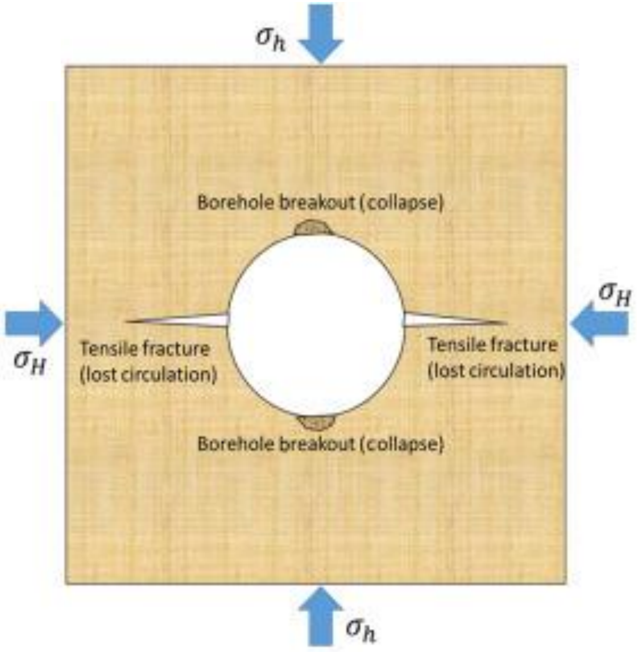
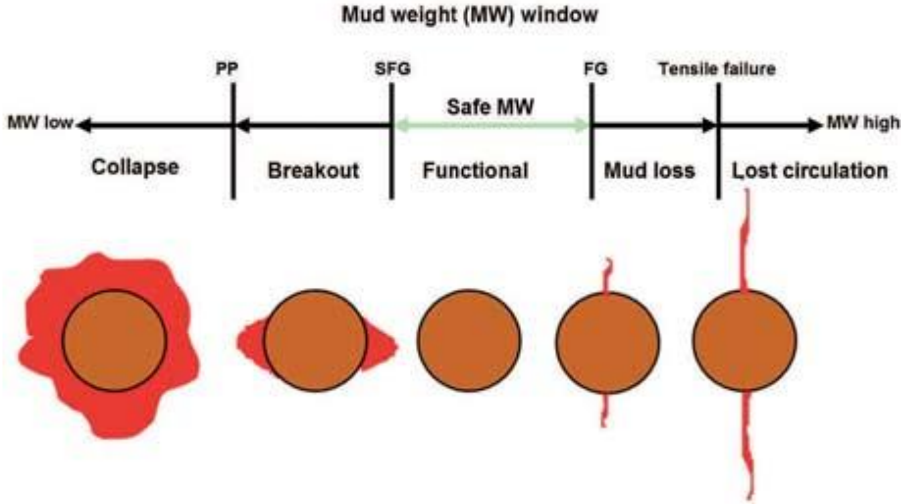
‘Analysis of Wellbore Stability with consideration of Stress Anisotropy and Wellbore Deviation’

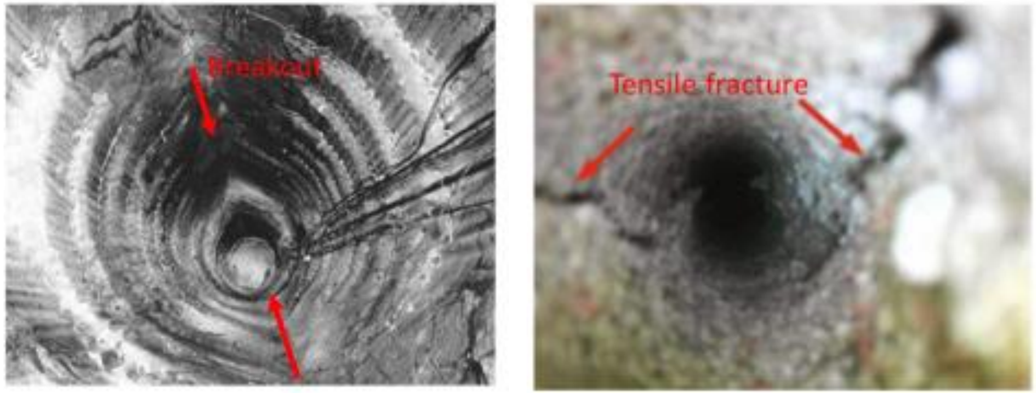
Python for Geoscience Research Final Project

Agustín Garbino - MS Student Petroleum Engineering

Project Goals and Motivations







- Wellbore breakouts can lead to well collapse, while tensile fractures can bring important fluid losses
- Calculating the stress limits for both events becomes crucial for calculating the mud weight window during drilling operations
- Stress anisotropy along the wellbore must be accounted for to get accurate results

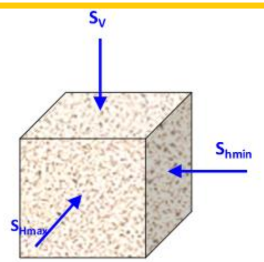
Theory & Steps

1. Principal Stresses

$$S_v(z) = \int_0^z \rho_{bulk}(z) g \, dz$$

Assuming elasticity and isotropy of mechanical properties:

$$\begin{cases} \sigma_{Hmax} = \dfrac{\nu}{1-\nu}\sigma_v + E'\varepsilon_{Hmax} + \nu E'\varepsilon_{hmin} \\ \sigma_{hmin} = \dfrac{\nu}{1-\nu}\sigma_v + \nu E'\varepsilon_{Hmax} + E'\varepsilon_{hmin} \end{cases}$$



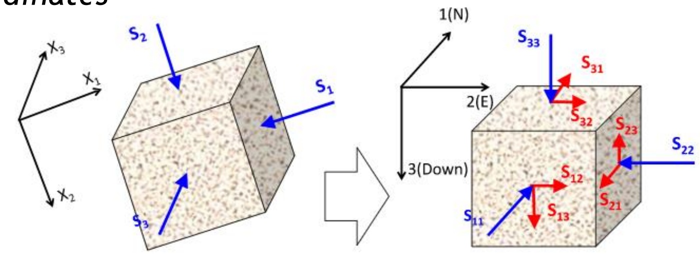
Rock mechanical properties:

$$E_{dyn} = \rho \, V_s^2 \frac{3V_p^2 - 4V_s^2}{V_p^2 - V_s^2}$$

$$\nu_{dyn} = \frac{V_p^2 - 2V_s^2}{2V_p^2 - 2V_s^2}$$

2. Rotation of Principal Stresses into Geographical Coordinates

$$R_{PG} = \begin{bmatrix} \cos\alpha\cos\beta & \sin\alpha\cos\beta & -\sin\beta \\ \cos\alpha\sin\beta\sin\gamma - \sin\alpha\cos\gamma & \sin\alpha\sin\beta\sin\gamma + \cos\alpha\cos\gamma & \cos\beta\sin\gamma \\ \cos\alpha\sin\beta\cos\gamma + \sin\alpha\sin\gamma & \sin\alpha\sin\beta\cos\gamma - \cos\alpha\sin\gamma & \cos\beta\cos\gamma \end{bmatrix}$$

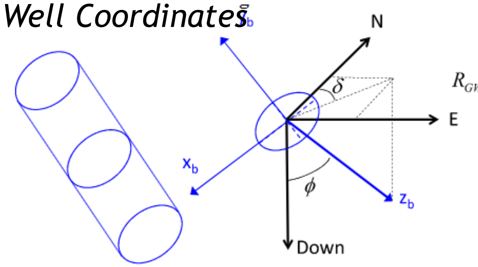


$$\underline{\underline{S}}_G = R_{PG}^T \underline{\underline{S}}_P R_{PG}$$

$$\underline{\underline{S}}_P = \begin{bmatrix} S_1 & 0 & 0 \\ 0 & S_2 & 0 \\ 0 & 0 & S_3 \end{bmatrix} \qquad \underline{\underline{S}}_G = \begin{bmatrix} S_{11} & S_{12} & S_{13} \\ S_{21} & S_{22} & S_{23} \\ S_{31} & S_{32} & S_{33} \end{bmatrix}$$

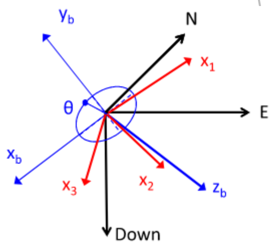
3. Rotation of Principal Stresses from Geo to Well Coordinates

$$\underline{\underline{S}}_W = R_{GW} \underline{\underline{S}}_G R_{GW}^T$$



$$R_{GW} = \begin{bmatrix} -\cos\delta\cos\phi & -\sin\delta\cos\phi & \sin\phi \\ \sin\delta & -\cos\delta & 0 \\ \cos\delta\sin\phi & \sin\delta\sin\phi & \cos\phi \end{bmatrix}$$

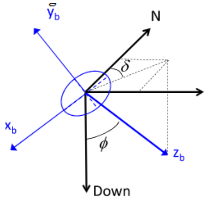
4. Kirsch Equations - Well Principal Stresses



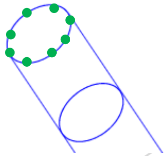
Stresses at the wellbore wall (Kirsch[PP]+Kirsch[S])

$$\begin{cases} \sigma_{rr} = \Delta P \\ \sigma_{\theta\theta} = \sigma_{11} + \sigma_{22} - 2(\sigma_{11} - \sigma_{22})\cos 2\theta - 4\sigma_{12}\sin 2\theta - \Delta P \\ \tau_{\theta z} = 2(\sigma_{23}\cos\theta - \sigma_{13}\sin\theta) \\ \sigma_{zz} = \sigma_{33} - 2\nu(\sigma_{11} - \sigma_{22})\cos 2\theta - 4\nu\sigma_{12}\sin 2\theta \end{cases}$$

5. Principal Stresses around the deviated wellbore perimeter



Wellbore principal stresses calculated for all possible values of azimuth and inclination



$$\begin{bmatrix} \sigma_{rr} & 0 & 0 \\ 0 & \sigma_{\theta\theta} & \sigma_{\theta z} \\ 0 & \sigma_{z\theta} & \sigma_{zz} \end{bmatrix} \xrightarrow{\text{Eigen values}} \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \end{bmatrix}$$

$$\boxed{UCS = \max(\max\_shear\ stress)}$$

$$\boxed{TS = \min(\min\_shear\ stress)}$$

Calculations – Input Datasets

The project was tested with the data from a deviated well (the information was provided by a professor from a different course):

- ***Well logs**: includes density, sonic and interpreted Pore pressure*
  - *This curves come from a real well near the Gulf of Mexico (well name is confidential)*
- ***Well Deviation survey**: Table with azimuth, Easting, Northing and inclination for the well*
  - *This data was obtained from the same well, but was modified synthetically to increase the deviation in this well for the results to be more applicable to deviated wells*

- Well Deviation survey - .xlsx file

Measured Depth (m)	Inclination (°)	Azimuth (°)	Vertical Depth (m)	+N/-S (m)	+E/-W (m)
0.00	0.000	0.00	0.00	0.00	0.00
66.73	0.360	310.89	66.73	0.14	-0.16
95.62	0.330	287.90	95.62	0.22	-0.31
108.44	0.310	306.29	108.44	0.25	-0.37
136.81	0.670	302.69	136.81	0.39	-0.57
154.70	0.800	262.36	154.70	0.43	-0.78
172.91	1.060	257.00	172.90	0.37	-1.07
201.62	1.670	261.90	201.61	0.26	-1.75
220.14	1.810	254.89	220.12	0.14	-2.30
238.17	1.950	252.42	238.14	-0.03	-2.86
257.13	1.400	254.74	257.09	-0.18	-3.39
285.98	0.850	229.06	285.93	-0.42	-3.90
314.75	0.680	262.60	314.70	-0.58	-4.23
343.33	0.320	287.20	343.28	-0.58	-4.47
371.60	0.530	36.85	371.55	-0.45	-4.47

- Well logs (Density, Sonic) – .LAS file

```
GR      .GAPI   :       5       Gamma   Ray
NPHI    .V/V    :       6       Neutron Porosity
PEF     .B/E    :       7       Photo-Electric  Factor
RHOB    .G/CC   :       9       Bulk    Density
RS      .OHMM   :       10      Shallow Resistivity
RT      .OHMM   :       8       Deep    Resistivity
RXO     .OHMM   :       11      Micro   Resistivity
SP      .MV     :       12      Spontaneous     Potential
~Parameter      Information     Block
#MNEM.UNIT      Value   Description
#----------     ----------------        -----------
SET     .       COMPOSITE:
BHIS    .       M:      BRIEF   HISTORY
BHT1    .DEGF   82.0:   BH      TEMP    1
BHT2    .DEGF   142.0:  BH      TEMP    2
BHT3    .DEGF   143.0:  BH      TEMP    3
BS1     .IN     17.50:  BIT     SIZE    1
BTD1    .       1056.0  M:      TOTAL   DEPTH   1
BTD2    .       2753.50 M:      TOTAL   DEPTH   2
BTD3    .       2753.50 M:      TOTAL   DEPTH   3
CBD1    .M      215.0:  CSG     1       DRILL   DEP
CBL1    .M      213.0:  CSG     1       LOG     DEP
CS1     .IN     30.0:   CSG     1       SIZE
EKB     .M      25:00:00        KB      ELEVATION
FHIS    .       MERGE:  FULL    HISTORY
LNAM    .       COMPOSITE:      NAME
LSOU    .       PAU:    LOG     SOURCE
LSRV    .       LCD:    SERVICE
LTYP    .       CORRECTED:      LOG     TYPE
LVSN    .       01:00   LOG     VERSION
SPUD    .       2001/09/23:     SPUD    DATE
~Other  Information     Block
#       DEPTH   CALI    DRHO    DT      GR      NPHI    PEF     RHOB    RS      RT      RXO     SP
~A
        114.19640       -999.2500       -999.250000     -999.2500       4.3438  -999.2500       -999.2500       -999.2500
        114.34880       -999.2500       -999.250000     -999.2500       5.2109  -999.2500       -999.2500       -999.2500
        114.50120       -999.2500       -999.250000     -999.2500       5.2422  -999.2500       -999.2500       -999.2500
        114.65360       -999.2500       -999.250000     -999.2500       5.8164  -999.2500       -999.2500       -999.2500
        114.80600       -999.2500       -999.250000     -999.2500       5.2539  -999.2500       -999.2500       -999.2500
        114.95840       -999.2500       -999.250000     -999.2500       5.8164  -999.2500       -999.2500       -999.2500
```

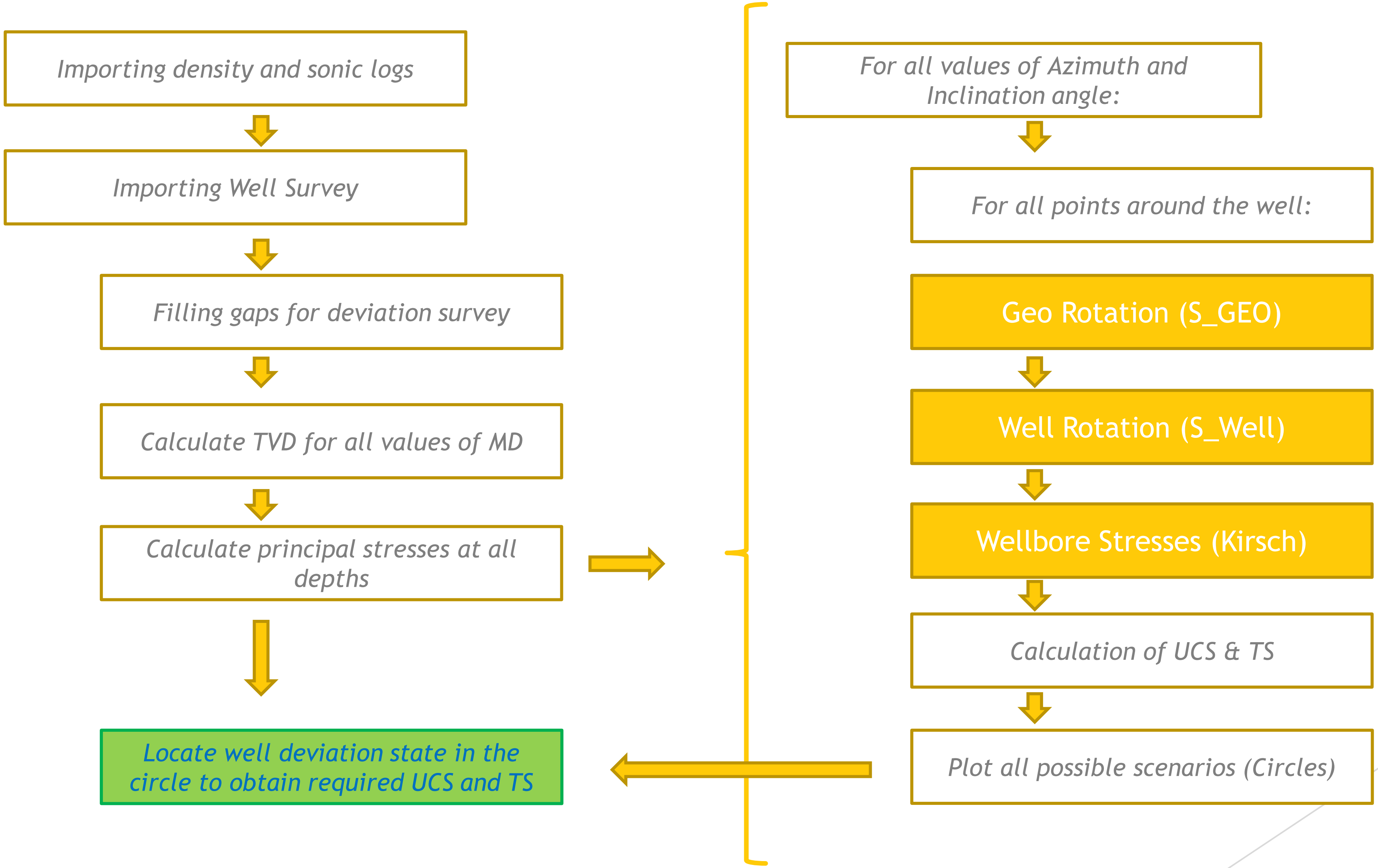
Calculations – Input Datasets

Some important definitions:

- ***Azimuth***: *the direction of an object/vector with respect to the North*
- ***Inclination***: *the degree of slope with respect to the vertical*
- ***Density log***: *a well logging tool that can provide a continuous record of a formation's bulk density along the length of a borehole*
- ***Sonic log***: *an acoustic log that emits sound waves which start at the source, travel through the formation, and return back to the well*
- ***Young Modulus***: *the slope of the linear part of the stress-strain curve for a material under tension or compression; it tells how easily a material can stretch and deform*
- ***Poisson Ratio***: *the amount of transversal elongation divided by the amount of axial compression; it shows how the cross-section of a deformable body changes under lengthwise stretching (or compression)*
- ***Principal Stresses***: *Set of 3 applied normal stresses in the principal planes, in a system defined where shear stresses are equal to zero*

Methods - Workflow

Workflow

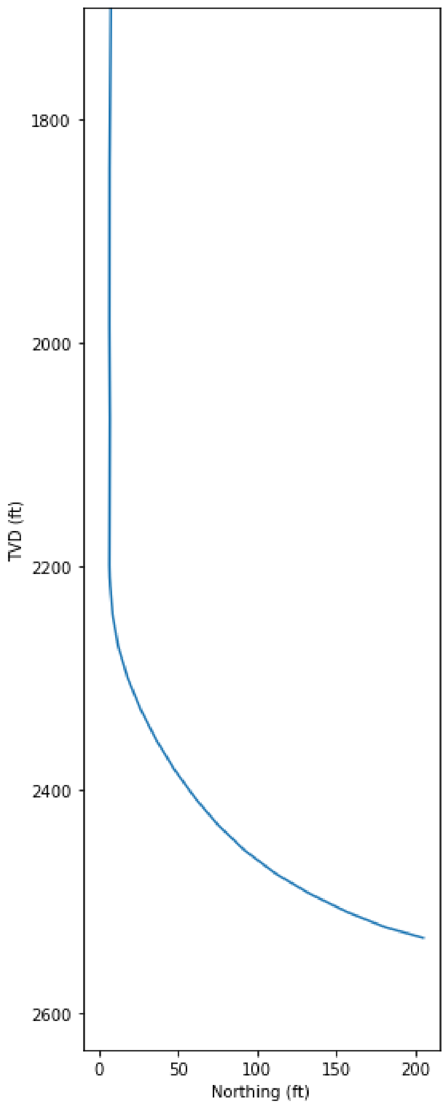


Python Libraries:

- Pandas
- Numpy
- Scipy
- Matplotlib
- Math
- Welly: *Useful to read .las files and convert into Pandas Dataframe*

Results

Well Deviation Survey

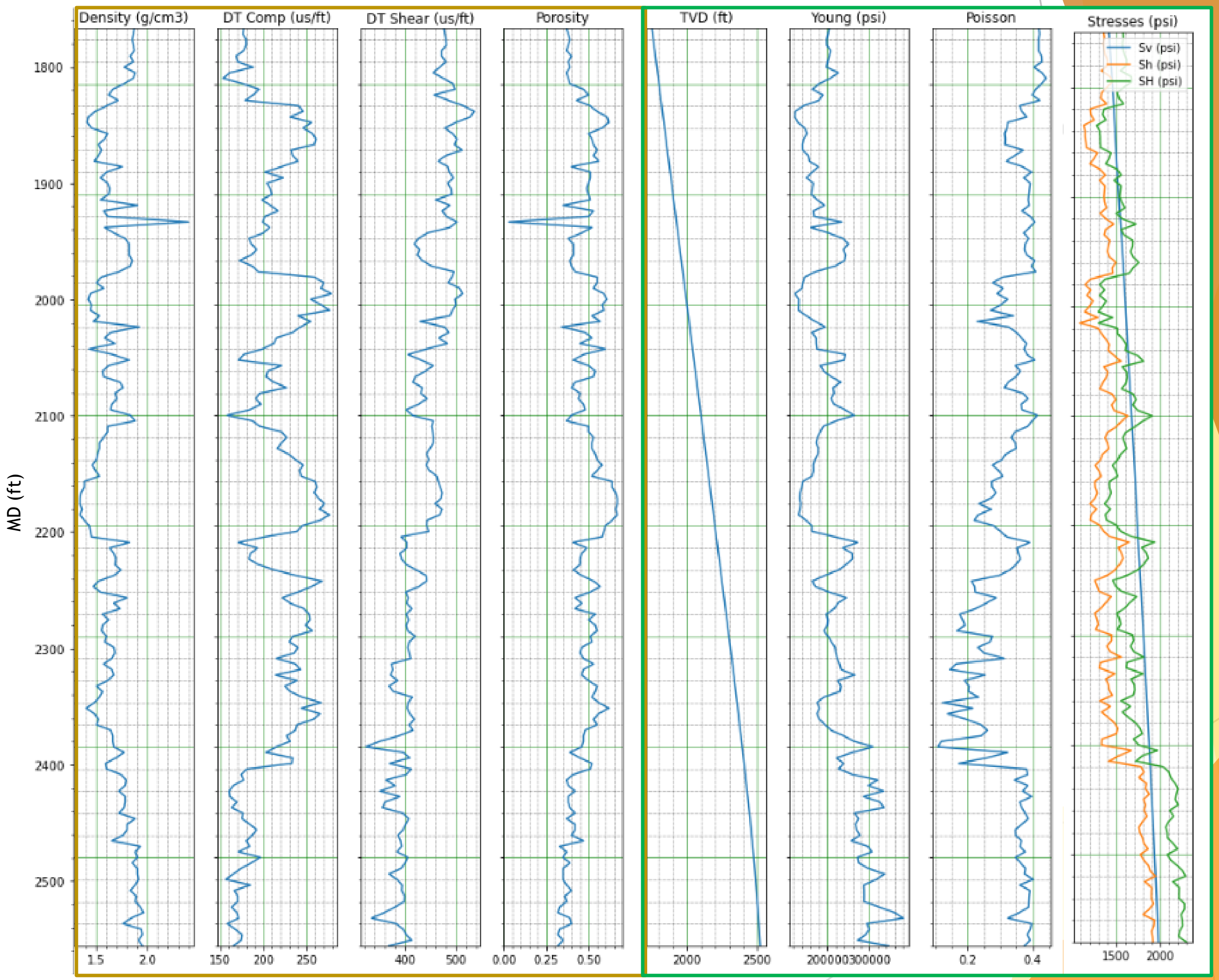


INPUTS

Density and sonic logs

OUTPUTS

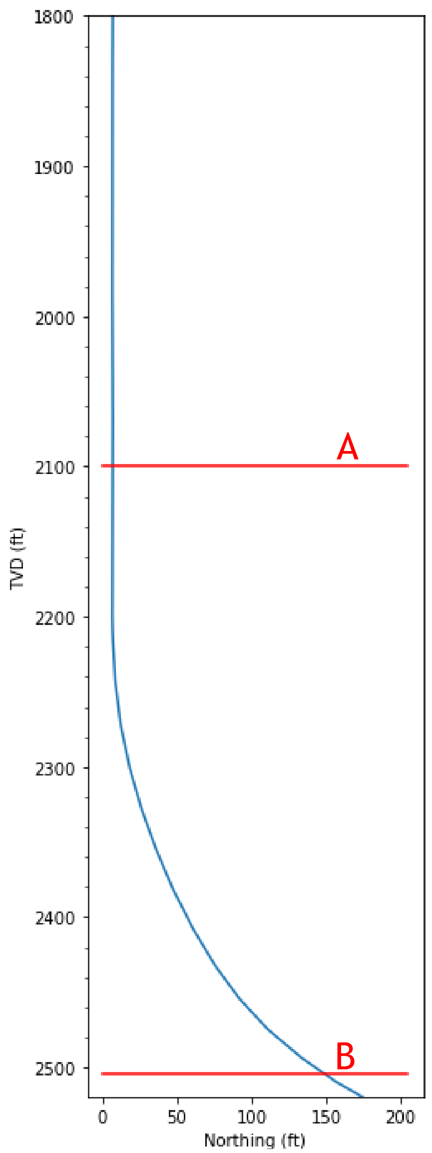
Poisson Ratio, Young Modulus and Total Stresses

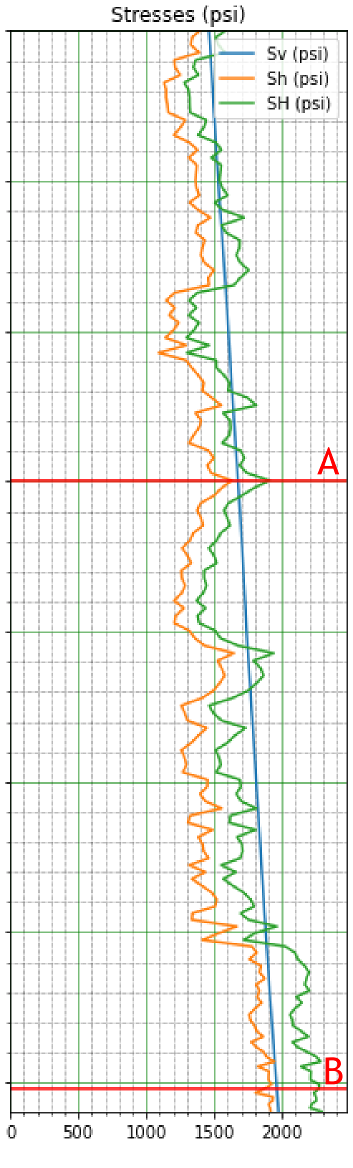


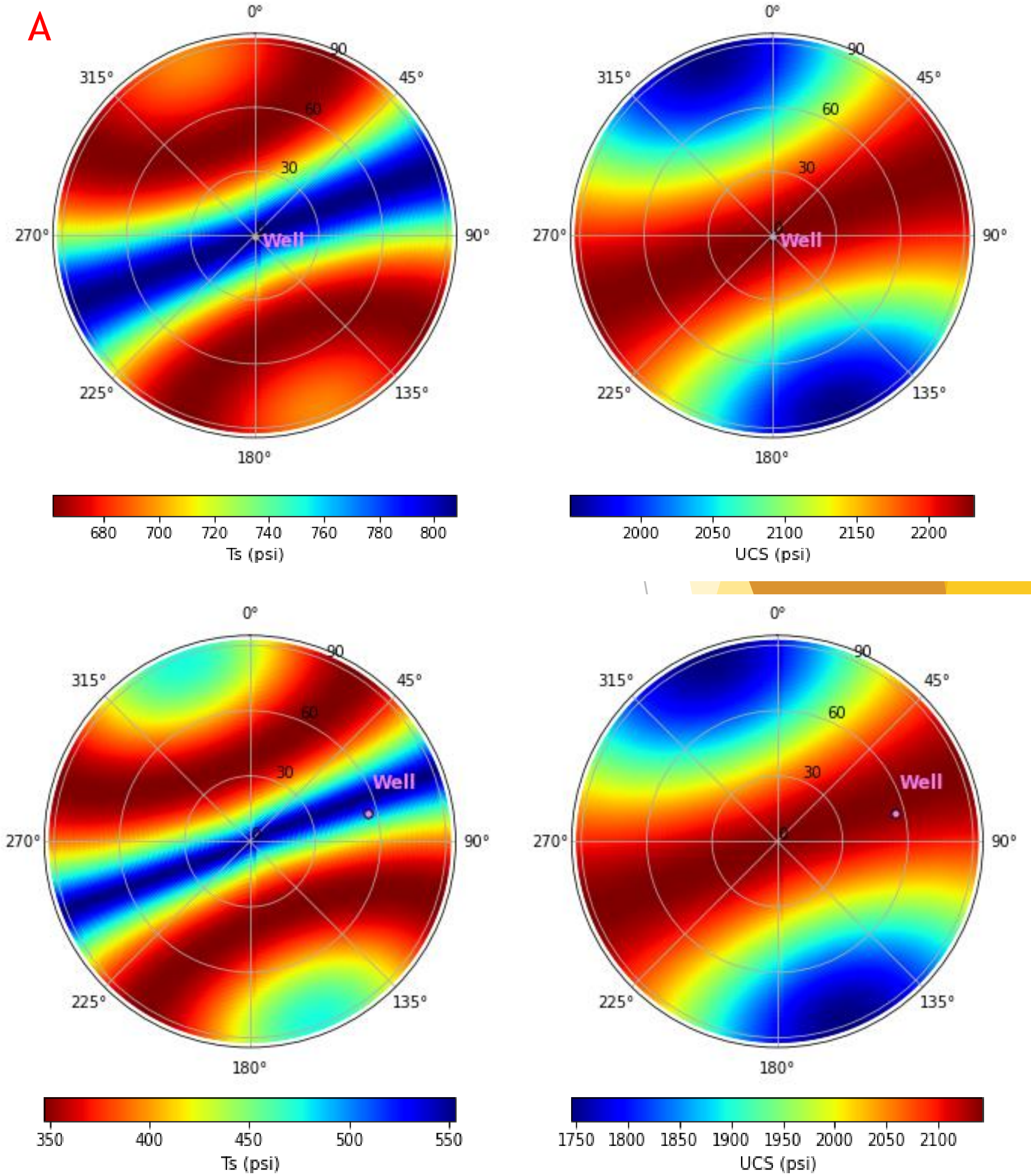
Stresses are calculated at all depths

- The figure on the left is a vertical cross section of the well trajectory (y axis is true vertical depth)
- The figures on the right are the well logs and the calculated properties and stresses at each depth (y axis is total well length, not vertical depth)

Results







- In this case both well trajectory and logs are displayed in true vertical depth
- The point represents well current deviation compared to all possible states of deviation
- The colors are a reference for UCS and TS requirements at current deviation and depth

Conclusions

- The deviation of the wellbore has a strong impact on the stresses around the well
- Depending on the depth, deviation of the wellbore and stress regime, different values for mud pressure are required to avoid tensile fractures and breakouts
- For different deviation angles, the necessary unconfined compressive strength and the tensile strength of the well can become sensitively higher
- It becomes critical to account for these differences in order to avoid tensile fractures and breakouts in the wellbore

Future Work

- A possibility for future work is considering the effect of temperature on stresses, to assess how much wellbore stability calculations are affected by mud temperature
- Another interesting approach would be addressing how overpressure affects these results by comparing a well in an overpressured formation with a normal pressure reservoir

Appendix - Code

Python Libraries:

- Pandas
- Numpy
- Scipy
- Welly
- Matplotlib
- Math

Function for GEO rotation of S matrix

```python
#I define a function for rotation of the stress tensor into geographical coordinates

def Geo_rot(Alpha, Betha, Gamma, STens):

    Alpha = Alpha*2*math.pi/360
    Betha = Betha*2*math.pi/360
    Gamma = Gamma*2*math.pi/360

    RPG = np.array([[np.cos(Alpha)*np.cos(Betha),                               np.sin(Alpha)*np.cos(Betha),
          [np.cos(Alpha)*np.sin(Betha)*np.sin(Gamma)-np.sin(Alpha)*np.cos(Gamma), np.sin(Alpha)*np.sin(Betha)*np.sin(Gamma)+
          [np.cos(Alpha)*np.sin(Betha)*np.cos(Gamma)+np.sin(Alpha)*np.sin(Gamma), np.sin(Alpha)*np.sin(Betha)*np.cos(Gamma)-

    # Transpose RPG

    RPG_T = RPG.transpose()

    # Calculate Stress Tensor in Geographical Coordinate System
    # SGEO = RGP_T * STens_E * RGP

    SGEO = np.round(np.dot(RPG_T,np.dot(STens,RPG)))


    return SGEO
```

Function for WELL rotation of S_GEO matrix

```python
#I define a function for rotation of the stress tensor in geographical coordinates to wellbore coordinates

def Well_rot(delta, phi, STens): #phi is the angle with respect to the vertical axis, delta is the azimuth

    delta = delta*2*math.pi/360
    phi = phi*2*math.pi/360


    RB = np.array([[-np.cos(delta)*np.cos(phi),          -np.sin(delta)*np.cos(phi),         np.sin(phi)],
           [np.sin(delta),                                    -np.cos(delta) ,                    0],
           [np.cos(delta)*np.sin(phi)   ,                     np.sin(delta)*np.sin(phi)    ,       np.cos(phi)]])

    # Transpose RB

    RB_T = RB.transpose()

    # Calculate Stress Tensor in Wellbore Coordinate System
    # SWell = RB * STens * RB_T

    SWell = np.round(np.dot(RB,np.dot(STens,RB_T)),2)


    return SWell
```

Function for calculating Kirsch stresses around the wellbore

```python
def kirsch(tens, theta, PW, PP, Poisson):

    theta = theta*2*math.pi/360

    sigma11 = tens[0][0] - PP
    sigma12 = tens[0][1]
    sigma13 = tens[0][2]
    sigma22 = tens[1][1] - PP
    sigma23 = tens[1][2]
    sigma33 = tens[2][2] - PP

    sigma_zz =  sigma33 - 2*Poisson*(sigma11 - sigma22)*np.cos(2*theta) - 4*Poisson*sigma12*np.sin(2*theta)
    sigma_tt = sigma11 + sigma22 - 2*(sigma11 - sigma22)*np.cos(2*theta) - 4*sigma12*np.sin(2*theta) - (PW - PP)
    tau_tz = 2*(sigma23*np.cos(theta) - sigma13 * np.sin(theta))
    sigma_rr = PW - PP

    sigma_max = round( (sigma_zz + sigma_tt) / 2 + math.sqrt(((sigma_zz-sigma_tt)/2)**2 + tau_tz**2) , 3)
    sigma_min = round( (sigma_zz + sigma_tt) / 2 - math.sqrt(((sigma_zz-sigma_tt)/2)**2 + tau_tz**2) , 3)

    return sigma_max, sigma_min
```

Function that calls previous functions and calculates UCS and TS for all possible scenarios of well deviation

```python
# Tensile Fractures Likelihood
def Tensile(Alpha, Beta, Gamma, S1, S2, S3, PW, PP, Poisson, inc = 0, azi = 0):

    s_tensor = np.array([[S1, 0 , 0], [0, S2, 0], [0, 0, S3]] )

    geo_s = Geo_rot(Alpha, Beta, Gamma, s_tensor) #principal stresses to geographical coordinates

    phi = np.linspace(0, 90, 90) #well dip array
    delta = np.linspace(0,180, 180) #well azimuth array
    angle = np.linspace(0,180, 180) #well border angle

    UCS = [] #UCS array
    T0 = []  # Tensile strength array
    Mohr_w = [] # width by Mohr criterium
    DP_w = [] # width by Drucker Prager criterium
    ML_w = [] #width by Modified Lade criterium
    x = [] #x coordinate for plotting
    y = [] #y coordinate for plotting
    deltar = []
    phir = []

    for p in phi:
        for d in delta:

            x.append(p*math.sin(d*2*math.pi/360)) #cos and sin are inverted because of the plot
            y.append(p*math.cos(d*2*math.pi/360))
            deltar.append(d*2*math.pi/360)
            phir.append(p)

            well_s = Well_rot(d, p, geo_s) #principal stress tensor to well coordinates
            max_s_list = []
            min_s_list = []
            width_MC = 0
            width_DP = 0
            width_ML = 0

            for ang in angle:
                max_s, min_s = kirsch(well_s, ang, PW, PP, Poisson) #calculate min and max stress for each point in the well
                max_s_list.append(max_s)

            UCS_n = max(max_s_list)
            T0_n = min(min_s_list)

            UCS.append(UCS_n)
            T0.append(T0_n)
            Mohr_w.append(width_MC/2)
            DP_w.append(width_DP/2)
            ML_w.append(width_ML/2)

    # Duplicating the different arrays to get the results for the full circle
    UCS += UCS.copy()
    T0 += T0.copy()
    Mohr_w += Mohr_w.copy()
    DP_w += DP_w.copy()
    ML_w += ML_w.copy()

    x= np.array(x)
    x_m = x*(-1)
    x = np.append(x, x_m )

    y= np.array(y)
    y_m = y*(-1)
    y = np.append(y, y_m )

    deltar = np.array(deltar)
    deltar_m = deltar+np.pi
    deltar = np.append(deltar, deltar_m )

    phir = np.array(phir)
    phir_m = phir
    phir = np.append(phir, phir_m )
```

Code inside this function that plots UCS and TS for all possible scenarios - 'Polar' subplot was used

```python
    #Tendency for tensile failure considering a constant Ts criterium

    fig, ax = plt.subplots(ncols = 2, subplot_kw={'projection': 'polar'}, figsize = (10, 8))
    fig.tight_layout(pad=3.0)

    color_map = plt.cm.get_cmap('jet')
    reversed_color_map = color_map.reversed()
    plot_T0 = ax[0].scatter(deltar, phir, c=T0, cmap=reversed_color_map )
#    plot_T0 = ax.imshow(grid_T0, extent=(np.min(deltar),np.min(deltar),np.min(phir),np.max(phir)), origin='lower')
    cb_0 = plt.colorbar(plot_T0, ax=ax[0] , orientation = 'horizontal')
    ax[0].set_rmax(90)
    ax[0].set_rticks([0, 30, 60, 90])  # Less radial ticks
 #  ax.set_rlabel_position(-22.5)  # Move radial labels away from plotted line
    ax[0].set_theta_zero_location('N')
    ax[0].set_theta_direction(-1)
    ax[0].grid(True)
    cb_0.set_label('Ts (psi)', fontsize=11)

    #Tendency for breakouts considering a constant UCS criterium

    plot_UCS = ax[1].scatter(deltar, phir, c=UCS , cmap='jet')
#    plot_UCS = ax.imshow(grid_UCS, extent=(np.min(deltar),np.min(deltar),np.min(phir),np.max(phir)), origin='lower')
    cb_1 = plt.colorbar(plot_UCS, ax = ax[1], orientation = 'horizontal')
    ax[1].set_rmax(90)
    ax[1].set_rticks([0, 30, 60, 90])  # Less radial ticks
 #  ax.set_rlabel_position(-22.5)  # Move radial labels away from plotted line
    ax[1].set_theta_zero_location('N')
    ax[1].set_theta_direction(-1)
    ax[1].grid(True)
    cb_1.set_label('UCS (psi)', fontsize=11)

    if inc > 0:
        ax[0].scatter(azi, inc, c='violet', s=20)
        ax[0].annotate('Well', (azi, inc+10), color = 'violet', size=12)
        ax[1].scatter(azi, inc, c='violet', s=20)
        ax[1].annotate('Well', (azi, inc+10), color='violet', size=12)
```

Appendix - Code

.LAS transformation into Pandas Dataframe - Welly Library

Import las file

```
project = welly.read_las('https://raw.githubusercontent.com/dnicolasespinoza/GeomechanicsJupyter/master/1_14-1_Composite.las'
well=project[0]
data = well.df()
```

```
1it [00:00,  1.60it/s]
```

```
data.reset_index(inplace=True,  level=['DEPTH']) #I turn DEPTH into a column
#data.reset_index(inplace=True, drop=True)
data.tail()
```

74]:

	DEPTH	CALI	DRHO	DT	GR	NPHI	PEF	RHOB	RS	RT	RXO	
25625	4019.4463999985	9.7786	-0.0030000000	117.1808999903	117.5173999506	32.0138999913	NaN	2.5683000000	2.2891000000	1.6457999982	NaN	39.
25626	4019.5987999985	9.7786	-0.0006000000	117.1874999999	112.4196000487	30.6962000126	NaN	2.5733000000	2.3664999993	1.6648999998	NaN	39.
25627	4019.7511999985	9.7786	-0.0007000000	116.5666000059	111.6314000075	30.3350000034	NaN	2.5550000002	2.5123999986	1.7463999992	NaN	37.
25628	4019.9035999985	9.7786	-0.0059999999	111.2228000510	105.8625000551	29.0127000126	NaN	2.5267000003	2.8176999971	1.7934999996	NaN	36.
25629	4020.0559999985	9.7786	-0.0097000000	107.4000000365	100.9794000466	28.6232000037	NaN	2.4956000003	3.0642999976	1.8752999992	NaN	36.

.xlsx import into Pandas Dataframe & filling gaps - scipy Library

Import survey ¶

```
surv = pd.read_excel('Survey - Python Final Project.xlsx')
surv.head()
```

4]:

	MD (ft)	Inclination (°)	Azimuth (°)	TVD (ft)	Northing (ft)	Easting (ft)
0	0.00	0.00	0.00	0.00	0.00	0.00
1	8.39	0.47	71.05	8.39	0.05	0.07
2	24.65	0.46	69.45	24.65	0.09	0.20
3	42.08	0.45	67.65	42.08	0.14	0.32
4	70.37	1.30	66.30	70.37	0.31	0.72

```
# Fit interpolants

x = surv['MD (ft)']
y = surv['TVD (ft)']
MD_to_TVD = interp1d(x, y, kind='linear')

y = surv['Inclination (°)']
MD_to_Inc = interp1d(x, y, kind='linear')

y = surv['Azimuth (°)']
MD_to_Azi = interp1d(x, y, kind='linear')

y = surv['Northing (ft)']
MD_to_North = interp1d(x, y, kind='linear')

y = surv['Easting (ft)']
MD_to_East = interp1d(x, y, kind='linear')

MD = np.arange(0,int(max(surv['MD (ft)'])))
Inc = MD_to_Inc(MD)
Azi = MD_to_Azi(MD)
TVD = MD_to_TVD(MD)
North = MD_to_North(MD)
East = MD_to_East(MD)

surv_data = np.stack([MD, Inc, Azi, TVD, North, East], axis=-1)
headers = surv.columns.values

surv = pd.DataFrame(surv_data, columns = headers)
```

```
surv.head()
```

	MD (ft)	Inclination (°)	Azimuth (°)	TVD (ft)	Northing (ft)	Easting (ft)
0	0.0	0.0000000000	0.0000000000	0.0	0.0000000000	0.0000000000
1	1.0	0.0560190703	8.4684147795	1.0	0.0059594756	0.0083432658
2	2.0	0.1120381406	16.9368295590	2.0	0.0119189511	0.0166865316
3	3.0	0.1680572110	25.4052443385	3.0	0.0178784267	0.0250297974
4	4.0	0.2240762813	33.8736591180	4.0	0.0238379023	0.0333730632

Calculus of Mechanic Properties and Stresses from logs

Process logs

```
#I calculate TVD based on the survey

data['TVD (ft)'] = np.round(MD_to_TVD(data['DEPTH (ft)'].values),2)
```

```
data.head()
```

	DEPTH (ft)	Res Press (psi)	Density (g/cm3)	DT Comp (us/ft)	DT Shear (us/ft)	Porosity	TVD (ft)
0	1750.0	700.0	1.87	177.0	477.0	0.37	1749.38
1	1755.0	702.0	1.86	176.0	479.0	0.38	1754.37
2	1760.0	704.0	1.85	180.0	481.0	0.39	1759.37
3	1765.0	706.0	1.86	180.0	482.0	0.38	1764.37
4	1770.0	708.0	1.87	177.0	473.0	0.37	1769.37

```
# Density is in g/cm3
# Depth is in ft
#Sv is in psi
```

```
#I define Sv as the total vertical stress
Sv = np.zeros(len(data))
dens = data['Density (g/cm3)'].values #all the other density points apart from the first one
dh = data['TVD (ft)'].diff().shift(0).values #Length interval between depth points
h = data['TVD (ft)'].values #DEPTH array
h0 = h[0]
d0 = data['Density (g/cm3)'][0] #first density point

Sv[0] = d0*1/2.3*h0
Sv[1:] = dens[1:]*1/2.3*dh[1:]

# this is a nice line of code: Sv[1:] = [(dens[i]*1/2.3*dh[i]+Sv[i]) for i in range(len(dens))]
```

```
# Compute dynamic Poisson's ratio and dynamic Young's modulus from compressive and shear slowness

#Convert slowness into velocity and modify units into m/seg
Ds = data['DT Shear (us/ft)'].values
Vs = 1/Ds*0.3048*1e6

Dp = data['DT Comp (us/ft)'].values
Vp = 1/Dp*0.3048*1e6
```

```
#Assuming Total Isotropy

#I calculate dynamic Young modulus (in psi, for which I set dens in kg/cm3, which gives Pa,
# and then I convert Pa into Mpa, then into psi) and Poisson's ratio
E_d = np.round(dens*(1000) * Vs**2 *(3*Vp**2 - 4*Vs**2)/(Vp**2-Vs**2)/(1e6)*145.038,0) #psi
P_d = np.round((Vp**2 - 2*Vs**2)/(2*(Vp**2-Vs**2)),3)

#I calculate Static Young Modulus and Poisson' ratio
E_s = 0.65*E_d
P_s = P_d

data['Young (psi)'] = E_s
data['Poisson'] = P_s
```

```
PP = data['Res Press (psi)'].values
Biot = 0.9
```

```
#Compute total maximum and minimum horizontal stress assuming theory of elasticity

TectStrain_h = 0
TectStrain_H = 0.0015

Sh = P_s/(1-P_s)*(Sv - Biot*PP) + E_s/(1-P_s**2) *(TectStrain_h + P_s*TectStrain_H) + Biot*PP
SH = P_s/(1-P_s)*(Sv-Biot*PP) + E_s/(1-P_s**2) *(TectStrain_H + P_s*TectStrain_h) + Biot*PP

data['Sv (psi)'] = Sv
data['Sv (psi)'] = data['Sv (psi)'].cumsum()

data['Sh (psi)'] = Sh
data['SH (psi)'] = SH
```

Plotting the logs

```
Depth = 2100
```

```
# Initialise the subplot function using number of rows and columns
#figure, axis = plt.subplots(2, 2)

logs = data.columns.values
cols = data.shape[1]


fig,ax = plt.subplots(ncols=cols-3, figsize= (20,15), sharey=True)

for i in range(1, cols-3):
    ax[i-1].plot(data.iloc[:,i].values,data.iloc[:,0].values)
    ax[i-1].set_ylim(max(data.iloc[:,0]),min(data.iloc[:,0]))
    ax[i-1].minorticks_on()
    ax[i-1].grid(which='major',linestyle='-',linewidth='0.5',color='green')
    ax[i-1].grid(which='minor',linestyle=':',linewidth='0.5',color='black')
    ax[i-1].set_title('%s' %logs[i])

#    ax[i-1].plot([max(data.iloc[:,i]),min(data.iloc[:,i])],[Depth, Depth], color = 'red')

ax[cols-4].plot(data.iloc[:,cols-3].values,data.iloc[:,0].values, label = ('%s' %logs[cols-3]) )
ax[cols-4].plot(data.iloc[:,cols-2].values,data.iloc[:,0].values, label = ('%s' %logs[cols-2]))
ax[cols-4].plot(data.iloc[:,cols-1].values,data.iloc[:,0].values, label = ('%s' %logs[cols-1]))

ax[cols-4].set_ylim(max(data.iloc[:,0]),min(data.iloc[:,0]))
ax[cols-4].minorticks_on()
ax[cols-4].grid(which='major',linestyle='-',linewidth='0.5',color='green')
ax[cols-4].grid(which='minor',linestyle=':',linewidth='0.5',color='black')
ax[cols-4].set_title('Stresses (psi)')
ax[cols-4].legend(loc = True)
#ax[cols-4].plot([max(data.iloc[:,cols-3]),0],[Depth, Depth], color = 'red')
```

Calling previous functions from a different phyton notebook

```
# Do a definitions-only import

import ipynb.fs  # Boilerplate required

from .defs.Geomechanics_functions import Mohr
from .defs.Geomechanics_functions import Geo_rot
from .defs.Geomechanics_functions import Well_rot
from .defs.Geomechanics_functions import kirsch
from .defs.Geomechanics_functions import Tensile
```

Defining principal stresses directions and inputting into the circles function

```
#Normal Regime

Alpha =160
Beta = 90
Gamma = 0

#Strike-slip Regime

#Alpha =70
#Beta = 0
#Gamma = 90

#Reverse Regime

#Alpha =70
#Beta = 0
#Gamma = 0

#Stresses

Sh = round(float(data[data['DEPTH (ft)']==Depth]['Sh (psi)'].values),2) #MPa
SH = round(float(data[data['DEPTH (ft)']==Depth]['SH (psi)'].values),2) #MPa
Sv = round(float(data[data['DEPTH (ft)']==Depth]['Sv (psi)'].values),2) #MPa

Stresses = [Sh, SH, Sv]
S1 = max(Stresses)
S3 = min(Stresses)
S2 = Stresses[Stresses not in [S1, S3]]
PP = round(float(data[data['DEPTH (ft)']==Depth]['Res Press (psi)'].values),2) #psi
PW = PP #psi mud pressure
Poisson = 0.3
inc = surv['Inclination (°)'][Depth]
azi = surv['Azimuth (°)'][Depth]

s_tensor = np.array([[S1, 0 , 0], [0, S2, 0], [0, 0, S3]] )
```

```
Tensile(Alpha, Beta, Gamma, S1, S2, S3, PW, PP, Poisson, inc, azi)
```