WP1 Problem 1

Author: John D'Angelo

Fall 2018, University of Texas at Austin

Part a, Calculate average pore pressure (PPRS_3) gradient between depth A and B in [MPa/km] and [psi/ft]

In [12]:
```
import numpy as np

depthSpacing_Metric = 25 #meters
depthSpacing_Field = depthSpacing_Metric*3.28084 #ft

#Instead of estimating actual values along the axes of the plot,
#I estimated the spacing between the lines on the plot and converted
#that into values based on the known spacing.

pressureSpacing = (12000-6000)/5 #psi
PPRS_3_A_Field = 6000 + 1.25*pressureSpacing #psi
PPRS_3_B_Field = 6000 + 1.75*pressureSpacing #psi
distanceAB_Field = 8*depthSpacing_Field #meters to feet

PPRS_3_A_Metric = 6000 + 1.25*pressureSpacing*0.00689476 #psi to MPa
PPRS_3_B_Metric = 6000 + 1.75*pressureSpacing*0.00689476 #psi to MPa
distanceAB_Metric = 8*depthSpacing_Metric/1000 #meters to kilometers

porePressureGradientAB_Field = (PPRS_3_B_Field-PPRS_3_A_Field)/distanceAB_Field
porePressureGradientAB_Metric = (PPRS_3_B_Metric-PPRS_3_A_Metric)/distanceAB_Metric
```

In [13]:
```
print("Pore Pressure Gradient:")
print()
print(np.round(porePressureGradientAB_Field,3)," [psi/ft] \n")
print(np.round(porePressureGradientAB_Metric,3)," [MPa/km] \n")
```

```
Pore Pressure Gradient:

0.914  [psi/ft]

20.684  [MPa/km]
```

Part b, Calculate average vertical stress gradient between depth A and B

In [14]:
```
stressSpacing = pressureSpacing #psi
SigV_A_Field = 6000 + 2.4*stressSpacing #psi
SigV_B_Field = 6000 + 3.0*stressSpacing #psi

SigV_A_Metric = 6000 + 2.4*stressSpacing*0.00689476 #MPa
SigV_B_Metric = 6000 +3.0*stressSpacing*0.00689476 #MPa

verticalStressGradientAB_Field = (SigV_B_Field-SigV_A_Field)/distanceAB_Field
verticalStressGradientAB_Metric = (SigV_B_Metric-SigV_A_Metric)/distanceAB_Metric
```

In [15]:
```
print("Vertial Stress Gradient:")
print()
print(np.round(verticalStressGradientAB_Field,3)," [psi/ft] \n")
print(np.round(verticalStressGradientAB_Metric,3)," [MPa/km] \n")
```

```
Vertial Stress Gradient:

1.097  [psi/ft]

24.821  [MPa/km]
```

Part c, Calculate a reasonable guess for depth A Assuming this is a land formation (water depth=0)

In [16]:
```
# Tempting to use the pore pressure, but since we do not know the extent
# of the over pressure here, it may yeild bad results.
# depthA_guess1 = PPRS_3_A_Field/(porePressureGradientAB_Field)
#Instead I will just use SigV
depthA_guess2 = SigV_A_Field/(verticalStressGradientAB_Field)

depthA_Guess= depthA_guess2
print("Estimate of True Value of Depth A: \n")
print(np.round(depthA_Guess,3),' [ft]')
```

```
Estimate of True Value of Depth A:

8092.739  [ft]
```

Part d, Assuming vertical stress is principal stress, we need to write the stress tensor at depths A,B,C,D,E At each depth we will need, Sv,SHmax,Shmin. Since we get to assume vertical stress is principal, the following function will be convenient. Also, since it was not specified whether we should provide the effective or total stress tensor, I will be using the total stress tensor.

In [17]:
```
def stressTensor(Sv,Shmax,Shmin):
    sValues = [Sv,Shmax,Shmin]
    sValues = sorted(sValues,reverse=True)
    return np.array([[sValues[0],0,0],
                     [0,sValues[1],0],
                     [0,0,sValues[2]]])
```

In [19]:
```python
#To avoid too many variables, lets just store the values as dictionaries.
#I will work consistently in field units (psi and ft from this point on)
#Note that once again I used spacing and proportions to estimate values from the graph.

#Pore Pressure values at each depth
dictPp = {"A":PPRS_3_A_Field,
         "B":PPRS_3_B_Field,
         "C":PPRS_3_A_Field+1*depthSpacing_Field*porePressureGradientAB_Field,
         "D":PPRS_3_A_Field+4.1*depthSpacing_Field*porePressureGradientAB_Field,
         "E":PPRS_3_A_Field+8.2*depthSpacing_Field*porePressureGradientAB_Field}
#Total stress values at each depth
dictSv = {"A":SigV_A_Field,
         "B":SigV_B_Field,
         "C":6000 + 2.5*stressSpacing,
         "D":6000 + 2.7*stressSpacing,
         "E":6000 + 3*stressSpacing}
dictShmax = {"A":6000 + 3.1*stressSpacing,
         "B":6000 + 4.0*stressSpacing,
         "C":6000 + 3.25*stressSpacing,
         "D":6000 + 4.0*stressSpacing,
         "E":6000 + 4.5*stressSpacing}
dictShmin = {"A":6000 + 2.1*stressSpacing,
         "B":6000 + 2.75*stressSpacing,
         "C":6000 + 2.15*stressSpacing,
         "D":6000 + 3*stressSpacing,
         "E":6000 + 2.8*stressSpacing}


dictStressTensor = {}
for entry in dictSv.keys():
    dictStressTensor[entry]=stressTensor(dictSv[entry],
                                        dictShmax[entry],
                                        dictShmin[entry])
    print("Total Stress Tensor "+entry + " (in psi)")
    print(dictStressTensor[entry])
    print()
```

```
Total Stress Tensor A (in psi)
[[9720.    0.    0.]
 [   0. 8880.    0.]
 [   0.    0. 8520.]]

Total Stress Tensor B (in psi)
[[10800.     0.     0.]
 [    0.  9600.     0.]
 [    0.     0.  9300.]]

Total Stress Tensor C (in psi)
[[9900.    0.    0.]
 [   0. 9000.    0.]
 [   0.    0. 8580.]]

Total Stress Tensor D (in psi)
[[10800.     0.     0.]
 [    0.  9600.     0.]
 [    0.     0.  9240.]]

Total Stress Tensor E (in psi)
[[11400.     0.     0.]
 [    0.  9600.     0.]
 [    0.     0.  9360.]]
```

Part e, Classify the stress regime at depths A,B,C,D,E (Normal, Strike-slip, Reverse)

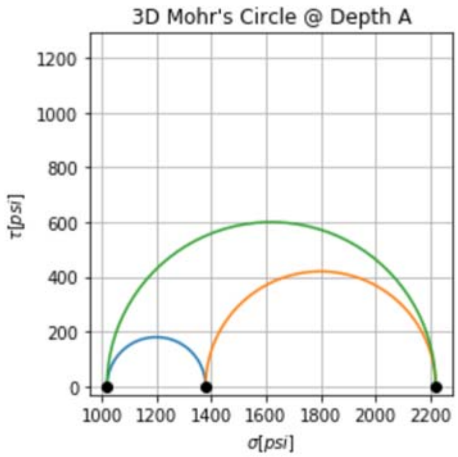
In [20]:
```python
#COMMENT: The minimum horizontal stress Shmin at DepthD appears
#to briefly jump above the vertical stress. Strictly speaking,
#the stress tensor at Depth D is indicative of reverse faulting (SH>Sh>Sv).
#However, since the rest of the formation is fairly consistently
#Strike-slip, this may have been the result of noise in the data.

def classifyRegime(Sv,Shmax,Shmin):
    if(Sv>Shmax and Shmax >Shmmin):
        return "Normal"
    elif(Shmax>Sv and Sv>Shmin):
        return "Strike-Slip"
    elif(Shmax>Shmin and Shmin>Sv):
        return "Reverse"
    else:
        print("Error: Potential issue with stresses provided")
        return "Bad Input"
Regime = {}
for entry in dictSv.keys():
    Regime[entry]=classifyRegime(dictSv[entry],
                                dictShmax[entry],
                                dictShmin[entry])
    print("Regime "+entry +": ")
    print(Regime[entry])
    print()
```

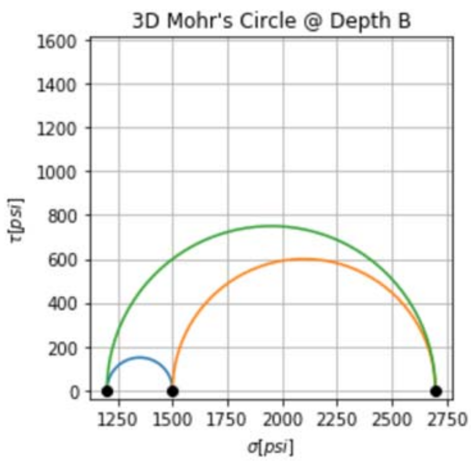
```
Regime A:
Strike-Slip

Regime B:
Strike-Slip

Regime C:
Strike-Slip

Regime D:
Reverse

Regime E:
Strike-Slip
```

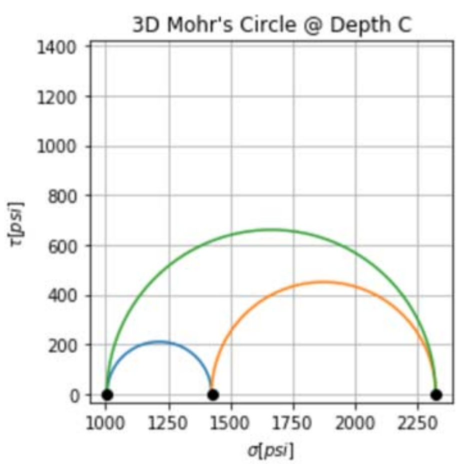
Part f, Plot 3D Mohr circles of **effective stresses** for A,B,C,D, E

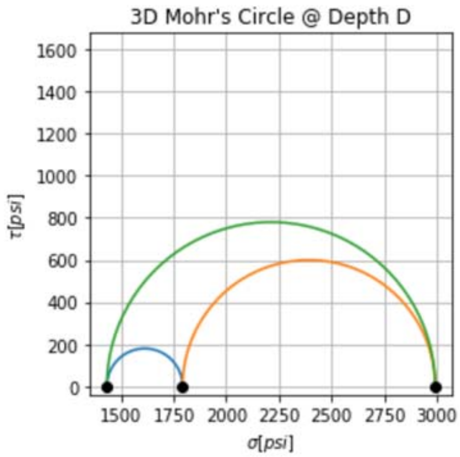
In [21]:

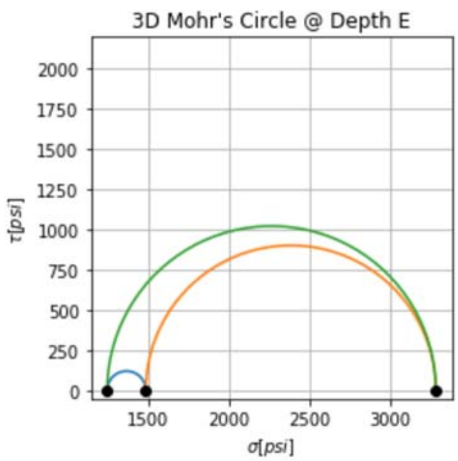
```
import matplotlib.pyplot as plt
%matplotlib inline

for entry in dictSv.keys():
    #Get effective stress, difference between total stress and pore
    #pressure at each depth
    sV=dictSv[entry]-dictPp[entry]
    sHmax=dictShmax[entry]-dictPp[entry]
    sHmin=dictShmin[entry]-dictPp[entry]
    #Determine sigma 1,2,3 where sig1>sig2>sig3. Here I used
    # a = sig3, b = sig2, c = sig1.
    arg = np.array([sV,sHmax,sHmin])
    a = np.min(arg)
    c = np.max(arg)
    arg = arg[np.where(arg!=a)]
    arg = arg[np.where(arg!=c)]
    b = arg[0]

    circle1X=[]
    circle1Y=[]
    circle2X=[]
    circle2Y=[]
    circle3X=[]
    circle3Y=[]

    for i in np.linspace(0,np.pi):
        circle1X.append((b-a)/2*np.cos(i) + (a+(b-a)/2))
        circle2X.append((c-b)/2*np.cos(i) + (b+(c-b)/2))
        circle3X.append((c-a)/2*np.cos(i) + (a+(c-a)/2))
        circle1Y.append((b-a)/2*np.sin(i) )
        circle2Y.append((c-b)/2*np.sin(i) )
        circle3Y.append((c-a)/2*np.sin(i) )
    fig, ax = plt.subplots()
    ax.plot(circle1X,circle1Y)
    ax.plot(circle2X,circle2Y)
    ax.plot(circle3X,circle3Y)
    ax.plot([a,b,c],[0,0,0],'ko')
    ax.grid()
    ax.set_xlabel(r'$\sigma [psi]$')
    ax.set_ylabel(r'$\tau [psi]$')
    ax.set_title("3D Mohr's Circle @ Depth "+entry)
    plt.axis('square')
    plt.tight_layout()
```


3D Mohr's Circle @ Depth A
τ[psi]
σ[psi]


3D Mohr's Circle @ Depth B
τ[psi]
σ[psi]


3D Mohr's Circle @ Depth C
τ[psi]
σ[psi]

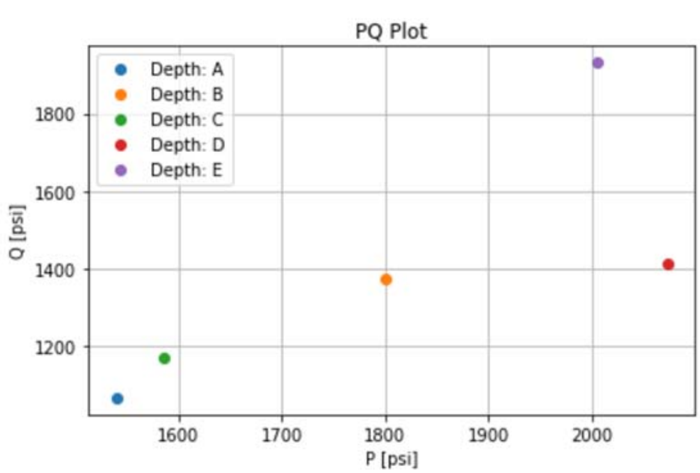




Part g, plot p-q poitns for A,B,C,D,E

In [22]:
```
import matplotlib.pyplot as plt
%matplotlib inline
fig, ax = plt.subplots()
pqOutput = []
for entry in dictSv.keys():
    #Get effective stress, difference between total stress and pore
    #pressure at each depth
    sV=dictSv[entry]-dictPp[entry]
    sHmax=dictShmax[entry]-dictPp[entry]
    sHmin=dictShmin[entry]-dictPp[entry]
    #Determine sigma 1,2,3 where sig1>sig2>sig3. Here I used
    # a = sig3, b = sig2, c = sig1.
    arg = np.array([sV,sHmax,sHmin])
    a = np.min(arg)
    c = np.max(arg)
    arg = arg[np.where(arg!=a)]
    arg = arg[np.where(arg!=c)]
    b = arg[0]

    p = (a+b+c)/3
    q = np.sqrt(3*(1/6)*((a-b)**2+(b-c)**2+(a-c)**2))
    pqOutput.append("Depth: " + entry + " \nP: " + str(np.round(p,2)) +" [psi]\nQ: "+st
r(np.round(q,2))+" [psi]")
    ax.plot(p,q,'o',label='Depth: '+entry)
    ax.grid()
    ax.set_xlabel('P [psi]')
    ax.set_ylabel('Q [psi]')
    ax.set_title("PQ Plot")

    plt.tight_layout()
ax.legend()
for item in pqOutput: print(item)
```

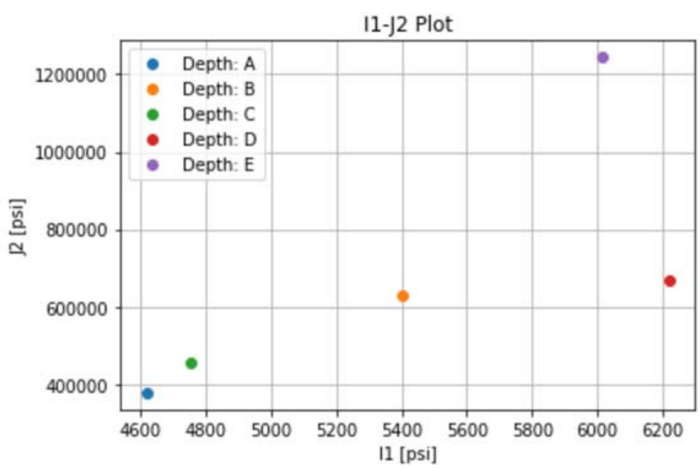
```
Depth: A
P: 1540.0 [psi]
Q: 1066.58 [psi]
Depth: B
P: 1800.0 [psi]
Q: 1374.77 [psi]
Depth: C
P: 1585.0 [psi]
Q: 1168.08 [psi]
Depth: D
P: 2072.5 [psi]
Q: 1414.78 [psi]
Depth: E
P: 2005.0 [psi]
Q: 1931.22 [psi]
```



Part h, Plot I1-J2 points for A,B,C,D,E

In [23]:
```
import matplotlib.pyplot as plt
%matplotlib inline
fig, ax = plt.subplots()
invariantOutput = []
for entry in dictSv.keys():
    #Get effective stress, difference between total stress and pore
    #pressure at each depth
    sV=dictSv[entry]-dictPp[entry]
    sHmax=dictShmax[entry]-dictPp[entry]
    sHmin=dictShmin[entry]-dictPp[entry]
    #Determine sigma 1,2,3 where sig1>sig2>sig3. Here I used
    # a = sig3, b = sig2, c = sig1.
    arg = np.array([sV,sHmax,sHmin])
    a = np.min(arg)
    c = np.max(arg)
    arg = arg[np.where(arg!=a)]
    arg = arg[np.where(arg!=c)]
    b = arg[0]

    I1 = (a+b+c)
    J2 = (1/6)*((a-b)**2+(b-c)**2+(a-c)**2)
    invariantOutput.append("Depth: " + entry + " \nI1: " + str(np.round(I1,2)) +" [ps
i]\nJ2: "+str(np.round(J2,2))+" [psi]")

    ax.plot(I1,J2,'o',label='Depth: '+entry)
    ax.grid()
    ax.set_xlabel('I1 [psi]')
    ax.set_ylabel('J2 [psi]')
    ax.set_title("I1-J2 Plot")

    plt.tight_layout()
ax.legend()
for item in invariantOutput: print(item)
```

```
Depth: A
I1: 4620.0 [psi]
J2: 379200.0 [psi]
Depth: B
I1: 5400.0 [psi]
J2: 630000.0 [psi]
Depth: C
I1: 4755.0 [psi]
J2: 454800.0 [psi]
Depth: D
I1: 6217.5 [psi]
J2: 667200.0 [psi]
Depth: E
I1: 6015.0 [psi]
J2: 1243200.0 [psi]
```



Part i, In which direction would a hydraulic fracture open-up in the interval under study in this formation? Justify.

The majority of the area under study in this formation is strike-slip. This means that the fracture will most likely occur vertically and perpendicular to the direction of Shmin (minimum horizontal stress). The orientation perpendicular to Shmin is the path of least resistance for a fracture since Shmin is the lowest principal stress in a Strike-slip regime.

It is nice to visualize this by pressing your hands together. If you push your hands together, the pressure you are applying is like a principal stress holding a rock formation together. If you kept pushing and someone came up and pushed one of your hands, your hand "formation" would separate and slide ("fracture") perpendicular to the direction in which you were pressing.