Empirical Industrial Organisations I: PSet 0

S M Sajid Al Sanai

October 10, 2018

Contents

1 Question I

The Ordinary Least Squares Regression conducted through minimisation of the mean squared error included all regressors with the exclusion of Fixed Effect for Sunday to avoid perfect collinearity. The matrix of regressors X_i includes a vector of ones in the initial column representing x_0. $\hat{\beta}_0$ represents the constant intercept.

1.1 Minimisation with Python using FMin

Minimisation by Python using *scipy.optimize.fmin*() function yields the $\hat{\beta}$ regression coefficients detailed below. The initial guess vector for coefficients used was $(0,0,0,0,0,0,0,0,0)'$. FMin is the Python SciPy library equivalent to Matlab's FMinSearch routine.

```
Warning: Maximum number of function evaluations has been exceeded.
[-0.3670506   -0.0036703    1.01575822  -0.42291809  -0.8131776    0.95473859
 -0.0401332   -0.44040212  -0.01237488]
Beta 0: -0.36705060338769724
Beta 1: -0.003670303573980958
Beta 2: 1.0157582164346053
Beta 3: -0.4229180860932915
Beta 4: -0.8131775992858263
Beta 5: 0.9547385945733078
Beta 6: -0.04013319650279834
Beta 7: -0.44040211745200697
Beta 8: -0.012374877796537835
```

1.2 Minimisation with Python using Basin-Hopping

Minimisation by Python using *scipy.optimize.basinhopping*() function yields the $\hat{\beta}$ regression coefficients detailed below. The initial guess vector for coefficients used was $(0,0,0,0,0,0,0,0,0)'$, with a maximum of four iterations specified.

```
[-1.26705331  -0.00313307   1.01656612   0.84355108  -0.07467264   1.01281306
  0.50899781  -0.8729692   -0.69728724]
Beta 0: -1.2670533120869718
Beta 1: -0.0031330720191008786
Beta 2: 1.0165661182808754
Beta 3: 0.8435510787546923
Beta 4: -0.07467264477145204
Beta 5: 1.0128130639764752
Beta 6: 0.5089978058816674
Beta 7: -0.8729691967245163
Beta 8: -0.6972872427184164
```

1.3 Comparison to Regression using Stata

Due to difficulty with typecasting in *statsmodels.api*, regression was unable to run in Python. The comparison was therefore done using Stata.

As seen in the code snippet of regression coefficients in Ordinary Least Squares Regression generated by Stata, we observe that the estimated coefficients on the regressors derived by the Basin-Hopping iterative method on Python have yielded the correct values as opposed to the FMin iterative method.

The function used was,

reg ARR_DELAY DISTANCE DEP_DELAY FE_MONDAY FE_TUESDAY FE_WEDNESDAY FE_THURSDAY FE_FRIDAY FE_SATURDAY

Source	SS	df	MS
Model	35412319.9	8	4426539.99
Residual	4323520.36	20403	211.90611
Total	39735840.3	20411	1946.78557

Number of obs	=	20412
F(8, 20403)	=	20889.16
Prob > F	=	0.0000
R−squared	=	0.8912
Adj R−squared	=	0.8912
Root MSE	=	14.557

ARR_DELAY	Coef.	Std. Err.	t	P>|t|	[95% Conf.	Interval]
DISTANCE	−.0031331	.0001723	−18.18	0.000	−.0034708	−.0027953
DEP_DELAY	1.016566	.0024917	407.98	0.000	1.011682	1.02145
FE_MONDAY	.8435255	.3917058	2.15	0.031	.0757507	1.6113
FE_TUESDAY	−.0746839	.387552	−0.19	0.847	−.834317	.6849492
FE_WEDNESDAY	1.012869	.3855522	2.63	0.009	.2571558	1.768582
FE_THURSDAY	.5090663	.3785944	1.34	0.179	−.2330092	1.251142
FE_FRIDAY	−.8729858	.3796207	−2.30	0.021	−1.617073	−.1288987
FE_SATURDAY	−.6973007	.4111434	−1.70	0.090	−1.503175	.1085735
_cons	−1.267056	.3148117	−4.02	0.000	−1.884112	−.6499993

2 Question II

Given that the dependent variable created was a discrete binary variable for the late arrival of flights, it was necessary that the summation of log of probabilities was done for both cases of flight arrivals, late or otherwise. In order to do this, I used the dependent variable *arr_late* as a means to index which probability would be active for the particular observation.

When $arr\_late = 0$, $(1 - arr\_late) = 1$ which is the same as evaluation to **True**. Conversely, when $arr\_late = 1$, $(1 - arr\_late) = 0$ which is the same as evaluation to **False**. Therefore, the correct approach would be to multiply $(1 - arr\_late)$ with the probability that the flight is not late, and *arr_late* with the probability that the flight is late in each iterative step of the for-loop over which summation occurs.

$$L(\beta) = \sum_{i=1}^{N} ln \left\{ (1 - arr\_late) \times \left(1 - \frac{e^{\beta X_i}}{1 + e^{\beta X_i}}\right) + (arr\_late) \times \left(\frac{e^{\beta X_i}}{1 + e^{\beta X_i}}\right) \right\}$$

2.1 Optimisation with Python using FMin

Optimisation by Python using $scipy.optimize.fmin()$ function yields the $\hat{\beta}$ regression coefficients detailed below. The initial guess vector for coefficients used was $(0, 0, 0)'$.

```
Optimization terminated successfully.
         Current function value: 4987.810711
         Iterations: 142
         Function evaluations: 259
[-2.61299801e+00 -1.36749723e-04  1.29496431e-01]
Beta 0: -2.6129980089111475
Beta 1: -0.0001367497232092546
Beta 2: 0.12949643091507046
```

2.2 Optimisation with Python using Minimise

Optimisation by Python using *scipy.optimize.minimize*() function yields the $\hat{\beta}$ regression coefficients detailed below. The initial guess vector for coefficients used was $(0, 0, 0)'$.

```
        x: array([-2.61303099e+00, -1.36734311e-04,  1.29495648e-01])
Beta 0: -2.6130309909349796
Beta 1: -0.00013673431142185487
Beta 2: 0.12949564776587152
```

2.3 Comparison to Logistic Regression using Stata

As seen in the code snippet of regression coefficients in Logistic Regression generated by Stata, we observe that the estimated coefficients on the regressors derived by both the FMin and Minimisation iterative method on Python have yielded correct values.

Stata is able to provide both the coefficients and the odds ratio of our specified regressors. Both are detailed below. The functions used were,

$$logistic\ ARR\_LATE\ DISTANCE\ DEP\_DELAY$$

$$logit\ ARR\_LATE\ DISTANCE\ DEP\_DELAY$$

```
Logistic regression                              Number of obs   =      20412
                                                 LR chi2(2)      =   11857.58
                                                 Prob > chi2     =     0.0000
Log likelihood = -4987.8107                      Pseudo R2       =     0.5431

---------------------------------------------------------------------------------
    ARR_LATE  Odds Ratio   Std. Err.       z     P>|z|      [95% Conf. Interval]
---------------------------------------------------------------------------------
    DISTANCE    .9998633    .0000461    -2.96    0.003      .9997729    .9999537
   DEP_DELAY    1.138254    .0023709    62.17    0.000      1.133617     1.14291
       _cons    .0733107    .0036468   -52.53    0.000      .0665005    .0808184
---------------------------------------------------------------------------------
Note: 0 failures and 255 successes completely determined.

Iteration 0:   log likelihood = -10916.599
Iteration 1:   log likelihood = -5100.9292
Iteration 2:   log likelihood = -5000.6587
Iteration 3:   log likelihood = -4987.8287
Iteration 4:   log likelihood = -4987.8107
Iteration 5:   log likelihood = -4987.8107

Logistic regression                              Number of obs   =      20412
                                                 LR chi2(2)      =   11857.58
                                                 Prob > chi2     =     0.0000
Log likelihood = -4987.8107                      Pseudo R2       =     0.5431

---------------------------------------------------------------------------------
    ARR_LATE       Coef.   Std. Err.       z     P>|z|      [95% Conf. Interval]
---------------------------------------------------------------------------------
    DISTANCE   -.0001367    .0000461    -2.96    0.003     -.0002272   -.0000463
   DEP_DELAY    .1294956    .0020829    62.17    0.000      .1254132     .133578
       _cons   -2.613048    .0497444   -52.53    0.000     -2.710545   -2.515551
---------------------------------------------------------------------------------
Note: 0 failures and 255 successes completely determined.
```

3 Question III

The Generalised Method of Moments Estimation required performing a minimisation on a criterion function. In the first stage, one uses the identity matrix as a weighting matrix in the first optimisation, followed by the use of estimated regression coefficients in the construction of a new weighting matrix for a second optimisation. This yields our 2-Stage Least Squares Estimators.

3.1 First Stage Minimisation with Minimize

The first stage function to be minimised was, $(Z'(Y-X\beta))'I_4(Z'(Y-X\beta))$ with $W = I_4$. From this, the weight matrix $W = \Sigma^{-1}(\hat{\beta}) = (\sum_{i=1}^{N}(Y - X\hat{\beta})_i^2 z_i z_i')^{-1}$ to be used in the second stage was calculated. The estimated errors represented the difference in the dependent variable Y and estimated values $X\hat{\beta}$ using the resulting regression coefficients from minimisation with $W = I_4$.

The regression coefficients from the first stage optimisation using the Minimize iterative method *scipy.optimize.minimize*() with an initial guess vector of $(0, 0, 0)'$ are as follows,

```
        x: array([1.91868978, 1.10360019, 3.65262034])
Beta 0: 1.9186897768620108
Beta 1: 1.1036001911432256
Beta 2: 3.652620340522879
```

3.2 Second Stage Minimisation with Minimize

The second stage function to be minimised was $(Z'(Y - X\beta))'W(Z'(Y - X\beta))$ with $W = \Sigma^{-1}(\hat{\beta}) = (\sum_{i=1}^{N}(Y - X\hat{\beta})_i^2 z_i z_i')^{-1}$. Estimated errors were then recalculated as the difference between Y and $X\hat{\beta}$.

The regression coefficients from the second stage optimisation using the Minimize iterative method *scipy.optimize.minimize*() with an initial guess vector of $(0, 0, 0)'$ are as follows,

```
        x: array([1.92104581, 1.09399361, 3.66145241])
Beta 0: 1.9210458074628922
Beta 1: 1.093993605947826
Beta 2: 3.661452412205812
```

3.3 Variance and Standard Errors

The Variance and Standard Errors from the first stage are presented below,

```
GMM Stage 1 Variance
[[ 0.034012    -0.00221611  0.00207005]
 [-0.00221611  0.03475512 -0.00921364]
 [ 0.00207005 -0.00921364  0.02753145]]

GMM Stage 1 Standard Errors
[[0.18442343        nan 0.04549784]
 [       nan 0.18642724        nan]
 [0.04549784        nan 0.16592603]]
```

The Variance and Standard Errors from the second stage are presented below,

```
GMM Stage 2 Variance
[[ 0.03401789 -0.0022231   0.00207841]
 [-0.0022231   0.03475173 -0.00921534]
 [ 0.00207841 -0.00921534  0.02752018]]

GMM Stage 2 Standard Errors
[[0.18443939        nan 0.04558958]
 [       nan 0.18641817        nan]
 [0.04558958        nan 0.16589207]]
```

3.4 Comparison Between Stages

The comparison between the Variance and Standard Errors matrices is the difference between that of the second and first stages. The difference in Variance and Standard Errors are presented below, with *nan* implying 0 or negligible, and the sign representing increase or decrease.

```
GMM Variance Difference
[[ 1.59577010e-05             nan  9.17404363e-05]
 [            nan -9.06897682e-06             nan]
 [ 9.17404363e-05             nan -3.39656336e-05]]

GMM Standard Errors Difference
[[ 1.59577010e-05             nan  9.17404363e-05]
 [            nan -9.06897682e-06             nan]
 [ 9.17404363e-05             nan -3.39656336e-05]]
```

4 Appendix: Source Code

4.1 Source

```
# Import Libraries
import numpy as np
import scipy as sp
from scipy import optimize
from scipy import io
#import statsmodels.api as sm

# Import Dataset
dataset_file = 'airline.csv'
dataset_raw  = open( dataset_file , 'rt' )
dataset_data = np.genfromtxt( dataset_raw , dtype=int , delimiter=',', names=True )

# Dataset Characteristics
N = dataset_data.size

# Generating New ndarray Variables from Dataset
arr_delay    = np.array( dataset_data['ARR_DELAY'] )
dep_delay    = np.array( dataset_data['DEP_DELAY'] )
distance     = np.array( dataset_data['DISTANCE'] )
fe_monday    = np.array( np.empty )
fe_tuesday   = np.array( np.empty )
fe_wednesday = np.array( np.empty )
fe_thursday  = np.array( np.empty )
fe_friday    = np.array( np.empty )
fe_saturday  = np.array( np.empty )
fe_sunday    = np.array( np.empty )

for i in range( N ):
    if ( dataset_data['DAY_OF_WEEK'].item(i) == 1 ):
        fe_monday = np.append( fe_monday , [1] )
        fe_tuesday = np.append( fe_tuesday , [0] )
        fe_wednesday = np.append( fe_wednesday , [0] )
        fe_thursday = np.append( fe_thursday , [0] )
        fe_friday = np.append( fe_friday , [0] )
        fe_saturday = np.append( fe_saturday , [0] )
        fe_sunday = np.append( fe_sunday , [0] )
    if ( dataset_data['DAY_OF_WEEK'].item(i) == 2 ):
        fe_monday = np.append( fe_monday , [0] )
        fe_tuesday = np.append( fe_tuesday , [1] )
        fe_wednesday = np.append( fe_wednesday , [0] )
        fe_thursday = np.append( fe_thursday , [0] )
        fe_friday = np.append( fe_friday , [0] )
        fe_saturday = np.append( fe_saturday , [0] )
        fe_sunday = np.append( fe_sunday , [0] )
    if ( dataset_data['DAY_OF_WEEK'].item(i) == 3 ):
```

```
            fe_monday = np.append( fe_monday, [0] )
            fe_tuesday = np.append( fe_tuesday, [0] )
            fe_wednesday = np.append( fe_wednesday, [1] )
            fe_thursday = np.append( fe_thursday, [0] )
            fe_friday = np.append( fe_friday, [0] )
            fe_saturday = np.append( fe_saturday, [0] )
            fe_sunday = np.append( fe_sunday, [0] )
        if ( dataset_data['DAY_OF_WEEK'].item(i) == 4 ):
            fe_monday = np.append( fe_monday, [0] )
            fe_tuesday = np.append( fe_tuesday, [0] )
            fe_wednesday = np.append( fe_wednesday, [0] )
            fe_thursday = np.append( fe_thursday, [1] )
            fe_friday = np.append( fe_friday, [0] )
            fe_saturday = np.append( fe_saturday, [0] )
            fe_sunday = np.append( fe_sunday, [0] )
        if ( dataset_data['DAY_OF_WEEK'].item(i) == 5 ):
            fe_monday = np.append( fe_monday, [0] )
            fe_tuesday = np.append( fe_tuesday, [0] )
            fe_wednesday = np.append( fe_wednesday, [0] )
            fe_thursday = np.append( fe_thursday, [0] )
            fe_friday = np.append( fe_friday, [1] )
            fe_saturday = np.append( fe_saturday, [0] )
            fe_sunday = np.append( fe_sunday, [0] )
        if ( dataset_data['DAY_OF_WEEK'].item(i) == 6 ):
            fe_monday = np.append( fe_monday, [0] )
            fe_tuesday = np.append( fe_tuesday, [0] )
            fe_wednesday = np.append( fe_wednesday, [0] )
            fe_thursday = np.append( fe_thursday, [0] )
            fe_friday = np.append( fe_friday, [0] )
            fe_saturday = np.append( fe_saturday, [1] )
            fe_sunday = np.append( fe_sunday, [0] )
        if ( dataset_data['DAY_OF_WEEK'].item(i) == 7 ):
            fe_monday = np.append( fe_monday, [0] )
            fe_tuesday = np.append( fe_tuesday, [0] )
            fe_wednesday = np.append( fe_wednesday, [0] )
            fe_thursday = np.append( fe_thursday, [0] )
            fe_friday = np.append( fe_friday, [0] )
            fe_saturday = np.append( fe_saturday, [0] )
            fe_sunday = np.append( fe_sunday, [1] )

# Removing Initial Empty Row in ndarray
fe_monday    = np.delete( fe_monday, (0) )
fe_tuesday   = np.delete( fe_tuesday, (0) )
fe_wednesday = np.delete( fe_wednesday, (0) )
fe_thursday  = np.delete( fe_thursday, (0) )
fe_friday    = np.delete( fe_friday, (0) )
fe_saturday  = np.delete( fe_saturday, (0) )
fe_sunday    = np.delete( fe_sunday, (0) )
```

```
# Question 1:
print('Question 1:')
print('')

print( "N = " + str( N ) )
print('')

# Define in-line SSE function for minimisation
# Fixed Effects for Sunday excluded in model specification
x0 = np.ones( shape=(arr_delay.size, ) )
f_sse = lambda b: np.sum( np.square( arr_delay - b[0] * x0 - b[1] * distance - b[2] * d

# Minimise defined OLS function
# Using FMinSearch
print('Minimisation using FMin')
sse = sp.optimize.fmin( f_sse, [0, 0, 0, 0, 0, 0, 0, 0, 0] )
print( sse )
for i in range(9):
    print( "Beta " + str(i) + ": " + str(sse[i]) )
print('')

# Using Basin-Hopping
print('Minimisation using Basin-Hopping')
sse = sp.optimize.basinhopping( f_sse, [0, 0, 0, 0, 0, 0, 0, 0, 0], 4 )
print( sse.x )
for i in range(9):
    print( "Beta " + str(i) + ": " + str(sse.x[i]) )
print('')
print('')

# Comparison to OLS regression
#regressors = np.concatenate( ( x0, np.array( distance, dtype=float ), np.array( dep_de
#regressors = ( np.reshape( regressors, (9, N) ) ).T
#regression = sm.OLS( exog=arr_delay, endog=regressors, hasconst=True )
#reg_fit    = regression.fit()
#print( reg_fit.summary() )


# Question 2:
print('Question 2:')
print('')

# Generate Binary Variable for Flights Arriving Later than 15 minutes
arr_late = np.array( np.empty, dtype=bool )
for i in range(N):
    if ( arr_delay[i] > 15 ):
        arr_late = np.append( arr_late, [1] )
    else:
        arr_late = np.append( arr_late, [0] )
arr_late = np.delete( arr_late, (0) )
```

```
#Define in-line function for minimisation
f_mle = lambda b: np.sum( -arr_late * ( b[0] * x0 + b[1] * distance + b[2] * dep_delay

# Minimise defined MLE function
# Using FMinSearch
print('Minimisation using FMin')
mle = sp.optimize.fmin( f_mle, [0, 0, 0] )
print( mle )
for i in range(3):
    print( "Beta " + str(i) + ": " + str(mle[i]) )
print('')

# Using Minimise
print('Minimisation using Minimise')
mle = sp.optimize.minimize( f_mle, [0, 0, 0] )
print( mle )
for i in range(3):
    print( "Beta " + str(i) + ": " + str(mle.x[i]) )
print('')
print('')


# Question 3:
print('Question 3:')
print('')

# Import Dataset
dataset_file = 'IV.mat'
dataset_raw  = sp.io.loadmat( dataset_file )

# Generating New ndarray Variables from Dataset
sp.io.whosmat( dataset_file )
Y  = np.array( dataset_raw['Y'] )
X  = np.array( dataset_raw['X'] )
X0 = X[:,0]
X1 = X[:,1]
X2 = X[:,2]
X0 = np.reshape( X0, (-1, 1) )
X1 = np.reshape( X1, (-1, 1) )
X2 = np.reshape( X2, (-1, 1) )
Z  = np.array( dataset_raw['Z'] )
Z0 = Z[:,0]
Z1 = Z[:,1]
Z2 = Z[:,2]
Z3 = Z[:,3]
Z0 = np.reshape( Z0, (-1, 1) )
Z1 = np.reshape( Z1, (-1, 1) )
Z2 = np.reshape( Z2, (-1, 1) )
Z3 = np.reshape( Z3, (-1, 1) )
```

```
I  = np.identity( 4 )

# Dataset Characteristics
N = Y.size
print( "N = " + str( N ) )
print('')

#Define first stage function to be minimised
def f_gmm1(b):
    bX  = np.matmul( b, X.T )
    bX  = bX.T
    e   = np.subtract( Y, bX )
    e   = np.diag( e )
    gw  = np.matmul( Z.T, e )
    return np.matmul( gw.T, gw )

gmm1 = sp.optimize.minimize( f_gmm1, [0, 0, 0] )
print( gmm1 )
for i in range(3):
    print( "Beta " + str(i) + ": " + str(gmm1.x[i]) )
print('')

e = np.matmul( gmm1.x, X.T )
e = e.T
e = np.subtract( Y, e )
e = np.diag( e )

# Constructing Weight Matrix
W = np.zeros( (4, 4) )
for i in range(N):
    w = e[i] * Z[i,:]
    w = np.reshape( w, (4, 1) )
    W = W + np.dot( w, w.T )
W = np.linalg.inv( W )

# Computing Standard Errors
Q = np.matmul( Z.T, X )
C = np.dot( np.dot( Q.T, W ), Q )
gmm1_variance = np.linalg.inv( C )

print( 'GMM Stage 1 Variance' )
print( gmm1_variance )
print('')

gmm1_stderror = gmm1_variance
for j in range(3):
    for i in range(3):
        gmm1_stderror[i, j] = np.sqrt( gmm1_variance[i, j] )

print( 'GMM Stage 1 Standard Errors' )
```

```python
print( gmm1_stderror )
print('')

#Define second stage function to be minimised
def f_gmm2(b):
    bX  = np.matmul( b, X.T )
    bX  = bX.T
    e   = np.subtract( Y, bX )
    e   = np.diag( e )
    gw  = np.matmul( Z.T, e )
    out = np.matmul( gw.T, W )
    out = np.matmul( out, gw )
    return out

gmm2 = sp.optimize.minimize( f_gmm2, [0, 0, 0] )
print( gmm2 )
for i in range(3):
    print( "Beta " + str(i) + ": " + str(gmm2.x[i]) )
print('')

e = np.matmul( gmm2.x, X.T )
e = e.T
e = np.subtract( Y, e )
e = np.diag( e )

# Constructing Weight Matrix
W = np.zeros( (4, 4) )
for i in range(N):
    w = e[i] * Z[i,:]
    w = np.reshape( w, (4, 1) )
    W = W + np.dot( w, w.T )
W = np.linalg.inv( W )

# Computing Standard Errors
Q = np.matmul( Z.T, X )
C = np.dot( np.dot( Q.T, W ), Q )
gmm2_variance = np.linalg.inv( C )

print( 'GMM Stage 2 Variance' )
print( gmm2_variance )
print('')

gmm2_stderror = gmm2_variance
for j in range(3):
    for i in range(3):
        gmm2_stderror[i, j] = np.sqrt( gmm2_variance[i, j] )

print( 'GMM Stage 2 Standard Errors' )
print( gmm2_stderror )
print('')
```

```
# Comparison of Standard Errors
print( 'GMM Variance Difference' )
print( gmm2_variance - gmm1_variance )
print('')

print( 'GMM Standard Errors Difference' )
print( gmm2_stderror - gmm1_stderror )
print('')

# EOF
```

4.2 Output

```
RESTART: C:\Users\Dell\Documents\Graduate - Economics\Empirical IO I\
    pset0_SMSajidAlSanai\pset0_SMSajidAlSanai.py
Question 1:

N = 20412

Minimisation using FMin
Warning: Maximum number of function evaluations has been exceeded.
[-0.3670506  -0.0036703   1.01575822 -0.42291809 -0.8131776   0.95473859
 -0.0401332  -0.44040212 -0.01237488]
Beta 0: -0.36705060338769724
Beta 1: -0.003670303573980958
Beta 2: 1.0157582164346053
Beta 3: -0.4229180860932915
Beta 4: -0.8131775992858263
Beta 5: 0.9547385945733078
Beta 6: -0.04013319650279834
Beta 7: -0.44040211745200697
Beta 8: -0.012374877796537835

Minimisation using Basin-Hopping
[-1.26703813 -0.00313308  1.01656563  0.84356766 -0.07467908  1.01283905
  0.50906709 -0.87301486 -0.69732288]
Beta 0: -1.2670381300700035
Beta 1: -0.003133082060877313
Beta 2: 1.0165656290708287
Beta 3: 0.8435676647590163
Beta 4: -0.07467907921680436
Beta 5: 1.0128390477318374
Beta 6: 0.5090670858834021
Beta 7: -0.8730148589627836
Beta 8: -0.6973228763733291
```

```
Question 2:

Minimisation using FMin
Optimization terminated successfully.
         Current function value: 4987.810711
         Iterations: 142
         Function evaluations: 259
[-2.61299801e+00 -1.36749723e-04  1.29496431e-01]
Beta 0: -2.6129980089111475
Beta 1: -0.0001367497232092546
Beta 2: 0.12949643091507046

Minimisation using Minimise
      fun: 4987.810709922602
 hess_inv: array([[ 3.67335564e-06,  3.20140048e-08, -3.63433468e-06],
       [ 3.20140048e-08,  4.14347694e-09, -3.20864615e-08],
       [-3.63433468e-06, -3.20864615e-08,  3.63374805e-06]])
      jac: array([1.83105469e-04, 2.18688965e-01, 1.95312500e-03])
  message: 'Desired error not necessarily achieved due to precision loss.'
     nfev: 156
      nit: 19
     njev: 31
   status: 2
  success: False
        x: array([-2.61303099e+00, -1.36734311e-04,  1.29495648e-01])
Beta 0: -2.6130309909349796
Beta 1: -0.00013673431142185487
Beta 2: 0.12949564776587152


Question 3:

N = 1000

      fun: 21424.513095692088
 hess_inv: array([[ 7.38969124e-07, -1.79826688e-08,  3.32671680e-07],
       [-1.79826688e-08,  2.12417040e-07,  8.96671604e-08],
       [ 3.32671680e-07,  8.96671604e-08,  1.94939051e-07]])
      jac: array([0.00463867, 0.00634766, 0.00610352])
  message: 'Desired error not necessarily achieved due to precision loss.'
     nfev: 315
      nit: 8
     njev: 61
   status: 2
  success: False
        x: array([1.91868978, 1.10360019, 3.65262034])
Beta 0: 1.9186897768620108
Beta 1: 1.1036001911432256
Beta 2: 3.652620340522879
```

```
GMM Stage 1 Variance
[[ 0.034012   -0.00221611  0.00207005]
 [-0.00221611  0.03475512 -0.00921364]
 [ 0.00207005 -0.00921364  0.02753145]]

GMM Stage 1 Standard Errors
[[0.18442343        nan 0.04549784]
 [       nan 0.18642724        nan]
 [0.04549784        nan 0.16592603]]

      fun: 1.1077417432251324
 hess_inv: array([[ 0.01700604, -0.00110826,  0.00103492],
       [-0.00110826,  0.01737666, -0.00460659],
       [ 0.00103492, -0.00460659,  0.01376592]])
      jac: array([2.68220901e-07, 4.17232513e-07, 3.27825546e-07])
  message: 'Optimization terminated successfully.'
     nfev: 50
      nit: 8
     njev: 10
   status: 0
  success: True
        x: array([1.92104581, 1.09399361, 3.66145241])
Beta 0: 1.9210458074628922
Beta 1: 1.093993605947826
Beta 2: 3.661452412205812

GMM Stage 2 Variance
[[ 0.03401789 -0.0022231   0.00207841]
 [-0.0022231   0.03475173 -0.00921534]
 [ 0.00207841 -0.00921534  0.02752018]]

GMM Stage 2 Standard Errors
[[0.18443939        nan 0.04558958]
 [       nan 0.18641817        nan]
 [0.04558958        nan 0.16589207]]

GMM Variance Difference
[[ 1.59577010e-05             nan  9.17404363e-05]
 [            nan -9.06897682e-06             nan]
 [ 9.17404363e-05             nan -3.39656336e-05]]

GMM Standard Errors Difference
[[ 1.59577010e-05             nan  9.17404363e-05]
 [            nan -9.06897682e-06             nan]
 [ 9.17404363e-05             nan -3.39656336e-05]]
```