

```
In [1]: 1 import scipy.optimize
        2 from functools import partial
```

```
In [2]: 1 ###Data Set###
        2 ###Region 1###
        3 #data are arranged [alfalfa, vine, corn]
        4 #price is $/ton
        5 crop_prices = [132, 700, 250]
        6 #yield in ton/acre
        7 crop_yields = [7, 6.5, 6]
        8 #costs are $/acre
        9 crop_costs = [681, 3478, 1000]
       10
```

```
In [3]: 1 ###Observed Data###
        2 #data are arranged [alfalfa, vine, corn]
        3 #acres of crops watered
        4 irrigated_crop_acres = [100, 30, 200]
        5 total_irrigated = sum(irrigated_crop_acres)
        6 total_available_land = total_irrigated
        7 #water application rates (ft per unit area)
        8 applied_water_rates = [4, 1.5, 2.5]
        9
```

```
In [4]: 1 #Loops to populate additional information
        2
        3 #revenues from crops, $/acre
        4 crop_revenues = []
        5 #total applied water, acre-ft
        6 applied_water = []
        7 #observed net returns for a given crop ($), acres * (revenue - cost)
        8 obs_net_returns = []
        9 net_return_per_acre = []
       10 for i in range(len(crop_prices)):
       11     crop_revenues.append(crop_prices[i] * crop_yields[i])
       12     applied_water.append(applied_water_rates[i] * irrigated_crop_acres[i])
       13     obs_net_returns.append(irrigated_crop_acres[i] * (crop_revenues[i] - crop
       14     net_return_per_acre.append(crop_revenues[i] - crop_costs[i])
       15
       16 total_applied_water = sum(applied_water)
```

```
In [5]: 1 #create calibration constraints for solver
        2 #simply multiplying each entry of acres by given constant
        3 cal_mult = 1.001
        4 cal_acre_constraints = []
        5 for i in range(len(irrigated_crop_acres)):
        6     cal_acre_constraints.append(irrigated_crop_acres[i] * cal_mult)
```

```

In [6]: 1 #define constraints for model
2 #note that for inequalities, the sum is to be calibrated to greater than or e
3 #So we need to make sure that extra water or land remains, so constraint minus
4 def irr_acres_constr(cal_irr_acres, total_irrigated_water=total_irrigated):
5     #limits the total irrigated acres in model to calibrated total
6     return total_irrigated_water - sum(cal_irr_acres)
7 def applied_water_constr(cal_irr_acres, applied_water_rates=applied_water_rat
8     #limits water applied in calibration to total applied in data
9     total_water = 0
10    for i in range(len(applied_water_rates)):
11        total_water += applied_water_rates[i] * cal_irr_acres[i]
12    return total_applied_water - total_water
13
14 #need to create number of constraints based on number of crops
15 def crop0(cal_irr_acres, cal_acre_constraints=cal_acre_constraints):
16     return cal_acre_constraints[0] - cal_irr_acres[0]
17
18 def crop1(cal_irr_acres, cal_acre_constraints=cal_acre_constraints):
19     return cal_acre_constraints[1] - cal_irr_acres[1]
20
21 def crop2(cal_irr_acres, cal_acre_constraints=cal_acre_constraints):
22     return cal_acre_constraints[2] - cal_irr_acres[2]
23
24 #create dictionary of constraints
25 cons = [{'type': 'ineq', 'fun': irr_acres_constr},
26         {'type': 'ineq', 'fun': applied_water_constr},
27         {'type': 'ineq', 'fun': crop0},
28         {'type': 'ineq', 'fun': crop1},
29         {'type': 'ineq', 'fun': crop2},
30         ]
31
32 #establish parameters for model
33 guess_irr_acres = [50, 50, 50]
34
35 #bounds for model
36 bnds = ((0, total_available_land), (0, total_available_land), (0, total_avail

```

```

In [7]: 1 def calc_observed_net_revenue(guess_irr_acres, applied_water_rates=applied_wa
2     """
3     Calculates the total net revenues for all crops, negative since minimizin
4     Inputs need to be lists of equal lengths with respective crops lined up
5     """
6     total_net_revenue = 0
7     for i in range(len(applied_water_rates)):
8         total_net_revenue += net_return_per_acre[i] * guess_irr_acres[i]
9
10    return -1 * total_net_revenue

```

```

In [8]: 1 #run the model to fit parameters
2 results = scipy.optimize.minimize(calc_observed_net_revenue, #function to min
3     x0=guess_irr_acres,
4     method='SLSQP',
5     bounds=bnds,
6     constraints=cons)
7

```

In [9]: 1 results

Out[9]: fun: -156536.27023315313
jac: array([-243., -1072., -500.])
message: 'Optimization terminated successfully.'
nfev: 13
nit: 3
njev: 2
status: 0
success: True
x: array([99.77000004, 30.03000017, 200.20000008])

In [10]:

```

1  ###shadow price crop 0###
2
3  def crop0(cal_irr_acres, cal_acre_constraints=cal_acre_constraints):
4      return cal_acre_constraints[0] - cal_irr_acres[0] + 1
5
6  def crop1(cal_irr_acres, cal_acre_constraints=cal_acre_constraints):
7      return cal_acre_constraints[1] - cal_irr_acres[1]
8
9  def crop2(cal_irr_acres, cal_acre_constraints=cal_acre_constraints):
10     return cal_acre_constraints[2] - cal_irr_acres[2]
11
12 #create dictionary of constraints
13 cons = [{'type': 'ineq', 'fun': irr_acres_constr},
14         {'type': 'ineq', 'fun': applied_water_constr},
15         {'type': 'ineq', 'fun': crop0},
16         {'type': 'ineq', 'fun': crop1},
17         {'type': 'ineq', 'fun': crop2},
18         ]
19
20 #establish parameters for model
21 guess_irr_acres = [50, 50, 50]
22
23 #bounds for model
24 bnds = ((0, total_available_land), (0, total_available_land), (0, total_avail
25
26 results0 = scipy.optimize.minimize(calc_observed_net_revenue, #function to mi
27                                   x0=guess_irr_acres,
28                                   method='SLSQP',
29                                   bounds=bnds,
30                                   constraints=cons)
31 shadow_0 = results.fun - results0.fun
32
33 ###shadow price crop 1###
34
35 def crop0(cal_irr_acres, cal_acre_constraints=cal_acre_constraints):
36     return cal_acre_constraints[0] - cal_irr_acres[0]
37
38 def crop1(cal_irr_acres, cal_acre_constraints=cal_acre_constraints):
39     return cal_acre_constraints[1] - cal_irr_acres[1] + 1
40
41 def crop2(cal_irr_acres, cal_acre_constraints=cal_acre_constraints):
42     return cal_acre_constraints[2] - cal_irr_acres[2]
43
44 #create dictionary of constraints
45 cons = [{'type': 'ineq', 'fun': irr_acres_constr},
46         {'type': 'ineq', 'fun': applied_water_constr},
47         {'type': 'ineq', 'fun': crop0},
48         {'type': 'ineq', 'fun': crop1},
49         {'type': 'ineq', 'fun': crop2},
50         ]
51
52 #establish parameters for model
53 guess_irr_acres = [50, 50, 50]
54
55 #bounds for model
56 bnds = ((0, total_available_land), (0, total_available_land), (0, total_avail

```

```

57
58 results1 = scipy.optimize.minimize(calc_observed_net_revenue, #function to mi
59                                     x0=guess_irr_acres,
60                                     method='SLSQP',
61                                     bounds=bnds,
62                                     constraints=cons)
63 shadow_1 = results.fun - results1.fun
64
65 ###shadow price crop 2###
66
67 def crop0(cal_irr_acres, cal_acre_constraints=cal_acre_constraints):
68     return cal_acre_constraints[0] - cal_irr_acres[0]
69
70 def crop1(cal_irr_acres, cal_acre_constraints=cal_acre_constraints):
71     return cal_acre_constraints[1] - cal_irr_acres[1]
72
73 def crop2(cal_irr_acres, cal_acre_constraints=cal_acre_constraints):
74     return cal_acre_constraints[2] - cal_irr_acres[2] + 1
75
76 #create dictionary of constraints
77 cons = [{'type':'ineq', 'fun': irr_acres_constr},
78         {'type':'ineq', 'fun': applied_water_constr},
79         {'type':'ineq', 'fun': crop0},
80         {'type':'ineq', 'fun': crop1},
81         {'type':'ineq', 'fun': crop2},
82         ]
83
84 #establish parameters for model
85 guess_irr_acres = [50, 50, 50]
86
87 #bounds for model
88 bnds = ((0, total_available_land), (0, total_available_land), (0, total_avail
89
90 results2 = scipy.optimize.minimize(calc_observed_net_revenue, #function to mi
91                                     x0=guess_irr_acres,
92                                     method='SLSQP',
93                                     bounds=bnds,
94                                     constraints=cons)
95 shadow_2 = results.fun - results2.fun
96
97 lagrange_mults = [shadow_0, shadow_1, shadow_2]

```

In [11]:

```

1 #calculate PMP parameters
2 alpha = []
3 gamma = []
4
5 for i in range(len(lagrange_mults)):
6     alpha.append(crop_costs[i] - lagrange_mults[i])
7     gamma.append(2 * lagrange_mults[i] / irrigated_crop_acres[i])

```

In [18]:

```

1  ##Solve calibrated model##
2
3  def pmp_net_revenue(guess_irr_acres, alpha=alpha, gamma=gamma, applied_water_
4      """
5      Calculates the total net revenues for all crops using PMP, negative since
6      Inputs need to be lists of equal lengths with respective crops lined up
7      """
8      revenue = 0
9      pmp_cost = 0
10     for i in range(len(guess_irr_acres)):
11         revenue += net_return_per_acre[i] * guess_irr_acres[i]
12         pmp_cost += alpha[i] * guess_irr_acres[i] + 0.5 * gamma[i] * guess_ir
13
14     return -1 * (pmp_cost - revenue)
15
16 #setting constraints
17 water_available = 756 #in acre-ft
18 land_available = 330 #in acres
19 def irr_acres_constr(cal_irr_acres, land_available=land_available):
20     #limits the total irrigated acres to land available
21     return land_available - sum(cal_irr_acres)
22 def applied_water_constr(cal_irr_acres, applied_water_rates=applied_water_rat
23     #limits water applied in calibration to total applied in data
24     total_water = 0
25     for i in range(len(applied_water_rates)):
26         total_water += applied_water_rates[i] * cal_irr_acres[i]
27     return water_available - total_water
28
29 #create dictionary of constraints
30 cons = [{'type':'ineq', 'fun': irr_acres_constr},
31         {'type':'ineq', 'fun': applied_water_constr}]
32
33 #establish parameters for model
34 guess_irr_acres = [50, 50, 50]
35
36 #bounds for model
37 bnds = ((0, land_available), (0, land_available), (0, land_available))
38
39 #run calibrated model
40 pmp_results = scipy.optimize.minimize(pmp_net_revenue, #function to minimize
41                                     x0=guess_irr_acres,
42                                     method='SLSQP',
43                                     bounds=bnds,
44                                     constraints=cons)
45 print(pmp_results)

```

```

fun: -3529679.098369063
jac: array([ -438., -19815., -243.])
message: 'Positive directional derivative for linesearch'
nfev: 15
nit: 7
njev: 3
status: 8
success: False
x: array([2.47383842e-05, 3.30000000e+02, 1.37724660e-05])

```

In []:

1

In []:

1

In []:

1