Petrel Plugins for Declustering and Debiasing

John Manchuk, Chad Neufeld and Clayton V. Deutsch

Centre for Computational Geostatistics
Department of Civil & Environmental Engineering
University of Alberta

Modeling reservoirs with few wells is challenging. Declustering and debiasing tools are useful with sparse or biased drilling; however, many commercial software packages do not implement declustering or debiasing. Using Petrel's Ocean API, the declustering and debiasing engines have been written as Petrel plugins. This allows Petrel users to do declustering or debiasing in Petrel and to use the output as part of the petrophysical modeling process.

Introduction

All geostatistical simulation algorithms aim to reproduce a set of input statistics. These include continuous histograms, facies proportions, and variograms. Getting representative statistics to input to the simulation can be challenging, especially with limited well data or clustered sampling.

There are two different methods that can be used to help determine the true distribution or proportions of a property; declustering and debiasing. Although their goals are similar, declustering and debiasing are two very different methods. Declustering is based on the spatial positions of the wells. Clustered wells will receive a lower weight than wells that are spaced far apart. Debiasing uses a secondary variable for determining the distribution of the primary variable conditional to the secondary. The goal of this paper is not to explain declustering or debiasing. There are many good references for this [1, 2, 3, 6]. The goal of this paper is to present a plugin for Petrel that does declustering and debiasing.

The current version of Petrel, or most other commercial software, does not allow the user to explicitly perform declustering and debiasing. Since these methods can have a huge impact on the simulation results, we felt it was important for declustering and debiasing to be the first plugins written by CCG for Petrel.

Petrel Ocean API

Schlumberger Information Systems released an application programming interface (API) for Petrel in its 2007.1 release called Ocean [8]. Ocean gives end users of Petrel the ability to add their own custom algorithms and processes right into Petrel. Each additional process can also have it own Petrel-like users interface. It means that we can transfer our algorithms to companies and they can be included right into Petrel. Someone who wants to try one of our tools can do everything in Petrel. They will not need to export data, reformat it, run the new algorithm from the command line, and import the results.

Ocean allows plugins access to many of the native Petrel data and object types. All of them can be read, some existing objects can be modified and other object types can be created. Ocean even allows custom objects to be defined and saved in Petrel. An example of a custom object type is the output from declustering. The custom object contains the mean versus cell size summary and the declustered distribution.

A special licence for Petrel and Ocean is required to develop plugins [8]. However, no special licence is required to use them, unless the plugin developer requires it.

Declustering and Debiasing DLLs

The first stage of the plugin development was to convert the declustering and debiasing from stand-alone programs to DLLs. The DLLs will then be called from the Ocean plugin. This allows a common engine for declustering and debiasing to be used. In addition, there will not be any errors introduced by translating from Fortran to C#.

The declustering and debiasing engines were combined into a single DLL, `ccgEng.dll`. The DLL can be called from an executable or another DLL. When the DLL is called, a set of arguments must be passed from the calling program. The arguments include input data for the DLL, parameters for running the algorithm within the DLL and arguments for the DLL to return to output to the calling program. See paper #402 in this years report for a discussion on programming for plugins [5].

The code exporting the declustering subroutine within the DLL is below. It lists the input and output arguments and what they are. The arguments can be real or interger, single values or arrays. The arguments passed to the DLL must be exactly the same as what the DLL is expecting for it to work right.

```
subroutine declus(ndata, x, y, z, v, numcells, cellmin, cellmax, numoff, yanis, &
                  zanis, tmin, tmax, option, dtype, cellsize, cellsizes, mean, &
                  means, weights, ierror )
   !DEC$ ATTRIBUTES DLLEXPORT::declus
   !DEC$ ATTRIBUTES DECORATE, ALIAS:'declus'::declus
   use gstype
   implicit none
   integer(IP), intent (in) :: ndata, numcells, numoff, option, dtype
   real(RP),    intent (in) :: x(ndata), y(ndata), z(ndata), v(ndata)
   real(RP),    intent (in) :: cellmin, cellmax, yanis, zanis, tmin, tmax
   integer(IP), intent (out) :: ierror
   real(RP), intent (out) :: cellsize, cellsizes(numcells), mean, means(numcells)
   real(RP), intent (out) :: weights(ndata)
   interface declusEng
      subroutine declusEng ( x, y, z, v, numcells, cellmin, cellmax, numoff, &
                             yanis, zanis, tmin, tmax, option, dtype, cellsize, &
                             cellsizes, mean, means, weights, ierror )
         use gstype
         implicit none
         real(RP),    intent (in) :: x( : ), y( : ), z( : ), v( : )
         real(RP),    intent (in) :: cellmin, cellmax, yanis, zanis, tmin, tmax
         integer(IP), intent (in) :: numcells, numoff, option, dtype
         integer(IP), intent (out) :: ierror
         real(RP), intent (out) :: cellsize, cellsizes( : ), mean, means( : )
         real(RP), intent (out) :: weights( : )
      end subroutine
   end interface

   call declusEng ( x, y, z, v, numcells, cellmin, cellmax, numoff, yanis, zanis,&
                    tmin, tmax, option, dtype, cellsize, cellsizes, mean, means,&
                    weights, ierror )
end subroutine declus
```

For example, the input to the DLL is passed in 4 one-dimensional arrays, `X, Y, Z` and `V,` 5 integers and 6 real numbers. The output is returned via a similar mixture of arrays and numbers. The debiasing DLL is setup in exactly the same fashion.

The details of how the arguments are passed and returned are hidden from the user. All of the data formatting, conversions and manipulations are handled as part of the Petrel plugin. There is no requirement for the user to do any pre- or post-processing.

Petrel Plugin Basics

The Petrel plugin is the data handling and interface for the declustering and debiasing processes. It gets the required input from the user, reads the data from Petrel, calls the DLL, receives the output from the DLL, and saves the data into Petrel objects.

The input from the user includes data sources, program parameters and output objects. The plugin will then read all of the input data from the different Petrel objects. Any data formatting or conversions are done inside the plugin. This may include reordering of grids and other pre-processing steps. Then the plugin will call the DLL and pass it all of the required information. There is a special section in the plugin that defines what arguments the ccgEng DLL is expecting. The C# code for setting up the plugin-DLL link is:

```
class dll_declus
{
    [DllImport("ccgEng.dll")]
    public static extern void declus(
        [MarshalAs(UnmanagedType.I4)] ref int ndata,
        [MarshalAs(UnmanagedType.LPArray)] double[] dataArrayX,
        [MarshalAs(UnmanagedType.LPArray)] double[] dataArrayY,
        [MarshalAs(UnmanagedType.LPArray)] double[] dataArrayZ,
        [MarshalAs(UnmanagedType.LPArray)] double[] dataArrayVar,
        [MarshalAs(UnmanagedType.I4)] ref int numberCells,
        [MarshalAs(UnmanagedType.R8)] ref double cellMin,
        [MarshalAs(UnmanagedType.R8)] ref double cellMax,
        [MarshalAs(UnmanagedType.I4)] ref int numberOriginOffsets,
        [MarshalAs(UnmanagedType.R8)] ref double YcellAnisotropy,
        [MarshalAs(UnmanagedType.R8)] ref double ZcellAnisotropy,
        [MarshalAs(UnmanagedType.R8)] ref double tmin,
        [MarshalAs(UnmanagedType.R8)] ref double tmax,
        [MarshalAs(UnmanagedType.I4)] ref int declusOption,
        [MarshalAs(UnmanagedType.I4)] ref int declusType,
        [MarshalAs(UnmanagedType.R8)] ref double optsize,
        [MarshalAs(UnmanagedType.LPArray)] double[] declusCellSizes,
        [MarshalAs(UnmanagedType.R8)] ref double optmean,
        [MarshalAs(UnmanagedType.LPArray)] double[] declusMeans,
        [MarshalAs(UnmanagedType.LPArray)] double[] declusWeights,
        [MarshalAs(UnmanagedType.I4)] ref int ierror
    );
}
```

The order of the arguments is exactly the same as the order shown in the Fortran DLL above. This lets the C# Petrel plugin know exactly what arguments need to be passed to the DLL.

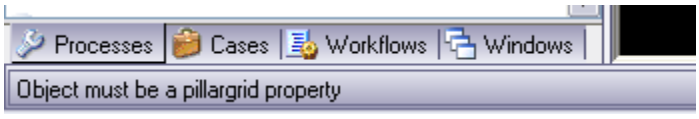
The DLL uses all of the inputs and runs the declustering or debiasing process with the same code that is part of the stand-alone program. This is attractive from a maintenance standpoint, because if an error is discovered, only one piece of code needs to be changed, the engine that is common the DLL and the stand-alone program. It is simpler than trying to maintain multiple pieces of the same code.

After the subroutine in the DLL is finished, it will return the output from the run to the calling program. For the declustering plugin, this includes a summary of the cell size versus declustered mean, the cell size that minimizes or maximizes the mean, and an array of the declustering weights. The output is stored in a custom domain object inside of Petrel.

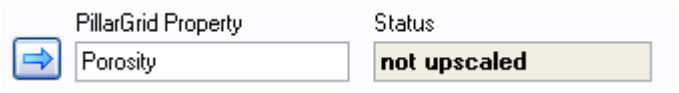
Plugin Details

The declustering and debiasing plugins can be run via the process menu or the workflow manager in Petrel. Double-clicking on the plugin in the process menu or in a workflow opens the user interface. Figure 1 shows how the plugins are access through Petrel. Figure 2 shows the declustering and debiasing user interfaces.

Adding the input data is done by selecting an object and clicking the blue arrow, or by dragging and dropping the object into the plugin window. The declustering and debiasing plugins require upscaled well data on a pillar grid. If you try to use a non-pillar grid object you get the following message:



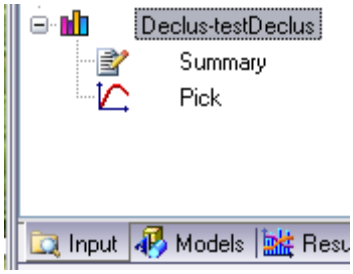
The plugin checks the pillar grid to ensure that an upscaled property is present. If there is no upscaled property, the following will be displayed:



When there is an upscaled property the status changes to:

After the pillar grid property is chosen, the declustering parameters need to be set. They include: (1) the y/x and z/x cell anisotropies, (2) the minimum and maximum cell sizes, (3) the number of cell sizes, (4) the number of origin offsets, and (5) to look for the minimum or maximum declustered mean. An example declustering setup is shown in Figure 3.

Clicking apply or ok will run the process. The output is stored in a custom declustering object. A 2-D plot can be made of the declustered mean versus cell size. After creating a function window, a check box will show up beside the Summary object. Selecting the checkbox will display the mean versus cell size plot. Figure **4** shows an example declustered mean versus cell size plot.



The output from the declustering or debiasing is a reference distribution. It is comprised of a set of data values and a weight for each of the values. We want to use this information for generating a simulated model. It is easy to use the reference distribution in the petrophysical modeling process. The second tab in the modeling process is for the distribution. Here, you specify the distribution type. There are three options: (1) from upscaled logs, (2) normal distribution and (3) from a general distribution. Usually the upscaled logs are used. This means that the data will be transformed to Gaussian units, the simulation ran, and then backtransformed using the histogram of the upscaled log data. The normal distribution option does not transform the data. It assumes that the input data is normal score transformed already. The last option is where we can use the declustering or debiasing output. The general distribution option lets us specify what the exact input and output distribution is. The input data will be transformed using the specified general distribution and the simulation results will be backtransformed using the same distribution. Figure 5 shows the general distribution option being used from the modeling process.

Conclusions

Schlumberger recently released the Ocean API for petrel. It allows custom process to be included inside of Petrel. Each process can have its own users interface and looks and feels just like Petrel.

Using Ocean, the declustering and debiasing routines have been written into a Petrel plugin. Declustering and debaising can be extremely important algorithms when working with sparse data or biased samples. This will allow petrel users to better model reservoirs with sparse data.

References

[1] J.P. Chilès and P. Delfiner. *Geostatistics: Modeling Spatial Uncertainty*. Wiley Interscience, New York, 1999.

[2] C.V. Deutsch. *Geostatistical Reservoir Modeling*. Oxford University Press, New York, 2002.

[3] C.V. Deutsch and T. Dose. *Programs for Debiasing and Cloud Transformation:* `bimodel` *and* `cltrans`. In *Centre for Computational Geostatistics, Paper 404, Annual Report 7*, 2005.

[4] C.V. Deutsch and A.G. Journel. *GSLIB: Geostatistical Software Library and Users Guide*. Oxford University Press, New York, second edition, 1998

[5] C. Neufeld and C.V. Deutsch. *Programming Tips for Plugins.* In *Centre for Computational Geostatistics, Paper 402, Annual Report 9*, 2007.

[6] M.J. Pyrcz and C.V. Deutsch. *Declustering and Debiasing*. In *Centre for Computational Geostatistics, Paper 62, Annual Report 4*, 2002.

[7] Schlumberger Information Systems. *Ocean 2006 Developer's Guide*. Schlumberger Information Solutions, Houston, Texas, 2006.

[8] Schlumberger Information Systems. *Ocean Portal*. www.ocean.slb.com

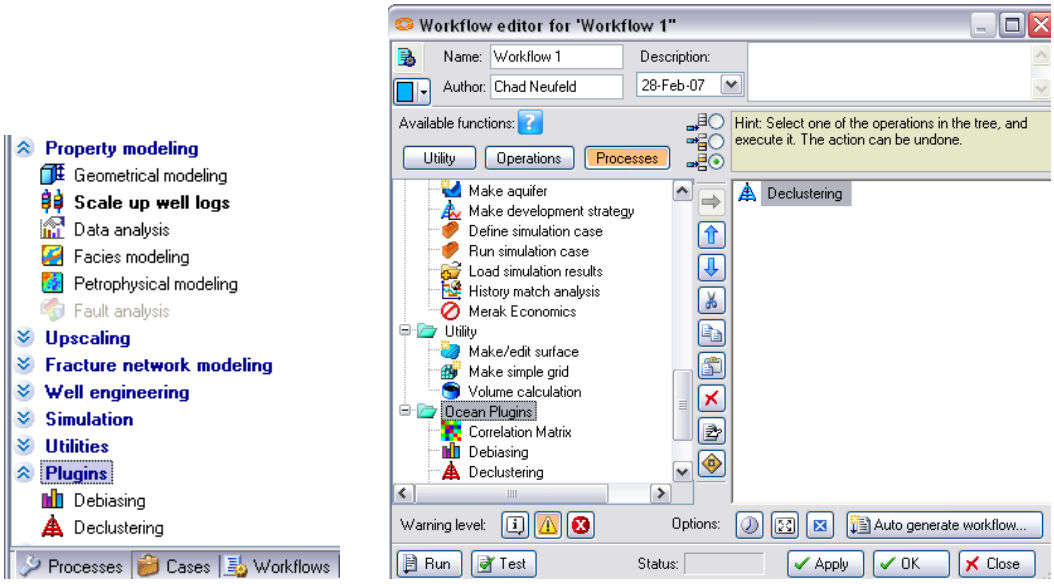


Figure 1: Accessing the declustering plugin from the process menu or the workflow manager.

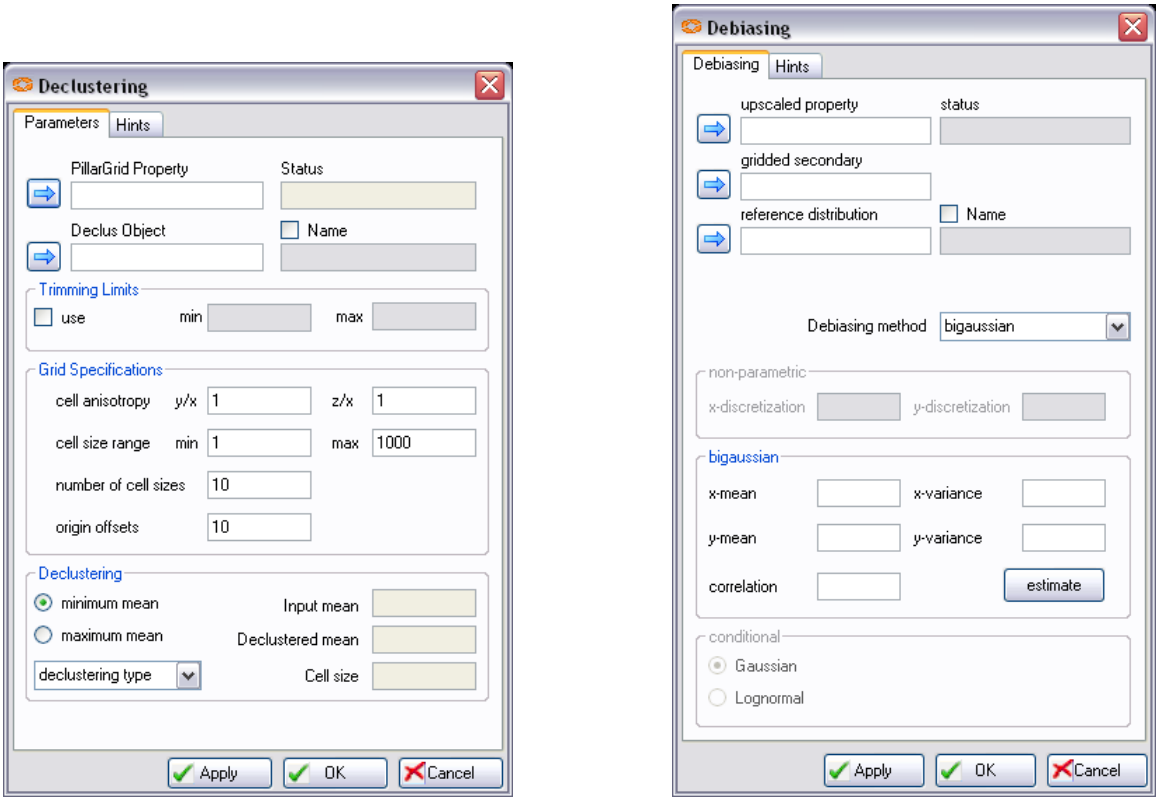


Figure 2: Declustering and debiasing user interfaces in Petrel.

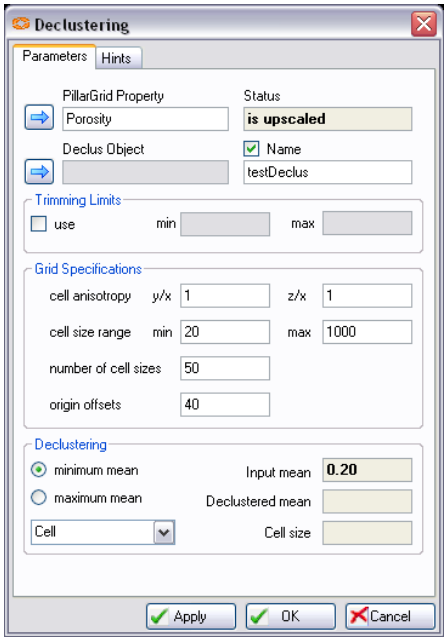


Figure 3: Example declustering setup in Petrel.

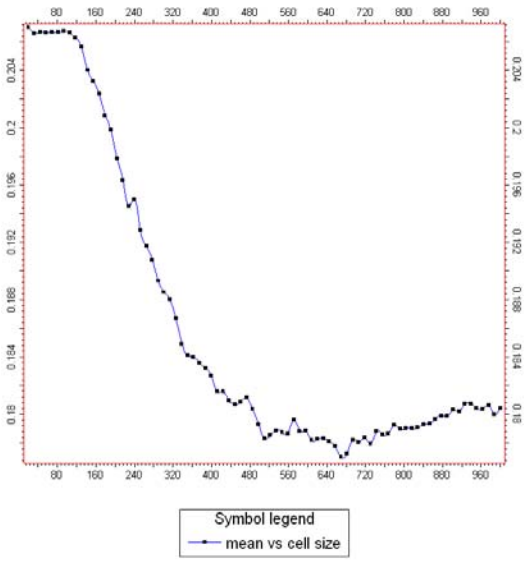


Figure 4: Declustered mean versus cell size in Petrel.

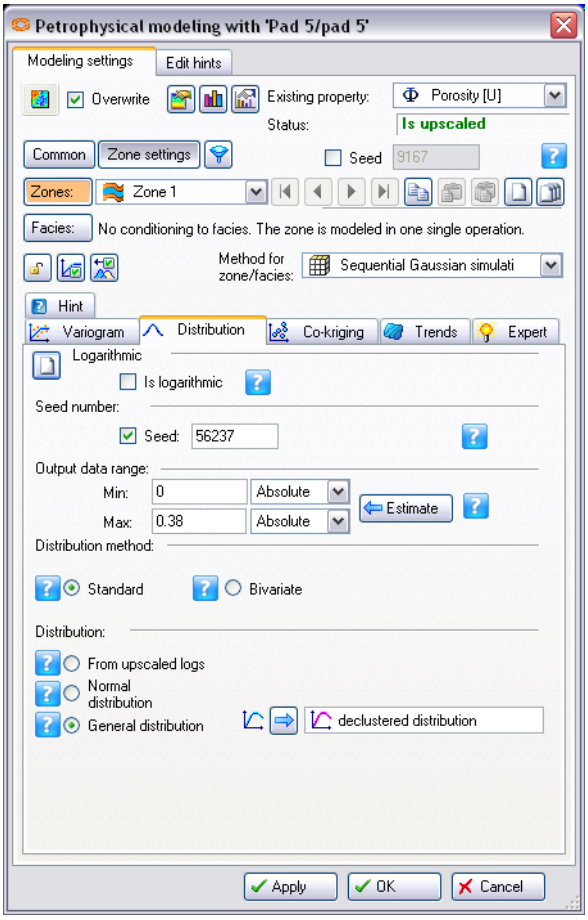


Figure 5: Using the declustering or debiasing output for petrophysical modeling.