LAS file QA-QC routines

The code provided in this notebook is used in the QA/QC review of LAS files. Files opened by these routines must be LAS 2.0 compliant. That is, the file must pass all checks performed by LAS Certify.

LAS
Log ASCII standard, "LAS" is a format for recording and exchanging digital well log information defined by the Canadian Well Logging Society. Most files are formatted using LAS version 2.0, for documentation see http://www.cwls.org/wp-content/uploads/2017/02/Las2_Update_Feb2017.pdf (http://www.cwls.org/wp-content/uploads/2017/02/Las2_Update_Feb2017.pdf). The most recent version is 3.0, see http://www.cwls.org/wp-content/uploads/2014/09/LAS_3_File_Structure.pdf (http://www.cwls.org/wp-content/uploads/2014/09/LAS_3_File_Structure.pdf). Any stratigraphic tops data sections potentially added to LAS files must be compliant with LAS 3.0.

LASIO
This notebook uses LASIO to read, potentially modify, and write LAS files. For documentation and information on the LASIO module see https://lasio.readthedocs.io/en/latest/ (https://lasio.readthedocs.io/en/latest/)

Installation: If Python and the Jupyter Notebook were installed using an Anaconda Python distribution:

1. From the system start menu, open the Anaconda group and run the Anaconda Prompt application.
2. At the prompt, enter "pip install lasio".
3. Close the command window when installation is complete.

Full installation instructions for LASIO are online at https://lasio.readthedocs.io/en/latest/installation.html.

Welly
Welly is a utility written in Python for handling various types of borehole data. LASIO is used for input of geophysical logs. In particular, some plot functionality is provided by Welly that can facilitate the QA/QC review of LAS files.

1. *Installation*: In the Anaconda Prompt window, enter "pip install welly"
2. *Documentation*: https://media.readthedocs.org/pdf/welly/latest/welly.pdf (https://media.readthedocs.org/pdf/welly/latest/welly.pdf)
3. *Examples and tutorials*: https://github.com/agile-geoscience/welly/tree/master/tutorial (https://github.com/agile-geoscience/welly/tree/master/tutorial)

See the last code block in this notebook for an example.

Basic Instructions
Start a session by running the code in steps 1) to 3). Repeat steps 4) through *nn)* for each LAS to be checked and verified. Additional steps using interactive plots are provided in the **Optional** section.

Use shift-enter to execute each code block. Some of the executable blocks will prompt you for input, like a path name to the folder where the LAS files are stored. Press enter after typing your response.

The example in this notebook uses an LAS file extracted directly from a Weatherford/Reeves/Precision Wireline data pack file. This LAS file and accompanying Jupyter Notebook file reside in the KGS active project shares \KYTypeLogs_2018\Software\ folder. This file share was mapped to the W:\ drive on the author's system. If that share is mapped otherwise, make sure you substitute the correct drive letter, or save a sample LAS file to a known folder.

1) Do imports and set up for plotting in notebook

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import lasio
%matplotlib inline
```

2) Define the plot function

In [2]:

```
def lasplot(las,**kwargs):
    """Plot specific curves from an LAS file

    Input:
        las -- an las file object as defined in the lasio module

    Key word arguments (kwargs):
        curves -- a list of mnemonics for two curves to plot, one per track, default is GR a
        lines -- a list of the attributes for the curves to be plotted (color and linestyle)
            default for left track is a solid black line
            default for right track is a solid blue line
        tops -- an optional list of tops to plot on the log [[measured_depth,'label'],[...],
        savefig -- name of an output image file for saving the plot, usually *.png (include
        depths -- a list of the top and bottom of an interval to plot, default is [0.0,99999
            when specified, if bottom depth="TD" then set bottom depth to maximum depth valu
        depthcurve -- mnemonic for the depth curve to use for plotting, default is 'DEPT'

            Note: in some cases, multiple depth curves are found in a merged LAS file export
            lasio adds a suffix (":1", ":2", etc) when a curve mnemonic is repeated. This is
            There should be only one depth curve.
    """
    curves=kwargs.setdefault('curves',['GR','DEN'])
    linetypes=kwargs.setdefault('lines',['k-','b-'])
    tracks=len(curves)
    tops=kwargs.setdefault('tops',[])
    savefile=kwargs.setdefault('savefig',"")
    depths=kwargs.setdefault('depths',[0.0,99999.0])
    depthcurve=kwargs.setdefault('depthcurve','DEPT')
    fig, axes = plt.subplots(nrows=1, ncols=tracks,sharey=True,squeeze=True,figsize=(10,12))
    title=None
    if "API" in las.well:
        title=las.well["API"].value
    elif "APIN" in las.well:
        title=las.well["APIN"].value
    else:
        title=las.well["WELL"].value
    lasdf=pd.DataFrame(las.data,columns=list(las.curvesdict.keys()))
    if depths[-1] in ["TD","td"]:
        depths[-1]=lasdf[depthcurve].max()
    lasdf=lasdf[(lasdf[depthcurve]>=depths[0]) & (lasdf[depthcurve]<=depths[-1])]
    plt.suptitle(title,weight="bold",size=24,y=0.925)
    plt.ylim(lasdf[depthcurve].max(),lasdf[depthcurve].min())
    for i in range(len(curves)):
        current=curves[i]
        if type(current)==type(str()): # plot only one curve
            axes[i].plot(lasdf[current],lasdf[depthcurve],linetypes[i],lw=2)
            axes[i].set_xlabel(current,weight="bold",size=18)
        else:
            for lc in range(len(current)):
                axes[i].plot(lasdf[current[lc]],lasdf[depthcurve],'k',lw=2)
                axes[i].set_xlabel(current[lc],weight="bold",size=18)
        xmn,xmx=axes[i].get_xlim()
        if len(tops)>0:
            for top in tops:
                if top[0]>=depths[0] and top[0]<=depths[-1]:
                    axes[i].plot([xmn,xmx],[top[0],top[0]],'k:',lw=2)
                    axes[i].text(xmn,top[0]+15,top[1],weight="bold",size=16)
        axes[i].grid(True)
    # axes[2].legend()
    if len(savefile)>0:
        fig.savefig(savefile,dpi=300)
```

3) Get the default path name for the folder with LAS files

Enter the path name for the folder where the LAS files are stored. Be sure to include the final "" in the path name. If the folder is on a networked Google Drive, the path name is case sensitive.

Example -- c:\users\nuttall\desktop

In [3]:
```
laspath=input("Enter path: ")
```

```
Enter path: w:\KYTypeLogs_2018\Software\
```

Start here for each new LAS file to examine

4) Enter the name of the file to open

In [4]:
```
while True:
    fname=input("Enter file name: ")
    lasfile=laspath+fname
    try:
        print("Trying to open: {}".format(lasfile))
        las=lasio.read(lasfile)
        print("Success!")
        break
    except FileNotFoundError:
        print('File not found, try again')
checkDEPT=False # set up for a later check of the depth track
```

```
Enter file name: begley 13 1up_001.las
Trying to open: w:\KYTypeLogs_2018\Software\begley 13 1up_001.las
Success!
```

List the ~Well section parameter items

In [5]:
```
list(las.well) # check the header information in the file
```

Out[5]:
```
[HeaderItem(mnemonic=STRT, unit=F, value=-5.249, descr=Start),
 HeaderItem(mnemonic=STOP, unit=F, value=4485.751, descr=Stop),
 HeaderItem(mnemonic=STEP, unit=F, value=0.5, descr=Step increment),
 HeaderItem(mnemonic=NULL, unit=, value=-999.25, descr=Null value),
 HeaderItem(mnemonic=COMP, unit=, value=DAUGHERTY PETROLEUM, descr=Company),
 HeaderItem(mnemonic=WELL, unit=, value=BEGLEY PROPERTIES-ORR TRUST # 13, des...),
 HeaderItem(mnemonic=FLD, unit=, value=WALLINS CREEK QUADRANGLE, descr=Field),
 HeaderItem(mnemonic=PROV, unit=, value=BELL, descr=Province),
 HeaderItem(mnemonic=CTRY, unit=, value=United States, descr=State / Country),
 HeaderItem(mnemonic=LOC, unit=, value=1521 FNL  0228 FWL, descr=Location),
 HeaderItem(mnemonic=LOC2, unit=, value=, descr=Location 2),
 HeaderItem(mnemonic=SRVC, unit=, value=Weatherford, descr=Service company),
 HeaderItem(mnemonic=UWI, unit=, value=, descr=Unique Well ID),
 HeaderItem(mnemonic=API, unit=, value=, descr=API Number),
 HeaderItem(mnemonic=LIC, unit=, value=95382, descr=Licence Number),
 HeaderItem(mnemonic=DATE, unit=, value=20-Nov-2003, descr=Logging Date)]
```

List the ~Curves section parameter items

In [6]:
```
list(las.curves) # show the curves found in the file
```

Out[6]:
```
[CurveItem(mnemonic=DEPT, unit=F, value=00 001 00 00, descr=Logged depth, original_mnemon
ic=DEPT, data.shape=(8983,)),
 CurveItem(mnemonic=AVOL, unit=F3, value=70 000 00 00, descr=Annular Volume, original_mne
monic=AVOL, data.shape=(8983,)),
 CurveItem(mnemonic=HVOL, unit=F3, value=70 000 00 00, descr=Hole Volume, original_mnemon
ic=HVOL, data.shape=(8983,)),
 CurveItem(mnemonic=MTXD, unit=G/C3, value=94 000 00 00, descr=Matrix density, original_m
nemonic=MTXD, data.shape=(8983,)),
 CurveItem(mnemonic=BHTD, unit=----, value=80 660 11 00, descr=Differential Temperature,
original_mnemonic=BHTD, data.shape=(8983,)),
 CurveItem(mnemonic=BHTF, unit=DEGF, value=80 660 00 00, descr=Borehole Temperature, orig
inal_mnemonic=BHTF, data.shape=(8983,)),
 CurveItem(mnemonic=DPOR, unit=PERC, value=45 890 13 00, descr=Base Density Porosity, ori
ginal_mnemonic=DPOR, data.shape=(8983,)),
 CurveItem(mnemonic=PDPE, unit=B/E, value=45 358 01 00, descr=PE, original_mnemonic=PDPE,
data.shape=(8983,)),
 CurveItem(mnemonic=DEN, unit=G/C3, value=42 350 01 00, descr=Compensated Density, origin
al_mnemonic=DEN, data.shape=(8983,)),
 CurveItem(mnemonic=DCOR, unit=G/C3, value=42 356 01 00, descr=Density Correction, origin
al_mnemonic=DCOR, data.shape=(8983,)),
 CurveItem(mnemonic=CLDC, unit=IN, value=45 280 01 00, descr=Density Caliper, original_mn
emonic=CLDC, data.shape=(8983,)),
 CurveItem(mnemonic=NPRL, unit=PERC, value=42 890 01 00, descr=Limestone Neutron Por., or
iginal_mnemonic=NPRL, data.shape=(8983,)),
 CurveItem(mnemonic=GRGC, unit=GAPI, value=30 310 01 00, descr=Gamma Ray, original_mnemon
ic=GRGC, data.shape=(8983,)),
 CurveItem(mnemonic=SMTU, unit=LB, value=30 635 00 00, descr=DST Uphole Tension, original
_mnemonic=SMTU, data.shape=(8983,)),
 CurveItem(mnemonic=GDLO, unit=----, value=68 545 00 00, descr=Lower Gas Detector, origin
al_mnemonic=GDLO, data.shape=(8983,)),
 CurveItem(mnemonic=GDUP, unit=----, value=68 545 00 00, descr=Upper Gas Detector, origin
al_mnemonic=GDUP, data.shape=(8983,)),
 CurveItem(mnemonic=BIT, unit=IN, value=70 282 00 00, descr=Bit size, original_mnemonic=B
IT, data.shape=(8983,))]
```

Does the curve data make sense?

*You might also want to look at the data using **welly**, see below*

In [7]:
```
lasdf=pd.DataFrame(las.data,columns=list(las.curvesdict.keys()))
lasdf.describe()
```

Out[7]:

	DEPT	AVOL	HVOL	MTXD	BHTD	BHTF	DPOR	PDI
count	8983.000000	3065.000000	3065.000000	3.072000e+03	3072.000000	3072.000000	3069.000000	3067.0000
mean	2240.251277	0.001088	0.002114	2.710000e+00	-0.001808	80.361095	2.763470	3.5135
std	1296.656054	0.000246	0.000246	1.865478e-14	0.023520	2.180547	4.973064	1.0001
min	-5.249000	0.000943	0.001969	2.710000e+00	-0.089311	76.707792	-9.376981	1.8467
25%	1117.501000	0.001050	0.002076	2.710000e+00	-0.002356	79.106934	-0.415369	2.9891
50%	2240.251000	0.001057	0.002083	2.710000e+00	-0.001283	80.426469	1.204566	3.3397
75%	3363.001000	0.001063	0.002089	2.710000e+00	-0.000366	81.357540	5.955962	3.8995
max	4485.751000	0.003116	0.004142	2.710000e+00	0.672664	90.591470	50.255394	11.6823

Check the depth curve

Look for oddities in the depth curve. Normally, the STEP value is 0.5 (can differ). That means that usually the depth values are like a sequence 0.0, 0.5, 1.0, 1.5, ...

In [8]:
```
# let d1 be the depth data from the file
d1=np.array(lasdf["DEPT"])
# let d2 be what the depths should be based on the STRT, STOP, and STEP values
d2=np.arange(las.well['STRT'].value,las.well['STOP'].value+las.well['STEP'].value,las.well['
diff=np.abs(d1-d2) # find the absolute value of the differences between the actual (d1) and
if diff.sum()>0.0:
    DEPTok=False
    print("Depth data do not match intervals set by STRT, STOP, and STEP values.")
else:
    DEPTok=True
    print("Depth data are OK.")
checkDEPT=True
```

```
Depth data do not match intervals set by STRT, STOP, and STEP values.
```

If the depth curve doesn't match the STRT, STOP, and STEP specification, replace it

This is often due to a rounding error where information has been recorded/reported at negative depths (that is, above the reference elevation). While mostly these data are missing (-999.250), it can cause LAS Certify to fail.

In [9]:
```
if checkDEPT: # do nothing if haven't checked the DEPT
    if not DEPTok: # do nothing if there isn't anything wrong with the DEPT curve
        las.curves['DEPT'].data=d2
        print('Replaced the DEPT curve, make sure to write a new LAS file')
```

```
Replaced the DEPT curve, make sure to write a new LAS file
```

Plot some curves

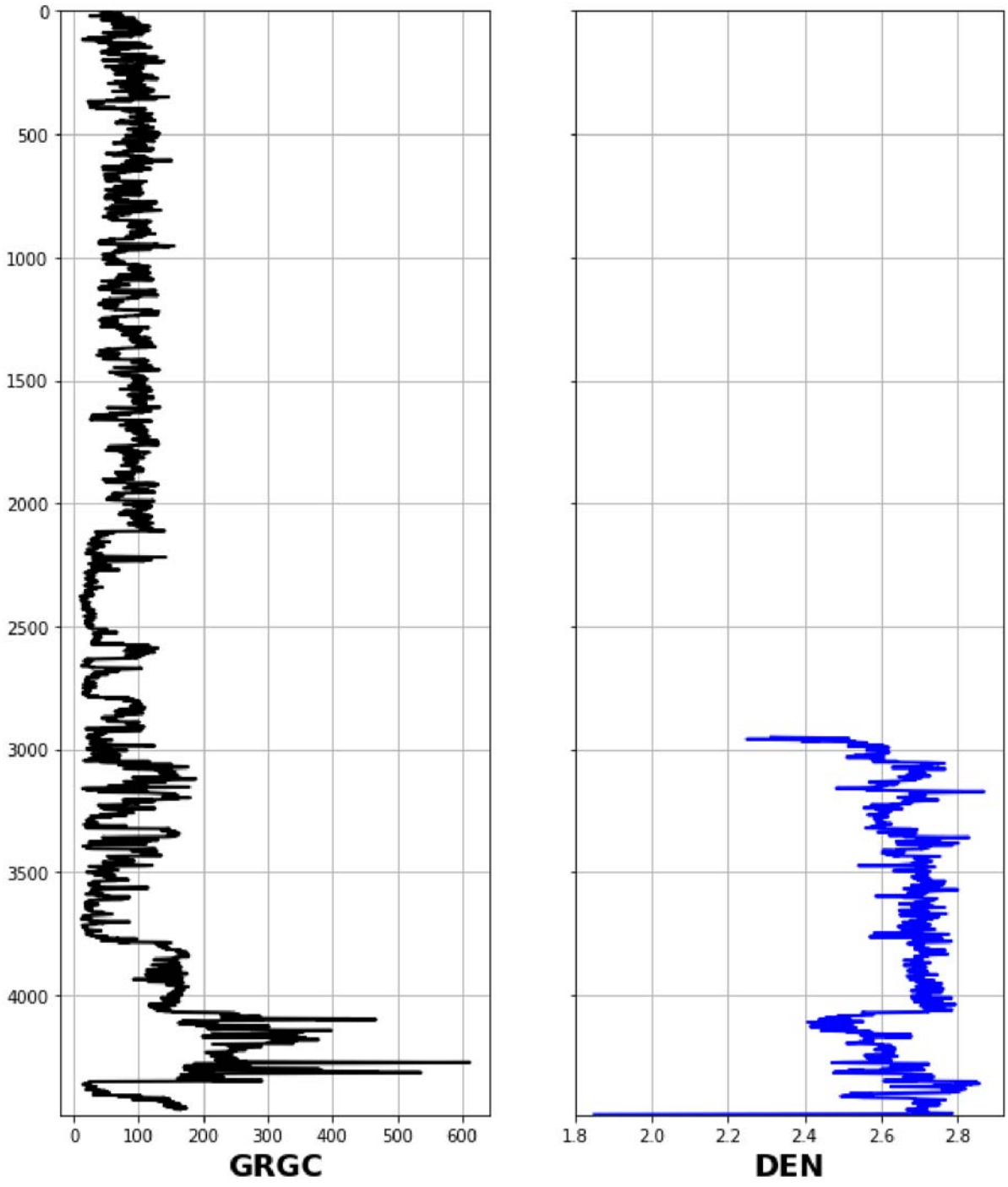
When prompted, type in the name of a curve and press enter. It doesn't matter whether the curve is entered using upper or lower case. Only two curves can be plotted.

In [10]:
```
# get the curves to plot
allcurves=list(las.curvesdict.keys())
ncurves=2 # can change this later if lasplot() function is updated
curves=[]
for i in range(ncurves):
    while True:
        trace=input('Enter a curve name: ').upper()
        if trace in allcurves:
            print('"{}" OK'.format(trace))
            curves.append(trace)
            break
        else:
            print('"{}" not found'.format(trace))
            print("Available: {}".format(list(las.curvesdict)))
print("\nWill plot:")
for trace in curves:
    print("{}: {}, min={}, max={}".format(trace,las.curves[trace].unit,lasdf[trace].min(),la
```

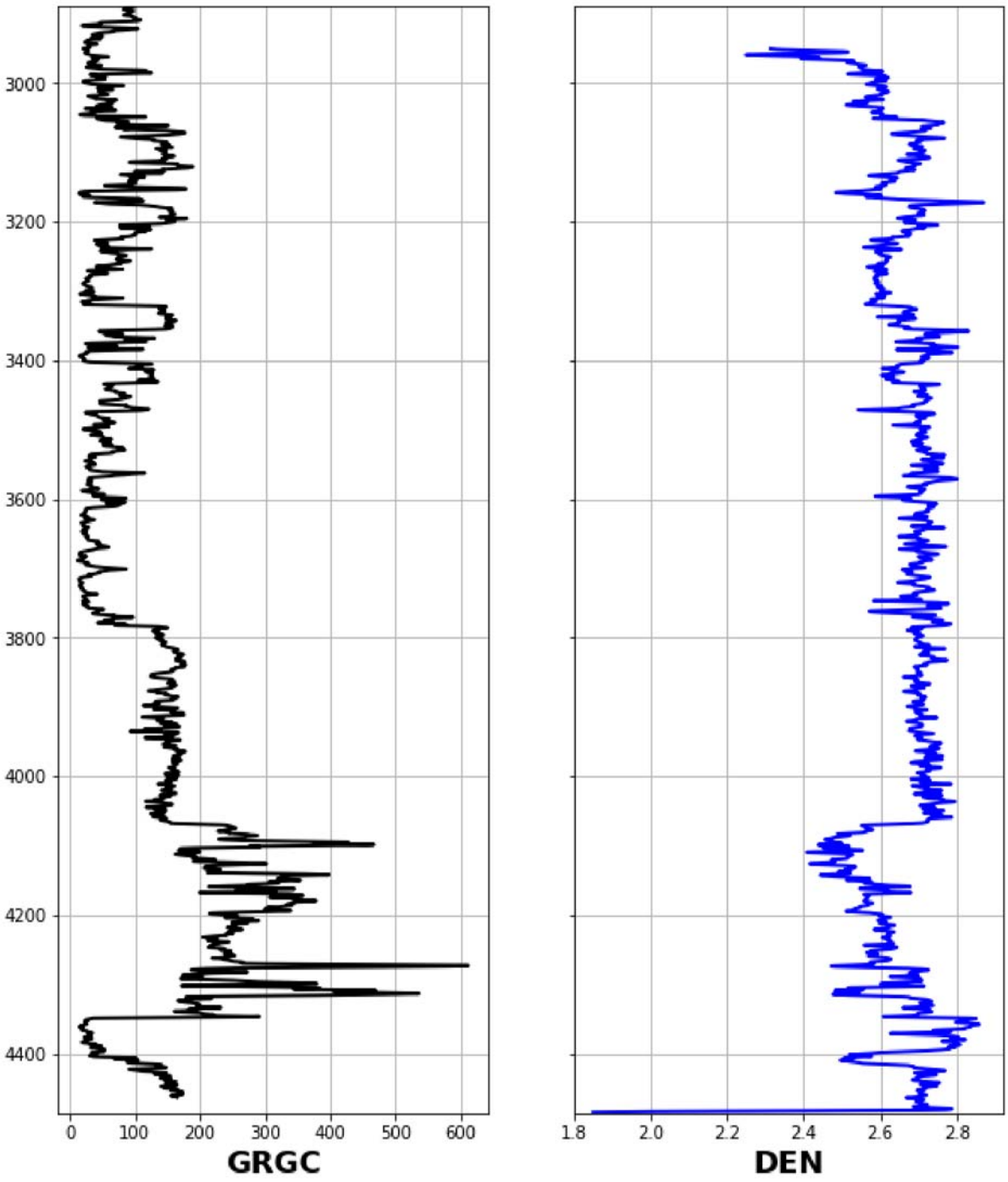
```
Enter a curve name: grgc
"GRGC" OK
Enter a curve name: den
"DEN" OK

Will plot:
GRGC: GAPI, min=9.988543, max=611.795096
DEN: G/C3, min=1.850633, max=2.870346
```

In [11]:
```
# Now plot the curves
"""
To plot between selected depths:
    lasplot(las,curves=curves,depths=[top,bottom])
    bottom depth can be "TD" -- depths=[2500,"TD"]
"""
lasplot(las,curves=curves)
```



In [12]:
```
# per the example, plot the curves from 2890 ft to TD
lasplot(las,curves=curves,depths=[2890,'TD'])
```



If a curve must be deleted:

Run this cell once for each curve to be deleted from the LAS
When prompted, type in the name of the curve to be deleted and press enter

In [13]:
```
# get a curve name
dodel=True
while True:
    trace=input('Enter a curve name (or NONE to skip): ').upper()
    if trace in allcurves:
        print('"{}" OK'.format(trace))
        break
    elif trace=="NONE":
        dodel=False
        break
    else:
        print('"{}" not found'.format(trace))
print('"{}" will be deleted'.format(trace))
if dodel:
    las.delete_curve(trace)
else:
    print('Nothing deleted')
```

```
Enter a curve name (or NONE to skip): NONE
"NONE" will be deleted
Nothing deleted
```

Optional: Make some comparisons between curves to check for duplication

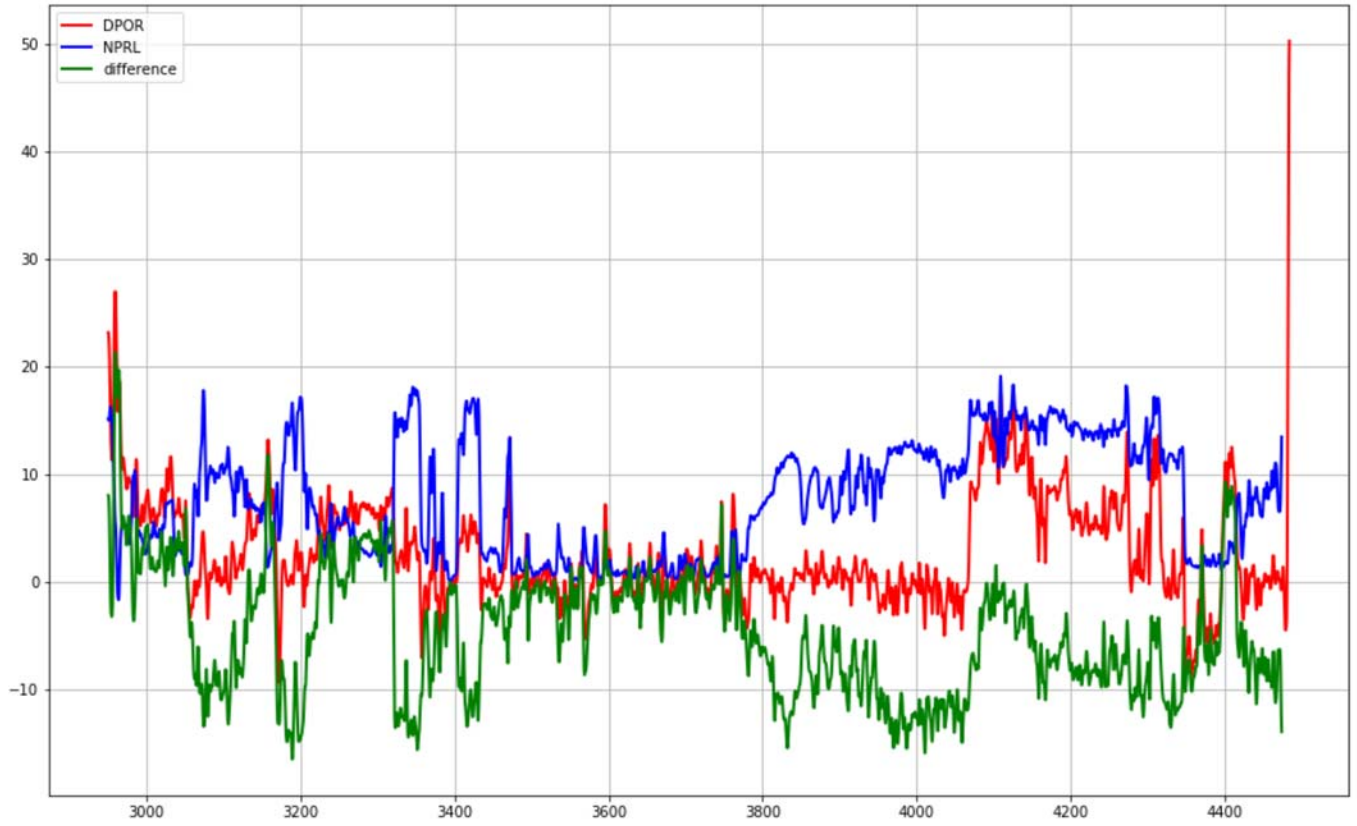
Convert the data to a pandas data frame to make calculations easier

In [14]:
```
logdf=pd.DataFrame(las.data,columns=list(las.curvesdict.keys()))
```

Up to 3 curves can be plotted

- curve1 will be **red**
- curve2 will be **blue**
- curve3 if==None, then plot the difference between curve1 and curve2 in **green**, otherwise plot curve3

In [15]:
```
curve1="DPOR"
curve2="NPRL"
curve3=None
depth="DEPT" # default depth curve (x-axis)
lasdf['DIFF']=lasdf[curve1]-lasdf[curve2] # calculate the difference between curve1 and curv
plt.figure(figsize=(16,10))
plt.plot(lasdf[depth],lasdf[curve1],'r-',label=curve1,lw=2)
plt.plot(lasdf[depth],lasdf[curve2],'b-',label=curve2,lw=2)
if curve3==None:
    plt.plot(lasdf[depth],lasdf['DIFF'],'g-',label="difference",lw=2)
else:
    plt.plot(lasdf[depth],lasdf[curve3],'g-',label=curve3,lw=2)
plt.legend()
plt.grid(axis='both')
plt.show()
```



For this well, the density porosity (DPOR) and neutron porosity (NPRL) differ. The difference (green) trace

Last) Write a new LAS file (required if curves were changed or deleted)

The new file will not overwrite the original input LAS file. The new file will be written to the same folder, but will have "new" inserted into the filename. For example, an updated las file for Rnnnnnnnn.las will be written to Rnnnnnnnn_new.las

In [16]:
```
# generate a new file name so the old file won't be overwritten
fn,ft=fname.split('.')
newfile=laspath+fn+"_new."+ft
print("Creating -- {}".format(newfile))
las.write(newfile,STEP=0.500)
print("Success!")
```

```
Creating -- w:\KYTypeLogs_2018\Software\begley 13 1up_001_new.las
Success!
```

Welly example

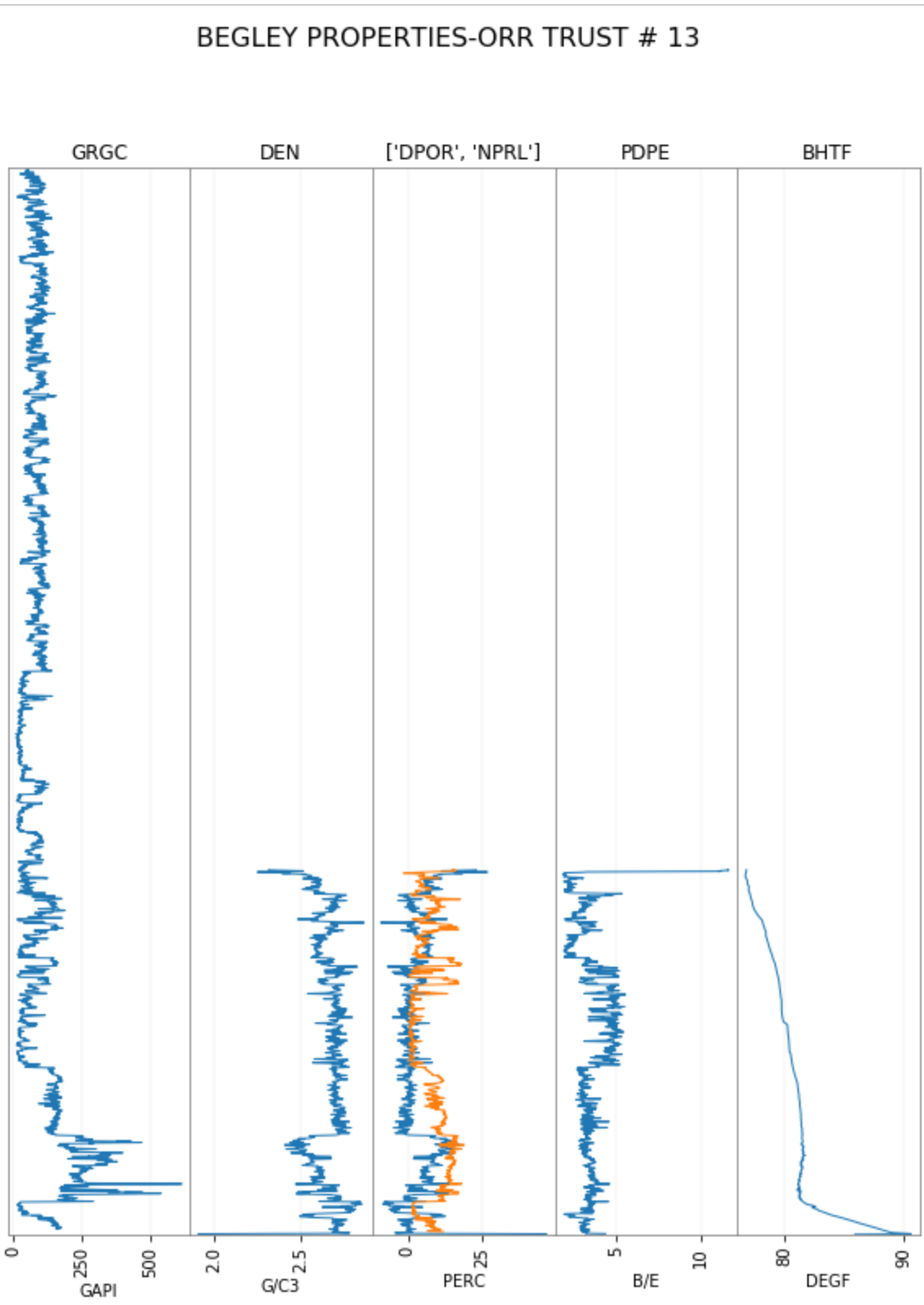
Mainly interested in investigating the scale and range of multiple log curves at the same time. See the documentation and tutorials for more capabilities.

In [17]:
```
import welly
from welly import Well
```

For this example, use the same file opened earlier with LASIO

In [18]:
```
w=Well.from_las(laspath+fname)
```

In [19]:
```
tracks=['GRGC','DEN',['DPOR','NPRL'],'PDPE','BHTF']
w.plot(tracks=tracks,extents='curves')
```



In [ ]: