U. S. Department of Commerce
National Oceanic and Atmospheric Administration
National Weather Service
National Centers for Environmental Prediction
5200 Auth Road Room 207
Camp Springs, MD 20746

Technical Note

A genetic optimization package for the Generalized Multiple DIA in WAVEWATCH III ® †.

Hendrik l. Tolman‡
Environmental Modeling Center
Marine Modeling and Analysis Branch

Version 1.2, July 2012

---

† MMAB Contribution No. 289.

‡ e-mail: Hendrik.Tolman@NOAA.gov

This page is intentionally left blank.

Abstract

This report describes a genetic optimization package for the Generalized Multiple DIA (GMD) in the WAVEWATCH III modeling framework. This report will be updated as needed, depending upon development of this package or of the underlying wave model.

Change log

ver.	WW	rev. *	date	comment
1.0	3.15.1	11596	December 23, 2010	Initial MMAB No. 289. Experimental WW version used for **?**.
1.1	3.15.2	18380	March 22, 2012	Bug fixes up to 3.14.13 Transition to 'zeus' and Intel.
1.2	4.08	20337	July 18, 2012	Transition of GMD to development trunk .

*) svn revision number refers to manual. Check tags for software package.

Acknowledgments. Code management for WAVEWATCH III® is provided by NCEP. Arun Chawla provided a first filter for this report.

This report is available as a pdf file from

http://polar.ncep.noaa.gov/mmab/notes.shtml

Contents

This page is intentionally left blank.

1 Introduction

This report describes a portable package of scripts and FORTRAN programs that has been designed to work transparently with the WAVEWATCH III® wave modeling framework (**?**, model henceforth denoted as WW3) to perform the genetic optimization (e.g., **?**) of the Generalized Multiple DIA (GMD, **???**). In this report, fonts as used in the WW3 manual are used, with the file, program and directory names, `code and variables in scripts and command lines`, and FORTRAN source code identified by the fonts used here. Familiarity with the WW3 code and manuals is assumed.

Section 2 describes elements of the packages, and Section 3 describes how to use this package. Section 4 presents several graphical tools provided with this package.

This page is intentionally left blank.

2 Description of package

The genetic optimization package for the GMD assumes an implementation of the WW3 model in a conventional way, with access to all different wave model executables through the default search path of the operating system. It should be noted that the wave model needs to be set up separately for producing either the baseline conditions to which the GMD is tuned (typically a full solution to the nonlinear interactions with the `!/NL2` switch), or the actual GMD model when tuning this model, or any other S_{nl} approach to be run for direct comparison.

The optimization package uses three main directories. The first is the main directory (`$genes_main`) that contains the scripts and files with data necessary to run the genetic optimization of the GMD. These are isolated in their own directory for easy maintenance and portability of the package. The second is the data directory (`$genes_data`) where model data are stored. Such data include baseline data to which the GMD based model is tuned, generational information as developed inside the genetic optimization routine, and model results for display and documentation of model behavior through the optimization procedure. The third is the work directory (`$genes_work`), which provides temporary storage used during (test) computations only. Note that most of the optimization work is performed in the data directories in which the generational data is stored, as will be illustrated below.

The main package as stored in the `$genes_main` directory consists of several sets of scripts, programs and files. At the core of the package are the actual test cases, stored in `$genes_main/tests`:

test_01	Deep water time-limited growth test (single point) with constants wind speed and direction.
test_0X	like test_01 but with much larger minimum source term time step, used for quick and dirty assessment if GMD configuration is viable (not to be used in error computation).
test_02	Deep water fetch-limited growth test with constant wind speed and direction.
test_03	Deep water 'homogeneous front' case of **?**, Fig. 5.
test_04	Deep water homogeneous rotating wind case.
test_05	Deep water slanting fetch case.
test_06	Like test_01 with background swell field added.
test_11	Time-limited growth is shallow water with reducing depth.
test_12	Wind sea breaking on beach.
test_13	Swell breaking on beach.

For final independent testing of GMD configurations using more realistic conditions, two "real-world" test have been added. These test are used only for comparison, not in the actual optimization.

test_hr	A synthetic moving hurricane based on mww3_test_05 as distributed with WW3.
test_LM	A storm case on Lake Michigan.

Unlike the test used for optimization, the latter two tests require ancillary information such as two-dimensional model grids and evolving wind fields. Such data are stored in the directories

```
$genes_main/tests/test_hr.data
```

and

```
$genes_main/tests/test_LM.data
```

.

All test cases run exclusively in the work directory from which the script is called, and all produce full spectral and source term output for a selected set of locations and/or times (depending on the actual test). These test are typically not run independently, but inside a scripting environment. The corresponding scripts are found in the main directory (`$genes_main`), which also includes setup and cleanup scripts. The following scripts are found in this directory:

run_setup.sh	Setup basic system and system shell scripts variables such as `$genes_main`.
run_clean.sh	Clean up main and work directories. Optionally clean up programs and executables directories.
run_make.sh	Compile or recompile all auxiliary programs. Note that the user needs to set up the compiler options inside this script.
run_test.sh	Run a single test script and put the test results in the main directory for inspection.
run_base.sh	Run a set of test as identified in genes.cases.env and store the test data in `$genes_data` for later use as benchmark or for model intercomparison.
run_comp.sh	Process raw data files of a run to produce secondary data files for display and (optionally) compute errors against a baseline data set.
run_comp.all	Run run_comp.sh for a set of runs at once.
control.sh	General management script for genetic GMD optimization.
control.cycle	Run control.sh until the optimization is finished, or until a problem in the optimization has stopped progress (i.e., generation of new generations as expected).
control.stop	Gracefully shut down control.sh and control.cycle (does not influence control.cron below).
descent.sh	General management script for steepest descent GMD optimization, starting from a member of a population

Table 2.1: Setup and environment files used by the genetic optimization algorithm for the GMD. See also Fig. 3.1 for storage locations.

File	Location	Maintained by	Description
.genes.env	`$home`	run_setup.sh	Maintain basic setup
genes.spec.env	`$genes_main`	User	Spectral grid settings
genes.source.env	`$genes_main`	User	Source term settings
genes.snl.env	work dir.	User/scripts	S_{nl} (GMD) settings
genes.cases.env	`$genes_main`	User	Test cases to use
genes.w_*nnn*.env	`$genes_main`	User	Weights for errors for case test_*nnn*
genes.weights.env	`$genes_main`	User	Default error weights
genes.stats.env	`$genes_main`	User	Stats for opt.
genes.mask.env	`$genes_main`	User	Mask for opt. pars.
genes.expdef.env	`$genes_main`	control.sh	Base setup of exp.
genes.maps.env	`$genes_main`	User	Setup for error mapping
genes.terr.env	generation dir.	control.sh	Set filter error level

of the genetic optimization.

descent_sort.sh Convert quadruplets resulting from descent algorithm to sorted form.

map_it.sh Map error in parameter space by generating a regular discrete grid of parameters to be optimized.

reset.sh Reset environment parameters and files to those selected for an existing experiment.

convert_hr.sh Auxiliary program to convert GrADS files at NCEP from big (IBM) to little endian (Linux) format, and to set up difference fields for hurricane test.

convert_LM.sh Idem for Lake Michigan test case.

To assure consistency of model setup across test and run scripts, and to allow for flexibility of operations within scripts, several setup or environment files are maintained. These files are kept at different locations and maintained in different ways, as indicated in Table 2.1.

The main run time scripts described above use utility scripts and FORTRAN codes for their operation. Utility scripts are gathered in `$genes_main/ush`. Note that these scripts are not intended to be run independently.

get_cases.sh Evaluate the contents of file genes.cases.env.

get_err_test.sh Get combined error for single test case.

get_err_tot.sh Get combined error for all test cases.

get_err_par.sh	Get combined error per parameter for active test cases.
get_terr.sh	Get/set filter error level.
make_init.sh	Generate first population.
make_next.sh	Make next generation.
run_thread.sh	Master execution script for a thread in the engine.
run_one.sh	Sub-script in run_thread.sh to get the errors for a single member of the population.
thread_start.sh	Start the computational engine for computing errors.
thread_stop.sh	Stop the computational engine for computing errors.
thread_wait.sh	Wait for the computational engine to finish.
thread_check.sh	Background check on health of engine.
	These four scripts are kept for different machine setups in the files thread_start.sh.`$genes_engn` etc., and are linked to the above file names in the script control.sh.
make_maps.sh	Creates population for error mapping..
colorset.gs	GrADS color table setup script.
spec.gs	GrADS script for plotting of spectra.
source.gs	GrADS script for plotting of source terms.
1source.gs	GrADS script for plotting of single source term or spectrum (including preset copies for various tests).
map_hr.gs	GrADS script for map plotting for hurricane test.
map_LM.gs	GrADS script for map plotting for Lake Michigan test.

Note that for most GrADS script links are provided in the main directory for easy interactive use. Source codes for programs used specifically for this package are gathered in `$genes_main/progs`, and their executables (replacing file name extension f90 with x) are stored in `$genes_main/exe`:

constants.f90, w3timemd.f90, w3dispmd.f90, w3arrymd.f90	
	Service routines from the WW3 code used in the optimization package.
random.f90	Subroutines for random number generation.
cgaussmd.f90	Subroutines for processing normal distributions.
qtoolsmd.f90	Tools for quadruplet processing (subr.).
restart_co.f90	Combine two restart files.
process.f90	Process raw data from one or two runs to get a data set that is readable by, for instance Matlab (single case use), or get individual error estimates (two case use).
err_test.f90	Combine errors per test.
err_tot.f90	Get overall error.
err_par.f90	Combine errors per parameter for active tests.
testerr.f90	Set filter error level based on current population.
reseed.f90	Get a new random seed.

initgen.f90	Set up first generation.
chckgen.f90	Process generation for the purpose of computing and including errors.
sortgen.f90	Final sorting of generation by error. Also produces the clean-up population file (sorted quadruplets).
nextgen.f90	Make next generation.
getmember.f90	Extract member information from population file.
descent*N*.f90	Auxiliary programs for steepest descent optimization.
mapsgen.f90	Make generation for error mapping.

This page is intentionally left blank.

3 Using the package

The first step of setting up the genetic optimization package is to set up the underlying wave model, taking into account which physics options the model GMD is to be tuned to, and selecting Cartesian grid options (`!/XYG`)[1]. For this reference is made to the system manual of WW3 (**?**). The second step is to install the basic package by unpacking the tar file genes.tar. When the tar file is unpack by executing

tar -xvf genes.tar

a new directory ./genes is automatically generated. This should be the main directory $genes_main in the package. If another directory name is desired, this directory name needs to be modified before the next step of the initialization. Co-developers of WW3 can alternatively obtain this directory with its contents from the subversion repository at NCEP[2] The basic setup of the package is finished by executing

run_setup.sh

This will take the user through an interactive process that sets several environmental parameters for the package. This script will set the following shell script variables:

`$genes_main`	Main directory
`$genes_data`	Directory where all data are stored, including benchmark, and optimization data, as well as validation data for documenting model behavior.
`$genes_work`	Work space for scripts (scratch).
`$genes_expN`	Identifiers for the optimization experiment that is worked on presently. These represents subdirectories under the work directory (see below).
`$genes_base`	Base run identifier used in optimization (typically `WRT`).
`$genes_engn`	Identifier for type of engine used to compute the errors of a population member. The default is `single`, which uses a single threaded engine that can be used anywhere. Examples of other multi-threaded engines are also provided (see below).

These variables are stored in the setup file .genes.env (see Table 2.1), which is used by virtually all executable scripts. The experiment identifiers can be

[1] Except for test_LM, which requires the `!/LLG` switch.

[2] https://svnemc.ncep.noaa.gov/projects/ww3_utils/trunk/genes_gmd or look for tags under https://svnemc.ncep.noaa.gov/projects/ww3_utils/tags

reset by rerunning the setup code. Note that the setup file can also be edited manually. The next step is to compile all auxiliary programs used by the package by executing

run_make.sh

Note that the user will need to provide the proper compiler commands and settings in the header of this script, before executing it. Note that the corresponding directories ($genes_main/progs and ($genes_main/exe) can be cleaned up (listings and executables removed) by executing

run_clean.sh

and answering affirmative to the appropriate questions. The latter script automatically cleans up the main and work directories $genes_main and $genes_work. This completes the initial setup of the package. It is now prudent to test some of the test cases by running the run_test.sh script. This script executes a single test case and puts output including GrADS data files in the main directory $genes_main. For instance, the first test case is run in this way by executing

run_test.sh test_01

Note that run_clean.sh should be executed between test runs to assure that the test starts with a clean environment.

After this initial testing, benchmark or baseline datasets need to be generated. These data set can be used both in the optimization of the GMD (e.g., the exact interaction), or to identify progress (e.g., the DIA). Before the benchmark datasets can be generated, the WW3 model has to be compiled with the appropriate switch settings and other options as needed, and the appropriate namelist options need to be set in the file genes.srce.env. Furthermore, test cases for which benchmark data needs to be generated need to be identified in genes.cases.env. The example versions of these files as provided with the package are internally documented with respect to the data format in the files. After these preparations, benchmark data is generated by running the command

run_base.sh baseID yes

For each test case, a directory $genes_data/baseID/test_$nn$ is generated, where the files log.ww3, spec.ww3, srce.ww3, and part.ww3 are stored. These files contain spectral, source term (s_{nl}) and partitioned wave data, respectively. The first command line argument of this script identifies the subdirectory for the baseline datasets. The second command line parameter is optional, and identifies if GrADS data sets are to be saved with the general baseline data. Note that efficient computation of the one-and two-dimensional test grids (test_02, test_05, test_12, test_13, test_hr and test_LM) using the full nonlinear interaction may be

time consuming, and require a parallel (MPI) computational environment. Note that the above scripts are set up to run tests in parallel, but that this will require some manual intervention (as documented in the scripts). Note, furthermore, that this generally requires the tests to be run in some kind of batch mode. As such batch scripts are highly dependent on the parallel compute environment, such a script is not provided with the package.

The above commands only generate raw test data, with the exception of the optional GrADS data that can be used to get a quick look at full two-dimensional spectra $F(f, \theta)$ and source terms $s_{nl}(f, \theta)$. Detailed diagnostics files (all_data.ww3 in data directories), specifically for processing with Matlab, can be generated with the command

run_comp.sh test_01 DIA

which processes DIA results for test case test_01. Errors for this case against WRT results are generated by running

run_comp.sh test_01 DIA WRT

which generates an error file errors.test_01.DIA.WRT in the work directory. To run this script for all tests for a given model setup the command

run_comp.all none DIA WRT

can be used, generating raw data files for the DIA and WRT model runs. To generate errors for WAM and WW3 runs against the WRT runs, this command is executed as

run_comp.all WRT WAM WW3

Note that the run_comp.sh or run_comp.all scripts together with several Matlab scripts have been used to generate most of the graphics in **?**.

This completes the setup of the test environment. The next step is to set up the GMD model for optimization. Before this is discussed, it is important to understand the file and directory structure used to perform the optimization and to save the relevant intermediate and final data. The directory and file structure is outlined in Fig. 3.1. In the setup steps performed so far, the three main directories have been defined by running run_setup.sh, and can be redefined by rerunning this script or by manually editing the file .genes.env in the users home directory. The directories with baseline information (identified here as WRT, WAM and WW3, actual names set by user) have already been created and filled by running run_base.sh and either run_comp.sh or run_comp.all. There are two levels of directories to hold the actual generational, descent and error mapping data from the optimization approach (`$genes_epx1` and `$genes_exp2`).

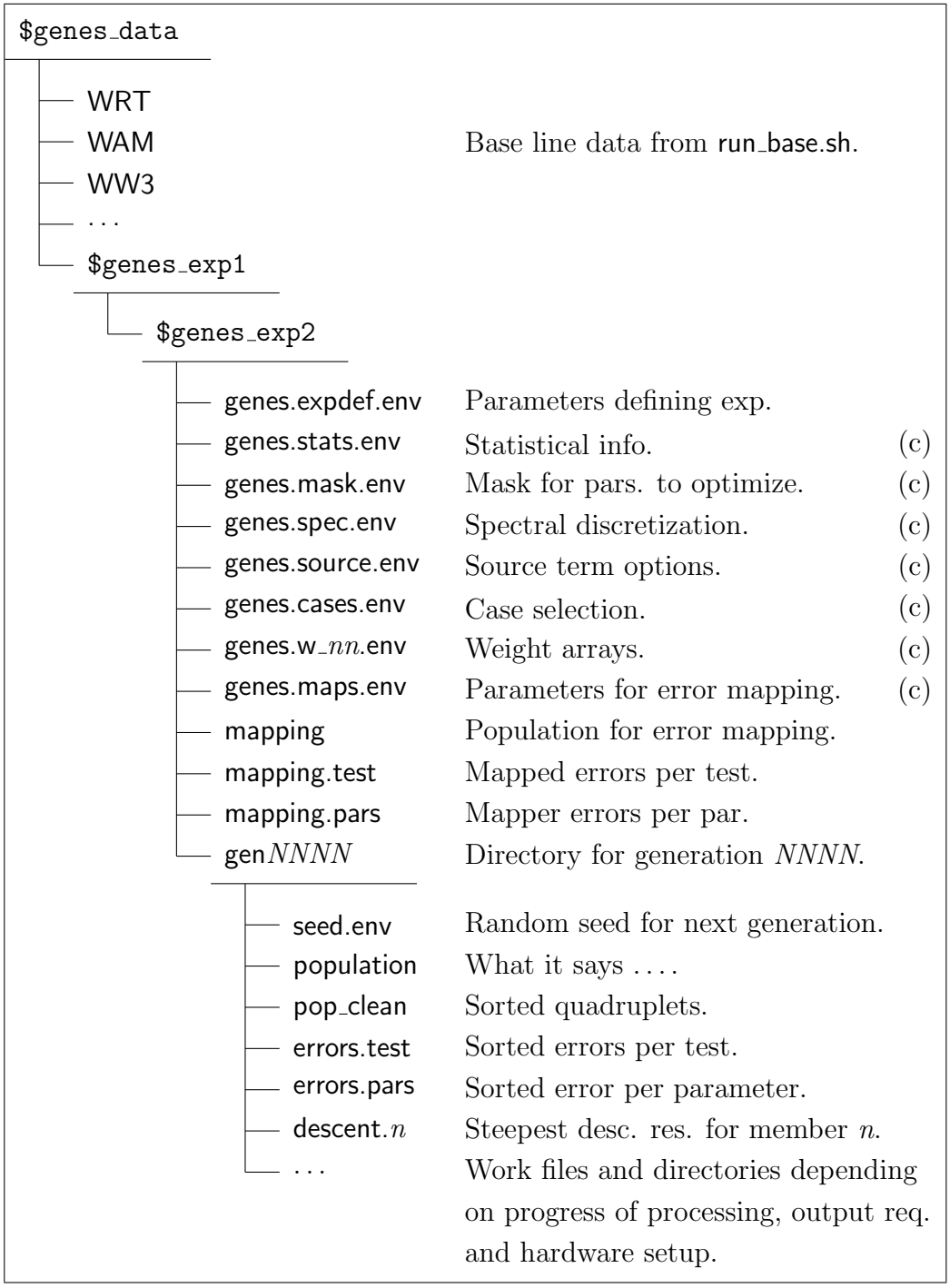

Fig. 3.1 : Layout of directories with data for the GMD optimization. (c) identifies copy of file with file used stored in `$genes_main`,

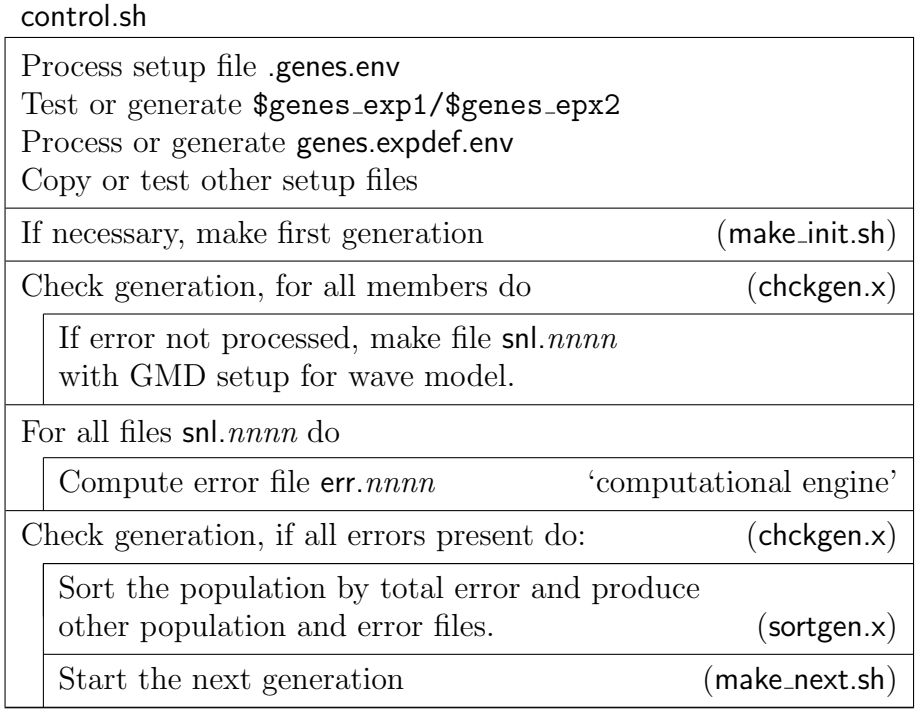

Fig. 3.2 : Structure of the main genetic optimization routine control.sh. *The 'computational engine' is described in the manuscript.*

The first is intended to hold all data for a general GMD layout. For instance, all experiments with a single component GMD with a traditional quadruplet layout could be saved under a single directory `$genes_epx1`. Several directories `$genes_epx2` then can hold experiments for deep versus shallow optimization, different initial optimization seeds etc. Files identified with '(c)' are kept in the directory for documentation only, the actual files used by the package are stored in the main directory `$genes_main`.

The main script controlling the optimization is the script control.sh. This script is designed to incrementally execute the model optimization, including the initial setup for the optimization process. The layout of the script is illustrated in Fig. 3.2. Running the command

control.sh

will get the optimization procedure started, or will continue it. When starting a new (part of an) experiment, this script should be run interactively first to set up the necessary directories and files. If the script is run to start a new experiment, run_setup.sh should be run first to set up the proper data directories. Furthermore, all environment files marked with '(c)' in Fig. 3.1 need to be set up properly in the main directory `$genes_main` before control.sh is executed for the

first time. Note that if work is to be continued on an existing experiment, the environment of the experiment can be restored by running

reset.sh $genes_exp1 $genes_exp2

which resets .genes.env and restores all other environment files as used with the selected experiment.

After processing the general setup file .genes.env in the users home directory, the first thing control.sh does is checking the existence of the file genes.expdef.env (see Fig. 3.1). If this file does not exist, it is generated from interactive information provided to control.sh. The setup file sets the following shell script variables:

`$genes_nq`	Number of representative quadruplets.
`$genes_npop0`	Size of initial population.
`$genes_npop`	Size of subsequent population.
`$genes_ngen`	Number of generations to be considered.
`$genes_seed`	Initial random seed. Note that the package contains its own random number generator, making genetic optimization experiments reproducible as long as the same random seed is used.

After this file is initially generated, it can only be modified by hand. The only parameter to modify in such a way is the requested number of generations or possibly the random seed. Modifying the other parameters will result in failure of an ongoing optimization, and will require the removal of all generational directories gen*NNNN*.

The second step is to copy in all the environment files as indicated in Fig. 3.1, or, if these files are already there, to compare them against the corresponding files in the main directory (`$genes_main`). This step is added to assure that the experiment is not accidentally continued with modified settings of the GMD or of the optimization experiment. It also allows for a simple way to redo an experiment or to expand on an experiment by copying all setup files that are expected in the main directory (marked with '(c)' in Fig. 3.1) back to the main directory `$genes_main` (see command reset.sh as discussed above).

The next step is to check if the experiment has been started by generating an initial generation. If this generation is not present, the script make_init.sh using the program initgen.x is used to produce the first generation. This sets up the initial GMD generation according to information in the setup (*.env file) is produced, and checks that errors are not yet computed. To indicate the, errors are initialized at 999.999.

Following this, the program chckgen.x checks the population for cases for which the error still needs to be evaluated. For each such population member *nnnn*, a file with the GMD namelist information is created as the file snl.*nnnn*.

Computing the corresponding error files err.*nnnn* represents the main effort of the optimization procedure, and is performed by the computational 'engine' as will be described below. After all errors have been evaluated, the program chckgen.x is run again to include the errors from the error files in the population data set population. Note that in its first call in the script, chckgen.x processes all error files err.*nnnn* from previously aborted runs of control.sh, hence avoiding duplication of work already performed.

If all errors have been computed, the program sortgen.x sorts the population by ascending error into the file population, and furthermore generates the file pop_clean with the same information but with the quadruplets sorted so that $\mu \leq \lambda$ and with ascending order for λ per quadruplet. Note that the latter sorted information is used to eliminate duplicates in the generation of the next generation. This program also produces the error files errors.pars and errors.test. Note that the latter files are saved for later analysis, but are not used in the optimization process and/or control.sh. Finally, the script make_next.sh and the program nextgen.x produce the next population, again with dummy values for the error for new members of the population.

This ends the description of the script control.sh. This script is not set up to cycle through consecutive generations. The script is cycled by the additional command

control.cycle

which sets up genes.expdef.env interactively as needed, and then cycles control.sh until it works on the same generation for the third time. The latter indicates a failure in the optimization attempt, or reaching the required number of generations. Each call of control.sh has its own output file control.*nnnn*.out for later inspection. Here *nnnn* identifies the sequence number of runs of control.sh, not the generation number. Finally, to cleanly stop the optimization process managed by control.cycle and control.sh, execute the command

control.stop

which updates information in the file temporary file control.status. This file is queried by the control.cycle and control.sh to decide keep providing new cases to the compute engine, and the file is removed as the scripts complete. Note that all started computations are completed before the scripts stop, so that control.stop does not result in an immediate stopping of the optimization scripts.

This leaves the description of the computational engine for computing errors for members of the population. This engine is designed to be able to run a set of parallel job streams, which are dynamically fed with work to do to optimize load balancing. At the center of the engine is the utility script run_thread.sh. This script manages a single thread of the computational engine. If the hardware

allows for parallel threads of computation, multiple versions of run_thread.sh are operating simultaneously. How this is achieved depends on the hardware and job scheduling software used. The starting and stopping of the threads of the engine is managed by the scripts

thread_start.sh	Start a number of copies of run_thread.sh.
thread_stop.sh	Stop all copies of run_thread.sh .
thread_wait.sh	Wait for all copies of run_thread.sh to stop.
thread_check.sh	Check health of each copy of run_thread.sh.

These scripts are actually links to scripts thread_start.sh,`$genes_engn` etc., where `$genes_engn` represents a setup of the computational engine. Options for for the engine (`$genes_engn`) provided with the package are:

`single`	A single copy of run_thread.sh runs in the background on the present machine.
`snits`	Many copies of run_thread.sh on several nodes of a Linux cluster (snits is Hendrik's present cluster). Note that this cluster approach used background runs of the script through ssh started from an interactive node, and does not use a batch scheduling system. The system is also set up to run some processes on the front end of the cluster if so desired.
`IBM_ll`	Running on an IBM supercomputer with LoadLeveler as a batch processor. control.sh or control.cycle is run interactively, whereas the computational engine is submitted as a batch job. The size of the engine is presently hardwired to 128, and can be reset in the '`total_tasks`' job card in section 1.a of the script thread_start.sh.IBM_ll.
`Moab`	Running on a Linux cluster with the Moab job scheduler, the same way as outlined for the IBM above.

For each copy of run_thread.sh with number *nn* the following files and directories are maintained in the generation directory:

thread_*nnn*	Work directory for this copy of the script.
thread_*nnn*.out	Output file for this copy of the script.
tdata.*nnn*.out	File used for communication between control.sh and each individual copy of run_thread.sh that is running.

All these files and directories are removed from the generation directory after all imputations for that directory have been completed. The file tdata.*nnn* contains a single text string. Valid values of this string are

`starting`	Initial string value set by thread_start.sh.
`ready to go`	run_thread.sh signaling control.sh that it is ready to accept work.
`snl.`*nnnn*	control.sh signaling run_thread.sh to work on the case as described by the file snl.*nnnn* in the work directory.
`done`	control.sh signaling run_thread.sh to stop operations and end its execution.

Finally, run_thread.sh uses the utility script run_one.sh to compute the errors for the GMD as described by snl.*nnnn*.

It should be noted that a disproportionally large part of the computational effort in the genetic optimization is lost for computing test cases with unstable model behavior. In such cases, the dynamic time step becomes extremely small, and hence the model run time becomes extremely long. This computational effort is essentially waisted, because the cases will have large errors and will hence have no impact on subsequent populations. To eliminate such useless computations, the error from case test_01 (or test_11 for the shallow water tests) is assessed in run_one.sh. If this error is too big, other test cases are skipped, and the error of test_01 is used as a proxy for the error of running all tests.

However, even test_01 can take up much computational effort by itself, due to the small minimum source term integration time step allowed (1 s). To speed up initial computations for unstable cases, the test case test_0X has been introduced. This test case is identical to test_01 with the exception that the minimum source term time step is set to 300 s. This test can give a 'quick and dirty' assessment of the viability of the configuration of the GMD, and will run fast, independent of the viability of the model configuration. Note that this test is run only to check viability. If viability is established, test_01 will be used to compute errors for this configuration, and results of test_0X will be ignored. In some cases the small time step in test_01 will lead to large model errors whereas test_0X shows much more moderate errors. Therefore, a second filter test is applied to the errors produced by test_01.

The filtering is performed in the basic computational script run_one.sh, which is used by all optimization methods (genetic, descent, mapping). The filter levels are set in the script get_terr.sh in the utility script directory. This script set a minimum and maximum error filter level, and a factor used to set the error filter level based on the best member of the previous population. If the minimum error is set to 999.999, no filtering will be performed. The latter setting should be used if error mapping is performed.

This completes the description of the genetic optimization as performed by control.sh. As mentioned in **?**, the genetic optimization can be augmented with a

steepest descent method starting from given members of given populations. This can be achieved by executing

descent.sh igen ipop

where igen represents the generation number, and ipop represents the sequence number of the member of the sorted population from which the steepest descent is to start. The resulting sequence of incrementally improved configurations is stored in descent.*igen* in the generation directory. The script descent.sh uses the computational engine designed for control.sh. Result of the steepest descent search are saved in the generational directory gen*nnnn* with *nnnn* corresponding to igen in the file descent.*nnnn*, where *nnnn* corresponds to ipop. This file contains a set of incrementally improved GMD layouts, ending with the final optimal GMD for the chosen initial condition.

descent_sort.sh igen ipop

will sort the resulting best configuration from the file descent.*igen* and store the results in the file descent_sort.*igen*.

Finally, the tools developed for the genetic search algorithm also make it easy to develop a simple script to systematically map errors in parameter space. After the optimization masks are set in genes.mask.env, and information for discretizing the parameters spaces is set in genes.maps.env, the mapping of errors in the selected parameter spaces can be performed with the command

map_it.sh

This command generates a population mapping representing a preset grid in parameter space. The script then generates error for each population member using the computational engine developed for the script control.sh. Note that the population will not be sorted by error per population member. Once all errors are computed, error files mapping.test and mapping.pars are generated corresponding to the similar two error files generated by control.sh. Note that the script can be interrupted, and will take off where it stopped when restarted, as does control.sh. Note, furthermore, that changing the grid of the parameter space will require a full rerun of the script. The script could be expanded to incrementally generate mapping in parameter space, but such an option has not been implemented yet. Note, finally, that only a single mapping data set is produced per directory setup.

4 Graphics tools

Two sets of graphics packages have been used for the production of graphics in **?**, and can be used with the optimization package. The first is the GrADS package, as used as a default graphics option for the WW3 code. Standard plotting scripts for source terms and spectra for the test cases are described in the Section 2, and can be found in the directory

$genes_main/ush

To use these script, links are generally made to the main directory or a work directory, and the corresponding GrADS files are copied or linked there from the storage area $genes_data or created there by the script run_test.sh.

The second package used is Matlab®. All Matlab scripts are gathered in he directory

$genes_main/matlab

and Matlab should be run from this directory. Under this directory is a utilities directory

$genes_main/matlab/util

which should be added to the default directory search path for the matlab scripts to work properly. The utility directory contains the following utility scripts

wavnu2.m	Solve the dispersion relation for given frequency σ and depth d.
read_all_data.m	Read the file all_data.ww3 for processing with Matlab.
read_mapping.m	Read the files with error data form error mapping scripts.
read_pop_clean.m	Read a population file pop_clean.
read_descent.m	Read the files with error data form error mapping scripts.

The main directory for Matlab files contains the scripts

makeplots_base.m	Make line plots for baseline cases from file all_data.ww3.
makeplots_co_5.m	Make line plots for a number of baseline cases (originally 5) from file all_data.ww3.
makeplots_maps.m	Plot error maps from file mapping and mapping.pars.
makeplots_errors.m	Plot error evolution as a function of generation.

maps_extract.sh	Auxiliary script to extract mapping data from data files.
maps_clean.sh	Auxiliary script to clean up temp files generated by maps_extract .sh.
pop_extract.sh	Auxiliary script to extract population counts.
pop_clean.sh	Auxiliary script to clean up temp files generated by pop_extract .sh.

Note that all these (Matlab) scripts require script parameters to be set inside the scripts that point to the proper data files from the proper model runs. The shell scripts are used to effectively extract data from data files before reading these data into Matlab scripts.

This page is intentionally left blank.

APPENDICES

This page is intentionally left blank.

A Banana peels

This appendix presents some thoughts and considerations that may be helpful, as well as some traps I have fallen into when using this package.

- When this package was first used in (**?**), some of the computations were done on he 'snits' Linux cluster, some on the operational IBM supercomputer of NCEP. To be able to move results easily, big-endian files were used throughout, with the exception of the GrADS files, which by definition used the local approach (big-endian on IBM, little-endian on Linux), requiring the use of the convert_hr.sh and convert_LM.sh scripts. Now that we are moving to a full Linux environment at NCEP, most work s done here fully little-endian, without the need for using the convert scripts. Be careful that the endian approach in the comp and link scripts in WW3, and in the run_make.sh here are set up consistently. Otherwise, file read errors will occur in the scripts.

- When running an error mapping experiment, filtering of computations using the error of test_0X and others should not be on. This should be automatic, as filtering is done through the file genes.terr.env in the generational directory, set by control.sh.

- When a new experiment is set up, it is critical to make sure that the genes.expdef.env and genes.mask.env are consistent with respect to the number of quadruplets defined. There is presently no elegant error capturing if this is not the case.

- genes.source.env can be used with the proper compilation of WW3 to run a baseline case with selected GMD settings, and can then be used to compute errors etc. If however, GMD options are left on in this file, and control.sh is run, these source term setting will override the one that the optimization is trying to set, and all population members will artificially get the same error (this will crash the generation of the next generation). So make sure genes.source.env is reset to all comment lines before trying to optimize again.

This page is intentionally left blank.