st002

Solving Computable General Equilibrium Models with SAS

Ban Chuan Cheah, Westat, Rockville, MD

Abstract

Computable general equilibrium (CGE) models are a system of (usually) non-linear equations that describe the behavior of a system. This model is generally used for what-if analysis: for instance, what happens to variable x if variable y changes by 0.05? CGE models can be found in fields like economics, and environmental, agricultural, and natural resource modelling. This paper demonstrates how to solve a computable general equilibrium model in economics. The model that is presented is a standard model that describes a simple economy without taxes. CGE models can be solved using SAS/ETS®(via PROC MODEL) and SAS/OR®(via PROC NLP). The shortcomings associated with each method are also noted.

1 Introduction

Economists have long used SAS®for its data step processing as well as its STAT®package for running regressions and statistical analysis. Less well-known is its ability to solve equations. This paper demonstrates two methods for solving equations. One method uses PROC MODEL which is available on the ETS®package and the other uses PROC NLP which is available from the OR®package. In both cases, we solve a standard 2-sector, 2-factor, 2-good (2x2x2) computable general equilibrium (CGE) model of Shoven and Whalley (Shoven and Whalley, 1992). While not as powerful as other software for solving equations such as GAMS (General Algebraic Modelling System), SAS has the advantage of being easier to use, understand, and set up. For each method used to solve the model, the deficiencies will also be noted.

CGE models are commonly used to produce counter-factual policy analysis.[1] For instance, the question commonly asked is what is the equilibrium (market clearing) quantities and prices if taxes had been at x instead of y? Considerations are given to the existence of such an equilibrium and the solution of this equilibrium when it exists. In finding the solution, economists have relied on some form of a fixed point theorem that guarantees convergence to the equilibrium. In practice however, finding the equilibrium is reduced to solving a system of nonlinear equations – the most common method is some form of Newton's algorithm, which does not however, guarantee convergence. As practitioners can attest, finding the solution to a system of non-linear equations can sometimes be more of an art than a science – it is dependent on the choice of initial values as well as how the problem is specified.

The simple Shoven-Whalley (Shoven and Whalley, 1992) model without taxes is presented next. Modifications to allow for tax analysis can easily be accommodated. The SAS statements used to produce the solutions using PROC MODEL and PROC NLP are then presented and discussed.

2 The Shoven-Whalley Model

The Shoven-Whalley model is an economy that consists of 2 sectors that produces manufacturing(Sector 1) and non-manufacturing goods(Sector 2), 2 factors of production (capital and labor), and 2 types of

[1] An inspection of the GAMS website www.gams.com shows that CGE is also used in environmental, agricultural, and natural resource modelling.

consumers. Consumers have initial endowments of factors but no goods. Rich consumers (Type R) own all of the capital while poor consumers (Type P) own all of the labor. Both types of goods are produced by a constant elasticity of substitution (CES) production function which differ in terms of parameter values. Consumers have demand functions for each type of good that are derived from maximizing a CES utility function subject to a budget constraint.

The Shoven-Whalley model is summarized by the following functional forms and parameter values shown in Table 1. The parameter α and μ in the utility functions are respectively, the share of income spent on the good and the elasticity of substitution between the two goods. $\bar{L}$ and $\bar{K}$ are respectively the initial endowments of labor and capital. On the production side, Φ is a scale factor, δ is the weight of that factor used in production, and σ is the elasticity of substitution in factors of production.

In addition to the goods and factor demand equations, the following constraints will also be required. The first is the cost minimizing factor demand equations per unit of output for good j, $j = 1, 2$. Given factor prices r and w (interest rates and wages respectively),

$$\frac{L^j}{Q^j} = l^j(r, w, 1) \quad (1)$$

$$\frac{K^j}{Q^j} = k^j(r, w, 1) \quad (2)$$

The second is the zero profit condition.

$$p_j(r, w) = wl^j(r, w, 1) + rk^j(r, w, 1) \quad (3)$$

The third constraint says that quantities produced are equal to quantities demanded – goods markets clear.

$$Q^j(r, w) = \sum_{m=R,P} X_j^m(r, w) \quad (4)$$

The final constraint says the factor markets must also clear – the quantities of labor and capital demanded are equal to that supplied.

$$\sum_{j=1}^{2} K^j(r, w) = \sum_{m=R,P} \bar{K}^m \quad (5)$$

$$\sum_{j=1}^{2} L^j(r, w) = \sum_{m=R,P} \bar{L}^m \quad (6)$$

$$(7)$$

Note that a CGE model can (and usually does) have more than 2 goods and 2 sectors so that the summation is over all of the goods and sectors.[2] This model will have 13 unknown values which will have to be solved for.

[2]GAMS provides a very compact method of declaring equations using indices which simplifies setting up the model especially when there are many sectors.

Utility functions	A. Demand Parameter Values	Demand functions
$U^m = \left(\sum_{i=1}^{2} (\alpha_i^m)^{1/\mu^m} (x_i^m)^{(\mu^m-1)/\mu^m}\right)^{\mu^m/(\mu^m-1)}$	$(\alpha_1^R, \alpha_2^R) = (0.5, 0.5)$	$x_i^m = \alpha_i^m \frac{(w\bar{L}^m + r\bar{K}^m)}{p_i^{\mu^m}(\sum_{i=1}^{2} \alpha_i^m p_i^{(1-\mu^m)}}$
(m=R,P)	$(\alpha_1^P, \alpha_2^P) = (0.3, 0.7)$	(m=R,P and i=1,2)
	$(\mu^R, \mu^P) = (1.5, 0.75)$	
	$(\bar{K}^R, \bar{K}^P) = (25, 0)$	
	$(\bar{L}^R, \bar{L}^P) = (0, 60)$	
Production functions	**B. Production Parameter Values**	**Factor Demand functions**
$Q^j = \Phi^j \left(\delta^j L^{j^{(\sigma^j-1)/\sigma^j}} + (1-\delta^j) K^{j^{(\sigma^j-1)/\sigma^j}}\right)^{\sigma^j/(\sigma^j-1)}$	$(\Phi^1, \Phi^2) = (1.5, 2.0)$	$L^j = \frac{1}{\Phi^j} Q^j \left[\delta^j + (1-\delta^j)\left(\frac{\delta^j}{(1-\delta^j)w}\right)^{(1-\sigma^j)}\right]^{\sigma^j/(1-\sigma^j)}$
(j=1 $[manufacturing]$, 2 $[non-manufacturing]$)	$(\delta^1, \delta^2) = (0.6, 0.7)$	$K^j = \frac{1}{\Phi^j} Q^j \left[\delta^j \left(\frac{(1-\delta^j)w}{\delta^j}\right)^{(1-\sigma^j)} + (1-\delta^j)\right]^{\sigma^j/(1-\sigma^j)}$
	$(\sigma^1, \sigma^2) = (2.0, 0.5)$	

Table 1: Parameters and functional forms for the numerical examples of a 2-function, 2-sector general equilibrium model in Shoven and Whalley (1984)

3 Solving the model with PROC MODEL

One way to solve the model using SAS is to use PROC MODEL which is available on SAS/ETS®. The code used to describe and solve the model is presented below. Defining the complicated mathematical equations is usually the most tedious part. First a data set `parms` is created containing the parameter values. Because the equations are complicated they are broken down into smaller subparts.

```
/* Data set for parameters */

data parms;
    /* Demand values */
    alpha1_r=0.5; alpha2_r=0.5;
    alpha1_p=0.3; alpha2_p=0.7;
    mu_r=1.5; mu_p=0.75;
    kbar_r=25; kbar_p=0;
    lbar_r=0; lbar_p=60;

    /* Production parameters */
    phi1=1.5; phi2=2.0;
    delta1=0.6; delta2=0.7;
    sigma1=2.0; sigma2=0.5;
    w=1.0;
run;

proc model data=parms;
    xr_denom=alpha1_r*(p1**(1-mu_r))+alpha2_r*(p2**(1-mu_r));
    xr_numer=w*Lbar_r+r*Kbar_r;
    xp_denom=alpha1_p*(p1**(1-mu_p))+alpha2_p*(p2**(1-mu_p));
    xp_numer=w*Lbar_p+r*Kbar_p;
    good1exp=sigma1/(1-sigma1);
    good2exp=sigma2/(1-sigma2);
    onesig1=1-sigma1;
    onesig2=1-sigma2;
    onedelt1=1-delta1;
    onedelt2=1-delta2;
    phi1q1=1/phi1*Q1;
    phi2q2=1/phi2*Q2;
    L1_inner1=(delta1*r)/(onedelt1*w);
    L1_inner2=L1_inner1**onesig1;
    L1_inner3=L1_inner2*onedelt1;
    L1_inner4=L1_inner3+delta1;
    L1_inner5=L1_inner4**good1exp;
    L2_inner1=(delta2*r)/(onedelt2*w);
    L2_inner2=L2_inner1**onesig2;
    L2_inner3=L2_inner2*onedelt2;
    L2_inner4=L2_inner3+delta2;
    L2_inner5=L2_inner4**good2exp;
    K1_inner1=(onedelt1*w)/(delta1*r);
    K1_inner2=K1_inner1**onesig1;
    K1_inner3=delta1*K1_inner2;
    K1_inner4=K1_inner3+onedelt1;
    K1_inner5=K1_inner4**good1exp;
    K2_inner1=(onedelt2*w)/(delta2*r);
    K2_inner2=K2_inner1**onesig2;
    K2_inner3=delta2*K2_inner2;
    K2_inner4=K2_inner3+onedelt2;
    K2_inner5=K2_inner4**good2exp;
```

```
    /* Demand functions */
    eq.lx1_r=x1_r - ((alpha1_r*xr_numer)/((p1**mu_r)*xr_denom));
    eq.lx2_r=x2_r - ((alpha2_r*xr_numer)/((p2**mu_r)*xr_denom));
    eq.lx1_p=x1_p - ((alpha1_p*xp_numer)/((p1**mu_p)*xp_denom));
    eq.lx2_p=x2_p - ((alpha2_p*xp_numer)/((p2**mu_p)*xp_denom));
    /* Factor demand functions */
    eq.lL1=L1-(phi1q1*L1_inner5);
    eq.lL2=L2-(phi2q2*L2_inner5);
    eq.lK1=K1-(phi1q1*K1_inner5);
    eq.lK2=K2-(phi2q2*K2_inner5);
    /* Excess factor demand functions */
*   eq.lcapital=(K1+K2)-(Kbar_r+Kbar_p); /* originally included */
    eq.llabor=(L1+L2)-(Lbar_r+Lbar_p); /* originally included */
    /* Commodity prices */
    eq.lp1=p1-((w*L1/Q1)+(r*K1/Q1));
    eq.lp2=p2-((w*L2/Q2)+(r*K2/Q2));
    /* Commodity demands */
    eq.lQ1=Q1-(x1_p+x1_r);
    eq.lQ2=Q2-(x2_p+x2_r); /* originally left out */

    solve r p1 p2 x1_r x2_r x1_p x2_p L1 L2 K1 K2 Q1 Q2
          / solveprint;
run;
quit;
```

The keyword `solve` is followed by a list of variables to be solved for. The option `solveprint` requests that the solution and residual values be printed for each observation. PROC MODEL solves this system in 28 iterations. The output that is produced is shown below[3]. The sections that are of interest are the Solution Values and the Solution Summary. The first item to check is whether the model converged. Under the section Solution Summary, the item "CONVERGE=" gives the number of decimal places where convergence was obtained. It also gives the number of iterations required to obtain convergence as well as the solution method (NEWTON which is the default).

The SAS System
The MODEL Procedure

Model Summary	
Model Variables	13
Equations	13
Number of Statements	45

Model Variables	$p1$ $p2$ r $Q1$ $Q2$ $x1\_r$ $x2\_r$ $x1\_p$ $x2\_p$ $L1$ $L2$ $K1$ $K2$
Equations	$lx1\_r$ $lx2\_r$ $lx1\_p$ $lx2\_p$ $lL1$ $lL2$ $lK1$ $lK2$ $llabor$ $lp1$ $lp2$ $lQ1$ $lQ2$

The SAS System
The MODEL Procedure

Simultaneous Simulation

Observation	1	Iterations	28	CC	0.000000	eq.lK2	-0.000000

Solution Values						
r	p1	p2	$x1_r$	$x2_r$	$x1_p$	$x2_p$
1.37347	1.39911	1.09308	11.51465	16.67451	13.42782	37.70366

[3]The SAS output in this paper is produced using ODS LaTeX. In Version 8.2 ODS LaTeXis experimental and heavy editing was required to get the output to compile.

Solution Values					
L1	L2	K1	K2	Q1	Q2
26.36558	33.63442	6.21178	18.78822	24.94247	54.37817

```
The SAS System
The MODEL Procedure

Simultaneous Simulation
```

Data Set Options	
DATA=	PARMS

Solution Summary	
Variables Solved	13
Implicit Equations	13
Solution Method	NEWTON
CONVERGE=	1E-8
Maximum CC	2.94E-15
Maximum Iterations	28
Total Iterations	28
Average Iterations	28

Observations Processed	
Read	1
Solved	1

Variables Solved For	r $p1$ $p2$ $x1\_r$ $x2\_r$ $x1\_p$ $x2\_p$ $L1$ $L2$ $K1$ $K2$ $Q1$ $Q2$
Equations Solved	$lx1\_r$ $lx2\_r$ $lx1\_p$ $lx2\_p$ $lL1$ $lL2$ $lK1$ $lK2$ $lp1$ $lp2$ $lQ1$ $lQ2$ $llabor$

Anyone who has done equation solving can attest to how quirky this process can be when the model does not converge. Unfortunately, there are few options that PROC Model has to offer. While the documentation indicates that the SOLVE statement with the INITIAL option can be used to specify alternate starting values, this does not work when performing equation solving.[4]

Whether the model converges can also depend on the system of equations specified. There is usually more than one way to specify the model. In the above example, note the two equations with the comment `/* originally included */` labelled in the code statements as eq.lcapital and eq.llabor. In this version, the equation eq.lcapital has been commented out. If we now include this equation in the model and exclude the equation with the comment `/* originally left out */` labelled as eq.LQ2 then SAS responds with the following error:

```
The solution failed because 4 equations are missing or have extreme values
for observation 1 at NEWTON iteration 1.
```

There is no way to escape from this error without respecifying the model.[5]

4 Solving the model with PROC NLP

Another option to solving the model is to use PROC NLP which is available with SAS/OR®. The system of equations can be considered as (nonlinear) constraints to some optimization problem (either a maximization or a minimization, it does not matter). An arbitrary constant can be used as the function to minimize. In this case, the variable *l_dummy* is assigned a value of 10 and is used as the objective function f which will be minimized. All the equations used in PROC Model are restated as nonlinear boundary conditions with the right hand side set to 0. The code used for this is shown below:

[4] See SAS Note at support.sas.com/techsup/unotes/V6/G/G577.html.

[5] Microsoft Excel's equation solver however, did not have any problems with this specification.

```
proc nlp;
    min f;
    decvar r p1 p2 x1_r x2_r x1_p x2_p L1 L2 K1 K2 Q1 Q2;
    nlincon nl1-nl13=0;
    l_dummy = 10;
    f=abs(l_dummy);

    /* Equations */
    alpha1_r=0.5; alpha2_r=0.5; alpha1_p=0.3; alpha2_p=0.7;
    mu_r=1.5; mu_p=0.75; kbar_r=25; kbar_p=0; lbar_r=0; lbar_p=60;
    phi1=1.5; phi2=2.0; delta1=0.6; delta2=0.7; sigma1=2.0; sigma2=0.5; w=1.0;
    xr_denom=alpha1_r*(p1**(1-mu_r))+alpha2_r*(p2**(1-mu_r));
    xr_numer=w*Lbar_r+r*Kbar_r;
    xp_denom=alpha1_p*(p1**(1-mu_p))+alpha2_p*(p2**(1-mu_p));
    xp_numer=w*Lbar_p+r*Kbar_p;
    good1exp=sigma1/(1-sigma1);
    good2exp=sigma2/(1-sigma2);
    onesig1=1-sigma1;
    onesig2=1-sigma2;
    onedelt1=1-delta1;
    onedelt2=1-delta2;
    phi1q1=1/phi1*Q1;
    phi2q2=1/phi2*Q2;
    L1_inner1=(delta1*r)/(onedelt1*w);
    L1_inner2=L1_inner1**onesig1;
    L1_inner3=L1_inner2*onedelt1;
    L1_inner4=L1_inner3+delta1;
    L1_inner5=L1_inner4**good1exp;
    L2_inner1=(delta2*r)/(onedelt2*w);
    L2_inner2=L2_inner1**onesig2;
    L2_inner3=L2_inner2*onedelt2;
    L2_inner4=L2_inner3+delta2;
    L2_inner5=L2_inner4**good2exp;
    K1_inner1=(onedelt1*w)/(delta1*r);
    K1_inner2=K1_inner1**onesig1;
    K1_inner3=delta1*K1_inner2;
    K1_inner4=K1_inner3+onedelt1;
    K1_inner5=K1_inner4**good1exp;
    K2_inner1=(onedelt2*w)/(delta2*r);
    K2_inner2=K2_inner1**onesig2;
    K2_inner3=delta2*K2_inner2;
    K2_inner4=K2_inner3+onedelt2;
    K2_inner5=K2_inner4**good2exp;
    /* Demand functions */
    nl1=x1_r - ((alpha1_r*xr_numer)/((p1**mu_r)*xr_denom));
    nl2=x2_r - ((alpha2_r*xr_numer)/((p2**mu_r)*xr_denom));
    nl3=x1_p - ((alpha1_p*xp_numer)/((p1**mu_p)*xp_denom));
    nl4=x2_p - ((alpha2_p*xp_numer)/((p2**mu_p)*xp_denom));
    /* Factor demand functions */
    nl5=(L1-(phi1q1*L1_inner5));
    nl6=L2-(phi2q2*L2_inner5);
    nl7=K1-(phi1q1*K1_inner5);
    nl8=K2-(phi2q2*K2_inner5);
    /* Excess factor demand functions */
*   nl13=(K1+K2)-(Kbar_r+Kbar_p); /* originally included */
    nl9=(L1+L2)-(Lbar_r+Lbar_p); /* originally included */
```

```
    /* Commodity prices */
    nl10=p1-((w*L1/Q1)+(r*K1/Q1));
    nl11=p2-((w*L2/Q2)+(r*K2/Q2));
    /* Commodity demands */
    nl12=Q1-(x1_p+x1_r);
    nl13=Q2-(x2_p+x2_r); /* originally left out */
run;
```

The keyword **decvar** declares the variables to be solved. The function to be minimized is a dummy function f that is assigned a value of 10. The keyword to do this is **min** followed by the name of the variable. In all, 13 nonlinear conditions are specified numbered here as nl1 to nl13. I generally use the statement **nlincon** even though the equation may appear to be linear in its variables, it may not be linear in its underlying parameters. Occasionally, SAS will respond with the following warning in the log.

```
WARNING: Your program statements cannot be executed completely.
WARNING: Your program statements cannot be executed completely.
WARNING: Your program statements cannot be executed completely.
WARNING: Your program statements cannot be executed completely.
WARNING: Your program statements cannot be executed completely.
WARNING: In a total of 5 calls an error occurred during execution
         of the program statements.  NLP attempted to recover by
         using a shorter step size.
NOTE: The PROCEDURE NLP printed pages 1-4.
```

Even though this model converged successfully, SAS issues a warning that the default step size used in the updating scheme for solving the system was changed in order to execute. One of the advantages of using PROC NLP is the following:

```
NOTE: Your code contains 64 program statements.
NOTE: Gradient is computed using analytic formulas.
NOTE: Jacobian of nonlinear constraints is computed using analytic formulas.
NOTE: Initial value of parameter r is set randomly to 0.0822180873.
NOTE: Initial value of parameter p1 is set randomly to 0.8829999216.
NOTE: Initial value of parameter p2 is set randomly to 0.8726145364.
NOTE: Initial value of parameter x1_r is set randomly to 0.3320990672.
NOTE: Initial value of parameter x2_r is set randomly to 0.1061694609.
NOTE: Initial value of parameter x1_p is set randomly to 0.5099485845.
NOTE: Initial value of parameter x2_p is set randomly to 0.4883645426.
NOTE: Initial value of parameter L1 is set randomly to 0.6836527594.
NOTE: Initial value of parameter L2 is set randomly to 0.9010621798.
NOTE: Initial value of parameter K1 is set randomly to 0.7557513578.
NOTE: Initial value of parameter K2 is set randomly to 0.3506155952.
NOTE: Initial value of parameter Q1 is set randomly to 0.819782799.
NOTE: Initial value of parameter Q2 is set randomly to 0.9638652531.
```

Each time PROC NLP is executed, the parameter values receive a random number to start. Therefore, if convergence is not obtained, the statements can simply be rerun – the initial values will be changed and usually, convergence is reached after two or three attempts.[6] Another advantage to using PROC NLP is that there are other optimization algorithms available using the TECH= and UPD= options. The SAS output from this run is shown below[7]:

```
                          The SAS System

                PROC NLP: Nonlinear Minimization
```

[6]Initial values can also be declared as a list following the **decvar** statement preceded by an = sign.
[7]The size of the output can be reduced using various options such as NOPRINT, PSHORT, PSUMMARY

Gradient is computed using analytic formulas.
Jacobian of nonlinear constraints is computed using analytic formulas.

The SAS System

PROC NLP: Nonlinear Minimization

Optimization Start

Parameter Estimates

N	Parameter	Estimate	Gradient Objective Function	Gradient Lagrange Function
1	r	0.082218	0	0
2	p1	0.883000	0	0
3	p2	0.872615	0	0
4	x1_r	0.332099	0	0
5	x2_r	0.106169	0	0
6	x1_p	0.509949	0	0
7	x2_p	0.488365	0	0
8	L1	0.683653	0	0
9	L2	0.901062	0	0
10	K1	0.755751	0	0
11	K2	0.350616	0	0
12	Q1	0.819783	0	0
13	Q2	0.963865	0	0

Value of Objective Function = 10
Value of Lagrange Function = 10

Values of Nonlinear Constraints

Constraint		Value	Residual	Lagrange Multiplier		
[1]	nl1	-0.8284	-0.8284	0	Violat.	NLEC
[2]	nl2	-1.0751	-1.0751	0	Violat.	NLEC
[3]	nl3	-19.9173	-19.9173	0	Violat.	NLEC
[4]	nl4	-47.6001	-47.6001	0	Violat.	NLEC
[5]	nl5	0.6467	0.6467	0	Violat.	NLEC
[6]	nl6	0.5004	0.5004	0	Violat.	NLEC
[7]	nl7	-1.6768	-1.6768	0	Violat.	NLEC
[8]	nl8	-0.5642	-0.5642	0	Violat.	NLEC
[9]	nl9	-58.4153	-58.4153	0	Violat.	NLEC
[10]	nl10	-0.0267	-0.0267	0	Violat.	NLEC
[11]	nl11	-0.0921	-0.0921	0	Violat.	NLEC
[12]	nl12	-0.0223	-0.0223	0	Violat.	NLEC
[13]	nl13	0.3693	0.3693	0	Violat.	NLEC

The SAS System

PROC NLP: Nonlinear Minimization

Dual Quasi-Newton Optimization
Modified VMCWD Algorithm of Powell (1978, 1982)
Dual Broyden - Fletcher - Goldfarb - Shanno Update (DBFGS)
Lagrange Multiplier Update of Powell(1982)

Parameter Estimates	13
Nonlinear Constraints	13
Nonlinear Equality Constraints	13

Optimization Start			
Objective Function	10	Maximum Constraint Violation	5.9768227132
Maximum Gradient of the Lagran Func	0		

Iteration		Restarts	Function Calls	Maximum Objective Function	Maximum Constraint Violation	Predicted Function Reduction	Step Size	Maximum Gradient Element of the Lagrange Function
1		0	47	10.00000	55.1504	10778.9	1.000	735.3
2		0	48	10.00000	25.0248	33422.5	1.000	347.5
3	'	0	49	10.00000	7.0921	44108.3	1.000	21910
4	'	0	50	10.00000	0.1448	2.0565	1.000	64.057
5	'	0	51	10.00000	0.0120	0.000319	1.000	0.253
6	'	0	52	10.00000	1.564E-6	9.35E-10	1.000	0.00165

Optimization Results			
Iterations	6	Function Calls	53
Gradient Calls	9	Active Constraints	13
Objective Function	10	Maximum Constraint Violation	1.563771E-6
Maximum Projected Gradient	0	Value Lagrange Function	10
Maximum Gradient of the Lagran Func	0	Slope of Search Direction	-9.34553E-10

FCONV2 convergence criterion satisfied.

The SAS System

PROC NLP: Nonlinear Minimization

Optimization Results

Parameter Estimates

N	Parameter	Estimate	Gradient Objective Function	Gradient Lagrange Function
1	r	1.373471	0	0
2	p1	1.399111	0	0
3	p2	1.093076	0	0
4	x1_r	11.514649	0	0
5	x2_r	16.674506	0	0
6	x1_p	13.427823	0	0
7	x2_p	37.703663	0	0
8	L1	26.365584	0	0
9	L2	33.634416	0	0
10	K1	6.211775	0	0
11	K2	18.788222	0	0
12	Q1	24.942472	0	0
13	Q2	54.378169	0	0

Value of Objective Function = 10
Value of Lagrange Function = 10

Values of Nonlinear Constraints						
Constraint		Value	Residual	Lagrange Multiplier	
[1]	nl1	4.045E-7	4.045E-7	0	*?*
[2]	nl2	6.208E-7	6.208E-7	0	*?*
[3]	nl3	-2.38E-7	-2.38E-7	0	*?*
[4]	nl4	-7.54E-7	-7.54E-7	0	*?*
[5]	nl5	1.108E-6	1.108E-6	0	*?*
[6]	nl6	1.188E-6	1.188E-6	0	*?*
[7]	nl7	-1.56E-6	-1.56E-6	0	*?*
[8]	nl8	-1.29E-6	-1.29E-6	0	*?*
[9]	nl9	0	0	0	Active NLEC
[10]	nl10	8.649E-8	8.649E-8	0	*?*
[11]	nl11	4.031E-8	4.031E-8	0	*?*
[12]	nl12	0	0	0	Active NLEC
[13]	nl13	-355E-17	-355E-17	0	Active NLEC

```
The 10 nonlinear constraints which are marked with *?* are not satisfied at the
accuracy specified by the LCEPSILON= option.  However, the default value of this
option seems to be too strong to be applied to nonlinear constraints.
```

The output first shows the initial parameter estimates and whether the nonlinear conditions are violated. There are two main items to verify – that convergence is reached and that all the nonlinear constraints are satisfied. After showing the start values, the output then shows the details of the optimization procedure. Convergence can be determined by checking for the statement `F2CONV convergence criteria satisfied`. Alternatively, we can also look at the value of `Objective Function` in the Optimization Results table. The output then shows the final parameter estimates as well as whether the constraints are active. This is the second item that has to be checked since it is possible to obtain convergence without satisfying all the constraints. We would like all constraints to be active as indicated by the message "Active NLEC" in the output. The constraints marked with `*?*` indicate that the constraints are not satisfied by the default criteria for LCEPSILON. However, inspection of the values for the items `Value` and `Residual` are within acceptable norms for convergence (in the range of 1E-6).[8]

Interestingly, if we remove the asterisk from the equation labelled n13 that had been commented out and reinstate the bottom equation (also labelled n13) as a comment, PROC NLP has no difficulty solving this problem. Recall, this is the specification that PROC Model was unable to find a solution. Similar to PROC MODEL, a data set containing the parameters can be used with the `data=` option[9]

5 Conclusion

This paper shows how PROC MODEL and PROC NLP can be used to solve computable general equilibrium models. PROC MODEL is a straightforward and intuitive method but suffers from problems of specification sensitivity and the inability to specify initial values. PROC NLP works well but is not as elegant since it relies on the minimization (or maximization) of an arbitrary function.[10] Moreover, when convergence is not achieved it has to be restarted manually. However, it is possible to use a macro to check for convergence and to restart the process if required. While GAMS is very well suited for solving CGE models, using SAS has a great deal of advantages as well. PROC MODEL or PROC NLP used in conjunction with ODS simplifies the reporting of results. Moreover, the user has a whole suite of SAS products that can be used to analyze and present the results especially if she is generating a lot different scenarios.

Extending the 2x2x2 model presented here to include taxes is straightforward. Many CGE models are based on social accounting matrices that denote various sectors of the economy. In general, these social accounting matrices can be as large as 20x20 denoting 20 interrelated sectors of the economy. The challenge would be to integrate these large multi-sector models into the above framework.

[8] The SAS Documentation does not indicate the default value for LCEPSILON but it appears to be in the range of 1E-10.

[9] It will not however solve a series of problems with differing parameter values.

[10] PROC NLP would work very well for a variety of typical economic optimization problems common in consumer and firm theory such as utility and profit maximization.

6 References

1. Shoven, John B. and John Whalley, *Applying General Equilibrium*, Cambridge University Press, 1992.
2. *SAS/OR Users Guide: Mathematical Programming*, SAS Institute, Online Documentation.

7 Author Contact Information

Ban Chuan Cheah
Westat
1650 Research Blvd.
Rockville, MD 20850
BanCheah@westat.com

DISCLAIMER: The contents of this paper is the work of the author and does not necessarily represent the opinions, recommendations, or practices of Westat.

SAS®and all other SAS Institute Inc. products or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ®indicates USA registration. Other brand and product names are registered trademark or trademarks of their respective countries.
Other brand and product names are registered trademarks or trademarks of their respective companies.