Counting Abscissas in Sparse Grids

John Burkardt
Department of Scientific Computing
Florida State University
.....
http://people.sc.fsu.edu/~jburkardt/presentations/sgmga_counting_2010_fsu.pdf

March 7, 2024

Abstract

Sparse grids are constructed as logical sums of product grids. In high dimensions, sparse grids can be more efficient than product grids, and more rapidly converging than Monte Carlo methods, for problems involving interpolation, integration or optimization. The standard measurement of efficiency is the number of abscissas at which the underlying function is to be evaluated. Determining the abscissa count for a sparse grid is not a straightforward task, particularly because the details vary depending on the underlying 1D rules used as factors. This article develops a general counting method and formulas for specific grid types.

1 Introduction

The sparse grid construction of Smolyak [**?**] produces a quadrature rule for a function of a multidimensional argument, by constructed a weighted sum of product rules; these product rules are, in turn, formed by the product of 1D quadrature rules selected from some family.

Common choices for these 1D rules are discussed in [**?**]. We are free, if we choose, to specify a different family of rules for each dimension. We may imagine that the family is some general class of quadrature rules, but it is important, for the Smolyak procedure, that we specify an indexing procedure, that is, a rule that selects, for each nonnegative index i, a particular quadrature rules Q^i from the family. Then, having specified the indexing procedure, there are several properties of the quadrature sequence that we may wish to understand as functions of the index i:

- the **order** (number of points);
- the **exactness** (exact integration for polynomials up to some degree);
- the **nestedness** (reuse of points);

When a sparse grid is constructed using the Smolyak procedure, a set of multidimensional product rules is generated from the 1D rules, and the sparse grid is the logical sum of these rules. Depending on the underlying 1D rules, it often happens that a given multidimensional abscissa may be generated more than once, as it is part of several product rules.

As far as the logical sparse grid is concerned, repeated abscissas do not cause any problems. Integrals can still be approximated correctly, for instance. However, since the number of abscissas is a measure of the efficiency of the rule, it can be very advantageous to identify repeated abscissas, so that they are replaced by a single copy. In the case when an integral is to be approximated, the corresponding weights are simply summed and associated with the single representative.

In the following discussion, it will be assumed that this reduction of multiple abscissas is to take place before the count is made. Determining how many such multiple abscissas have been generated can be a considerable task in the counting process.

2 Terminology

For clarity, we recall some items of terminology here.

A quadrature rule, defined on an interval $[a, b]$, is called a *closed rule* if the endpoints a and b are included as abscissas in the rule. A quadrature rule which includes neither a or b is called an *open rule.* Although we will not encounter them in this discussion, a rule which includes just one of the endpoints is known as a *half-open rule.* Originally, the distinction was made because open rules (such as those in the Gauss-Legendre family) could be used to avoid endpoint singularities in the integrand. Now the distinction is of some interest because the pattern of growth in a family of quadrature rules depends in part on whether the rules are open or closed.

The midpoint rule is obviously an open rule. However, for various reasons it is useful to include it as the first element of several sequences that are otherwise closed, such as the Newton Cotes Closed family and the Clenshaw Curtis family. This minor contravention of the definition will cause no trouble, and in general, every 1D family used for sparse grids must start with a one-point rule, which is nearly always taken as the midpoint rule.

A *family of quadrature rules* or "family of rules" or simply "family" is an indexed set of quadrature rules Q^k. It is a matter of taste whether to start the index k at 0 or 1, but for this discussion, the first index will always be 0. The *order* of a quadrature rule is the number of abscissas it uses; the midpoint rule has order 1. We can indicate this value by $o(k)$. The *exactness* of a quadrature rule is the highest value e such that the rule can compute exactly the integral of every monomial from x^0 to x^e, and we may indicate this value by $e(k)$. By linearity of integration and quadrature, a rule of exactness e can integrate exactly any polynomial of degree e or less. It is assumed that the rules in a given family increase (or at least, don't decrease!) in order and exactness as the index k increases.

A family of quadrature rules is *fully nested* if, for every index k, all the abscissas in rule k are included as abscissas in rule $k + 1$. A family is *non-nested* if, for every k, *no* abscissa in rule k is included in rule $k + 1$; it is typically the case that *no* abscissa in rule k is included in *any* subsequent rule. Actually, this is a fairly strong condition; we really only mean that abscissas are almost never repeated, and not repeated in a way that is regular enough to take advantage of. Finally, a family is *weakly nested* if, for every index k, a small but nonzero subset of the abscissas of rule k are included in rule $k + 1$. For our purposes, this behavior will actually mean that exactly *one* abscissa value is included in *all* the rules, as typified by the value 0.0 in the Gauss Legendre Linear Growth family.

Smolyak's sparse grid construction rule allows for many variations. The integration region must be a product region, but the factors in each dimension may be different. Correspondingly, a different quadrature family could be associated with each dimension, and each dimension could be assigned a different importance using a weighting function. But thoughout this discussion, we will assume the simplest version of the Smolyak procedure, in which the integration region is an M-dimensional product of a single 1D region, and one quadrature family is used to produce factors in every dimension, and all dimensions are treated equally.

3 Closed Fully Nested Family, Exponential Growth "CFN_E"

We begin by considering families of 1D quadrature rules which are closed and fully nested. Two particular instances are Newton Cotes Closed (NCC) quadrature rules, and Clenshaw Curtis (CC) quadrature rules, both of which estimate integrals $I(f)$ by specifying n points x and weights w to make an estimate $Q(f)$ of the following form:

$$I(f) = \int_{-1}^{+1} f(x)dx \approx Q(f) = \sum_{i=1}^{n} w_i f(x_i)$$

As it happens, the NCC rules, which use equally spaced points, are numerically unstable, and so this section will focus on CC rules, which are a standard tool for reliable integral estimates.

In general, a rule with order n will have exactness $e = n - 1$, because there are n degrees of freedom in the quadrature weights w, and n distinct monomials to integrate. However, because of symmetry, a CC rule of odd order n has degree of exactness $e = n$, that is, it can integrate exactly the $n + 1$ monomials $1, x, x^2, \ldots, x^n$. (In fact, it can integrate exactly *any* monomial of odd degree, but the fact that it can integrate x^{n+2} is not useful because it can't handle x^{n+1}, and the lowest degree monomial that we can't handle is that one that determines the error behavior.

To construct a quadrature family, we need to select a sequence of CC rules. It is common to choose the rules in such a way that the sequence of rules is nested, that is, that the abscissas of each rule are included in the next. The standard way of doing this starts with a 1 point rule with abscissa 0.0, then moves to the 3 point rule with abscissas -1, 0 and 0. The next rule inserts one new abscissa between each existing pair, resulting in a rule of order 5. Continuing this pattern produces a family of Clenshaw Curtis rules with the following *order vector*:

$$\text{order\_1d} = \{1, 3, 5, 9, 17, 33, 65, 129, 257, 513, 1025, \ldots\}.$$

The order of the rules is exponential, essentially doubling with each increase in the index. Therefore, we denote this particular family as the CC_E family, that is, Clenshaw Curtis with exponential growth. A corresponding family of NCC rules could be generated, which would be designated NCC_E, and the generic designation for families with this behavior is CFN_E, that is, closed and fully nested, with exponential growth.

The orders have been chosen so that the resulting family is fully nested. If we let the index be l, and count from 0, then with the exception of the initial rule, the order of the rule of index l is $o(l) = 2^l + 1$. This kind of growth is known as *exponential*, and means that by level 10 we are using about 1,000 points. Since the orders are odd, the exactness of each rule in the family is equal to its order: $e(l) = 2^l + 1$.

As a companion to the order vector, we may also derive a vector *new_1d* which counts the number of points which do not appear in the rule of level $l - 1$, but do appear in the rule of level l. It is easy to see that for the CFN_E family, this vector is:

$$\text{new\_1d} = \{1, 2, 2, 4, 8, 16, 32, 64, 128, 256, 512, \ldots\},$$

so that with the exception of the first two indices, the number of new points added to compute the rule of index l is 2^{l-1}.

Now we are ready to try to count the points in a sparse grid formed using a CFN_E family. The formula for the point count takes advantage of the nestedness of the CFN_E family. To construct a sparse grid in dimension M and level L, Smolyak's formula tells us to combine product rules whose product levels lie in the limited range between $\max(0, L - M + 1)$ and L. When the underlying family is nested, and we are only interested in what points are included in the sparse grid, we can think of the sparse grid as being formed from the *entire range* of product rule levels, 0 to L. The advantage to thinking this way is that we can now logically assign each point in the final grid a time of “first appearance” in the grid. We do this by ordering the product rules by their product rule level (that is, the sum of the levels of the 1d rules used to form the product rule). For each point in the grid, there is a minimum product rule level at which it appears, and for that level it appears in exactly one product grid. This means that one way to count all the points in the CFN_E sparse grid is to count only the first appearances of points in the component product grids.

But this turns out to be easy. Any component product rule is defined by its level vector, **level_1d**, which in turn gives us a decomposition of that product grid into its 1D factors. The number of points that make their first appearance in this grid is simply

$$\text{new}(\text{level\_1d}) = \prod_{i=1}^{M} \text{new\_1d}(\text{level\_1d}(i))$$

and therefore the number of points in the sparse grid is

$$\text{order}(L, M) = \sum_{l=0}^{L} \sum_{|level\_1d|=l} \prod_{i=1}^{M} \text{new\_1d}(\text{level\_1d}(i))$$

which is the sum, over all product rule levels l from 0 to L, of the sum over all product grids of level exactly l, of the number of points which first appear in that product grid.

For a 2D grid, the result can easily be worked out by performing what amounts to a convolution on the **new_1d** vector. For instance, the order of the CFN_E sparse grid rule of level 3, dimension 2 is

$$\begin{aligned}\text{order}(3, 2) &= 1 * 1\\ &+ (2 * 1 + 1 * 2)\\ &+ (2 * 1 + 2 * 2 + 1 * 2)\\ &+ (4 * 1 + 2 * 2 + 2 * 2 + 1 * 4)\\ &= 29\end{aligned}$$

For 2D, the procedure can also be visualized in a tabular array. Here we compute the number of points in a 2D CFN_E sparse grid of level 5.

L	New						
5	16	16					
4	8	8	16				
3	4	4	8	8			
2	2	2	4	4	8		
1	2	2	4	4	8	16	
0	1	1	2	2	4	8	16
	New	1	2	2	4	8	16
	L	0	1	2	3	4	5

This table shows the number of points that have their “first appearance” in each of the product rules that is used to build the sparse grid. Totalling the values inside the box produces 145, the number of points in the sparse grid.

For the general case, the count is easy to program, as long as we have some procedure for generating all possible level vectors that have a given level. Here is the body of a C++ function that computes **order**, the order of a sparse grid. It is to be understood that the function **comp_next** produces, one at a time, the level vectors **level_1d** that sum to l.

```
new_1d = new int[level+1];
new_1d[0] = 1;
new_1d[1] = 2;
j = 1;
for ( l = 2; l <= level; l++ )
{
  j = j * 2;
  new_1d[l] = j;
}
level_1d = new int[m];
order = 0;
for ( l = 0; l <= level; l++ )
{
  more = false;
  for ( ; ;)
  {
    comp_next ( l, m, level_1d, &more );
    v = 1;
    for ( dim = 0; dim < m; dim++ )
    {
      v = v * new_1d[level_1d[dim]];
    }
    order = order + v;
    if ( !more )
    {
      break;
    }
  }
}
```

Novak and Ritter [?] computed and displayed the point counts for sparse grid rules based on the standard Clenshaw Curtis family, for levels 1 through 8, and dimensions 5, 10, 15, 20 and 25. Here, we present results for dimensions 1 through 10 and levels 0 through 10. Where they overlap, the tables agree with Novak and Ritter.

DIM:	1	2	3	4	5
CFN_E LEVEL					
0	1	1	1	1	1
1	3	5	7	9	11
2	5	13	25	41	61
3	9	29	69	137	241
4	17	65	177	401	801
5	33	145	441	1,105	2,433
6	65	321	1,073	2,929	6,993
7	129	705	2,561	7,537	19,313
8	257	1,537	6,017	18,945	51,713
9	513	3,329	13,953	46,721	135,073
10	1,025	7,169	32,001	113,409	345,665

DIM:	6	7	8	9	10
CFN_E LEVEL					
0	1	1	1	1	1
1	13	15	17	19	21
2	85	113	145	181	221
3	389	589	849	1,177	1,581
4	1,457	2,465	3,937	6,001	8,801
5	4,865	9,017	15,713	26,017	41,265
6	15,121	30,241	56,737	100,897	171,425
7	44,689	95,441	190,881	361,249	652,065
8	127,105	287,745	609,025	1,218,049	2,320,385
9	350,657	836,769	1,863,937	3,918,273	7,836,545
10	943,553	2,362,881	5,515,265	12,133,761	25,370,753

4 Open Fully Nested Family, Exponential Growth "OFN_E"

We now consider the effect of switching from closed to open quadrature rules, while preserving full nesting and exponential growth. The resulting quadrature families will be designated as having generic type OFN_E.

There are three interesting rules that can be used to generate families of this type: the Newton Cotes Open (NCO), the Fejer Type 2 family (F2), and the Gauss Patterson (GP).

The NCO family is numerically unstable, so we will not be employing it for sparse grid construction. The F2 and GP families are logically quite similar, except that the GP family has superior exactness. If we have reason to specify the particular rule being use, we can indicate that an exponential growth rate is being employed by using the designation such as "F2_E", "GP_E", or "NCO_E",

An open fully nested family with exponential growth can be created by selecting successively the rules of orders

$$\text{order\_1d} = \{1, 3, 7, 15, 31, 63, 127, 255, 511, 1023, 2047, \dots\}.$$

If we let the index be l, and count from 0, then the order of the rule of index l is $o(l) = 2^{l+1} - 1$. By level 10 we are using about 2,000 points. Because the order is always odd, the exactness of an NCO_E or F2_E rule is equal to the order; the exactness of a GP_E rule follows a more complicated formula.

Since these rules are fully nested, it is easy to construct the vector listing the number of new abscissas that appear at each element of the family. This list begins:

$$\text{new\_1d} = \{1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, \dots\},$$

and in general, the new order for level l is 2^l.

For a 2D grid, we can again perform a convolution on the **new_1d** vector to count the points. The order of the OFN_E sparse grid rule of level 3, dimension 2 is

$$\begin{aligned}\text{order}(3,2) &= 1*1\\ &+ (2*1+1*2)\\ &+ (4*1+2*2+1*4)\\ &+ (8*1+4*2+2*4+1*8)\\ &= 1*1+2*2+3*4+4*8\\ &= 49\end{aligned}$$

This suggests that for 2D, the order formula can be summarized as

$$\text{order}(l,2) = \sum_{i=0}^{l} (i+1)\,2^i$$

and that for general dimensions m, we have

$$\text{order}(l,m) = \sum_{i=0}^{l} \binom{i+m-1}{m-1} 2^i$$

We can also form a tabular array, as for example the following, which handles a 2D OFN_E sparse grid of level 5:

L	New						
5	32	32					
4	16	16	32				
3	8	8	16	32			
2	4	4	8	16	32		
1	2	2	4	8	16	32	
0	1	1	2	4	8	16	32
	New	1	2	4	8	16	32
	L	0	1	2	3	4	5

This table shows the number of points that have their "first appearance" in each of the product rules that is used to build an OFN sparse grid in dimension 2 of level 5, with a total of 321.

Here, we present point counts for OFN_E sparse grids of dimensions 1 through 10 and levels 0 through 10.

DIM:	1	2	3	4	5	6	7	8	9	10
OFN_E LEVEL										
0	1	1	1	1	1	1	1	1	1	1
1	3	5	7	9	11	13	15	17	19	21
2	7	17	31	49	71	97	127	161	199	241
3	15	49	111	209	351	545	799	1,121	1,519	2,001
4	31	129	351	769	1,471	2,561	4,159	6,401	9,439	13,441
5	63	321	1,023	2,561	5,503	10,625	18,943	31,745	50,623	77,505
6	127	769	2,815	7,937	18,943	40,193	78,079	141,569	242,815	397,825
7	255	1,793	7,423	23,297	61,183	141,569	297,727	580,865	1,066,495	1,862,145
8	511	4,097	18,943	65,537	187,903	471,041	1,066,495	2,228,225	4,361,215	8,085,505
9	1,023	9,217	47,103	178,177	553,983	1,496,065	3,629,055	8,085,505	16,807,935	32,978,945
10	2,047	20,481	114,687	471,041	1,579,007	4,571,137	11,829,247	28,000,257	61,616,127	127,574,017

5 Open Non-Nested Family, Linear Growth, "ONN_L2"

The next case we turn to involves families created from open quadrature rules in which there is no nesting at all (or at least, none with a pattern regular enough that we can take advantage of it), and a linear growth rule, increasing by 2 each time, is used.

The most familiar example of a quadrature rule with no nesting is the Gauss Laguerre rule ("LG"), and its variant the Generalized Gauss Laguerre rule ("GLG"). Other examples include the Gauss Jacobi rule ("GJ"), and, in general, rules formed by the Golub Welsch procedure ("GW").

The families we are interested in will use a linear growth rule for the order. Thus we might designate this class of families by the name "ONN_L2", or, if we wish to indicate the particular rule being used, then by "LG_L2", "GLG_L2", "GJ_L2" or "GW_L2".

Sparse grids made from these rules will get no reduction in the abscissa count because of nesting. It turns out that this makes it relatively easy to compute the point count. Any family in this class will have the 1D order vector

$$\text{order\_1d} = \{1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, \ldots\}.$$

Because there is no nesting, the **new_1d** vector would be identical to the **order_1d** vector. Since each multidimensional point in each product grid is now guaranteed to be unique, the point count is very simple: we add together the orders of all the constituent product grids.

For a 2D grid of level L, we can again perform a sort of convolution on the **order_1d** vector to count the points, but we have to remember that in 2D we only consider product rules that lie on the diagonal L and diagonal $L - 1$. The order of the ONN_L2 sparse grid rule of level 3, dimension 2 can be determined by:

$$\begin{aligned} \text{order}(3,2) = & \\ & + (5 * 1 + 3 * 3 + 1 * 5) \text{ (diagonal 2)} \\ & + (7 * 1 + 5 * 3 + 3 * 5 + 1 * 7) \text{ (diagonal 3)} \\ & = 63 \end{aligned}$$

We can also form a tabular array, as this example for a 2D level 5 ONN_L2 sparse grid:

L	Order						
5	11	11					
4	9	9	27				
3	7		21	35			
2	5			25	35		
1	3				21	27	
0	1					9	11
	Order	1	3	5	7	9	11
	L	0	1	2	3	4	5

Counting the points in a 2D ONN_L2 sparse grid of level 5.

This table shows the number of points used in each of the product grids that form an ONN_L2 sparse grid in dimension 2 of level 5, with a total of 231.

Here, we present point counts for ONN_L2 sparse grids, dimensions 1 through 10 and levels 0 through 10.

DIM:	1	2	3	4	5
ONN_L2 LEVEL					
0	1	1	1	1	1
1	3	7	10	13	16
2	5	25	52	87	131
3	7	63	189	403	736
4	9	129	543	1,461	3,206
5	11	231	1,320	4,433	11,583
6	13	377	2,834	11,739	36,218
7	15	575	5,531	27,911	100,893
8	17	833	10,013	60,809	255,663
9	19	1,159	17,062	123,253	598,538
10	21	1,561	27,664	235,135	1,310,165

DIM:	6	7	8	9	10
ONN_L LEVEL					
0	1	1	1	1	1
1	19	22	25	28	31
2	184	246	317	397	486
3	1,216	1,870	2,725	3,808	5,146
4	6,190	10,900	17,903	27,847	41,461
5	25,954	52,074	96,055	165,844	271,467
6	93,535	212,738	439,019	838,915	1,506,232
7	298,357	765,313	1,760,035	3,711,040	7,290,952
8	860,455	2,476,883	6,323,269	14,666,470	31,453,182
9	2,279,829	7,329,934	20,693,565	52,638,759	122,920,642
10	5,618,754	20,087,574	62,483,217	173,788,146	440,815,035

6 Open Non-Nested Family, Exponential Growth, “ONN_E”

If an exponential growth rule is used with an Open Non-Nested Family, we get sparse grids of type “ONN_E”.

The change in the growth rule means that the order vector will be

$$\text{order\_1d} = \{1, 3, 7, 15, 31, 63, 127, 255, 511, 1023, 2047, \ldots\}.$$

A tabular array for a 2D level 5 ONN_E sparse grid looks like this:

L	Order						
5	63	63					
4	31	31	93				
3	15		45	105			
2	7			49	105		
1	3				45	93	
0	1					31	63
	Order	1	3	7	15	31	63
	L	0	1	2	3	4	5

Counting the points in a 2D ONN_E sparse grid of level 5.

This table shows the number of points used in each of the product grids that form an ONN_E sparse grid in dimension 2 of level 5, with a total of 723.

Here, we present point counts for ONN_E sparse grids, dimensions 1 through 10 and levels 0 through 10.

DIM:	1	2	3	4	5
ONN_E LEVEL					
0	1	1	1	1	1
1	3	7	10	13	16
2	7	29	58	95	141
3	15	95	255	515	906
4	31	273	945	2,309	4,746
5	63	723	3,120	9,065	21,503
6	127	1,813	9,484	32,259	87,358
7	255	4,375	27,109	106,455	325,943
8	511	10,265	73,915	330,985	1,135,893
9	1,023	23,579	194,190	980,797	3,743,358
10	2,047	53,277	495,198	2,793,943	11,775,507

DIM:	6	7	8	9	10
ONN_E LEVEL					
0	1	1	1	1	1
1	19	22	25	28	31
2	196	260	333	415	506
3	1,456	2,192	3,141	4,330	5,786
4	8,722	14,778	23,535	35,695	52,041
5	44,758	84,708	149,031	247,456	392,007
6	204,203	428,772	828,795	1,499,773	2,571,712
7	849,161	1,966,079	4154,403	8,158,810	15,089,932
8	3,275,735	8,316,605	1,9122,245	40,599,130	80,725,502
9	11,876,081	32,894,998	8,1953,165	187,432,959	399,429,602
10	40,869,038	122,928,088	33,0545,025	811,645,950	1,848,483,779

7 Open Weakly Nested Family, Linear Growth, "OWN_L"

We now turn to the case of Open Weakly Nested families. The typical example here is the family formed from the Gauss Legendre rule ("GL"). The Gauss Hermite ("GH") and Generalized Gauss Hermite ("GGH") rules also exhibit this behavior.

We will use a strict linear growth rule, so that the rule of index l will have order $o(l) = l+1$ and exactness $e(l) = 2l + 1$ (because we are assuming the underlying rules are derived from the usual Gauss procedure). We designate this class by "OWN_L". We see that the abscissa 0.0 is common to the odd rules, and that no other value is repeated.

We have an **order_1d** vector of

$$\text{order\_1d} = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, \ldots\}.$$

Since the value 0.0 comes and goes every other rule, we don't want to keep track of the "new" values, but rather the zero and nonzero ones:

$$\begin{aligned}\text{zero\_1d} =& \{1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, \ldots\}\\ \text{nonz\_1d} =& \{0, 2, 2, 4, 4, 6, 6, 8, 8, 10, 10, \ldots\}.\end{aligned}$$

Counting the abscissas of an OWN_L family is more difficult than for the fully nested and non-nested cases. I have not worked out a general algorithm for counting these points. You can make a start at one by noticing that you can first count all the points which have no zero component, by using the procecure for counting the points in an ONN_L sparse grid of the same level and dimension, but using the **nonz_1d** vector as though it were the **order_1d** vector.

But then you have to work out how many points contain one or more zero components, and, especially in higher dimensions, this seems to be a difficult thing to handle. Since we can do this for the OWN_L2 case, the problem with the OWN_L case seems to arise from the fact that the zero value only appears in the 1D rules of even level.

8 Open Weakly Nested Family, Double Linear Growth, "OWN_L2"

There is a simple variation of the OWN_L family, in which *two* points are added with each increase in the level index. We refer to this family as having the type **OWN_L2**. Again, we assume the usual examples come from the Gauss Legendre rule ("GL"), Gauss Hermite ("GH") and Generalized Gauss Hermite ("GGH") rules.

In this case, because every rule has odd order, the abscissa 0.0 is common to every rule.

The linear growth rule has the form $o(l) = 2\,l + 1$. This gives us an **order_1d** vector of

$$\text{order\_1d} = \{1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, \ldots\}.$$

In turn, the vector of "new" (that is, in this case, nonzero!) points is

$$\text{new\_1d} = \{0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, \ldots\}.$$

Counting the abscissas of an OWN_L2 family is more difficult than for the fully nested and non-nested cases. The algorithm that was discovered begins by counting the points in which no 0.0 occurs. This is the same as counting the number of points in an ONN_L sparse grid of the same level and dimension, but with a modified **order_1d** vector. The modifications involve setting the first entry to zero, and setting subsequent entries to the corresponding entries of the original **new_1d**.

For 2D, this first step can be visualized in a tabular array, here for a level 4, dimension 2 OWN_L2 sparse grid, where “Order” now is actually counting the number of points in each 1D rule that are not equal to 0.0:

L	Order					
4	8	0				
3	6	0	12			
2	4		8	16		
1	2			8	12	
0	0				0	0
	Order	0	2	4	6	8
	L	0	1	2	3	4

This clearly shows that the number of points with no 0.0 is 56.

The next step is to compute the number of points with exactly one occurrence of 0.0. We essentially have to collapse one dimension of the above table, and assign a single ”Order” value of 1 (for the single value of 0.0). One version of the collapsed table looks like this:

L	Order	
4	8	8
3	6	6
2	4	4
1	2	2
0	0	0
	Order	1
	L	0

and of course we could also have collapsed the table horizontally. Since each table yields a total of 20 points with exactly one 0.0, the total is 40.

Finally, to compute the number of points with exactly two 0.0’s, we collapse the table yet again, giving the collapsed dimension an ”Order” of 1, to get:

L	Order	
0	1	1
	Order	1
	L	0

So our total is 56+40+1 which correctly produces the result 97.

To generalize this to higher dimensions and levels, we have to think as follows. First, we compute the completely nonzero points, which can be thought of as counting an ONN_L grid with the modified version of **order_1d**. Recall that the standard sparse grid computation includes product grids with levels from $L - M + 1$ to L.

Now we have to count points with $N = 1, 2, \ldots M$ zeroes in them. We do this by counting the number of points in a sparse grid of dimension $M - N$, but now we allow the level to range from 0 to L, that is. Once we’ve counted the number of such points in one example sparse grid of dimension $M - N$, we must multiply the number of points by $\binom{M}{N}$, to account for the fact that there are this many ways to choose where the 0.0’s occur.

Here is a MATLAB routine which carries out this calculation. Again, we assume that **comp_next** returns on each call a new composition of the input value **level**.

```
if ( level_max < 0 )
  point_num = 0;
```

```
    return
  end

  if ( level_max == 0 )
    point_num = 1;
    return
  end

  new_1d = zeros ( level_max+1, 1 );
  new_1d(1) = 0;
  for l = 1 : level_max
    new_1d(l+1) = 2 * l;
  end

  point_num = 0;

  for dim_num2 = dim_num : -1 : 0

    if ( dim_num2 == dim_num )
      level_min = max ( 0, level_max - dim_num + 1 );
    else
      level_min = 0;
    end

    if ( dim_num2 == 0 )
      point_num2 = 1;
    else
      level_1d = zeros ( dim_num2, 1 );
      point_num2 = 0;
      for level = level_min : level_max
        more = 0;
        h = 0;
        t = 0;
        while ( 1 )
          [ level_1d, more, h, t ] = comp_next ( level, dim_num2, level_1d, ...
            more, h, t );
          point_num2 = point_num2 + prod ( new_1d(level_1d(1:dim_num2)+1) );
          if ( ~more )
            break
          end
        end
      end
    end

    point_num = point_num + i4_choose ( dim_num, dim_num2 ) * point_num2;

  end
```

Here is a table of the point counts for an OWN_L2 sparse grid, for dimensions 1 through 10 and levels 0 through 10.

DIM:	1	2	3	4	5
OWN_L2 LEVEL					
0	1	1	1	1	1
1	3	5	7	9	11
2	5	17	31	49	71
3	7	45	105	201	341
4	9	97	297	681	1,341
5	11	181	735	2,001	4,543
6	13	305	1,631	5,257	13,683
7	15	477	3,305	12,609	37,433
8	17	705	6,209	28,017	94,473
9	19	997	10,951	58,297	222,563
10	21	1,361	18,319	114,561	493,935

DIM:	6	7	8	9	10
OWN_L2 LEVEL					
0	1	1	1	1	1
1	13	15	17	19	21
2	97	127	161	199	241
3	533	785	1,105	1,501	1,981
4	2,381	3,921	6,097	9,061	12,981
5	9,113	16,703	28,577	46,303	71,785
6	30,869	62,735	117,713	207,355	347,005
7	94,601	212,481	436,033	833,017	1,501,545
8	266,489	659,585	1,476,673	3,053,065	5,916,505
9	698,373	1,899,663	4,629,457	10,338,603	21,503,085
10	1,718,697	5,124,927	13,566,753	32,667,567	72,810,297

9 Open Weakly Nested Family, Exponential Growth, "OWN_E"

If we choose exponential growth instead of linear growth for our Open Weakly Nested family, we get the "OWN_E" class. The analysis is almost identical as for the OWN_L case.

The exponential growth rule has the form $o(l) = 2^{l+1} - 1$. This gives us an **order_1d** vector of

$$\text{order\_1d} = \{1, 3, 7, 15, 31, 63, 127, 255, 511, 1023, 2047, \ldots\}.$$

In turn, the vector of "new" (that is, nonzero!) points is

$$\text{order\_1d} = \{0, 2, 6, 14, 30, 62, 126, 254, 510, 1022, 2046, \ldots\}.$$

Here is a table of the point counts for an OWN_E sparse grid, for dimensions 0 through 10 and levels 0 through 10.

DIM:	1	2	3	4	5	6	7	8	9	10
OWN_E LEVEL										
0	1	1	1	1	1	1	1	1	1	1
1	3	5	7	9	11	13	15	17	19	21
2	7	21	37	57	81	109	141	177	217	261
3	15	73	159	289	471	713	1,023	1,409	1,879	2,441
4	31	221	597	1,265	2,341	3,953	6,245	9,377	13,525	18,881
5	63	609	2,031	4,969	10,363	19,397	33,559	54,673	84,931	126,925
6	127	1,573	6,397	17,945	41,913	86,517	163,213	287,409	479,233	764,365
7	255	3,881	18,943	60,577	157,583	357,153	731,951	1,388,737	2,478,511	4,208,385
8	511	9,261	53,365	193,441	557,693	1,382,361	3,067,669	6,253,537	11,916,685	21,493,065
9	1,023	21,553	144,351	589,625	1,875,443	5,065,693	12,136,743	26,516,113	53,833,083	102,935,845
10	2,047	49,205	377,661	1,727,625	6,037,137	17,709,469	45,683,389	106,723,249	230,380,089	466,201,781

10 Clenshaw Curtis Family, Slow Exponential Growth, "CC_SE"

The CC_SE family is the first example of the slow exponential growth families. It was developed in an attempt to maintain the advantages of a nested rule while restraining the inherent exponential growth in order as a function of level.

The idea was to begin with the CC_E exponential growth family, indexed by k, and then to construct a new slow exponential growth family, indexed by j, in such a way that the j-th rule had the lowest index k for which the rule exactness was at least $2j + 1$.

Since a formula for the exactness of each CC_E rule is known, this procedure can easily be made automatic. The order vector then shows occasional signs of "stuttering":

$$\textbf{order\_1d} = \{1, 3, 5, 9, 9, 17, 17, 17, 17, 33, 33, \ldots\}.$$

and the vector that counts new points hence will now contain some 0's:

$$\textbf{new\_1d} = \{1, 2, 2, 4, 0, 8, 0, 0, 0, 16, 0, \ldots\}.$$

The CC_SE family is a closed fully nested rule, so our counting procedures that we used for the CC_E family will work here as well. For instance, we can make a tabular array for a Level 5, 2D case:

L	New						
5	8	8					
4	0	0	0				
3	4	4	8	8			
2	2	2	4	4	8		
1	2	2	4	4	8	0	
0	1	1	2	2	4	0	8
	New	1	2	2	4	0	8
	L	0	1	2	3	4	5

Counting the points in a 2D CC_SE sparse grid of level 5.

Summing up the entries within the box, we compute the number of points in a CC_SE sparse grid for dimension 2, level 5, is 81.

The code to compute the order for a CC_SE sparse grid is quite similar to that for a CC_E sparse grid, and in fact, only the definition of **new_1d** needs to be revised:

```
new_1d = new int[level_max+1];
new_1d[0] = 1;
new_1d[1] = 2;
e = 3;
o = 3;
for ( l = 2; l <= level; l++ )
{
  e = 2 * l + 1;
  if ( o < e )
  {
    new_1d[l] = o - 1;
    o = 2 * o - 1;
  }
  else
  {
    new_1d[l] = 0;
  }
}
```

Again, we compute the number of points in sparse grids of levels 0 through 10, and dimensions 1 through 10. Because we are using the more economical CC_S rule, however, there are some dramatic differences when compared to the tables for the CC_E sparse grid. In particular, the values in column 1 decrease drastically.

In general, since the CC_SE and CC_E 1D rules do not differ until level 4, we do not expect to see differences in the CC_SE table until that point. At level 4 and beyond, the difference is most noticeable for low dimensional grids. Starting at any fixed location in the table, the relative improvement from using the CC_SE family becomes stronger if we decrease the dimension or increase the level.

DIM:	1	2	3	4	5
CC_SE LEVEL					
0	1	1	1	1	1
1	3	5	7	9	11
2	5	13	25	41	61
3	9	29	69	137	241
4	9	49	153	369	761
5	17	81	297	849	2,033
6	17	129	545	1,777	4,833
7	17	161	881	3,377	10,433
8	17	225	1,361	5,953	20,753
9	33	257	1,953	9,857	38,593
10	33	385	2,721	15,361	67,425

DIM:	6	7	8	9	10
CC_SE LEVEL					
0	1	1	1	1	1
1	13	15	17	19	21
2	85	113	145	181	221
3	389	589	849	1,177	1,581
4	1,409	2,409	3,873	5,929	8,721
5	4,289	8,233	14,689	24,721	39,665
6	11,473	24,529	48,289	88,945	155,105
7	27,697	65,537	141,601	284,209	536,705
8	61,345	159,953	377,729	823,057	1,677,665
9	126,401	361,665	930,049	2,192,865	4,810,625
10	244,289	765,089	2,136,577	5,4363,21	12,803,073

11 Fejer Type 2 Family, Slow Exponential Growth, "F2_SE"

The development of F2_SE, the slow exponential growth version of the Fejer Type 2 family F2_E, is quite similar to that of CC_SE from the CC_E family. The main difference is that Fejer Type 2 rules are open, so their pattern of growth is slightly different.

However, the criterion for the F2_SE family is the same. When we choose the index k of an F2_E rule, which is to be used as the j-th rule in the F2_SE family, we assume that the sparse grid exactness requirement is that $2j+1 \leq e(j)$. So we choose the lowest value of k which satisfies the exactness requirement. For Fejer Type 2 rules of odd order, the order and exactness are equal, and the order satisfies the rule $o(k) = 2^{k+1} - 1$. Thus, for instance, our first five 5 exactness requirements are $e_{min}(0:4) = \{1, 3, 5, 7, 9\}$ and we can satisfy these by using rules of order $o(0:4) = \{1, 3, 7, 7, 15\}$.

Since the F2_SE rules are fully nested, we can use the same approach to counting them as we did for the F2_E rule. The only difference is that the **new_1d** vector is different now. In particular, we can see from the above that the first five entries are { 1, 2, 4, 0, 8 }.

MATLAB code to do the counting follows. Note that MATLAB does not allow arrays to be indexed by 0, so whenever we index **new_1d** we make a point of writing the index as $i+1$.

```
    if ( level_max < 0 )
    point_num = 0;
    return
  end
  if ( level_max == 0 )
    point_num = 1;
    return
  end

  new_1d = zeros ( level_max+1, 1 );
  new_1d(0+1) = 1;
  e = 1;
  o = 1;
  for l = 1 : level_max
    e = 2 * l + 1;
    if ( o < e )
      new_1d(l+1) = o + 1;
      o = 2 * o + 1;
    else
      new_1d(l+1) = 0;
    end
  end

  level_1d = zeros ( dim_num, 1 );
  point_num = 0;

  for level = 0 : level_max

    more = 0;
    h = 0;
    t = 0;
    while ( 1 )
      [ level_1d, more, h, t ] = comp_next ( level, dim_num, level_1d, more, h, t );
      point_num = point_num + prod ( new_1d(level_1d(1:dim_num)+1) );
      if ( ~more )
        break
      end
    end
  end
```

Here is a table of the point counts for the F2_SE family, for dimensions 1 through 10 and levels 0 through 10.

DIM:	1	2	3	4	5
F2_SE LEVEL					
0	1	1	1	1	1
1	3	5	7	9	11
2	7	17	31	49	71
3	7	33	87	177	311
4	15	65	207	513	1,071
5	15	97	399	1,217	3,023
6	15	161	751	2,625	7,503
7	15	161	1,135	4,929	16,463
8	31	257	1,759	8,705	33,183
9	31	321	2,335	13,697	60,703
10	31	449	3,679	21,889	105,887

DIM:	6	7	8	9	10
F2_SE LEVEL					
0	1	1	1	1	1
1	13	15	17	19	21
2	97	127	161	199	241
3	497	743	1,057	1,447	1,921
4	1,985	3,375	5,377	8,143	11,841
5	6,497	12,559	22,401	37,519	59,745
6	18,401	40,111	79,745	147,343	256,545
7	46,049	112,815	249,217	506,767	963,105
8	104,705	286,303	699,393	1,559,839	3,227,905
9	217,281	663,071	1,787,649	4,362,783	9,809,985
10	421,185	1,423,327	4,217,601	11,231,007	27,377,857

12 Gauss Patterson Family, Slow Exponential Growth "GP_SE"

The original Gauss Patterson family shares the same structure as the F2_E family. However, GP_SE and F2_SE, the slow exponential growth versions, differ markedly because the higher exactness of the Gauss Patterson rules allows us to meet the exactness requirements with rules of lower order. This is then reflected in a substantially reduced point count for GP_SE rules. For a more elaborate discussion of the GP_SE family, refer to [**?**].

Since the formula for the exactness of the Gauss Patterson rules is known, and we have already considered the rationale for the slow exponential growth several times, we now simply present a FORTRAN code to count the points in a GP_SE sparse grid:

```
if ( level_max < 0 ) then
  point_num = 0
  return
end if

if ( level_max == 0 ) then
  point_num = 1
  return
end if

allocate ( order_1d(0:level_max) )
order_1d(0) = 1
```

```
  do level = 1, level_max
    e = 5
    o = 3
    do while ( e < 2 * level + 1 )
      e = 2 * e + 1
      o = 2 * o + 1
    end do
    order_1d(level) = o
  end do

  allocate ( new_1d(0:level_max) )
  new_1d(0) = 1
  do level = 1, level_max
    new_1d(level) = order_1d(level) - order_1d(level-1)
  end do

  allocate ( level_1d(1:dim_num) )
  point_num = 0

  do level = 0, level_max

    more = .false.
    h = 0
    t = 0

    do
      call comp_next ( level, dim_num, level_1d, more, h, t )
      point_num = point_num + product ( new_1d(level_1d(1:dim_num)) )
      if ( .not. more ) then
        exit
      end if
    end do

  end do
```

Here we present a table of point counts for GP_SE sparse grids of dimensions 1 through 10 and levels 0 through 10.

DIM:	1	2	3	4	5
GP_SE LEVEL					
0	1	1	1	1	1
1	3	5	7	9	11
2	3	9	19	33	51
3	7	17	39	81	151
4	7	33	87	193	391
5	7	33	135	385	903
6	15	65	207	641	1,743
7	15	97	399	1,217	3,343
8	15	97	495	1,985	6,223
9	15	161	751	2,881	10,063
10	15	161	1,135	4,929	17,103

DIM:	6	7	8	9	10
GP_SE LEVEL					
0	1	1	1	1	1
1	13	15	17	19	21
2	73	99	129	163	201
3	257	407	609	871	1,201
4	737	1,303	2,177	3,463	5,281
5	1,889	3,655	6,657	11,527	19,105
6	4,161	8,975	17,921	33,679	60,225
7	8,481	19,855	43,137	87,823	169,185
8	16,929	42,031	97,153	211,087	434,145
9	30,689	83,247	206,465	477,327	1,041,185
10	53,729	154,927	411,265	1,014,159	2,347,809

Note that if we were to judge simply by how many millions of points are needed for a sparse grid of dimension 10, level 10, then the GP_SE family is by far the winner (2), followed by CC_SE (12) and F2_SE (27) with the very worst performance by the ONN_E family (1,848).

13 Conclusion

We have shown how to count the number of unique abscissas for an isotropic sparse grid which is constructed using the same quadrature family for each dimension. A previous attempt to do this computation involved an *a posteriori* process, that is, we generated all the points from the Smolyak procedure, sorted them, and identified points that were equal. The *a priori* procedure described here is more efficient and satisfying; it does not need to generate the points, but rather counts them simply by virtue of certain known properties.

The point counts discussed in this document have been implemented in C++, FORTRAN90, and MATLAB, and are available online. The C++ versions, for instance, may be found at [**?**], and the web page includes links to the versions in other languages.

Accurate point counts are important in sparse grid computations, since they measure the efficiency of a rule. Making point count tables, as in this document, allows one to examine the growth trends as the underlying rules are varied, and to judge the value of techniques such as slow exponential growth. Of course, during the actual computation of a sparse grid, point counts are necessary so that sufficient space can be allocated for arrays. Finally, the efforts made in computing a point count can reveal patterns in the sparse growth construction that can be exploited for greater efficiency.

An important extension of this investigation would consider *anisotropic* sparse grids, that is, sparse grids which used higher order rules in certain dimensions. At least for some of the families considered here, it seems likely that the procedures can be extended in a natural way to cover these cases.

A more difficult case arises when we allow *mixed* sparse grids, that is, sparse grids for which different quadrature families are used for the various dimensions. It is felt that the rather elaborate form of the calculation for the OWN family may suggest the complications that could arise when trying to deal with the effects of dimensional variation in the rule types.

A natural final point would be to consider *anisotropic mixed* sparse grids, in which we allow dimensional variation in both the order and quadrature family.

References

[1] John Burkardt, SPARSE_COUNT: Sparse grids using a single factor, http://people.sc.fsu.edu/∼jburkardt/cpp_src/sparse_count/sparse_count.html.

[2] John Burkardt, 1D quadrature rules for sparse grids, http://people.sc.fsu.edu/∼jburkardt/presentations/sgmga_1d_rules.pdf.

[3] John Burkardt, Slow exponential growth for Clenshaw Curtis sparse grids, http://people.sc.fsu.edu/∼jburkardt/presentations/sgmga_ccs.pdf.

[4] John Burkardt, Slow exponential growth for Gauss Patterson sparse grids, http://people.sc.fsu.edu/∼jburkardt/presentations/sgmga_gps.pdf.

[5] Erich Novak, Klaus Ritter, Simple cubature formulas with high polynomial exactness, Constructive Approximation, Volume 15, Number 4, December 1999, pages 499-522.

[6] Sergey Smolyak, Quadrature and interpolation formulas for tensor products of certain classes of functions, Doklady Akademii Nauk SSSR, Volume 4, 1963, pages 240-243.