In [1]:

```python
import scipy.optimize
from functools import partial
```

In [2]:

```python
###Data Set###
###Region 1###
#data are arranged [alfalfa, vine, corn]
#price is $/ton
crop_prices = [132, 700, 250]
#yield in ton/acre
crop_yields = [7, 6.5, 6]
#costs are $/acre
crop_costs = [681, 3478, 1000]

```

In [3]:

```python
###Observed Data###
#data are arranged [alfalfa, vine, corn]
#acres of crops watered
irrigated_crop_acres = [100, 30, 200]
total_irrigated = sum(irrigated_crop_acres)
total_available_land = total_irrigated
#water application rates (ft per unit area)
applied_water_rates = [4, 1.5, 2.5]

```

In [4]:

```python
#loops to populate additional information

#revenues from crops, $/acre
crop_revenues = []
#total applied water, acre-ft
applied_water = []
#observed net returns for a given crop ($), acres * (revenue - cost)
obs_net_returns = []
net_return_per_acre = []
for i in range(len(crop_prices)):
    crop_revenues.append(crop_prices[i] * crop_yields[i])
    applied_water.append(applied_water_rates[i] * irrigated_crop_acres[i])
    obs_net_returns.append(irrigated_crop_acres[i] * (crop_revenues[i] - crop
    net_return_per_acre.append(crop_revenues[i] - crop_costs[i])

total_applied_water = sum(applied_water)
```

In [5]:

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

In [6]:
```
#define constraints for model
#note that for inequalities, the sum is to be calibrated to greater than or e
#So we need to make sure that extra water or land remains, so contraint minus
def irr_acres_constr(cal_irr_acres, total_irrigated_water=total_irrigated):
    #limits the total irrigated acres in model to calibrated total
    return total_irrigated_water - sum(cal_irr_acres)
def applied_water_constr(cal_irr_acres, applied_water_rates=applied_water_rat
    #limits water applied in calibration to total applied in data
    total_water = 0
    for i in range(len(applied_water_rates)):
        total_water += applied_water_rates[i] * cal_irr_acres[i]
    return total_applied_water - total_water

#need to create number of constraints based on number of crops
def crop0(cal_irr_acres, cal_acre_constraints=cal_acre_constraints):
    return cal_acre_constraints[0] - cal_irr_acres[0]

def crop1(cal_irr_acres, cal_acre_constraints=cal_acre_constraints):
    return cal_acre_constraints[1] - cal_irr_acres[1]

def crop2(cal_irr_acres, cal_acre_constraints=cal_acre_constraints):
    return cal_acre_constraints[2] - cal_irr_acres[2]

#create dictionary of constraints
cons = [{'type':'ineq', 'fun': irr_acres_constr},
        {'type':'ineq', 'fun': applied_water_constr},
        {'type':'ineq', 'fun': crop0},
        {'type':'ineq', 'fun': crop1},
        {'type':'ineq', 'fun': crop2},
       ]

#establish parameters for model
guess_irr_acres = [50, 50, 50]

#bounds for model
bnds = ((0, total_available_land), (0, total_available_land), (0, total_avail
```

In [7]:
```
def calc_observed_net_revenue(guess_irr_acres, applied_water_rates=applied_wa
    """
    Calculates the total net revenues for all crops, negative since minimizin
    Inputs need to be lists of equal lengths with respective crops lined up
    """
    total_net_revenue = 0
    for i in range(len(applied_water_rates)):
        total_net_revenue += net_return_per_acre[i] * guess_irr_acres[i]

    return -1 * total_net_revenue
```

In [8]:
```
#run the model to fit parameters
results = scipy.optimize.minimize(calc_observed_net_revenue, #function to mir
                                  x0=guess_irr_acres,
                                  method='SLSQP',
                                  bounds=bnds,
                                  constraints=cons)

```

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]:

```
###shadow price crop 0###

def crop0(cal_irr_acres, cal_acre_constraints=cal_acre_constraints):
    return cal_acre_constraints[0] - cal_irr_acres[0] + 1

def crop1(cal_irr_acres, cal_acre_constraints=cal_acre_constraints):
    return cal_acre_constraints[1] - cal_irr_acres[1]

def crop2(cal_irr_acres, cal_acre_constraints=cal_acre_constraints):
    return cal_acre_constraints[2] - cal_irr_acres[2]

#create dictionary of constraints
cons = [{'type':'ineq', 'fun': irr_acres_constr},
         {'type':'ineq', 'fun': applied_water_constr},
         {'type':'ineq', 'fun': crop0},
         {'type':'ineq', 'fun': crop1},
         {'type':'ineq', 'fun': crop2},
        ]

#establish parameters for model
guess_irr_acres = [50, 50, 50]

#bounds for model
bnds = ((0, total_available_land), (0, total_available_land), (0, total_avail

results0 = scipy.optimize.minimize(calc_observed_net_revenue, #function to mi
                                  x0=guess_irr_acres,
                                  method='SLSQP',
                                  bounds=bnds,
                                  constraints=cons)
shadow_0 = results.fun - results0.fun

###shadow price crop 1###

def crop0(cal_irr_acres, cal_acre_constraints=cal_acre_constraints):
    return cal_acre_constraints[0] - cal_irr_acres[0]

def crop1(cal_irr_acres, cal_acre_constraints=cal_acre_constraints):
    return cal_acre_constraints[1] - cal_irr_acres[1] + 1

def crop2(cal_irr_acres, cal_acre_constraints=cal_acre_constraints):
    return cal_acre_constraints[2] - cal_irr_acres[2]

#create dictionary of constraints
cons = [{'type':'ineq', 'fun': irr_acres_constr},
         {'type':'ineq', 'fun': applied_water_constr},
         {'type':'ineq', 'fun': crop0},
         {'type':'ineq', 'fun': crop1},
         {'type':'ineq', 'fun': crop2},
        ]

#establish parameters for model
guess_irr_acres = [50, 50, 50]

#bounds for model
bnds = ((0, total_available_land), (0, total_available_land), (0, total_avail
```

```
results1 = scipy.optimize.minimize(calc_observed_net_revenue, #function to mi
                                   x0=guess_irr_acres,
                                   method='SLSQP',
                                   bounds=bnds,
                                   constraints=cons)
shadow_1 = results.fun - results1.fun

###shadow price crop 2###

def crop0(cal_irr_acres, cal_acre_constraints=cal_acre_constraints):
    return cal_acre_constraints[0] - cal_irr_acres[0]

def crop1(cal_irr_acres, cal_acre_constraints=cal_acre_constraints):
    return cal_acre_constraints[1] - cal_irr_acres[1]

def crop2(cal_irr_acres, cal_acre_constraints=cal_acre_constraints):
    return cal_acre_constraints[2] - cal_irr_acres[2] + 1

#create dictionary of constraints
cons = [{'type':'ineq', 'fun': irr_acres_constr},
        {'type':'ineq', 'fun': applied_water_constr},
        {'type':'ineq', 'fun': crop0},
        {'type':'ineq', 'fun': crop1},
        {'type':'ineq', 'fun': crop2},
       ]

#establish parameters for model
guess_irr_acres = [50, 50, 50]

#bounds for model
bnds = ((0, total_available_land), (0, total_available_land), (0, total_avail

results2 = scipy.optimize.minimize(calc_observed_net_revenue, #function to mi
                                   x0=guess_irr_acres,
                                   method='SLSQP',
                                   bounds=bnds,
                                   constraints=cons)
shadow_2 = results.fun - results2.fun

lagrange_mults = [shadow_0, shadow_1, shadow_2]
```

In [11]:

```
#calculate PMP parameters
alpha = []
gamma = []

for i in range(len(lagrange_mults)):
    alpha.append(crop_costs[i] - lagrange_mults[i])
    gamma.append(2 * lagrange_mults[i] / irrigated_crop_acres[i])
```

In [18]:

```
##Solve calibrated model##

def pmp_net_revenue(guess_irr_acres, alpha=alpha, gamma=gamma, applied_water_
    """
    Calculates the total net revenues for all crops using PMP, negative since
    Inputs need to be lists of equal lengths with respective crops lined up
    """
    revenue = 0
    pmp_cost = 0
    for i in range(len(guess_irr_acres)):
        revenue += net_return_per_acre[i] * guess_irr_acres[i]
        pmp_cost += alpha[i] * guess_irr_acres[i] + 0.5 * gamma[i] * guess_ir

    return  -1 * (pmp_cost - revenue)

#setting constraints
water_available = 756 #in acre-ft
land_available = 330 #in acres
def irr_acres_constr(cal_irr_acres, land_available=land_available):
    #limits the total irrigated acres to land available
    return land_available - sum(cal_irr_acres)
def applied_water_constr(cal_irr_acres, applied_water_rates=applied_water_rat
    #limits water applied in calibration to total applied in data
    total_water = 0
    for i in range(len(applied_water_rates)):
        total_water += applied_water_rates[i] * cal_irr_acres[i]
    return water_available - total_water

#create dictionary of constraints
cons = [{'type':'ineq', 'fun': irr_acres_constr},
        {'type':'ineq', 'fun': applied_water_constr}]

#establish parameters for model
guess_irr_acres = [50, 50, 50]

#bounds for model
bnds = ((0, land_available), (0, land_available), (0, land_available))

#run calibrated model
pmp_results = scipy.optimize.minimize(pmp_net_revenue, #function to minimize
                                  x0=guess_irr_acres,
                                  method='SLSQP',
                                  bounds=bnds,
                                  constraints=cons)
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
```