Project Report

Minh Nguyen - 8/15/2024

In this project, my goal was to create a Python program for modeling corrosion. To do so, I primarily used several equations.

Electrochemical corrosion depends on reactions in the form

$$aA + mH + ne^{-} = bB + dH_2O.$$

Therefore, to model electrochemical corrosion, I used the Nernst equation

$$e = e^{o} - RT/nF * ln(\frac{(B)^{b}*(H_2O)^{d}}{(A)^{a}(H^{+})^{m}}),$$

where e is equal to the electrode potential, e^{o} is equal to the standard potential, R is equal to the ideal gas constant, T is equal to absolute temperature, n is equal to the ion charge, and F is equal to Faraday's constant. Additionally, I used the polarization equation

$$\eta = \beta * log(i/i_o) + 2.3RT/nF * log[1 - i/i_L],$$

where η is the amount of overpotential, β is the Tafel slope, i is the current density, i_o is the exchange current density, and i_L is the limiting current to generate the Tafel plots for each reaction. Using these Tafel plots, I was able to generate mixed potential graphs for a given set of corrosion reactions.

While modeling stress corrosion, I used a similar version of the polarization equations not accounting for limiting current,

$$i = i_o 10^{\eta/\beta}.$$

To incorporate elasto−plastic tension, I added the elastic deformation and plastic deformation equations to the anodic equilibrium potentials. The elastic deformation equation is

$$\Delta\varphi^{e} = -\frac{\Delta P V_m}{nF},$$

where $\Delta\varphi^{e}$ is shift in equilibrium potential due to elastic deformation, ΔP is excess pressure, and V_m is the molar volume of steel. The plastic deformation equation is

$$-\frac{TR}{nF} * ln(\frac{v\alpha}{N_o}\varepsilon_p + 1),$$

where ν is an orientation-dependent factor, α is a coefficient, N_o is initial density of dislocations prior to plastic deformation and ε_p is plastic strain. Varying ε_p to form a graph of the new equilibrium potentials for an iron corrosion reaction incorporates increasing strain. Finding the intersection of these plots yielded i_{corr} and E_{corr}, corresponding the equilibrium current density and potential, respectively.

While I did work on both mixed potential and stress corrosion calculations, as I did not know how to find the corresponding variables in stress corrosion for a mixed potential reaction, mixed potentials are currently not factored into my stress corrosion modeling.

In these simulations, I utilized common assumptions for some of my variables. For example, the Tafel constant was assumed to be 0.1 for all reactions. Additionally, I used $V_m = 9.99 * 10^{-6} m^3/mol$, ν = 0.45, α = $1.67 * 10^{15}$, and $N_0 = 10^{12}$.

Stress Corrosion

The code requires reaction equations ($aA + mH + ne^- = bB + dH_2O$) to be imported from a file called "reactions.csv", in which each line of the file contains information about the reaction's elements in the form

```
Unset
element, standard potential of reaction, concentration of reactant A,
coefficient of reactant A, concentration of reactant B, coefficient of reactant
B, concentration of hydrogen, ion charge, limiting cathodic current, limiting
anodic current.
```

For instance, I used these equations to test out my code:

```
Unset
H, 0.020, 0.0000002, 0, 0.0000002, 0, 2, 2, 0.0001, 0.2
Fe, 0.850, 0.00008, 1, 0.00008, 1, 0, 1, 0.0002, 0.006
```

Since these reactions involve the corrosion of iron, I set $\Delta P = \Delta P_m$ (the elastic deformation limit, equal to 1/3 of yield strength of iron) = 16666.6667 Pa. Limiting current wasn't considered in this simulation, but I included it for possible future use in more complicated simulations.

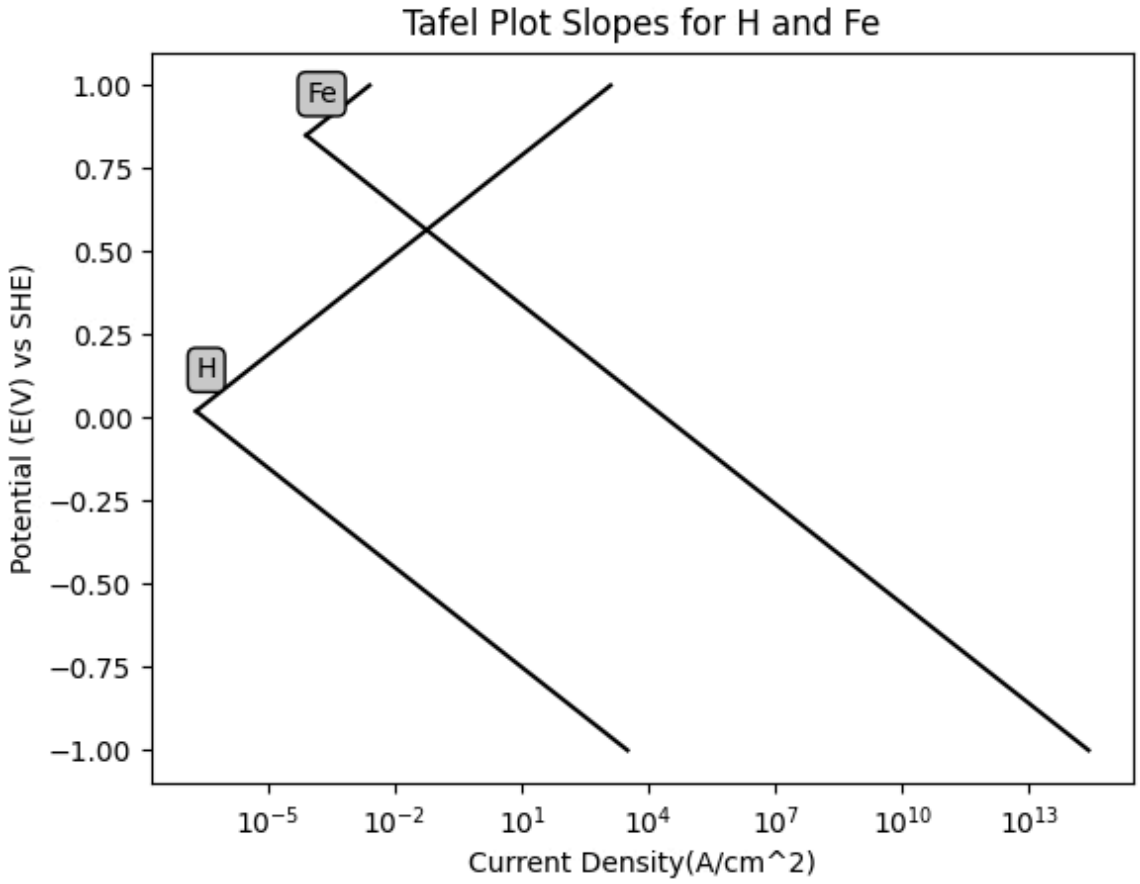
Python

```
import numpy
import matplotlib.pyplot as plt
import matplotlib.scale as mscale
from numpy import inf, log as ln, minimum
from numpy import log10 as log
from matplotlib.ticker import FixedLocator, NullFormatter
from scipy.optimize import fsolve
import matplotx

#constants for Fe
P = 16666.6667
metal = "Fe"

#given constants for simulation
Vm = 9.99E-6
v = 0.45
a = 1.67E15
N0 = 1E12

R = 8.3144598
F = 96485
TafelC = 0.1

T = 298.15

#setting up files and graph
f = open('reactions.csv')

fig, (ax1) = plt.subplots(1)
ax1.set_xscale("log")

plt.xlabel("Current Density")
plt.ylabel("Voltage")

reactants = {}

plt.xlabel("Current Density"+ "(A/cm^2)")
plt.ylabel("Potential (E(V) vs SHE)")
plt.title("Tafel Plot Slopes for H and " + metal)
```

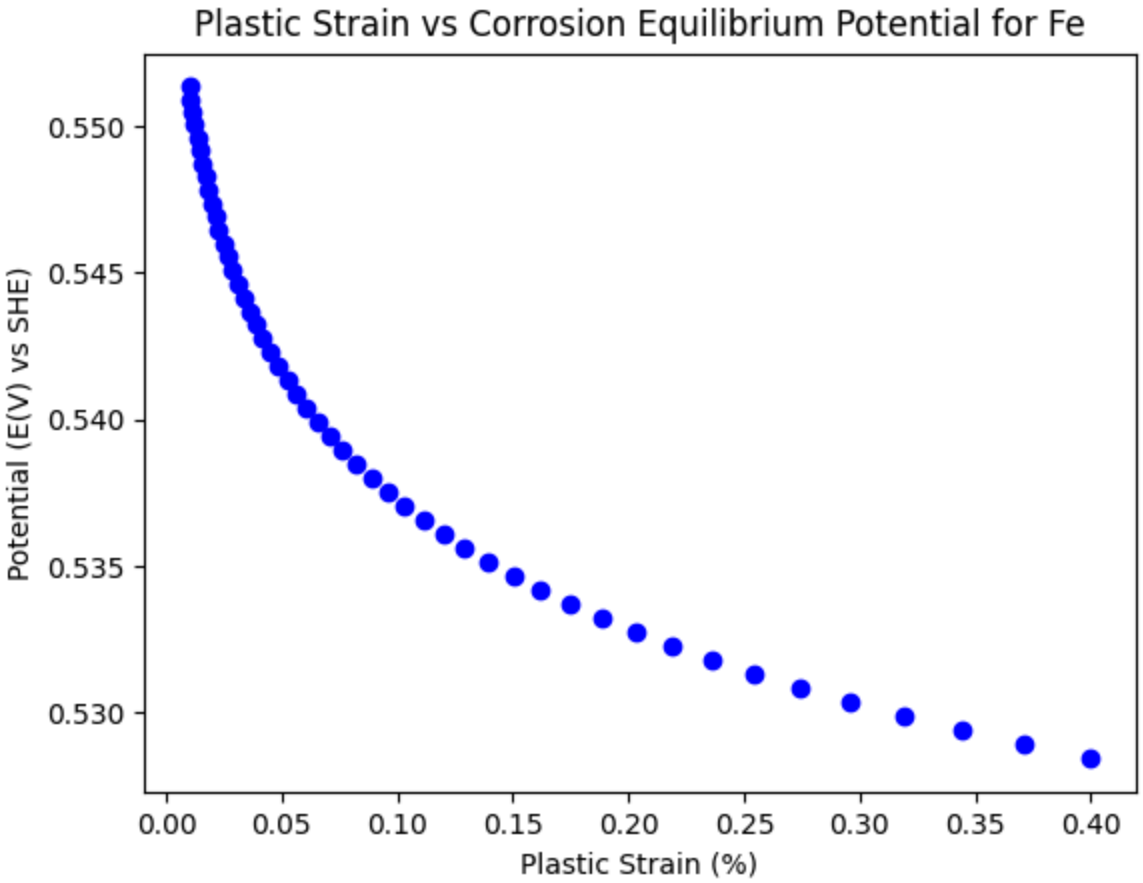
```
#iterates through each reaction- calculating reaction for electrochemical only
for row in f:
    l = row
    name, standardPotential, Aconc, Acoeff, Bconc, Bcoeff, m, n, iLc, iLa,
*_rest = l.split(',')
    standardPotential = float(standardPotential)
    Aconc = float(Aconc)
    Acoeff = float(Acoeff)
    Bconc = float(Bconc)
    Bcoeff = float(Bcoeff)
    m = int(m)
    n = int(n)
    iLc = float(iLc)
    iLa = float(iLa)

    #calculating Tafel slopes for electrochemical reaction
    e = standardPotential + 2.3 * R * T / n / F * ln(Aconc**Acoeff / Bconc **
Bcoeff)
    reactants[name] = [e, Aconc, Acoeff, Bconc, Bcoeff, m, n, iLc, iLa]

    Ec = numpy.linspace(e, 1, 10000)
    Ea = numpy.linspace(e, -1, 10000)

    def oxi(Ec):
        return Aconc*10**((Ec-e)/TafelC)
    def red(Ea):
        return Bconc*10**((e-Ea)/TafelC)

    plt.plot(oxi(Ec), Ec, color = "black")
    plt.plot(red(Ea), Ea, color = "black")
    plt.annotate(name, xy=(oxi(e), e+0.1), bbox = dict(boxstyle="round",
fc="0.8"))

plt.show()

#factoring in stress corrosion
plt.xlabel("Plastic Strain (%)")
plt.ylabel("Potential (E(V) vs SHE)")
plt.title("Plastic Strain vs Corrosion Equilibrium Potential for " + metal)

if (reactants["H"][0] > reactants[metal][0]):

    for strain in numpy.geomspace(1, 1000):
```

```
        Ec = numpy.linspace(reactants[metal][0], 1, 10000)
        Ea = numpy.linspace(reactants["H"][0], -1, 10000)

        #incorporating continous elasto-plastic tension
        def oxi(Ec):
            return reactants["H"][1]*10**((Ec-reactants[metal][0]-
P*Vm/(reactants[metal][6]*F)-T*R/(reactants[metal][6]*F)*ln(v*a/N0 * strain +
1))/TafelC)

        def red(Ea):
             return
reactants["H"][1]*10**((reactants["H"][0]-Ea)/TafelC)*10**(strain*Vm/(-6*F*Tafe
lC))

        def solve(E):
            return oxi(E)-red(E)

        plt.scatter(strain, fsolve(solve, 0.0001)[0])

else:

    for strain in numpy.geomspace(0.01, 0.4):
        Ec = numpy.linspace(reactants["H"][0], 1, 10000)
        Ea = numpy.linspace(reactants[metal][0], -1, 10000)

        #incorporating continous elasto-plastic tension
        def oxi(Ec):
            return reactants["H"][1]*10**((Ec-reactants["H"][0])/TafelC)

        def red(Ea):
             return reactants[metal][1]*10**((reactants[metal][0]-Ea-
P*Vm/(reactants["H"][6]*F)-T*R/(reactants["H"][6]*F)*ln(v*a/N0 * strain +
1))/TafelC)*10**(strain*Vm/(-6*F*TafelC))

        def solve(E):
            return oxi(E)-red(E)

        plt.scatter(strain, fsolve(solve, 0.0001)[0], color = "blue")
plt.show()
```

This resulted in two graphs. The first is of current density vs voltage for the hydrogen and iron reactions without considering strain; fsolve can be further used to find the intersection point between the two graphs to reveal i_{corr} and E_{corr}.



The second was of strain vs. voltage, where I varied the plastic strain component ε_p of the strain/stress corrosion equation while setting ΔP at 16666.6667 Pa. This resulted in the following graph:



Mixed Potential

To find the mixed potential of the reactions, I summed their current densities per potential to produce the following graph. To test out my code, I used the following reactions, written in a file with the same format as the reactions file in the stress corrosion simulation.

```
Unset
H, 0.020, 0.0000002, 0, 0.0000002, 0, 2, 2, 0.0001, 0.2
Cu, 0.350, 0.0001, 1, 0.0001, 1, 0, 2, 1000000000, 1000000000
Fe, 0.850, 0.00008, 1, 0.00008, 1, 0, 1, 0.0002, 0.006
I, 0.430, 0.001, 1, 0.001, 1, 0, 2, 0.02, 0.002
```

```
Python
import numpy
import matplotlib.pyplot as plt
```

```
import matplotlib.scale as mscale
from numpy import inf, log as ln, minimum, maximum
from numpy import log10 as log
from matplotlib.ticker import FixedLocator, NullFormatter
from numpy.core.function_base import geomspace
from scipy import interpolate
from scipy.optimize import fsolve

#given constants for simulation
R = 8.3144598
F = 96485
TafelC = 0.1

T = 298.15

#setting up files and graph
f = open('reactions.csv')

fig, (ax1) = plt.subplots(1)
ax1.set_xscale("log")

plt.xlabel("Current Density "+ "(A/cm^2)")
plt.ylabel("Potential (E(V) vs SHE)")
plt.title("Tafel Plot Slopes with Mixed Potentials")

mixedVoltageDataXOxi = []
mixedVoltageDataYOxi = []

mixedVoltageDataXRed = []
mixedVoltageDataYRed = []

#iterates through each reaction
for row in f:
    l = row
    name, standardPotential, Aconc, Acoeff, Bconc, Bcoeff, m, n, iLc, iLa,
*_rest = l.split(',')
    standardPotential = float(standardPotential)
    Aconc = float(Aconc)
    Acoeff = float(Acoeff)
    Bconc = float(Bconc)
    Bcoeff = float(Bcoeff)
    m = int(m)
    n = int(n)
    iLc = float(iLc)
```

```
    iLa = float(iLa)
    #calculates Tafel plots for each reaction
    e = standardPotential + 2.3 * R * T / n / F * ln(Aconc**Acoeff / Bconc **
Bcoeff)

    ic = numpy.geomspace(Aconc, iLc, 10000)
    ia = numpy.geomspace(Bconc, iLa, 10000)
    yc = []
    ya = []

    maxi = e
    for x in ic:
        val = e + TafelC * log(x/Bconc) - 2.3*R*T/n/F * log(1-x/iLc)
        if (val == inf):
            yc.append(maxi)
        else:
            yc.append(val)
            maxi = val

    mini = e
    for x in ia:
        val = e - TafelC * log(x/Aconc) + 2.3*R*T/n/F * log(1-x/iLa)
        if (val == inf):
            ya.append(mini)
        else:
            ya.append(val)
            mini = val

    #adds to mixed potential reaction for oxidizing reaction
    if (len(mixedVoltageDataYOxi) == 0):
        for i in range(0, len(ic)):
            mixedVoltageDataXOxi.append(ic[i])
            mixedVoltageDataYOxi.append(yc[i])
    else:
        oxiFunction = interpolate.interp1d(mixedVoltageDataYOxi,
mixedVoltageDataXOxi)
        newOFunction = interpolate.interp1d(yc, ic)

        rangeBelowDown = minimum(min(mixedVoltageDataYOxi), min(yc))
        rangeBelowUp = minimum(min(mixedVoltageDataYOxi), max(yc))
        rangeAboveDown = maximum(max(mixedVoltageDataYOxi), min(yc))
        rangeAboveUp = maximum(max(mixedVoltageDataYOxi), max(yc))
```

```
        #computes new oxidizing current densities and potentials for range of
existing mixed voltage function
        for i in range(0, len(mixedVoltageDataYOxi)):
            try:
                mixedVoltageDataXOxi[i] = newOFunction(mixedVoltageDataYOxi[i])
+ mixedVoltageDataXOxi[i]
            except ValueError:
                pass

        #computes new oxidizing current densities and potentials for values
below existing range of existing mixed voltage function
        try:
            for value in numpy.geomspace(rangeBelowUp, rangeBelowDown):
                mixedVoltageDataXOxi.insert(0, newOFunction(value))
                mixedVoltageDataYOxi.insert(0, value)
        except ValueError:
            pass

        #computes new oxidizing current densities and potentials for values
above existing range of existing mixed voltage function
        try:
            for value in numpy.geomspace(rangeAboveDown, rangeAboveUp):
                mixedVoltageDataXOxi.append( newOFunction(value) +
max(mixedVoltageDataXOxi))
                mixedVoltageDataYOxi.append(value)
        except ValueError:
            pass

    #adds to mixed potential reaction for reducing reaction
    if (len(mixedVoltageDataYRed) == 0):
        for i in range(0, len(ia)):
            mixedVoltageDataXRed.append(ia[i])
            mixedVoltageDataYRed.append(ya[i])
    else:
        redFunction = interpolate.interp1d(mixedVoltageDataYRed,
mixedVoltageDataXRed)
        newRFunction = interpolate.interp1d(ya, ia)

        rangeBelowDown = minimum(min(mixedVoltageDataYRed), min(ya))
        rangeBelowUp = minimum(min(mixedVoltageDataYRed), max(ya))
        rangeAboveDown = maximum(max(mixedVoltageDataYRed), min(ya))
        rangeAboveUp = maximum(max(mixedVoltageDataYRed), max(ya))
```

```
        #computes new reducing current densities and potentials for range of
existing mixed voltage function
        for i in range(0, len(mixedVoltageDataYRed)):
            try:
                mixedVoltageDataXRed[i] = newRFunction(mixedVoltageDataYRed[i])
+ mixedVoltageDataXRed[i]
            except ValueError:
                pass

        #computes new reducing current densities and potentials for values
above existing range of existing mixed voltage function
        try:
            for value in numpy.geomspace(rangeAboveDown, rangeAboveUp):
                mixedVoltageDataXRed.insert(0, newRFunction(value))
                mixedVoltageDataYRed.insert(0, value)
        except ValueError:
            pass

        #computes new reducing current densities and potentials for values
below existing range of existing mixed voltage function
        try:
            for value in numpy.geomspace(rangeBelowUp, rangeBelowDown):
                mixedVoltageDataXRed.append(newRFunction(value) +
max(mixedVoltageDataXRed))
                mixedVoltageDataYRed.append(value)
        except ValueError:
            pass

    #graphs Tafel plots for individual reactions
    plt.plot(ic, yc, color = "black")
    plt.plot(ia, ya, color = "black")
    #adds label to plot
    plt.annotate(name, xy=(ic[0], yc[0]), xytext=(ic[0], yc[0]+0.1), bbox =
dict(boxstyle="round", fc="0.8"), arrowprops=dict(facecolor='gray', color =
"gray", width = 0.01, headwidth = 0.01))

#graphs oxidizing and reduced reactions
plt.plot(mixedVoltageDataXOxi, mixedVoltageDataYOxi, color = "red", label =
"mixed oxidizing reaction")
plt.plot(mixedVoltageDataXRed, mixedVoltageDataYRed, color = "blue", label =
"mixed reduced reaction")
plt.legend()

plt.show()
```

This produced the following graph: