Economic Modelling using Constraint Logic Programming

Nelson Donovan & David Gilbert
Computer Science Department
City University
Northhampton Square
London EC1V OHB
Tel: + 44 171 477 8444
Fax: + 44 171 477 8587
Email: {be140, drg}@soi.city.ac.uk

Abstract

This paper investigates the use of constraint logic programming (CLP) in economic modelling through the design and implementation of two economic models. The first model, the Desai-Henry model contains only linear equations while the second model, constructed by the author, contains non-linear elements. In order to implement the second model, a non-linear constraint solver was constructed. This was necessary because, although CLP is a very powerful programming paradigm, currently available implementations lack any on-linear constraint solving mechanisms.

1. Introduction

The aim of this research was to implement two models of part of the UK economy using constraint logic programming (CLP). The first model was to consist of only linear equations; the second model to contain non-linear elements. Implementing the second model had associated with it the problem of non-linear constraint solving - no such solving mechanism to the author's knowledge being currently available in commercial CLP languages. Part of the objective of the project was therefore to investigate ways of solving non-linear constraints.

This paper is organised as follows: Section 2 provides a brief introduction to economics, econometrics and Eclipse - the CLP language used in this research; Section 3 details the implementation of first model - the Desai-Henry model; Section 4 describes the implementation of the non-linear model and a non-linear constraint solver; and finally section 5 presents the conclusions of the research and suggests ideas for further work.

2. Background

2.1 Economics and econometrics

Economics is the study of how societies manage, produce and consume resources. Economic knowledge is useful to individuals, businesses and governments alike. Learning about the stockmarket or about interest rates may help individuals manage their finances better; heightened awareness of the determinants of cost and revenue might assist companies in making better business decisions; a knowledge of imperfect competition may help governments control pricing.

Econometrics is the quantitative analysis of economic theory. The end product of econometrics is usually an equation or a series of equations which can be used to build a model describing some behavioural aspect of an economic system. Models are useful because they force the economist to think clearly about the relationships involved in an economic system. They also allow him/her to test theories and make forecasts.

2.2 Eclipse

The constraint programming language used in this project is Eclipse *(*ECRC Common Logic Programming System*)*. Eclipse is a Prolog based system which serves as a platform for integrating various logic programming extensions. The Eclipse kernel is an implementation of standard Prolog. It is built around an incremental compiler which compiles the Prolog source into Warren Abstract Machine-like code, and an emulator of this abstract code. The various constraint logic programming libraries available with Eclipse are:

- library(fd) - finite-domain constraint solver;
- library(r) - linear rational arithmetic constraint solver;
- library(chr) - constraint handling rules system;
- library(propia) - generalised propagation system.

The programs written for this project use library(r). The library(r) solver uses a combination of the Simplex algorithm and Gaussian elimination to solve a system of arithmetic constraints consisting of linear inequalities and inequalities. Instead of the usual syntax for the rational constraint relations (i.e. '=', '<', '>', '<=', etc.), the symbols '$=', '$<>', '$<', '$<=', '$>', '$>=' are used. The reason for this is that '=' operates over Herbrand terms and the usual arithmetic operators are already used by Prolog.

3. Implementation of the Desai-Henry model

3.1 Overview of the model

The model of the UK economy chosen for the first part of this project is a linear model proposed by M. Desai and S. G. B. Henry [DH70]. The model was not intended to be comprehensive but rather designed for the purpose of simulating aspects of government policy. The constant parameters were estimated from quarterly data from the period 1955-66 using a development of the *Least Squares* technique.

The Desai-Henry model is defined by the following equations:

1. $C_t = 2472.43 + 0.2673Y^d_{t-1} + 22.52t$
2. $I_t = 102.80 + 0.3560(Y - Y^*)_{t-2} + 0.8851I_{t-1}$
3. $M_t = -230.32 + 0.064R_{t-1} + 0.4889M_{t-1} + 0.1545Y_{t-1}$
4. $U_t = 1994.74 + 17.726t - 0.3999Y_{t-1} + 0.4499U_{t-1}$
5. $T_{Yt} = -256.68 + 0.0836Y_t + 0.607T_{Yt-1}$
6. $T_{et} = 60.12 + 0.1296C_t + 0.44T_{et-1}$
7. $Y_t = C_t + I_t + G_t + E_t - M_t - T_{et} + S_t$
8. $Y^d_t = Y_t - T_{Yt}$
9. $R_t = R_{t-1} + E_t - M_t$

where

C = Consumer expenditure
Y^d = Personal disposable income
I = Investment
Y = Gross domestic product (GDP) at factor cost
Y^* = Moving average of previous four quarters of Y. Thus

$$Y^*_t = (Y_{t-1} + Y_{t-2} + Y_{t-3} + Y_{t-4}) / 4.$$

M = Imports of goods and services.
R = Reserves.
U = Unemployed (in thousands).
t = Time trend (in quarters with 1955Q1 = 1).
T_Y = Tax on the income.
T_e = Tax on expenditure.
G = Government expenditure on goods and services.
E = Exports of goods and services.
S = Government subsidies.

All units are in millions of pounds unless stated.

3.2 Implementation requirements

The two main requirements, regarding the implementation of the model were:

1. It should be able to run forward from a starting set of data and produce values for each variable at each time period.
2. It should be able to run backwards given certain data at a certain time period and produce results for each variable for each time period.

It was felt that the best way to tackle the design of the Desai-Henry implementation in order to achieve these aims was "bottom-up". That is, construct a design for each equation in isolation ensuring that data produced by each equation was available to other equations. The latter being necessary in order to allow the model to run forwards or backwards over time.

3.3 Running the model forwards

To illustrate the design process, a single equation is selected as an example. The design of the other equations following a similar pattern.

Recall that the equation for investment is:

$I_t = 102.80 + 0.3560(Y - Y^*)_{t-2} + 0.8851 I_{t-1}$

This equation can be described in a kind of pseudo logic as follows:

```
investment at time t is Inv if
      Find GDP at time t-2,
      Find GDP_STAR at time t-2,
      Find Inv at time t-1,
      Inv = 102.80 + (0.3560 * ((GDP - GDP_STAR) at
      time t-2)) + (0.8851 * (Inv at time t-1)).
```

where `Inv` = I = Investment
`GDP` = Y = Gross domestic product
`GDP_STAR` = Y^* =Average of previous four quarters of Y.

It will also be necessary to somehow store the value of `Inv` so it can be used by other equations (and actually by `Inv` itself at the next time period). The predicate *asserta/1* can be used which adds facts to the program database. The '*a*' at the end of *asserta* signifies that the new fact is to be added in front of the other clauses of the same predicate name. The Eclipse code is:

```
inv(AnyTime, Inv) :-
      TimeMinusTwo $= AnyTime - 2,
      TimeMinusOne $= AnyTime - 1,
      gdp(TimeMInusTwo, GDP_2),
      gdp_star(TimeMinusTwo, GDP_STAR),
      inv(TimeMinusOne, Inv_1),
      Inv $= 102.80 + (0.3560 * (GDP_2 - GDP_STAR +
            (0.8851 * Inv_1),
      asserta(inv(AnyTime, Inv)).
```

Data to start the model can be held as facts. The amount of data needed for each variable depends on the equation. The equation which requires the most data to be held is `GDP_STAR` which needs data on GDP for the previous six quarters. If these quarters are called time periods 1-6 then the next time period - the first one to be predicted - will be time period 7. Those equations that only require data to be held for the previous quarter will need only single entry at time period 6. For example

```
inv(6, 2767).
```

To run the model as a complete unit, a predicate *start/2* can be written that calls a predicate *economy/13* (which calls each model equation in turn) recursively.

```
start(Time,To):-
      Time $> To.

start(Time,To):- Time $<= To, nl,
      economy(Time, Con, Inv, Imp, Unem, TOI, TOE, GDP, PDI,
            Res, Exp, GovExp, Subs),
      writeln(con(Time, Con)),
      writeln(inv(Time, Inv)),
      writeln(imp(Time, Imp)),
      ........                /*rest of 'writeln' statements*/
      TimePlusOne $= Time + 1,
      start(TimePlusOne, To).
```

3.4 Running the model backwards

To run the model backwards in time a different clause is needed. This is due to the fact that if an equation predicate is called with an instantiated second argument then the equation contained in the predicate will contain no unknowns. Therefore when the model is run backwards it will only be able to return the result "*yes*" or "*no*" depending on whether the equations balance or not (which they almost certainly will not). However, what is required from the model is that it should be possible to assign values to variables at specific times in the future and see what effect that has on other variables at the same time period. The Eclipse code for investment to run the equation backwards is:

```
inv(AnyTime, Inv) :-
      AnyTime $>= 9,
      nonvar(Inv),
      Inv $= 102.8 + (0.3560 * (Gdp_2-Gdp_Star_2)) + (0.8851 *
            Inv_1),
      TimeMinusOne $= AnyTime - 1,
      TimeMinusTwo $= AnyTime - 2,
      inv(TimeMinusOne, Inv_1),
      gdp_star(TimeMinusTwo, Gdp_Star_2),
      asserta(gdp(TimeMinusTwo, Gdp_2)),
      asserta(inv(AnyTime, Inv)).
```

The line '`AnyTime $>= 9`' is required to ensure that the equations that run backwards are not called at an inappropriate time period. For example, investment depends on the value of GDP two quarters previous. As mentioned above, the most recent time period will be called time period 6. Therefore the value of GDP at time 6 will already be stored in the program as a fact. If *inv* is called earlier than time 9 with a given value then a new value for GDP at time 6 will be produced.

When running the model backwards the order that the predicates for the equations are called using *economy/13* is important. This is due to the fact that each equation depends on other variables (and predominantly those from previous time periods). What is required is that the value given be used as the starting point and that the value of subsequent variables are derived from it. Therefore, when running the model backwards, the predicate that calculates the given variable must be called first.

4. Implementation of the non-linear model

4.1 Overview of the model

The following model was constructed by the author. The model was produced using elements of the Desai-Henry model combined with new equations based on economic theory. The model is closed (i.e. there are no variables for which the model generates no values) and was based on data from the period 1981-1995. The time periods in this model are yearly, rather than quarterly as in the Desai-Henry model. All the constants for the equations were calculated on the statistical package Minitab using least squares regression.

The model contains six variables. Of these six, only one needs introducing (the other five being introduced in the Desai-Henry model), *P*, which represents yearly inflation. All the variables are endogenous. All units are in thousands of pounds per capita except for unemployment (millions of people) and inflation (percentage).

Below is a list of the equations defining the model:

1. $C_t = 477 + 0.903Y^d_{t-1} - 46U_{t-1}$
2. $P_t = -34.5 + 0.474P_{t-1} + 34.4(Y_t/Y_{t-1})$
3. $P_t = -2.26 + (18.2/U_t)$
4. $T_{Yt} = 268 + 0.0904Y_t + 0.349T_{Yt-1}$
5. $Y_t = 420 + 1.01Y_{t-1}$

4.2 Implementation of equation 3

The interesting equation is (3) - the second inflation equation:

$$P_t = -2.26 + (18.2/U_t)$$

Since the model contains another inflation equation (2) the value of *P* can be determined. This leaves *U* (unemployment) as the only unknown. Therefore this equation can be used to find the value of unemployment at time t. The Eclipse code is:

```
unem(AnyTime, Unem) :-
      inf(AnyTime, Inf),
      Inf $= -2.26 + (18.2/Unem),
      asserta(unem(AnyTime, Unem)).
```

Assuming appropriate predicates for the other equations in the model, data for time period 1 and a predicate start/2, that behaves the same as the one used for the Desai-Henry model, what happens when the following call is made:

```
start(2, 3)
```

The result is:

```
con(2, Con)
unem(2, Unem)
inf(2, 7.73278)
gdp(2, 4696.34)
pdi(2, 3709.58)
toi(2, 986.756)
```

```
con(3, Con)
      unem(3, Unem)
      inf(3, 6.98578)
      gdp(3, 5163.3)
      pdi(3, 4084.16)
      toi(3, 1079.14)

      Delayed goals:
            $$=(8.6, -2.26 + 18.2 / _424)
            $$=(7.73277664, -2.26 + 18.2 / _1575)
            $$=(6.98577881, -2.26 + 18.2 / _3858)

      Linear Store:
      Con_7320 $= 4523.26855 + -46 * Unem_1_7278
      Con_5047 $= 4164.99902 + -46 * Unem_1_5005
```

Eclipse could not solve the unemployment constraint because it was non-linear, even though the equation contained only one unknown variable, namely `Unem`. The above screen dump shows the unemployment equations at times 1, 2 and 3 in the 'Delayed goals' store. The 'Linear Store' holds the linear constraints that could not be solved because `Unem` was unknown.

4.3 Solving non-linear constraints

To overcome this restriction what is needed is:

1. A way of solving these constraints.
2. A way of detecting what constraints are unsolvable and passing them to the new solver.

As an example, consider the equation $y = x^2$. A well known way of solving this equation is the *Formula Iteration Method*. The Formula Iteration Method is a particular type of iterative process. The general features of an iterative process for finding the solution to a given equation are:

1. A starting value.
2. A rule for generating new values (hopefully better approximations to the solution).
3. A means of deciding when to stop the calculation.

When finding a solution of a given equation by Formula Iteration, the new value is calculated from the old by using some fixed computational rule obtained by rearranging the equation. For the equation $y = x^2$, the equation is rearranged in the form

$$x = {}^1/_2 (x + y/x)$$

and the function

$$x \rightarrow {}^1/_2 (x + y/x)$$

is implicitly used with starting value x_1 to obtain a sequence x_1, x_2, x_3, ... of better and better approximations to $\sqrt{y}$. Also, because the square root of most numbers is a long decimal (often infinite) it is necessary to decide on an appropriate termination point. This can be achieved by calculating the percentage error. The percentage error in the $y = x^2$ case can be arrived at by taking x^2 away from the current guess, dividing the result by the current estimate and

multiplying the final result by 100. The absolute value of the percentage is then taken. Therefore it is necessary to specify in advance what percentage error will be tolerated. The Eclipse code to solve $y = x^2$ is:

```
sqrt(0, X1, Perr, 0).
sqrt(Y, X1, Perr, Answer) :-
      Y $> 0,
      X2 $= (X1 + (Y/X1)) / 2,
      PE $= ((Y - (X2 * X2)) / Y) * 100,
      absolute(PE, PEabs),
      check(Perr, PEabs, Y, X2, Answer).
```

where *absolute/2* calculates the absolute value of the percentage error and *check/5* tests whether the current guess is acceptable (i.e. the current percentage error is less the required percentage error). If it is the result (Answer is returned). Otherwise, *sqrt/4* is called again.

Now that a predicate to solve a non-linear equation has been written, a way of accessing the delayed constraints is needed. This is provided for in Eclipse through the use of the *delayed_goals/1* call which binds a given variable to a list of all delayed goals. Each delayed goal in the list can then be parsed to see if it is of a recognised type (in the above case $y = x^2$) and if it is then the predicate to solve the equation (i.e. *sqrt/4*) called. In Eclipse:

```
nlsolver(Answers) :-
      delayed_goals(ListofGoals), solve(ListofGoals, Answers).

solve([], []).
solve([D1|Ds], [Result|Results]) :-
      parse(D1, Result),
      solve(Ds, Results).
```

Where *parse/2* succeeds if the delayed goal is of a recognised type (in this case the form $y=x^2$) and returns the result of the solved equation.

A whole library of predicates to solve non-linear equations could be built up. Assuming a predicate to solve non-linear equations of the form $y = a + b/x$, the *nlsolver/1* predicate can now be used in the non-linear model to solve equation 3. The code for the predicate *unem/2* now becomes:

```
unem(AnyTime, Unem) :-
      inf(AnyTime, Inf),
      Inf $= -2.26 + (18.2/Unem),
      nlsolver(List),
      head(List, Unem).
```

5. Conclusions and further work

Through the design and implementation of the Desai-Henry and non-linear models it has been shown that it is possible to use CLP to build economic models that can run both forwards and backwards in time. This allows the user to predict values in the future from starting data, and also to set values at certain future time periods and see what affect this has on the system as a whole. For example, the user may wish to see what the effect on gross domestic product will be if unemployment is 2m in two years time. This flexibility allows the user to test out many different scenarios.

The construction of the non-linear solver for the second model showed that, as long as the set of delayed constraints can be accessed, it is possible to write a predicate that can be used to produce answers to previously unsolved constraints. The non-linear solver can be thought of as a glass-box solver in the sense that the user sees the implementation of the predicate and can add further non-linear solvers.

There are many further areas of work that could be carried out in relation to this research. Some of the more interesting of these are:

- the implementation of a meta-interpreter that includes a call to a non-linear solver, such as *nlsolver*, when the set of goal calls is empty but delayed constraints still exist;
- the extension of the library of non-linear solvers;
- the implementation in Eclipse of a larger non-linear model, for example, the treasury model [HM93] of the UK economy;
- the addition of a graphical user interface to the programs;
- the placing of the economic programs onto the world wide web.

References

[CSO] The Central Statistical Office, UK.

[CT78] The Course Team. Approximations - The Open University Mathematics Foundation Course, The Open University Press, 1978.

[D80] E. T. Dowling. Mathematics for Economists, McGraw-Hill, 1980.

[DH70] M. Desai, S. G. B. Henry. Fiscal Policy Simulation for the UK Economy,.1955-66, 1970, contained in [HH70].

[ECRC92] ECRC. Eclipse 3.4 User Manual, ECRC, 1992.

[ET] Economic Trends, CSO, 1958 - 1994.

[FH93] Fruhwirth, A. Herold, V. Kuchenhoff, T. Le Provost, P. Lim, E. Monfroy, M. Wallace. Constraint Logic Programming - An Informal Introduction, technical report ECRC-93-5.

[HM93] H.M. Treasury Macroeconomic Model Documentation, UK, 1993.

[J82] D. Jackson. Introduction to Economics - Theory and Data, MacMillan, 1982.

[K92] P. Kennedy. A guide to Econometrics, Basil Blackwell Ltd, UK, 1992.

[MN73] P. Mottershead and J. Naughton. Modelling Economic systems, Module eight of Systems Behaviour Course, The Open University Press, UK, 1973.

[S84] J. Stewart. Understanding Econometrics, Unwin Hyman Ltd, UK, 1984.

[SN89] P. A. Samuelson, W. D. Nordhaus. Economics, McGraw-Hill, 1989.

[SS86] L. Stirling, E. Shapiro. The Art of Prolog, MIT Press, 1986.