Solve the 2-D Laplace in Excel

I created a 28 by 28 grid of the 2-D Laplace Equation. I included three internal "boundary values"; one high value of 4 and two low values of -2 and -3. The two low values were near each other compared to their respective distances to the high value. I allowed excel to iteratively calculate for 10,000 iterations with a minimum change of 0.0001. I saved the file as a CSV file after including explicit zeros surrounding the formulas. The dimensions of my data were 30 by 30. I rounded to four significant digits to see if it would mean that the stagnation areas would be more pronounced.

Read in and plot contours

I tried working on this part of the assignment in R, but I ran short on time. Instead, I pivoted to running python inside of my R Studio Project. This first chunk of code is mostly commented out. I may consider returning to this. Since much of my work is in R, I think it might still be useful for me to bring the python code and examples into my Tidyverse and R for Data Science knowledge foundation.

```
# Load libraries
library(plot3D)
library(ggplot2)
library(reticulate)

# Read in csv file
h = as.matrix(read.csv("tripole.csv", header = FALSE, row.names = NULL))

# Create vectors
x_vec <- seq(-1.5, 1.4, by = 0.1)
y_vec <- seq(-1.5, 1.4, by = 0.1)

# Create "np.meshgrid"
X <- matrix(rep(x_vec, each = length(y_vec)), nrow = length(y_vec))
Y <- matrix(rep(y_vec, times = length(x_vec)), nrow = length(y_vec))

# # Plot the surface
# surf3D(
#   x = X,
#   y = Y,
#   z = h,
#   bty = "b2",
#   ticktype = "detailed",
#   phi = 20, theta = 30
```

```
# )
#
# # Plot the contour
# library(ggplot2)
#
# # Example: compute gradients with finite differences
# dhdx <- apply(h, 1, diff)
# dhdx <- cbind(dhdx, NA)
#
# dhdy <- apply(h, 2, diff)
# dhdy <- rbind(dhdy, NA)
#
# # Build data frame
# dat <- expand.grid(x = x_vec, y = y_vec)
# dat$h <- c(h)
# dat$dhdx <- c(dhdx)
# dat$dhdy <- c(dhdy)
#
# dat <- expand.grid(x = x_vec, y = y_vec)
# dat$h <- c(h)
# dat$dhdx <- c(dhdx)
# dat$dhdy <- c(dhdy)
#
# ggplot(dat, aes(x, y)) +
#   geom_contour_filled(aes(z = h)) +
#   theme_minimal()
```

Switch to Python

Switch to using python to create Figure 1, Figure 2 and Figure 3. I did not identify any stagnation points. My low values were close together, so a saddle effect didn't appear as it would if they were spaced farther apart.

```
# Import packages
import numpy as np
import matplotlib.pyplot as plt

# Load csv file from excel
h=np.loadtxt('tripole.csv',delimiter=',')

# Create a grid of x and y coordinates
```

```
x_vec=np.linspace(-1.5, 1.4, 30)
y_vec=np.linspace(-1.5, 1.4, 30)
X,Y = np.meshgrid(x_vec,y_vec)

# Calculate gradient/partial derivatives
[dhdy,dhdx]=np.gradient(h,x_vec,y_vec)
```

Surface plot

```
fig = plt.figure(figsize=[4,4],dpi=300)
ax = plt.axes(projection = '3d')
ax.set_title(" " * 20 + 'Surface plot using Python'+ " " * 20)
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_zlabel('H-axis', rotation=90)
ax.zaxis.labelpad=-0.7
ax.tick_params(axis='x', labelsize=8)
ax.tick_params(axis='y', labelsize=8)
ax.tick_params(axis='z', labelsize=8)
plt.tight_layout()
surf = ax.plot_surface(X,Y,h)
plt.show()
```

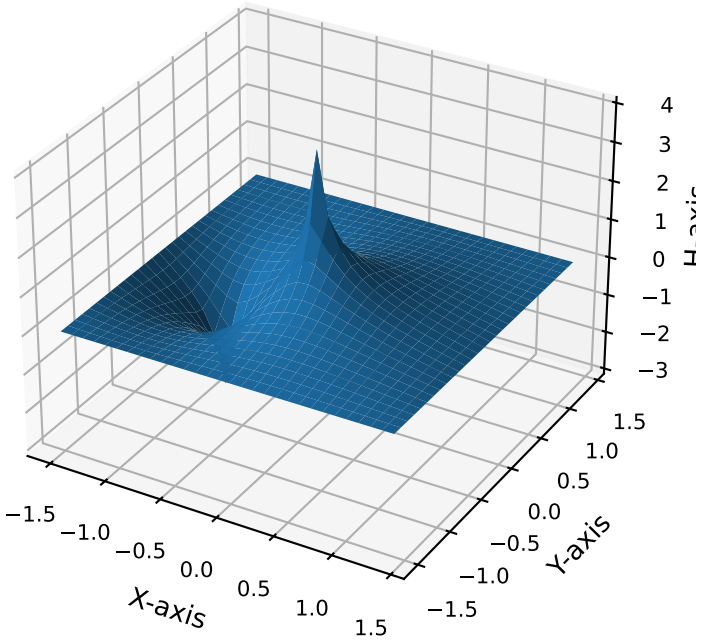

Figure 1: Plot of vector arrows using Python. The vectors indicate strength and direction of the negative gradient. The vectors are displayed over the contours of a tri-pole solution with a high value of 4 and lows of -2 and -3. The two low values were near each other compared to their respective distances to the high value.

```
plt.close('all')
```

Plot contour map and flow vectors

```
plt.contourf(X,Y,h)
cbar=plt.colorbar()
cbar.set_label('Ground water potential surface (h)')
plt.axis('equal');
qplt=plt.quiver(X,Y,-dhdx,-dhdy, scale=360)
plt.title('Contour map and flow vectors')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.show()
```

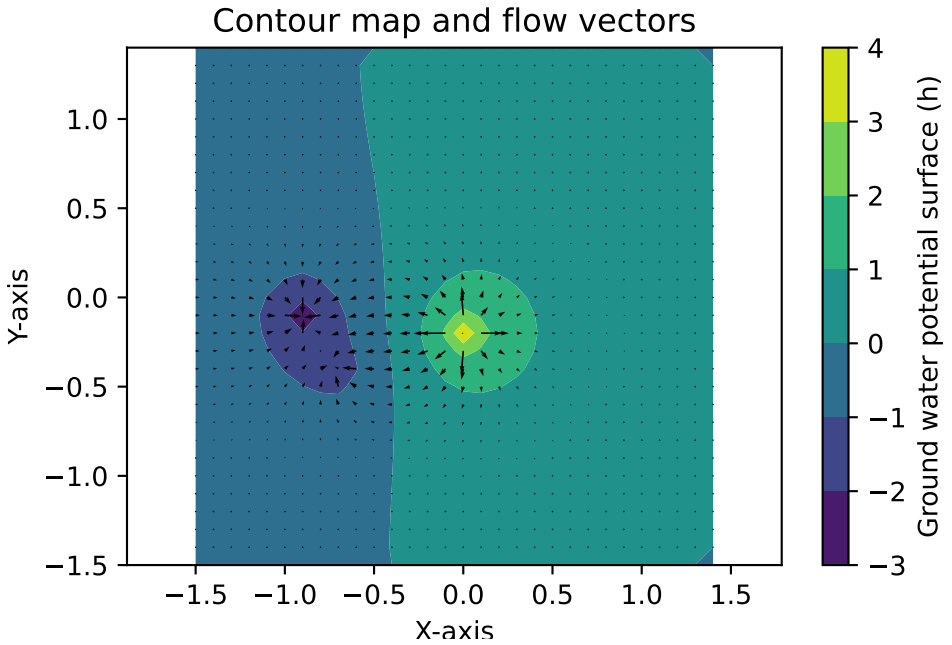

Figure 2: Plot of vector arrows using Python. The vectors indicate strength and direction of the negative gradient. The vectors are displayed over the contours of a tri-pole solution with a high value of 4 and lows of -2 and -3. The two low values were near each other compared to their respective distances to the high value.

Plot streamlines instead of arrows in Section

```
# Plot streamlines
plt.contourf(X,Y,h)
plt.streamplot(X, Y, -dhdx, -dhdy);
plt.axis('equal');
plt.show()
```

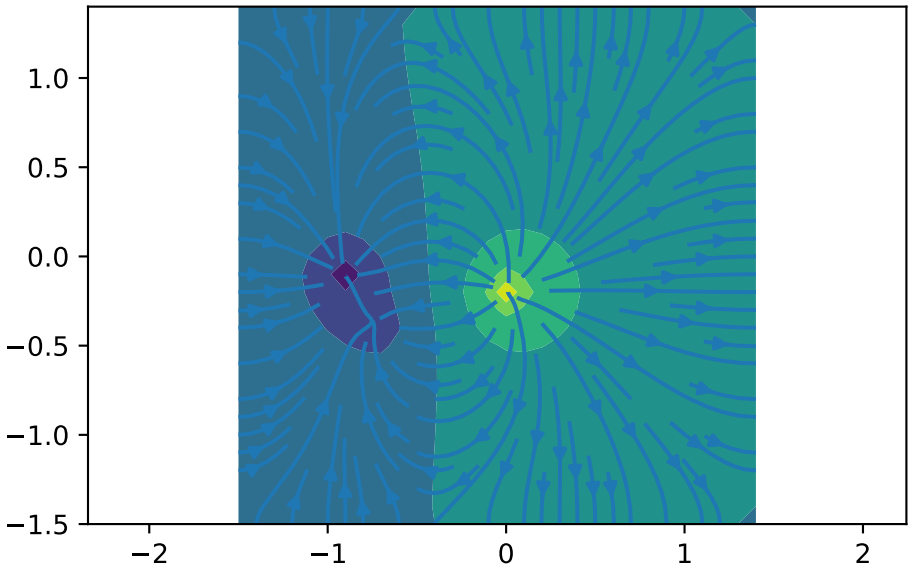

Figure 3: Plot of streamlines using Python. The streamlines are displayed over the contours of a tri-pole solution with a high value of 4 and lows of -2 and -3. The two low values were near each other compared to their respective distances to the high value.