README

The purpose of this file is to:

1. describe the elements used to optimize the objective function in order to find parameter estimates and run policy simulations (see “Folder Summary” below);
2. explain how to optimize the objective function and run policy simulations (see “Instructions for Execution” below);
3. direct the reader to additional documents regarding data and estimation (see “Additional Documentation” below).

All of the code is written in ANSI C++. It is a moderate sized class hierarchy and uses extensively class templates. Contained with the code is a project file for Microsoft Visual Studio Express 2005 (available for free download from Microsoft’s website). Makefiles for compiling on Unix systems are not provided.

To open the Visual Studio project, simply double click on the general_equilibrium\michigan\michigan.sln file. And then you can build and run the project from the Visual Studio menus. More details are provided below on how to run policy simulations.

The project code is divided into four folders, which will be described below.

1. Folder summary

common

This folder contains utility classes and functions. The Matrix class, which is a four dimensional matrix representation, is used throughout the code base. The code in the common folder is compiled into a library called common.lib

models

This folder contains base classes that are used by the classes in the **michigan** folder. These base classes are from the Ferreyra 2007 paper, and they serve as the foundation of this new paper. The code in this folder is compiled into a library called michigan.lib

Below is a brief description of the base classes used in this paper:

CNewModel – computes and stores the benchmark equilibrium

CLossFunction – inputs data, computes the benchmark or policy equilibrium, and outputs the distance between the predicted and the observed equilibria

optimization

This folder contains optimization routines for the maximum likelihood problem. Namely, there is a Cyclical Coordinate Descent implementation and line search routines. Non-gradient optimization methods are used with our objective functions because they are not differentiable.

The routines in the optimization folder are compiled into the library optimization.lib

michigan

This folder contains the core code developed for this paper. Its classes extend those found in the **models** folder. The main classes are detailed below:

CMIchiganBaseModel extends CNewModel

This class contains extensions to CNewModel (defined in the **models** folder) that are common to both the DPE and ProposalA models of the project. It is therefore the base class for the following two classes.

CDPEModel extends CMichiganBaseModel

This class defines the first phase of the Michigan model experiment.

CProposalAModel extends CMichiganBaseModel

This class defines the second phase of the Michigan model experiment.

CMichiganModel

This class is a templated class whose first template parameter is a class derived from CDPEModel and whose second template parameter is a class derived from CProposalAModel. An instance of each these classes is created. The Michigan model first executes the DPE instance as phase I, saves the state of the model, and then executes the instance of the ProposalA class as phase II. Thus, the execution of the MichiganModel is a hybrid execution.

Policy experiments are run by instantiating a CMichiganModel object with a different ProposalA derived second template parameter, such as the following:

CProposalAModelAdequacy
CProposalAModelEqualization
CProposalAModelOpenEnrollment
CProposalAModelTaxReform
CProposalAModelUnrestricted

Each of these will create a different policy experiment.

data

This folder contains all data required by the program when executing. By default, the michigan\src\main.cpp will read data from this directory. There are two subfolders: one

that contains data for running the DPE phase I (the subfolder is DPEData), and the second contains data for running the ProposalA phase II (the subfolder is ProposalA). UP TO HERE.

2. Instructions for Execution

The code may be executed for two main purposes: (1) estimating the model, or (2) computing the benchmark equilibrium or running a policy experiment.

Estimating the model means to define an objective function and then optimize it. The available objective functions are in the optimize_lib folder described above. A policy experiment is simply an evaluation of one of the functions from the voucher folder described above.

The file voucher\src\main.cpp contains code that shows how to (1) construct an objective function and then optimize it, and (2) define a policy experiment or benchmark equilibrium function and evaluate it.

Both the estimation (see "optimize" below) and the equilibrium computation (see "evaluate" below) require a parameter point ("parameterPoint"). This is the starting point for the parameter search in the case of the estimation, and the desired parameter point (usually the vector of estimates) for the benchmark equilibrium and policy computations. The parameter point is specified via the command line in the following order: alpha, beta, rho, r, delta, and b. An optional seventh command line parameter can be used to label the parameter point – and it is used when creating the log file filenames.

The objective function has multiple local minima, therefore different starting points yield different optimal points in the optimization. Because the estimation procedure involves two stages, the optimization needs to be run twice – the first time for untransformed variables; the second time for transformed variables (see more below regarding the variable transformation and the specification of the estimation stage).

All output files are saved to the current working directory.

To find parameter estimates, uncomment the following lines:

```
CVoucherFunctionT<CNewModelRel> voucherFunction(functionLog);
voucherFunction.setDataDirectory("../data/");
voucherFunction.addAllCities();
voucherFunction.setFirstStage(true);

optimize(voucherFunction, strJobNumber, lower, upper, parameterPoint);
```

The first line creates a log file in the current working directory; setDataDirectory() specifies the data directory which must contain all of the input files (provided with the code); addAllCities() specifies to use all metropolitan areas; setFirstStage(true) specifies the first of the two stages of the estimation procedure. If running the second stage, which

requires transforming the variables using the first stage residuals (see “Additional Documentation” below), then specify setFirstStage(false).

To compute the benchmark equilibrium, use the following code:

```
// non-voucher - computes the benchmark equilibrium
CVoucherFunctionT<CNewModelRel> voucherFunction(functionLog);
voucherFunction.setDataDirectory("../data/");
voucherFunction.addCity(“chicago”);
voucherFunction.setFirstStage(true);

evaluate(voucherFunction, parameterPoint);
```

This defines the benchmark equilibrium function for Chicago, and then evaluates the function. To compute the benchmark equilibrium for all cities, then replace addCity(“chicago”) with addAllCities().To specify any of the other metropolitan areas, change “chicago” to either of the following: boston, detroit, newyork, philad, pittsburgh, stlouis.

The files with the equilibrium outcomes are saved to the current working directory.

To run a policy experiment, uncomment one of the voucherFunction declarations (except the benchmark equilibrium one) and the “evaluate” function. For example,

```
// Universal Vouchers - computes the benchmark equilibrium first,
// and then the voucher equilibrium
CVoucherFunctionT<CUniversalVoucher<CNewModelRel> >
voucherFunction(functionLog);
voucherFunction.setDataDirectory("../data/");
voucherFunction.setVoucherAmount(0.10);
voucherFunction.addCity("chicago");
voucherFunction.setFirstStage(false);

evaluate(voucherFunction, parameterPoint);
```

In this code, the voucher amount is specified via setVoucherAmount() (dollar amounts are expressed in $10,000); setFirstStage() could be set either to true or false because this is not part of the estimation procedure.

The files with the simulation outcomes are saved to the current working directory.

3. Additional Documentation

The document DataFilesDescription.doc explains the content of each data file that is read by the code in order to compute the equilibrium. In addition, the document Data-Estimation.doc has the following purposes:

1. document the sources of data which give rise to the input data files documented in DataFilesDescription.doc

2. document the calculation of rental values and the data sources used to calculate them
3. explain the estimation of standard errors