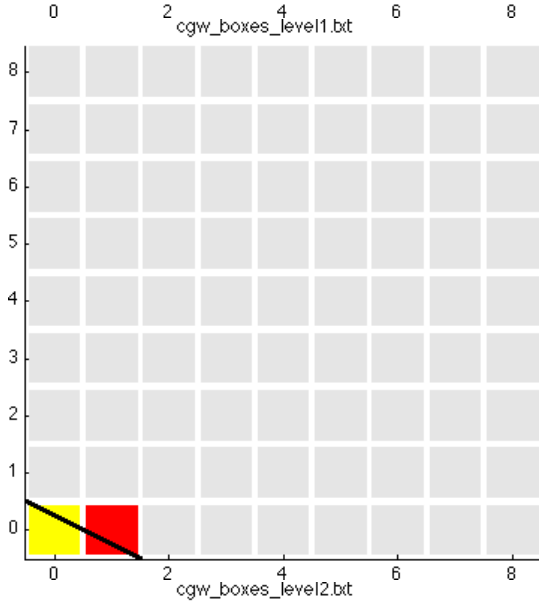
It is no accident that the previous table contains much of the information we will need, since it was calculated for precisely this importance vector.

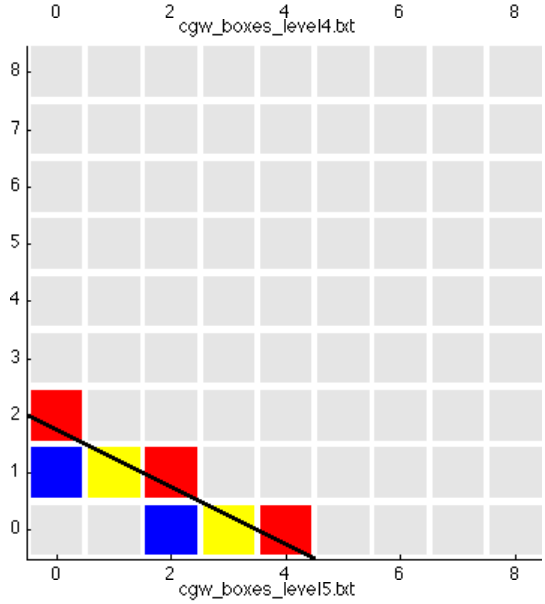
What remains is to specify the underlying 1D quadrature rules. We will suppose, then, that both x_1 and x_2 are to be treated with a Hermite quadrature rule, whose domain is $(-\infty, +\infty)$ and whose weight function is $e^{-\frac{x^2}{2}}$.

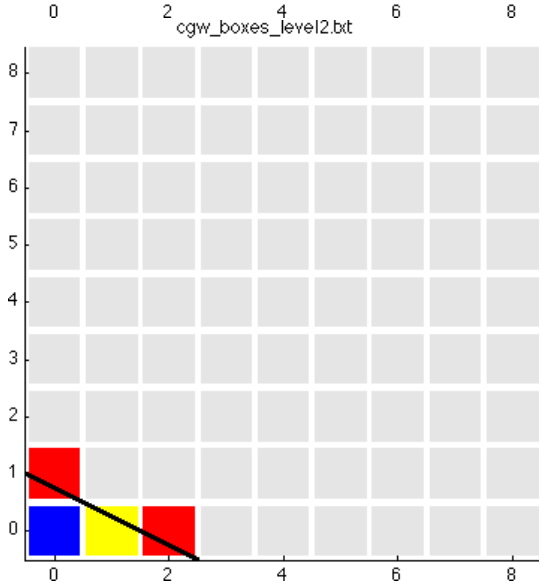
Then the points and weights for the sparse grids of sparse grid levels 0, 1, 2, and 3 will be listed in the following table. Plots of the grids up to level 5 are also displayed.

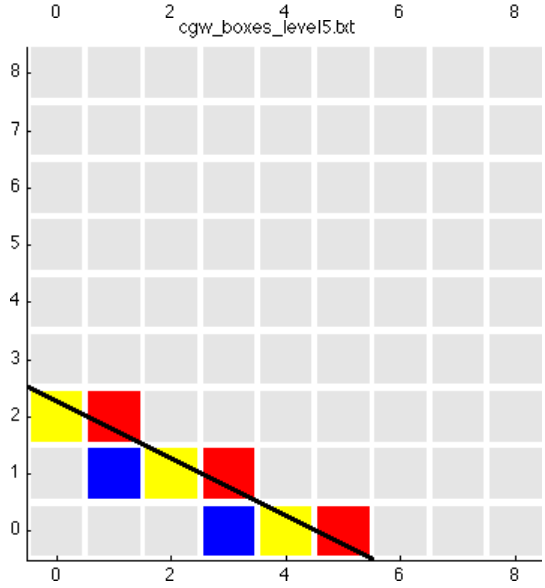

cgw_boxes_level0.txt
0
1
2
3
4
5
6
7
8
0
2
4
6
8


cgw_boxes_level3.txt
0
1
2
3
4
5
6
7
8
0
2
4
6
8


cgw_boxes_level1.txt
0
1
2
3
4
5
6
7
8
0
2
4
6
8


cgw_boxes_level4.txt
0
1
2
3
4
5
6
7
8
0
2
4
6
8


cgw_boxes_level2.txt
0
1
2
3
4
5
6
7
8
0
2
4
6
8


cgw_boxes_level5.txt
0
1
2
3
4
5
6
7
8
0
2
4
6
8

Sparse Grid Level	Point index	X	Y	W
0	0	0.000	0.000	12.566
1	0	-0.707	0.000	-6.283
	1	0.000	0.000	25.132
	2	0.707	0.000	-6.283
2	0	-1.650	0.000	-0.105
	1	-0.524	0.000	-10.366
	2	0.000	-0.707	-6.283
	3	0.000	0.000	46.077
	4	0.000	0.707	-6.283
	5	0.524	0.000	-10.366
	6	1.650	0.000	-0.105
3	0	-2.350	0.000	-0.002
	1	-1.335	0.000	-0.312
	2	-0.707	-0.707	3.141
	3	-0.707	0.000	-6.283
	4	-0.707	0.707	3.141
	5	-0.436	0.000	-13.508
	6	0.000	-0.707	-12.566
	7	0.000	0.000	65.346
	8	0.000	0.707	-12.566
	9	0.436	0.000	-13.508
	10	0.707	-0.707	3.141
	11	0.707	0.000	-6.283
	12	0.707	0.707	3.141
	13	1.335	0.000	-0.312
	14	2.350	0.000	-0.002

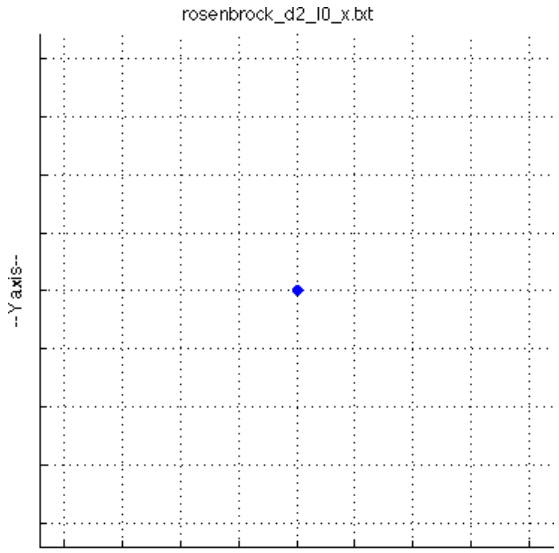
The following images depict the computed sparse grids for levels 0 through 5. Because the Hermite rule was chosen, which is not a nested family, the arrangement of points along lines represents several superimposed Hermite rules.

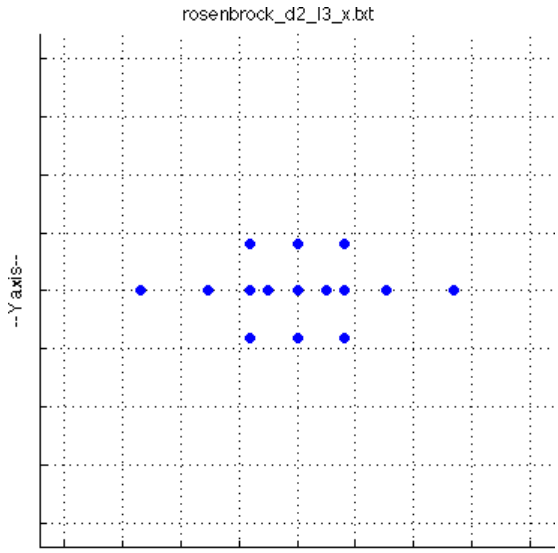
9 Exponential Cost of the Computation

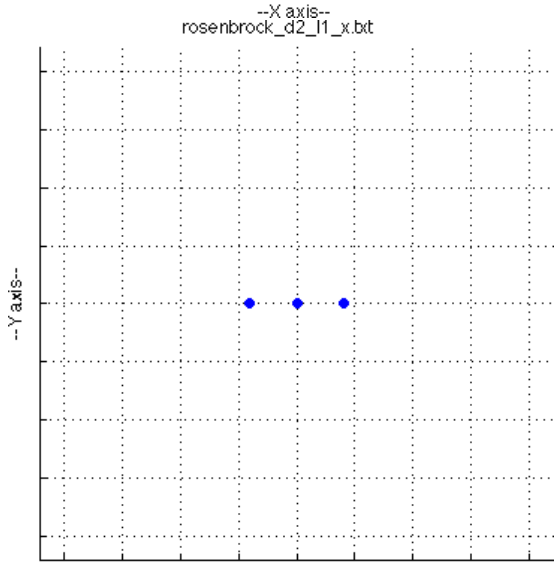
The simplicity of the formula [equation 3] for the combining coefficient is counterbalanced by the fact that the index on the summation indicates that we must consider all possible increments of the base level vector **level_1d** by a nonzero binary vector **j**. A naive algorithm would proceed to generate every such increment and evaluate the constraints. The cost of calculating a single combining coefficient of the sparse grid would then be comparable to the cost of evaluating the product grid, so that the sparse grid approach would offer no improvement whatsoever.

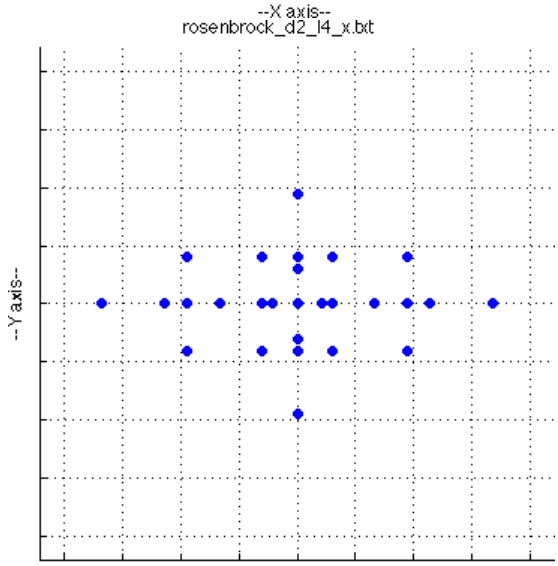
In most cases of interest, the values of the anisotropic weight vector and the level are such that the total number of level vectors satisfying the level constraints is very small. Hence, if we ask what subset of those solutions can be reached by a binary perturbation of some particular solution, the count must be even smaller, and the upper limit of $2^d - 1$ is an absurd overestimate. There is hope, then, that at least for modest values of the level, we can replace the naive approach by an intelligent, efficient search procedure.

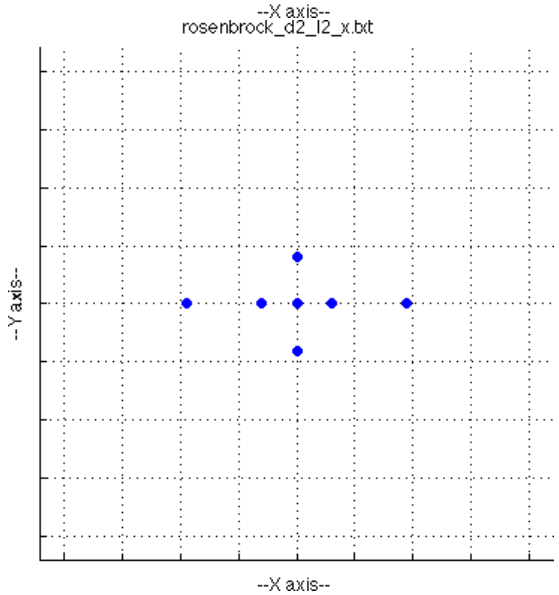
The devising, analysis, implementation and benchmarking of such a procedure is left to an ensuing document.

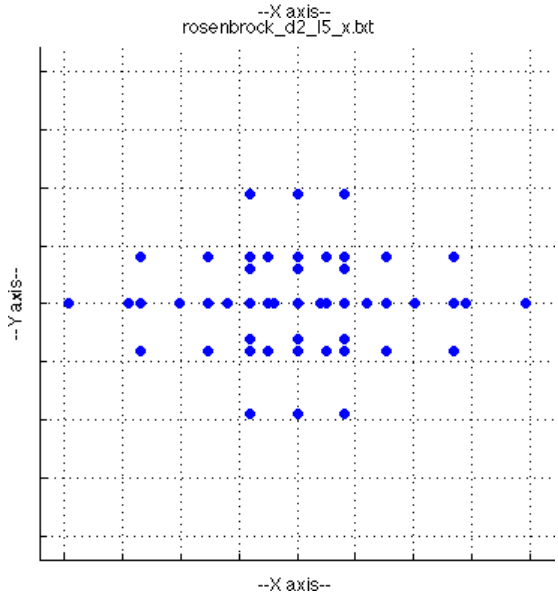

rosenbrock_d2_l0_x.txt
--Y axis--
--X axis--


rosenbrock_d2_l3_x.txt
--Y axis--
--X axis--


rosenbrock_d2_l1_x.txt
--Y axis--
--X axis--


rosenbrock_d2_l4_x.txt
--Y axis--
--X axis--


rosenbrock_d2_l2_x.txt
--Y axis--
--X axis--


rosenbrock_d2_l5_x.txt
--Y axis--
--X axis--

References

[1] John Burkardt, SGMGA: Sparse Grid Mixed Growth Anisotropic Rules, http://people.sc.fsu.edu/~jburkardt/cpp_src/sgmga/sgmga.html.

[2] Fabio Nobile, Raul Tempone, Clayton Webster, A Sparse Grid Stochastic Collocation Method for Partial Differential Equations with Random Input Data, SIAM Journal on Numerical Analysis, Volume 46, Number 5, 2008, pages 2309-2345.

[3] Fabio Nobile, Raul Tempone, Clayton Webster, An Anisotropic Sparse Grid Stochastic Collocation Method for Partial Differential Equations with Random Input Data, SIAM Journal on Numerical Analysis, Volume 46, Number 5, 2008, pages 2411-2442.

[4] Sergey Smolyak, Quadrature and Interpolation Formulas for Tensor Products of Certain Classes of Functions, Doklady Akademii Nauk SSSR, Volume 4, 1963, pages 240-243.