Optimizing Permeable Reactive Barriers for Aquifer Recharge of Secondary Wastewater

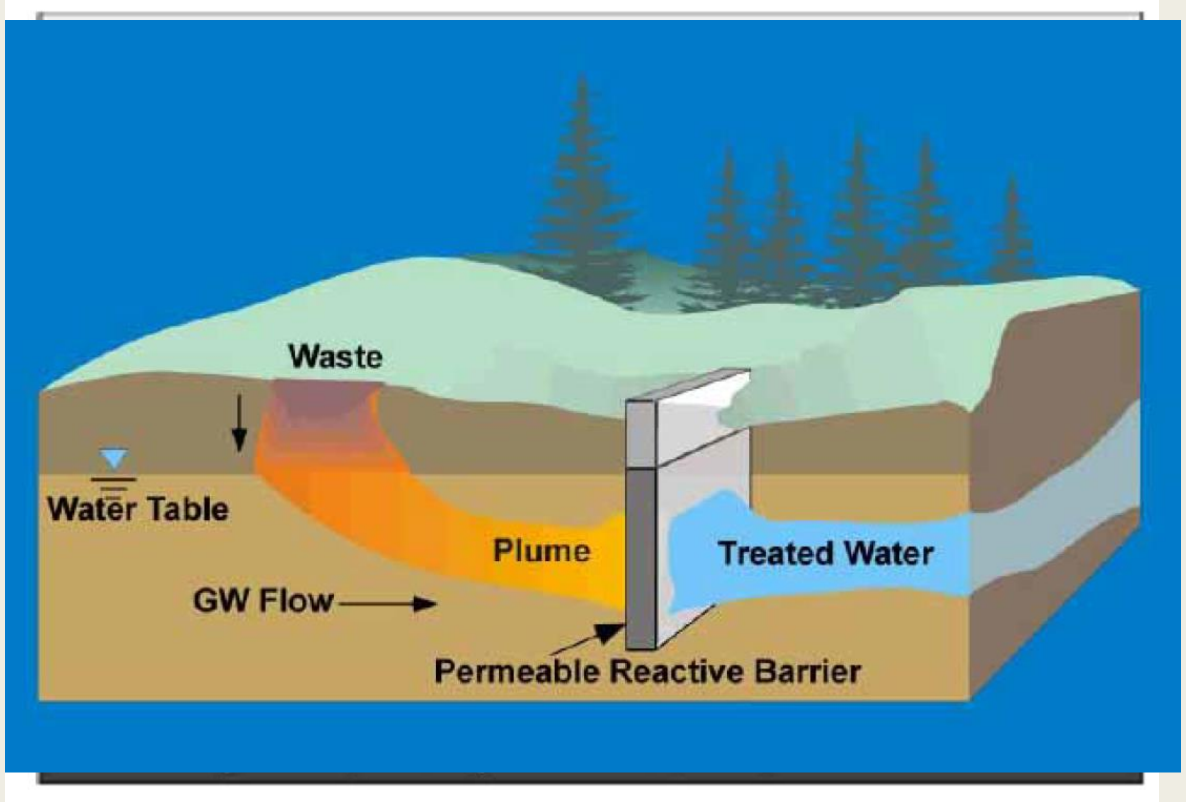
Sabrina Diemert
sdiemert at hawaii dot edu
CEE 696, UH Mānoa
May 7, 2018

Presentation outline

- Introduction (PRBs, iron-oxide coated sand)
- Implementation
- Results
- Conclusion

Introduction: Permeable Reactive Barriers (PRBs)

- “An emplacement of reactive materials in the subsurface designed to intercept a contaminant plume, provide a flow path through the reactive media, and transform the contaminant(s) into environmentally acceptable forms to attain remediation goals down-gradient of the barrier” (U.S. EPA, 1998)
- Can flow with passive gradient or be assisted by injection/extraction wells



Adapted from EPA 542-R-13-018 (2013)

Introduction: Iron-oxide coated sand

- Allow a surface to grow iron-reducing bacteria (IRB) biofilms
- Better removal of *E. coli*: consider first-order decay constants (Kim, 2015):
  - *Iron-coated sand: 0.85 – 1.85 day^{-1}*
  - *Regular sand: 0.036 day^{-1}*
- Adsorption also plays a role, but ignore this for now!

Motivation: Iron-coated sand PRB for secondary wastewater treatment on O’ahu

- May be able to use for more decentralized treatment of wastewater
  - *For example, Millilani + Wahiawa wastewater = 1.50 MG/day = 5.68 ML/day (AECOM, 2016)*
- Want to determine the smallest cost PRB (i.e. smallest length/width) which will adequately treat secondary wastewater
  - *Initial E. coli concentration = concentration after secondary treatment*
  - *=~ 300 CFU/mL (AECOM, 2016)*
  - *Assume 3 log reduction is adequate for disinfection of E. coli*
- Test a range of *E. coli* decay rates, in case sand doesn’t perform the same as in laboratory settings

Goal of model

- **The goal of the model is to reduce *E. coli* concentration by 3 log while minimizing the cost of the PRB.**
  - The cost of the PRB includes production of iron coated sand and excavation.
    - Ferric chloride ($FeCl_3$) costs 400-700 USD/ton; assume 700 USD/ton, or 0.77 USD/kg (estimated from alibaba.com)
    - Sand requires 141 kg $FeCl_3/m^3$ sand (S. Diemert lab experiments, 2018)
    - Rough estimate for excavation for local area code yielded 1.8 hours per cubic meter of material excavated, 200 USD/hour, and material/rental/staging costs of approximately 2100 USD (estimated from www.homewyse.com/services/cost_to_excavate_land.html)
- **Therefore, Cost[USD] = 3000 + 469*length_prb*width_prb*depth**
  - Width and length are the variables being manipulated for optimization

Implementation

```
def ecoli_model(prb_dims, *k_val):
    prb_real_width = prb_dims[0]
    prb_real_length = prb_dims[1]
```

- Start from example_reactive_barrier.py
  - *Use MT3DMS for E. coli particulate flow*
- Define function of ecoli_model, which includes flopy model running and returns a Boolean (discuss later)
- Change grid to fit study area around Millilani (2 x 2 km)
- Add pumping and extraction wells with 5.68 ML/d flow rates
- Input PRB length and width, convert into index and plottable values
- Define E. coli decay rate as "k_val" at PRB indices, 0 day^{-1} at other points
- Set boundary conditions:
  - *Constant head = 5.6 m at north boundary*
  - *Constant head = 5.2 m at south boundary*
  - *No flux at east/west (Oki, 2005 and initial simulation results)*

Implementation (cont.)

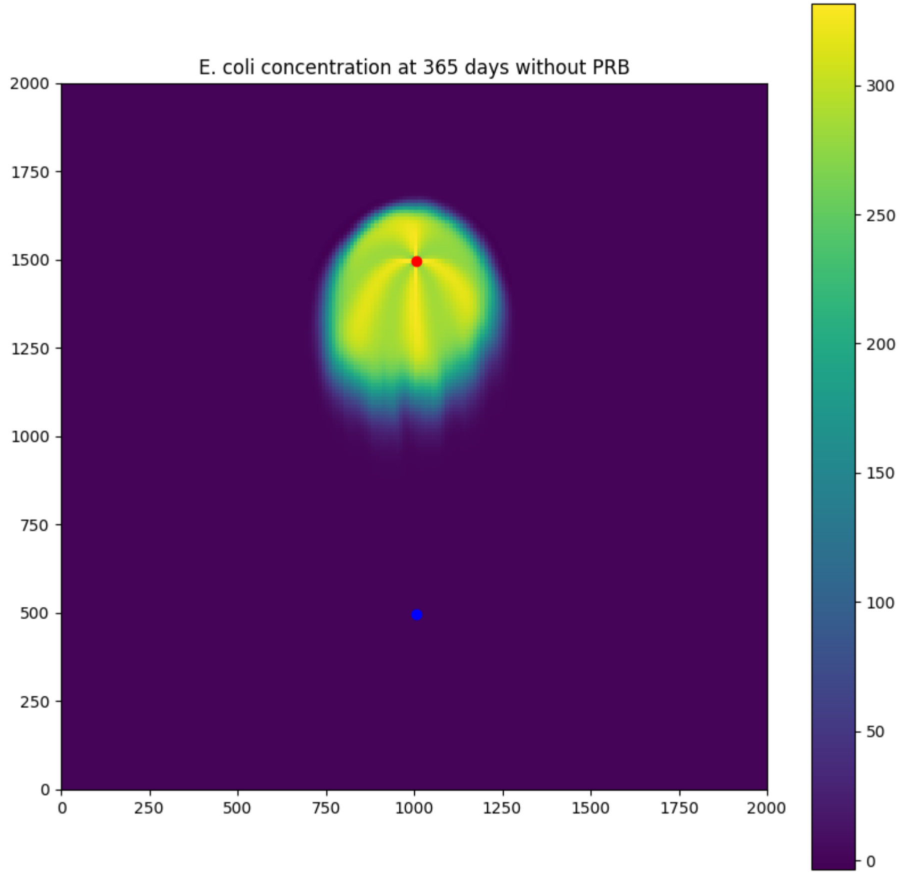
- Load heterogeneous horizontal hydraulic conductivity value text generated by Harry, normalize to appropriate value for Oahu aquifer (457.2 m/d, Oki, 2005)
- Set hydraulic conductivity for PRB (717.1 m/d, Elder, 2000)
- Use dispersivity of 10.0 m (not 76.2 m/d per Oki, 2005)
- Use porosity, vertical hydraulic conductivity, diffusivity per (Oki, 2005)
- Add initial E. coli concentration at Well #1 as 300 (CFU/mL) in source/sink package
- Run model (silent = True), collect head and concentration information
- Test if model meets removal criteria of 3 log reduction (300 -> 0.3)

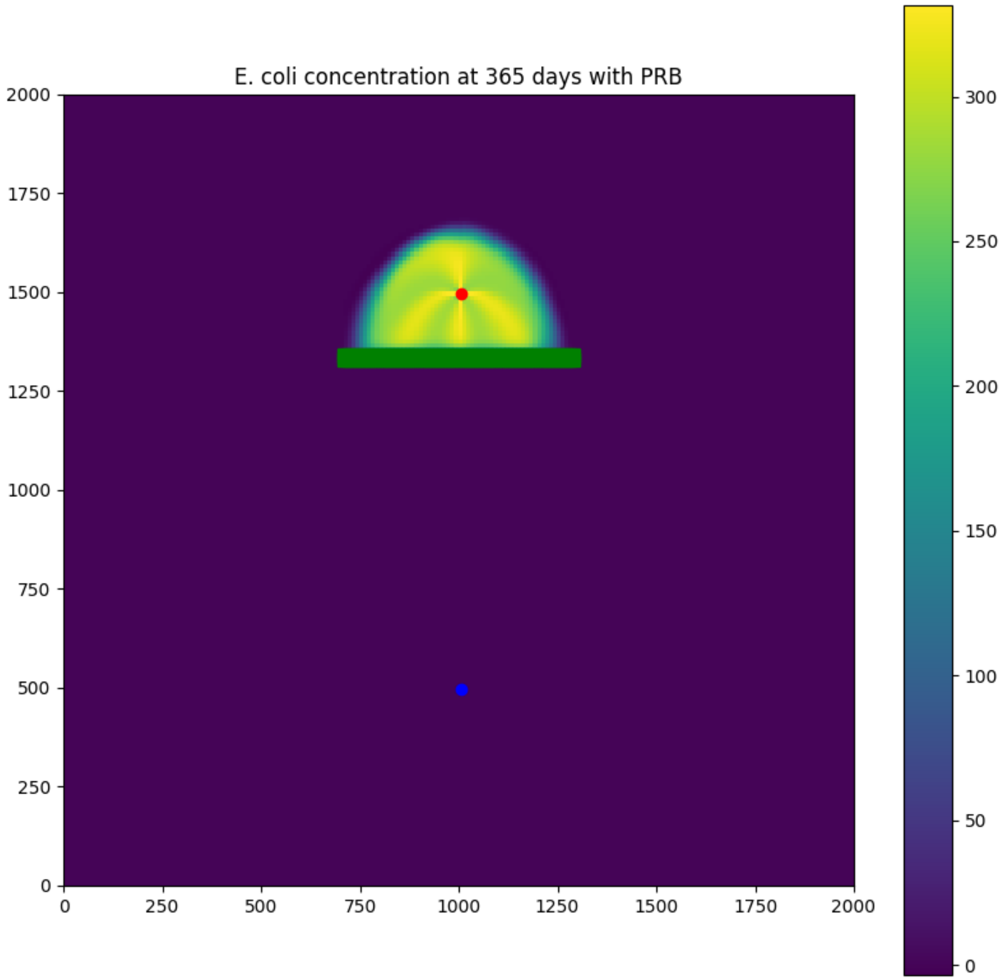
```
if np.any(conc[0, rrow_end:, :] > 0.3):
    print("Penalty applied")
    return False
else:
    print("No penalty")
    return True
```

Implementation (cont.)

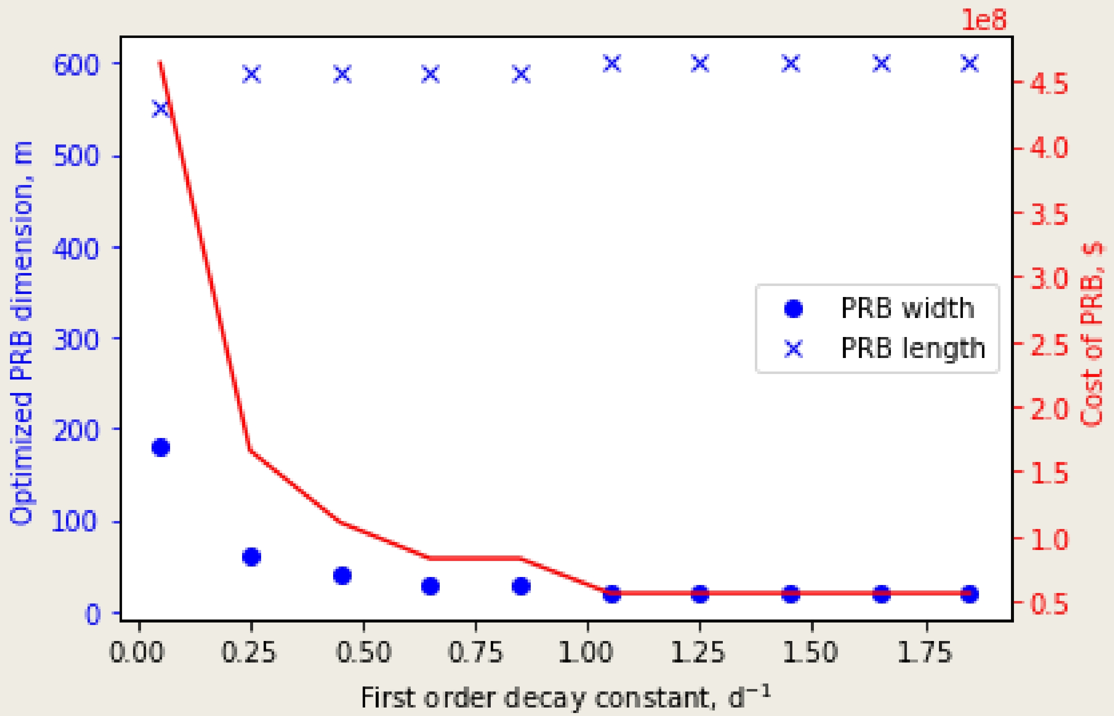
- Need integer programming/optimization
- Originally used opt.brute
- Then, wrote own loops to speed up the process
- Define “best area” = max width * max length
  - Loop over decay rate constants to test
    - Loop over width range to test
      - Loop over length range to test
        - Check if width*length < best_area
          - If ecoli_model(width, length, decay rate):
            - Save width, length
            - Update best_area

Results: heterogeneous hydraulic conductivity field





Results: heterogeneous hydraulic conductivity field



Conclusions

- Minimum PRB dimensions = 20 m x 600 m
- Minimum cost = $56M
  - *Feasible??*

References

- AECOM. (2016) Honouliuli Wastewater Treatment Plant Secondary Treatment and Facilities. Environmental Impact Statement.
- Elder (2000). Evaluation and Design of Permeable Reactive Barriers Amidst Heterogeneity. *PhD Thesis. University of Wisconsin-Madison, Madison, WI*.
- Kim, L. (2015). Low cost soil-based filtration for water reclamation. *PhD Thesis. University of Hawaii at Manoa, Honolulu, HI.*
- Oki, D.S. (2005). Numerical Simulation of the Effects of Low-Permeability Valley-Fill Barriers and the Redistribution of Ground-Water Withdrawals in the Pearl Harbor Area, Oahu, Hawaii. U.S.G.S. Scientific Investigations Report 2005-5253.
- Painter, B. (2005) Optimisation of PRB Systems for the Remediation of Contaminated Groundwater. *PhD Thesis, Lincoln University, Lincoln, NZ.*
- U.S. EPA (2013) Introduction to the In Situ Bioremediation of Groundwater, EPA 542-R-13-018

Implementation

```python
# brute optimization to restrict to integer values for PRB
# loop over different k-values
# changed brute to personalized optimizer for speed
# added heterogeneous HK field

import numpy as np
import flopy
import matplotlib.pyplot as plt
import flopy.utils.binaryfile as bf
#import scipy.optimize as opt


def ecoli_model(prb_dims, *k_val):
    prb_real_width = prb_dims[0]
    prb_real_length = prb_dims[1]

    # Assign name and create modflow model object
    modelname = 'mf-mt'
    mf = flopy.modflow.Modflow(modelname, exe_name='./mf2005')

    # Model domain and grid definition
    # zoom in on Milliani
    Lx = 2000.
    Ly = 2000.
    ztop = 0.
    zbot = -50.
    nlay = 1
    # keep individual grid sizes consistent with previous class egs.
    nrow = 200  # deleted zero
    ncol = 200  # deleted zero
    delr = Lx / ncol
    delc = Ly / nrow
    delv = (ztop - zbot) / nlay
    botm = np.linspace(ztop, zbot, nlay + 1)
```

```python
# define wells
# injection well
# first pump (injection)
pumping_rate1 = 5680.  # 5.68 ML/d injection flow
wcol1 = round(ncol / 2)  # test
wrow1 = round(nrow / 4)  # test
# second pump (extraction)
pumping_rate2 = -5680.  # 5.68 ML/d injection flow
wcol2 = round(ncol / 2)
wrow2 = round(nrow * 3 / 4)

# define decay rate
rc1 = np.full((nlay, nrow, ncol), 0.0)  # zero decay rate
# location of reactive barrier
# first, define dimensions and convert to grid units
prb_width = np.round(prb_real_width / delr).astype(int)
length_prb = np.round(prb_real_length / delc).astype(int)  # length in units of grid
# locate columns and rows
rcol1 = (round(Lx / 2 / delc) - round((length_prb - 1) / 2)).astype(int)
rcol2 = rcol1 + (length_prb - 1)  # run length of PRB
rrow = wrow1 + 15  # somewhat arbitrary start location (optimize later?)
rrow_end = rrow + prb_width  # define increment in optimization routine?

# define reactive PRB area
rc1[:, rrow:rrow_end + 1, rcol1:rcol2 + 1] = k_val

# define a function for plotting the PRB in the given location
# fill y and x arrays and combine into one
def fill_y(startcol, endcol, startrow, endrow):
    y_arr = np.empty([1, (endcol - startcol + 1) * (endrow - startrow + 1)], dtype=int)
    count = 0
    row = -1
    for i in range((endrow - startrow + 1)):
        row += 1
        for j in range(endcol - startcol + 1):
            y_arr[0, count] = startrow + row
            count += 1
    return y_arr
```

Implementation

```python
def fill_x(startcol, endcol, startrow, endrow):
    x_arr = np.empty([1, (endcol - startcol + 1) * (endrow - startrow + 1
                      dtype=int)
    count = 0
    row = 0
    for i in range((endrow - startrow + 1)):
        row += 1
        col = 0
        for j in range(endcol - startcol + 1):
            x_arr[0, count] = startcol + col
            col += 1
            count += 1
    return x_arr

def fill_xy(startcol, endrcol, starrtrow, endrowd):
    x_array = fill_x(startcol, endrcol, starrtrow, endrowd)
    y_array = fill_y(startcol, endrcol, starrtrow, endrowd)
    xy_array = np.append(x_array, y_array, axis=0)
    return xy_array

# call function
prb_xy = fill_xy(rcol1, rcol2, rrow, rrow_end)

# convert to correct origin for plotting
prb_xy_adj = np.zeros(prb_xy.shape)
prb_xy_adj[0, :] = (prb_xy[0, :] + 0.5) * delr
prb_xy_adj[1, :] = Ly - (prb_xy[1, :] + 0.5) * delc

# Create the discretization object
dis = flopy.modflow.ModflowDis(mf, nlay, nrow, ncol, delr=delr, delc=delc
                               top=ztop, botm=botm[1:])

# Variables for the BAS package
ibound = np.ones((nlay, nrow, ncol), dtype=np.int32)
ibound[:, 0, :] = -1  # constant head along north boundary
ibound[:, -1, :] = -1  # constant head along south boundary
```

```python
# simplify initial head levels
strt = np.ones((nlay, nrow, ncol), dtype=np.float32)
strt[:, 0, :] = 5.6  # 19.7 ft asl along north boundary (Oki et al. 2005), adjusted for zoom
# strt[:, :, -1] = 0.
strt[:, -1, :] = 5.2  # adjusted for zoom from previous simulations
bas = flopy.modflow.ModflowBas(mf, ibound=ibound, strt=strt)

# Add LPF package to the MODFLOW model
# define hk array (leave vka for now)
HK3 = np.loadtxt("HK3.txt")
hka = np.zeros((nlay, nrow, ncol), dtype=np.int32)
hka[0,:,:] = HK3*457.2
  # 457.2 m/d = 1500 ft/d (Oki, 2005)
hka[:, rrow, rcol1:rcol2+1] = 717.1
#  fix hk for PRB to = 717.1 m/d = 8.3 x 10 -3 m/s
# (Max range for Peerless iron PRB material, Elder PhD thesis, 2000)

lpf = flopy.modflow.ModflowLpf(mf, hk=hka, vka=2.3, ipakcb=53)
# parameters from Oki et al (2005)

# Add OC package to the MODFLOW model
spd = {(0, 0): ['print head', 'print budget', 'save head', 'save budget']}
oc = flopy.modflow.ModflowOc(mf, stress_period_data=spd, compact=True)

# well package

# fix well_sp so that you have two wells
wel_sp = [[0, wrow1, wcol1, pumping_rate1], [0, wrow2, wcol2, pumping_rate2]]
# lay, row, col index, pumping rate
stress_period_data = {0: wel_sp}  # define well stress period {period, well info dictionary}
wel = flopy.modflow.ModflowWel(mf, stress_period_data=stress_period_data)

# PCG package for matrix computation
pcg = flopy.modflow.ModflowPcg(mf)

# linkage to mt3dms LMT package
lmt = flopy.modflow.ModflowLmt(mf, output_file_name='mt3d_link.ftl')

# Write the MODFLOW model input files
mf.write_input()
```

Implementation

```
# Run the MODFLOW model
# add silent coding
success, buff = mf.run_model(silent=True, pause=False, report=True)
if not success:
    raise Exception('MODFLOW did not terminate normally.')

# create mt3dms model object
mt = flopy.mt3d.Mt3dms(modflowmodel=mf, modelname=modelname,
                       exe_name='./mt3dms', ftlfilename='mt3d_link.ftl')

# basic transport package
btn = flopy.mt3d.Mt3dBtn(mt, prsity=0.2, icbund=1, sconc=0.0,
                         ncomp=1, perlen=365, nper=1, nstp=50,
                         tsmult=1.0, nprs=-1, nprobs=10, cinact=-1, chkmas=True)

# changed perlen = 365

# advection package
adv = flopy.mt3d.Mt3dAdv(mt, mixelm=-1, percel=0.75)
# dispersion package
dsp = flopy.mt3d.Mt3dDsp(mt, al=10.0, trpt=0.01, trpv=0.01, dmcoef=1.e-9)
# changed dispersivity from 76.2 to 10

# source/sink package
ssm_data = {}
itype = flopy.mt3d.Mt3dSsm.itype_dict()
ssm_data[0] = [(0, wrow1, wcol1, 300., itype['WEL'])]
# initial E. coli concentration = RW level = 3 x 10^7 CFU/100mL
# for secondary effluent, 300 CFU/100mL

ssm = flopy.mt3d.Mt3dSsm(mt, stress_period_data=ssm_data)

rct = flopy.mt3d.Mt3dRct(mt, ireact=1, rc1=rc1, igetsc=0)
# igetsc = 0 : no sorption

# matrix solver package
gcg = flopy.mt3d.Mt3dGcg(mt, cclose=1e-6)

# write mt3dms input
mt.write_input()
# run mt3dms
mt.run_model(silent=True)
```

```
'''
fig = plt.figure(figsize=(10, 10))
ax = fig.add_subplot(1, 1, 1, aspect='equal')
'''
# define location of wells
wpt = ((wcol1 + 0.5) * delr, Ly - ((wrow1 + 0.5) * delc))  # origin at low upper..
wpt2 = ((wcol2 + 0.5) * delr, Ly - ((wrow2 + 0.5) * delc))

hds = bf.HeadFile(modelname + '.hds')
times = hds.get_times()  # simulation time, steady state
head = hds.get_data(totim=times[-1])

cbb = bf.CellBudgetFile(modelname + '.cbc')  # read budget file
frf = cbb.get_data(text='FLOW RIGHT FACE', totim=times[-1])[0]
fff = cbb.get_data(text='FLOW FRONT FACE', totim=times[-1])[0]
'''
# create flopy plot object, plot grid and contour
modelmap = flopy.plot.ModelMap(model=mf, layer=0)
lc = modelmap.plot_grid() # grid
cs = modelmap.contour_array(head, levels=np.linspace(head.min(), head.max(), 21)) # head contour
plt.clabel(cs, fontsize=20, fmt='%1.1f', zorder=1) # contour label
quiver = modelmap.plot_discharge(frf, fff, head=head) # quiver
plt.plot(wpt[0],wpt[1],'ro') # well location
plt.plot(prb_xy_adj[0], prb_xy_adj[1],'gs') #PRB location
plt.show()
'''
# plot conc
ucnobj = bf.UcnFile('MT3D001.UCN')
# print(ucnobj.list_records()) # get values
times = ucnobj.get_times()  # simulation time
time3 = times[-1]  # the last simulation time

conc = ucnobj.get_data(totim=time3)
'''
# conc at time3
fig = plt.figure(figsize=(10,10))
ax = fig.add_subplot(1, 1, 1, aspect='equal')
modelmap = flopy.plot.ModelMap(model=mf, layer=0)
#lc = modelmap.plot_grid() # grid
cs = modelmap.plot_array(conc) # head contour
plt.colorbar(cs) # colorbar
plt.plot(wpt[0],wpt[1],'ro')  # well location
plt.plot(wpt2[0],wpt2[1],'bo')  #extraction well location
plt.plot(prb_xy_adj[0], prb_xy_adj[1],'gs') # well location #PRB location
plt.title('C  %g day' % time3)
plt.show()
'''
```

Implementation

```
    # demand at least log 3 removal (300 -> 0.3)
    # test returns whether PRB successfully meets removal criteria

    if np.any(conc[0, rrow_end:, :] > 0.3):
#           penalty = 10 ** 15
        print("Penalty applied")
        return False
    else:
        print("No penalty")
        return True


# optimize using brute method, constrained to multiples of 10 (grid si
# Go up to the original k value intended (1.85 d^-1)
# start by setting up output

brute_output = open("brute_output_hk3.txt", "w")
brute_output.write("k_val prb_opt_width prb_opt_length prb_opt_cost \n'
brute_output.close()

# ranges to check: 10 - 200 for width, 450 - 650
kval_test = np.linspace(0.05,1.85,10)
best_area = 200*650 + 100      # size of entire study grid
prb_dim_temp = [0,0]
prb_best_dim = [0,0]
```

```
for kval in reversed (kval_test):
    kvals = kval.astype(float)
    print("k_val = " + str(kvals))
    best_area = 200*650 + 100     # size of entire study grid
    prb_dim_temp = [0,0]
    prb_best_dim = [0,0]
    for w in range(20,201,10):
        print("PRB width = " + str(w))
        for l in range(450,651,10):
            if w*l < best_area:
                print("PRB length = " + str(l))
                prb_dim_temp = [w,l]
                if ecoli_model(prb_dim_temp,kvals):
                    prb_best_dim = prb_dim_temp
                    best_area = w*l
                    print("best area updated, w, l, = " + str(best_area) +
                          " " + str(w) + " " + str(l))
    best_cost = 3000 + 469 * prb_best_dim[0] * prb_best_dim[1] * 10
    brute_output = open("brute_output_hk3.txt", "a")
    brute_output.write(str(kvals) + " " + str(prb_best_dim[0]) + " " +
                       str(prb_best_dim[1])  + " " + str(best_cost) + " \n")
    print("writing to output " + str(kvals) + " " + str(prb_best_dim[0]) + " " +
                       str(prb_best_dim[1])  + " " + str(best_cost))
    brute_output.close()
```