Problem Set Five

Timothy Schwieg

Design Document

This Code should be able to take some input values for $\bar{y}, p, \tau$ and initialization methods for a Newton Method. It should return the Method of Moments estimators for pi_0 and λ. It will go through this by Iterating Newton steps on the system described by the two moment conditions given in the Problem Ste Before any steps are taken, the function will parse the input, checking to see that they are valid, and then checks to ensure that the initialiations for Newton's Method are feasible. It will then run Newton steps, stopping after 25 iterations if convergence is not reached, and will check the answer for feasibility. If the value of the estimators is not feasible, it will throw an error and blame the initializations.

Flow Chart

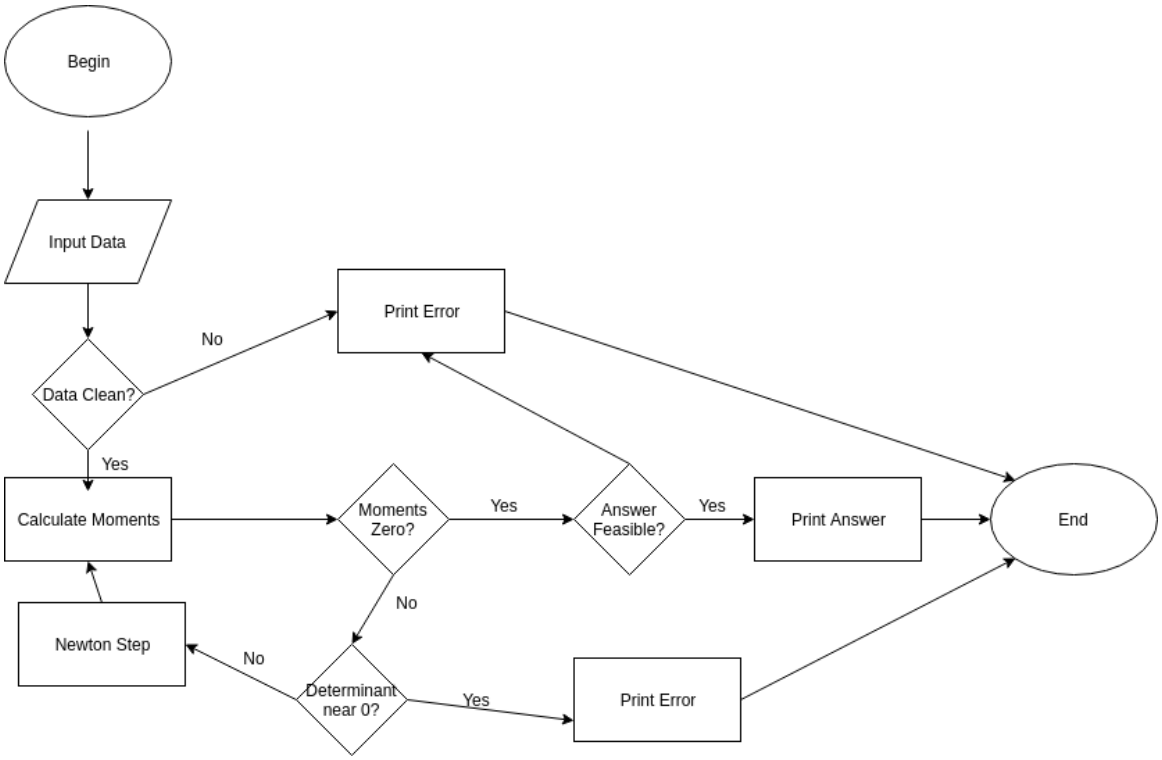

Fig. 1: Program Flow

Test Data

Several Test Data with impossible values such as a negative p, $\bar{y}$ or τ will be tested, as well as impossible initializations such as: $\pi_0 < 0$. The edge cases to test will be: Small $\bar{y} = .01$ and Large $\bar{y} = 500$ We will also try $p = .99$ and $\tau = 1,100$ The last test cases will be where initialization is very far away $\hat{\lambda} = 50, \hat{\pi}_0 = .9$. As we can see, the code does not break with these absurd cases, but it does not suceed either. The second to last case also highlights the importance of initialization being close for $\hat{\lambda}$ and the last case shows its irrelevance for $\hat{\pi}_0$

```
python3 ps5.py -50 .1 3 2 .1
Input error in barY, p, or tau, ITS NOT EVEN WRONG
python3 ps5.py 1 -.1 3 2 .1
Input error in barY, p, or tau, ITS NOT EVEN WRONG
python3 ps5.py 1 .1 -6 2 .1
```

```
Input error in barY, p, or tau, ITS NOT EVEN WRONG
python3 ps5.py .01 .1 3 2 .1
Input led to Matrix with determinant close to zero. You've gone off the reservation
ps5.py 500 .1 3 2 .1
Input error in barY, p, or tau, ITS NOT EVEN WRONG
ps5.py 1 .99 3 2 .1
Input led to Matrix with determinant close to zero. You've gone off the reservation
ps5.py 1 .1 1100 2 .1
Input led to Matrix with determinant close to zero. You've gone off the reservation
ps5.py 1 .1 3 50 .1
Convergence reached, but converged to impossible answer. Initialization problem.
[[  1.00000000e-01]
 [ -2.49045000e+03]]
ps5.py 1 .1 3 2 .99
Reached Convergence with hat pi_0 = 0.0690532611049 and hat lambda = 1.13522746211
```

Code

Running the code we reach the result:

```
ps5.py 1 .1 3 2 .1
Reached Convergence with hat pi_0 = 0.0690532611049 and hat lambda = 1.13522746211
```

```
import numpy as np
import math
import sys
from numpy.linalg import inv

exp = np.exp

def FirstMomentPI( barY, tau, lamda ):
        """This function calculates the value for pi_0 from the first moment restriction, using barY and tau as given"""
        numerator = lamda*barY - 1 + exp( -lamda*tau )
        denom = lamda*tau-1+exp( -lamda*tau )
        return numerator / denom

def SecondMomentPI( p, tau, lamda ):
        """This gives us the value for pi_0 from the second moment restriction, using p, tau as given"""
        numerator = p - exp( -lamda*tau )
        denom = ( 1 - exp( -lamda*tau ) )
        return numerator / denom


def MinFunc( barY, p, tau, lamda ):
        """This returns FirstMomentPI - SecondMomentPI and is the function minimized by the newton method"""
        return FirstMomentPI( barY, tau, lamda ) - SecondMomentPI( p, tau, lamda )

def FirstMomentDeriv( barY, p, tau, lamda ):
        """This returns the derivative of the first moment"""
        firstMomentDeriv = ( (barY - tau )*( 1 - exp( -tau*lamda )*(tau*lamda+1) )) / (((tau*lamda -1) + exp( -tau*lamda) )**2)
        return firstMomentDeriv

def SecondMomentDeriv( barY, p, tau, lamda ):
        """This returns the derivative of the second moment"""
        secondMomentDeriv = -( tau*(p-1)*exp( -tau*lamda) ) / ((1-exp(-tau*lamda))**2)
        return secondMomentDeriv

def BuildJacobian( barY, p, tau, lamda ):
        """This builds the Jacobian used in Newton's Method"""
        mat = np.matrix( [[1,FirstMomentDeriv(barY,p,tau,lamda)],[1,SecondMomentDeriv(barY,p,tau,lamda)]] )
        return mat

def BuildFunctionVec( barY, p, tau, lamda, pi ):
        """This builds the vector that mutliplies with the Jacobian Inverse in the Newton Step"""
        mat = np.matrix( [[pi - FirstMomentPI( barY,tau, lamda )],[pi - SecondMomentPI( p, tau, lamda )]] )
        return mat

def GradientFunc( barY, p, tau, lamda ):
        """This is the gradient of the function defined by MinFunc, used in the 1D Newton Step"""

        firstMomentDeriv = ( (barY - tau )*exp( tau*lamda )*( -tau*lamda + exp( tau*lamda ) - 1 ) ) / ( ( exp( tau*lamda)*(tau*lamda - 1) + 1

        secondMomentDeriv = -( tau*( p - 1)*exp( tau*lamda) )/( exp( tau*lamda) - 1 )**2
        return -(firstMomentDeriv - secondMomentDeriv)

def NewtonStep( barY, p, tau, Xk ):
        """Takes a single variable Newton Step using the tuple lamda value Xk and returns X_(k+1)"""
        return Xk - MinFunc( barY, p, tau, Xk ) / GradientFunc( barY, p, tau, Xk )

def MultiVariateNewtonStep( barY, p, tau, Xk ):
        """This takes a newton step"""
        #We're sinning with a matrix inversion here, but its a 2x2 so it should be clean as long as we don't hit the condition number
        return Xk - inv( BuildJacobian( barY, p, tau, Xk[1,0] ) )*BuildFunctionVec( barY, p, tau, Xk[1,0], Xk[0,0] )


def MutliVariateProblemSetFive( barY, p, tau, lamdaStart, piStart ):
        """This completes the requirements for PS5 using the multivariate Newton's Method"""
        if( barY <= 0  or tau <= 0 or p > 1 or p < 0 or barY > tau ):
                print( "Input_error_in_barY,_p,_or_tau,_ITS_NOT_EVEN_WRONG" )
                return
        if( lamdaStart < 0 or piStart < 0 or piStart > 1 ):
                print( "Input_error_in_lamdaStart_or_piStart,_ITS_NOT_EVEN_WRONG" )
                return

        Xk = np.matrix( [[piStart],[lamdaStart]] )
        steps = 0
```

```
        funcvec = BuildFunctionVec( barY, p, tau, Xk[1,0], Xk[0,0] )
        while( funcvec[0,0]**2 + funcvec[1,0]**2 > .00000001 and steps < 25 ):
                if( np.linalg.det( BuildJacobian( barY, p, tau, Xk[1,0] ) ) < .00000001 ):
                        print( "Input led to Matrix with determinant close to zero. You've gone off the reservation" )
                        return
                Xk = MultiVariateNewtonStep( barY, p, tau, Xk )
                funcvec = BuildFunctionVec( barY, p, tau, Xk[1,0], Xk[0,0] )
                steps += 1

        if( funcvec[0,0]**2 + funcvec[1,0]**2  > .00000001 ):
                print( "Failed to reach Convergence. You're hunting with Ray Charles" )
                return

        if( Xk[0,0] < 0 or Xk[0,0] > 1 or Xk[1,0] < 0 ):
                print( "Convergence reached, but converged to impossible answer. Initialization problem." )
                print( Xk )
                return

        print( "Reached Convergence with hat pi_0 = " + str( Xk[0,0] ) + " and hat lambda = " + str( Xk[1,0] ) )


if len( sys.argv ) < 4 :
        barY = 1
        p = .1
        tau = 3
        lamdaStart = 2
        piStart = .1
elif len(sys.argv) < 6 :
        barY = np.float( sys.argv[1] )
        p = np.float( sys.argv[2] )
        tau = np.float( sys.argv[3] )
        lamdaStart = 2
        piStart = .1
else:
        barY = np.float( sys.argv[1] )
        p = np.float( sys.argv[2] )
        tau = np.float( sys.argv[3] )
        lamdaStart = np.float( sys.argv[4] )
        piStart = np.float( sys.argv[5] )


MutliVariateProblemSetFive( barY, p, tau, lamdaStart, piStart )
```