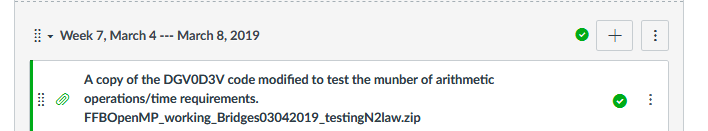
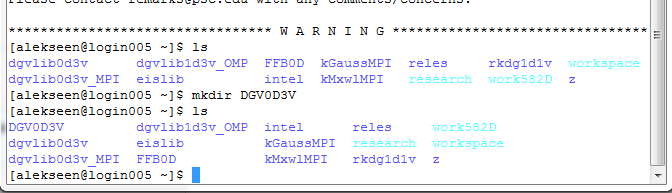
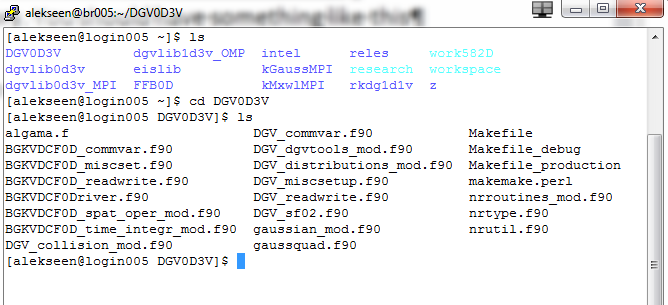
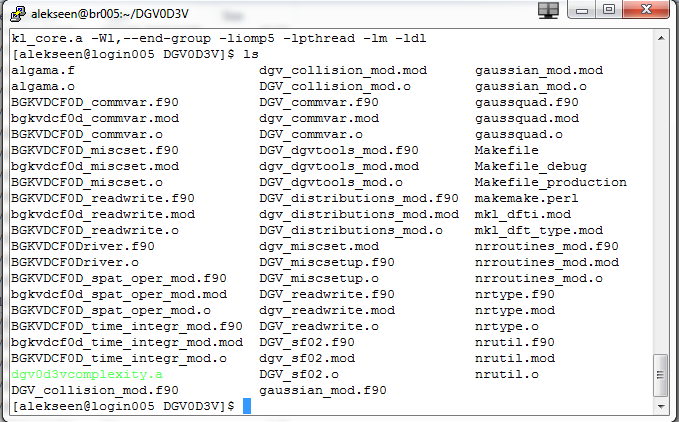
Steps. Week 7  
 Setting up DGVlib0D3V code for Evaluating Complexity of the Method.

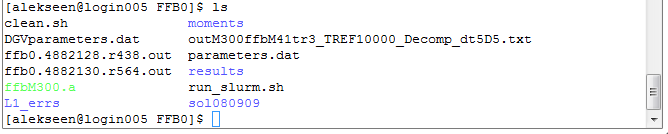
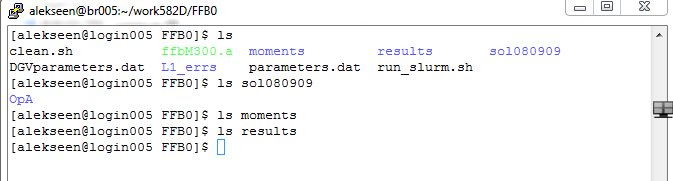
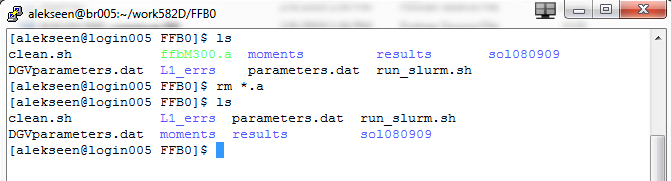
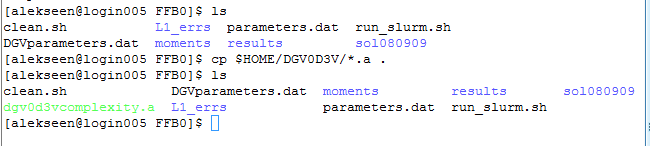
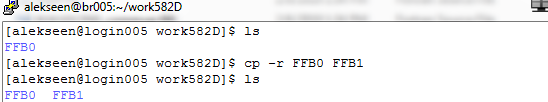
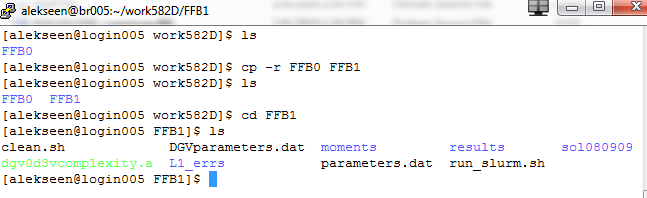
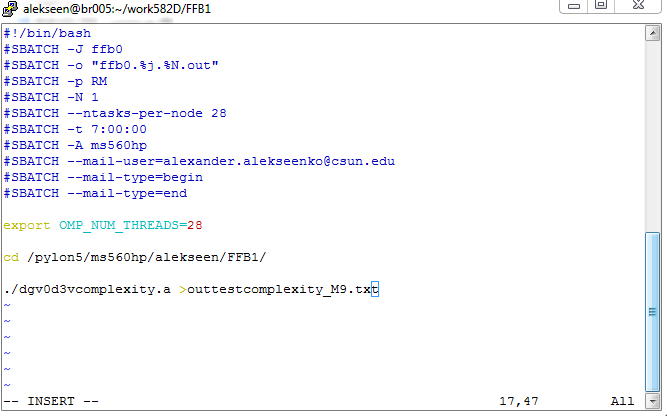
*Steps to perform various tasks related to the DGVlib0D3V topics.*

**Part 1. Steps to compile DGVlib0D3V codes.**

* A copy of the DGV0D3V code have been placed on the course Moodle page for Week 7.  
    
    
    
   This is the same code implementing nodal DG velocity discretizations of the collision integral that we were using in Project 1 to produce spatially homogeneous solution. However for the purposes of Project 2, the code was modified to perform a single evaluation of the collision operator and then stop.
* Our goal for the first simulation is to use time measuremements to estimate complexity of the nodal-DG discretization algorithms. We will consider two ways to evaluate the collision operator: direct evaluation of the convolution form and evaluation using the Fourier transform.
* Start by downloading a copy of the code from the course web page and moving the copy to Bridges.
  + On Bridges, in your HOME directory you should create a separate directory for the, e.g. DGV0D3V.   
    
  + Copy the code to DGV0D3V directory. You should have something like this  
    
  + If you have objective files in the directory, delete them by running the command  **rm \*.mod \*.o**  
      
    When there no .o and .mod files present, run command   
      
    **make**  
      
    to compile the code and produce an executable:  
      
      
      
    the default name for the executable is dgv0d3vcomplexity.a
  + Please note that in the future we will modify this code. Repeat the previous step to produce an executable if necessary.

**Part 2. Steps to set up working directory for DGV0D3V**

Great news! If you were able to run the code in Project 1, then it means that a working directory was properly set up. You can use the same directory that you used for Project 1. All you have to do is clean the directory, update the executable and set the run\_slurm.sh for the run. Below are the steps to prepare directory for the new run

* After a successful run you will have a number of files in the working directory. It will look something like this:  
  
* The easiest way to clean the directory is to use the shell file clean.sh that should be included in the directory. If you have this file, run the following command:   
    
  **bash cleah.sh**  
  it should remove all the results of previous run.
* If the file is missing or if you like to try your hand at deleting files, you can clean the directory by typing a bunch of **rm** command, or by typing one rm command but pointing to many files:   
    
  **rm \*.txt \*.out sol080909/\*.dat results/\*.dat moments/\*.txt**
* Once finished, you should have no files in directories moments and results and no files in sol080909 (but there will be sub directories). You can check that using the command ls. Here is what you should be expecting to see:   
    
  
* The executable files \*.a are usually not deleted between different simulations of the same problem. However, we prepared a modified code to use in Project 2. This means that we will need to delete the executable and replace is with a different one. Run command   
    
  **rm \*.a**  
  to delete the executable   
  
* Now we need to copy the newly compiled executable to the work directory. We know that the executable is in the folder $HOME/DGV0D3V and it is the only .a file there, so we just issue the command to copy all .a to the current directory. Since only one \*.a exists in the directory, only one is copied  
    
  **cp $HOME/DGV0D3V/\*.a .**  
  Here is the result:  
  
* Now you may want to clone the working directory. This may become handy if you will decide to submit several jobs to execution at once. A good rule is that you have each job is started in a separate working directory. Top clone a directory you can use **cp** command. For example,   
   **cp –r FFB0 FFB1**  
    
    
    
  Please note that **run\_slurm.sh** in the new directory will have to be adjusted to point to correct directory.
* Next we will discuss how to make sure that run\_slurm.sh points to the correct directory
* Switch to the new directory, e.g., FFB1. You will see something like this:  
  
* Use **vim** editor to open **run\_slurm.sh:**  
  
* Make sure that the script switches to the correct working directory. For example, if you are in directory FFB1, then make the following adjustments   
    
  cd /pylon5/ms560hp/yourusername/FFB1/
* You also need to make sure that the name of the executable is correct. Since the name of the executable that we need is dgv0d3vcomplexity.a, we adjust the last line to   
    
  ./dgv0d3vcomplexity.a >outtestcomplexity\_M9.txt
* Here is the final result  
    
    
  Save and exit for now. You will have to come back and adjust other parameters to work on the simulation. The working directory is set. We now need to describe how to change the parameters and to start a simulation.

**Part 3. A brief description of the modification in the code.**

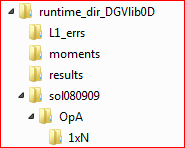
We now need to discuss how to set up our numerical experiment. In this experiment we will perform evaluations of the Boltzmann collision operator using nodal-velocity discretization with different numbers of velocity cells.

For the sake of this experiment, the code was modified to only do what we need to do. Actually, one subroutine in one file was modified. The file is **DGV\_collision\_mod.f90** and the Subroutine is **UniversalCollisionOperator0DDGV.** The change is located **near line 3587** of the file. The corresponding fragment of the file is given below. You can notice that in this fragment we call two subroutines:  
**EvalCollisionPeriodicMixedTermsA\_DGV\_OMP** and **EvalFourierColTwoZeroPadF\_OMP**. Both subroutines do the same thing --- they evaluate the Boltzmann collision integral using nodal-DG binary convolution form. However, they do it differently. The first subroutine evaluates the convolution directly and the second uses the Fourier transform. You may also have noticed calls of Fortran built-in subroutine **CPU\_TIME().** This subroutine returns time on the system clock in seconds. These subroutines are often used to measure how long does it take to compute things. Specifically, by taking the time before evaluating collision integral and right after evaluating the collision integral we can tell how long did it take to evaluate the collision integral. One important detail is that both subroutines are parallelized using OpenMP parallelization. This means that work is shared between a number of processors when these subroutines are executed. For Bridges, the work is shared between 24 cores. A special feature of CPU\_TIME() is that time is measured for each of the 24 cores and combined. So, the number that is obtained by these measurements is a fraction of the actual time, called in this case “the all time”. The measured time is printed to the standard output. Standard output, by default, is the computer screen. We usually re-direct the standard output to a text file using the command “>” when the program is called into execution. Last, but not least, the mass, momentum, and energy moments of collision operator are computed and printed to standard output for both ways of evaluation. We will need these number for the second part of the Project 2.



**Part 4. Steps to change the number of mesh cells and to set up simulation to check the code complexity.**

In the following steps we will discuss how to modify the parameters of the code using the parameter files and to how to set up the simulations.

* Numerical experiments of Project 2 require simulation of the same numerical experiment, as a rule, requires modification of three files:
  + The parameter file DGVparameters.dat
  + The SLURM scheduler script shell file, e.g., run-slurm.sh
* Before each simulation we also need to clean the following directories from previous results (pay attention to capitalization of names):
  + - L1\_errs
    - moments
    - results
    - sol080909
* Directory sol080909 also contains the pre-computed collision kernels. The files for the kernels are located in subfolder OpA. Specifically, your copy of work folder has the following structure:  
    
    
   The folder 1xN should contain pre-computed collision kernel for different cases of number of cells and the order of DG method. DO NOT MODIFY THESE FILES OR CHANGE THEIR LOCATION

In this simulation, we will evaluate nodal-DG discretization of the Boltzmann collision operator using different number of velocity cells in the DG approximation. The number of cells is specified in the parameter file DGVparameters.dat. Please note that many of the parameters file are not used. Also, many parameters are used but you do not need to change them for this simulation. In most cases, you only want to change a few parameters. For numerical experiments in Project 2 you only would like to change the values that are in blue  
  
! This file contains parameter setting for DGV library

!

! DO NOT CHANGE THE PARAMETER NAMES! INTEGERS are limited to 6 digits!

!

!

! "," is a separator, IMPORTANT: ",4" will be read as two numbers, "0" and "4"

! "." is the decimal point

!

! parameter description (name) must not change including no adding extra spaces between words!

!

! the parameter description ends with "=" whish is followed by the parameter value

!

! the program will ignore all lines that start with "!" in the first column

!

!

!!!!!!!!!!!!!!!!!!!!!!!!

left endpoint in u = -3.0 !! u\_L

right endpoint in u = 3.0 !! u\_R

left endpoint in v = -3.0 !! v\_L

right endpoint in v = 3.0 !! v\_R

left endpoint in w = -3.0 !! w\_L

right endpoint in w = 3.0 !! w\_R

uniform mesh in u = yes !! mesh\_u\_uniform

uniform mesh in v = yes !! mesh\_v\_uniform

uniform mesh in w = yes !! mesh\_w\_uniform

use secondary velocity grid = no !! if this parameter is selected provide at least two numbers for each list below: the first number will determine the primary grid and the second number.

!! determines the secondary grid. The third (also fourth and fifth) numbers are optional

!! and is not used at the moment

!set the parameters of the primary and secondary grid

number of cells in u = 9,21,4 !! Mu\_list

degree of local Lagrange basis in u = 1,1,1 !! su\_list

number of cells in v = 9,21,4 !! Mv\_list

degree of local Lagrange basis in v = 1,1,1 !! sv\_list

number of cells in w = 9,21,4 !! Mw\_list

degree of local Lagrange basis in w = 1,1,1 !! sw\_list

! directory and the base name for the solution

!

current solution save directory = sol080909/ !! needs a comment, otherwise includes CR in the name

current solution base name = DGVcompl\_dcmp\_ !!

current A operator base name = OpA/1xN/exp0912\_ !! chunk reader will add ch000\_ to the name (max 20 symbols)

secondary operator A base name = OpA/1xN/exp0912\_ !!

!current A operator base name = Posops/PosOp\_ !!

!secondary operator A base name = PosOps/PosOp\_ !!

num of chunks for Aarray = 1 !! numchnks --- the total number of files in which the Aarray is stored,

num of chunks for sec Aarray = 1 !! numchnks II -- the total number of files that contain the Aarray for the secondary mesh

! Parameters related to the integration of moments

gauss order for moments in u = 9 ! moments\_u\_gauss\_order

mesh refinement in u for moments = 1 ! moments\_refine\_u

! Parameters of Calculating the A-array:

cutoff radius of A-array = 3.00 ! Trad -- velocities xi1 and xi that farther apart than Arad are neglected

error in integral in Chi = 1.0E-4 ! ErrChi -- error in the outside double integral

error in integral in Epsilon = 1.0E-5 ! ErrEps -- error in the inside double integral

cutoff values for A-array = 1.0E-11 ! min\_sens -- the values of A-array below thi slevel are not accepted

list of basis functions for A-array = 365,14896 ! I1\_list -- list of basis functions numbers for which A-array will be calculated!

range of node numbers I2 = 1,729 ! I2\_from and I2\_to - the range of the indices in I2. Values ignored if I1\_list has more then 2 records

! OpenMP parameters: (can ignore if not using OpenMP)

number of OMP threads = 28 ! Num\_OMP\_threads

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! Model threshholds: The order in which the models are envoked are

! 1) the Boltzmann equation, non-split formulation

! 2) the Boltzmann equation, split f=f\_M + df, formulation

! 3) the linearized Boltzmann equation (unless solving spatially homogeneous problem, does not give an advantage)

! 4) the BGK-type Model with velocity dependent collision frequency

! 5) the ES-BGK/BGK or Shakhov model

!

! the parameters below define the thresholds for switching between models. They represent the maximum allowed values of the L1-norm of the

! deviation of the solution from the local maxwellian. When the L1-norm of the deviation drops below the treshhold, the appropriate model is envoked.

! NOTICE: keep linear\_lev at least an order of magnitude less than decomp\_lev

!

! VERY IMPORTANT: the tresholds for the models above should be given in decreasing order.

! Otherwise the algorithms will make a mistake.

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

error maxwell ESBGK = 0.000000001 ! ES\_lev

error maxwell Vel ESBGK = 0.00001 ! vel\_lev -- velocity dependent regime

error maxwell linearization = .0002 ! linear\_lev,

error maxwell decomposition = 1.7 ! decomp\_lev

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! Parameters for dimensionless reduction

!

! C\_inf = thermal velocity in m/s

! L\_inf = characteristic length in m

! N\_inf = the total number of molecules in the volume

! T\_inf = the normalization for time is selected from the condition T\_inf\*C\_inf = L\_inf calculated automatically

! Temp\_inf = the characteristic temperature

! mol\_diam = molecular diameter in hard shperes model

!!!!!!!!!!!!!!!

ref termal velocity = 1290.370883961734 ! C\_inf m/s

ref characteristic length = 1.0 ! L\_inf m

ref number of molecules = 1.0E+20 ! N\_inf

ref molecular diameter = 3.76E-10 ! mod\_diam m-

!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!

! Parameters of the gas and ES-BGK

the ordinary gas constant = 208.1322 ! gasR (=208.1322\_DP for Ar; =296.95 for N\_2)

gas alpha = .5 !! gas\_alpha (when compare to Boltzmann hard spheres use .5)

gas viscosity = 5.337379160633744E-005 !! gas\_viscosity

Temperature reference = 2000 !! gas\_T\_reference

ESBGK modifier = -0.5 !! alpha for prandtl number, -0.5 corresponds to 2/3 , 0 to BGK if ES model is used.

!!!!!!!!!!!!!!!!!!!!!!!

!! Parameters VD-BGK (BGK model with velocity dependent collision frequency

N of enforced moments = 6 !! Order -- the variable determining how many or the built-in moments are enforced in relaxation

! !! order=1 invokes the first enforced moment. Current allowed Maximum is 15 moments total

N of coefficients in VDCF = 6 !! Order\_nu -- the number of coefficients/basis functions in the representation of the velocity-dependent collision

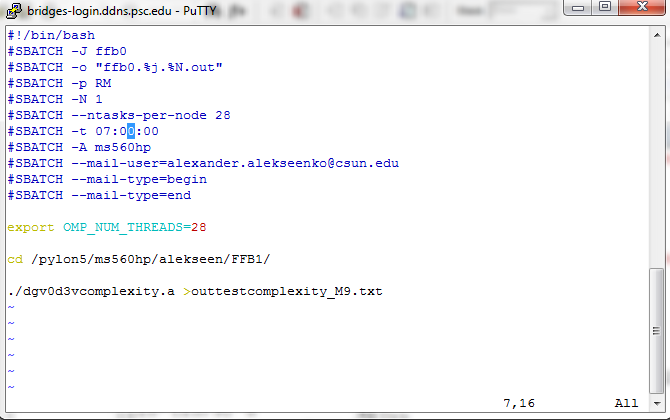
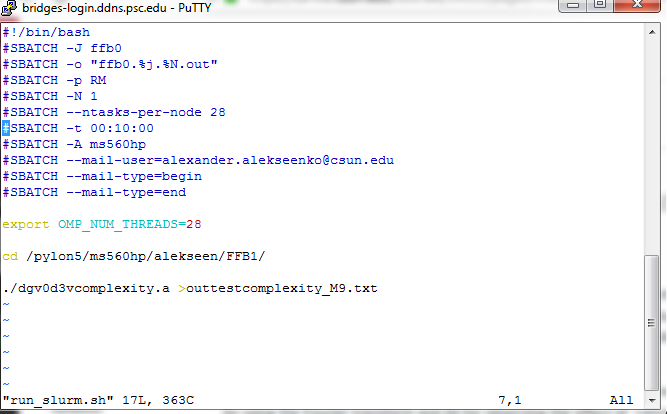
! !! frequency ATTENTION: Currently only the same number if allowed for both Order and Order\_nu

mean free time rates update = 0.5 !! mft\_coeff -- time period in mean free times between eadh

! end of file

* Let us explain certain rules about the parameter files. The parameter files consist of a bunch of commented lines and the parameter lines.   
  The commented lines start with “!” in the first position. The parameter lines start with a key phrase. For example, in the line   
    
  number of cells in u = 9,21,4 !! Mu\_list  
    
  the part “number of cells in u” is the key phrase. It is used in the code to identify the correct variable. Changing the key phrase by adding spaces, letters, or changing letters will result in code not being to interpret the phrase. After the phrase goes the “=” which alerts the code that the provided value of the parameter comes next. Different key phrases are associated with different allowed parameter values. For this particular key, a list of integer values can be provided. In the case above, the list consists of three values and they will be recorded into the array Mu\_list as suggested by the comment. The comment starts after the values indicated by an “!” symbol. (You can put “!”, “!!!!”, or “!!”) --- there has to be at least one “!” to declare the comment. In your case, only the first value in the list matters and the other values are ignored. However, please keep at least two values in the list.   
    
  Other key phrases may expect a real number as a parameter, or perhaps a string of symbols.
* Let us go over the changes in the DGVparameters.dat file, which are only in two groups of parameters:
  + number of cells in u = 9,21,4 !! Mu\_list
  + number of cells in v = 9,21,4 !! Mv\_list
  + number of cells in w = 9,21,4 !! Mw\_list

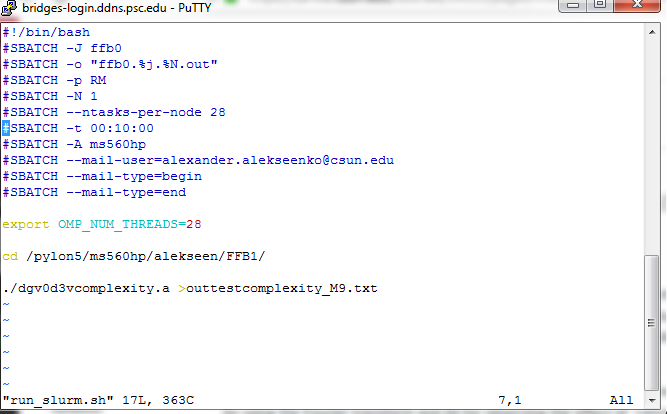
Change the first number to the desired number of cells in velocity. Please be aware that not all numbers can be used, because the code needs pre-computed collision kernels that are only available for a limited number of cases. Currently, cases of 9,15,21,27,33,41 are available.

* + current solution base name = DGVcompl\_dcmp\_ !!  
    provide a string of symbols to become the beginning part of the saved file names. You do not need to change it when you change number of cells--- this information is attached to the file name later. However if you do other changes, you may want to name the solution files differently to help you remember the changes.
  + Let us go over the changes in run\_slurm.sh file. Suppose that you have completed a successful simulation   
    We use vim editor to open the run\_slurm.sh file that was use in the previous run of the software:  
      
    The remaining parameter to change is the total requested time. The time is allocated by SLURM scheduler. Lines that contain SLURM parameters start with #SBATCH followed by an option key. The total time option is “-t HH:MM:SS”. So we are looking for the line   
      
    #SBATCH -t 07:00:00   
      
    this parameter asks to run the code for max of 7 hours. After 7 hours, if the code is still running, it will be terminated. You will receive an e-mail that you code timed out. If the code completes its normal execution within 7 hours, it will be considered as a successful execution. It seems that 7 hours is way too much time for our needs. It is best to estimate the necessary time more accurate. We will try to request 10 minutes instead. The new line will read   
      
    #SBATCH -t 00:10:00   
      
      
    We now save the file and exit. We are ready to submit the code into execution.

**Part 5. Steps to submit a dgv0d3vcomplexity.a run into execution**

* All production runs are submitted to execution through the SLRUM scheduler. To submit the job, use command  
    
  **sbatch run-slurm.sh**
* To check the status of the submitted job, use the command   
    
  **squeue –u username**
* To delete a submitted job, including terminating execution of a submitted job use command:  
    
  **scancel #job**  
    
  The number of the job is determined by running **squeue –u username** command.

**Part 6. Steps to perform numerical simulations to measure the code complexity.**

* Repeat the steps in Part 4 and Part 5 for M=15,21,27,33,41.
  + You will need to make changes to DGVparameters.dat, specifically parameters:   
    number of cells in u = 9,21,4 !! Mu\_list  
    number of cells in v = 9,21,4 !! Mv\_list  
    number of cells in w = 9,21,4 !! Mw\_list  
    need to be adjusted. In the case M=15 you will have   
    number of cells in u = 15,21,4 !! Mu\_list  
    number of cells in v = 15,21,4 !! Mv\_list  
    number of cells in w = 15,21,4 !! Mw\_list  
    In the case M=21 you will have   
    number of cells in u = 21,21,4 !! Mu\_list  
    number of cells in v = 21,21,4 !! Mv\_list  
    number of cells in w = 21,21,4 !! Mw\_list  
    and so on. Please note that only first number needs to be changes in all three lines.
  + Two changes will need to be done to run\_slurm.sh. It is expected that M=15 simulations will take a longer time than M=9 simulations. Then the time Let us use vim editor to open the run\_slurm.sh file for editing.   
    To set up the new run we need to change two parameters.
  + The first parameter to change is the total requested time to run the software. The time is allocated by SLURM scheduler. Lines that contain SLURM parameters start with #SBATCH followed by an option key. The total time option is “-t HH:MM:SS”. So we are looking for the line   
      
    #SBATCH -t 00:10:00   
      
    this parameter asks to run the code for max of 10 minutes. After 10 minutes, if the code is still running, it will be terminated. You will receive an e-mail that you code timed out. If the code completes its normal execution within 10 minutes, it will be considered as a successful execution. It is estimated that the time to evaluate the collision operator will increase as we increase the number of velocity cells M. So, code with M=15 will run longer than code with M=9. It is therefore important to adjust the parameter –t to reflect that change. Luckily, we have some theoretical predictions that help us estimate the time.   
      
    In the past, we predicted that the number of arithmetic operations to evaluate the convolution form of the collision operator directly depends on the number of velocity cells as O(M8). This will be, in fact, the most time consuming operation in the code. Thus, we can use this law to estimate the time.   
      
    For example, if we know that period of time of t9 seconds was sufficient to compute the M=9 case, then for M=15, we will estimate   
      
    t15=t9(15/9)8  
      
    so, for if 10 min was necessary for M=9, then for M=15 we will need 10\*(15/9)^8 ≅ 595 minutes which is about 6 hours.   
      
    It turns out that the 10 minutes was not an accurate estimate. It actually takes less than 10 seconds to compute the collision operator for M=9, so more accurate estimate will be 11 minutes for M=15. So we can have   
      
    -t 00:11:00   
      
    for M=15, but this will most likely have to change for M=21, M=27, 33, and 41. Always record the actual time the simulation ran. This will allow you to estimate the time of the next simulation more accurately.
  + The second parameter we will need to change is the name of the file where the output is saved. This parameters is in the last line  
      
    ./dgv0d3vcomplexity.a >outtestcomplexity\_M9.txt   
      
    Note the symbol “>”, which is actually an option to the system to re-direct the standard output, which is the terminal window, to a text file. Thus in the last simulation, all terminal output was redirected to the file “outtestcomplexity\_M9.txt”. You will need to save this file since it contains important information about the run. For the new simulation, we need a new file. The “\_M9” suffix in the previous file name was there to designate M=9 case. Next we will run M=15 case, so   
      
    ./dgv0d3vcomplexity.a >outtestcomplexity\_M15.txt  
      
    will be quite appropriate.
  + After the adjustment, the run\_slurm.sh file looks like this  
      
    We save the file and exit from vim. All is ready to submit the next case of M for run.

**Part 7. Steps to organize results of simulations on a local computer.**  
General rules:

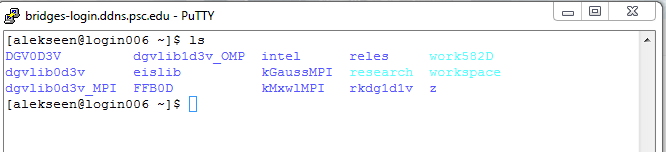
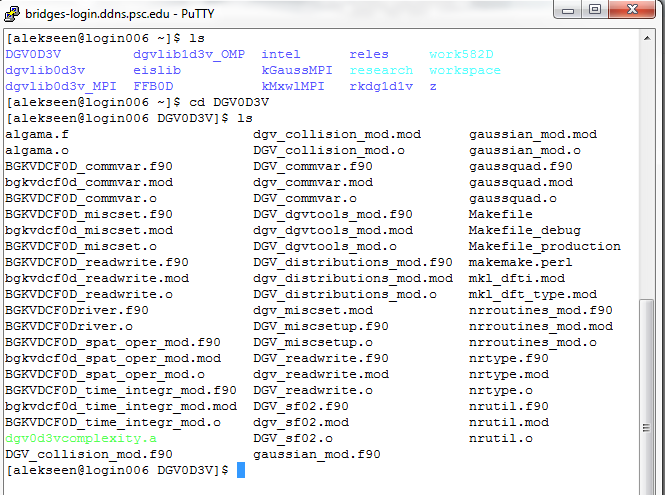
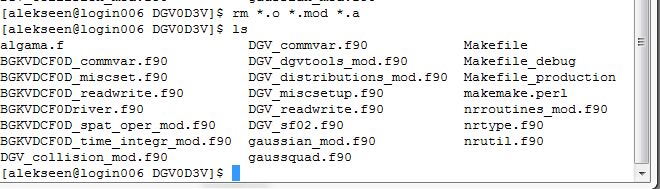
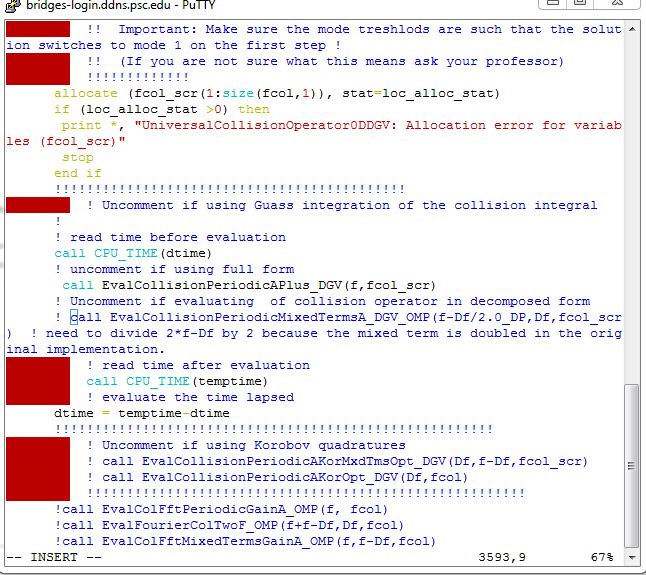
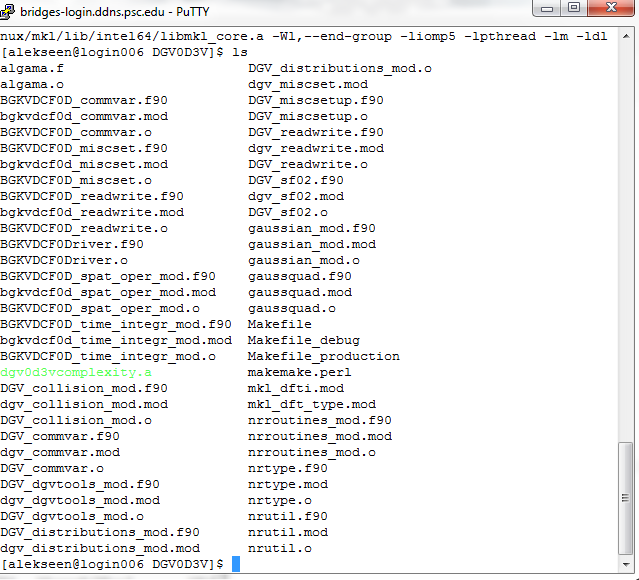
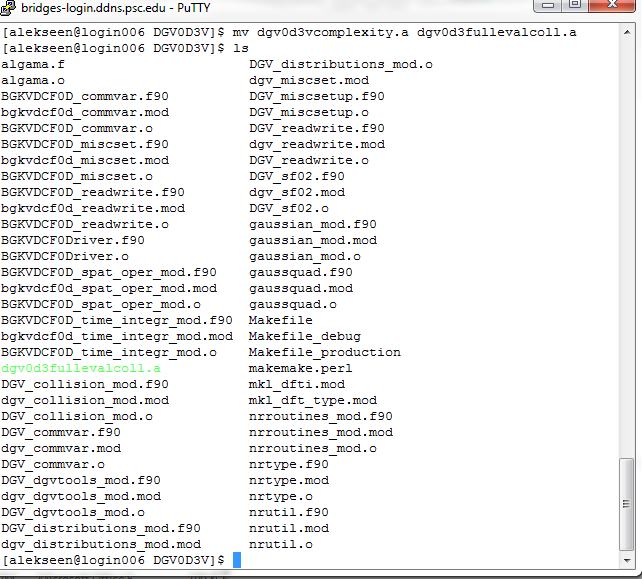
* Create a directory that will contain all simulations that you will perform using dgvlib0d3v code. Copy the matlab files that contain visualization subroutines into this directory. Create one or more subdirectories to organize simulation results.
  + Ideally, each group of simulations goes into separate folders.
  + Please include all output files produced by the dgvlib0d3v software and
  + Please include the runtime output file \*.txt
  + Prepare a copy of the \*.f90 files, Makefile or similar that were substantially modified to perform the numerical simulation
  + You may consider adding a Readme file that gives any additional detail about the simulations in the folder

Specific rules

* Create a folder for the numerical experiment to estimate the complexity. E.g. create folder “TestCompexity\_decomposed”
* In this folder, create a subfolder for each simulation, e.g., “M9”, “M15”, etc. So the path to keep the M=9 results will be “TestCompexity\_decomposed/M9”
* In the subfolder save the following files:
  + DGVparameters.dat
  + Parametes.dat
  + run\_slurm.sh
  + outtestcomplexity\_M9.txt (the output file)

**Part 8. Steps to modify the code to perform evaluation of the collision operator in non-decomposed form.**

The code provided by the Instructor calls subroutines that evaluate the collision operator in decomposed form, when solution is presented as a sum of local Maxwellian distribution and an additional term. It turns out that this evaluation of the collision operator is more accurate than the full form evaluation, when the solution is not decomposed. To force the code to perform the non-decomposed evaluation, different subroutine calls need to be made. The following steps explain how to make the change in the subroutine calls.

* Log in into Bridges, and navigate to the directory that contains the code used to measure the complexity.   
    
    
  If you used names suggested in the Part 1 of these steps, then the right directory should be **DGV0D3V** . Switch to this directory.  
    
  
* Clear the directory from \*.mod, \*.o files and also delete the executable  
    
  **rm \*.mod \*.o \*.a**  
    
  
* You will need to make changes to DGV\_cooolision\_mod.f90. Open this file in the **vim** editor.   
  Navigate to line number 3591. To navigate you can use arrows, which is very slow. Instead, *while in command mode*, you can type number of the line followed by Shift+G  
    
  **3591 Shift+G**  
    
  and you will see something like this  
  
* To force the code to use the full form of direct evaluation of collision integral, we need to
  + First, uncomment the subroutine call in line 3591 by removing the symbol “!” in front of it.
  + Second, comment the subroutine call in line 3593 by adding the symbol “!” in front of it.
  + After you finished, lines 3591 and 3593 will look like this:  
      
    
  + Then we move a little down in the code to line 3613, where evaluation of the collision operator using the Fourier transform is done. :  
    
  + We uncomment the subroutine call in line 3613 by removing a symbol “!” from it.
  + We comment the subroutine call in line 3615 by adding a symbol “!” in front of it.
  + The result will look like this:   
    
* We are done modifying the code. We now switch to the command mode, save the file and exit (Esc, followed by :wq).  
  
* Next we use command make to create the new executable. After compiler will compile the code successfully, you will see something like this:   
    
    
  If the compile responds with an error, you may have modified something inadvertently. The easiest solution is to get a clean copy of the file in question (the compiler will tell you what file generates an error) and follow the steps again to make the change.
* Last, we will re-name the executable so that we do not mix it with the executable used to measure complexity.   
    
  mv dgv0d3vcomplexity.a dgv0d3fullevalcoll.a  
    
  
* We are done preparing the new executable.

**Part 9. Steps to perform testing of evaluation of the collision operator in non-decomposed form.**

**Repeat steps in parts 2 – 6 to compute collision integral using non-decomposed form of the solution. Perform evaluations for cases M=9,15,21,27,33,41. Copy the results to your simulation using guidelines in Part 7.**