Time-Depth Curve & Velocity Generation from VSP Data using Python

Theory/ Method

As an initial approach and to get acquainted with python in geophysics, first an algorithm was written to read the segy traces. The below code generates amplitude values for each time sample.

```
        # code for reading seismic data #

import sys
import os
import struct

def getEBCDIC():
   inf=open("/root/Desktop/GMT01_vz-
OCT.sgy", "rb")
   ASCII_txt = " "
   for c in range(0,40):
         EBCDIC=inf.read(80)
                        # Format to ascii with extra
new line at end for alligned view
         ASCII_txt += codecs.decode(EBCDIC,
"cp500") + "\n"
   return ASCII_txt[0:3199]

print(getEBCDIC())

def ibm2ieee(ibm):
  if ibm==0:
    return 0.0
  sign = ibm >> 31 & 0x01
  exponent = ibm >> 24 & 0x7f
  mantissa = ibm & 0x00ffffff
  mantissa = (mantissa * 1.0) / pow(2, 24)
  ieee = (1 - 2 * sign) * mantissa * pow(16,
exponent -64)
  return ieee
def segyReport():
   bps = [0,4,4,2,4,4] # bytes per sample for
codes 0,1,2,3,4 at byte 25

fs=os.path.getsize("/root/Desktop/GMT01_vz-
OCT.sgy") # file size
   ins=open("/root/Desktop/GMT01_vz-
OCT.sgy", "rb")
   out=open("/root/Desktop/test1.txt","w")

   ehdr=3200          # reel header size
   bhdr=400           # binary header size
   ins.seek(ehdr+16,0)
   si=struct.unpack('>H', ins.read(2))[0] #
number of samples
   si=si/1000
   ins.seek(ehdr+20,0)
   ns=struct.unpack('>H', ins.read(2))[0]#
number of samples
   ins.seek(ehdr+24,0)
   sf=struct.unpack('>H', ins.read(2))[0] #
sample format 16/24/32 kind
   bytes=bps[sf]      # number of bytes per
samples
   rl=(ns*bytes)+240
   thdr=rl
   traces=(fs-3600)/thdr

   print("\n Sample Interval: ",si,"ms","\n No of
Samples: ",ns,"\n Format: ",sf,"\n Traces:
",traces,"\n Record Length: ",rl,"\n Filesize:
",fs)

   tr=3600
   trno=1
   lsamp1 = [[0] * n for i in range(fs)]
   while tr < fs:
      ins.seek(tr+188,0)
      Iline=struct.unpack('>i', ins.read(4))[0]
      ins.seek(tr+192,0)
      Xline=struct.unpack('>i', ins.read(4))[0]
      print(Iline,Xline,file=out)

      sr=240
      tr1=1
      while sr < rl:
         ins.seek(tr+sr,0)
         lsamp=struct.unpack('>L', ins.read(4))[0]
#         lsamp1[trno][tr1] =
int(ibm2ieee(lsamp))
         lsamp1[trno][tr1]=int(ibm2ieee(lsamp))

print(trno,tr1,int(ibm2ieee(lsamp)),file=out)
         tr1=tr1+1
         sr=sr+4
      trno=trno+1
      tr=tr+rl

segyReport()
```

Once amplitude values for each time samples were generated, next approach was for finding the time of first energy outbreak.

For first break picking, we took help of open source software available in GitHub called AIpycker. Using this software, first break picks were generated.
As these pick values are usually in text format in AIpycker, they were copied into Excel sheet.

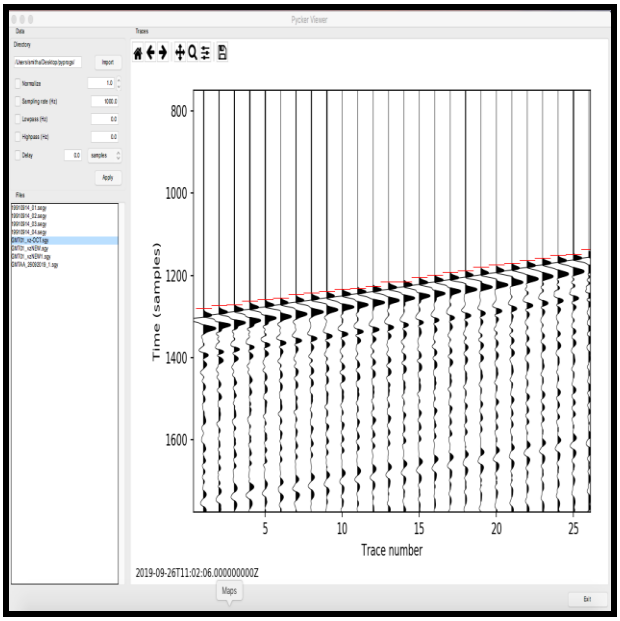

Figure 1: First break picking

In another excel sheet, shot hole depth information for respective depths were included. By using python codes, these two excel sheets were linked with the coordinate information of each shot point available in observer's sheet that comes from field. This entire exercise was done to avoid any type of time shifts that comes in the data due to variation in shot depth values or due to variation in offset positions of each shot point from well head. Most of the time, shots are taken at various nearby positions so when we approximate offset distance to a single value and assign them commonly to all shot points, it brings time shifts in the data.

Time-Depth Curve & Velocity Generation from VSP Data using Python

To generate interval velocity and time–depth relationship, an algorithm was written and successfully executed which is given below.

```
# CODE FOR TIME-DEPTH RELATION #

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from pandas import ExcelWriter

df1=pd.read_excel('/root/Desktop/pic_TD/fina
l_proj8.xlsx',sheet_name='Sheet8')
df2=pd.read_excel('/root/Desktop/pic_TD/fina
l_proj8.xlsx',sheet_name='Sheet2')
df7=pd.read_excel('/root/Desktop/pic_TD/fina
l_proj8.xlsx',sheet_name='Sheet3')
writer =
ExcelWriter('/root/Desktop/pic_TD/final_proj1
.xlsx')
results=df1.merge(df2,on='SHOTLOC')
results1=results.merge(df7,on='MD')
results1.to_excel(writer,'Sheet1')
writer.save()
df4=pd.read_excel('/root/Desktop/pic_TD/fina
l_proj1.xlsx',sheet_name='Sheet1')
df4.to_excel('/root/Desktop/pic_TD-
test/final_proj1.xlsx')
df4["SRO"]= np.sqrt(((df4["RCX"]-
df4["b"])*(df4["RCX"]-df4["b"]))+((df4["RCY"]-
df4["c"])*(df4["RCY"]-df4["c"])))
df4
SRE=6.52
WRE=14.14
df4["Ts"]=((df4["SD"]-SRE)/1850)
df4["TVDSD"]=df4["MD"]-WRE
df4["TVDSS"]=df4["TVDSD"]+SRE-df4["SD"]
df4["VT"]=df4["Tt"]*np.cos(np.arctan(df4["SR
O"]/df4["TVDSS"]))
#df4["VT"]=df4["VT"].round(6)
df4["Tv"]=df4["VT"]+df4["Ts"]
#df4["Tv"]=df4["Tv"].round(6)
df4["TWT"]=df4["Tv"]*2
#df4["TWT"]=df4["TWT"].round(6)
df4.sort_values(["MD"], axis=0,
ascending=True, inplace=True)
df4.drop(df4.columns[df4.columns.str.contains
('unnamed',case =False)],axis = 1,inplace=True)
df4.to_excel('/root/Desktop/pic_TD-test/final-
sorted.xlsx')
df4=pd.read_excel('/root/Desktop/pic_TD-
test/final-sorted.xlsx',sheet_name='Sheet1')
df4.to_excel('/root/Desktop/pic_TD-test/final-
sorted.xlsx')
df4["TWT_SM"]=df4["TWT"].rolling(window=5
).mean()
df4["TWT_SM1"]= df4["TWT_SM"].shift(-2)
df4.loc[0,'TWT_SM1'] =(df4.loc[0,'TWT'])
df4.loc[1,'TWT_SM1'] =(df4.loc[1,'TWT'])
df4.loc[159,'TWT_SM1'] =(df4.loc[159,'TWT'])
df4.loc[160,'TWT_SM1'] =(df4.loc[160,'TWT'])
#df4["TWT_SM1"] =
df4["TWT_SM1"].round(6)
df4.drop(df4.columns[df4.columns.str.contains
('unnamed',case =False)],axis = 1,inplace=True)
df4["AV.VEL"]=df4["TVDSD"]/df4["TV"]
df4["AV.VEL"]=df4["AV.VEL"].round(0)
df4["AV.VEL1"]=df4["AV.VEL"].rolling(window=5
).mean()
df4["AV.VEL1"]= df4["AV.VEL1"].shift(-2)
df4.loc[0,'AV.VEL1'] =(df4.loc[0,'AV.VEL'])
df4.loc[1,'AV.VEL1'] =(df4.loc[1,'AV.VEL'])
df4.loc[159,'AV.VEL1'] =(df4.loc[159,'AV.VEL'])
df4.loc[160,'AV.VEL1'] =(df4.loc[160,'AV.VEL'])
for i in range(1, len(df4)):
    df4.loc[0,'INT.VEL']
=(df4.loc[0,'TVDSD']/df4.loc[0,'TWT_SM1'])*2
    df4.loc[i,'INT.VEL'] = ((df4.loc[i,'TVDSD'] -
df4.loc[i-1,'TVDSD'])/(df4.loc[i,'TWT_SM1'] -
df4.loc[i-1,'TWT_SM1']))*2
for i in range(1, len(df4)):
    df4.loc[0,'V2'] = df4.loc[0,'AV.VEL1']**2
    df4.loc[i,'V2'] = df4.loc[i,'INT.VEL']**2
    df4.loc[0,'dT'] = df4.loc[0,'TWT_SM1']/2
    df4.loc[i,'dT'] = (df4.loc[i,'TWT_SM1'] -
df4.loc[i-1,'TWT_SM1'])/2
    df4.loc[0,'NUM'] =
df4.loc[0,'V2']*df4.loc[0,'dT']
    df4.loc[i,'NUM'] =
df4.loc[i,'V2']*df4.loc[i,'dT']
Nsum = 0
Tsum = 0
for i in range(0, len(df4)):
    Nsum = Nsum + df4.loc[i,'NUM']
    Tsum = Tsum + df4.loc[i,'dT']
    df4.loc[i,'RMS.VEL'] = (Nsum/Tsum)**0.5
    df4.loc[i,'RMS.VEL'] =
df4.loc[i,'RMS.VEL'].round(0)
df4["INT.VEL"] = df4["INT.VEL"].round(0)
df4["AV.VEL1"] = df4["AV.VEL1"].round(0)
df4.sort_values(["MD"], axis=0,
ascending=True, inplace=True)
df4.to_excel('/root/Desktop/pic_TD-
test/final_proj1.xlsx')
df4.plot(x='MD',y='TWT')
plt.show()
df4.plot(x='MD',y='INT.VEL',color='red')
plt.xticks(rotation=90)
plt.yticks(rotation=90)
fig = plt.gcf()
fig.set_size_inches(8,3)
plt.ylabel('INTERVAL VELOCITY ', fontsize = '12')
plt.xlabel('DEPTH '+ '$[m]$', fontsize = '12')
plt.locator_params(axis='y',nbins=6)
plt.show()
df4.plot(x='MD',y='AV.VEL',color='blue')
plt.xticks(rotation=90)
plt.yticks(rotation=90)
fig = plt.gcf()
fig.set_size_inches(8,3)
plt.ylabel('AVERAGE VELOCITY ', fontsize = '12')
plt.xlabel('DEPTH '+ '$[m]$', fontsize = '12')
plt.show()
df4.plot(x='MD',y='RMS.VEL',color='green')
plt.xticks(rotation=90)
plt.yticks(rotation=90)
fig = plt.gcf()
fig.set_size_inches(8,3)
plt.ylabel('RMS VELOCITY ', fontsize = '12')
plt.xlabel('DEPTH '+ '$[m]$', fontsize = '12')
plt.show()
```

The velocity list generated is given below:

MD	TVD	SD	SHOTLOC	Tt	RCX	RCY	SCX	SCY	SRO	Ts	TVDSD	VT	TWT	AV.VEL	AV.VEL1	INT.VEL	RMS.VEL
1020	1020	33	4	0.4655	575999	1838631	576041	1838588	60.10824	0.014314	1005.86	0.4646	0.9575	2100	2101	2533	2120
1040	1040	33	4	0.473	575999	1838631	576041	1838588	60.10824	0.014314	1025.86	0.4721	0.9724	2109	2110	2685	2129
1060	1060	33	4	0.4815	575999	1838631	576041	1838588	60.10824	0.014314	1045.86	0.4807	0.9872	2113	2119	2713	2139
1080	1080	30.5	3	0.4875	575999	1838631	576038	1838591	55.86591	0.012962	1065.86	0.4868	1.0011	2133	2129	2869	2151
1100	1100	30.5	3	0.4955	575999	1838631	576038	1838591	55.86591	0.012962	1085.86	0.4948	1.0145	2138	2140	2998	2164
1120	1120	30.5	3	0.5015	575999	1838631	576038	1838591	55.86591	0.012962	1105.86	0.5008	1.0275	2152	2152	3068	2178
1140	1140	30.5	3	0.5075	575999	1838631	576038	1838591	55.86591	0.012962	1125.86	0.5068	1.0418	2166	2161	2791	2188
1160	1160	33	4	0.514	575999	1838631	576041	1838588	60.10824	0.014314	1145.86	0.5133	1.0557	2172	2170	2871	2198
1180	1180	33	4	0.522	575999	1838631	576041	1838588	60.10824	0.014314	1165.86	0.5213	1.0701	2177	2179	2791	2207
1200	1200	33	4	0.529	575999	1838631	576041	1838588	60.10824	0.014314	1185.86	0.5283	1.0846	2185	2186	2752	2215
1220	1220	33	4	0.536	575999	1838631	576041	1838588	60.10824	0.014314	1205.86	0.5353	1.0986	2194	2195	2850	2225
1240	1240	31	3	0.5435	575999	1838631	576038	1838591	55.86591	0.013232	1225.86	0.5429	1.1123	2204	2204	2934	2235
1260	1260	31	3	0.55	575999	1838631	576038	1838591	55.86591	0.013232	1245.86	0.5494	1.1255	2214	2214	3023	2246
1280	1280	31	3	0.557	575999	1838631	576038	1838591	55.86591	0.013232	1265.86	0.5564	1.1387	2222	2223	3024	2256
1300	1300	31	3	0.563	575999	1838631	576038	1838591	55.86591	0.013232	1285.86	0.5624	1.1522	2234	2232	2980	2266
1320	1320	33	4	0.569	575999	1838631	576041	1838588	60.10824	0.014314	1305.86	0.5684	1.1653	2241	2241	3044	2276
1340	1340	33	4	0.576	575999	1838631	576041	1838588	60.10824	0.014314	1325.86	0.5754	1.1777	2248	2251	3220	2288
1360	1360	33	4	0.5818	575999	1838631	576041	1838588	60.10824	0.014314	1345.86	0.5812	1.1900	2260	2262	3262	2300
1380	1380	33	4	0.587	575999	1838631	576041	1838588	60.10824	0.014314	1365.86	0.5864	1.2024	2274	2272	3220	2312
1400	1400	31	3	0.5936	575999	1838631	576038	1838591	55.86591	0.013232	1385.86	0.5931	1.2143	2286	2282	3350	2324
1420	1420	31	3	0.601	575999	1838631	576038	1838591	55.86591	0.013232	1405.86	0.6005	1.2260	2291	2293	3419	2337
1440	1440	31	3	0.6068	575999	1838631	576038	1838591	55.86591	0.013232	1425.86	0.6063	1.2379	2301	2304	3385	2349
1460	1460	31	3	0.612	575999	1838631	576038	1838591	55.86591	0.013232	1445.86	0.6115	1.2496	2314	2314	3396	2361
1480	1480	31	3	0.6175	575999	1838631	576038	1838591	55.86591	0.013232	1465.86	0.6170	1.2613	2326	2324	3443	2374

Figure 2: Velocity listing

The time value for each associated depth is shown in Fig3. Various velocity curves generated is given in Fig4.

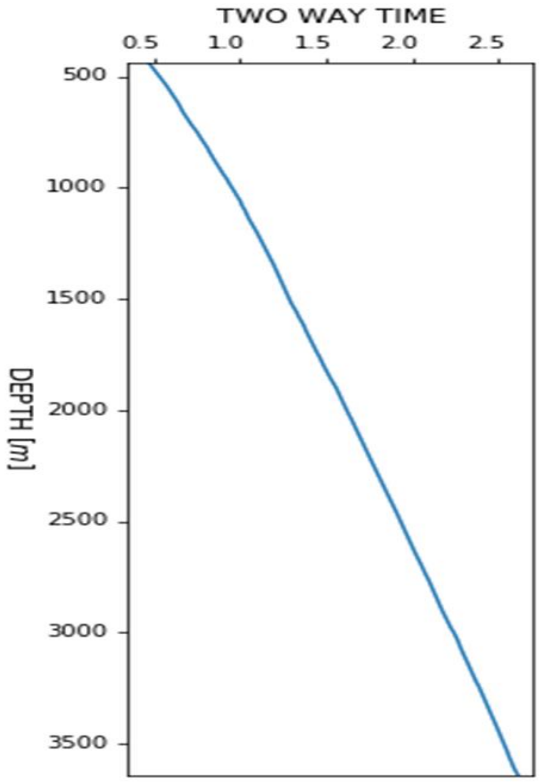

Figure 3: Time-Depth curve

Time-Depth Curve & Velocity Generation from VSP Data using Python

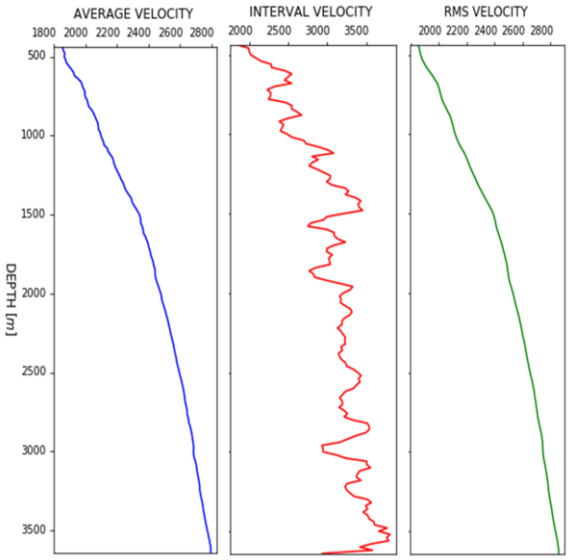

Figure 4: Velocity curves

A comparison was made with the interval velocity generated from available software and the one generated by us.

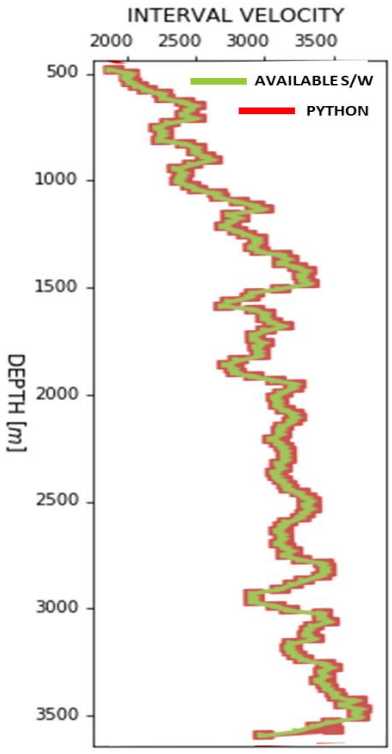

Figure 5: Comparison of Velocities

Conclusion

Time-Depth curve and Velocity information are the main outputs in any VSP survey. Albeit many software suits are available for VSP data processing in the industry, a special need exists to use Python for various reasons explained in the foregoing pages. An attempt has been made to generate T-D curve and velocity information using Python and the outputs are compared with that of existing software. It is found that the results are perfectly comparable. The way forward is to continue the work till a complete suite for VSP data processing is written in Python.

References

1. Github.com

2. Rizvandi, N.B., Boloori, A.J., Kamyabpour, N., and Zomaya, A.Y., Map Reduce implementation of Prestack Kirchhoff Time Migration(PKTM) on seismic data, Proceedings of the 2011 12th International Conference on Parallel and Distributed Computing, Applications and Tecnologies, PDCAT, 2011,pp. 86-91, IEEE, Gwangju, South Korea.

3. Schwehr,K., Seismic–Py: Reading Seismic data with Python, Center for Coastal and Ocean Mapping, pp. 472, 2008, View at Google Scholar.

4. Suyanto, W and Irnaka, T.M., Web based application for inverting one-dimensional magneto-telluric data using Python, Computer & Geosciences, vol. 96, pp. 77-86, 2016.

Acknowledgements

Thanks are due to Director (E), ONGC, for according permission to publish/present this paper.

Authors are indebted to M Hanuman Sastry, CGM-Geophysics, HGS, ONGC, Dr. G V R Kumar, GM-Geophysics, Head-RCC, ONGC, K. Parasuraman, GM-Geophysics, ONGC and B. Ravindranath, GM-Programming, ONGC for their constant encouragement and support throughout this study.

The views expressed in this paper are solely of the authors and do not necessarily endorsed by ONGC.