

# Specialized Regression Testing for Griz2(Taurus) verses Griz4(Taurus and Mili) Codes

Victor M. Castillo  
Methods Development Group

December 16, 1999

## 1 Introduction

The new version of GRIZ (GRIZ4) has the capability of reading the Mili database format. For backward compatability, it retains the ability to read the Taurus database. This specialized regression test is used to compare the results of GRIZ2 and GRIZ4 using the Taurus database. A second comparison is then made using GRIZ4 to compare the results of reading the Taurus and Mili databases.

|                   |             |                   |             |                 |
|-------------------|-------------|-------------------|-------------|-----------------|
| GRIZ2<br>(Taurus) | — compare — | GRIZ4<br>(Taurus) | — compare — | GRIZ4<br>(Mili) |
|-------------------|-------------|-------------------|-------------|-----------------|

Comparisons are made for a variety of historic sample cases, including those using the SAND model. The set of sample cases used in this test are referred to as {**SAMP1**, **SAMP2**, **SAMP4**, **SAMP6**, **SAMP8**, **SND1**, **SND2**, **SND3**}. For each of these sample cases, derived and primitive results are compared for each element type available. In addition, a time history of one of the results (preferably a derived result) is compared for each element type (see Table 1).

### 1.1 Using the regression test

The operation of the regression test relies on a particular directory structure and a set of particular filenames, most of which are created by the program if needed. For each sample case (listed in the **CASE** dictionary), a directory must be created by the user which contains the Tuarus and Mili database for the **dyna3d** runs. The subdirectories **G2T**, **G4T**, and **G4M** are created within each of these case directories to store the results of the GRIZ2(Taurus), GRIZ4(Taurus), and GRIZ4(Mili) runs. The results are written to a file with the name convention

`casedir+“/”+testdir+“/”+result_type+elem_type+result+“his”`.

For example, the derived result **velx** for the **Nodal** element types from GRIZ4 using the Taurus database from **SAMP1** is called “**SAMP1/G4T/DRNodalvelx.his**”.

Table 1: Tested results for each element type

| Result       | Nodal   | Shell  | Brick  | Shared<br>or<br>Global   | Material |
|--------------|---|--|--|--|----------|
| Derived      | 'dispx'<br>'dispy'<br>'dispz'<br>'dispmag'<br>'velx'<br>'vely'<br>'velz'<br>'velmag'<br>'accx'<br>'accy'<br>'accz'<br>'accmag'<br>'pvmag' | 'surf1'<br>'surf2'<br>'surf3'<br>'surf4'<br>'surf5'<br>'surf6'<br>'eff1'<br>'eff2'<br>'effmax' | 'ex' 'ey'<br>'ez' 'exy'<br>'eyz' 'ezx'<br>'pdstrn1'<br>'pdstrn2'<br>'pdstrn3'<br>'pshrstr'<br>'pstrn1'<br>'pstrn2'<br>'pstrn3'<br>'relvol'<br>'evol' | 'sx'<br>'sy'<br>'sz'<br>'sxy'<br>'syz'<br>'syz'<br>'press'<br>'seff'<br>'pdev1'<br>'pdev2'<br>'pdev3'<br>'maxshr'<br>'prin1'<br>'prin2'<br>'prin3' |          |
| Primal       | 'nodpos[ux]'  | 'eeff_mid'   | 'eeff'   | 'ke'   | 'matpe'  |
| Time History | 'dispmag'   | 'effmax'   | 'prin1'  | 'ke'   | 'matpe'  |

The Unix **diff** operation are done on comparable result files and stored in the common case directory. These **diff** files are named by the convention:

`casedir+"/dir1+dir2+element_type"+"result_type`.

For example, the comparison between GRIZ2(Taurus) and GRIZ4(Taurus) for **Brick** element type for all **derived** results of **SAMP1** is called "SAMP1/G2TG4TBrick.dr".

Within each case directory, a **.info** file is used to indicate to the regression test what element types are available for testing. This file is generated by a separate **Python** script, **rtest\_init.py** (Generated by **rtest\_init.nw** – see **rtest\_init.ps|pdf|html|dvi** for more documentation. If the **.info** the regression test will not continue.

The **Python** script can be modified to select the cases to be used and the result type(s). For development of the regression routines, set **CASELIST=SHORTLIST**. For exhaustive testing, set **CASELIST=LOGLIST**. Also, for development, the time history (TH) comparisons are quickest. For more detailed testing uncomment the lines in **main** for the DR and PR result types.

## 1.2 Revision History

*Also see headers in individual functions.*

- 8-Jun-1999 Original version. (Victor M. Castillo)

- 14-Dec-1999 Converted to noweb form (This document) (VMC)

### 1.3 Program flow

This section describes the flow of the program and is used by NoWeb to construct the source code.

```

3a  <* 3a>≡
      #!/usr/bin/env /grdev/regrtest/bin/python
      <Define global variables and dictionaries 3b>
      <Main 4a>
      <Read info file 4b>
      <Make grizinit file 6>
      <Griz it 5>
      <Onespace 8b>
      <Diff it 7>
      <Rprint 8a>
      <Exit code 8c>

```

## 2 Code listing and description

Each code section is listed below with a brief description.

### 2.1 Global variables and dictionaries

Most of the global variables and dictionaries are defined in the file `rtest_support.py` (see Appendix). Here `CASELIST` and `verbose` are defined for development convenience.

```

3b  <Define global variables and dictionaries 3b>≡ (3a)
      <rtest support 9>
      #from rtest_support import * #Not used for NoWeb version
      SHORTLIST=CASE['SAMP'][0],
      LONGLIST=CASE['SAMP']+CASE['SND']
      CASELIST=SHORTLIST
      verbose=1
      clean=0

```

## 2.2 Main

Function `main` calls the function `readlist` to get the element type list appropriate for each sample in `CASELIST`. For each result type, `main` calls `griz_it` to do the rest.

4a     $\langle \text{Main 4a} \rangle \equiv$  (3a)

```
def main():
    elem_type_list=[]
    for sample in CASELIST:
        os.chdir(RHOME+sample+"/")
        elem_type_list=read_info()
        print sample,": ",elem_type_list
        #ok=griz_it(sample,'DR',elem_type_list)
        ok=griz_it(sample,'TH',elem_type_list)
        #ok=griz_it(sample,'PR',elem_type_list)
    return 0
```

## 2.3 Parse the .info file

The `.info` file, created by `rtest_init`, is parsed to determine what element type(s) are available for testing.

4b     $\langle \text{Read info file 4b} \rangle \equiv$  (3a)

```
def read_info():
    "Parse ./info file to determine relevant element types"
    if not os.path.exists('.info'):
        print("ERROR: No info file found. Run rtest_init.py.")
        return -1
    info=open('.info','r+')
    raw = info.read()
    info.close()
    keylist=[]
    for key in IS.keys():
        etest = re.compile(IS[key][0])
        if etest.search(raw):
            keylist.append(key)
    return keylist
```

## 2.4 Griz loop

The function `griz_it` does the bulk of the work of the regression test. For each element type, `make_g_file` is called to generate a grizinit file. The appropriate GRIZ binary is then executed, generating result data files. Care is taken to make sure that the GRIZ process is finished before comparing result data files by checking for the file named "done" (created by GRIZ). The function `diff_it` is then called to compare the result data files.

```

5  <Griz it 5>≡ (3a)
    def griz_it(sample,result_type,elem_type_list):
        result_type_string=RS[result_type]
        os.chdir(RHOME+sample)
        for elem_type in elem_type_list:
            rprint("Testing "+elem_type+" "+result_type_string+" with "+sample)
            for i in range(len(TEST['bin'])):
                GRIZBIN=TEST['bin'][i] #selects the binary, Griz2 or Griz4
                DB=TEST['db'][i] #selects the database, Taurus or Mili
                DIR=TEST['dir'][i] #selects the appropriate directory for results
                if not os.path.exists(DIR):
                    # if appropriate directory does not exist, create it
                    if verbose:print "creating directory: "+DIR
                    os.mkdir(DIR)
                os.chdir(DIR) #change to appropriate directory
                ok=make_g_file(result_type, elem_type) #create grizinit file
                err=os.system(GRIZBIN+" -i "+DB) #run griz with new grizinit file
                os.chdir('..')
            #force all runs to finish before comparing results
            for i in [0,1,2]:
                DIR=TEST['dir'][i]
                if not os.path.exists(DIR):
                    print("ERROR: Directory "+DIR+"was not created")
                if not os.path.exists(DIR+'/done'):
                    print("waiting for "+DIR+"...")
                while not os.path.exists(DIR+'/done'):
                    pass
            for i in [0,1,2]:
                os.remove(TEST['dir'][i]+'done')
            ok=diff_it(result_type,elem_type)
        rprint("fini")
    return 1

```

## 2.5 Make grizinit file

Here the `grizinit` is created for each result type. Each `grizinit` file instructs GRIZ to run the sample case for all results associated with each element type available. A GRIZ history file of the name `result_type+elem_type+result+ ".his"` is created by GRIZ for the result data.

```
6  <Make grizinit file 6>≡ (3a)
    def make_g_file(result_type,elem_type):
        g=open('grizinit','w+')      #create grizinit file
        g.write(HEADER)
        if result_type=='DR':
            for result in DR[elem_type]:
                hisfile='DR'+elem_type+result+".his"
                g.write("show "+result+"\n")
                g.write("savtxt "+hisfile+"\n")
                g.write("telltmm\n")
                g.write("endtxt\n")
            g.write("savtxt done\n")
            g.write("endtxt\n")
        elif result_type=='PR':
            for result in PR[elem_type]:
                hisfile='PR'+elem_type+result+".his"
                g.write("show "+result+"\n")
                g.write("savtxt "+hisfile+"\n")
                g.write("telltmm\n")
                g.write("endtxt\n")
            g.write("savtxt done\n")
            g.write("endtxt\n")
        elif result_type=='TH':
            for result in TH[elem_type]:
                hisfile='TH'+elem_type+result+".his"
                g.write("select node 1\n")
                g.write("select beam 1\n")
                g.write("select shell 1\n")
                g.write("select brick 1\n")
                g.write("show "+result+"\n")
                g.write("timhis \n")
                g.write("outth "+hisfile+"\n")
            g.write("savtxt done\n")
            g.write("endtxt\n")
        else:
            g.write("quit\n")
            g.close()
            return 0
    g.write("quit\n")
    g.close()
```



## 2.7 Rprint

This routine sends messages to the log file and to the standard output if verbose. I might add a ranking to each message so that it is printed to standard output only if it is below the "verbosity" number.

```
8a  <Rprint 8a>≡ (3a)
    def rprint(text):
        "Reports information to report file and console if verbose"
        # r.write(text+"\n")
        if verbose: print(text)
        return 0
```

## 2.8 Onespace

This routine takes the GRIZ result files and converts it to one where the data is delimited by a single space. This is needed in cases where the versions have different formatting.

```
8b  <Onespace 8b>≡ (3a)
    def onespace(filename):
        tempfilename='tempfile'
        tempfile=open(tempfilename,'w+')
        datafile=open(filename,'r+')
        datin=datafile.readlines()
        for i in range(len(datin)):
            j=string.join(string.split(datin[i]))
            tempfile.write(j+"\n")
        datafile.close()
        tempfile.close()
        err=os.system('mv '+tempfilename+' '+filename)
        return 1
```

## 2.9 Exit

Normal system exits come from Main.

```
8c  <Exit code 8c>≡ (3a)
    if __name__ == '__main__':
        sys.exit(main())
```



### 3 Appendix: rtest\_support

This file defines all of the Python dictionaries used for the regression testing. Dictionaries have an unordered set of *key:value* pairs where the key can be a string.

9  $\langle rtest\_support\ 9 \rangle \equiv$  (3b)

```
import os,sys,re,string

verbose=1
RHOME="/grdev/regptest/"
GRIZ2=RHOME+"bin/griz2"
GRIZ4=RHOME+"bin/griz4"
TAURUS="../d3plot"
MILI="../m_plot"
H1="# This grizinit file was automatically generated\n"
H2="# by regrtest.py for regression testing of Griz4\n"
H3="# -Vic Castillo\n#\n"
HEADER=H1+H2+H3

TEST={}
TEST['bin']=GRIZ2,GRIZ4,GRIZ4
TEST['db']=TAURUS,TAURUS,MILI
TEST['dir']='G2T','G4T','G4M'

CASE={}
CASE['SAMP']='SAMP1','SAMP2','SAMP4','SAMP6','SAMP8'
CASE['SND']='SND1','SND2','SND3'

# Derived Results Dictionary
DR={}
drnd1='dispx','dispy','dispz','dispmag'
drnd2='velx','vely','velz','velmag'
drnd3='accx','accy','accz','accmag','pvmag'
DR['Nodal']=drnd1+drnd2+drnd3
drshr1='sx','sy','sz','sxy','syz','szz'
drshr2='press','seff','pdev1','pdev2','pdev3'
drshr3='maxshr','prin1','prin2','prin3'
DR['Share']=drshr1+drshr2+drshr3
drshl1='surf1','surf2','surf3','surf4','surf5','surf6'
drshl2='eff1','eff2','effmax'
DR['Shell']=drshl1+drshl2
drbrk1='ex','ey','ez','exy','eyz','ezx'
drbrk2='pdstrn1','pdstrn2','pdstrn3','pshrstr'
drbrk3='pstrn1','pstrn2','pstrn3','relvol','evol'
```

```
DR['Brick']=drbrk1+drbrk2+drbrk3

# Primal Results Dictionary
PR={}
PR['Nodal']='nodpos[ux]',
PR['Global']='ke',
PR['Mat']='matpe',
PR['Brick']='eeff',
PR['Shell']='eeff_mid',

# Time History Dictionary
# These are the results selected for the time histories
TH={}
TH['Nodal']='dispmag',
TH['Global']='ke',
TH['Mat']='matpe',
TH['Brick']='prin1',
TH['Shell']='effmax',

# Info String Dictionary
# These are the string patterns from the griz 'info' command.
# (Why does it not work without the commas?)
IS={}
IS['Nodal']='Nodes:',
IS['Brick']='Hex elements:',
IS['Shell']='Shell elements:',
# IS['Beam']='Beam elements:',

# These are the strings describing result types
RS={}
RS['DR']='Derived Results'
RS['PR']='Primitive Results'
RS['TH']='Time History Results'
```