CE 5364 Groundwater Transport Phenomena
Fall 2025 Exercise Set 4

LAST NAME, FIRST NAME

R00000000

Purpose :

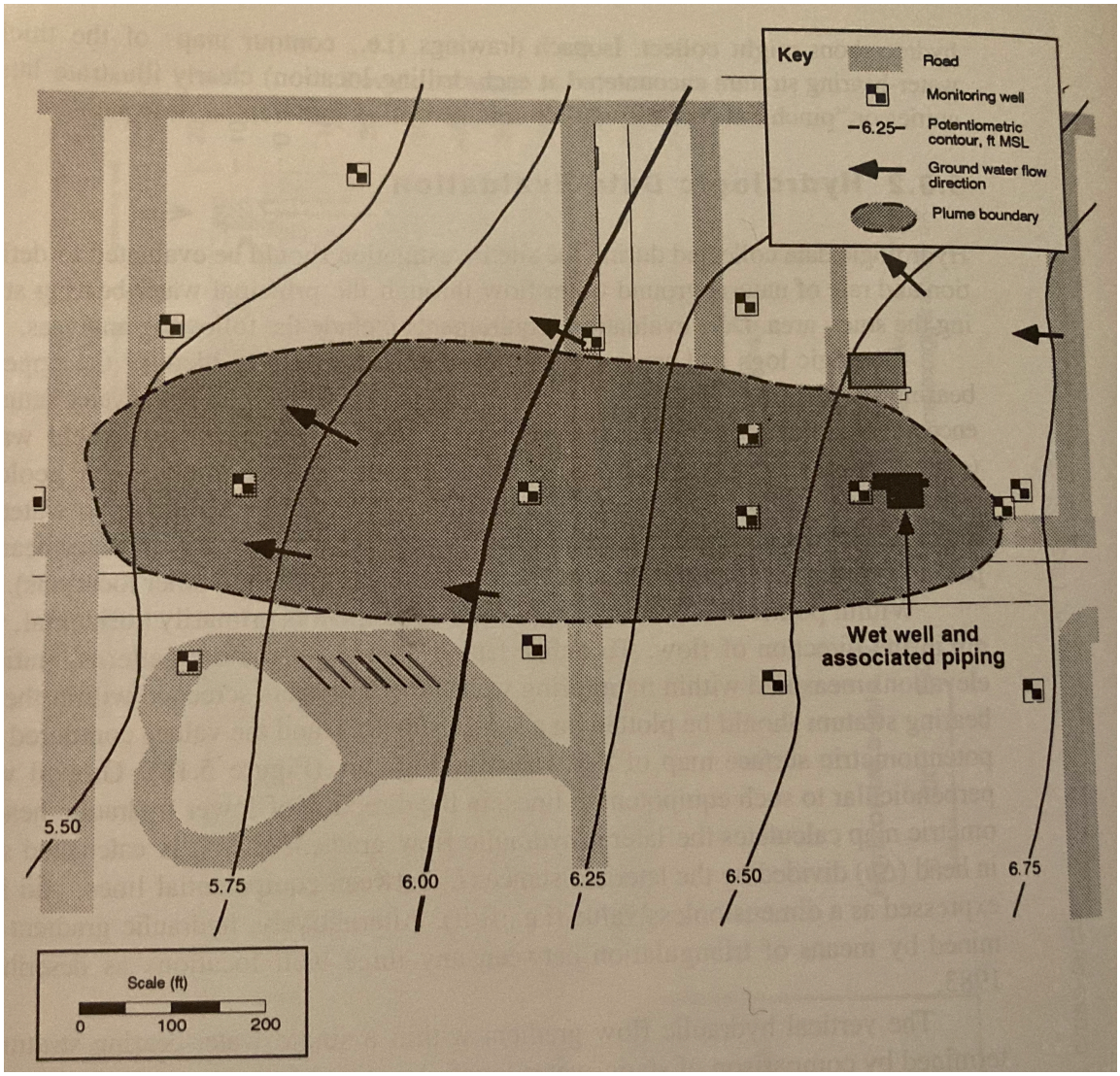
Apply advection principles to quantitative cases

Assessment Criteria :

Completion, results plausible, format correct, example calculations shown.

Problem 1 (Problem 2-8, pg. 578)

The figure below shows a piezometric map for a shallow sand aquifer. The hydraulic conductivtiy is estimated to be $1.5 \times 10^{-2}\ \frac{cm}{s}$, the saturated thickness is 40 feet, and the effective porosity is 0.3.



Determine:

1. Which well is expected to be the most contaminated.
2. The groundwater velocity and seepage velocity across the plume.
3. The duration that the source has been contaminating the aquifer (neglect dispersion, diffusiom, and adsorption).
4. The flow rate across the plume.
5. An explaination for contamination upgradient of the source zone.

sketch(s)

list known quantities

- Head map (provided)
- $b \approx 40\ feet$

- $K \approx 1.5 \times 10^{-2} \frac{cm}{s} \cdot \frac{1\ in}{2.54\ cm} \cdot \frac{1\ ft}{12\ in} \cdot \frac{86400\ s}{1\ day} = 42.52 \frac{ft}{day}$
- $n \approx 0.30$

list unknown quantities

1. Which well is expected to be the most contaminated.
2. The groundwater velocity and seepage velocity across the plume.
3. The duration that the source has been contaminating the aquifer (neglect dispersion, diffusiom, and adsorption).
4. The flow rate across the plume.
5. An explaination for contamination upgradient of the source zone.

governing principles

1. Darcy's law; $Q = KA\frac{dh}{dl}$
2. Definition(s) of discharge and seepage velocity

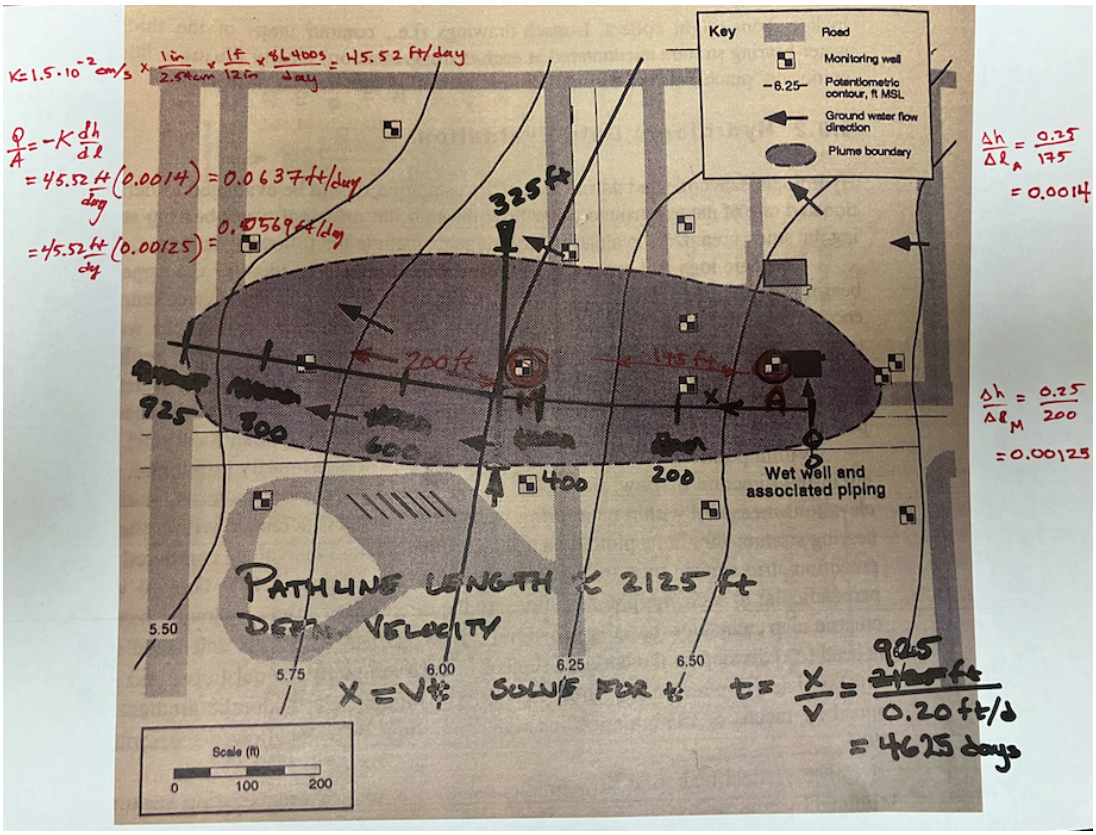
solution details (e.g. step-by-step computations)

1. Which well is expected to be the most contaminated.

- If release is ongoing, probably well A in annotated map below.
- If release is "past history" then probably well M near middle of the plume.

2. The groundwater velocity and seepage velocity across the plume.

- See annotated map below.



- Gradients near well A and well M are calculated directly on the map annotations.
- Specific discharge values:
  $q_{near\ well\ A} = \frac{Q}{A}_{near\ well\ A} \approx 45.52\frac{ft}{day}(0.0014) = 0.0637\frac{ft}{day}$
  $q_{near\ well\ M} = \frac{Q}{A}_{near\ well\ M} \approx 45.52\frac{ft}{day}(0.00125) = 0.0569\frac{ft}{day}$
  A reasonable average is $q = \frac{Q}{A}_{average} \approx 0.06\frac{ft}{day}$
- Species (seepage) velocity is
  $U = \frac{q}{n} = \frac{1}{n}\frac{Q}{A} = \frac{1}{0.3}0.06\frac{ft}{day} = 0.2\frac{ft}{day}$

3. The duration that the source has been contaminating the aquifer (neglect dispersion, diffusiom, and adsorption).

- Choose a reasonable pathline like the one on the drawing below and measure its length, then using definition of velocity, solve for duration.
  Pathline is $\approx 925\ feet$
  $t = \frac{x}{U} = \frac{925}{0.02} = 4625\ days$

4. The flow rate across the plume.

- Apply Darcy's law
  $Q = qA = 0.06\frac{ft}{day} \cdot 325ft \cdot 40ft = 780\frac{ft^3}{day}$

5. An explaination for contamination upgradient of the source zone.

- Dispersion/diffusion of constituients, and/or
- The regional water table is not static and at some point in past, gradient was in Easterly direction.

discussion

- The non-static water table is very likely, and quarterly monitoring is a usual standard to detect and account for seasonal/temporal variability.
- We will revisit this problem later on when we discuss advection modeling.

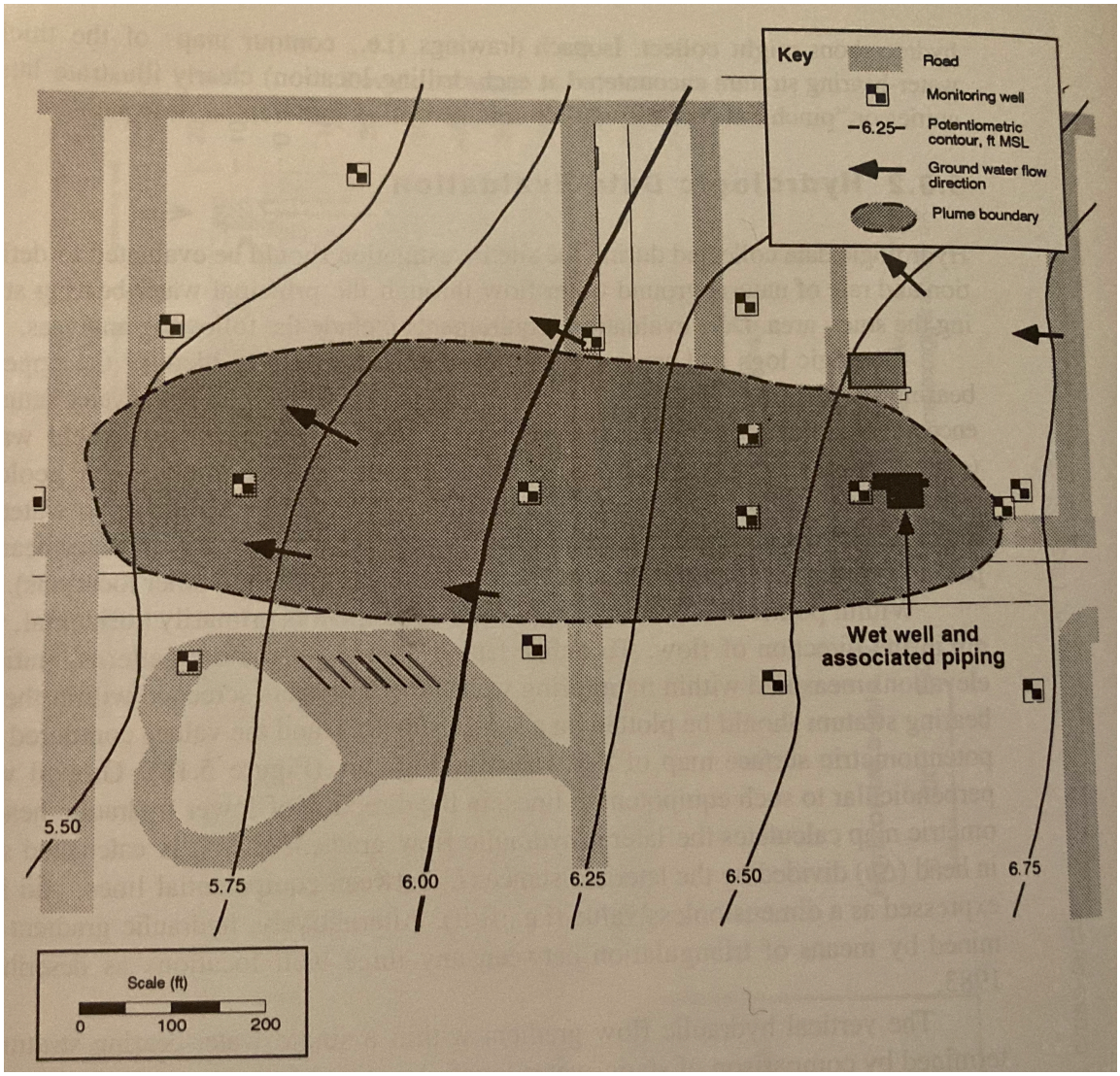
Addendum

This problem could also be addressed using a programmatic approach as in lecture notes (repeated below). When using a progrmmatic approach we anticipate a different result, but should be close.

Particle Tracking in Python

Here we illustrate using particle tracking to address a homework problem.

The figure below shows a piezometric map for a shallow sand aquifer. The hydraulic conductivtiy is estimated to be $1.5 \times 10^{-2} \ \frac{cm}{s}$, the saturated thickness is 40 feet, and the effective porosity is 0.3.



Determine:

1. Which well is expected to be the most contaminated.
2. The groundwater velocity and seepage velocity across the plume.
3. The duration that the source has been contaminating the aquifer (neglect dispersion, diffusiom, and adsorption).
4. The flow rate across the plume.
5. An explaination for contamination upgradient of the source zone.

Step 1. Need to Digitize the Head Map

Use G3DATA or something similar to digitize the head map.

Step 2 Check the Digitization Results

Here we will now produce a contour map from the digitized map, adjust sizes and overlay on the original map to get an idea of how good we did.

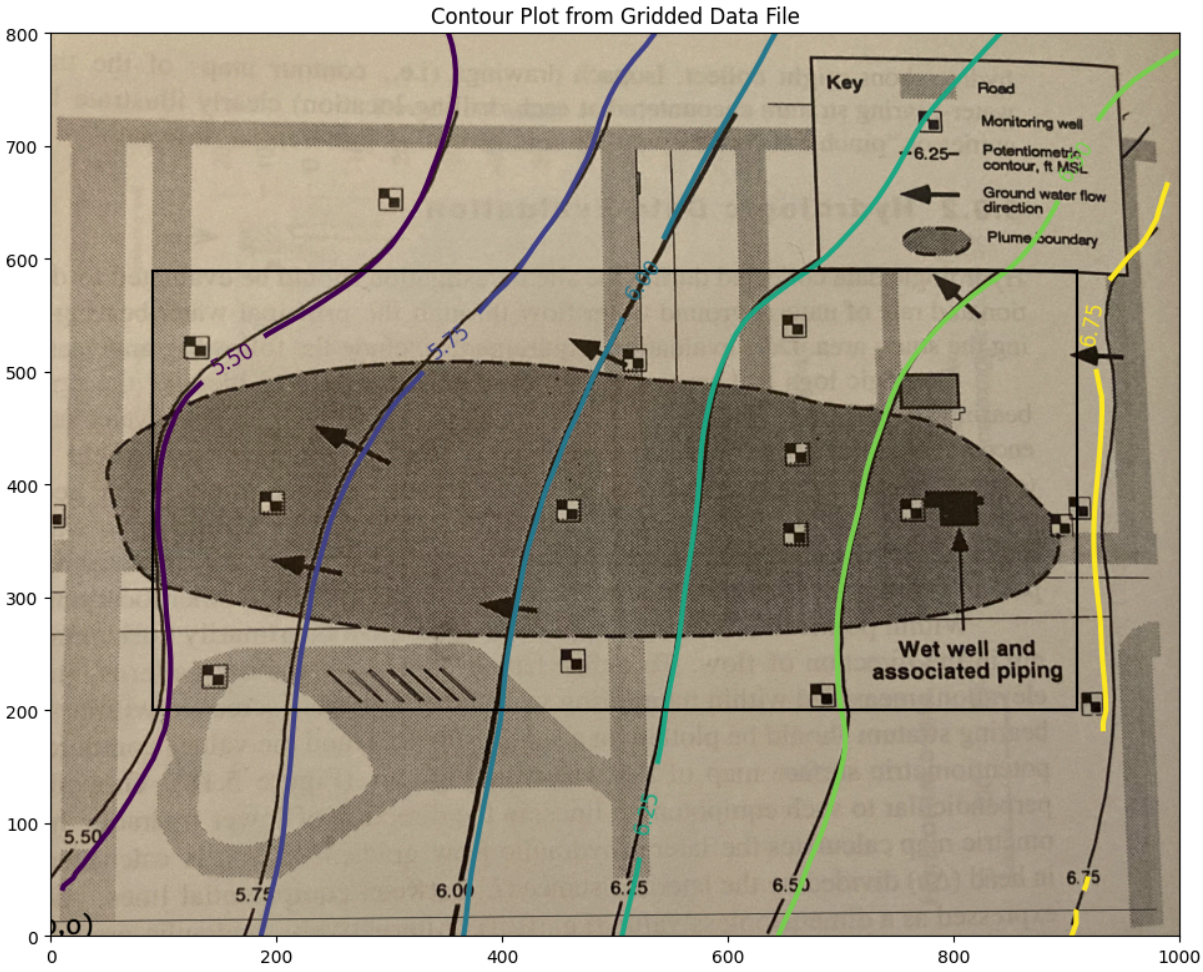
Also will use trial-and-error to find a useable hull within the contour plot where we can capture heads and use for a particle tracking effort - we have to find a rectangle with non-null interpolation values; easiest way (my opinion) is to try indices until can find a reasonable rectangle then sample from that rectangle. The process is documented below.

:::{note} The interpolation algorithm does not estimate vales outside the convex hull defined by the observation data. So the box below is by trial-and-error to stay within the convex hull; the interpolator in this region produces useable estimates for computing the heads and velocities from just the mapped information> :::

In [10]:
```
# CCMR from ENGR-1330:
# http://54.243.252.9/engr-1330-webroot/8-Labs/Lab07/Lab07.html
# https://clouds.eos.ubc.ca/~phil/docs/problem_solving/06-Plotting-with-Matplotlib/
# https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.griddata.h
# https://stackoverflow.com/questions/332289/how-do-you-change-the-size-of-figures-
# https://stackoverflow.com/questions/18730044/converting-two-lists-into-a-matrix
# https://stackoverflow.com/questions/3242382/interpolation-over-an-irregular-grid
# https://stackoverflow.com/questions/33919875/interpolate-irregular-3d-data-from-a
import pandas
my_xyz = pandas.read_csv('Fig5.18-LevelSets.png.dat',sep='\t') # read an ascii file
#my_xyz = pandas.read_csv('XYZSomewhereUSA.txt',sep=' ') # read an ascii file alrea
my_xyz = pandas.DataFrame(my_xyz) # convert into a data frame
#print(my_xyz) #examine the dataframe
import numpy
import matplotlib.pyplot
from scipy.interpolate import griddata
# extract lists from the dataframe
coord_x = my_xyz['X-Easting'].values.tolist()
coord_y = my_xyz['Y-Northing'].values.tolist()
coord_z = my_xyz['Z-Elevation'].values.tolist()
coord_xy = numpy.column_stack((coord_x, coord_y))
# Set plotting range in original data units
lon = numpy.linspace(min(coord_x), max(coord_x), 100)
lat = numpy.linspace(min(coord_y), max(coord_y), 100)
X, Y = numpy.meshgrid(lon, lat)
# Grid the data; use cubic spline interpolation (other choices are nearest and line
Z = griddata(numpy.array(coord_xy), numpy.array(coord_z), (X, Y), method='cubic', f
# Build the map
print("Indices of black box on overlay below:")
ixl = 9
ixh = 90
iyl = 27
iyh = 62
print("xyz lower left corner ",X[iyl][ixl],Y[iyl][ixl],Z[iyl][ixl])
print("xyz lower right corner",X[iyl][ixh],Y[iyl][ixh],Z[iyl][ixh])
print("xyz upper left corner ",X[iyh][ixl],Y[iyh][ixl],Z[iyh][ixl])
print("xyz upper right corner ",X[iyh][ixh],Y[iyh][ixh],Z[iyh][ixh])
xxl = X[iyl][ixl]
xxh = X[iyh][ixh]
yyl = Y[iyl][ixl]
yyh = Y[iyh][ixh]
flag=True
if flag:
```

```
    matplotlib.pyplot.rcParams["figure.figsize"] = [10.00, 8.00]
    matplotlib.pyplot.rcParams["figure.autolayout"] = True
    im = matplotlib.pyplot.imread("Fig5.18-Coordinates.png") # base image
fig, ax = matplotlib.pyplot.subplots()
if flag:
    im = ax.imshow(im, extent=[-25, 1000, -125, 800])# sets X and Y plot window of
#fig.set_size_inches(14, 7)
levels=[5.50,5.75,6.00,6.25,6.50,6.75]
CS = ax.contour(X, Y, Z, levels, linewidths=3)
ax.clabel(CS, inline=2, fontsize=12)
ax.set_title('Contour Plot from Gridded Data File')
ax.set_xlim([0,1000])
ax.set_ylim([0,800])
ax.plot([xxl,xxh],[yyl,yyl],color="black")
ax.plot([xxl,xxh],[yyh,yyh],color="black")
ax.plot([xxl,xxl],[yyl,yyh],color="black")
ax.plot([xxh,xxh],[yyl,yyh],color="black");
```

```
Indices of black box on overlay below:
xyz lower left corner  90.9090909090909 200.0 5.478952367771485
xyz lower right corner 909.090909090909 200.0 6.7224489855441645
xyz upper left corner  90.9090909090909 588.8888888888889 5.302942021834025
xyz upper right corner  909.090909090909 588.8888888888889 6.663235234576626
```



This is acceptable. Now we will extract velocities from the gridded data files - using the "box". Recall to use the spreadsheet version of the particle tracker we need the U and V fields from the head distribution.

We will use the supplied hydraulic conductivity and porosity

- $K \approx 1.5 \times 10^{-2} \frac{cm}{s} \cdot \frac{1\ in}{2.54\ cm} \cdot \frac{1\ ft}{12\ in} \cdot \frac{86400\ s}{1\ day} = 42.52 \frac{ft}{day}$
- $n \approx 0.30$

And apply Darcy's law as

- $\mathbf{u(x,y)} = -\frac{K}{n} \cdot \frac{\Delta h}{\Delta x}$
- $\mathbf{v(x,y)} = -\frac{K}{n} \cdot \frac{\Delta h}{\Delta y}$

In [11]:
```
Conductivity = 42.52 # feet/day
Porosity = 0.30 # voids/bulk
nrows = iyh - iyl
ncols = ixh - ixl
#Zero vectors for the velocity fields
U = [[0 for j in range(ncols)] for i in range(nrows)]
V = [[0 for j in range(ncols)] for i in range(nrows)]
#Zero vectors for realinged mesh grid
XG = [[0 for j in range(ncols)] for i in range(nrows)]
YG = [[0 for j in range(ncols)] for i in range(nrows)]
for j in range(ncols):
    for i in range(nrows): #range(nrows)
        U[i][j] = -1.0*(Conductivity/Porosity)*(Z[iyl+i][ixl+j]-Z[iyl+i][ixl+j-1])/
        V[i][j] = -1.0*(Conductivity/Porosity)*(Z[iyl+i][ixl+j]-Z[iyl+i-1][ixl+j])/
        XG[i][j] = X[iyl+i][ixl+j]
        YG[i][j] = Y[iyl+i][ixl+j]
```

Suppose we want a plot of the vector field just to check our work, a nice tool is `quiver` and/or `streamplot` which are demonstrated below

In [12]:
```
# convert to numpy arrays
UU = numpy.asarray(U)
VV = numpy.asarray(V)
XX = numpy.asarray(XG)
YY = numpy.asarray(YG)
```

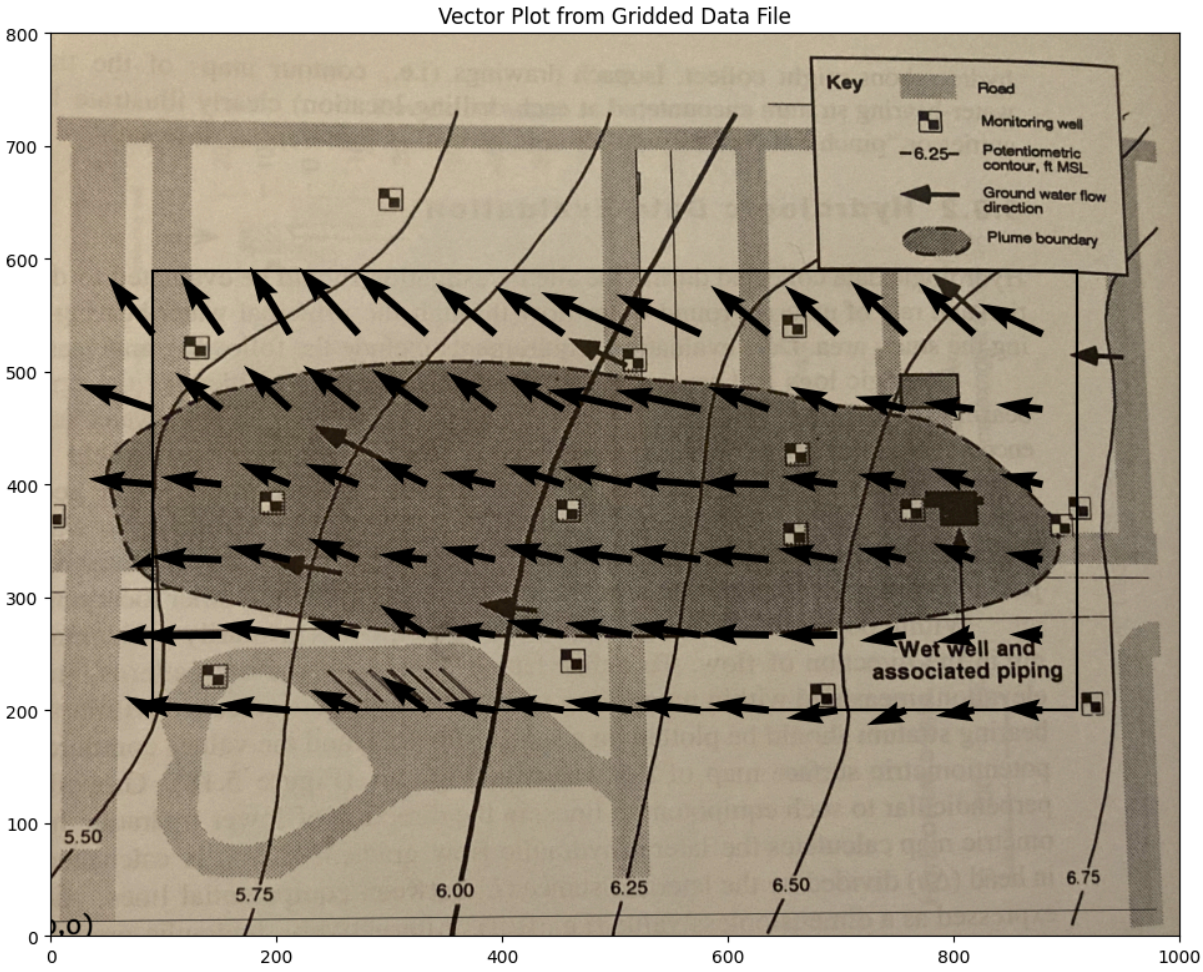
In [13]:
```
UU[30][30]
```

Out[13]:
```
np.float64(-0.2250452891747141)
```

Now a cool "quiver" plot

In [14]:
```
if flag:
    matplotlib.pyplot.rcParams["figure.figsize"] = [10.00, 8.00]
    matplotlib.pyplot.rcParams["figure.autolayout"] = True
    im = matplotlib.pyplot.imread("Fig5.18-Coordinates.png") # base image
fig, ax = matplotlib.pyplot.subplots()
if flag:
    im = ax.imshow(im, extent=[-25, 1000, -125, 800])# sets X and Y plot window of
#fig.set_size_inches(14, 7)
```

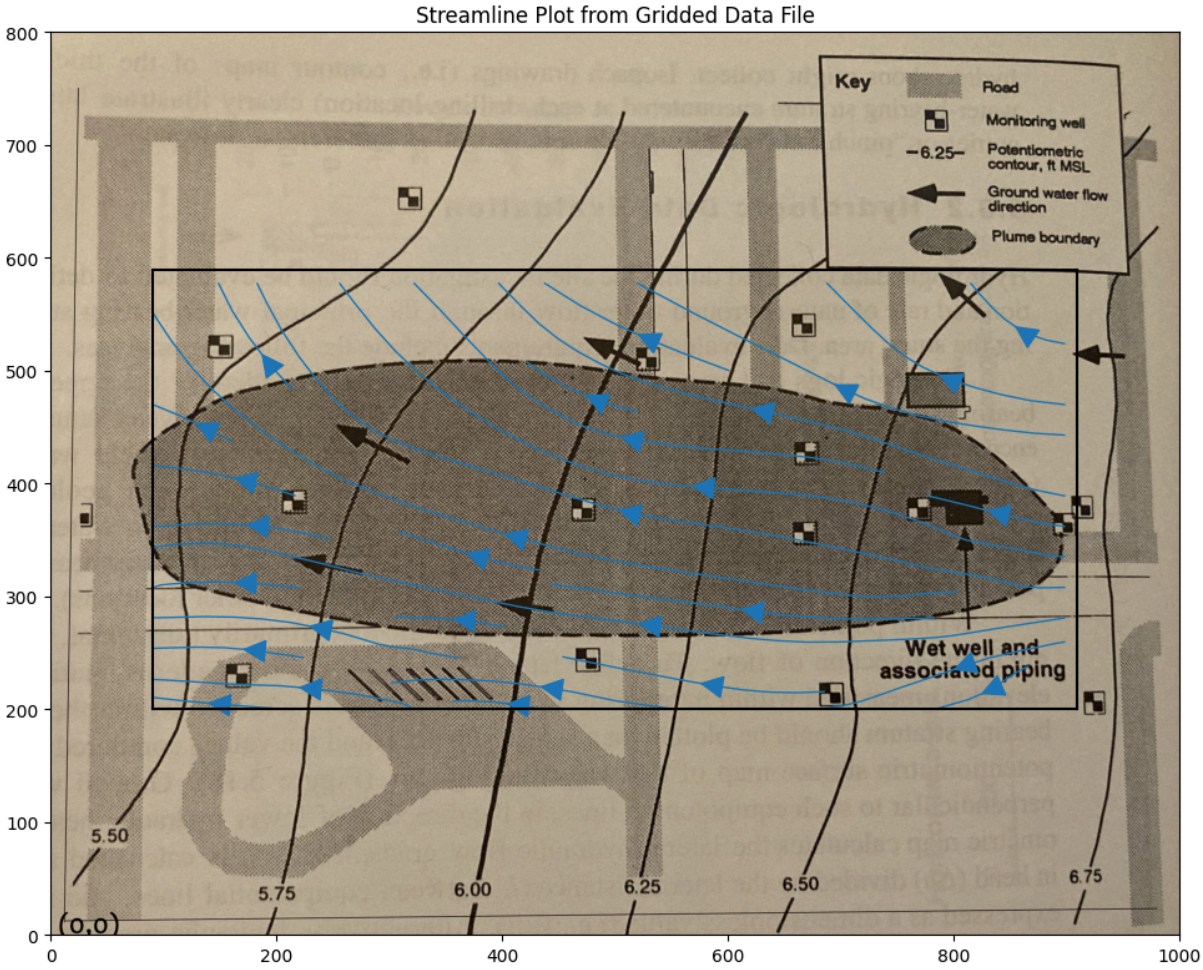
```
levels=[5.50,5.75,6.00,6.25,6.50,6.75]
CS = ax.quiver(XX[::6, ::6], YY[::6, ::6], UU[::6, ::6], VV[::6, ::6], units='width
#ax.clabel(CS, inline=2, fontsize=12)
ax.set_title('Vector Plot from Gridded Data File')
ax.set_xlim([0,1000])
ax.set_ylim([0,800]);
ax.plot([xxl,xxh],[yyl,yyl],color="black")
ax.plot([xxl,xxh],[yyh,yyh],color="black")
ax.plot([xxl,xxl],[yyl,yyh],color="black")
ax.plot([xxh,xxh],[yyl,yyh],color="black");

#matplotlib.pyplot.quiver(XX[::6, ::6], YY[::6, ::6], UU[::6, ::6], VV[::6, ::6], u
#matplotlib.pyplot.streamplot(XX, YY, UU, VV, density=0.5, linewidth=2, color=None)
```



In [15]:

```
if flag:
    matplotlib.pyplot.rcParams["figure.figsize"] = [10.00, 8.00]
    matplotlib.pyplot.rcParams["figure.autolayout"] = True
    im = matplotlib.pyplot.imread("Fig5.18-Coordinates.png") # base image
fig, ax = matplotlib.pyplot.subplots()
if flag:
    im = ax.imshow(im, extent=[0, 1000, -125, 800])# sets X and Y plot window of ba
#fig.set_size_inches(14, 7)
levels=[5.50,5.75,6.00,6.25,6.50,6.75]
CS = ax.streamplot(XX, YY, UU, VV,  density=0.5, linewidth=1, color=None, arrowsize
#ax.clabel(CS, inline=2, fontsize=12)
ax.set_title('Streamline Plot from Gridded Data File')
```

```
ax.set_xlim([0,1000])
ax.set_ylim([0,800]);
ax.plot([xxl,xxh],[yyl,yyl],color="black")
ax.plot([xxl,xxh],[yyh,yyh],color="black")
ax.plot([xxl,xxl],[yyl,yyh],color="black")
ax.plot([xxh,xxh],[yyl,yyh],color="black");
```



Now we can implement a particle tracking script. Using the spreadsheet as a guide we can write it in python, and access the cool graphics.

Here is the basic script, one simply has to wrap it into a time-stepping loop to produce a trajectory. Multiple particles can be managed

In [28]:
```
# U array x-velocity at XG,YG
# V array y-velocity at XG,YG
# XG array X-value of cell center
# YG array Y-value of cell center
# XP X-value of particle position
# YP Y-value of particle position
# UP x-velocity of particle
# VP y-velocity of particle
# TX x-component particle trajectory
# TY y-component particle trajectory

import math
```

```
verbose=False
terse=False
deltaT = 100
etime = 0
numTime = 38
XP = []
YP = []
UP = []
VP = []
TX = [] #trajectory vector
TY = []
np = 1 # Total particles
XP.append(850)
YP.append(300)
UP.append(0)
VP.append(0)
ip=np-1
print("                    Initial Particle Position",round(XP[ip],2),round(YP[ip],2

# move particles this time step
for it in range(numTime):
    for ip in range(np):
    # Build Particle Distance Table
        dist = []
        index = []
        count = 0
        for j in range(ncols):
            for i in range(nrows): #range(nrows)
                dist.append(math.sqrt((XX[i][j]-XP[ip])**2 + (YY[i][j]-YP[ip])**2))
                index.append([count,i,j]) # use to find i,j for a given index
                count = count +1
    # find closest cell
        for i in range(count):
            if dist[i] <= min(dist):
                #print(index[i],dist[i])
                ixx=index[i][1]
                jyy=index[i][2]
    # use nearest cell assignment - aka simple scheme
                UP[ip] = UU[ixx][jyy]
                VP[ip] = VV[ixx][jyy]
                if verbose: print("Particle Position and Velocities Before Move",ro
                break #exits the loop - we stop at the first nearest cell encounter
    # move the particle
        XP[ip]=XP[ip]+UP[ip]*deltaT
        YP[ip]=YP[ip]+VP[ip]*deltaT
        etime=etime+deltaT
        if terse: print(" Particle Position and Velocities After Move",round(XP[ip]
        TX.append([ip,XP[ip],etime])
        TY.append([ip,YP[ip],etime])
```

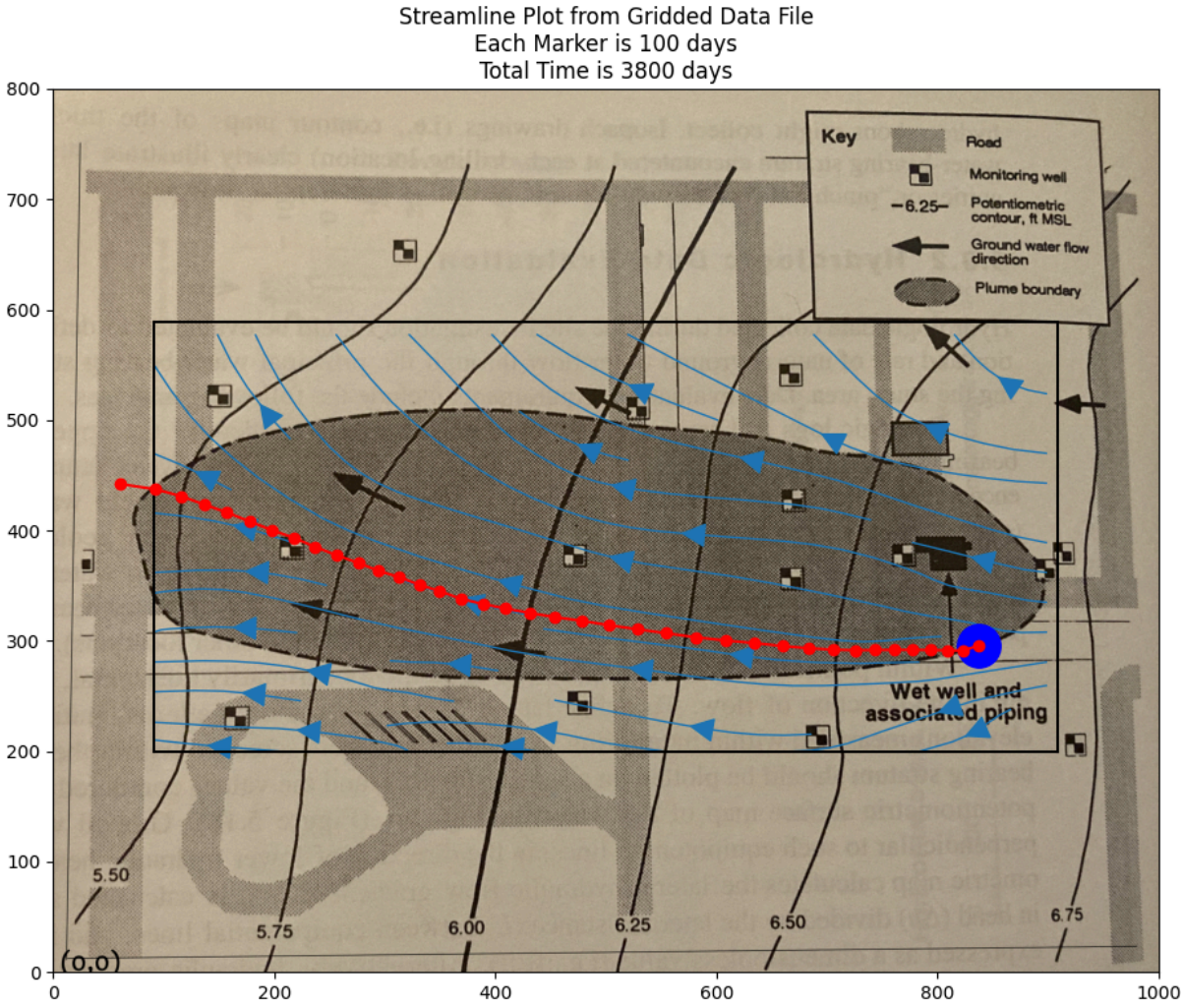
```
Initial Particle Position 850 300 0 0 0
```

In [29]:

```
if flag:
    matplotlib.pyplot.rcParams["figure.figsize"] = [10.00, 8.00]
    matplotlib.pyplot.rcParams["figure.autolayout"] = True
    im = matplotlib.pyplot.imread("Fig5.18-Coordinates.png") # base image
```

```
fig, ax = matplotlib.pyplot.subplots()
if flag:
    im = ax.imshow(im, extent=[0, 1000, -125, 800])# sets X and Y plot window of ba
#fig.set_size_inches(14, 7)
levels=[5.50,5.75,6.00,6.25,6.50,6.75]
CS = ax.streamplot(XX, YY, UU, VV,  density=0.5, linewidth=1, color=None, arrowsize
#ax.clabel(CS, inline=2, fontsize=12)
ax.set_title('Streamline Plot from Gridded Data File\n' +\
             'Each Marker is ' + str(deltaT) + ' days\n' +\
             'Total Time is ' + str(etime) + ' days')
ax.set_xlim([0,1000])
ax.set_ylim([0,800]);
ax.plot([xxl,xxh],[yyl,yyl],color="black")
ax.plot([xxl,xxh],[yyh,yyh],color="black")
ax.plot([xxl,xxl],[yyl,yyh],color="black")
ax.plot([xxh,xxh],[yyl,yyh],color="black")
xtrajectory = [sublist[1] for sublist in TX]
ytrajectory = [sublist[1] for sublist in TY]
ax.plot(xtrajectory[0],ytrajectory[0],marker="o",color="blue",markersize=24)
ax.plot(xtrajectory,ytrajectory,marker="o",color="red")
;
```

Out[29]: ''



In [ ]:

In [ ]: