Tutorial1_PRE_data

May 10, 2024

[ ]:
```
%matplotlib inline
```

This script presents a work flow for ingesting AEM data for further preprocessing. @author: vrath nov 2020

[ ]:
```
import os
import sys
from sys import exit as error
from time import process_time
from datetime import datetime
import warnings

import numpy


AEMPYX_ROOT = os.environ["AEMPYX_ROOT"]
mypath = [AEMPYX_ROOT+"/aempy/modules/", AEMPYX_ROOT+"/aempy/scripts/"]
for pth in mypath:
    if pth not in sys.path:
        sys.path.insert(0,pth)

from version import versionstrg
import util
import aesys

AEMPYX_DATA = os.environ["AEMPYX_DATA"]
```

[ ]:
```
version, _ = versionstrg()
titstrng = util.print_title(version=version, fname=__file__, out=False)
print(titstrng+"\n\n")

now = datetime.now()
Header = titstrng
```

Now some parameters controlling the work flow need to be defined.

OutInfo = *True* will activate the output of some intermediate information. This parameter, as well as most of the header, is common to all tutorial scripts.

OutInfo = True will activate the output of some intermediate information. This parameter, as well as most of the header, is common to all tutorial scripts.

FileList = "search" will choose a search of files to be read based on a string search (including wildcards). This parameter, as well as most of the header, is common to all tutorial scripts.

SearchStr = ".xyz" is only necessary when "search" is chosen.

FileList = "set" will require a list of files stored in the variable *DataSet*.

[ ]:
```
OutInfo = True

FileList = "search"
SearchStr = ".xyz"
DataSet = []

# FileList = "set"
# DataSet = ["File1", "File2"]
```

The following parameter control the treatment and output of data. *CheckNaN = True* will look for invalid data (e.g., "*" when exported by Geosoft). *MergeOut = True* and *LinesOut = True* will activate the output of full data set and individual fligh tlines, respectively. The former is convenient for any task requiring spatial information. The minimum number of sites can be set, if individual flight lines are exported.

We suggest using the ".npz" ouput format, as it is the easiest and fastest to be read by any python software, an uses compression, leading to smaller file sizes.

[ ]:
```
CheckNaN = True
MergeOut = True
LinesOut = True
LinesMin = 30

OutFileFmt = ".npz" #".asc"
```

The following two parameters allow to change the projections (now redundant, as this is already done in module *aesys.py*). GSI chose the ITM system (EPSG=2157), which is not known to many useful software, e.g., google earth. Within *AEMpyX*, the coordinates are in UTM (Zone 29N, EPSG==32629). Flight lines should also be stored in the same direction, setting *CorrectDirection=True*. This is convenient for practical reasons, as comparing plots.

[ ]:
```
SetProj = False
if SetProj:
    ProjInp = ""
    ProjOut = ""
```

[ ]:
```
TellusAng = 345.
Spread = 5.
CorrectDirection = True
```

The following block defines the choice of date to be processed. Most often the choice is simply a rectangle, as demonstrated here. The original data files provided by GSI are too large to be stored with git, thus need to be downloaded from https://www.gsi.ie/en-ie/data-and-maps/Pages/Geophysics.aspx, and stored in a local directory of the user's choice (see below).

We need define some necessary paramters controlling the reading of the original data, and the choice of an appropriate subset. In this case it is a rectangle covering the outcrops of black shapes in Co. Limerick, south of the Shannon estuary.

[ ]:
```
RectCorners = []
PolyFiles = []
DataSelect = ""
```

[ ]:
```
AEM_system = "aem05"
_, NN, _, _, _, = aesys.get_system_params(AEM_system)
nD = NN[0]

AEMPYX_DATA = AEMPYX_ROOT+"/work/"
DataSelect = "Rectangle"   # "Polygon", "Intersection", "Union"
InDatDir = AEMPYX_DATA+"/Limerick/"
OutDatDir = InDatDir+"/raw/"
RectCorners = [486000., 5815000., 498000., 5828000.]
InSurvey = "A5"
OutStrng = InSurvey+"_rect_shale"
```

After this, generally no code changes are necessary.

[ ]:
```
print("Data read from dir:  %s" % InDatDir)
print("Data written to dir: %s" % OutDatDir)
print("Flightline ID string: %s \n" % OutStrng)

if "search" in FileList.lower():
    DataSet = []

    files = os.listdir(InDatDir)
    for entry in files:
        # print(entry)
        if SearchStr in entry.lower():
            DataSet.append(entry)

DataSet = sorted(DataSet)
ns = numpy.size(DataSet)

if not os.path.isdir(OutDatDir):
    print("File: %s does not exist, but will be created" % OutDatDir)
    os.mkdir(OutDatDir)
```

```
dcount=0
for dset in DataSet:
    dcount=dcount+1
    start = process_time()
    file = InDatDir + dset
    print("\nRaw data read from: %s" % file)
    Datar = aesys.read_survey_data(DatFile=file, Survey=InSurvey, OutInfo=True)
    if dcount == 1:
        Data = Datar
    else:
        Data = numpy.vstack((Data, Datar))
    print("Read time taken = ", process_time() - start, "s \n")


if SetProj:
    start = process_time()
    itm_e = Data[:,1]
    itm_n = Data[:,2]
    utm_e, utm_n = util.project_itm_to_utm(itm_e, itm_n) #, utm_zone=32629)
    Data[:,1] = utm_e
    Data[:,2] = utm_n
    print("ITM Transformed to UTM")
    print("Projection time taken = ", process_time() - start, "s \n")
```

Data subsets based on rectangle, polygons or operators on polygons

[ ]:
```
start = process_time()
print("In: "+str(numpy.shape(Data)))
print("Data select is "+DataSelect+" \n")

if "rec" in DataSelect.lower():
    head = aesys.grow_header(Header, "Subset: " + str(RectCorners))
    start = process_time()
    Rect = util.extract_data_rect(Data, RectCorners)
    if Rect.size != 0:
        Data = Rect
    else:
        error("No data found in rectangle!\n")

if "pol" in DataSelect:
    head = aesys.grow_header(Header,
            " | Subset: " + str(DataSet)+": "+PolyFiles[0])
    Polygon = numpy.load(PolyFiles[0], allow_pickle=True)["Poly"][0]
    start = process_time()
    Poly= util.extract_data_poly(Data, Polygon, method="shp")
    if Poly.size != 0:
        Data = Poly
```

```
    else:
        error("No data found in polygon!\n")

if ("uni" in DataSelect.lower()) or ("int" in DataSelect.lower()):
    for polyfile in PolyFiles:
        head = aesys.grow_header(Header,
                " | Subset: "+DataSelect.lower()[0:3]+": "+polyfile)
    Polygon1 = numpy.load(PolyFiles[0], allow_pickle=True)["Poly"][0]
    Polygon2 = numpy.load(PolyFiles[1], allow_pickle=True)["Poly"][0]
    start = process_time()
    Polygon= util.modify_polygon([Polygon1, Polygon2], Operator=DataSelect)
    Poly= util.extract_data_poly(Data, Polygon, method="shp")
    if Poly.size != 0:
        Data = Poly
    else:
        error("No data found in polygons!\n")

print("Data select time taken = ", process_time() - start, "s \n")
print("Out: "+str(numpy.shape(Data)))

if MergeOut:
    head = aesys.grow_header(Header,"All Lines")
    f = OutDatDir + OutStrng+"_Full"+OutFileFmt
    aesys.write_aempy(File=f, Data=Data, System=AEM_system,
                      Header=head, OutInfo=OutInfo)
    print("All data written to File: " + f )
    print("Header written: ")
    print(head)
    print("time taken = ", process_time() - start, "s \n")


if LinesOut:
    bad_files = 0
    startlines = process_time()
    Lines = sorted(numpy.unique(Data[:, 0]))
    print(">Flight lines in data set:")
    print(Lines)
    for s in Lines:
        tmp = Data[numpy.where(Data[:, 0] == s), :]
        ns = numpy.shape(tmp)
        tmp = numpy.reshape(tmp, (ns[1], ns[2]))
        print("OutInfo: "+str(numpy.shape(tmp)))

        if numpy.size(tmp)<=nD*LinesMin:
            print("Not enough data! Not written")
            continue
```

```
        if CheckNaN:
                nn = numpy.count_nonzero(numpy.isnan(tmp))
                print (str(nn)+" NaNs in Data Block")
                if nn >0:
                    bad_files = bad_files+1
                    print("Too many NaNs = "+str(nn)+" in block, not written")
                    continue


        if CorrectDirection:
            AngLimits = [TellusAng-5., TellusAng+5. ]
            nd =numpy.shape(tmp)[0]
            spoint = [tmp[round(nd*0.3),1], tmp[round(nd*0.3),2]]
            epoint = [tmp[round(nd*0.6),1], tmp[round(nd*0.6),2]]
            ang, _ = util.get_direction_angle(spoint, epoint)
            if (ang < TellusAng-Spread) or (ang > TellusAng+Spread):
                tmp = numpy.flipud(tmp)
                print(" Angle = "+str(round(ang,1))
                    +" not in interval "
                    +str(round(AngLimits[0],1))+" - "
                    +str(round(AngLimits[1],1)))
                print("Flightline direction has been reversed.")
                chdir = ", direction has been reversed"
            else:
                print("Flightline direction is approx. 345 degrees")
                chdir = ""

        head = aesys.grow_header(Header, "Flightline " + str(s))

        f = OutDatDir + OutStrng + "_FL" + str(s).replace(".", "-")+OutFileFmt
        aesys.write_aempy(File=f, Data=tmp, System=AEM_system,
                          Header=head, OutInfo=OutInfo)
        print("Flight line written to File: " + f)
        print("Header written: ")
        print(head)
        print("time taken = ", process_time() - start, "s \n")

    print("Flight line data, time taken = ",
          process_time() - startlines, "s \n")
```