Introduction to `SparseGrid` [*]

Jelmer Ypma

October 31, 2024

Abstract

This vignette describes how to use `SparseGrid`, which is an R translation[1] of the Matlab code on `http://www.sparse-grids.de` (Heiss and Winschel, 2008). Sparse grids can be used to numerically approximate integrals of high dimension, with fewer nodes than grids constructed by a product rule.

1 Introduction

Integrals arise in many places in statistics, for instance when calculating likelihood contributions in latent variable models (where some of the underlying variables are not observed), or when calculating conditional expectations or moments of functions of random variables. When an analytic solution to these integrals is not available, they have to be evaluated numerically. There are many texts providing a description of how to numerically approximate integrals (e.g. see Judd, 1998; Miranda and Fackler, 2002, for an introduction to different approximation methods). Basically, many of the methods to approximate an integral in 1 dimension, rewrite the integral as a weighted sum

$$\int_\Omega g(x)f(x)dx \approx \sum_{r=1}^{R} w_r g(x_r),$$

where Ω is the domain that we'd like to integrate over. The function $g(x)$ is the function of interest, and $f(x)$ is a weighting function. In statistics $f(x)$ will usually be the probability density function of x. For instance, if we're interested in the mean of x, where x is normally distributed, then we can write this as an integral

$$E[x] = \int_{-\infty}^{\infty} \underbrace{x}_{g(x)} \cdot \underbrace{\frac{1}{\sqrt{2\pi\sigma}} e^{\frac{(x-\mu)^2}{2\sigma^2}}}_{f(x)} dx.$$

For this choice of $g(x)$ there is of course a closed-form solution, but the integral has to be approximated numerically for more complicated functions $g(x)$. Depending on the function $f(\cdot)$, there are standard (quadrature) rules to choose w_r, referred to as **weights**, and x_r, referred to as **nodes**.

[*]This package should be considered in beta and comments about any aspect of the package are welcome. This document is an R vignette prepared with the aid of `Sweave` (Leisch, 2002).

[1]Florian Heiss and Viktor Winschel kindly provided permission to make this R package based on their Matlab version publicly available.

Four quadrature rules are included with `SparseGrid`. The first two, `GQU` and `KPU`, can be used for unweighted integration on a unit domain, $\Omega = [0,1]^D, f(x) = 1$. These can be used if the random variables that we want to integrate out have a uniform distribution, since $f(x) = 1$ is the probability density function of a uniformly distributed random variable. The other two quadrature rules, `GQN` and `KPN`, can be used to approximate a Guassian-weighted integral on $\mathcal{R}^D$.

2 Integration over multiple dimensions

In multiple dimensions, a straightforward way to combine the nodes and weights of single dimensions, is by using a product rule. The nodes of the multidimensional approximation are the kronecker product of the separate dimensions

$$x = x_1 \otimes x_2 \otimes \cdots \otimes x_D,$$

where D is the number of dimensions. For instance, for two dimensions

$$x = x_1 \otimes x_2 = \begin{pmatrix} x_{1,1} & x_{2,1} \\ x_{1,1} & x_{2,2} \\ \vdots & \vdots \\ x_{1,1} & x_{2,R_2} \\ x_{1,2} & x_{2,1} \\ \vdots & \vdots \\ x_{1,2} & x_{2,R_2} \\ \vdots & \vdots \\ x_{1,R_1} & x_{2,R_2} \end{pmatrix},$$

where $x_{d,k}$ is the k-th node in dimension d. For the two-dimensional example, the total number of integration nodes is $R_1 \cdot R_2$. In general, when we have R nodes in each dimension, the number of nodes needed to approximate a D dimensional integral is R^D.

Table 1: Number of nodes used by SGI or product rule, $k = 5$, type=`GQU`

D	1	2	3	4	5	6	7	8	9
SGI	5	53	165	385	781	1433	2437	3905	5965
k^D	5	25	125	625	3125	15625	78125	390625	1953125

Heiss and Winschel (2008) describe how sparse grids can be used to approximate multi-dimensional integrals, that occur for instance in estimation problems[2]. As an example they compare different ways of approximating the high-dimensional integrals that arise in mixed logit models.

The benefit of using sparse grids, is that fewer integration nodes are needed than in the case of the product rule, except for low dimensional integrals. Table 1 shows the number of nodes that are needed for the two methods for different

[2]Another way to deal with the curse of dimensionality is to use Monte Carlo metohds to approximate the integral.

dimensions. For instance, in 8 dimensions the product rule uses 3905 integration nodes and sparse grid integration requires 390625 nodes.

Of course, this reduction in number of integration nodes comes at a cost (see Heiss and Winschel, 2008, for details). The parameter, k, in table 1 controls the accuracy of the approximation. A sparse grid of accuracy k can accurately integrate the function $g(x)$ if $g(x)$ is a polynomial of total order $2k-1$. For example, the polynomial

$$a_1x_1^3 + a_2x_1^2x_2 + a_3x_1x_2^2 + +a_4x_1 + a_5x_2^2 + a_6x_2^2 + a_7x_2^3,$$

is of total order 3, since the maximum of the sum of the exponents in each term is 3. The product rule grid based on the same one-dimensional quadrature nodes is accurate for higher dimensions, terms such as $x_1^3x_2^2, x_1^3x_2^3, x_1x_2^3$ can also occur. Since these higher orders grow exponentially[3], the number of nodes needed to approximate this polynomial without error grows exponentially. The number of nodes used in sparse grid integration are smaller by bounding the total order of the polynomial that is integrated exactly. This difference in accuracy is the reason why sparse grids contain less nodes then grids constructed by the product rule.

For general problems you usually want to approximate integrals of functions $g(x)$ that can not be written as a polynomial. However, the function $g(x)$ might be well approximated by a low-order polynomial. In these cases it is good practice to check if the approximation to the integral is accurate enough for your specific problem. One way to get a sense of the accuracy of the approximation, is to compare approximations obtained using different grids. For instance by increasing the number of nodes, using sparse grids, product rule grids or Monte Carlo grids with different seeds for the random number generator.

3 Installation

The package can be installed from CRAN

```
> install.packages('SparseGrid')
```

After which you should be able to load the package using

```
> library('SparseGrid')
```

And get help for the main function with

```
> ?SparseGrid
```

Information on how to cite the original paper on which this code is based is obtained through

```
> citation('SparseGrid')
This paper describes the theory/code on which
SparseGrid is based:
```

[3]If there are 5 dimensions, the tensor product of the same univariate integration grid is also exact for the term $x_1^3x_2^3x_3^3x_4^3x_5^3$. The total order of this term is 15, and grows with the dimension of the integration.

```
  Florian Heiss, Viktor Winschel, Likelihood
  approximation by numerical integration on sparse
  grids, Journal of Econometrics, Volume 144, Issue
  1, May 2008, Pages 62-80

A BibTeX entry for LaTeX users is

  @Article{,
    title = {Likelihood approximation by numerical integration on sparse grids},
    author = {Florian Heiss and Viktor Winschel},
    journal = {Journal of Econometrics},
    volume = {144},
    number = {1},
    pages = {62--80},
    year = {2008},
  }

More information is available from the website:
http://www.sparse-grids.de
```

4 Overview of available functions

There are four different functions to create grids for integration. Each of these functions returns a list with two elements; `nodes`, a matrix of nodes, and `weights`, a vector of weights. More information can be obtained from their respective help pages.

```
> ?createSparseGrid
> ?createProductRuleGrid
> ?createMonteCarloGrid
> ?createIntegrationGrid
```

Another function that is included in the packages is `readASCGrid`, which can be used to read files with integration grids as available on the website `http://www.sparse-grids.de`.

5 Example

This section contains an example based on the one available from `http://www.sparse-grids.de`. The example shows how to approximate an integral with sparse grids, and compares it to approximating the same integral using Monte Carlo simulation. The integral that we want to approximate is

$$\mathcal{I} = \int_0^1 \cdots \int_0^1 \underbrace{\left(\prod_{d=1}^{D} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x_d^2} \right)}_{g(x)} dx_D \cdots dx_1 .$$

The integration domain of this function is $[0, 1]^D$ and we can use a unit weighting function $f(x) = 1$, so we can use the KPU or GQU methods to create sparse grids for numerical approximation of the integral.

First, load the library,

```
> library('SparseGrid')
```

and create a sparse grid of `dimension = 10`, and with accuracy `k = 2`

```
> dimension <- 10
> k         <- 2
> sgrid     <- createSparseGrid( type='KPU', dimension=dimension, k=k )
```

This grid has

```
> length( sgrid$weights )
[1] 21
```

nodes for the chosen accuracy. Usually we only have to create this grid once at the beginning of an estimation procedure, and can re-use the same grid to approximate different integrals (e.g. the likelihood contributions of different individuals, or the approximation to an integral in different iterations of an optimization method).

Then, we define the function g in R that calculates the function $g(x)$ defined above at a given point x

```
> g <- function( x, mu=0, sigma=1 ) {
      return( prod( exp(-.5*((x-mu)/sigma)^2)/sqrt(2*pi*sigma^2) ) )
  }
```

Note that this function only works on one gridpoint at a time. Because we want to evaluate the function at many gridpoints (each row in sgrid$nodes is a separate gridpoint), we write this convenience function that performs the approximation.

```
> approximate.integral <- function( func, sgrid, ... ) {
      gx <- apply( sgrid$nodes, 1, function(x) { func(x, ...) } )
      return( sum( gx * sgrid$weights )  )
  }
```

The first line of this function loops over the rows (gridpoints) of the integration grid and evaluates the function at each gridpoint. This results in a vector gx. We then multiply this vector by the corresponding weights and take the sum. This weighted sum is our approximation to the integral.

```
> sigma <- 2
> approximate.integral( g, sgrid, mu=0, sigma=sigma )
[1] 6.58969e-08
```

In R we can get the 'exact' solution[4] using

```
> trueval <-
      ( pnorm( 1, mean=0, sd=2 ) - pnorm( 0, mean=0, sd=sigma ) )^dimension
```

[4]pnorm itself is not completely exact, but is an approximation using the erf-function.

so we can compare the approximation using different accuracies for the sparse grid, and the Monte Carlo integration with the 'true' value.

We can also approximate the integral using a grid of `num.sim = 1000` random points drawn from the uniform distribition

```
> num.sim <- 1000
> set.seed( 3141 )
> mcgrid <- createMonteCarloGrid(
                 runif, dimension=dimension, num.sim=num.sim )
```

and evaluate the function that we want to integrate over to get an approximation by Monte Carlo simulation

```
> approximate.integral( g, mcgrid, mu=0, sigma=sigma )
[1] 6.58229e-08
```

Below, we compare the error for this specific case for different accuracies of the sparse grid. The number of nodes used in the Monte Carlo approximation to the integral are the same as the number of nodes in the sparse grid. This enables us to compare the error of both methods when the computation time is the same, since conditional on the number of integration nodes, the computation time for the two methods is the same.

First we set the random seed, the dimension of the integration and the maximum accuracy level for which we want to approximate the integral.

```
> set.seed( 3141 )
> dimension   <- 10   # dimension of integral
> maxk        <- 4    # max. accuracy level (pol. exactness wil be 2k-1)
```

Then we create a matrix of the right dimensions that will hold the results.

```
> # create matrix to hold results
> res <- matrix( NA, nrow=maxk-1, ncol=5 )
> colnames( res ) <- c("D", "k", "nodes", "SG error", "MC error")
> rownames( res ) <- rep( "", maxk-1 )
```

The comparision is performed by looping over the requested accuracy levels. For each accuracy level we create a sparse grid, and do the approximation and calculate the approximation error by comparing the approximated value to the 'true' value. Since Monte Carlo approximations are different for each seed of the random generator, we use the mean of 1000 approximated values and calculate the approximation error based on this mean. The results are then saved in `res`.

```
> # loop over different accuracy levels
> for ( k in 2:maxk ) {

      # sparse grid integration
      sgrid   <- createSparseGrid('KPU', dimension, k)
      SGappr  <- approximate.integral( g, sgrid, mu=0, sigma=sigma )
      SGerror <- sqrt((SGappr - trueval)^2) / trueval

      # Monte Carlo integration with the same number of nodes
```

```
        # 1000 simulation repetitions
        num.nodes <- length( sgrid$weights )
        MCappr    <- rep(0, 1000)
        for (r in 1:1000) {
            mcgrid      <- createMonteCarloGrid(
                                 runif,
                                 dimension=dimension,
                                 num.sim=num.nodes )
            MCappr[ r ] <- approximate.integral(
                                 g, mcgrid, mu=0, sigma=sigma )
        }
        MCerror = sqrt(mean((MCappr-trueval)^2)) / trueval

        # save results in row of matrix res
        res[k-1,] <- c(dimension, k, num.nodes, SGerror, MCerror)
    }
```

The results for this comparison are given in the following table

```
    > res
      D k nodes     SG error    MC error
     10 2    21 4.528995e-03 0.024644168
     10 3   201 1.189321e-04 0.007877159
     10 4  1201 2.073774e-06 0.003383167
```

For this specific example we see that the error for sparse grid integration declines much more rapidly when increasing the number of nodes than the error when approximating the integral using Monte Carlo simulation.

References

Florian Heiss and Viktor Winschel. Likelihood approximation by numerical integration on sparse grids. *Journal of Econometrics*, 144(1):62–80, 2008.

K.L. Judd. *Numerical methods in economics.* MIT Press, Cambridge, MA, USA, 1998. ISBN 9780262100717.

Friedrich Leisch. Sweave: Dynamic generation of statistical reports using literate data analysis. In Wolfgang Härdle and Bernd Rönz, editors, *Compstat 2002 — Proceedings in Computational Statistics*, pages 575–580. Physica Verlag, Heidelberg, 2002. URL `http://www.stat.uni-muenchen.de/~leisch/Sweave`. ISBN 3-7908-1517-9.

Mario J. Miranda and Paul L. Fackler. *Applied Computational Economics and Finance.* MIT Press, Cambridge, MA, USA, 2002. ISBN 0262134209.