A Survey of Ensemble Filtering in the Data Assimilation Research Testbed

(Note: In the following, input that students need to enter are in bold, underlined text. Text that will be output to a workstation window is indicated by italicized text.)

I. Overview of DART

This section will give a brief high-level overview of DART. Section II will give a hands-on recapitulation of much of this material preparing students to use the system.

The Data Assimilation Research Testbed (DART) is designed to facilitate the combination of assimilation algorithms, models, and observation sets to allow increased understanding of all three. (see www.cgd.ucar.edu/DART/obs_set_specification.pdf for a more in-depth design discussion). For the ASP colloquium, a subset of the complete DART facility will be used to examine ensemble filter assimilation algorithms using synthetic observations. Synthetic observations are generated from a 'perfect' model integration, which is often referred to as the 'truth' or a 'nature run'. A model is integrated forward from some set of initial conditions and observations are generated as

$$y = H(x) + \varepsilon$$

where H is an operator on the model state vector, x, that gives the expected value of a set of observations, y, and ε is a random variable with a distribution describing the error characteristics of the observing instrument(s) being simulated. Using synthetic observations in this way allows students to learn about assimilation algorithms while being isolated from the additional (extreme) complexity associated with model error and unknown observational error characteristics. In other words, for the real world assimilation problem, the model has (often substantial) differences from what happens in the real system and the observational error distribution may be very complicated and is certainly not well known. Students should be careful to keep these issues in mind while exploring the capabilities of the ensemble filters with synthetic observations.

A. Working with DART

For the ASP workshop, DART programs must be compiled and run on the compute server 'ocotillo' while analysis is mostly performed on the DEC workstations in the Layton computer lab. Unfortunately, the MMM compute environment does not currently have a disk system that is cross-mounted between both the slave computing nodes on 'ocotillo' and the DEC systems. This means that students will be producing DART output on slave nodes on 'ocotillo', but will have to explicitly transfer these files via 'ftp' to the DEC machines for analysis. This is less than ideal and students are forewarned to remember to transfer files before doing each analysis step. A convenient method of operation is to open three windows on the DEC workstation: one open to the directory on the DEC system in which analysis is being performed, one open to the 'ocotillo' slave node on which computation is being performed, and one an ftp window with a pre-established connection from the DEC analysis directory to the 'ocotillo' working directory. A single execution of the command

mget *.nc

in this ftp window will move the latest analysis files from the compute to the analysis platforms. See the exercise introduction (http://www.cgd.ucar.edu/DART/exercise_overview.pdf) and the

slides at the end of the introductory presentation (http://www.cgd.ucar.edu/DART/asp_summer_talk.pdf) for more on the computing environment.

DART is designed to be a highly ‘modular’ code, meaning that different parts of the system may be readily modified, or replaced by pieces of code with similar interfaces but different internal actions. At the highest level, this means that complete models, observation set definitions, or assimilation algorithms can be swapped into the system to form fundamentally different assimilation systems. For instance, for the ASP workshop exercises a number of modules for completely different numerical models will be swapped into a complete assimilation system all using the same assimilation algorithm module. Many of the models will be able to use exactly the same modules to define observation sequences; however, some of the more complicated models will require that different observation modules be swapped into the system.

At a lower level, each part of the assimilation system (models, observations, and assimilation algorithms) are composed of a set of Fortran-90 modules. The system is designed so that any one of these modules can be easily replaced by another Fortran-90 module with similar interfaces but different functionality. For instance, in one of the low-order models that will be examined, there is a circumstance in which adding random noise to the model’s time derivative is potentially interesting. Students could write a new version of the model’s module, adding a random number to the time tendency by writing a completely new module and swapping this for the existing one.

B. Getting the DART code

The DART system is maintained using a simple software version control facility called CVS. As students in the ASP workshop are not anticipated to make major software updates to the DART software repository, CVS will not be used to control the software. Instead, students will be able to copy a complete directory tree containing all components of DART to their own working directory (/ocotillo2/mmm??/DART) on ‘ocotillo’ where ?? is your mmm account number. Students can then compile default versions of DART modules, make modifications to modules, or create their own versions of modules that can be used in DART assimilation systems.

Choosing the appropriate computer languages to create scientific software is a complex and evolving process. For a variety of reasons, the computational portions of DART have been written using the Fortran90 language. While Fortran continues to offer a convenient way to create efficient executables on a variety of high-end platforms, it is clearly no longer a mainstream language with users confined to narrow sectors of the scientific and engineering fields. Even more unusual are the applications of Fortran90 in DART where an attempt is made to mimic more modern ‘object-oriented’ languages. Students who have used Fortran before should find it relatively easy to understand the internals of most of the DART modules, however, those with no Fortran experience will probably not have enough time to learn sufficient Fortran during the two week colloquium. These students should be quick to ask for assistance from the workshop organizers if they need to understand the function of particular Fortran modules.

C. Compiling the DART code

DART executable programs are constructed using two dependency analysis tools, ‘make’ and ‘mkmf’. The ‘make’ utility is a relatively common piece of software used to record dependencies between different files and then to perform a hierarchy of actions when one or more of the files is modified. The ‘mkmf’ utility is a preprocessor that generates a ‘make’ input file (Makefile) and is designed specifically to work with object-oriented Fortran90 (and other languages) for systems like DART. As input, ‘mkmf’ requires two separate files. The first is a ‘mkmf template’ file which specifies details of the commands required for a specific Fortran90 compiler and may also contain pointers to directories containing pre-compiled utilities required by the DART system. The second ‘mkmf’ input is a ‘path_names’ file which includes a complete list of the locations (either relative or absolute) of all Fortran90 source files that are required to produce a particular DART program. Each path_names file must contain a path for exactly one Fortran90 file containing a main program, but may contain any number of additional paths pointing to files containing Fortran90 modules. An ‘mkmf’ command is executed which specifies the path_names file and the mkmf template file and the result is the creation of a Makefile which can be used as input for the standard make utility. Shell scripts that execute the mkmf command for all standard DART executables are provided as part of the standard DART software. For more information on mkmf see

(http://www.gfdl.noaa.gov/~lat/fms_public_release/public_manual_fms/
havana_public_manual.html#compiling%20the%20code).

Once mkmf is executed for a given program, the command

make

can be entered. The make program will survey existing compiled Fortran-90 modules and compile those which depend on modules that have been updated. The result is either a compilation error and premature termination or the creation of an executable program.Once an executable is created, it can be executed on one of the ‘ocotillo’ slave nodes.

WARNING: At present, ‘ocotillo’ is experiencing a persistent clock skew in which the slave nodes do not share a synchronized time with the master node. In some unfortunate cases, this can lead to problems with the make program in two ways. First, make can produce an error stating that clock skew has been detected. This may indicate that the executable program has not been properly linked. Second, make may fail to notice that a particular source file has been updated and changes to the code will fail to be incorporated. For this reason, we strongly recommend that the command

make clean

be executed before each use of make. While this reduces the efficiency of the make process, it avoids most of the potential problems associated with clock skew on ‘ocotillo’.

D. Running DART Programs

DART programs can require three different types of input. First, some of the DART programs, those for creating synthetic observational datasets, require interactive input from the keyboard. For simple cases, this interactive input can be made directly from the keyboard. In more complicated cases, a file containing the appropriate keyboard input can be created and this file can be directed to the standard input of the DART program. Second, many DART programs expect one or more input files in DART specific formats to be available. For instance, the program ‘perfect_model_obs’ that creates a synthetic observation set given a particular model and a

description of a sequence of observations requires an input file that describes this observation sequence. At present, all DART specific files are inefficient but human-readable ascii files. Third, many DART modules (including main programs) make use of the Fortan90 namelist facility to obtain values of certain parameters at run-time. All programs look for a namelist input file called 'input.nml' in the directory in which the program is executed. The input.nml file can contain a sequence of individual Fortran90 namelists which specify values of particular parameters for modules that compose the executable program. A complete list of individual namelists that are relevant to the ASP colloquium can be found in Appendix VI. Unfortunately, the Fortran90 namelist interface is poorly defined in the language standard, leaving considerable leeway to compiler developers in implementing the facility. The Intel 7.0 compiler being used to create DART programs on 'ocotillo' has some particularly unpleasant behavior when a namelist file contains an entry that is NOT defined in the program reading the namelist. Error behavior is unpredictable, but often results in read errors for other input files opened by DART programs. If you encounter run-time read errors, don't hesitate to consult one of the colloquium staff to make sure that the error is not a result of namelist inconsistencies.

As noted, each student team will acquire their own copy of the complete DART software system. When students want to make modifications to a particular DART module or other pieces of DART, they can use their favorite editor to modify their copy. Following this with mkmf, make clean, and make will create executables that include the code modifications. In some instances, students may wish to create entirely new Fortran90 modules that provide a revised functionality. The preferred method for doing this would be for students to create a new module with the same Fortran module name and public interfaces as the original, but stored in a different file. The path for this new file should then replace the old path in the path_names file for mkmf for all programs that use this module. It is essential that

make clean

be run when replacing a module in this fashion or the new module may not be incorporated in the newly created executable.

E. DART analysis files

DART uses the NetCDF (http://www.unidata.ucar.edu/packages/netcdf/) self-describing data format with a particular metadata convention to describe output that is used to analyze the results of assimilation experiments. These files have the extension .nc and can be read by a number of standard data analysis tools. Three sets of tools are available to work with NetCDF files for the ASP colloquium. First, the simple tool ncview is provided to do rudimentary graphical display of slices of output data fields. ncview (http://meteora.ucsd.edu/~pierce/ncview_home_page.html) will be of most use for output of the more comprehensive models at the end of the exercise set. Second, a set of tools called the NCO tools, produced by UCAR's Unidata group, are available to do operations like concatenating, slicing, and dicing of NetCDF files (http://nco.sourceforge.net). Finally, a set of Matlab input scripts (Appendix I) designed to produce graphical diagnostics from DART NetCDF output files are available. These are used with the proprietary Matlab software (http://www.mathworks.com/). In later portions of the exercises, students may wish to write their own matlab scripts to do additional types of analysis. Matlab includes an on-line help facility. Several matlab experts will also be available to consult during the scheduled exercise times for students who wish to do enhanced analysis.

II. A hands-on example: The Lorenz-63 model

This initial sequence of exercises includes detailed instructions on how to work with the DART code and allows students to investigate basic features of one of the most famous dynamical systems, the 3-variable Lorenz-63 model (Appendix II.1). The remarkable complexity of this simple model will also be used as a case study to introduce a number of features of a simple ensemble filter data assimilation system.

A. Getting a local copy of the DART system

Begin by logging on to your account on the MMM DEC/COMPAQ workstations. From there proceed to login to the 'ocotillo' computing cluster master node via the command

ssh ocotillo

Next, login to your assigned 'ocotillo' slave node by issuing the command:

ssh node?

where ? is 1,2, ..., 10. (See Table 1 at the end of this document for a list of account numbers and corresponding ocotillo compute nodes). Storage for your account will be available in the directory

/ocotillo2/mmm??

where ?? is the number of your assigned mmm temporary account. Change to your working directory:

cd /ocotillo2/mmm??

and copy the DART system directory tree

cp -rf /ocotillo2/jla/DART . (don't forget to type the . at the end).

You are now ready to compile a set of DART programs to do synthetic observation experiments in the L63 model. If at any time you need to retrieve the original version of any DART modules or utilities, they can be obtained from /ocotillo2/jla/DART.

B. Compiling the DART code

For the ASP colloquium, DART executables will be constructed in a work subdirectory under the directory containing code for the given model. In the DART directory you have just copied, change to the L63 work directory

cd DART/models/lorenz_63/work

Listing the contents of this directory should reveal the following files:

CVS
filter_ics
input.nml
mkmf_create_obs_sequence
mkmf_create_obs_set_def
mkmf_filter
mkmf_perfect_model_obs
path_names_create_obs_sequence
path_names_create_obs_set_def
path_names_filter
path_names_perfect_model_obs

perfect_ics
work.pcl

There are four mkmf files for the programs create_obs_set_def, create_obs_sequence, perfect_model_obs, and filter along with the corresponding path_names files. You can examine the contents of one of the path_names files, for instance path_names_filter, to see a list of the relative paths of all files that contain Fortran90 modules required for the program filter for the L63 model. All of these paths are relative to the DART directory that you copied into your local storage. The first path is the main program (filter.f90) and it is followed by all the Fortran90 modules used by this program. The corresponding mkmf shell script files are considerably more cryptic and their contents should not be relevant for the asp colloquium.

To create the first executable, execute the mkmf_create_obs_set_def file,

csh mkmf_create_obs_set_def

The result should be the creation of a file, Makefile, which contains input for the make utility (if you're familiar with make you could take a look). Executing the command:

make

should result in a series of Fortran90 modules being compiled and a final link step that produces the executable file, create_obs_set_def. The make utility will display a sequence of commands followed by a list of all functions and subroutines in each module. A series of .o and .mod files for each module compiled will also be left in the work directory.

You can proceed to create the other three programs needed to work with L63 in DART by doing mkmf and make in turn for each:

csh mkmf_create_obs_sequence
make clean; make
csh mkmf_perfect_model_obs
make clean; make
csh mkmf_filter
make clean; make

C. Running the DART code

The DART system uses files called 'observation sequence definition' and 'observation sequence' files to control the execution of an assimilation. An observation sequence definition file contains a complete description of a time series of observations but contains no observation values. Each observation in an observation sequence definition file has a time, a location and type, and an observational error characterization associated with it. An observation sequence file contains all the information in an observation sequence definition file plus an observed value for each observation. To perform a synthetic observation assimilation experiment for the L63 model, the following steps must be performed (an overview of the 5 steps is given first, followed by a recapitulation with detailed procedures for each step):

1. Integrate the L63 model for a long time starting from arbitrary initial conditions in order to generate a model state that lies on the attractor. The ergodic nature of the L63 system means a

long enough integration always converges to some point on the computer's finite precision representation of the model's attractor.

2. Generate a set of ensemble initial conditions from which to start an assimilation. Since L63 is ergodic, the ensemble members can be designed to look like random samples from the model's 'climatological distribution'. To generate an ensemble member, very small perturbations can be introduced to the state on the attractor generated by step 1. This perturbed state can then be integrated for a very long time until all memory of its initial condition can be viewed as forgotten. Any number of ensemble initial conditions can be generated by repeating this procedure.

3. An 'observation sequence definition' is created to simulate a particular observing system. This step is performed using the program create_obs_set_def which requires interactive input describing the observations.

4. An 'observation sequence' is created by integrating the model and using the information in the observation sequence definition file to create simulated observations as directed. This entails operating on the model state at the time of the observation definition with an appropriate forward operator (a function that operates on the model state vector to produce the expected value of the particular observation definition) and then adding on a random sample from the observation error distribution specified in the observation sequence definition. At the same time, diagnostic output about the 'true' state trajectory can be created.

5. The filter is run and the synthetic observations are assimilated; diagnostic output is generated.

These 5 steps can be performed using the DART software as outlined below. The process of creating the initial conditions for the truth integration and the filter are non-trivial and are described in the rather lengthy steps 1 and 2. The L63/work directory contains files perfect_ics and filter_ics which contain sample output from steps 1 and 2. While diagnosing the behavior of the spin-up in these two steps is intriguing, students may want to skip to step 3 and come back to steps 1 and 2 if time permits.

1. Integrating the L63 model for a long time. Begin by creating an observation sequence definition file that spans a long time. When this is available as input to the program perfect_model_obs, it will integrate the L63 model for the times spanned by the observation sequence definition. Creating an observation sequence definition is a two step procedure involving the program create_obs_set_def followed by the program create_obs_sequence. Program create_obs_set_def creates an observation set definition, the time-independent part of an observation sequence. An observation set definition file contains the location, type, and observational error characteristics (normally just the diagonal observational error variance) for a related set of observations. For step 1, all we are interested in is integrating the L63 model, not in generating any particular synthetic observations. Begin by creating a minimal observation set definition. Enter:

create_obs_set_def

This program prompts for interactive input of a filename in which to create the observation set definition:

Input the filename for output of observation set_def_list? [set_def.out]

A suggested default value, **set_def.out**, is provided so enter this name. Next, the program asks for the number of sets of observations you want to define. In general, for the exercises for the ASP colloquium, only a single set is defined, so enter **1** in response to the query:

Input the number of unique observation sets you might define

Next, the number of individual scalar observations (like a single surface pressure observation) in the set is needed:

How many observations in set

To spin-up an initial condition for the L63 model, enter a **1** to define a single observation. Next, the error variance for this observation must be entered. Since we are not interested in this observation having any impact on an assimilation (it will only be used for spinning up the model and the ensemble), enter a very large value for the error variance, like **10000000**, in response to the query:

Input error variance for this observation definition

An observation with a very large error variance has essentially no impact on deterministic filter assimilations like the default variety implemented in DART.

Finally, the location and type of the observation need to be defined. For all types of models, the most elementary form of synthetic observations are called 'identity' observations. These observations are generated simply by adding a random sample from a specified observational error distribution directly to the value of one of the state variables. In response to:

Input an integer index if this is identity observation, else -1

enter the number **1**. This defines the observation as being an identity observation of the first state variable in the L63 model, normally referred to as x. The program will respond by terminating after generating a file called set_def.out that defines the single identity observation of the x-variable.

Next, an observation sequence definition file needs to be created containing this single observation. The program create_obs_sequence takes an observation set definition file as input and asks for information about when this observation set is observed in order to generate an observation sequence definition file. Enter:

create_obs_sequence

and then enter:

set_def.out

to indicate the location of the observation set definition that you've just created. The option exists to create sequences in which the observation sets are observed at regular intervals or irregularly in time. Here, all we need is a sequence that takes observations over a long period of time. Enter a 1 to generate a regular sequence. Then enter 1000 in response to:

Input number of observations in sequence

Although the L63 system normally is defined as having a non-dimensional time step, the DART system somewhat arbitrarily defines the model timestep as being 3600 seconds. We want to go through several thousand time steps in order to spin-up the model onto the attractor, so enter

1, 0 (that's a comma between the 1 and 0, a blank works, too),

(this is 1 day and 0 seconds) in response to both questions:

Input time of initial ob in sequence in days and seconds

and

Input period of obs in days and seconds

This will create an observation sequence file that contains a single observation each day for 1000 days (24000 time steps) after the initial time which is assigned to the initial model state. Finally, input **obs_seq.in** as the name for the observation sequence definition file in response to:

Input file name for output of obs_sequence? [obs_seq.in]

Now, it's time to use the program perfect_model_obs to advance an arbitrary initial state for 24,000 time steps to move it onto the attractor. To do this, you need to make sure that the values in the Fortran-90 namelist file, input.nml, are set to appropriate values. In input.nml, the namelist for perfect_model_obs should have the following values set:

```
&perfect_model_obs_nml
  obs_seq_in_file_name = "obs_seq.in",
  obs_seq_out_file_name = "obs_seq.out",
  start_from_restart = .false.,
  output_restart = .true.,
  restart_in_file_name = "perfect_ics",
  restart_out_file_name = "perfect_restart",
  init_time_days = 0,
  init_time_seconds = 0,
  output_interval = 1
&end
```

The first two entries specify the file names for the input observation sequence definition file and the output observation sequence file generated by perfect_model_obs. Entry start_from_restart is set to false, telling perfect_model_obs to generate an arbitrary initial condition that is not known to be on the L63 attractor. Output_restart is set to true indicating that the model state at the end of this integration will be output to the file perfect_restart which is specified as the restart_out_file_name. Executing perfect_model_obs will integrate the model 24,000 steps and output the resulting state in the file perfect_restart.

2. Generating ensemble initial conditions: The file perfect_restart can be now be copied to perfect_ics

cp perfect_restart perfect_ics

and the namelist for perfect_model_obs can be modified to set start_from_restart = .true. A subsequent integration of perfect_model_obs should now be performed which will advance the model state from the end of the first 24,000 steps to the end of an additional 24,000 steps and place the final state in the file perfect_restart.

The namelist for program filter should now be set to:

```
&filter_nml
  ens_size = 20,
  cutoff = 0.0
  cov_inflate = 1.0,
  start_from_restart = .false.,
  output_restart = .true.
  obs_sequence_file_name = "obs_seq.out",
  restart_in_file_name = "perfect_ics",
```

```
   restart_out_file_name = "filter_restart",
   init_time_days = 0,
   init_time_seconds = 0,
   output_state_ens_mean = .true.
   output_state_ens_spread = .true.
   num_output_ens_members = 20
   output_interval = 1
&end
```

The ensemble size is set to 20 (used in most of the exercise sets) and the cut_off parameter is set to 0 which limits the impact of any observations (this is discussed more below in Section III) since in this ensemble spin-up run no impact from observations is desired. The filter is told to generate its own ensemble initial conditions since start_from_restart is false. However, it is important to note that the filter still makes use of the file perfect_ics which is set to be the restart_in_file_name. This is the model state generated from the first 24,000 step model integration by perfect_model_obs and the filter program generates its ensemble initial conditions by randomly perturbing the state variables of this state. The arguments output_state_ens_mean and output_state_ens_spread are true so that these quantities are output at every time for which there are observations (once a day here) and num_output_ens_members means that the same diagnostic files, Posterior_Diag.nc and Prior_Diag.nc also contain values for all 20 ensemble members once a day. Once the namelist is set, execute program filter to integrate the ensemble forward for 24,000 steps with the final ensemble state written to the file filter_restart.

The spin-up of the ensemble can be viewed by examining the output in the netcdf files True_State.nc generated by perfect_model_obs and Posterior_Diag.nc and Prior_Diag.nc generated by program filter. To do this, see the detailed discussion of matlab diagnostics in Appendix I. Try to understand how the ensemble errors grow. Taking a look at the L63 attractor time evolution as displayed by the built-in matlab command **lorenz** may help, too. This will also give you a chance to learn to use the interface to matlab's three-dimensional visualization packages.

Copy the perfect_model_obs and filter restart files, perfect_restart (the 'true state') and filter_restart to perfect_ics and filter_ics so that assimilation experiments can be initialized from these spun-up states:

cp perfect_restart perfect_ics; cp filter_restart filter_ics

Two entries in the namelist for program filter should also be modified. Change start_from_restart to true (you've just generated an ensemble restart file for filter) and set restart_in_file_name to "filter_ics". This process has now generated files perfect_ics and filter_ics that are qualitatively equivalent to those residing in the lorenz_63/work directory in /ocotillo2/jla/DART.

3. (Start here if you are using the pre-existing perfect_ics and filter_ics from the lorenz_63/work directory). Now, an observation sequence definition for doing assimilation tests can be generated. Begin by using create_obs_set_def to generate an observation set in which each of the 3 state variables of L63 is observed with an observational error variance of 1.0 for each observation. To do this, run create_obs_set_def using the following input sequence (the text including and after # is a comment and does not need to be entered):

set_def.out # Output file name

1 # Number of sets
3 # Number of observations in set (x, y, and z)
1.0 # Variance of first observation
1 # First ob is identity observation of state variable 1 (x)
1.0 # Variance of second observation
2 # Second is identity observation of state variable 2 (y)
1.0 # Variance of third ob
3 # Identity ob of third state variable (z)

Now, generate an observation sequence definition by running **create_obs_sequence** with the following input sequence:

set_def.out # Input observation set definition file
1 # Regular spaced observation interval in time
1000 # 1000 observation times
0, 43200 # First observation after 12 hours (0 days, 3600 * 12 seconds)
0, 43200 # Observations every 12 hours
obs_seq.in # Output file for observation sequence definition

4. An observation sequence file is now generated by running perfect_model_obs with the namelist values (these are the same as those in the last phases of step 1):

```
&perfect_model_obs_nml
  obs_seq_in_file_name = “obs_seq.in”,
  obs_seq_out_file_name = “obs_seq.out”,
  start_from_restart = .true.,
  output_restart = .true.,
  restart_in_file_name = “perfect_ics”,
  restart_out_file_name = “perfect_restart”,
  init_time_days = 0,
  init_time_seconds = 0,
  output_interval = 1
&end
```

This integrates the model starting from the state in perfect_ics for 1000 12-hour intervals outputting synthetic observations of the three state variables every 12 hours and producing a NetCDF diagnostic file in True_State.nc.

5. Finally, the filter can be run with the namelist set to:

```
&filter_nml
  ens_size = 20,
  cutoff = 22222222.0
  cov_inflate = 1.0,
  start_from_restart = .true.,                  # This value is changed from step 2
  output_restart = .true.
  obs_sequence_file_name = “obs_seq.out”,
  restart_in_file_name = “filter_ics”,              # Value changed from step 2
  restart_out_file_name = “filter_restart”,
  init_time_days = 0,
  init_time_seconds = 0,
```

```
   output_state_ens_mean = .true.
   output_state_ens_spread = .true.
   num_output_ens_members = 20
   output_interval = 1
&end
```

The large value for the cutoff allows each observation to impact all other state variables (see Appendix V for localization). Program filter produces two output diagnostic files, Prior_Diag.nc which contains values of the ensemble members, ensemble mean and ensemble spread for 12-hour lead forecasts before assimilation is applied and Posterior_Diag.nc which contains similar data for after the assimilation is applied (sometimes referred to as analysis values).

Now try applying all of the matlab diagnostic functions described in Appendix I. Try to understand what the different tools can tell you about the performance of the assimilation and about the attractor of the L63 model. Try looking at the joint behavior of the control integration and the assimilation (both the ensemble mean and individual ensemble members) in phase space, too. You can learn a lot about how assimilations fail even in this simple system.

III. Bias, filter divergence and covariance inflation (with theL63 model)

One of the common problems with ensemble filters is filter divergence, which can also be an issue with a variety of other flavors of filters including the classical Kalman filter. In filter divergence, the prior estimate of the model state becomes too confident, either by chance or because of errors in the forecast model, the observational error characteristics, or approximations in the filter itself. If the filter is inappropriately confident that its prior estimate is correct, it will then tend to give less weight to observations then they should be given. The result can be enhanced overconfidence in the model's state estimate. In severe cases, this can spiral out of control and the ensemble can wander entirely away from the truth, confident that it is correct in its estimate. In less severe cases, the ensemble estimates may not diverge entirely from the truth but may still be too confident in their estimate. The result is that the truth ends up being farther away from the filter estimates than the spread of the filter ensemble would estimate. This type of behavior is commonly detected using rank histograms (also known as Talagrand diagrams, etc., Appendix IV). You can see the rank histograms for the L63 initial assimilation by using the matlab script plot_bins.

A simple, but surprisingly effective way of dealing with filter divergence is known as covariance inflation. In this method, the prior ensemble estimate of the state is expanded around its mean by a constant factor, effectively increasing the prior estimate of uncertainty while leaving the prior mean estimate unchanged. The program filter has a namelist parameter that controls the application of covariance inflation, cov_inflate. Up to this point, cov_inflate has been set to 1.0 indicating that the prior ensemble is left unchanged. Increasing cov_inflate to values greater than 1.0 inflates the ensemble before assimilating observations at each time they are available

$$x_i^{inflated} = \gamma(x_i - \bar{x}) + \bar{x}$$

where γ is the covariance inflation factor, x_i is the ith prior ensemble member for a state variable, x, and $\bar{x}$ is the ensemble mean of all members for x. Values smaller than 1.0 would actually contract (reduce the spread) of prior ensembles before assimilating.

Examine the impacts of varying the value of covariance inflation for the L63 assimilation case above. You can do this by modifying the value of cov_inflate in the namelist, (try 1.05 and 1.10 and other values at your discretion) and run the filter as above. In each case, use the diagnostic matlab tools to examine the resulting changes to the error, the ensemble spread (via rank histogram bins, too), etc. What kind of relation between spread and error is seen in this model (see Appendix IV)?

IV. Impacts of observation quality and frequency (L63)

Assimilation algorithms can be viewed as extracting information from observations using models to provide estimates of time evolution of the system state. The amount of information available from observations can be viewed as increasing when the observational error variance is reduced (in the case of gaussian error distributions that are used in the current DART implementation) or when the frequency of observations is increased. To observe this effect, try a sequence of values for the observational error variance in the L63 experiments performed above. Since the assimilated distributions are already very tight, it is most illuminating to begin by reducing the specified error variance, for instance, try 2.0, 4.0, 8.0, etc. To do this, you will need to generate new observation sequences as outlined in Section II.3 above. As you continue to increase the error variance, the observations should have less and less impact on the assimilation. Eventually, for large enough error variance the ensemble distribution should be indistinguishable from the climatology of the L63 model. Be sure to assess not only the error but also the ensemble spread, consistency (bins), etc.

The impact of observation frequency with fixed observational error variance can be examined in the same way. Try increasing the period of observations to 1 day, 2 days, etc. To do this, you can generate an observation set definition file and then run program create_obs_sequence with different observation frequency specified to generate different observation sequences. Programs perfect_model_obs and filter can be run and the output analyzed in each case. As observations become less frequent, the prior distributions before the next observation should become less and less certain until they asymptote to the climatological distribution. Again, carefully analyze the behavior of the assimilation with the tools detailed in Appendix I. It is particularly interesting to try to understand how the attractor structure comes into play in influencing these distributions.

The amount of information available from observations can also be reduced in this L63 case if not all of the components of the state vector are observed at every observation time. Examine what happens if only state variable 1 (x), state variable 2 (y), or state variable 3 (z) are observed. Using an observation frequency of 12 hours (43200 seconds) and an observation error variance of 1.0 is a good place to start these experiments. Are some of the state variables more informative? Can you relate this behavior to the attractor structure?

A final exploration of limiting information content can be taken by observing all three state variables, but only letting an observation impact the corresponding state variable. In other words, observations of state variable 1 (x) are only allowed to change state variable 1, etc. In the L63 model, DART assigns the three state variables to equally spaced locations on a one-dimensional

unit length periodic domain. Variable 1 (x) is at location 0.0, variable 2 at 1/3, and variable 3 at 2/3. There is no notion of physical location in this sense in the underlying dynamical system, but this allows DART features that are designed for models with the notion of a physical location for state variables to be used with L63. The namelist parameter 'cutoff' controls the weight with which observations at one physical location impact state variables at other physical locations. This coefficient gives the half-width of a gaussian like envelope that multiplies the impact of an observation on a remote state variable (you can see the specifics of the localization in the code at DART/cov_cutoff/cov_cutoff_mod.f90). So far, we have been using a very large value for cutoff, essentially allowing all observations to impact all state variables with full weight. If cutoff is reduced to a very small value, say cutoff = 0.00001, observations will only impact state variables at the same location (the impact at remote state variables will be multiplied by a value very close to 0). Try changing 'cutoff' in the name list to this value and repeating the experiment in which all 3 state variables are observed every 12 hours with observational error variance 1.0 (or try error variance 4.0).

A final experiment with the L63 model examines the use of non-identity observation operators. In this case, the DART system allows observations which are linear combinations of two of the three state variables to be constructed by again viewing the three state variables as being equally spaced on a periodic unit domain. So far, we have only defined identity observations with create_obs_set_def, but it can also define observations that are linear interpolations to arbitrary locations on this unit domain. For instance, defining an observation at location 0.11111 will generate a forward operator that is 2/3 x + 1/3 y since x is at position 0.0 and y at position 1/3. Explore what happens when a single observation that is located 'equidistant' between two of the state variables is defined. For instance, create an observation set definition with create_obs_set_def proceeding as when you designed the set with a single observation of a state variable. When create_obs_set_def prompts:

Input an integer index if this is identity observation, else -1

Input a **-1**. Following this, the program will output:

Input location for this obs: value 0 to 1 or a negative number for
Uniformly distributed random location

Input the value **0.1666666**, halfway between the locations of x and y. Finally, enter **1** in response to the query

input obs kind: u = 1, v = 2, ps = 3, t = 4

since the current DART version does not distinguish observation types for the L63 model. Next, proceed to generate an observation sequence with create_obs_sequence and then try the assimilation. You might also want to take a look at other combinations comparing error, etc.

V. The Lorenz 9-variable model (see Appendix II for model details)

This model a highly truncated primitive equation model. DART experimentation on the 9var model can be done in the directory DART/models/9var/work. Here are found the same types of files as for the L63 model (see section II above) including input files for mkmf and initial condition files for the truth control run and 20-member ensemble assimilations in the files perfect_ics and ensemble_ics.

As for the L63 model, a somewhat arbitrary assignment of 3600 seconds to the non-dimensional timestep prescribed in the 9var model has been made. Also, although again there is no notion of a ‘physical location’ for the 9 state variables, they have been arbitrarily assigned to have equally spaced locations on a periodic [0, 1] domain with variable 1 (x_1) at 0.0 and variable 9 (z_3) at location 8/9.

The initial condition files were generated in a fashion similar to those for the L63 model. First, the true state initial condition was created by integrating the model for 10,000 1 day intervals from an arbitrary initial condition (the slow dynamics of the 9var model has a considerably longer spin-up period than for the L63 model with the arbitrary assignment of 3600 seconds to the underlying model timestep). Next, a 1000-member ensemble is generated by adding very small perturbations to this spun-up state and then integrating each ensemble member for 10,000 1 day intervals; the final states are in filter_ics. The truth is also integrated for this second 10,000 day interval and the final state is in perfect_ics.

An interesting assimilation problem for the 9-variable model can be constructed using identity observations (there will be 9 observations in the set) with an observational error variance of 1.0 for each observation, and observations taken every hour (3600 seconds). Problems with this assimilation are particularly apparent in the divergence state variables (1 through 3). Clearly, some of the prior ensemble estimates are highly inconsistent with the truth (use the standard matlab tools you used with the L63 model to see the time series, bins, etc.).Try introducing even a small amount of covariance inflation to try to deal with this bias problem (say cov_inflate = 1.01 in the namelist). Explore what happens to the error, spread, and bins. Try even larger values of cov_inflate like 1.05; it is also interesting to try some covariance deflation, with cov_inflate=0.99.

Another interesting sequence of experiments in the 9var model involves only observing one type of state variable, say only the vorticity, height, or divergence variables. Try setting up a sequence of 3 experiments in which only these components are observed (in each case the observation set will include 3 observations, the 3 wave components of height for instance). Set the observational error variance to 1.0 for the height and vorticity fields but try something smaller, say 0.1 for the divergence which has a much smaller ‘climatological range’. Examine the performance of the filter in all cases. It is especially interesting to see if one can get something to work when only observing the divergence fields; why?

A number of additional optional experiments can be performed with the 9var model. Try varying the observation frequency or the observational error variance and observing the relative performance of the assimilation. Try to find cases for which the high frequency ‘gravity waves’ that often plaque the assimilation are the worst. Consider some ways to try to deal with these ‘gravity waves’ which can seriously degrade an assimilation. One interesting experiment requires making a modification to the 9var model code which can be found in the directory DART/models/9var/model_mod.f90. The subroutine comp_dt has a number of commented lines that allow random noise to be added to the time tendency computation for this model. Remove the comments from these lines, and recompile the filter program only (make sure that you don’t recompile the perfect_model_obs program unless you want to have noise in the truth, too). Now try some assimilations and observe what happens to the ‘gravity waves’, the error, etc., when noise is introduced

in the assimilating model. You might want to vary the amplitude of the noise, also, which is set to 1/10 of the amplitude of the time tendency in the default version of the code.

VI. The Lorenz-96 model (see Appendix II)

This 40-variable model has become one of the tools of choice for data assimilation technique researchers. It is one of the mainstays of DART and is also explored in Jenny Sun's and Dale Barker's exercise set on 4D-Variational assimilation.

The DART working directory for the L96 model is DART/models/lorenz_96/work where you will find a set of files similar to that in the work directories for the L63 and 9var models. Initial conditions for the truth integration and a 20-member ensemble can be found in the files perfect_ics and filter_ics. These were generated in a similar fashion to those for the L63 and 9var models. The model was integrated from arbitrary initial conditions for 1000 1-day periods to produce a state on the attractor. Small perturbations were then made around this state for each ensemble member and each ensemble member and the true control were integrated for 1000 1-day intervals. The L96 model has had an arbitrary time interval of 3600 seconds assigned to its non-dimensional timestep for use in DART. The L96 variables have a well-defined physical location; in DART, they are equally spaced on a unit periodic interval with variable 1 at 0.0 and variable 40 at location 39/40.

The L96 model is big enough to examine a number of issues that arise in filters when the ensemble size is not significantly larger than the number of state variables. In particular, the impact of observations has to be localized for successful assimilations with a 20-member ensemble. To see this, first create an observation set with 40 randomly located observations on the unit interval. An input file for create_obs_set_def that generates such an observation set definition can be found in DART/models/lorenz_96/random_obs.input. In the work directory, enter

create_obs_set_def <../random_obs.input

to use this file as input to create_obs_set_def. Next, create an observation sequence definition by running create_obs_sequence with the default file set_def.out selected. Try a sequence with 200 observation times each separated by 12 hours. For the first case, set the namelist parameter as follows:

```
&filter_nml
   ens_size = 20,
   cutoff = 1000000.0
   cov_inflate = 1.0,
   start_from_restart = .true.,
   output_restart = .true.
   obs_sequence_file_name = "obs_seq.out",
   restart_in_file_name = "filter_ics",
   restart_out_file_name = "filter_restart",
   init_time_days = 0,
   init_time_seconds = 0,
   output_state_ens_mean = .true.
   output_state_ens_spread = .true.
   num_output_ens_members = 20
   output_interval = 1
```

&end

Use the standard matlab tools to examine the attractor of the model and the behavior of this assimilation. It is more challenging to get a feeling for the attractor behavior in higher order models like this. Can you see structures by looking at various triplets of state variables?

It should be apparent that the filter is diverging from the truth. Next, modify the cov_inflate in the namelist to 1.05 and see if this helps. Finally, try localizing the impact of the observations by setting cutoff = 0.2 in the namelist while retaining the covariance inflation of 1.05. You may want to try other values of inflation and cutoff to see how the behavior of the assimilation is affected. Be sure to try to understand how the correlations between state variables may be related to the need for localization. Use the matlab tools to study the spatial and temporal scales of correlation between a given state variable and all others for the various cases.

Once you have gained an understanding of how covariance inflation and localization impact performance in this model, you can compare filter results to some of the 4D-Variational results you obtained for the L96 model in exercises presented by Sun and Barker. Note that the version of DART you are using here does NOT support observations that are not taken at model timesteps (i.e. observations must be on the hour for the model as configured here).

In the L63 and 9var model, a 20 member ensemble was larger than the number of state variables. In the L96 model, this is not the case. It is interesting to explore the impacts of ensemble size on assimilation characteristics in this larger model (you might also want to go back to the smaller models and see if ensemble size matters at all; the filter_ics files in the base directory for L63 and 9var have 1000 ensemble members of initial conditions available so you can increase the ensembles up to 1000). The filter_ics file for the L96 model has 200 ensemble members available. Try increasing you ensemble size to 50, 100 and 200. In each case, explore the impacts of modifying the localization and covariance inflation parameters. Try to gain an understanding of the performance enhancements available from increased ensemble size.

For those with lots of spare time, an additional interesting comparison is to look at the performance of the filter for the L96 model with 40 identity observations (each variable observed exactly with error covariance 4.0) as opposed to the 40 randomly located variables in the cases done so far. To do this, you will need to create an observation set definition with the file DART/models/lorenz_96/identity_obs.input used as standard input to create_obs_set_def and then proceed as above.

VI. Dealing with big biases in models

The perfect model filter simulations that have been studied here neglect model error which is one of the biggest problems with real assimilation applications. To get a feeling for how easily model error can become the dominant issue, you can do a very simple simulation of model error in all of the DART models being used for the ASP colloquium. To do this, generate a true state and observations using perfect_model_obs, a given model, and a given observation set. Then, use the model's namelist to change the model timestep. For instance, in the L63 model, the model timestep is set to 0.01 by default. Try changin it to 0.095, 0.09 and 0.11 and observe the error

characteristics of the filter. Try using covariance inflation to deal with this problem. How far can the model error be increased while still having a covariance inflated filter that can generate some useful information? You can try this in the L96 model (or the bgrid model) if you have the inclination.

A number of interesting research directions could proceed from this study of systematic error. One could try to implement some of the ideas in Dick Dee's talks about model and observation error. For instance, the update in obs_increment could be changed to allow prior distribution estimates with tails longer than gaussians. Talk to Jeff if you're interested in pursuing some of these ideas.

VII. A non-deterministic filter (classical ENKF)

Peter Houtekamer's talk on ensemble filters described the 'classical' ensemble Kalman filter which uses a random number generator to generate a random sample of the observation distribution. The basic filter implementation in DART is a determinstic filter which does not use any random numbers during its execution. You can compare these two methods if you want by making a modification to the filter module. Edit your copy of DART/filter/filter.f90. Change the two occurences of obs_increment in this file to obs_increment4. If you look in the assim_tools module in DART/assim_tools/assim_tools_mod.f90 you'll be able to see the code for both of these subroutines. Delete the filter.o file in the working directory of whatever model you want to use (or run make clean) and then do

csh mkmf_filter; make

to get a filter that uses the stochastic filter. You can now compare your results for the different filters. Good places to do this are in the L63 model with observations that allow quite a bit of spread in the assimilating ensemble and in the 9var model in cases where significant gravity wave amplitude occurs in the original deterministic filter results.

An examination of the obs_increment4 subroutine in the assim_tools module will reveal a section of commented code near the bottom of the routine. This code performs a sort to guarantee that the observation increments are as small as possible (avoiding possible scrambling caused by the stochatic algorithms generation of perturbed observations). Uncommenting this code and recompiling the filter program will allow you to examine the impact of this sorting.

VIII. Particle filters

VIII. Estimation of model parameters

In addition to estimating values of model state variables, assimilation can also be used to generate a Bayesian estimate of model parameters. In a filter, the easiest way to do this is to recast a parameter as an additional state variable; in general, the time tendency of the estimate of a parameter is zero. The DART L96 model could be modified to allow the single model parameter (which has a fixed value of 8.0 in the standard case) to be assimilated by a filter. This would require modifying the code to be a 41 variable model with the estimate of this parameter being added into the state. This would be a non-trivial code modification, but could be completed in several hours if someone was interested.

IX. Localized observations

Many assimilation algorithms can be challenged by having spatially inhomogeneous distributions of observations. When some areas or variables are observed well but others are not, there is a potential for numerical problems where some dimensions in a probability distribution are very tightly constrained compared to others. Ensemble filters generally handle this situation well although there are potentially grave problems with the use of covaraiance inflation in models that have some poorly observed quantities. Try designing a localized observation set in the L96 model, say only observing in one half or one quarter of the domain while leaving the rest unconstrained. You should be able to see how 'information' is 'created' in the well-observed regions and then propagates out of this region into the rest of the domain.

What are the problems with covariance inflation in cases like this? If time permits, a regional observation case in the bgrid dynamical core is also interesting. Observations could be confined to a quarter of the globe (say 1/2 of the northern hemisphere). Again, it is interesting to see 'information' propagate in this system.

X. Assimilation results from a dry dynamical core (Appendix II.4)

Technical difficulties still preclude the exectution of the bgrid core on ocotillo. We hope to have these resolved shortly.

The GFDL B-grid dynamical core can be run in a low resolution configuration on ocotillo nodes and produce a roughly earth-like large-scale circulation. Filter assimilations can be carried out in the DART/models/bgrid_solo/work directory. You can begin by compiling the programs as for the low-order models using mkmf and make.

You can create observations sets and sequences as for the low-order models. However, there are also two additional programs that help to build observation sets: column_rand and ps_rand_local. You can build these programs using mkmf and make. Column_rand builds a set of randomly located column observations through an interactive interface. Be sure to specify 5 vertical levels. The results is a file column_rand.out which can be used as standard input for create_obs_set_def. Ps_rand_local works in a similar fashion but builds a set of randomly located surface pressure observations. These can be confined within a particular longitude-latitude rectangle if desired. A file ps_rand.out which can be used as standard input for create_obs_set_def is created. Note that the units for pressure in the model are Pascals and that error variances are input to the programs. An error variance of 10000.0 is equivalent to an error standard deviation of 1.0 millibars.

To get a feeling for the resources required to run the b-grid model, a suggested first run is as follows. Try designing an observation set with 400 randomly located column observations with ps error variance of 10000 and all other error variances 1.0. Then, build a sequence that observes these column obsevations every 12 hours (43200 seconds) for a total of 5 days (10 observation periods). Proceed to run perfect_model_obs using the perfect_ics file in the work directory. Then try a 20 member ensemble assimilation with covariance inflation of 1.0 and localization of 0.2 (the units on the sphere are radians).

The standard set of matlab analysis tools can be applied to b-grid model output. The script plot_ens_err_spread plots diagnostics by variable and level for the bgrid model while all the other standard scripts work in the same fashion as for the low order models.

Because there is much more spatial structure to the b-grid state vector, it is convenient to examine it using the simple NetCDF viewing tool ncview. Ncview can be applied directly to the True_State.nc, Prior_Diag.nc and Posterior_Diag.nc files. In addition, the NCO tools can be used to extract portions of these files and compute differences for further analysis. To extract the ensemble mean from a diagnostic file, use a command like:

ncrcat -dim,copy,11,11 Prior_Diag.nc Prior_Ens_Mean.nc

This extracts the 11th copy of the data (the ensemble mean is copy n+1 where n is the number of ensemble members output to the diagnostic file) from file Prior_Diag.nc and puts it in Prior_Ens_mean.nc. The command

ncdiff Prior_Ens_Mean.nc True_State.nc Prior_Err.nc

produces file Prior_Err.nc which contains the prior error.

ncdiff Posterior_Ens_Mean.nc Prior_Ens_Mean.nc Innovation.nc

produces a file containing the innovations if Posterior_Ens_Mean.nc is generated analogously to Prior_Ens_mean.nc. You can extract the ensemble spread by

ncrcat -dim,copy,11,11 Prior_Diag.nc Prior_Ens_Mean.nc

(the ensemble mean is copy n+2 where n is the number of ensemble members output to the diagnostic file).

Those of you with time can think of a variety of experiments you might want to try with the b-grid model. Please be realistic about compute times (you are free to run long jobs overnight on ocotillo nodes if you want) and especially storage. For some longer runs you may want to think about modifying the frequency with which diagnostic output is generated using the namelist variable output interval.

Appendix I: Summary of Generic Matlab Diagnostic Tools available

A set of matlab scripts/functions is available to do most basic diagnostics for DART ensemble assimilations. Most of these scripts are generic for all of the models used in the workshop and are located in the directory DART/matlab in the DART directory tree. All matlab scripts in this directory will need to be copied (sftp'ed) to an appropriate work directory on a computing platform where matlab can be run. For the ASP summer colloquium, the most convenient place is to create a directory called matlab_scripts on your DEC/COMPAQ workstation home directory. Once the scripts are copied, executing matlab in this directory will allow you to analyze output from DART experiments. The scripts expect to have NetCDF output files for the truth (generated by perfect_model_obs) and for an ensemble assimilation (generated by filter) available, either in the matlab_scripts directory or another directory which can be specified for matlab use.

For the ASP colloquium, we recommend you enter the command

matlab -nojvm

which presents a more stable interface to matlab. The startup.m script sets default values for the matlab variables truth_file and diagn_file. The truth_file is set to 'True_State.nc' in the working

directory and the diagnostic file to ‘Prior_Diag.nc’ in the working directory. To change this, issue a matlab command like

diagn_file = ‘../my_dir/Posterior_Diag.nc’

It is most convenient to keep your matlab scripts and output NetCDF files in the same directory. Then, the only need for changing the default files will normally be to toggle between prior and posterior diagnostics, i.e.

diagn_file = ‘Posterior_Diag.nc’

Eight basic diagnostic scripts are documented briefly below. On-line help for each can be obtained by typing (for instance);

help plot_bins

to a matlab window.

1. plot_bins

Plots rank order histograms (see Appendix V) for a set of state variables; the truth comes from the truth_file and the ensembles from the diagn_file (see Matlab intro above). Ensemble size is determined by the number of ensemble members available in the NetCDF diagnostic files output from the filter program (controlled via namelist parameter num_output_ens_members). For the L63 and 9var models, plots are automatically produced for all (3, 9) state variables. For the L96 model, the default is to produce plots for state variables 1, 13, and 27, however, different state variables can be selected using the matlab script. This same script can be used to select a set of variables from the bgrid core model.

2. plot_correl

The user is queried for a given state variable and a time referred to as the base point. The function creates a space-time plot of the sample correlations with the base point from all other variables at all times in the NetCDF files. This function is not currently supported for the Bgrid model where the large number of state variables precludes a meaningful display.

3. plot_ens_err_spread

Plots time series of the absolute error of the ensemble mean and the ensemble spread (standard deviation) for a set of state variables. For L63 and 9var, plots are produced for all state variables. For the L96 model, the default plot variables are 1, 13, and 27. Users can interactively specify a set of variables to be plotted for the L96 and Bgrid models. A summary statistic of the time average of the errors and the spread is given.

4. plot_total_err

Plots a time series of the total error (the distance in the full phase space of the model) of the ensemble mean from the truth and the ensemble spread.

5. plot_ens_mean_time_series

Plots a time series of the ensemble mean and the corresponding truth for a set of state variables. For L63 and 9var, plots are produced for all state variables. For L96 and the bgrid core, variables are selected as for plot_bins.

6. plot_ens_time_series
Plots time series of the set of available individual ensemble members, the ensemble mean and the truth for a set of state variables. For L63 and 9var, plots are produced for all state variables. For L96 and the bgrid core, variables are selected as for plot_bins.

7. plot_var_var_correl
Plots a time series of the correlation between a single base state variable at a base time and another state variable (could be the same one as the base variable) at all available times. This is a slice across the plotted output from plot_correl.

8. plot_phase_space
Allows three-d visualization of time evolution on a three-dimensional slice through a models phase space. Slices from several different time evolutions can be superposed in different colors. For instance, a slice across variables 1, 2, and 3 from the truth for a 9var model integration could be superposed with the corresponding slice for the ensemble mean of an assimilation. The interface to plot_phase space requires specification of a number of matlab variables. Here, sample specifications are provided along with discussion of what these variables do. Typing help **plot_phase_space** in a matlab window also gives a detailed discussion.

```
fname = 'True_State.nc'      #NetCDF file to get data from
var1 = 1                     #var1, var2, and var3 select the three dimensional slice
var2 = 2                     # through phase space for the plot
var3 = 3                     # Additional tools are provided for selecting bgrid slices
ens_mem = 'true_state'       # Selects which time series in the file is plotted
                             # Other possibilities are 'ensemble mean',
                             # 'ensemble spread', or 'ensemble member?'
                             #where ? mark is the
                             # number of an individual ensemble member
ltype = 'b-'                 # Specifies the color and type of line plotted; see help plot
```

The help file also describes how to overlay two or more trajectories in phase space on the same plot. Particularly revealing can be plots that overlay the truth with the ensemble mean, the truth with an individual ensemble member, the ensemble mean with an individual ensemble member, or two different ensemble members.

The resulting plots can be viewed using matlab's three dimensional visualization tools which are accessed by menus across the top of the plot. Ask for help to learn how to play with this if you don't know already.

Appendix II: Dynamical systems

1. Lorenz-63 Model

The first dynamical system examined with the methods described in section 2 is the 3 variable convective model of Lorenz (1963), referred to here as the Lorenz-63 model, which has

become one of the mainstays for the study of chaotic systems (Palmer 1993, Pasmanter 1995).

The model's 3 equations are:

$$\dot{x} = -\sigma x + \sigma y \quad (1)$$

$$\dot{y} = -xz + rx - y \quad (2)$$

$$\dot{z} = xy - bz \quad (3)$$

where the dot represents a derivative with respect to time. The model is integrated using the standard values for the parameters σ, b and r and the time step described in the original Lorenz paper resulting in a system with chaotic dynamics.

2. Lorenz 9-variable model

The 9-variable model used here is a truncated version of the primitive equations (Lorenz 1980) used to study the behavior of gravity waves (Lorenz and Krishnamurthy 1987) and some ensemble assimilation methodologies in Anderson and Anderson (1999). The model equations are:

$$\dot{X}_i = U_j U_k + V_j V_k - v_0 a_i X_i + Y_i + a_i z_i \quad (1)$$

$$\dot{Y}_i = U_j Y_k + Y_j V_k - X_i - v_0 a_i Y_i \quad (2)$$

$$\dot{z}_i = U_j (z_k - h_k) + (z_j - h_j) V_k - g_0 X_i - K_0 a_i z_i + F_i \quad (3)$$

$$U_i = -b_j x_i + c y_i \quad (4)$$

$$V_i = -b_k x_i - c y_i \quad (5)$$

$$X_i = -a_i x_i \quad (6)$$

$$Y_i = -a_i y_i \quad (7)$$

where each equation is defined for cyclic permutations of the indices (i, j, k) over the values (1, 2, 3). The X, Y and z variables can be thought of as representing divergence, vorticity and height respectively while the subscripts can be viewed as representing a zonal mean plus two wave components for each of the three fields. Parameters are selected as in Lorenz (1980) in order to pro-

duce a chaotic system. The timestep is set to 1/12 of a non-dimensional time unit; it is convenient to use timesteps as the unit of time throughout the remainder of this report.

There has been a great deal of discussion about the detailed structure of the attractor of this model (existence, non-existence reference). For the purposes of this discussion, the most important point is that long integrations of the model equilibrate to trajectories that have an approximately balanced flow with most of the variation occurring on time scales of significantly greater than 100 steps. However, if even small ‘unbalanced’ perturbations are added to an equilibrated model state, a period of transient evolution occurs, characterized by ‘gravity waves’ with periods of roughly 26 steps superposed on the underlying slow evolution. These gravity waves are most pronounced for the divergence variables where even small perturbations can lead to gravity wave amplitudes that are large compared to the amplitude of the slower equilibrated dynamics, however, for sufficiently large perturbations the impact of gravity waves also become apparent in the vorticity variables and to a lesser extent the height variables.

Perfect model assimilation studies in this simple model isolate several interesting facets of the performance of ensemble filter methodologies. The model’s interesting transient response to ‘off-attractor’ perturbations is a challenge to many simple assimilation methodologies. Methodologies that ignore prior constraints related to the local attractor structure will produce a large gravity wave response which may obscure the interesting slow evolution. Closely related is the fact that, once the assimilated states are off the attractor, the dynamics of the assimilated trajectory are statistically quite unique from those of the equilibrated control integration from which observations are taken. In this way, experiments in the 9-variable model present one simple way to examine the implications of model systematic error on ensemble filter assimilations.

Conveniently, the 9-variable model also removes several key challenges to filter assimilations methodologies that could obscure investigation of the issues in the previous paragraph. First, even the smallest meaningful ensembles are larger than the number of state variables so issues related to degeneracy of estimates of the prior probability distributions are not relevant. Second, it is simple to constrain the state of the system with a small number of observations which negates problems that develop in larger models in which many ‘remote’ observations that are not expected to be closely related to a particular state variable can lead to noise that swamps the signal from a smaller set of more relevant observations. Finally, the model is small enough that a detailed diagnosis of the behavior of all variables can be performed.

Many data assimilation schemes, especially of older vintage, have required the application of an initialization scheme when applied to large primitive equation models. Initialization schemes attempt to ‘balance’ arbitrary model states so that the slow dynamics is left unchanged while the amplitude of gravity waves is greatly reduced. An initialization procedure for the 9-variable model was developed by and can be used to provide some analogy to the results of applying initialization schemes in more realistic models.

3. Lorenz-96 model

The Lorenz 96 model is a variable size low-order dynamical system used by Lorenz (1996) and more recently by others including Lorenz and Emanuel(1998). The model has N state variables, X_1, X_2, ..., X_N and is governed by the equation

$$dX_i / dt = (X_{i+1} - X_{i-2})X_{i-1} - X_i + F$$

where i = 1, ..., N with cyclic indices. The results shown are for parameters with a sensitive dependence on initial conditions: N = 40, F = 8.0, and a 4th-order Runge-Kutta timestep with dt=0.05 is applied as in Lorenz and Emanuel.

4. GFDL B-grid dynamical core

This is the dynamical core of a full atmospheric GCM which is used by NOAA’s Geophysical Fluid Dynamics Lab for its ‘operational’ climate prediction runs (with parameterizations included of course). You can run a low resolution (60 longitudes by 30 latitudes by 5 levels) version of this model on the ocotillo nodes in a reasonable amount of time. This version has baroclinic instability and looks vaguely earth-like. Diagnostics and some assimilation results can be seen at ???

. Full documentation of the model can be found at:

(http://www.gfdl.noaa.gov/~fms/havana/havana_public_manual.html).

Appendix III: Ensemble filtering overview

The ensemble filter methods begin by integrating an ensemble of model states from the previous time at which data was assimilated forward in time until the next set of observations becomes available. For the results here, the prescribed observational error distributions are independent for each observation, so observations can be assimilated sequentially. Anderson (2002) demonstrates that for ensemble filters of the class used here, state variables can also be updated sequentially when an observation becomes available, so the update of a single state variable by a single observation is described here without loss of generality.

Let **h** be the forward observation operator so that **h**(**x**) gives the expected value of the observation given a state vector, **x**. An ensemble prior estimate of the observation is generated by applying **h** to each prior ensemble state estimate in turn,

$$\boldsymbol{y}_i^p = \boldsymbol{h}(\boldsymbol{x}_i^p), i = 1, \ldots, N \qquad (8)$$

where N is the ensemble size, subscripts index ensemble member, and the superscript p indicates a prior sample. Increments for the prior observation samples are generated by one of the two algorithms below (EnKF or EAKF) so that

$$\boldsymbol{y}_i^u = \boldsymbol{y}_i^p + \Delta y_i . \qquad (9)$$

Let s_i^p be a single scalar element of the state vector, $\boldsymbol{x}_i^p$. Then, updated values for the ensemble sample of s are generated by linear regression onto the increments for **y**

$$s_i^u = s_i^p + \Delta s_i = s_i^p + cov(\boldsymbol{s}^p, \boldsymbol{y}^p)/var(\boldsymbol{y}^p), i = 1, \ldots N \qquad (10)$$

where cov() is the prior sample covariance and var() is the prior sample variance from the ensemble.

The only difference between the EnKF and EAKF is the way in which the increments for the prior observation ensembles, Δy_i, are computed. For the EAKF, the variance and mean of the updated estimate of **y** are computed as:

$$\mathrm{var}(\mathbf{y}^u) = \{\mathrm{var}^{-1}(\mathbf{y}^p) + \mathrm{var}^{-1}(\mathrm{obs})\}^{-1} \qquad (11)$$

$$\overline{y}^u = \mathrm{var}(\mathbf{y}^u)\{\mathrm{var}^{-1}(\mathbf{y}^p)\,\overline{y}^p + \mathrm{var}^{-1}(\mathrm{obs})\,y^o\} \qquad (12)$$

where var(obs) is the observational variance (a function of the observing system) of the scalar observation y^o, and the overbar indicates an ensemble mean. The updated ensemble for **y** is com-

puted by shifting the mean and compacting the prior distribution around the mean so that the updated ensemble has exactly the mean and variance from (11) and (12),

$$y_i^u = (y_i^p - \bar{y}^p)\sqrt{var(\boldsymbol{z}^u)/var(\boldsymbol{z}^p)} + \bar{z}^u . \quad (13)$$

The perturbed observation EnKF uses a random number generator to create N samples from the observational error distribution (given as a Gaussian with variance var(obs) here) and creates a sample of the 'observation distribution' by adding these to the observation value, y^o. Members of this 'observation distribution' are then paired with the prior estimate samples, y_i, and the updated values are computed as

$$y_i^u = var(\boldsymbol{y}^u)\{var^{-1}(\boldsymbol{y}^p)y_i^p + var^{-1}(obs)y_i^o\} \quad (14)$$

where y_i^o is the ith sample of the 'observation distribution', and var($\mathbf{y}^u$) is computed as in (11).

Implementations of the ensemble Kalman filters that exactly follow the algorithms above often suffer from filter divergence in which the observations receive progressively less weight as prior estimates of the state become increasingly overconfident. In some cases below, a covariance inflations factor, α with values slightly greater than 1 is applied to reduce the certainty of the prior estimates at each observation time by increasing the ensemble 'spread' of the prior

$$\boldsymbol{x}^p = \alpha(\boldsymbol{x}^p - \bar{\boldsymbol{x}}) + \bar{\boldsymbol{x}} \quad (15)$$

before applying the filter updates. Covariance inflation of this form is consistent with prior ensembles that have insufficient variance but no systematic error.

A closely related method for dealing with filter divergence modifies the weight given to the prior relative to the observation in (12) as

$$\bar{y}^u = var(\mathbf{y}^u)\{[\beta var(\mathbf{y}^p)]^{-1}\,\bar{y}^p + var^{-1}(obs)\,y^o\}. \quad (16)$$

Values of β greater than 1 weight the observations more heavily in computing the updated mean without impacting the updated variance. This would be appropriate if prior state estimates had appropriate variance but were biased so that the ensemble variance indicated too much confidence in the estimate of the prior values.

Appendix IV: Measuring filter performance

A number of metrics are used to evaluate the performance of the ensemble filter assimila-

tions. The time-averaged RMS error of the ensemble mean and the time-averaged mean RMS error of the ensemble members are the primary measure of assimilation error. In addition, the ratio, *Ra*, of the time-averaged RMS error of the ensemble mean to the time-averaged mean RMS error of the ensemble members is computed (for the complete state vector and individual state components). Murphy (1988, 1990) shows that the expected value of this ratio (referred to as the RMS ratio hereafter) is

$$E(Ra) = \sqrt{(N+1)/2N} \tag{17}$$

for an ensemble in which the truth is statistically indistinguishable from a member of the analysis ensemble for an N-member ensemble. The ratio of *Ra* to the expected value for a given experiment,

$$r = (Ra)/E(Ra) \tag{18}$$

referred to as the normalized RMS ratio, is used here to evaluate ensemble performance as in Anderson (2001).

Rank histograms (also known as Talagrand diagrams) are also used to evaluate the performance of assimilations for individual state variables. The order statistics of the analysis ensemble of a scalar quantity are used to partition the real line into n+1 intervals (bins) at each analysis time; the truth falls into one of these n+1 bins. A necessary condition for the analysis ensemble to be a random sample of the distribution of the truth is that the distribution of the truth into the n+1 bins be uniform (Anderson 1996). This is evaluated with a standard chi-square test applied to the distribution of the truth in the n+1 bins with the null hypothesis that truth and the analysis ensemble are drawn from the same distribution.

Appendix V: Localization for 'Distant' Observations and Maintaining Prior Covariance

One of the advantages of the EAKF and EnKF is that they can maintain much of the prior covariance structure even when applied independently to small subsets of state variables. This is particularly important in applications where each state variable is updated independently from all others. If, however, two state variables that are closely related in the prior distribution are impacted by very different subsets of observations they may end up being too weakly related in the updated distribution.

One possible (expensive) solution, would be to let every state variable be impacted by all observations. This can, however, lead to another problem that has been noted for the EnKF. Given a large number of observations that are expected to be physically unrelated to a particular state variable, say because they are observations of physically remote quantities, some of these observations will be highly correlated with the state variable by chance and will have an erroneous impact on the updated ensemble. The impact of spuriously correlated remote observations can end up overwhelming more relevant observations (Hamill et al 2001).

Following Houtekamer and Mitchell (2001), one can multiply the covariances between prior state variables and observation variables in the joint state space by a correlation function with local support. The correlation function used is the same fifth-order piece wise rational function presented for use in R3 by Gaspari and Cohn (1999, their equation 4.10) and used in Houtekamer and Mitchell. This correlation function is characterized by a single parameter, c, that is the half-width of the correlation function. The Schur product method used in Houtekamer and Mitchell can be easily computed in the single state variable cases presented here by simply multiplying the sample covariance between the single observation and single state variable by the distance dependent factor from the fifth-order rational function.

Appendix VI: Namelist variables for DART modules

A list of the namelists for DART modules used for the ASP colloquium and the purpose of each variable follow. The default value (the value used if the entry is not specified in the namelist in input.nml) of each quantity is listed in parentheses.

A. Program filter

1. async (false): Used if model is to be integrated by a separate process. True should not be selected for asp exercises (exceptions may exist for large models, check with Jeff).
2. ens_size (20): Size of ensemble to be used in filter.
3. cutoff (200.0): Half-width of localization function. Units depend on the location_mod that is being used for the model used.
4. cov_inflate (1.0): Value of covariance inflation factor.

5. start_from_restart (false): True if restart should be read from file. False if filter should perturb around single state. In either case, file specified by restart_in_file_name is read.
6. output_restart (false): True if a restart file is to be output so that filter run could be extended.
7. obs_sequence_file_name (‘obs_sequence’): File from which the input observation sequence file should be read to control the filter execution.
8. restart_in_file_name (‘filter_restart_in’): File from which a restart is read in if start_from_restart is true. Otherwise, file from which to read a base state to be perturbed to get initial ensemble.
9. restart_out_file_name (‘filter_restart_out’): File name for output restart file.
10. init_time_days (-1): If this value is negative, the initial time of the ensemble members is set to the time found in the restart file. If the value is not negative, then the initial time for the ensemble members is set to be (init_time_days, init_time_seconds). For idealized model experiments, it is often convenient to reset the time to (0 days, 0 seconds) after each run of filter, thus allowing the same observation set to be used repeatedly.
11. init_time_seconds (-1): See init_time_days.
12. output_state_ens_mean (true): If true, the ensemble mean is output to the Prior_Diag.nc and Posterior_Diag.nc files.
13. output_state_ens_spread (true): If true, the ensemble spread is output to the Prior_Diag.nc and Posterior_Diag.nc files.
14. num_output_ens_members (0): The number of ensemble members to be output to the Prior_Diag.nc and Posterior_Diag.nc files. If less that the ensemble size, the first num_output_ens_members members are output.
15. output_interval (1): Specifies how often output is to be written to Prior_Diag.nc and Posterior_Diag.nc. Units are assimilation periods (1 means output every time there are observations, 2 means every other time there are observations, etc.).

B. Program perfect_model_obs

1. async (false): See filter.
2. obs_seq_in_file_name (obs_seq.in): File from which input observation sequence definition that controls perfect_model_obs execution is to be read.

3. obs_seq_out_file_name (obs_seq.out): File to which output observation sequence file is to be written.
4. start_from_restart (false): If true, initial condition is read from file specified by restart_in_file_name. If false, model starts from model-specific arbitrary initial condition.
5. output_restart (false): Output a restart file if true.
6. restart_in_file_name (‘perfect_restart_in’): File from which to read initial conditions.
7. restart_out_file_name (‘perfect_restart_out’): File to which to write restart.
8. init_time_days (-1): See filter.
9. init_time_seconds (-1): See filter.
10. output_interval (1): Frequency with which to write output to True_State.nc file. See filter.

C. Module model_mod.f90 (Lorenz-63)

1. sigma (10.0): Model parameter.
2. r (28.0): Model parameter.
3. b (8/3) Model parameter.
4. deltat (0.01): Non-dimensional timestep for model. Associated time increment is always 3600 seconds.
5. output_state_vector (true): Chooses netcdf output format. False is not supported for Lorenz-63.

D. Module model_mod.f90 (9 variable model)

1. g (8.0): model parameter. Value of 9.9 gives a very different attractor structure.
2. deltat (1 / 12): Non-dimensional timestep for model. Associated time increment is always 3600 seconds.
3. output_state_vector (true): Select whether to write single 9 element vector or triplet of 3 element vectors with specified types. False not currently supported.

E. Module model_mod.f90 (Lorenz-96)

1. model_size (40): Size of model state vector.
2. forcing (8.0): Model forcing parameter.

3. output_state_vector (true): Chooses netcdf output format. False is not supported for Lorenz-96.

F. GFDL bgrid dynamical core:

Appendix VII: Extending idealized assimilation cases

More intensive exploration of assimilation problems may require extending assimilation runs or doing runs that are long enough that they are conveniently split into several parts. In idealized perfect model cases, the most straightforward way to do this is to use the same observation sequence file repeatedly. After the first segment of an experiment (perfect_model_obs followed by filter) is complete, the filter_restart and perfect_restart files can be copied to filter_ics and filter_restart. If the init_time_days and init_time_seconds namelist entries are set to 0 (see Appendix VI) for both perfect_model_obs and filter, a subsequent run of perfect_model_obs and filter will extend the previous segment. The True_State.nc, Prior_Diag.nc and Posterior_Diag.nc files will be overwritten, so if you need a long series of diagnostics, you must rename these after each segment to a unique name. The sequence of True_State.nc files may be concatenated by using the NCO command ncrcat followed by the names of all the individual segment files and a destination file for the concatenated series. For instance,

ncrcat True_State1.nc True_State2.nc True.nc

creates a file True.nc with the concatenated output (ignore the warnings from ncrcat about a non-increasing time index). The same can be done for the Prior_Diag.nc and Posterior_Diag.nc and the standard analysis tools can then be applied.

Table 1: Ocotillo node assignments

Team	mmm Accounts	Members	Ocotillo Slave node
1	mmm01 mmm02		node1
2	mmm03 mmm04		node2
3	mmm05 mmm06		node3
4	mmm07 mmm08		node4

Table 1: Ocotillo node assignments

Team	mmm Accounts	Members	Ocotillo Slave node
5	mmm09 mmm10		node5
6	mmm11 mmm12		node6
7	mmm13 mmm14		node7
8	mmm15 mmm16		node8
9	mmm17 mmm18		node9
10	mmm19 mmm20		node10
11	mmm21 mmm22		node1
12	mmm23 mmm24		node2
13	mmm25 mmm26		node3
14	mmm27 mmm28		node4