Nested Fixed Point Algorithm Documentation Manual

Version 6: October 2000

Professor John Rust
Department of Economics
Yale University
New Haven, CT 06520
(203) 432-3569
jrust@gemini.econ.yale.edu
http://gemini.econ.yale.edu

NFXP Documentation Manual

Table of Contents

1. Overview of the NFXP Algorithm

1.1 Background

The nested fixed point algorithm is a maximum likelihood estimation algorithm that computes maximum likelihood estimates of structural parameters of a wide class of discrete control processes; i.e. stochastic processes that are solutions to infinite horizon Markovian decision problems with discrete choice or control variables. In general these problems do not possess analytic or closed-form solutions, but must be solved numerically. Under certain assumptions one can show that the numerical solution to such problems is equivalent to the computation of a fixed point to a differentiable contraction mapping. The simple idea behind the nested fixed point algorithm is to solve the stochastic control problem numerically by computing the associated functional fixed point as a subroutine nested within a standard nonlinear maximum likelihood optimization algorithm; hence the name, "nested fixed point algorithm". A paper entitled "Maximum Likelihood Estimation of Discrete Control Processes" (Rust 1988) introduced a class of discrete control processes and the nested fixed point algorithm, but concentrated on developing the asymptotic estimation theory for such processes instead of focusing on the computational details of the algorithm itself. In order to provide a concrete application of the theory presented in this paper, a subsequent paper was written to illustrate the use of the nested fixed point algorithm to estimate the structural parameters of a simple model of optimal replacement of bus engines at the Madison Metropolitan Bus Company. These results are presented in a paper entitled "Optimal Replacement of GMC Bus Engines: An Empirical Model of Harold Zurcher" (Rust, 1987), winner of the 1992 Ragnar Frisch Medal for "best empirical paper in *Econometrica* in the preceding 5 years".

This distribution contains the data and nested fixed point algorithm used to estimate the model of optimal bus engine replacement presented in the 1987 *Econometrica* paper. I hope that by providing a concrete, detailed application of the general algorithm, users will be able to see how to modify this code (or construct new code) in order to apply the algorithm to other specific problems.

1.2 What do I need to run the algorithm?

The programs and data are coded in the GAUSS matrix programming langauge. In order to run the programs you will need:

1. GAUSS programming language. Available from Aptech Systems, Inc. 23804 SE Kent-Kangley Road, Maple Valley, Washington 98038, `www.aptech.com` `sales@Aptech.com` (206) 432-7855, (206) 432-7832 (fax).

2. an Intel or Sparc workstation running Windows/Dos or a supported version of Unix (e.g. Linux or Sun Solaris).

The decision to code the algorithm in the GAUSS language instead of a standard programming language such as FORTRAN was made because:

1. the GAUSS langauge is a high-level symbolic language similar to APL which enables a nearly 1:1 translation of mathematical formulae into computer code. Matrix operations of GAUSS replace cumbersome do-loops of FORTRAN.

2. GAUSS has built-in linear algebra and matrix inversion routines, saving the need to include external linear algebra libraries such as LAPACK.

3. efficient use of GAUSS requires the user to "vectorize" a program in a manner very similar to efficient vectorization on large-scale vector supercomputers.

The last point is particularly important. Analysis of the nested fixed point algorithm shows that 99% of the work occurs in a few basic operations: linear equation solution, matrix multiply, vector product, exponential and logarithm evaluation, and coordinate assignment. These basic operations of GAUSS are precisely the operations that supercomputers are highly optimized for. For each GAUSS command there is a corresponding high efficiency subroutine coded on the CRAY supercomputer. The conceptualization process to get programs to vectorize in GAUSS is very close to that needed to get FORTRAN programs to run efficiently on a vector supercomputers. My hope is that the GAUSS language is sufficiently easy to learn that even inexperienced users will find it relatively easy to see how the general equations of the model are transformed into concrete computer code, and possibly to go on and write their own code for the specific problems they encounter.

1.3 What's Included in the Distribution?

The NFXP distribution is available via anonymous ftp at `gemini.econ.yale.edu` in the `pub/johnrust/nfxp` subdirectory or at `http://gemini.econ.yale.edu/jrust/nfxp`. The distribution contains the following types of files:

1. **Data Files.** These files contain monthly data on bus maintenance histories for a sample of 162 buses in the fleet of the Madison (Wisconsin) Metropolitan Bus Company followed over the period December, 1974 to May, 1985. The "raw data" is contained in ascii files with ".asc" file extensions. Each file, such as `a530875.asc`, contains monthly data on the odometer readings of all buses of a given make, model, and vintage (in this case, 1975 GMC model 5308 buses). The files also contain headers with information on the date and mileage of engine replacement/overhauls for each bus. The raw data in the data files must be discretized to run the nested fixed point algorithm. A pre-processor program, `STORDAT.GPR`, discretizes the data for the NFXP algorithm. This recoded and discretized data resides in the GAUSS data file `bdt.dat` and its associated header file `bdt.dht` (Unix versions of Gauss include the header information in the main data file, `bdt.dat`). These latter files are the actual input files used by the nested fixed point program, `NFXP.GPR`.

2. **Outer BHHH Optimization Algorithm.** The nested fixed point algorithm resides in the master file `NFXP.GPR`, calling procedures `FFXP`, `DISPLAY`, `PARTSOLV`, `E`, `CONTRACT`, `FUNCTION`, `CDTP`, `EVAL`, and `EVAL1`. The code for the latter routines are contained in files with a `G` extension. The file `NFXP.GPR` contains a standard nonlinear maximum likelihood optimization algorithm known as BHHH (named after an article by Berndt, Hall, Hall, and Hausman, 1976, "Estimation and Inference in Nonlinear Structural Models") accelerated by the BFGS (Broyden, Fletcher, Goldfarb and Shannon) method. The BHHH algorithm is a Newton-like gradient algorithm that uses the outer product of the gradients of the loglikelihood function as an approximation to the hessian to maximize a likelihood function $L(\theta)$ or partial likelihood function $L_p(\theta)$ over the model's unknown parameters $\theta \in R^k$. The BFGS method is a recursive updating procedure that generally provides more accurate estimates of the inverse of the hessian of the likelihood function than the BHHH algorithm provides, and thereby accelerates the convergence of the optimization algorithm. However the BFGS updating procedure can sometimes yield vastly poorer estimates of the inverse hessian when the algorithm is relatively far from converging, and can lead to explosive oscillations in the sequence of trial parameter values and cause the algorithm to fail to converge. Thus, the

best compromise is to begin with BHHH, which is fairly robust and can be proven to always lead to a direction of improvement in the likelihood, but to switch to BFGS when the BHHH algorithm starts to converge, since at that point the BHHH starts to exhibit roughly linear rates of convergence whereas the BFGS algorithm becomes increasingly reliable as the algorithm enters a domain of attraction of a local optimum parameter vector and exhibits approximately quadratic rates of convergence. Thus, the combination of the BHHH and the BFGS algorithm results in a useful compromise that has the reliability of the BHHH algorithm, but can also take advantage of the faster asymptotic rates of convergence of the BFGS method.

The difference between `NFXP.GPR` and standard implementations of the BHHH/BFGS algorithm is that in this case the likelihood $L(\theta)$ does not have an *a priori* known functional form. Instead, $L(\theta)$ depends on an unknown function EV_θ which is a fixed point to EV_θ to a differentiable contraction mapping T_θ. The NFXP software computes the value of EV_θ and its derivatives numerically in a subroutine `FFXP.G` of `NFXP.GPR`. This implies that we can numerically compute $L(\theta)$ and its derivatives for each trial value θ encountered in the course of the process of maximizing $L(\theta)$. In order to do this, we need a very efficient and accurate algorithm for computing the fixed point $EV_\theta = T_\theta(EV_\theta)$.

3. **Inner Fixed Point Algorithm.** The procedure `FFXP.G` contains the code to implement the contraction mapping fixed point (poly)algorithm. The algorithm combines contraction iterations with Newton-Kantorovich iterations to efficiently compute the functional fixed point $EV_\theta = T_\theta(EV_\theta)$, where $EV_\theta(x, d)$ is the expectation of the discounted value of the agent's future utility stream, conditional on being in observed state x and taking discrete action d in a finite choice set $D(x)$. The vector θ denotes the unknown parameters of the utility function $u(x, d, \theta)$ specifying theu agent's utility of taking action d in state x. Also included in θ are the agent's discount factor $\beta \in [0, 1)$ and parameters specifying the Markov transition density for the state variable x: $p(x_{t+1}|x_t, d_t, \theta)$. If the underlying state variable x is continuous, it must be discretized in order to allow computation of EV_θ on a digital computer. In the case of the bus problem, the state variable x is the accumulated mileage on the bus which we discretize into a finite number of *mileage ranges.* The discretization implies that EV_θ is a finite dimensional vector. In the bus example this vector is interpreted as the conditional expectation of the discounted value of future operating and maintenance costs for the bus. The file `FFXP.PRC` contains a more elaborate version of the fixed point polyalgorithm that also computes the derivatives $\partial EV_\theta/\partial\theta$ which are used to construct derivatives of the likelihood function $L(\theta)$ for `NFXP.GPR`.

4. **Data Creation Files.** The file STORDAT.GPR allows you to select a subsample of buses from the raw data files in the DAT subdirectory for estimation by NFXP.GPR. The program automatically discretizes the data and codes it in a form conformable to NFXP.GPR. The output of this program is the GAUSS data file BDT.DAT and its header file BDT.DHT (in the Unix version of Gauss the header BDT.DHT is combined into the data file BDT.DAT). The file SIMDAT.GPR allows you to create BDT.DAT filled with simulated data generated from a structural model of your choice. If the structural model is correctly specified, the simulated data should be statistically indistinguishable from the actual bus data.

5. **Derivative Check Routines.** The NFXP.GPR program contains a built-in option to conduct a numerical checks of the analytic derivatives used in various optimization and fixed point programs. This provides an indirect check on the correctness of the analytic formulas for the derivatives of the operating cost function $c(x, \theta)$ that the user supplies to NFXP.GPR in the procedure FUNCTION.G. The program DEVCHK.GPR checks the derivatives of the expected value function EV_θ with respect to the parameter vector θ, $\partial EV_\theta / \partial \theta$.

6. **Hypothesis Testing Routines.** The file PROB.GPR estimates the transition probability distributions for mileage for each bus group, and allows you to test the hypothesis of parameter homogeneity between and within each bus group. The file LRTEST.GPR allows you to conduct likelihood ratio and chi-square goodness of fit tests of the validity of the structural model.

7. **Non-parametric Estimation Routines.** The file NPMLE.GPR allows you to compute the non-parametric maximum likelihood estimate of the hazard function $h(x) = P(1|x)$, which equals the probability of replacing a bus as a function of its accumulated mileage x. The non-parametric hazard is also automatically computed and graphed when you run STORDAT.GPR.

8. **Simulation Programs.** The file BANACH.GPR computes the cylinder distributions of the estimated value and hazard functions by Monte Carlo simulation (for details, see Rust 1988). The file SIMDAT.GPR allows the user to create a simulated GAUSS data file BDT.DAT (and its header BDT.DHT) filled with artificial data generated by a structural model of your choice. You can use this artificial data set to run the program NFXP.GPR to "back out" estimates of your a priori choice of structural parameters. If the program is working correctly, then in large samples the maximum likelihood estimates computed by NFXP.GPR should be close to the parameters you selected, and the covariance matrix of sampling errors between the estimated and actual structural parameters should be close to the asymptotic covariance

matrix produced by NFXP.GPR. By attempting to estimate incorrect structural specifications, you can also judge the sensitivity of the maximum likelihood estimates to specification errors.

9. **Data Plotting Programs.** The file PLOT.GPR automatically plots the hazard function $P(1|x,\hat{\theta})$, the value function $V_{\hat{\theta}}(x) = c(x,\hat{\theta}) - \beta EV_{\hat{\theta}}(x)$, the cost function $c(x,\hat{\theta})$, and the expected value function $-\beta EV_{\hat{\theta}}(x)$ computed by FFXP.G. NFXP.GPR also produces high resolution graphics plots of the same functions. The program is particularly useful for interpreting the parameter estimates produced by NFXP. In some cases one may not have good intuition about the values of the estimated coefficients, but one might have quite a lot of intuition about the shapes of the estimated value and hazard functions. By plotting out the estimated cost and hazard functions, one can get a better handle on whether or not the estimated model is reasonable. One can also use comparisons of the parametric and non-parametric hazard functions printed at each iteration for a visual representation of how the NFXP algorithm attempts to find a value of $\hat{\theta}$ to "best fit" the data.

10. **Utility procedures.** To make the NFXP algorithm more comprehensible, I have attempted to modularize the the basic operations of the NFXP algorithm into calls to a series of subtasks coded as Gauss procedures. The files DISPLAY.G, FFXP.G, EQ11.G, EQ22.G, FUNCTION.G, EVAL.G and EVAL1.G are called by the main program NFXP.GPR. DISPLAY shows the current model and parameter settings, the EQ11 and EQ22 procedures update the Markov transition matrix for the state variable x, FUNCTION updates the cost function and its derivatives, and EVAL and EVAL1 evaluate the loglikelihood function and its derivatives. The file FFXP.G contains the code that computes the fixed point EV_θ and its partial derivatives $\partial EV_\theta/\partial\theta$. This file in turn calls the procedures CONTRACT.G, E.G, PARTSOLV.G, and CDTP.G. The CONTRACT.G procedure contains the code to compute a single evaluation of the contraction operator T_θ, and calls in turn the procedure E.PRC to compute the conditional expectation of the value function involved in evaluating T_θ. PARTSOLV.PRC contains the code to solve the linear system $[I - T'_\theta]V = W$, where T'_θ is the Fréchet derivative of T_θ. This code is used to construct the more efficient Newton-Kantorovich iteration for the fixed point EV_θ as well as to compute the derivatives $\partial EV_\theta/\partial\theta$ using the implicit function theorem. CDTP.G contains code to compute the analytic derivatives of T_θ with respect to components of θ representing the state transition probabilities. All of these routines will be described in detail in chapters 2 and 3.

11. **Miscellaneous Statistical Routines.** These programs include files such as OVHTIM.GPR that computes basic sample statistics on the time and mileage ofoverhaul for each bus

group, and PROB.GPR which computes the discrete probability distributions for monthly mileage for each bus group. The files EQUIL.GPR and EQDIST.GPR compute the equilibrium distributions of the controlled Markov process $P(d_{t+1}|x_{t+1},\theta)p(x_{t+1}|x_t,d_t,\theta)$ for any value of the parameter vector θ. The file DEMAND.GPR traces out how the equilibrium distribution shifts with changes in the replacment cost parameter, tr allowing one to compute an "expected demand curve" for replacement investment.

Appendix I provides a listing and description of the 52 files provided in the NFXP distribution. Note that due to changes in the Gauss language from Version 1.49 to 2.0 and higher, not all the files in 6-10 above are guaranteed to run without modification. The only files that have been tested and are "guaranteed" to run are the files in items 1-5 above.

1.4 How do I get started?

Step 0: Install the NFXP distribution. On DOS/Windows systems retrieve the binary file nfxp.zip via anonymous ftp to gemini.econ.yale.edu in the pub/johnrust/nfxp directory. Use the unzip command to install the source code, documentation and data files on your system: unzip nfxp.zip. On a Unix system use the zcat command to install the source code, documentation and data files on your system: zcat nfxp.tar.Z |tar xf -. This will create a directory tree nfxp under the directory from which you ran the unzip or zcat commands. If you don't have zcat or zip make an nfxp directory on your system and copy the directories src, dat, and doc in the nfxp directory of gemini.econ.yale.edu using anonymous ftp. When the files are installed change into the src subdirectory of the nfxp directory tree.

Step 1: Start GAUSS, version 2.0 or later.

Step 2: Run nfxp.gpr. This program will automatically detect whether proper initializations have been done to create an estimation data set and to select functional forms to be estimated, and settings for the NFXP algorithm. It will automatically spawn calls to setup.gpr and stordat.gpr to create the data sets and to interactively prompt the user for the desired model specification and NFXP algorithm settings.

If you are confused about answering some of the questions the programs ask of you, Appendix 4 presents a sample run of a GAUSS session that estimates a model using these programs. At first you may want to simply copy the responses given in the example in Appendix 4 just to make sure

the programs run correctly on your computer. Then as you learn more about the program and its various internal tolerance settings you can input your own answers and attempt to estimate other models from the model menus presented in the program SETUP.GPR, or add your own choices of functional forms to the menus in FUNCTION.G and DISPLAY.G.

NFXP.GPR computes problems with arbitrary dimensional fixed points, a parameter specified by the user. For the bus problem, fixed points of dimension 90 or larger provide more than adequate resolution to guarantee that results are not sensitive to approximation (discretization) error.

Important: If you decide to change the dimension n of the fixed point from 90 to some lower dimension, make sure that you re-run the program STORDAT.GPR to recreate the data files BDT.DAT and BDT.DHT with entries which are conformable to the new value of n. Running the program NFXP.GPR with a dimension n larger than the dimension n of the discretized data in the file BDT.DAT will cause the program to crash with an "INDEX OUT OF RANGE" error message. However, running NFXP.GPR with a dimension n smaller than the dimension n of the discretized data in the file BDT.DAT will not produce an error message. Nevertheless, the results of running NFXP.GPR without a conformable data file BDT.DAT will be incorrect. NFXP.GPR attempts to determine whether such changes have occurred, and automatically reruns STORDAT.GPR to prevent inconsistencies.

1.5 Guide to the rest of this manual

Chapter 2 derives the functional fixed point equations that characterize the optimal dynamic behavior of rational agents and translates these equation into GAUSS code.

Chapter 3 describes the general structure of the nested fixed point algorithm and translates the basic iterative equations of the algorithm into GAUSS code.

Chapter 4 describes the format of the raw data stored in the dat subdirectory of the main nfxp directory. The chapter also describes how the program STORDAT.GPR selects subsamples in order to produce the discretized BDT.DAT file used by NFXP.GPR.

Appendix 1 provides a description of each of the 52 files in the NFXP distribution.

Appendix 2 defines all the variable names used in NFXP.GPR and its subfiles.

Appendix 3 provides sample sessions of some of the GAUSS programs contained in the NFXP distribution.

2. Mathematical Derivation of the Fixed Point Equations

2.1 Overview

The nested fixed point program `NFXP.GPR` applies the general nested fixed point algorithm described in Rust (1988) to the problem of optimal replacement of bus engines. In order to best understand the origin of the code in `NFXP.GPR`, I describe the derivation of the functional fixed point equation solved by the nested fixed point algorithm in sections 2.2 to 2.4. In section 2.2 I present the fixed point equation for the general case. In section 2.3 I show how this general equation can be specialized to the particular problem of optimal replacement of bus engines. Finally section 2.4 shows how to translate each equation of the bus engine replacement problem into GAUSS code. Although this section summarizes the equations the algorithm must solve, it does not explain how to solve them. Chapter 3 explains the nested fixed point algorithm that solves these equations, and describes how to translate the solution algorithm into GAUSS code.

2.2 The general case

An econometrician observes time series data on the sequences of discrete choices d_t and observed states x_t of one or more decision-makers. The data are denoted by the vector $(d_1, \ldots, d_T, x_1, \ldots, x_T)$. In the case where the data consists of separate time series realizations for multiple decision-makers we superscript each data vector by the variable l, $l = 1, \ldots, L$ where L is the number of individuals in the sample. In order to keep the notation simple, I will derive the sample likelihood function for the case of a single agent. In the case of panel data the full sample likelihood function will simply be a product of each of the single agent likelihood functions.

The basic hypothesis is that the data series observed for each agent is a realization of a controlled stochastic process. This means that at each time t the agent selects a decision d from a choice set $D(x)$ to maximize expected discounted utility. In the case where the $D(x)$ is a finite set, we say that the controlled stochastic process $\{d_t, x_t\}$ is a *discrete control process*, and if $D(x)$ is a convex subset of a Euclidean space, we say that $\{d_t, x_t\}$ is a *continuous control process*. The nested fixed point algorithm is applicable only to the case of discrete control processes; a subclass of continuous control processes is handled in a paper by Hansen and Singleton, (1982).

Under very general conditions, the solution to a stochastic control problem takes the form of a deterministic decision rule $d = \delta_t(x)$, which specifies the agent's optimal action d at time t as a deterministic function of the agent's state x_t. Unfortunately, this produces a statistically degenerate

behavioral model; it implies that if one knew the functional form for δ_t, one could perfectly predict the behavior of the agent. Of course, no structural model will ever perfectly predict the choices of an agent, so one needs to introduce an "error term" ϵ_t to account for the discrepancies between the predictions of the model and the observed choices of the agent. One reason why econometric models fail to perfectly predict agents' choices is that the set of state variables observed by the econometrician, x_t, is generally only a subset of the set of state variables actually observed by the agent. Let ϵ_t denote a vector of state variables which are observed by the agent, but which are not observed by the econometrician. In this case, the true model is of the form $d_t = \delta_t(x_t, \epsilon_t)$, which is no longer statistically degenerate. I will now derive such a model, starting from basic assumptions about the agent's preferences and the stochastic law of motion for the state variables (x_t, ϵ_t).

Suppose that the vector of state variables obey a Markov process with transition density given by a parametric function $\pi(x_{t+1}, \epsilon_{t+1}|x_t, \epsilon_t, d_t, \theta)$ that depends on a vector of unknown parameters θ and the action $d \in D(x_t)$ selected by the agent. Assume that $\epsilon_t \equiv \{\epsilon_t(d)|d \in D(x_t)\}$, i.e., that ϵ_t is a vector with dimension equal to the number of alternatives in the choice set $D(x_t)$. Assume further that the function $u(x_t, d_t, \theta) + \epsilon_t(d)$ gives the realized single period reward or utility value when alternative d is selected and the state variable is x_t. The utility function u also depends on the unknown parameter vector θ. The behavioral hypothesis is that the agent chooses a decision rule $d_t = \delta(x_t, \epsilon_t, \theta)$ to maximize his expected discounted utility over an infinite horizon where the discount factor $\beta \in [0, 1)$. The solution to this intertemporal optimization problem is given recursively by *Bellman's Equation:*

$$V_\theta(x, \epsilon) = \max_{d \in D(x)} \left[u(x, d, \theta) + \epsilon(d) + \beta \int V_\theta(x', \epsilon,) \pi(dx', d\epsilon'|x, \epsilon, \theta) \right] \tag{2.1}$$

The function $V_\theta(x, \epsilon)$ is the maximum expected discounted utility obtainable by the agent when the state variable is (x, ϵ). It will be convenient to define the function EV_θ, the expected value function, by

$$EV_\theta(x, \epsilon, d) = \int V_\theta(x', \epsilon') \pi(dx', d\epsilon'|x, \epsilon, d, \theta). \tag{2.2}$$

This allows us to rewrite Bellman's equation in slightly simpler notation as:

$$V_\theta(x, \epsilon) = \max_{d \in D(x)} \left[u(x, d, \theta) + \epsilon(d) + \beta EV_\theta(x, \epsilon, d) \right]. \tag{2.3}$$

To summarize: I assume the data $\{x_1, \ldots, x_T, d_1, \ldots, d_T\}$ is a realization of a controlled stochastic process whose solution is an optimal decision rule $d_t = \delta(x_t, \epsilon_t)$ where the function δ is defined for each (x, ϵ) pair as the value of $d \in D(x)$ that attains the maximum in Bellman's equation (2.1).

The objective is to use the data to infer the unknown parameter vector θ, where for notational simplicity we subsume the discount factor β as one of the components of θ.

The procedure for inferring the unknown parameters is the method of *maximum likelihood.* The method requires us to derive the probability density function $L(x_1, \ldots, x_T, d_1, \ldots, d_T|\theta)$ for the data and compute the parameter estimate $\hat{\theta}$ that maximizes the likelihood function over all possible values of θ. Unfortunately, we cannot compute the probability density for the data until we solve the dynamic programming problem (2.1). The following *Conditional Independence Assumption* yields a simple formula for the likelihood function and substantially simplifies the problem of computing the solution to (2.1).

$$(CI) \qquad \pi(x_{t+1}, \epsilon_{t+1}|x_t, \epsilon_t, d_t, \theta) = p(x_{t+1}|x_t, d_t, \theta)q(\epsilon_t|x_t, \theta). \tag{2.4}$$

If the density q is further assumed to be a multivariate extreme value distribution, then the likelihood function for the sample of data can be shown to be given by the function

$$L(\theta) \equiv L(x_1, \ldots, x_T, d_1, \ldots, d_T|\theta) = \prod_{t=2}^{T} P(d_t|x_t, \theta)p(x_t|x_{t-1}, d_{t-1}, \theta). \tag{2.5}$$

where the *conditional choice probability,* $P(d|x, \theta)$, is given by the classic *multinomial logit* formula (see McFadden, 1973):

$$P(d|x, \theta) = \frac{exp\{u(x, d, \theta) + \beta EV_\theta(x, d)\}}{\sum_{d' \in D(x)} exp\{u(x, d', \theta) + \beta EV_\theta(x, d')\}} \tag{2.6}$$

and the expected value function EV_θ is given by the unique fixed point to the contraction mapping $T_\theta(EV_\theta) = EV_\theta$ defined by

$$EV_\theta(x, d) = T_\theta\left(EV_\theta\right)(x, d) \equiv \int log\left[\sum_{d' \in D(x')} exp\{u(x', d', \theta) + \beta EV_\theta(x', d')\}\right] p(dx'_x, d, \theta). \tag{2.7}$$

One can show (see Rust, 1988, 1994) that T_θ is a *contraction mapping* which implies that EV_θ is the unique solution to (2.7). Furthermore consistent estimates of θ can be obtained by maximizing the following "partial likelihood"function

$$L_p(\theta) \equiv \prod_{t=1}^{T} P(d_t|x_t, \theta). \tag{2.8}$$

To conclude, the general nested fixed point optimization problem is to find a value of θ that maximizes either the full likelihood function $L(\theta)$ given in (2.5) or the partial likelihood function $L_p(\theta)$ given in (2.8), subject to the constraint that the function EV_θ is given by the unique fixed point to the functional equation (2.7).

2.3 Applying the general approach to the bus engine replacement problem

The bus engine replacement problem is to determine how long one should continue to operate and maintain a bus before it is optimal to remove the engine and replace it with either a new or rebuilt engine. The idea is that since a bus engine is a portfolio of individual components, a failure of any single component could put the bus out of service. A naive strategy would be to simply replace each component at time of failure, and leave the other components in the engine alone. However, it is reasonable to expect that once one component in the engine has failed, the other components are more likely to fail in the near future. To the extent that the bus company imputes a high cost to bus failures on the road (due to towing costs, lost bus driver time, and loss of customer goodwill), and to the extent that the fixed costs of opening up a bus engine for repair are high, then a policy of periodic bus engine replacement would seem to make sense as a cost effective preventive maintenance strategy. Under the maintained hypothesis that such a preventive replacement strategy is optimal, the problem is to construct a model that predicts the time and mileage at which engine replacement occurs.

To fit the bus replacement problem into the general framework presented above, we must:

1. specify who the decision-maker or agent is,
2. specify the state variables (x, ϵ)
3. specify the state-dependent choice sets, $D(x)$
4. specify functional forms for the utility function $u(x, d, \theta)$ and $p(x'|x, d, \theta)$.

Our data set consists of monthly observations of 162 buses in the fleet of the Madison Metropolitan Bus Company followed from December, 1974 to May, 1985. The data for each bus consists of a vector $\{x_1, \ldots, x_T, d_1, \ldots, d_T\}$ where $d_t = 1$ if the bus engine was replaced in month t, $d_t = 0$ otherwise, and x_t equals the accumulated mileage on the bus (odometer reading). I will now show how the general framework of the preceding section can be specialized to handle the bus engine replacement problem.

The agent is Harold Zurcher, maintenance manager at Madison Metropolitan Bus Company. The observed state variable x_t is the accumulated mileage on the bus *since last replacement.* The choice set is $D(x_t) = \{0, 1\}$, where 1 denotes the replacement decision, and 0 denotes the decision

to keep the current bus engine. The utility function u has the following form

$$u(x,d,\theta) = \begin{cases} -tr - c(0,\theta) + \epsilon(1) & \text{if } d = 1 \\ -c(x,\theta) + \epsilon(0) & \text{if } d = 0 \end{cases} \tag{2.9}$$

where tr denotes the total replacement costs for removing the old engine and installing a new engine (net of scrap value of the old engine). In the formulation of the model, tr is actually a component of θ (i.e. it is a parameter to be estimated) but it is notationally more convenient to give it a mnemonic symbol. The utility function (2.9) indicates why I call the model a *regenerative optimal stopping model* of bus engine replacement: once the bus engine is replaced it is assumed to be "as good as new" so the state of the system regenerates to the state $x = 0$. This regeneration property is formally defined by the stochastic process governing the stochastic evolution of x given by the transition probability $p(x_{t+1}|x_t, d_t, \theta)$ below.

$$p(x_{t+1}|x_t, d_t, \theta) = \begin{cases} g(x_{t+1} - 0|\theta) & \text{if } d = 1 \\ g(x_{t+1} - x_t|\theta) & \text{if } d = 0 \end{cases} \tag{2.10}$$

If d is fixed at 0, formula (2.10) implies that total bus mileage follows a random walk, where the monthly incremental mileage $(x_{t+1} - x_t)$ is distributed according to the density $g(dx|\theta)$ with support on the interval $[0, \infty)$. However, when $d = 1$ the engine is replaced, the state x_t is reset to 0, and the process begins again. Thus, (2.10) defines a *regenerative random walk* for the controlled stochastic process $\{x_t\}$.

I assume that the unobserved state variables $\epsilon_t = \{\epsilon_t(0), \epsilon_t(1)\}$ are an *IID* bivariate extreme value process with mean normalized to $(0, 0)$ and variance normalized to $(\pi/6, \pi/6)$. $\epsilon_t(0)$ should be interpreted as an unobserved component of maintenance and operating costs for the bus in period t. A large negative value for $\epsilon_t(0)$ (representing a large positive cost associated with keeping the current bus engine) could indicate an unobserved component failure that reveals other problems in the bus engine, making it economic to replace the entire engine rather than repair the individual component. A large positive value for $\epsilon_t(0)$ could be interpreted as a report from the bus driver that the bus is operating exceptionally well, making it uneconomic to replace the engine. $\epsilon_t(1)$ should be interpreted as an unobserved component of the costs associated with replacing an old bus engine with a newly rebuilt engine. A large negative value for $\epsilon_t(1)$ (representing a large positive cost for replacing the engine) could indicate that all available service bays in the company shop are full, or alternatively, that there are no available rebuilt engines at time t. A large positive value of $\epsilon_t(1)$ could indicate empty service bays, unoccupied mechanics, or surplus inventories of rebuilt engines.

Neither the location nor the scale of these unobserved costs are identifiable without additional information, the reason for the arbitrary normalizations of the mean and variance. Estimates of the θ parameters (including β and tr) must be interpreted relative to this normalization of unobserved costs.

The assumption that unobserved costs $\{\epsilon_t\}$ follow an *IID* process is admittedly quite strong. However Rust (1987,1988) showed that there is an easy way to test assumption (CI): simply include the lagged control variable d_{t-1} as a covariate in the current utility function (2.9). If (CI) holds, then the coefficient α on d_{t-1} will tend asymptotically to 0. If (CI) doesn't hold, then $\{\epsilon_t\}$ is a serially correlated process, and lagged values of the control variable d_t will be useful for predicting future values of ϵ_t and therefore future decisions since d_{t-1} is generally a function of ϵ_{t-1}. A Wald, Likelihood Ratio or LaGrange multiplier test of the hypothesis $\alpha = 0$ provides a simple way to test the validity of (CI). The estimation software automatically provides an option to include lagged dependent variables (in `STORDAT`), allowing one to see whether or not (CI) is a good approximation in the bus problem.

We are now in position to show how the general equations (2.6) and (2.7) translate into the specific equations of the bus engine replacement problem. The first thing to note is that regeneration property of the markov process for x_t given in (2.10) implies that for all x we have $EV_\theta(x,1) = EV_\theta(0,0)$; i.e., the decision to replace the bus engine regenerates the state of the system to $x = 0$ so that for any x the expectation of the value function when $d = 1$ is identical to the expectation of the value function when $x = 0$ and $d = 0$. It is a simple exercise to formally verify this identity by substituting the transition probability function given in (2.10) into the general fixed point equation for EV_θ given in (2.7). This means that we can reduce the number of unknown functions $EV_\theta(x,d)$ in the bus replacement problem from 2 to 1, and hereafter $EV_\theta(x)$ will denote the expected value of not replacing the bus engine, $EV_\theta(x,0)$. Using this expression it is straightforward to show that the functional equation (2.7) can be written as:

$$\begin{aligned} EV_\theta(x) = \int_0^\infty log\,[\,exp\{-c(x+y,\theta) + \beta EV_\theta(x+y)\} + \\ exp\{-tr - c(0,\theta) + \beta EV_\theta(0)\}]g(dy|\theta). \end{aligned} \tag{2.11}$$

The conditional choice probability (2.6) reduces to

$$P(0|x,\theta) = \frac{1}{[1 + exp\{c(x,\theta) - tr - \beta EV_\theta(x) - c(0,\theta) + \beta EV_\theta(0)\}]}. \tag{2.12}$$

$P(0|x,\theta)$ represents the probability of keeping the current bus engine as a function of accumulated mileage x; its complement $P(1|x,\theta)$ is the hazard function that gives the probability of replacing a bus engine as a function of accumulated mileage x.

2.4 Translating the bus engine replacement problem into GAUSS code

Since the observed state variable x is a continuous variable, the required fixed point EV_θ is actually an infinite dimensional object: it lies in the Banach space B of all bounded, measurable functions of x under the supremum norm. In order to compute EV_θ on a digital computer it is necessary to discretize the state space so that the state variable x takes on only finitely many values. [1] Once this is done the function EV_θ becomes a finite dimensional vector, with dimension equal to the number of possible values the discretized state variable x can assume. Thus, if we discretize x so that it can assume n possible values, the required fixed point EV_θ becomes a vector in R^n. In essence, the discretization procedure amounts to approximating the fixed point to a contraction mapping T_θ on an infinite dimensional Banach space B by the fixed point to another contraction mapping $T_{n,\theta}$ on a high dimensional Euclidean space R^n. For further discussion of approximation methods for solving dynamic programming problems, see Rust, 1996.

A natural way to discretize the state space in the bus engine replacement problem is to divide mileage into equally sized ranges, say, of length 5,000. If I put an upper bound on mileage at 450,000 miles (in the data set no bus ever got more than 400,000 miles between engine replacements), then $n = 90$, and EV_θ is a vector in R^{90}. Hereafter I will allow the state variable x to assume only the discrete integer values in the set $\{1, \ldots, n\}$, which are interpreted as mileage ranges in the underlying continuous state space.

Using the discretized state variable x, and assuming d is given by some trial decision rule $d = \delta(x)$, the transition probability function $p(x'|x, \delta(x), \theta)$ becomes an $n \times n$ Markov transition matrix. In this particular case, the monthly mileage distribution $p(x'|x, \delta(x), \theta)$ equals the density $g(x' - x|\theta)$ when $d = 0$, which reduces to a simple multinomial distribution on the discretized state space. For example, in the case where $n = 90$, For example consider the case of a trinomial distribution on the set $\{0, 1, 2\}$, corresponding to monthly mileage in the intervals $[0, 5000)$, $[5000, 10000)$ and $[10000, \infty)$, respectively. Note that by putting an upper bound on mileage x requires us to place an absorbing state at $x = n$. Thus, if at time t a bus is in state $x = n$ and no

[1] An alternative procedure, which does not require discretization of the state space, is to use the "random multigrid algorithm" introduced by Rust, 1995.

replacement occurs, then at time $t+1$ it will also be in state $x = n$ with probability 1. If we let $\pi_j = Pr\{x_{t+1} = x_t + j\}$, $j = 0, 1, 2$, and assuming a decision rule $\delta(x) = 0$ (never replace the bus engine) then we can represent the discrete transition probability by an $n \times n$ Markov transition matrix Π given by:

$$\Pi = \begin{bmatrix} \pi_0 & \pi_1 & \pi_2 & 0 & 0 & \cdot & \cdot & \cdot & & & 0 \\ 0 & \pi_0 & \pi_1 & \pi_2 & 0 & 0 & \cdot & \cdot & \cdot & & 0 \\ 0 & 0 & \pi_0 & \pi_1 & \pi_2 & 0 & 0 & \cdot & \cdot & \cdot & 0 \\ 0 & 0 & 0 & \pi_0 & \pi_1 & \pi_2 & 0 & 0 & \cdot & \cdot & \cdot \\ \cdot & \cdot & & & \cdot & \cdot & \cdot & & & & \cdot \\ \cdot & \cdot & & & & \cdot & \cdot & \cdot & & & \cdot \\ 0 & & & & & & & \pi_0 & \pi_1 & \pi_2 & 0 \\ 0 & & & & & & & & \pi_0 & \pi_1 & \pi_2 \\ 0 & 0 & & & & & & & & \pi_0 & 1-\pi_1 \\ 0 & 0 & 0 & & & & & & & & 1 \end{bmatrix}. \tag{2.13}$$

This is the transition probability matrix for the uncontrolled Markov process (i.e. assuming replacement never occurs). Notice how the (n, n) element of Π reflects the presence of the absorbing state at $x = n$.

I am now ready to show how equations (2.11) and (2.12) of the bus replacement problem translate into GAUSS code. Let `x` represent the $n \times 1$ vector of all the values x can assume, i.e. `x=(1,...,n)'` (note that the apostrophe after a vector denotes the transpose operator). Let `c` denote the $n \times 1$ vector of all values the cost function can assume, i.e. `c=(c(1,`θ`),...,c(n,`θ`))'`. Let `ev` denote the $n \times 1$ vector of all values the expected value function can assume, i.e. `ev=`$(EV_\theta(1), \ldots, EV_\theta(n))'$. Let `pk` denote the $n \times 1$ vector of all the values that $P(0|x, \theta)$ can assume, i.e. `pk=`$(P(0|1, \theta), \ldots, P(0|n, \theta))'$. Finally, let `P` denote the $n \times n$ markov transition matrix given in (2.13) and let `bet` denote the discount factor β. Using these symbols, the GAUSS code for equation (2.11) is given by

```
ev=P*ln(exp(-c+bet*ev)+exp(-tr-c[1,1]+bet*ev[1,1]));
```
(2.14)

and the GAUSS code for equation (2.12) is given by

```
pk=1/(1+exp(c-bet*ev-tr-c[1,1]+bet*ev[1,1]));
```
(2.15)

The GAUSS code is somewhat confusing at first glance since it appears to mix scalars and vectors. For example, in (2.15) one adds the scalar `-tr` to the $n \times 1$ vector `(c-bet*ev)`. Just remember that

whenever GAUSS encounters the sum of a scalar and a vector, it adds the scalar to each component of the vector. Similarly, when GAUSS evaluates a function such as "ln" or "exp" which has an $n \times 1$ vector argument, the result is also an $n \times 1$ vector that equals the function evaluated on a component by component basis. Thus, if $x = (1, ..., n)'$, then $exp(x) = (exp(1), ..., exp(n))'$. Given this syntax, you can see how the equation for `pk` given in (2.15) is the natural vectorized version of the conditional choice probability function given in (2.12). Clearly `1-pk` is the vectorized version of the hazard function.

In order to see how the GAUSS code in (2.14) corresponds to the fixed point equation given in (2.11), note that left multiplication by the transition probability matrix `P` corresponds to taking the conditional expectation of a vector. For example, `P*c` is the conditional expectation function of the cost function c when $d = 0$: `P*c`$= (\int c(y)p(dy|1, 0), \ldots, \int c(y)p(dy|n, 0))'$, where the integral sign signifies a finite sum. It follows that left multiplication by P in (2.14) is the discrete analog to integration with respect to the probability density function $p(x'|x, d, \theta)$ in equations (2.7) and (2.11).

This completes the description of the basic equations of the model and their translations into GAUSS code. The problem now is to design an algorithm that iteratively searches over values of θ to maximize the likelihood function $L(\theta)$ given in (2.5) or the partial likelihood function $L_p(\theta)$ given in (2.8) and an algorithm that iteratively searches over functions EV to find the unique fixed point $EV_\theta = T_\theta(EV_\theta)$ to the contraction mapping given in (2.14). In the next chapter I describe the nested fixed point optimization algorithm that acomplishes this, and present its translation into specific GAUSS code.

3. The NFXP Algorithm: Theory and Computer Code

3.1 Overview

The nested fixed point algorithm is really two algorithms, an "outer" optimization algorithm that searches over values of the unknown parameters θ to maximize the likelihood function $L(\theta)$ (or partial likelihood $L_p(\theta)$), and an "inner" fixed point algorithm that computes the fixed point EV_θ corresponding to the current value of θ. The outer optimization algorithm is based on a method known as *BHHH*, a quasi-Newton method that uses the outer product of the derivatives of the likelihood function as an approximation to the hessian matrix. The inner fixed point algorithm is known as a *polyalgorithm* since it combines two algorithms — *contraction iterations* and *Newton-Kantorovich iterations* — in order to efficiently compute the fixed point of the contraction mapping T_θ. The fixed point algorithm must be nested inside the outer BHHH optimization algorithm since the likelihood function depends on the fixed point function EV_θ. Formally, one can view the nested fixed point algorithm as solving the following constrained optimization problem:

$$\max_{\theta, EV} L(\theta, EV) \quad \text{subject to } EV = T_\theta(EV). \tag{3.1}$$

Since the contraction mapping T_θ always has a unique fixed point, the constraint $EV = T_\theta(EV)$ implies that the fixed point EV_θ is an implicit function of θ. Thus, the constrained optimization problem (3.1) reduces to the unconstrained optimization problem

$$\max_{\theta} L(\theta, EV_\theta). \tag{3.2}$$

where EV_θ is the implicit function defined by $EV_\theta = T_\theta(EV_\theta)$. It follows that one has to compute the fixed point EV_θ in order to evaluate the likelihood function $L(\theta)$. Thus, we seek an inner fixed point algorithm that computes EV_θ as rapidly as possible and for an outer optimization algorithm that finds the maximizing value of θ with the smallest number of function evaluations.

3.2 The Fixed Point Polyalgorithm

I use the term "polyalgorithm" since the fixed point EV_θ is calculated by a combination of two iterative methods: 1) contraction iterations, and 2) Newton-Kantorovich iterations. To describe these methods, it is useful to develop some compact notation. The equation defining the fixed point EV_θ can be written as $EV_\theta = T_\theta(EV_\theta)$, where the nonlinear operator $T_\theta : B \to B$ (B the Banach space of bounded, measurable functions on $[0, \infty)$) is defined by

$$T_\theta(W)(x) = \int_0^\infty log\,[\,exp\{-c(x+y,\theta) + \beta W(x+y)\} + exp\{-tr - c(0,\theta) + \beta W(0)\}]g(dy|\theta). \tag{3.3}$$

It is easy to show that T_θ is a contraction mapping, i.e., $\|T_\theta(W) - T_\theta(V)\| \leq \beta\|W - V\|$, $\beta \in [0, 1)$. The Banach fixed point theorem guarantees that T_θ has a unique fixed point EV_θ that can be computed by a very simple algorithm: the method of *successive approximations.* Under this method one starts with an arbitrary initial guess for the fixed point, say $EV_0 = 0$, and generates an updated estimate EV_k by the formula $EV_{k+1} = T_\theta(EV_k)$. By successive substitutions we can rewrite the k^{th} stage estimate as $EV_k = T_\theta^k(EV_0)$, where $T_\theta^2(W) = T_\theta(T_\theta(W))$, and higher powers of T_θ are defined recursively. As $k \to \infty$ one can show that $EV_k \to EV_\theta$ in the sense of the uniform or sup norm, i.e. $\|EV_k - EV_\theta\| \to 0$. Since the successive approximations method simply consists of iterating the contraction mapping T_θ beginning with an arbitrary initial estimate, the method is also known as "contraction iterations".

$$EV_k = T_\theta(EV_{k-1}) = T_\theta^k(EV_0). \tag{3.4}$$

An alternative method is the "Newton-Kantorovich" iteration. The idea behind this method is to convert the problem of finding a fixed point $EV_\theta = T_\theta(EV_\theta)$ into the problem of finding a zero of the nonlinear operator $F = (I - T_\theta)$ since $EV_\theta = T_\theta(EV_\theta)$ *iff* $[I - T_\theta](EV_\theta) = 0$, where I is the identity operator on B, and 0 is the zero element of B (i.e. the zero function). One can show that the nonlinear operator $[I - T_\theta]$ has a Fréchet derivative $[I - T_\theta']$ which is a bounded linear operator on B with a bounded inverse $[I - T_\theta']$. The invertibility of $[I - T_\theta']$ allows one to do a standard Taylor series expansion of the equation $[I - T_\theta](EV_k) = 0$ about the current fixed point estimate EV_k:

$$0 = [I - T_\theta](EV_{k+1}) \sim [I - T_\theta](EV_k) + [I - T_\theta'](EV_{k+1} - EV_k). \tag{3.5}$$

Using the invertibility of $[I - T_\theta']$ we can solve for EV_{k+1} to obtain the following formula for the Newton-Kantorovich iteration:

$$EV_{k+1} = EV_k - [I - T_\theta']^{-1}(I - T_\theta)(EV_k). \tag{3.6}$$

Each of the iterations defined in (3.4) and (3.6) has its own strengths and weaknesses. The virtue of the contraction iteration is that it is easy to program and fast to compute. Unfortunately, while contraction iterations are always guaranteed to converge, after the first 20 or 30 iterations the method begins to slow down, with the errors satisfying the following bounds $\|EV_{k+1} - EV_\theta\| \leq \beta\|EV_k - EV_\theta\|$. Thus, contraction iterations are said to converge linearly since the approximation error at iteration $k+1$ is a linear function of the approximation error at iteration k. When β is very close to 1, the rate of convergence of the successive approximations becomes particularly slow, taking many hundreds or even thousands of iterations in order to reduce the error from, say, $.01$, to an acceptable value in the range 10^{-6} or smaller. In practice, contraction iterations work well for the first 20 or 30 iterations when the estimate EV_k is far away from EV_θ but its progress (measured by the absolute decline in $\|EV_k - EV_{k-1}\|$ over successive iterations) slows way down once we get within a sufficiently small neighborhood of EV_θ. This is a result of *geometric convergence* of contraction iterations.

The Newton-Kantorovich algorithm behaves very differently: it works very poorly far away from EV_θ but converges very rapidly beginning from any estimate EV_0 which is in a sufficiently small neighborhood of the solution EV_θ. Once in an appropriately small neighborhood of EV_θ, (known as a *domain of attraction*) the Newton-Kantorovich iterations converge at a *quadratic rate,* i.e., $\|EV_{k+1} - EV_\theta\| \leq A\|EV_k - EV_\theta\|^2$ for some positive constant A. Quadratically convergent algorithms generally work well in practice, producing very accurate estimates in a small number of iterations. The complimentary behavior of the two algorithms suggests the following *polyalgorithm:* begin with contraction iterations until one gets within a domain of attraction of EV_θ and then switch to Newton-Kantorovich iterations to converge rapidly to the solution. The combined algorithm is guaranteed to converge, since the Newton-Kantorovich method is guaranteed to converge starting from any estimate EV_0 in a sufficiently small neighborhood of the true solution EV_θ, and the former estimate can always be produced using contraction iterations. The drawback of the Newton-Kantorovich algorithm is that it requires computing the inverse of the linear operator $[I - T'_\theta]$ at each iteration. However in practice, the substantial reduction in the number of iterations required by the method far outweighs the increase in time per iteration.

A modification of the Newton-Kantorovich algorithm, Werner's Method, (Werner, 1983) yields even faster convergence than Newton-Kantorovich at essential no extra costs. To understand Werner's method, note that implicit in the iteration (3.6) is the fact that the Fréchet derivative T'_θ is evaluated at the point EV_k. Consider instead evaluating T'_θ at the point $dEV_k + (1-d)T_\theta(EV_k)$

for some fixed constant $d \in [0, 1]$. Computing the iteration (3.6) with $d = 1$ yields the Newton-Kantorovich method, with $d = 0$ yields `Stirling's Method`, and setting $d = .5$ yields Werner's method. Werner (1983) showed that setting $d = .5$ produces a faster rate of convergence than $d = 1$ or $d = 0$. The NFXP algorithm allows the user to choose the value of d; however based on Werner's results, I set the default value equal to $d = .5$.

To summarize: the fixed point algorithm is a hybrid algorithm that combines contraction iterations with Newton-Kantorovich iterations. The algorithm begins with contraction iterations in order to generate an estimate EV_0 in a domain of attraction of EV_θ, and then switches over to the Newton-Kantorovich method in order to rapidly converge to EV_θ.

3.3 The BHHH optimization algorithm

The BHHH algorithm is a quasi-Newton gradient optimization algorithm to find the maximum likelihood parameter estimate $\hat{\theta}$; i.e. the value of θ that maximizes the likelihood function $L(\theta)$ or $L_p(\theta)$. BHHH can be viewed as a hill-climbing algorithm which chooses a search direction using the outer product of the gradients of the log-likelihood function as an approximation to the (negative of) the hessian matrix. The basic parameter iteration under BHHH can be written as

$$\theta_{k+1} = \theta_k + \lambda D(\theta_k), \tag{3.7}$$

where θ_{k+1} is the updated parameter estimate, λ_k is a stepsize parameter, and $D(\theta_k)$ is the direction vector. BHHH reduces to Newton's method in the special case where $\lambda = 1$ and

$$D(\theta) = -\left[\partial^2 L(\theta)/\partial\theta\partial\theta'\right]^{-1} \partial L(\theta)/\partial\theta. \tag{3.8}$$

In an optimization problem the search direction parameter λ plays a useful role in speeding up convergence: given the search direction $D(\theta)$ one moves along this search direction not just for a distance $\lambda = 1$ as in Newton's method, but chooses λ iteratively to approximately maximize the univariate function $L(\theta_k - \lambda D(\theta_k))$.

Given that the likelihood function $L(\theta) = L(\theta, EV_\theta)$ depends on the expected value function EV_θ (see formula (2.13) in chapter 2), it is easy to see that use of Newton's method to determine the direction vector in (3.7) would require computing the second derivative of the expected value function, $\partial^2 EV_\theta/\partial\theta\partial\theta'$ which is quite difficult to compute. The BHHH method is similar to Newton's method, but it uses a different choice for the direction vector $D(\theta)$ that only requires first derivatives of the loglikelihood function. The method is based on a well-known identity

from statistics, the *information equality*, that equates the expectation of the outer product of the first derivatives of the loglikelihood function to the negative of the expectation of the hessian of the loglikelihood function when both are evaluated at the "true" parameter vector θ^*. Let $L_m(\theta)$ denote the *log*-likelihood for a single individual m, $L_m(\theta) = \sum_{t=1}^{T} log[P(d_t^m|x_t^m, \theta)] + \sum_{t=1}^{T} log[p(x_t^m|x_{t-1}^m, d_{t-1}^m, \theta)]$. In the bus example, "individuals" are buses so m indexes different buses. Then if $L(\theta)$ is the log-likelihood for the full sample we have $L(\theta) = \sum_{m=1}^{L} L_m(\theta)$ and the information identity can be written as:

$$E\left\{[\partial L(\theta^*)/\partial\theta][\partial L(\theta^*)/\partial\theta']\right\} = -E\left\{\partial^2 L(\theta^*)/\partial\theta\partial\theta'\right\}. \tag{3.9}$$

This identity is the key to the BHHH method: if sample sizes are large and if the parameter estimate θ is close to the true value θ^*, then the sample average of the outer products of the first derivatives of the loglikelihood should be a good approximation to the negative of the hessian. Thus, BHHH uses the direction vector specified by

$$D(\theta) = \frac{1}{M}\sum_{l=1}^{M}[\partial L_m(\theta)/\partial\theta][\partial L_m(\theta)/\partial\theta']. \tag{3.10}$$

Since BHHH requires only first derivatives of the loglikelihood function, one only needs to calculate the first derivatives of the expected value function, $\partial EV_\theta/\partial\theta$. These derivatives can be calculated analytically using the implicit function theorem:

$$\partial EV_\theta/\partial\theta = [I - T'_\theta]^{-1}\partial T_\theta(EV_\theta)/\partial\theta. \tag{3.11}$$

To complete the discussion of the BHHH algorithm, I need to specify the algorithm used to generate successive estimates of the step size parameter, λ. The algorithm attempts to find a stepsize λ^* that maximizes the univariate function $f(\lambda) \equiv L(\theta + \lambda D(\theta))$ in the variable λ. At the maximum, we have $\partial f(\lambda)/\partial\lambda = 0$. This suggests attempting to find λ^* iteratively by solving the equation $\partial f(\lambda^*)/\partial\lambda = 0$ using Newton's method, which yiels iterations of the form

$$\lambda_{k+1} = \lambda_k - \left[\partial^2 f(\lambda_k)/\partial^2\lambda\right]^{-1}\partial f(\lambda_k)\partial\lambda. \tag{3.12}$$

The difficulty with using Newton's method directly is the need to compute the second derivative of $f(\lambda)$. This would require computing second derivatives of $L(\theta)$ and hence calculation of $\partial EV_\theta/\partial\theta\partial\theta'$. To avoid this, I use the *secant iteration.* The secant iteration is simply Newton's method with a finite difference approximation to $\partial^2 f(\lambda)/\partial\lambda$. Thus, the secant iteration for determining the stepsize λ^* takes the form

$$\lambda_{k+1} = \lambda_k - \frac{(\lambda_k - \lambda_{k-1})\partial f(\lambda_k)/\partial\lambda}{\partial f(\lambda_k)/\partial\lambda - \partial f(\lambda_{k-1})/\lambda}. \tag{3.13}$$

Since the secant iteration has nearly the same rate of convergence as Newton's method (see Luenberger 1984) but requires computation of only the first derivatives of the loglikelihood function, it provides a nearly ideal linesearch method for the nested fixed point algorithm.

To complete the linesearch algorithm, one needs a stopping rule to terminate the linesearch. An overly accurate linesearch may require many likelihood function evaluations without obtaining significant improvement in the likelihood function. On the other hand an inaccurate linesearch may fail to move far enough along a direction of increase, requiring many additional iterations in order to converge. Since the marginal cost of a linesearch is less than the cost of an extra iteration (due to the fact that the moment matrix of first derivatives need not be computed during the linesearch), it pays to continue the linesearch as long as there is "sufficient" improvement in the likelihood function. There are many alternative criteria for determining when there is sufficient improvement in the likelihood function. Perhaps the best known criteria are the Goldstein and Armijo step size rules. In addition to these criteria, I have found that a relatively simple termination rule works well in practice: continue the linesearch as long as the percentage increase in the likelihood function exceeds a user-specified tolerance, subject to constraints on minimum and maximum number of linesearch iterations.

Figure 1 (which can be viewed by clicking on the NFXP.GIF icon on the NFXP home page, `http://gemini.econ.wisc.edu/jrust/nfxp`) summarizes the complete nested fixed point algorithm. The box on the right contains the outer BHHH optimization algorithm, and the box on the left contains the inner fixed point polyalgorithm.

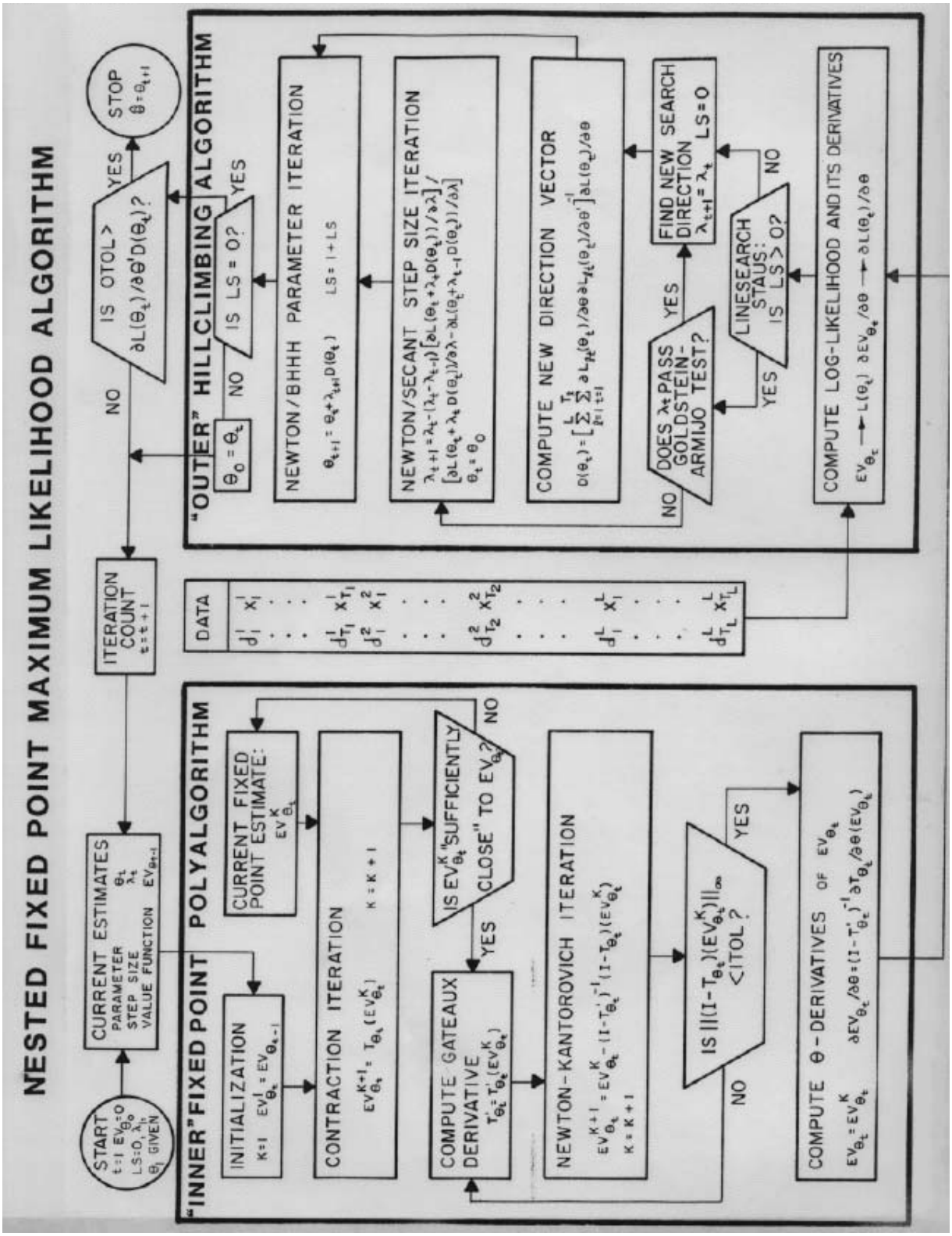
3.4 Translating the fixed point algorithm into GAUSS code

The GAUSS code for the fixed point algorithm has two sections; a section that performs the contraction iterations and a second section that performs the Newton-Kantorovich iterations. The first section is the easiest to explain, so I'll begin there. In what follows I will be using the symbols `x`, `c`, `ev`, and `P` which were defined in chapter 2.

Let `ev` be an initial estimate of the fixed point, EV_θ. A single contraction iteration in GAUSS produces an updated estimate `ev1` given by the formula

```
ev1=P*ln(exp(-c+bet*ev)+exp(-tr-c[1,1]+bet*ev[1,1]));        (3.14)
```

where `P` is the GAUSS symbol for the transition matrix Π defined in (2.13), and `bet` is the GAUSS symbol for the discount factor β. Equation (3.14) provides the code for a single contraction iteration


NESTED FIXED POINT MAXIMUM LIKELIHOOD ALGORITHM
START
CURRENT ESTIMATES
PARAMETER
STEP SIZE
VALUE FUNCTION
ITERATION COUNT
DATA
"INNER" FIXED POINT POLYALGORITHM
INITIALIZATION
CONTRACTION ITERATION
CURRENT FIXED POINT ESTIMATE:
IS "SUFFICIENTLY CLOSE" TO ?
COMPUTE GATEAUX DERIVATIVE
NEWTON-KANTOROVICH ITERATION
IS ?
COMPUTE Θ-DERIVATIVES OF
"OUTER" HILLCLIMBING ALGORITHM
IS OTOL>
NO
YES
STOP
NEWTON/BHHH PARAMETER ITERATION
IS LS = 0?
NEWTON/SECANT STEP SIZE ITERATION
COMPUTE NEW DIRECTION VECTOR
DOES PASS GOLDSTEIN-ARMIJO TEST?
FIND NEW SEARCH DIRECTION
LINESEARCH STAUS:
IS LS > 0?
COMPUTE LOG-LIKELIHOOD AND ITS DERIVATIVES

$EV_{k+1} = T_\theta(EV_k)$. In order to save computer memory space, instead of defining the next iterate to be `ev2`, I use `ev` to store the output of the second contraction iteration:

```
ev=P*ln(exp(-c+bet*ev1)+exp(-tr-c[1,1]+bet*ev1[1,1]));
```
(3.15)

The code for the contraction iteration simply loops around equations (3.14) and (3.15) until it generates an estimate in a domain of attraction of the true fixed point EV_θ, at which time it switches to the Newton-Kantorovich iterations. Since the heuristic for determining when one is in a domain of attraction of the true fixed point is somewhat complicated, I will defer explaining it until I have explained the Newton-Kantorovich iterations.

Computation of the Newton-Kantorovich iterations requires computation of the Fréchet derivative of the contraction mapping T_θ and computation of the inverse of the linear operator $[I - T'_\theta]$. Consider first the computation of T'_θ. In terms of its finite-dimensional approximation, T'_θ takes the form of an $n \times n$ matrix equal to the partial derivatives of the $n \times 1$ vector $T_\theta(EV_\theta)$ with respect to the $n \times 1$ vector EV_θ. Using equation (2.111) of chapter 2 it is straightforward to compute the elements of this matrix. It turns out that T'_θ is simply β time the transition probability matrix for the controlled process $\{d_t, x_t\}$. Since T'_θ is a transition probability matrix, its rows sum to 1. Therefore I present columns 2 through `n` of T_θ, using the shorthand notation, $P_x = P(0|x)$, and $\pi_j = Pr\{x_{t+1} = x_t + j\}$, $j = 0, 1, 2$, and $x = 1 \ldots, n$.

$$T'_\theta = \begin{bmatrix} P_2\pi_1 & P_3\pi_2 & 0 & 0 & 0 & \cdot & \cdot & 0 \\ P_2\pi_0 & P_3\pi_1 & P_4\pi_2 & 0 & 0 & \cdot & \cdot & 0 \\ 0 & P_3\pi_0 & P_4\pi_1 & P_5\pi_2 & 0 & \cdot & \cdot & 0 \\ 0 & 0 & \cdot & \cdot & \cdot & & & 0 \\ \cdot & \cdot & & \cdot & \cdot & \cdot & & 0 \\ \cdot & \cdot & & & \cdot & \cdot & \cdot & 0 \\ \cdot & \cdot & & & & \cdot & \cdot & \cdot \\ 0 & \cdot & & & & & \cdot & \cdot \\ 0 & 0 & 0 & \cdot & \cdot & 0 & 0 & P_n \end{bmatrix}. \tag{3.16}$$

The GAUSS code for the matrix representation of T'_θ is given below.

$$T'_\beta = \texttt{bet*(1-sumc((P[.,2:n].*pk[2:n,1]')'))} \sim \texttt{(P[.,2:n].*pk[2:n,1]'));} \tag{3.17}$$

To understand (3.17), note that it specifies T'_θ as the horizontal concantenation of an $n \times 1$ vector and and $n \times (n-1)$ matrix, joined using the GAUSS concantenation operator $\sim$. Consider the

second part, `(P[.,2:n].*pk[2:n,1]')`. This is the GAUSS coding of the last $n-1$ columns of T'_θ presented in (3.16). The term `P[.,2:n]` specifies the $n \times (n-1)$ submatrix consisting of columns 2 to n of the transition probability matrix `P`. This matrix is "swept out" by the $1 \times (n-1)$ vector `pk[2:n,1]` which is simply the probability of keeping the current engine in states 2 through n. The element by element multiplications that occur in this sweeping out operation are induced by the GAUSS operator `.*`. The other term, `(1-sumc((P[.,2:n].*pk[2:n,1]')')`, simply fills in the first column of the T'_θ matrix making use of the fact that T'_θ is a transition probability matrix and its rows must sum to 1.

Having derived the coding for the linear operator T'_θ, it is now a simple matter to display the GAUSS code for the Newton-Kantorovich iterations. Let `ev1` be the result of one contraction iteration starting from the initial estimate `ev`. Then the formula for a single Newton-Kantorovich iteration is given by

```
ev1=ev-(ev-ev1)/(eye(n)-bet*((1-sumc(P[.,2:n].*pk[2:n,1]')')) ~
                          (p[.,2:n].*pk[2:n,1]')));
```
(3.18)

To understand (3.18), recall the general formula for a single Newton-Kantorovich iteration: $EV_{k+1} = EV_k - [I - T'_\theta]^{-1}(I - T_\theta)(EV_k)$. In formula (3.18) `ev1` corresponds to the output of the Newton-Kantorovich iteration EV_{k+1}; `(ev-ev1)` corresponds to the term $(I - T_\theta)(EV_k)$, and `eye(n)` corresponds to the $n \times n$ identity matrix I. The "/" sign instructs GAUSS to invert the matrix code for $[I - T'_\theta]$ and multiply the resulting matrix with the vector $(I - T_\theta)(EV_k)$. Actually, the divide sign solves the linear system $[I - T'_\theta](EV_{k+1} - EV_k) = (I - T_\theta)(EV_k)$ for the solution vector $(EV_{k+1} - EV_k)$. This is faster than inverting the matrix $[I - T'_\theta]$ and multiplying it by the vector $(I - T_\theta)(EV_k)$.

In order to save memory space for the computer, the output `ev1` of the first Newton-Kantorovich iteration (3.18) becomes the input to the next. More precisely, let `ev` equal one contraction iteration from the output `ev1` of (3.18), then the second Newton-Kantorovich iteration produces an estimate `ev` given by

```
ev=ev1-(ev1-ev)/(eye(n)-bet*((1-sumc(P[.,2:n].*pk[2:n,1]')')) ~
                          (P[.,2:n].*pk[2:n,1]')));
```
(3.18)

Similar to the contraction iteration phase, the Newton Kantorovich algorithm loops around equations (3.18) and (3.19) until convergence is achieved. The convergence test is based on the GAUSS variable `tol`, given by `tol=maxc(abs(ev-ev1));`. The variable `tol` is simply the sup norm, $\|EV_k - T_\theta(EV_k)\|$. When `tol` is less than a fixed tolerance chosen by the user, `FFXP` terminates

the Newton-Kantorovich loop with one final iteration that simultaneously computes the derivatives $\partial EV_\theta/\partial\theta$ along with a final estimate of the fixed point EV_θ. Given the quadratically convergent property of the Newton-Kantorovich iteration, setting a stopping tolerance of 10^{-6} produces a final estimate of EV_θ with an error of 10^{-12} or smaller.

Before describing the calculation of $\partial EV_\theta/\partial\theta$ in the final iteration, it is important to note that recentering of the value function in the `contract.g` procedure is critical for insuring numerical stability of the fixed point algorithm. The recentering algorithm is necessary due to the exponential terms in equation (2.11) can lead to serious overflow problems when the value function is large. Fortunately there is a simple way to deal with this problem, via recentering. To see this note that for and K, (2.11) can be rewritten as:

$$\begin{aligned} EV_\theta(x) = &\int_0^\infty log\,[\,exp\{-c(x+y,\theta)+\beta EV_\theta(x+y)\}+ \\ &exp\{-tr-c(0,\theta)+\beta EV_\theta(0)\}]g(dy|\theta) \\ =&K+\int_0^\infty log\,[\,exp\{-c(x+y,\theta)+\beta EV_\theta(x+y)-K\}+ \\ &exp\{-tr-c(0,\theta)+\beta EV_\theta(0)-K\}]g(dy|\theta). \end{aligned} \tag{3.20}$$

Thus, we can choose $K = max_x[-c(x+y)+\beta EV_\theta(x+y)]$ thereby eliminating any overflow problems.

Now consider the problem of calculating the derivatives of the expected value function, $\partial EV_\theta/\partial\theta$. From formula (3.11) $\partial EV_\theta/\partial\theta = [I - T'_\theta]\partial T_\theta(EV_\theta)/\partial\theta$ so the problem reduces to calculating the θ derivatives of T_θ. Using formula (3.3) it is easy to compute these derivatives and translate them into the following GAUSS code:

$$\begin{aligned} \partial T_\theta(EV)\partial\beta &= \texttt{(-bet*(1-bet)*(ev[1,1]+P*(ev-ev[1,1]).*pk)))} \\ \partial T_\theta(EV)/\partial tr &= \texttt{(P*pk-1)} \\ \partial T(EV)/\partial\theta &= \texttt{-(P*dc).*pk} \end{aligned} \tag{3.21}$$

Note that in (3.21) the symbol `dc` denotes the derivative of the cost function $c(x,\theta)$ with respect to θ. To compute the derivatives of $T_\theta(EV)$ with respect to $\theta_3 \equiv (\pi_0,\pi_1,\pi_2)$, note that the vector (π_0,π_1,π_2) satisfies the constraint $1 = \pi_0+\pi_1+\pi_2$. Eliminating the parameter π_2 using the constraint leaves us with the unconstrained parameter vector $\theta_3 = (\pi_0,\pi_1)$ (we ignore the constraints that the π_j's must lie within the unit interval). The GAUSS code for the derivatives of $T_\theta(EV)$ with respect to this reduced parameter vector are given below:

$$\begin{aligned} \partial T_\theta(EV)/\partial\theta_3 =&\texttt{((dtp[1:n-2,1]-dtp[3:n,1])|(dtp[n-1,1]-dtp[n,1])|0)} \sim \\ &\texttt{((dtp[2:n-1,1]-dtp[3:n,1])|0|0)} \end{aligned} \tag{3.22}$$

where `dtp=ln(exp(-c+bet*ev)+exp(-tr-c[1,1]+bet*ev[1,1]))`.

Let the GAUSS symbol for $\partial EV_\theta/\partial\theta$ be `dev`, and the GAUSS symbol for $\partial T_\theta(EV)/\partial\theta$ be `dtp`. Then the GAUSS code for $\partial EV_\theta/\partial\theta$ is given by the following formula

```
dev=dtp/(eye(n)-bet*((sumc(P[.,2:n].*pk[2:n,1]')')) ~
                    (P[.,2:n].*pk[2:n,1]')));
```
(3.23)

This formula is the GAUSS analog of the formula for $\partial EV_\theta/\partial\theta$ given in equation (3.11). Note that the program `FFXP.PRC` uses a different version of formula (3.23). The reason has to do with efficiently computing `dev` allowing for user options to estimate certain subsets of the full parameter vector θ.

To conclude the description of the fixed point algorithm, I explain how the program determines when a given estimate EV_k is "sufficiently close" to the true fixed point EV_θ to safely switch from contraction iterations to Newton-Kantorovich iterations. The decision rule is an heuristic that determines the switch point from the contraction phase to the Newton-Kantorovich phase based on two factors:

1) the absolute change in the expected value function as measured by `tol`,

2) the rate of progress of the contraction iterations as measured by the ratio of successive changes in the value function, `tol1/tol`.

In the case where the fixed point algorithm is imbedded inside the outer BHHH optimization algorithm, one can adjust the switch point adaptively based on the previous performance of the fixed point algorithm. This leads to two additional factors that determine the switch point:

3) the number of Newton-Kantorovich iterations used in the last fixed point computation,

4) the quality of the starting estimate $EV_{\theta_{k-1}}$ as indicated by the magnitude of change in the parameter values $\|\theta_k - \theta_{k-1}\|$.

The switch point is adaptively modified as follows. At the beginning of `NFXP.GPR` the user specifies an initial range for the number permissible contraction iterations. The GAUSS variable `cstp` specifies the upper bound and `minstp` specifies the lower bound. Initially the program sets the default value `minstp=cstp-2`. Each time `NFXP.GPR` calls the fixed point algorithm `FFXP.PRC`, the latter must perform a minimum of minstp contraction iterations before switching to Newton Kantorovich iterations, with one exception: if at any point the change in the expected value

function over successive contraction iterations, `tol`, is less than a specified value (currently .001), then regardless of the values of `cstp` and `minstp`, the algorithm will immediately branch to the Newton Kantorovich iteration phase. However, provided `tol` is greater than .001, the algorithm will always perform at least `minstp` contractions. After performing the `minstp` iterations, the algorithm will continue doing contractions until it hits the upper bound `cstp` unless the following condition is satisfied: if the ratio `tol1/tol` of successive contraction steps is sufficiently close to β, then the algorithm will immediately switch to Newton Kantorovich iterations.

To see the intuition behind switching to the Newton Kantorovich iterations when `tol1/tol >bet-rtol`, (where `rtol` is a small value adaptively determined by the algorithm) consider the following example. Suppose the initial estimate EV_0 equals the true fixed point EV_θ plus an arbitrary constant: $EV_0 = EV_\theta + k$. Intuitively, EV_0 is an excellent starting estimate, and one wouldn't want to perform many contraction iterations before switching to Newton Kantorovich iterations since the latter iterations will immediately "strip away" the irrelevant constant k in the first iteration. However, if we were to decide to switch depending on the value of `tol`, we would be mislead into performing many contraction iterations since `tol`$= \|EV_0 - T_\theta(EV_0)\| = (1-\beta)k$. If k is sufficiently large, then by looking only at the value of `tol` one will incorrectly conclude that they are far away from the true fixed point. However if we perform another contraction iteration we get $EV_1 = T_\theta(EV_0)$ and compute `tol1`$= \|EV_1 - T_\theta(EV_1)\| = \beta(1-\beta)k$, then the ratio `tol1/tol` is identically β. This example lead me to conjecture that whenever the ratio `tol1/tol` is sufficiently close to β, one has a good indication that the fixed point estimate is in a domain of attraction, even though the values of `tol` and `tol1` may not be small themselves.

It remains to describe how the values of `cstp`, `minstp` and `rtol` are adjusted adaptively when `FFXP.PRC` is imbedded in `NFXP.GPR`. These are all adjusted with the following objective in mind: try to set the switch point so that the fixed point algorithm converges in at most 2 Newton Kantorovich iterations. My experience has been that the optimal switch point should be set so that one requires at most two Newton Kantorovich iterations in order to converge (to tolerance 10^{-16}). The time required to perform the extra contraction steps necessary to guarantee that the algorithm converges in only 1 Newton Kantorovich iteration generally far outweighs the time saved by cutting out the second Newton Kantorovich iteration. On the other hand, it is generally not worthwhile to reduce the contraction iterations to the point where 3 or more Newton Kantorovich iterations are required. Thus, the algorithm adjusts `cstp`, `minstp` and `rtol` adaptively in order to hit a target value of at most Newton Kantorovich iterations per fixed point calculation (in the "end-game" iterations when the outer algorithm is close to convergence the algorithm shoots for a target of only

1 Newton Kantorovich iteration per likelihood function evaluation since the θ_k' s are changing by only very small amounts then). We have the following cases:

Case 1: if the last fixed point went to the upper limit of `cstp` contraction iterations but performed more than 2 Newton Kantorovich iterations, then I increase `cstp` by 10, and set `minstp=2`.

Case 2: if the last fixed point switched before the `cstp` limit due to the fact that `tol1/tol >bet-rtol` and performed more than 2 Newton Kantorovich iterations, then I cut the tolerance `rtol` in half.

These simple rules are designed to avoid taking too many Newton Kantorovich iterations. On the other hand, there is also of danger of performing too many contraction iterations. Let `ftol` equal the change in the expected value function after the first Newton Kantorovich iteration. Using `ftol` I can identify two additional cases:

Case 3: if `ftol` is too small, say less than 10^{-7}, then if the last fixed point hit the upper limit of `cstp` iterations before switching to Newton Kantorovich iterations, then I decrease `cstp` by 10 and set `minstp=2`.

Case 4: if `ftol` is too small, but the previous fixed point switched before `cstp` contractions (due to the fact that `tol1/tol>bet-rtol)`, then I double `rtol` and set `minstp=2`.

There is one final case to consider.

Case 5: if the change in θ_k from successive BHHH iterations is sufficiently small, then it is very likely that the previous fixed point EV_{θ_k} is likely to be a very good starting estimate for the current fixed point. Thus, if $\|\theta_k - \theta_{k-1}\|$ is sufficiently small, I set `cstp=minstp=2`.

Note that case 5 is applicable only between successive iterations (direction changes) of the outer BHHH algorithm: during linesearch one sometimes gets small changes in the step size parameter λ that trick the heuristic into setting `cstp=minstp=2` even though there are still substantial changes in the parameters between successive iterations.

This concludes the discussion of the fixed point algorithm. I make no claim that these heuristics for adaptively determining the time to switch from contraction iterations to Newton Kantorovich iterations are in any sense "optimal". I encourage you to experiment with your own heuristics to get faster execution times. If you want more detail about the specific operation of `FFXP.PRC`, consult Appendix 3 for an annotated listing of the program (it may be helpful to begin

with the stripped down version `FP.GPR` that exposes the basic structure of the algorithm without the extra complexity of the calculations of $\partial EV_\theta/\partial\theta$.

3.5 Exploiting the sparsity structure of the transition matrix

The fixed point algorithm described in section 3.4 treated the transition probability Π as a dense matrix, using ordinary matrix multiplication to compute the conditional expectations in (3.14) and (3.15), and a standard Crout decomposition algorithm to solve the linear system $[I - T'_\theta]W = [T_\theta(EV) - EV]$ required by the Newton-Kantorovich iterations (3.18) and (3.19). However one can see from (2.13) and (3.16) that both Π and T'_θ are highly sparse, well-structured matrices. Π is a *band matrix* with a bandwidth that is small relative to the dimension of Π. T'_θ is known as a singly bordered band diagonal matrix (Press, Flannery, Teukolsky, Vetterling, 1990). Matrices of these forms arise regularly in finite element discretizations of partial differential equations problems in physics. One strategy is to import specialized matrix multiplication and linear equation solvers that account for the special structure of these matrices, using the GAUSS foreign language facility. Unfortunately, GAUSS did not have this facility at the time I wrote this code, so I essentially re-invented the wheel by coding versions of band matrix linear algebra routines within GAUSS. It is instructive, however, to show how one can exploit special structure by constructing macro algorithms that call dense linear algebra kernels already highly efficiently coded within GAUSS.

Consider first how to efficiently code the conditional expectation operator EF defined by $Ef(x) = \int f(y)p(dy|x)$. In a discretized state space, we have seen in (2.12) that `Ef` takes the form `Ef=Pf`, where `P` is the transition probability matrix defined in (2.13). It is easy to see that the banded structure of `P`, particularly the fact that each band contains the same elements, allows one to write a simple algorithm for `Pf` that fully exploits the sparsity of `P`, substantially reducing the number of required calculations. The GAUSS code for `Ef` in the case of bandwidth of 2 is given below:

$$Ef = Pf = \texttt{(P*f[1:n-1,.]+(1-P)*f[2:n,.])|f[n,.]}. \tag{3.24}$$

The procedure `E.PRC` contains code to calculate `Ef` for problems with bandwidths from 2 to 6. The resulting code is 4 to 5 times as fast as the dense code that simply multiplies `P*f`.

Consider next the solution of linear systems involving the matrix $[I - T'_\theta]$ needed to compute the Newton-Kantorovich iterations and the partial derivatives $\partial EV_\theta/\partial\theta$. If it weren't for the border in the first column of T'_θ, $[I - T'_\theta]$ would already be in upper triangular form, requiring only a simple

back-substitution to solve any system of linear equations. Unfortunately, the process of Gaussian elimination to get $[I - T'_\theta]$ in upper triangular form creates fill-in that destroys the sparsity pattern of the matrix. An alternative is to exploit the sparsity using a partitioned-elimination algorithm that decomposes $[I - T'_\theta]$ as follows:

$$[I - T'_\theta] = \begin{bmatrix} \underset{m\times m}{Q_{11}} & \underset{m\times k}{Q_{12}} \\ \underset{k\times m}{Q_{21}} & \underset{k\times k}{Q_{22}} \end{bmatrix} \quad k + m = n. \tag{3.25}$$

Notice from (3.16) that Q_{12} and Q_{21} will have mostly zero elements. The partitioned elimination algorithm solves the linear system $[I - T'_\theta]V = W$ by partitioned inversion of (3.25), accounting for the zeros in Q_12 and Q_21. The code used in the procedure `PARTSOLV.PRC` that implements this algorithm actually only requires the matrices Q_{11} and Q_22, where Q_11 is the upper $m \times m + b$ lefthand corner of Π and Q_{22} is the lower $k \times k$ righthand corner of Π where b denotes the bandwidth of Π. `PARTSOLV.PRC` internally constructs the partition elements in (3.25), solving the system $[I - T'_\theta]V = W$ for V in the following GAUSS code

```
proc partsolv(z); local cq,cy,y1,y2,q12,q21;
y1=z[1:m,.]; y2=z[m+1:n,.]; cy=cols(y1); cq=modnum[1,5];
q21=(y2 ~ (bet-sumc((q22.*(bet.*pk[m+1:n,1])')')))/
    (eye(n-m)-(q22.*(bet*pk[m+1:n,1])'));
q12=(q11[.,m+1:m+cq].*(bet*pk[m+1:m+cq,1])')*q21[1:cq,.];
y1=(y1+q12[.,1:cy])/
    (eye(m)-((bet-sumc((q11[.,2:m+cq].*(bet*pk[2:m+cq,1])')')
    +q12[.,cy+1]) ~ (q11[.,2:m].*(bet*pk[2:m,1])')));
y2=q21[.,1:cy]+q21[.,1+cy].*y1[1,1:cy];
retp(y1|y2); endp;
```

By staring at this code for a while you should be able to convince yourself that it does indeed solve the linear system $[I - T'_\theta]V = W$. Although more cumbersome to understand, this new code runs 3 to 4 times faster than code that treats $[I - T'_\theta]$ as a dense matrix. Further speedups can be obtained by importing special FORTRAN algorithms for solution of singly bordered band diagonal systems.

3.6 Translating the BHHH algorithm into GAUSS code

I will be brief in this section since the BHHH algorithm is well-known. I focus on presenting GAUSS code for the evaluation of the loglikelihood function and its derivatives.

The basic parameter iteration in BHHH given in equations (3.7), (3.10), and (3.13) have a translation into GAUSS code given by

```
q=q0+ss2*direc;
direc=invpd(h)*dll;
ss2=ss1-df1*(ss0-ss1)/(df0-df1);
df1=dll'*direc;
```
(3.26)

The first formula is the GAUSS analog of (3.7): $\theta_{k+1} = \theta_k + \lambda D(\theta_k)$. Since GAUSS does not have the symbol θ, I use `q` instead. The matrix `h` (for hessian) is the cumulated information matrix, and the vector `dll` contains the derivatives of the loglikelihood function. The third equation is the GAUSS analog of the Secant iteration for stepsize given in (3.13). In the latter formula, we have the derivative $\partial f(\lambda)/\partial\lambda = \partial L(\theta_k + \lambda D(\theta_k))/\partial\lambda = \partial L(\theta_k + \lambda D(\theta_k))/\partial\theta D(\theta_k)$. The formula for `df1` corresponds to $\partial f(\lambda_k)/\partial\lambda$ and `df0` corresponds to $\partial f(\lambda_{k-1})/\partial\lambda$. To complete the explanation of the BHHH algorithm, I need to show how the information matrix `h` and the loglikelihood derivatives `dll` are computed.

The procedure `EVAL.PRC` contains the code which evaluates the loglikelihood function `ll`, its derivatives `dll`, and the information matrix (the cumulative outer product of the derivatives of the individual loglikelihood terms). Consider first the evaluation of the loglikelihood function. The general formula for the full and partial likelihood function is given in equation (2.5) and (2.8) in chapter 2. The GAUSS code for these functions is given below:

```
lp=submat(pk,dt[.,2],0);
lnp=ln(q )|ln(q );
ll=sumc(ln(lp+(1-2*lp).*dt[.,1])+
  submat(lnp,(1+dt[.,3]),0));
```
(3.27)

The first equation creates a $T * M \times 1$ vector `lp` defined by `lp`$= (P(0|x_1^m), \ldots, P(0|x_T^m))'$, $m = 1, \ldots, M$ and $t = 1, \ldots, T$ where recall M is the total number of buses and T is the number of time periods each bus is observed. This vector is formed by simply filling in the sample values of the state variables $\{x_t^m\}$ contained in the $M * T \times 1$ vector `dt[.,2]` into the choice probability

function $P(0|x,\theta_k)$. The filling in operation is done by the GAUSS `submat` command. The command `lp=submat(pk,dt[.,2],0)` instructs the computer to take the data in the $M*T\times 1$ vector `dt[.,2]` and fill in the entries of the choice probability function defined by the $n\times 1$ vector `pk`, producing the $M*T\times 1$ vector `lp`. The second equation shows that the 2×1 vector `lnp` contains the logs of the parameters and defining the probabilities that $(x_{t+1}-x_t)$ equals 0 or 1, respectively. Note that GAUSS does not have the symbol θ so I use the last two elements of the `q` vector, `q[1,dm-1]` and `q[1,dm]`, respectively, where `dm=dim(q)`. The $M*T\times 1$ vector `dt[.,3]` contains the sample values of the change in mileage $(x_{t+1}-x_t)$. It follows that the $M*T\times 1$ vector `submat(lnp,(1+dt[.,3]),0)` contains the loglikelihood terms $\{log[p(x_{t+1}^m|x_t^m,d_t^m,\theta)]\}$. The $M*T\times 1$ vector `dt[.,1]` contains the entries $\{d_t^m\}$, $t=1,\ldots,T$, $m=1,\ldots,M$. It follows that the $M*T\times 1$ vector `submat(ln(lp+(1-2*lp).*dt[.,1]))` contains the loglikelihood terms $\{log[P(d_t^m|x_t^m,\theta)\}$. Combining these terms and summing over the sample yields the full loglikelihood function `ll`$=L(\theta)$ given in the third equation of (3.25).

It remains to specify the GAUSS equations for `dll` and `h`. The derivative of the partial loglikelihood function $L_p(\theta)$ given in equation (2.12) of chapter 2 yields the formula

$$\partial\, log\left[P(d|x,\theta)\right]/\partial\theta = \left[1-P(d|x,\theta)\right]\partial\left[V_\theta(0)-V_\theta(x)\right]/\partial\theta. \tag{3.28}$$

where $V_\theta(x)=[-c(x,\theta)+\beta EV_\theta(x)]$. Plugging the sample data $(d_1,\ldots,d_T,x_1,\ldots,x_T)$ into (3.28) yields a $T\times dim(\theta)$ vector of the derivatives of the individual loglikelihood terms. The GAUSS code for this $T\times dim(\theta)$ matrix `d1v` is given by:

```
d1v=(-ones(T,1)) ~ submat(dc,dt[.,2],0) ~ zeros(T,2);
d1v=d1v+dev[1,.].*ones(T,1)-submat(dev,dt[.,2],0);
d1v=d1v.*(lp-1+dt[.,1]);
```
(3.29)

Thus, the vector d1v contains the derivatives of each of the individual loglikelihood terms. To obtain the information matrix `h` and the derivatives of the loglikelihood function `dll`, we merely sum:

```
h=moment(d1v,1);
dll=sumc(d1v);
```
(3.30)

The moment command makes use of the symmetry of the matrix `h=d1v'd1v`, using only $n(n+1)/2$ inner products instead of the n^2 products required by calculating `h=d1v'd1v` directly.

This concludes the description of the nested fixed point algorithm. If you want more detail about the specific operation of `NFXP.GPR`, edit or print out the source code for NFXP and its procedures using the list of variables in appendix 3 to assist you in following the variable mnemonics.

4. Documentation of the Data Set

4.1 Overview

The original "raw" bus data supplied by Madison Metropolitan Bus Company reside in 8 ascii files in the `nfxp/dat` subdirectory. Each file, such as `a530875.asc` contains monthly data on the odometer readings of all buses of a given make, model, and vintage (in this case, 1975 GMC model 5308 buses). This chapter provides background information about the data contained in these files, and explains the format in which the data are stored. I also briefly describe the program `STORDAT.GPR` that discretizes the raw data to conform to the fixed point dimension n selected by the user.

4.2 Background on the data set

Thanks to the cooperation of Harold Zurcher, maintenance manager of the Madison (Wisconsin) Metropolitan Bus Company, I was able to acquire maintenance records for a sample of 162 buses in the fleet of Madison Metro over the period December, 1974 to May, 1985. The data consists of monthly observations on the odometer readings for each bus, plus data on the date and odometer readings at which maintenance was performed on the bus. The company'smaintenance records are fairly detailed, indicating the dates and mileages at which each major component of the bus was serviced. From these maintenance records I extracted the dates and mileages at which engine replacements occurred. I define an engine replacement to be either 1) an actual physical replacement of the existing engine with either a new engine or a used engine which has been rebuilt in the company shop, or 2) an overhaul of the existing engine involving replacing or rebuilding a majority of the components in the engine. Due to the time required to overhaul and rebuild bus engines, the majority of the engine replacements were of the first kind, involving actual physical replacement of the bus engine.

In some cases the data on the date and odometer reading at which engine replacement occurred did not match the date and odometer reading of the independently recorded bus mileage sheets. In such a case I assumed that the odometer reading at which replacement occurred was correct, and changed the date of the overhaul to match the dates recorded on the bus mileage sheets. These inconsistencies occurred in about 9 or 10 cases and the change in date required to insure consistency was never more than one or two months. The odometer readings represent the exact monthly sequence with the exception of atwo month gap during July and August 1980 when the

company suffered a strike. Since no buses operated during the strike, I simply deleted these two months of unchanged odometer values from the data set.

Some summary information about the 162 buses included in the sample is presented in my 1987 *Econometrica* article. For each bus group listed in table 1 of that paper there is a corresponding GAUSS data file whose name is a mnemonic based on the model number of each bus group. Thus, the file `g870.asc` contains the data for the 15 Grumman model 870 buses, the file `rt50.asc` corresponds to the Chance RT-50 buses, the file `t8h203.asc` corresponds to the GMC model T8H203 buses, and so on. Individual buses in each group are identified by a four digit bus number. For example, each of the 15 Grumman 870 buses are numbered consecutively from 4403 to 4417.

4.3 Format of the raw data files

Each of the 8 raw data files is a GAUSS matrix file. That is, each file is read into memory as a $T \times M$ matrix, where M is the number of buses in the file and T is the number of data records per bus. Thus, each column of the matrix file contains data for a single bus. The first eleven rows of the matrix are the file "header" that contains information on the bus number, its date of purchase, the dates and odometer readings of engine replacements, and the month and year of the first odometer observation. The remaining $T - 11$ rows of the matrix contain the consecutive monthly odometer readings for each bus (with the exception of a two month gap to account for the strike during July and August, 1980). Specifically, the header contains the following information

Entries of 0 for the date and odometer reading at replacement indicate that no replacement occurred. In the sample above, for example, the data indicate that bus number 5297 had an engine replaced in April, 1979, but it has not had a second engine replacement as of May, 1985. I made no provision for a third engine replacement in the data headers since no bus in my sample had more than two engine overhauls.

4.4 Discretizing the raw data

The program `CONFIG.GPR` converts the ASCII bus data files into GAUSS matrix files with a `FMT` file extension. Thus, the ASCII data file `a530872.asc` is converted to matrix form and stored as the file `a530872.fmt` in the `nfxp/dat` subdirectory. The program `STORDAT.GPR` discretizes the underlying continuous mileage data from the raw bus data contained in the `FMT` files, creating the binary GAUSS data file `BDT.DAT` (and its header file `BDT.DHT` in Dos/Windows versions of GAUSS). The data is discretized into a finite number of mileage ranges that correspond to the fixed

Row	Item	Sample Entries
1	bus number	5297
2	month purchased	8
3	year purchased	75
4	month of 1st engine replacement	4
5	year of 1st engine replacement	79
6	odometer at replacement	153400
7	month of 2nd replacement	0
8	year of 2nd replacement	0
9	odometer at replacement	0
10	month odometer data begins	9
11	year odometer data begins	75
12	odometer reading	2353
13	·	6299
14	·	10479
.	.	.
.	.	.

point dimension selected by the user. The program prompts you for a fixed point dimension `n` and an upper bound for odometer readings, and then computes the implied discrete mileage ranges. For example if `n=90` and the upper bound for odometer is chosen to be 450000, then each discrete mileage state represents a mileage interval of length 5,000. The output of t `STORDAT.GPR` is a binary GAUSS data file `BDT.DAT` which contains 3 columns of data used by the nested fixed point program `NFXP.GPR`. The first column is `dtc`, the binary dependent variable d_t^m which is 1 if a replacement occurred for bus m in month t, 0 otherwise. The second column contains `dtx`, the discretized value of x_t^m, mileage since last replacement for bus m at month t. The final column is `mil`, the discretized value of monthly mileage for each bus. Usually `mil` will take the values $\{0, 1, 2\}$, but if the state space is sufficiently coarse (i.e. small dimension `n`), then `mil` will only take the values $\{0, 1\}$. If the user decides to include the lagged control variable d_{t-1}^m as an explanatory variable in the cost function (as part of a specification test of assumption (CI)), then a fourth column is added to `BDT.DAT` and the first month of data for each bus is deleted.

Important: after running `STORDAT.GPR` you must run the program `SETUP.GPR` in order to tell `NFXP.GPR` the correct dimension `n` of the data in `BDT.DAT`. If you run `NFXP.GPR` with a dimension

`n` smaller than the dimension of the discretized data in BDT.DAT, the program will crash with an "INDEX OUT OF RANGE" error message. If you run `NFXP.GPR` with a larger dimension than the discretized data, the program may run without error, but the results will be incorrect. Similarly, you must run `SETUP.GPR` in order to tell `NFXP.GPR` whether or not to expect discretized values of monthly mileage in the set $\{0, 1, 2\}$, $\{0, 1\}$ and so forth. If the data contains the former but the program expects the latter, then `NFXP.GPR` will crash with an "INDEX OUT OF RANGE" error message. On the other hand if the discretized mileage data takes on only the two values $\{0, 1\}$ but `NFXP.GPR` expects values in $\{0, 1, 2\}$ then no error message will occur but the parameter estimates will be incorrect.

This concludes the description of the data set. For more detailed information about the operation of `SETUP.GPR` and `STORDAT.GPR`, consult the annotations to the source code for these programs in the `nfxp/src` subdirectory.

5. References

This section contains references to literature on numerical and functional analysis used to design the NFXP algorithm, as well as references to empirical applications of the NFXP method in other areas.

Aedo, M. (1989) “The Schooling Decision: A Dynamic Model” PhD Dissertation, Department of Economics University of Minnesota.

Akin, J.E. (1982) *Application and Implementation of Finite Element Methods* Academic Press.

Anselone, P.M. and R. Ansorge (1981) "A Unified Framework for the Discretization of Nonlinear Operator Equations"Numerical Functional Analysis and Optimization **4** 61–97.

Barrett, W.W. and P.J. Feinsilver (1981) “Inverses of Banded Matrices” *Linear Albegra and its Applications* **41** 111–130.

Berkovec, J. and S. Stern (1991) “Job Exit Behavior of Older Men” *Econometrica* **59-1** 189–210.

Berman, A. and R.J. Plemmons (1979) *Nonnegative Matrices in the Mathematical Sciences* Academic Press.

Berndt, E. Hausman, J. Hall, B. and R. Hall (1976) “Estimation and Inference in Nonlinear Structural Models” *Annals of Economic and Social Measurement* **3** 653–665.

Bertsekas, D. (1995) *Dynamic Programming and Stochastic Control* Academic Press.

Bojanczyk, A. (1984) “Optimal Asynchronous Newton Method for the Solution of Nonlinear Equations” *Journal of the ACM* **31-4** 792–803.

Brent, R.P. (1973) “Some Efficient Algorithms for Solving Systems of Nonlinear Equations” *SIAM Journal on Numerical Analysis* **10** 327–344.

Christensen, B.J. (1990) *Estimation of Dynamic Programming Models with Applications to Intertemporal Equilibrium Asset Pricing* manuscript, New York University (based on Ph.D. dissertation submitted to Cornell University).

Das, S. (1993) “A Micro Econometric Model of Capital Utilization and Retirement: The Case of the Cement Industry” *Review of Economic Studies*

Deaton, A. (1991) “Saving and Liquidity Constraints” *Econometrica*

Gohberg, I. Lancaster, P. and L. Rodman (1982) *Matrix Polynomials* Academic Press.

Gotz, G. and J.J. McCall (1984) “A Dynamic Retention Model for Air Force Officers” Report R-3028-AF, the RAND Corporation, Santa Monica, California.

Harrod, W.J. and R.J. Plemmons (1984) “Comparision of Some Direct Methods for Computing Stationary Distributions of Markov Chains” *SIAM Journal on Scientific and Statistical Computing* **5-2** 453–469.

Kennet, M. (1990) "Airline Deregulation and Aircraft Engine Maintenance: An Empirical Policy Analysis"manuscript, Tulane University.

Kulisch, U.W. Miranker, W.L. (1986) “The Arithmetic of the Digital Computer: A New Approach” *SIAM Review* **28-1** 1–40.

Lumsdaine, R., Stock, J. and Wise, D. (1990) “Three Models of Retirement: Computational Complexity vs. Predictive Validity” manuscript, Harvard University.

Martin, R.S. Wilkinson, J.H. (1967) "Solution of Symmetric and Unsymmetric band equations and the calculation of eigenvectors of band matrices"Numerische Mathematik 9, 279-301.

Montgomery, M. (1991) "A Dynamic Model of Contraceptive Choice"manuscript, Department of Economics, State University of New York at Stony Brook.

Montgomery, M. (1991) "Migration and Job Search in Malaysia: Estimates from Dynamic Models"manuscript, State University of New York at Stony Brook.

Ortega, J.M. and W.C. Rheinboldt (1970) *Iterative Solution of Nonlinear Equations in Several Variables* Academic Press.

Press, W.H. Teukolsky, S. Vetterling, W.T. and B.P. Flannery (1992) *Numerical Recipes in Fortran: The Art of Scientific Computing* Cambridge University Press.

Rheinboldt, W.C. (1986) *Numerical Analysis of Parameterized Nonlinear Equations* Wiley, New York.

Rice, J.R. (1983) *Numerical Methods, Software, and Analysis* McGraw-Hill, New York.

Rust, J. (1987) “Optimal Replacement of GMC Bus Engines: An Empirical Model of Harold Zurcher” *Econometrica* **55-5** 999–1033.

Rust, J. (1988) “Maximum Likelihood Estimation of Discrete Control Processes” *SIAM Journal on Control and Optimization* **26-5** 1006–1024.

Rust, J. (1989) “A Dynamic Programming Model of Retirement Behavior” in David Wise (ed.) *The Economics of Aging* Chicago, University of Chicago Press, 359–398.

Rust, J. (1990) “Behavior of Male Workers at the End of the Life-Cycle: An Empirical Analysis of States and Controls” in David Wise (ed.) *Issues in the Economics of Aging* Chicago, University of Chicago Press.

Rust, J. (1994a) “Estimation of Dynamic Structural Models: Problems and Prospects” chapter 4 in C. Sims (ed.) *Advances in Econometrics: Proceedings of the Sixth World Congress Vol II.* 119–170, Cambridge University Press. Rust, J. (1994b) “Structural Estimation of Markov Decision Processes” in R. Engle and D. McFadden (eds.) *Handbook of Econometrics* Volume 4, 3082–3139, Elsevier, Amsterdam.

Rust, J. (1997) “Using Randomization to Break the Curse of Dimensionality” *Econometrica*

Rust, J. (1996) “Numerical Dynamic Programming in Economics” in H. Amman, D. Kendrick and J. Rust (eds.) *Handbook of Computational Economics* Elsevier, Amsterdam.

Rust, J. and G. Rothwell (1995) “Optimal Response to Shift in Regulatory Regime: The Case of the U.S. Nuclear Power Industry” *Journal of Applied Econometrics.*

Seneta, E. (1973) *Non-Negative Matrices* Wiley, New Jersey. Steinberg, D.I. (1974) *Computational Matrix Algebra* McGraw-Hill, New York.

Sturm, Roland (1990) "A Structural Economic Model of Operating Cycle Management in European Nuclear Power Plants"manuscript, Department of Economics, Stanford University.

Traub, J.F. and H. Wozniakowski (1984) “On the Optimal Solution of Large Linear Systems” *Journal of the ACM* **31-3** 545–555.

Van Der Klaauw, Wilbert (1990) "Female Labor Supply and Marital Status Decisions: A Life-Cycle Model", manuscript, Department of Economics, Brown University.

van der Wal, J. (1980) *Stochastic Dynamic Programming: Successive Approximations and Nearly Optimal Strategies for Markov Decision Processes and Markov Games* Mathematical Center Tract 139, Mathematisch Centrum, Amsterdam.

Werner, W . (1984) “Newton-Like Methods for the Computation of Fixed Points” *Computational Mathematics with Applications* **10-1** 77–86.

Zijm, W.H.M. (1980) *Nonnegative Matrices in Dynamic Programming* CWI Tract 167, Mathematische Centrum, Amsterdam.