User Manual

http://people.sc.fsu.edu/~jburkardt/presentations/cvt_uman.pdf

.....

John Burkardt

February 2, 2024

1 Introduction

In this chapter, we present a library of routines that implement the Centroidal Voronoi Tessellation algorithm. The library is written in FORTRAN 90 ("F90"). It is intended to allow a user to define the geometry of a problem, carry out the CVT algorithm to determine generator points, and to collect data about the moments of the resulting Voronoi cells.

The user must prepare a calling program that specifies parameter values, calls the library routines in an appropriate order, and saves the desired output to files. One of the major tasks for the user is defining the geometry of the region to be analyzed. This task is simplified, however, if the user already has a **DIATOM** input deck, because the library has an interface with **DIATOM**, and can be instructed to have **DIATOM** read that deck and participate in the analysis of the region.

A sample version of a calling program and a **DIATOM** input deck will be briefly described below, and may be the quickest way of understanding how the library is intended to work. Further guidance will be provided on the purpose of the individual routines that make up the library, and the meaning of the program parameters of user interest.

2 User Control

The current version of the code includes a great deal of flexibility for the user. The user can specify the problem in several ways, and modify the way the CVT algorithm is carried out. In many cases, a great deal of programming is available, which the user can access by simply setting a switch or a parameter.

The most critical information that the code needs is the geometry of the region. The user must specify **NDIM**, the spatial dimension of the region, and **BOX_MIN** and **BOX_MAX**, the size of a bounding box that contains the region. The code also must be able to query the region, that is, find out whether a given point is in the region. By choosing the value of **USE_DIATOM**, the user may specify that inquiries about the region will be passed, via **DIATOM_POINT_TEST2**, to **DIATOM**, which will then access the user's **DIATOM** geometry input file, or that queries will be handled directly by an appropriately modified version of the example routine **TEST_REGION**.

For the CVT iteration, the user may specify how the generators are initialized by setting **RANDOM_GENERATOR**. The user then calls the CVT iteration step routine a number of times, with control returning to the user after each step. In the sample main program, the iteration is simply carried out **MAXIT** times, with no other iteration control.

The CVT iteration requires the random sampling of the region. The user may specify the value of **NS_CVT**, the number of sampling points per Voronoi generator.

As part of the CVT iteration, a nearest neighbor calculation is carried out. For problems with a large number of generator points, this calculation can dominate the problem execution time. In such cases, the code offers an alternative method to the naive nearest neighbor calculation, using bins. The user can try out

this option by setting **USE_BINS**, and then specifying in the **NBINS** array the number of bins to use in each dimension.

Once the CVT iteration is completed, the moments are calculated, again using random sampling. The user may specify the value of **NS_MOM**, the number of sampling points per Voronoi generator, to control the accuracy of this calculation.

3 The Overall Structure of the Code

The information that comprises a complete running version of the code can be divided into five logical groups.

The first portion of the code is the master routine, a user main program which defines parameters, sets up data structures, calls library routines to carry out the algorithm, and writes out important data.

The second portion of the code is the CVT library. This comprises a number of F90 routines that carry out the CVT algorithm. Many of these routines have optional switches that the user can set, in order to vary the nature of the algorithm.

The third portion of the code is **DIATOM**, a library of routines in C and FORTRAN, supplied by Sandia.

The fourth portion of the code comprises two interface routines, both of which are written in C, and currently included in the **DIATOM** library. The first routine, **DIATOM_SETUP**, causes a **DIATOM** input file to be read and stored in the **DIATOM** geometry structure. This routine currently forces the geometry to be 3D, rectilinear, and contained in the bounding box [0,100] by [0,100] by [0,20]. It would be much preferable to have these choices made by the user and passed from the main program to **DIATOM_SETUP** as arguments. However, it was thought unnecessary to refine this interface with the current experimental version of the code. The second routine, **DIATOM_POINT_TEST2** allows the CVT routines to query the physical region stored in **DIATOM**, with a response of +1, -1, or 0 depending on whether the point is inside, on the boundary, or outside the region.

The fifth portion of the code is simply a **DIATOM** input file that defines the geometry of the region of interest. In order to have **DIATOM** read this file, the main program must call the interface routine **DIATOM_SETUP**. As currently written, **DIATOM** expects the input file to have the name *diatom_test.in*, another hardwired feature that would be made more flexible in a production version of the program.

4 The Main Program

We present here a schematic version of a main program, emphasizing the input and output variables of primary interest to the user, and the order in which the CVT library routines should be called. Later sections of this report list the definitions of all the variables of interest to the user, and the purpose of the various subroutines.

For clarity, we suppress the display of less important data and routines, particularly those used for bin searches. For a full working example, refer to the main program included with the code distribution. For an explanation of the purpose of the individual subroutines invoked here, see the section which discusses the library subroutines. For an explanation of the meaning and purpose of the variables, see the section containing the variable glossary.

```
integer, parameter :: n = 256
integer, parameter :: ndim = 3

real, dimension ( ndim ) :: box_min = (/   0.0,   0.0, 0.0 /)
real, dimension ( ndim ) :: box_max = (/ 100.0, 100.0, 5.0 /)
integer, parameter :: maxit = 10
integer, parameter :: ns_cvt = 5000
```

```
  integer, parameter :: ns_mom = 5000
  logical, parameter :: use_bins = .false.
  logical, parameter :: use_diatom = .true.

  double precision cell_centroid(ndim,n)
  double precision cell_generator(ndim,n)
  double precision cell_moment(ndim,ndim,n)
  double precision cell_volume(n)

!  Initialize DIATOM.

  call diatom_setup ( )

!  Set initial estimate of cell generators.

  call generator_init ( ndim, box_min, box_max, n, cell_generator, &
    use_diatom, ... )

!  Carry out the CVT iteration.

  do it = 1, maxit

    call cvt_iteration ( ndim, box_min, box_max, n, cell_generator, ns_cvt, &
      use_diatom, use_bins, ... )

  end do

!  Compute moments.

  call vcm ( ndim, box_min, box_max, n, cell_generator, ns_mom, use_diatom, &
    use_bins, ..., cell_volume, cell_centroid, cell_moment )

end
```

5 The DIATOM Geometry File

If, in the main program, the user sets the logical parameter **USE_DIATOM** to **TRUE**, and calls **DIATOM_SETUP** before the CVT routines are invoked, then the code will expect the region to be defined by a **DIATOM** input file.

The general structure of such input files is outside the scope of this report. However, we display below the 3D version of the **DIATOM** test region, which is the union of a box and a cylindrical shell. The **DIATOM** input file is simply:

```
diatom

  package 'box'
    density 1
    insert box
      p1 = 45,35,0
      p2 = 55,90,20
```

```
  endinsert
endpackage

package 'cylinder'
  density 1
  insert cylinder
    ce1 = 50,0,0
    ce2 = 50,0,20
    r = 40
    ri = 30
  endinsert
endpackage
```

6 Library Subroutines

Another way of looking at the code is to consider how control passes from one routine to another and back, as the algorithms that make up the computation are carried out. Starting with the user's main program, we will look at the more interesting routines that are called.

Note that we will not delve any further into **DIATOM** beyond the two interface routines, and that the nearest neighbor bin routine includes a 2D version, and many lower level routines that will not be considered here.

6.1 The User Main Program

The user main program **"MAIN"** is responsible for setting parameters, defining data, calling and controlling the library routines, and processing output of interest to the user. It generally calls the following set of routines:

- **RANDOM_INITIALIZE** initializes the F90 random number seed.
- **DIATOM_SETUP** initializes **DIATOM** and reads the input file.
- **GENERATOR_INIT** initializes the Voronoi cell generators.
- **CVT_ITERATION** takes one step of the CVT iteration.
- **VCM** calculates Voronoi cell volumes, centroids and second moments.

If bins are to be used for the nearest neighbor calculation, then **MAIN** will also need to preprocess the generator points after they are initialized and whenever they are adjusted. (See chapter 1 for details on the nearest neighbor algorithm, listed as Algorithm 5). The master routine for this operation is:

- **BIN_PREPROCESS** controls the bin preprocessing step.

BIN_PREPROCESS calls the appropriate routines to set up the auxilliary data structures required. For the 3D case, these routines are:

- **D3VEC_BIN_EVEN3** places the elements of an array of 3D vectors into evenly spaced bins.
- **D3VEC_BINNED_REORDER2** reorders the elements of an array by bin;
- **D3VEC_BINNED_SORT_A2** sorts the elements within each bin.

There is a corresponding set for 2D problems. These routines, in turn, call many lower level routines which will not be discussed here.

6.2 The CVT algorithm

The CVT calculation, listed as Algorithm 3 in Chapter 1, is embodied in the routines **GENERATOR_INIT** and **CVT_ITERATION**.

GENERATOR_INIT uses sampling to produce a set of **N** points to be used as initial values for the generators, and calls

- **REGION_SAMPLER** produces a random point X in the region.

CVT_ITERATION is called repeatedly to apply one step of the probabilistic Lloyd's method. To do so, it calls the routine:

- **REGION_SAMPLER** produces a random point X in the region.

and then, to find the nearest generator to the point X, it calls *one* of the following two routines:

- **FIND_CLOSEST** uses a naive algorithm.
- **POINTS_NEAREST_POINT_BINS3_3D** uses bins.

6.3 The Moment Calculation

The calculation of volumes, centers of mass, and second moments, listed as Algorithm 6 in Chapter 1, is done by **VCM**, which uses sampling to estimate the integrals in the definitions of these quantities. It calls the routine:

- **REGION_SAMPLER** produces a random point X in the region.

and then, to find the nearest generator to the point X, it calls *one* of the following two routines:

- **FIND_CLOSEST** uses a naive algorithm.
- **POINTS_NEAREST_POINT_BINS3_3D** uses bins.

6.4 Sampling

A recurring task in the code is the sampling of the region, carried out by Algorithm 4, as described in Chapter 1. Sampling is done by choosing a random point in the bounding box and determining if the point lies within the physical region.

The routine **REGION_SAMPLER** supervises the sampling procedure. A random point within the bounding box is chosen by calling either the F90 routine **RANDOM_NUMBER**, or the Halton routine **I_TO_HALTON_VECTOR**. Then the point is tested to determine whether it is within the physical region, by calling either **TEST_REGION** or **DIATOM_POINT_TEST2**. The generation and testing process is repeated, if necessary, until the sampling routine can return a suitable point.

Depending on the user options, **REGION_SAMPLER** calls one of:

- **RANDOM_NUMBER**, the F90 uniform random number generator,
- **I_TO_HALTON_VECTOR** computes an element of a vector Halton sequence.

and then, depending on the value of **USE_DIATOM**, the point is tested by calling one of:

- **TEST_REGION** determines if a point is within a region defined by formulas.
- **DIATOM_POINT_TEST2** determines if a point is within a region defined by **DIATOM** input.

6.5 Nearest Neighbor

Given a point X in the region, and the current set of CVT generators, both **CVT_ITERATION** and **VCM** need to determine the generator that is nearest to X. The user's choice of **USE_BINS** determines which of the following routines is called:

- **FIND_CLOSEST** finds the nearest generator to X, naively.
- **POINTS_NEAREST_POINT_BINS3_3D** finds the nearest generator to X, using bins.

POINTS_NEAREST_POINT_BINS3_3D is an implementation of Algorithm 5.

Note that, if bins are to be used, a preprocessing step must be carried out by the user main program, and certain auxilliary bin data must be passed to the routine carrying out the nearest neighbor calculation. The required data structures and preprocessing calls for the bin approach can be determined by examining the sample calling program. The bin preprocessing and computation routines call some lower level routines which will not be discussed here.

7 Variable Glossary

In the following tables, we give the names and meanings of various program variables that may be of interest to a user of the program.

7.1 Geometric Variables

The geometric variables are used to define the physical region being analyzed. The user must provide the spatial dimension, a rectangular box surrounding the region, and indicate whether **DIATOM** is to be used.

Name	Description
NDIM	integer, the spatial dimension, which should be 2 or 3.
BOX_MIN	double precision (NDIM), the minimum coordinates of the bounding box.
BOX_MAX	double precision (NDIM), the maximum coordinates of the bounding box.
USE_DIATOM	logical, is TRUE if **DIATOM** is to be called to determine whether a point lies in the physical region; if it is FALSE than a user-supplied subroutine **TEST_REGION** is called.
DR	double precision, a tolerance used by **DIATOM** when testing whether a point is within, outside of, or on the boundary of the physical region. This is typically a very small value, of the order of 10^{-5} or smaller.

7.2 CVT Algorithm Variables

The CVT algorithm variables specify the number of generators, how they are initialized, how the sampling steps are carried out, and how many steps are to be taken. Once the CVT algorithm is completed, the CVT generators are known.

Name	Description
N	integer, the number of Voronoi cells to generate. The sample problem uses a value of 256.
MAXIT	integer, the maximum number of correction iterations used in the CVT calculation. A typical value is 100.
NS_CVT	integer, the average number of sampling points tested per Voronoi cell, on each step of the CVT iteration. A typical value is 5000.
RANDOM_GENERATOR	integer, specifies how the Voronoi cell generators are to be initialized. 0: use the F90 RANDOM_NUMBER routine; 1: use the sequence (PREFERRED).
CELL_GENERATOR	double precision (NDIM,N), the CVT generator points. This is the output quantity of most interest.

7.3 Moment Calculation Variables

The moment calculation follows the determination of the CVT generators. It also uses a sampling method; in particular, the same value of **NS** is used to determine the number of sampling points. This calculation produces the values of the cell volumes, centroids, and second moments.

Name	Description
NS_MOM	integer, the average number of sampling points tested per Voronoi cell for the moment calculations. A typical value is 5000.
REGION_VOLUME_GIVEN	logical, TRUE: the region volume is input in REGION_VOLUME. FALSE: the region volume must be approximated.
REGION_VOLUME	double precision, the volume of the region. If REGION_VOLUME_GIVEN is TRUE, then REGION_VOLUME is input by the user. Otherwise, REGION_VOLUME is approximated computationally.
CELL_VOLUME	double precision (N), the volume of the Voronoi cells.
CELL_CENTROID	double precision (NDIM,N), the centroids of the Voronoi cells.
CELL_MOMENT	double precision (NDIM,NDIM,N), the second moments of the Voronoi cells.

7.4 Nearest Neighbor Variables

If efficiency is an issue, the sampling method may be speeded up by using bins to group the generators. The coding for this procedure is somewhat complicated. However, to use bins, it is only necessary to specify **USE_BINS** to be **TRUE**, to choose appropriate values **NBIN** for the number of bins in each dimension, and to call the appropriate routines, as presented in the sample calling program. The other quantities are computed automatically.

Name	Description
USE_BINS	logical, is TRUE if the bounding box is to be divided up into bins to speed up the nearest neighbor search; FALSE if the nearest neighbor seach is to be done naively.
NBIN	integer(3), the number of bins to use in each direction. For 2D problems, set NBIN(3) = 1. For efficiency, these values should be set in such a way that the bins are nearly square or cubical.
BIN_START	integer (NBIN(1),NBIN(2),NBIN(3)), the index of the first cell center in the bin, or -1 if none.
BIN_LAST	integer (NBIN(1),NBIN(2),NBIN(3)), the index of the last cell center in the bin, or -1 if none.
BIN_NEXT	integer(N), the index of the next cell center in the bin containing this cell center.

8 Miscellaneous

The CVT library is written in FORTRAN 90.

Floating point quantities are declared DOUBLE PRECISION. The **DIATOM** library declares floating point quantities using the symbolic type **REAL**, which we currently have set to "double". The F90 code and **DIATOM** must always agree in this way on the size of floating point values.

UNIX FORTRAN compilers have a tradition of storing FORTRAN symbolic names with an appended underscore. C compilers do not do this. So if a FORTRAN routine wants to call a C procedure named "solve", a problem arises, because the FORTRAN compiler will, by default, convert this into a call to "solve_". Most FORTRAN compilers have a way of requesting that underscores not be appended to symbolic names during compilation, and this was necessary in the development of the CVT library, in order to interface with **DIATOM**.

Another issue is the difference in the default handling of the arguments of a procedure in FORTRAN and C. The interface routine **DIATOM_POINT_TEST**, supplied by Sandia, had a call by value. This is the reason that a revised version, called **DIATOM_POINT_TEST2** and using call by address, had to be written, for use as an interface between the F90 CVT library and **DIATOM**.

During development of the code, CPU and real time measurements were made. There is an F90 standard routine called **CPU_TIME** for CPU time measurement. The implementation of this routine on the DEC Alpha was unreliable for CPU times that were more than about 30 minutes. Apparently, an internal integer would wrap around, resulting in negative CPU time intervals. Hence, CPU timings were also done with the UNIX routine **ETIME**. However, because the compiler was instructed not to append an underscore to symbolic names, but the compiled version of **ETIME** actually has an underscore, the calls to **ETIME** had to be explicitly made to **ETIME_**. Real time measurements were made by calls to the F90 routine **SYSTEM_CLOCK**, for which wrap-around problems did not seem to arise.

DIATOM has some internal source code settings that specify the spatial dimension, bounding box, and geometry type. These quantities are currently hard-wired in **DIATOM_SETUP**. This means that a user may have to enter certain information in two places in order to get a correct run. A proper approach would allow the user to choose these values in the main program and pass them to **DIATOM_SETUP** by its argument list, avoiding the possibility of a catastrophic disagreement in problem definition.