

# Documentation of the linear programming code used in “Firms as Clubs in Walrasian Markets with Private Information”

Edward Simpson Prescott\*

Federal Reserve Bank of Richmond

Robert M. Townsend\*

University of Chicago

Federal Reserve Bank of Chicago

July 11, 2007

## Abstract

This note documents the linear programming code used in “Firms as Clubs in Walrasian Markets with Private Information.” It spells out the linear program and then describes the structure of the code.

## 1 Introduction

This note describes the algorithm and the computer code used to solve the examples in Prescott and Townsend (2006). The examples report Pareto optimal allocation and prices and incomes that support them as competitive equilibria. The Pareto optimum are found by solving a Pareto program, which is a linear program. Prices and incomes were then determined from the allocation and the Lagrangian multipliers.

In principle, this is a straightforward exercise. In our club economies with private information, however, the first step – solving the Pareto program – is difficult. For any

---

\*The views expressed in this paper are solely those of the authors and do not necessarily reflect the views of the Federal Reserve Banks of Chicago and Richmond or the Federal Reserve System. Contact information: Prescott, Federal Reserve Bank of Richmond, P.O. Box 27622, Richmond, VA 23261, Edward.Prescott@rich.frb.org; Townsend, Dept. of Economics, University of Chicago, 1126 E. 59th St., Chicago, IL 60637, rtownsen@midway.uchicago.edu.

reasonable application, a large number of possible contracts is needed. In our economy, each possible contract is a different commodity, so there are a huge number of variables and constraints in the linear program. The appendix to Prescott and Townsend (2006) reformulates the Pareto program into an alternative formulation, which removes the large number of club constraints. However, this program still has an enormous number of variables, too large to directly enter into computer memory.

Instead, the program is solved by using the Dantzig-Wolfe decomposition algorithm. This algorithm is a simplex-based algorithm that can be used to solve linear programs with a block angular constraint matrix, that is, a constraint matrix with blocks of variables and constraints with non-zero coefficients that are only connected by a few constraints. The algorithm was first developed by Dantzig and Wolfe (1960). Descriptions of it can be found by in many advanced linear programming textbooks like Bertsimas and Tsitsiklis (1997). Prescott (2004) uses this algorithm to solve moral-hazard programs.

This document gives a short summary of the code. For a description of the algorithm, as it is applied to club economies, see the appendix in Prescott and Townsend (2006).

## 2 The Code

The code is written in a programming language called GAMS. Programming languages like GAMS are well suited to solving linear programs. The programmer only needs to program the parameters. The programming language handles the interface with the linear programming subroutine `solver`.<sup>1</sup> In this problem, we used the BDMLP linear program solver that comes with GAMS.

The code is divided into five parts.

1. Define the parameters.
2. Define the subprograms.

---

<sup>1</sup>The code was originally programmed in Matlab for a different example than the one reported in the final version of the paper. This code is available on request. It is longer and not so well documented. Furthermore, it called a linear programming routine that we no longer use. A user of this code would have to get hold of a linear programming routine callable from Matlab and then modify the code to enter the coefficients in the form needed by the routine.

3. Define the master program.
4. Run the algorithm.
5. Calculate prices and incomes and write results to a text file.

The first step in the algorithm is to find a feasible starting point. Simplex based algorithms find a feasible starting point by solving a particular auxiliary linear program. This step is commonly called Phase one of the algorithm. Given all the data, most linear programming routines automatically find a starting point, but because we cannot specify the whole problem – remember, we only generate columns of the master program with each iteration – we have to use the Dantzig-Wolfe algorithm to solve Phase one. See Bertsimas and Tsitsiklis (1999) for more details on this step.

Once a feasible solution is found, the algorithm searches for an optimum. This is commonly called Phase two of the simplex algorithm. We also solve this step with the Dantzig-Wolfe algorithm. Given a feasible solution to the master program, the dual variables are calculated, then these variables are used to solve the subprograms. The solutions to the subprograms are then used to check the optimality conditions and entered into the master program if needed. See the appendix of Prescott and Townsend (2006) for more details.

Once a solution to the master program is found, the code solves the appropriate subprograms to calculate the solution. It then calculates prices and incomes, and writes all the results to a text file.

Finally, when we define the subprograms and the master program, we define two versions of each program. One version is used when solving Phase one and the other version is used when solving Phase two of the algorithm. Part 4 consists of one single loop. The same loop is used for both phases of the algorithm. If-then statements are used to keep track of whether the Phase one or Phase two version of a program needs to be called. More details are contained in the code.

### 3 References

#### References

- [1] Bertsimas, Dimitris and John N. Tsitsiklis. *Introduction to Linear Optimization*. Belmont, Massachusetts: Athena Scientific, 1997.
- [2] Dantzig, George B. and Philip Wolfe. “The Decomposition Principle for Linear Programs.” *Econometrica* 29 (October 1961): 767-78.
- [3] Prescott, Edward Simpson “Computing Solutions to Moral-Hazard Programs Using the Dantzig-Wolfe Decomposition Algorithm.” *Journal of Economic Dynamics and Control* 28 (January 2004): 777-800
- [4] Prescott, Edward Simpson and Robert M. Townsend. “Firms as Clubs in Walrasian Markets with Private Information.” *Journal of Political Economy* 114 (August 2006): 644-71.

```

1 $title Example in paper: Solved with D-W
2 *****
3 * This program solves for a Pareto optimum and then finds prices and
4 * incomes that support it as a competitive equilibrium. The example
5 * solved is the one used in the paper, "Firms as Clubs in Walrasian
6 * Markets with Private Information." The program uses the Dantzig-Wolfe
7 * algorithm to solve the Alternative Pareto Program in the Appendix. A
8 * supporting competitive equilibrium is then calculated.
9 *
10 * Note 1: The program is setup to loop over a range of Pareto weights and
11 * capital levels. What is shown below is the single run for the
12 * pareto weight and capital level combination used in the example
13 * reported in the paper. The bigger loops are commented out. If these
14 * are to be implemented changes need to be made to numbers of values
15 * in the sets ik and ipw. (The indices for capital levels and Pareto
16 * weights.)
17 *
18 * Note 2: the code is written to only handle case where agents are not
19 * intrinsically different. They can differ in their Pareto weights. Also,
20 * it is written for only two types. Would require some modification
21 * for more than two types.
22 *
23 * Note 3: Idle firms are lumped together with self-employment firms
24 * by solving one subprogram. They are differentiated by using
25 * constraints. For example, the idle firms are handled by requiring
26 * that if k=0 then a=0 (and vice versa) and dropping the incentive
27 * constraints in this case. Similarly, self-employment firms are handled
28 * by requiring that if k>0 then a>0, and imposing the incentive
29 * constraints.
30 *
31 *
32 *
33 *
34 * Note 4: This code could be substantially improved by keeping track
35 * of subprogram results and then using these results in other loops.
36 *
37 *****
38
39 * PART I - DEFINE THE PARAMETERS
40 * NOTE: thi has to be defined first because it starts with 0, we have
41 * lots of other ordered sets that start with 1, and we want thi to be
42 * an ordered set.
43 set      thi      position for the supervisor /0*2/;
44 set      c         consumption /1*61/
45 set      q         output /1*2/
46 set      a         actions /1*3/
47 set      k         capital input /1*3/
48 set      job      job /1*2/
49 set      th       agent types /1*2/
50 set      ik       index capital endowment levels /1/
51 set      ipw      index Pareto weights /1/ ;
52 *
53 *      ik         index capital endowment levels /1*21/
54 *      ipw       index Pareto weights /1*17/ ;
55 * Define an index of the different firm-agent combinations
56 * Use (th,thi) combinations to index firms.
57 * The SE firms are (1,0) and (2,0).
58 * The SW firms are (1,1), (1,2), (2,1), and (2,2).
59 set      se(th,thi) set of SE firms /((1*2).0)/
60 set      sw(th,thi) set of SW firms /((1*2).(1*2))/
61 set      ds1(th,thi) firms with type-1 as sup /((1*2).1)/
62 set      ds2(th,thi) firms with type-2 as sup /((1*2).2)/;
63
64 alias (c,cw,cs), (q,qbar), (a,abar,ahat), (k,kbar), (th,thp,thr);

```

```

65 parameter      c0(c)      Consumption
66 parameter      q0(q)      Output
67 parameter      a0(a)      Action
68 parameter      k0(k)      Capital input
69 parameter      u(c,a,job,k) Utility function
70 parameter      dis(job)   Disutility parameter on job
71 parameter      p(q,a,k)   Technology
72 parameter      rc1(c,q)   resource usage by self-employment firm
73 parameter      rc2(cw,cs,q) resource usage by supervisor-worker firm
74 parameter      kappa      capital endowment for given loop
75 parameter      kap(ik)    Capital endowment levels
76 parameter      ParW(ipw)  Type-1 Pareto weight
77 parameter      lam(th)    Pareto weights for a given loop
78 parameter      alp(th)    Each type's fraction of the population;
79
80 * Set the grids, technology, and resource level.
81 c0(c) = (1.2/(card(c)-1))*(ord(c)-1);
82 q0("1") = 0; q0("2") = 1;
83 a0("1") = 0; a0("2") = 1; a0("3") = 2;
84 k0(k) = ord(k)-1;
85 * Set probabilities for the zero capital and zero effort cases
86 p("1",a,"1") = 1;      p("2",a,"1") = 0;
87 p("1","1",k) = 1;      p("2","1",k) = 0;
88 * Set probabilities for the other cases
89 p("1","2","2") = 0.80; p("2","2","2") = 0.20;
90 p("1","3","2") = 0.50; p("2","3","2") = 0.50;
91 p("1","2","3") = 0.60; p("2","2","3") = 0.40;
92 p("1","3","3") = 0.20; p("2","3","3") = 0.80;
93 * Set fractions of population
94 alp("1") = 0.5; alp("2") = 0.5;
95 * Set capital levels for loop
96 * Creates an equally spaced grid over the range [0.2,1.2]
97 * kap(ik) = (1/(card(ik)-1))*(ord(ik)-1)+ 0.2;
98 kap("1") = 0.6;
99
100 * Set Pareto weights for loop
101 * Creates and equally spaced grid over the range [0.01,0.49]
102 * ParW(ipw) = (.48/(card(ipw)-1))*(ord(ipw)-1)+ 0.01;
103 ParW("1") = 0.16;
104
105 * Define utility and resource usage in terms of the grids.
106 * job = 1 corresponds to worker, job = 2 corresponds to supervisor
107 dis("1") = 1; dis("2") = 0.1;
108 u(c,a,job,k) = 2*c0(c)**0.5 - dis(job)*((a0(a)/4));
109 rc1(c,q) = c0(c) - q0(q);
110 rc2(cw,cs,q) = c0(cw) + c0(cs) - q0(q);
111
112 * Open the output file for the solutions set some parameters that
113 * control the display of output.
114 file mfile /FinalLoop.out/;
115 mfile.pc = 0;
116 mfile.ps = 120;
117 * Open an output file for writing data about the type of
118 * supervisor-worker firms that are created.
119 file mfile2 /FinalLoop.m/;
120
121 *****
122 * The following parameters are updated by the master program. Because
123 * they are used by all of the subprograms they are defined first.
124
125 scalar      lamw      worker's Pareto weight
126 scalar      lams      supervisor's Pareto weight
127 scalar      muc       multiplier on consumption resource constraint
128 scalar      muk       multiplier on capital resource constraint

```

```

129          muw      multiplier on worker's (or SE) meas. constraint
130          mus      multiplier on supervisor's measure constraint;
131 parameter      mum(th)  multipliers on agent's measure constraint;
132
133 *****
134 * PART II - DEFINE THE SUBPROBLEMS
135 * Setup the self-employment subproblem. The scalar variables above are
136 * updated by the algorithm.
137 positive variables      pil(c,q,a,k)  Probability;
138
139 * These variables are useful because they are generate numbers needed
140 * by the master program. The 1 at the end means that this problem refers
141 * to the single agent subproblem.
142 variables      utils1      Agents utility
143                resc1      Consumption resource usage
144                resk1      Capital resource usage
145                obj1      Value of subproblem objective function;
146
147 equations      sublobj, sublutils, sublresc, sublresk, subltech,
148                sublic, sublmeas, sublobjph1, sublk1, sublal;
149
150 sublobjph1..  obj1  =e= -muc*resc1-muk*resk1-muw;
151 sublobj..    obj1  =e= lamw*utils1 - muc*resc1-muk*resk1-muw;
152 sublutils..  utils1 =e= sum((c,q,a,k), pil(c,q,a,k) * u(c,a,"1",k));
153 sublresc..   resc1  =e= sum((c,q,a,k), pil(c,q,a,k) * rcl(c,q));
154 sublresk..   resk1  =e= sum((c,q,a,k), pil(c,q,a,k) * k0(k));
155 subltech(qbar,abar,kbar)..
156          sum(c, pil(c,qbar,abar,kbar))
157          =e= p(qbar,abar,kbar) * sum((c,q), pil(c,q,abar,kbar));
158 * No IC for k=0 and a=0. (Idle "firms"). For SE firms, IC only
159 * allow deviations to non-zero a.
160 sublic(a,ahat,k)$(not sameas(a,ahat) and k0(k) ne 0 and a0(a) ne 0
161          and a0(ahat) ne 0)..
162          sum((c,q), pil(c,q,a,k) * u(c,a,"1",k)) =g=
163          sum((c,q), pil(c,q,a,k) * p(q,ahat,k)/p(q,a,k) * u(c,ahat,"1",k));
164 sublmeas..   sum((c,q,a,k), pil(c,q,a,k)) =e= 1;
165 * Constraints that guarantee that a=0 if k=0 and vice versa
166 sublk1..    sum((c,q,k)$(k0(k) ne 0), pil(c,q,"1",k)) =e= 0;
167 sublal..    sum((c,q,a)$(a0(a) ne 0), pil(c,q,a,"1")) =e= 0;
168
169 model sub1ph1 /sublobjph1, sublutils, sublresc, sublresk, subltech,
170          sublic, sublmeas, sublk1, sublal/;
171 model sub1 /sublobj, sublutils, sublresc, sublresk, subltech,
172          sublic, sublmeas, sublk1, sublal/;
173
174 *****
175 * Setup the supervisor-worker subproblem. The scalar variables are
176 * updated by the algorithm.
177 * NOTE: pi2 is only indexed by one effort level. In the paper, we
178 * constrain worker's and supervisor's effort to be equal. Rather than
179 * distinguishing between their efforts and putting a constraint that
180 * requires them to be equal, only one effort level is written and it is
181 * applied to both agents. The generalization is easy to do though it
182 * makes the equations longer.
183
184 positive variables      pi2(cw,cs,q,a,k)  Probability;
185
186 * These variables are useful because they generate numbers needed
187 * by the master program. The 2 at the end means that this problem refers
188 * to the supervisor-worker subproblem.
189 variables      utilsw2      Worker's utility
190                utilss2      Supervisor's utility
191                resc2      Consumption resource usage
192                resk2      Capital resource usage
193
194          obj2      Value of subproblem objective function;
195
196 equations      sub2obj, sub2utilsw, sub2utilss, sub2resc, sub2resk,
197                sub2tech, sub2meas, sub2objph1, sub2k1, sub2al;
198
199 sub2objph1..  obj2  =e= -muc*resc2-muk*resk2-muw-mus;
200 sub2obj..    obj2  =e= lamw*utilsw2+lams*utilss2
201                - muc*resc2-muk*resk2-muw-mus;
202 sub2utilsw..  utilsw2 =e= sum((cw,cs,q,a,k),
203                pi2(cw,cs,q,a,k) * u(cw,a,"1",k));
204 sub2utilss..  utilss2 =e= sum((cw,cs,q,a,k),
205                pi2(cw,cs,q,a,k) * u(cs,a,"2",k));
206 sub2resc..    resc2  =e= sum((cw,cs,q,a,k),
207                pi2(cw,cs,q,a,k) * rc2(cw,cs,q));
208 sub2resk..    resk2  =e= sum((cw,cs,q,a,k),
209                pi2(cw,cs,q,a,k) * k0(k));
210 sub2tech(qbar,abar,kbar)..
211          sum((cw,cs), pi2(cw,cs,qbar,abar,kbar))
212          =e= p(qbar,abar,kbar)*sum((cw,cs,q), pi2(cw,cs,q,abar,kbar));
213 sub2meas..    sum((cw,cs,q,a,k), pi2(cw,cs,q,a,k)) =e= 1;
214 * A simple way to guarantee that supervisor-worker firms can't choose
215 * zero capital or zero effort.
216 sub2k1..      sum((cw,cs,q,a), pi2(cw,cs,q,a,"1")) =e= 0;
217 sub2al..      sum((cw,cs,q,k), pi2(cw,cs,q,"1",k)) =e= 0;
218
219 model sub2ph1 /sub2objph1, sub2utilsw, sub2utilss, sub2resc, sub2resk,
220          sub2tech, sub2meas, sub2k1, sub2al/;
221 model sub2 /sub2obj, sub2utilsw, sub2utilss, sub2resc, sub2resk,
222          sub2tech, sub2meas, sub2k1, sub2al/;
223
224 *****
225 * PART III - DEFINE THE MASTER PROGRAM
226 *****Master program*****
227 * slab labels the iterations for the master program. The columns are
228 * indexed by the cross-product of the subproblem, (th,thi), with slab.
229
230 sets slab      master program iterations /1*1000/
231          s(th,thi,slab)  generated columns from subproblems;
232
233 * s(th,thi,slab) is a dynamic set. As master program iterations are run,
234 * we will make new elements of it active. The (th,thi) pairs cover the
235 * combinations (1,0),(2,0),(1,1),(1,2),(2,1),(2,2). Each corresponds to
236 * a subproblem: (1,0) refers to an agent 1 self-employment firm,
237 * while (1,2) refers to a sup-worker firm with agent 1 as the worker and
238 * agent 2 as the supervisor.
239
240 * To start the algorithm, make all the columns inactive.
241 s(th,thi,slab) = no;
242
243 * mutils() will equal lam(th)u(b1,w) for a SE firm and it will
244 * equal lam(th)u(b2,w)+lam(thi)u(b2,s) for a SW firm.
245 parameter      mutils(th,thi,slab)  contrib to obj from subprob
246                mresc(th,thi,slab)    net cons. usage by subprob
247                mresk(th,thi,slab)    capital usage by subprob
248                meas(thp,th,thi,slab)  contrib. to measure constr.;
249
250 * The probability measure coefficients are defined here
251 * This is inelegant. Should be a better way to do this.
252 parameter      th0(th)
253                th1(thi);
254 th0(th) = ord(th); th1("0") = 0; th1("1") = 1; th1("2") = 2;
255 meas(thp,th,thi,slab)$(th0(thp)=th0(th) and th0(thp) ne th1(thi)) = 1;
256 * Pick up SE firms and SW firms with different types
257 * where thp is the worker
258 * Pick up SW firms with different types where thp is the supervisor

```

```

257 meas(thp,th,thi,slab)$(th0(thp) ne th0(th) and th0(thp)=th1(thi)) = 1;
258 * Pick up SW firms where thp is both the worker and supervisor.
259 * Counted twice.
260 meas(thp,th,thi,slab)$(th0(thp) = th0(th) and th0(thp)=th1(thi)) = 2;
261
262 * Create a parameter that keeps track of the simplex multipliers used
263 * to solve the subprograms.
264 parameter mnc(slab) Consumption multiplier
265 mmk(slab) Capital multiplier
266 mm1(slab) Agent 1's prob measure constraint
267 mm2(slab) Agent 2's prob measure constraint
268 aux(slab) 1 means aux lp while 2 means reg lp;
269
270 positive variables mprob(th,thi,slab)
271 excess(thp) used for phase 1
272 slc1 slack var on cons. res. constraint
273 slk1 slack var on cap. res. constraint;
274 variables mobj;
275
276 * Variables that end with ph1 are only used when solving Phase I of
277 * the algorithm.
278
279 equations masobj objective function
280 masresc net consumption resource constraint
281 masresk capital resource constraint
282 masmeas(thp) probability measure constraints
283 masobjph1 Phase 1 objective function
284 masmeasph1(thp) Phase 1 probability measure constraints»
;
285
286 masobj.. mobj =e= sum(s,mprob(s)*mutils(s));
287 masobjph1.. mobj =e= sum(thp,-excess(thp));
288 masresc.. sum(s, mprob(s)*mresc(s)) + slc1 =e= 0;
289 masresk.. sum(s, mprob(s)*mresk(s)) + slk1 =e= kappa;
290 masmeas(thp).. sum(s, mprob(s)*meas(thp,s)) =e= alp(thp);
291 masmeasph1(thp).. sum(s,mprob(s)*meas(thp,s))+excess(thp) =e= alp(thp)»
;
292
293 * Auxiliary master lp used in Phase I to find a feasible starting point.
294 model masterph1 /masobjph1, masresc, masresk, masmeasph1/;
295 * Master lp used in Phase II to find a solution.
296 model master /masobj, masresc, masresk, masmeas/;
297
298 * Set print and workspace options on the all the lp's
299 * Used the bdm1p solver
300 option lp = bdm1p;
301 option limcol=0, limrow=0, solprint= off, sysout=off;
302 * The bratio option makes sure that GAMS doesn't use information
303 * from previous solves in generating a basis. This is important
304 * because subprograms that are part of the solution to the master
305 * program are resolved to generate the optimal contracts. By not
306 * using information from previous solves, each lp will reach the
307 * same solution if resolved. This prevents the possibility of a
308 * subprogram having multiple solutions and the code generating
309 * a master program column with one solution and then calculating
310 * prices, incomes, and optimal contracts from a different solution.
311 * The gap variable is a partial check for this.
312 option bratio = 1;
313 master.bratio = 1;
314 sub1.bratio = 1; sub1ph1.bratio = 1;
315 sub2.bratio = 1; sub2ph1.bratio = 1;
316
317 *****
318 * PART IV - RUN THE DANTZIG-WOLFE ALGORITHM

```

```

319 *
320 * The simplex algorithm needs a feasible solution to start, so the first
321 * thing to do is solve Phase one of the simplex algorithm. This is done
322 * by using the Dantzig-Wolfe algorithm to solve the auxiliary linear
323 * program. However, the auxiliary linear program also needs a feasible
324 * solution to start. Given the way we set the Phase 1 problem up, this is
325 * easy. The auxiliary master program will just choose non-zero values of
326 * the slack variables.
327
328 * As long as the master program has a feasible solution, the solution to
329 * the auxiliary lp will satisfy excess(thp)=0 and it will be a feasible
330 * solution to the master lp. We can then initiate D-W with that feasible
331 * solution.
332 *
333 * Both phase one and phase two are solved using the same while loop. When
334 * different operations are used by the two phases, if-then statements are
335 * used to perform the right operation. The parameter phase = 1 means
336 * phase one is being run, while phase = 2 means phase two is being run.
337 *
338 *****
339
340 * Do some necessary declarations
341 * submax is the the maximum number of loops allowed
342 * (needed in case of cycling or really slow convergence)
343 * done=0 means the while loop is not done, =1 means done
344 * phase indicates which phase the algorithm is in
345 scalar submax /1000/
346 done /0/
347 phase /1/;
348 parameter presc multiplier on net consumption
349 presk multiplier on capital
350 pmc(th) multipliers on prob. measure constraints;
351
352 * Set some values that will be used for calculating things used in
353 * creating the output
354 set j /1*4/;
355 parameter prices(thr,thi,slab,job) keep track of prices
356 voffirm(thr,thi,slab) value of a firm
357 sumsw(ipw,ik,j) keeps track of sw firms for each run»
;
358 sumsw(ipw,ik,j)=0;
359 scalar zs
360 pk;
361 parameter expenditures(th) total expenditures by an agent type
362 exper(th) per capital expenditures;
363 parameter autill type-1 agents' utility
364 autill2 type-2 agents' utility
365 count1 useful counter
366 count2 another useful counter
367 gap;
368 loop(ipw,
369 lam("1") = ParW(ipw); lam("2") = 1-lam("1");
370 loop(ik,
371 kappa = kap(ik);
372 * reset the dynamic sets to restart the algorithm
373 s(th,thi,slab) = no;
374 * Need to reset probabilities to zero for each run, because GAMS
375 * does not reset values of this variable.
376 mprob.l(th,thi,slab) = 0;
377 * This first loop finds a starting point for the auxiliary lp. It
378 * generates an arbitrary column for the master program from the
379 * subprograms. By putting a column in the master constraint matrix
380 * the auxiliary lp has some data from which it can always get an
381 * initial feasible solution. With an arbitrary column, there will

```

```

382 * always be a solution that puts zero weight on the column and sets
383 * the slack variables equal to the values of the constraints. This
384 * suggests we could skip the column but GAMS did not like having no
385 * values in some of the master auxiliary lp (e.g. mresc or mresk).
386
387 * Pick some arbitrary simplex multiplier values.
388   muw = 1; mus = 1; muc = 1; muk = 1;
389   loop((thi,thr),
390 *     Check to see if it is a single-agent subproblem
391       if (se(thr,thi),
392           lamw = lam(thr);
393           solve sublph1 using lp maximizing obj1;
394           abort$(sublph1.modelstat=4) "SE subproblem infeasible";
395           abort$(sublph1.modelstat<>1) "SE subproblem not solved to opti>
mum";
396 *     Add the solution to the master program
397       mutils(thr,thi,"1") = lamw*utils1.l;
398       mresc(thr,thi,"1") = resc1.l; mresk(thr,thi,"1") = resk1.l;
399 *     If it is a supervisor-worker subproblem
400       else
401 *         Set parameters for worker position
402         lamw = lam(thr);
403 *         Set parameters for supervisor position (there should be a bette>
r
404 *         way to do this). First, check to see if type-1 is the sup.
405         if (dsl(thr,thi),
406             lams = lam("1");
407         else
408             lams = lam("2");
409         );
410         solve sub2ph1 using lp maximizing obj2;
411         abort$(sub2ph1.modelstat=4) "SW subproblem infeasible";
412         abort$(sub2ph1.modelstat<>1) "SW subproblem not solved to optim>
um";
413 *     Add the solution to the master program
414       mutils(thr,thi,"1") = lamw*utilsw2.l+lams*utilss2.l;
415       mresc(thr,thi,"1") = resc2.l; mresk(thr,thi,"1") = resk2.l;
416       );
417 *     Make the column of the master program active
418       s(thr,thi,"1") = yes;
419       );
420       mmc("1") = muc; mmk("1") = muk;
421       mml("1") = muw; mm2("1") = mus;
422       aux("1") = 1;
423
424 *****
425 *****
426 * Start the D-W loop
427 * First, set some parameter values that control the loop.
428   done = 0; phase = 1;
429
430   loop(slab$(ord(slab) ne 1 and not done),
431
432       if (phase=1,
433           solve masterph1 using lp maximizing mobj;
434           abort$(masterph1.modelstat=4) "Auxiliary lp infeasible";
435           abort$(masterph1.modelstat<>1) "Auxiliary lp not solved to optimu>
m";
436           mum(th) = masmeasph1.m(th);
437           aux(slab) = 1;
438       else
439           solve master using lp maximizing mobj;
440           abort$(master.modelstat=4) "Master lp infeasible";
441           abort$(master.modelstat<>1) "Master lp not solved to optimum";
442
443           mum(th) = masmeas.m(th);
444           aux(slab) = 2;
445       );
446   muc = masresc.m; muk = masresk.m;
447 *   Store the simplex multipliers so they can be recovered later
448   mmc(slab) = muc; mmk(slab) = muk;
449   mml(slab) = mum("1"); mm2(slab) = mum("2");
450   submax = 0;
451
452 *   Now solve the subproblems. Loop over all the different types of
453 *   firms, i.e., (1,0), (2,0), (1,1), (1,2), (2,1), and (2,2).
454   loop((thi,thr),
455 *     If it is a self-employment subproblem
456       if (se(thr,thi),
457           lamw = lam(thr); muw = mum(thr);
458           if (phase=1,
459               solve subl using lp maximizing obj1;
460               abort$(subl.modelstat=4) "SE subproblem infeasible";
461               abort$(subl.modelstat<>1) "SE subproblem not solved to op>
timum";
462           else
463               solve subl using lp maximizing obj1;
464               abort$(subl.modelstat=4) "SE subproblem infeasible";
465               abort$(subl.modelstat<>1) "SE subproblem not solved to optim>
um";
466           );
467 *     Add the solution to the master program
468       mutils(thr,thi,slab) = lamw*utils1.l;
469       mresc(thr,thi,slab) = resc1.l; mresk(thr,thi,slab) = resk1.l;
470       submax = max(submax,obj1.l);
471 *     If it is a supervisor-worker subproblem
472       else
473           Set parameters for worker position
474           lamw = lam(thr); muw = mum(thr);
475 *           Set parameters for supervisor position (there should be a bette>
r
476 *           way to do this). First, check to see if type-1 is the sup.
477           if (dsl(thr,thi),
478               lams = lam("1"); mus = mum("1");
479           else
480               lams = lam("2"); mus = mum("2");
481           );
482           if (phase=1,
483               solve sub2ph1 using lp maximizing obj2;
484               abort$(sub2ph1.modelstat=4) "SW subproblem infeasible";
485               abort$(sub2ph1.modelstat<>1) "SW subproblem not solved to opt>
imum";
486           else
487               solve sub2 using lp maximizing obj2;
488               abort$(sub2.modelstat=4) "SW subproblem infeasible";
489               abort$(sub2.modelstat<>1) "SW subproblem not solved to optimu>
m";
490           );
491 *     Add the solution to the master program
492       mutils(thr,thi,slab) = lamw*utilsw2.l+lams*utilss2.l;
493       mresc(thr,thi,slab) = resc2.l; mresk(thr,thi,slab) = resk2.l;
494       submax = max(submax,obj2.l);
495       );
496       s(thr,thi,slab) = yes;
497       );
498 *   submax is the highest value of the optimality conditions
499   if ((phase = 1 and submax < 0.00001),
500       phase = 2; submax = 10000;

```

```

501      abort$(mobj.l>.00001) "Value of auxiliary LP positive, so LP infeasible";
502      display "Auxiliary LP solved";
503      display mobj.l;
504      display slcl.l;
505      display slkl.l;
506      elseif (phase = 2 and submax < 0.00001),
507      display "Master LP solved";
508      done = 1;
509      );
510      display submax;
511      abort$(ord(slab)=card(slab)) "D-W algorithm did not converge"
512 );
513
514 * Store the values of the multipliers to the master problem, so they can
515 * be used to calculate prices. Do not index by
516 * slab because we are only interested in these variables at the optimum.
517 presk = masresk.m; presc = masresc.m;
518 pmc(th) = masmeas.m(th);
519
520 *****
521 * PART V- WRITE SOLUTION TO TEXT FILE
522 *
523 * Generate solution by solving appropriate subprograms
524 * and write solution to a text file.
525
526 put mfile;
527 put "PARETO OPTIMUM AND SUPPORTING PRICES AND INCOMES":<>80 //;
528 put "Values of parameters" //;
529 put "-----" //;
530 put "Aggregate capital endowment:      ", kappa:<4:2 //;
531 put "Fraction of agents that are type-1: ", alp("1"):<4:2 //;
532 put "Fraction of agents that are type-2: ", alp("2"):<4:2 //;
533 put "Pareto weight on type-1 agents:   ", lam("1"):<4:2 //;
534 put "Pareto weight on type-2 agents:   ", lam("2"):<4:2 //;
535
536 put ///;
537 put "SOLUTION TO PARETO PROGRAM":<>54 /
538      "Reported as different basic feasible solutions":<>54 /
539      "-----":<>54 //;
540 put /;
541
542 count1 = 1; count2 = 1;
543 * Total utility of each agent is calculated by determining the
544 * contribution to an agent's total utility from each subproblem solution
545 * that is part of the master program solution. Consequently, we start the
546 * utilities at a value of zero and then sequentially add each
547 * contribution.
548 autill = 0; autil2 = 0;
549
550 * Now resolve appropriate subproblems to generate the solution and the
551 * write the solution to the output file.
552 autill = mm1("2"); display autill; autil2 = mm2("2"); display autil2;
553 autill = 0; autil2 = 0;
554 loop((thi,thr),
555     loop(slab$(mprob.l(thr,thi,slab)>0),
556         display$(aux(slab) = 1) "Solution includes point from aux lp";
557         Recover the values of the dual variables and Pareto weights
558         corresponding to slab.
559         lamw = lam(thr); muc = mmc(slab); muk = mmk(slab);
560         if ((ord(thr)=1), muw = mm1(slab);
561         else muw = mm2(slab); );
562         if ((thl(thi)=1), mus = mm1(slab); lams = lam("1");
563         elseif (thl(thi)=2), mus = mm2(slab); lams = lam("2"); );

```

```

564
565 * Check to see if is a SE club. If so, resolve the subproblem,
566 * write the solution to the output file, update utilities, and
567 * calculate prices.
568 if (se(thr,thi),
569     if (aux(slab)=1,
570         solve sublphl using lp maximizing obj1;
571         abort$(sublphl.modelstat=4) "SE subproblem infeasible";
572         abort$(sublphl.modelstat<>1) "SE subproblem not solved to optimum";
573     else
574         solve subl using lp maximizing obj1;
575         abort$(subl.modelstat=4) "SE subproblem infeasible";
576         abort$(subl.modelstat<>1) "SE subproblem not solved to optimum";
577     );
578 * Update agent's utility
579 if ((ord(thr)=1),
580     autill = autill + mprob.l(thr,thi,slab)*utils1.l/alp(thr)
581 );
582 else
583     autil2 = autil2 + mprob.l(thr,thi,slab)*utils1.l/alp(thr)
584 );
585 );
586 prices(thr,thi,slab,"1") = (lam(thr)*utils1.l-pmc(thr))/presk;
587 voffirm(thr,thi,slab) = (rescl.l*presc+reskl.l*presk)/presk;
588 * Theory says the following expression should equal zero.
589 Abort if it doesn't
590 gap = prices(thr,thi,slab,"1")-voffirm(thr,thi,slab);
591 abort$(gap>0.000001 or gap<-0.000001) "Problem with SE prices"
592 * Write the solution to the output file
593 put "Self-employment club number ", count1:1:0 /
594     "Club type is (", th0(thr):1:0, ",0)" /
595     "The number of these clubs is ", mprob.l(thr,thi,slab):<4:~
596 3 //;
597 put "probability      c      q      a      k " //;
598 loop((c,q,a,k)$ (pil.l(c,q,a,k)>0),
599     put pil.l(c,q,a,k):<>12:3, c0(c):8:2, q0(q):8:1,
600     a0(a):8:1, k0(k):8:1 //;
601 );
602 put //;
603 count1 = count1 + 1;
604 * For SW clubs, solve the subproblem, write the solution to the output
605 * file, and calculate the prices.
606 else
607     if (aux(slab)=1,
608         solve sub2phl using lp maximizing obj2;
609         abort$(sub2phl.modelstat=4) "SW subproblem infeasible";
610         abort$(sub2phl.modelstat<>1) "SW subproblem not solved to optimum";
611     );
612     else
613         solve sub2 using lp maximizing obj2;
614         abort$(sub2.modelstat=4) "SW subproblem infeasible";
615         abort$(sub2.modelstat<>1) "SW subproblem not solved to optimum";
616     );
617 );
618 * Calculate prices and update agents' utilities
619 prices(thr,thi,slab,"1") =
620     (lam(thr)*utilsw2.l-pmc(thr))/presk;
621 if ((ord(thr)=1),
622     autill = autill + mprob.l(thr,thi,slab)*utilsw2.l/alp(thr);
623 else

```

```

619         util2 = util2 + mprob.l(thr,thi,slab)*utilss2.l/alp(»
thr);
620     );
621     if ((th1(thi)=1),
622         util1 = util1 + mprob.l(thr,thi,slab)*utilss2.l/alp("1»
");
623         prices(thr,thi,slab,"2") =
624             (lam("1")*utilss2.l-pmc("1"))/presk;
625     elseif (th1(thi)=2),
626         util2 = util2 + mprob.l(thr,thi,slab)*utilss2.l/alp("2»
");
627         prices(thr,thi,slab,"2") =
628             (lam("2")*utilss2.l-pmc("2"))/presk;
629     );
630     voffirm(thr,thi,slab) = (resc2.l*presc+resk2.l*presk)/presk;
631 * Theory says the following expression should equal zero.
632 * Abort if it doesn't
633 * gap = prices(thr,thi,slab,"1")+prices(thr,thi,slab,"2")
634 *         -voffirm(thr,thi,slab);
635 * abort$(gap>0.000001 or gap<-0.000001) "Problem with SW prices"
636 * Write the solution to the output file
637 * put "Supervisor-Worker club number ", count2:1:0 /
638 * "Club type is (", th0(thr):1:0, ", ", th1(thi):1:0,") /
639 * "The number of these clubs is ", mprob.l(thr,thi,slab):<4:3»
/;
640     put "probability   cw       cs       q       a       k " /»
;
641     loop((cw,cs,q,a,k)$(pi2.l(cw,cs,q,a,k)>0),
642         put pi2.l(cw,cs,q,a,k):<>12:3, c0(cw):<>8:2, c0(cs):<>8:2,
643             q0(q):<>8:1,a0(a):<>8:1, k0(k):<>8:1 /;
644     );
645     put /;
646     count2 = count2 + 1;
647 * sumsw indicates whether a particular type of s-w firm exists
648 * if a (1,1) firm exists then sumsw(ipw,ik,1) = 1 and 0 otherwise
649 * if a (1,2) firm exists then sumsw(ipw,ik,2) = 1
650 * if a (2,1) firm exists then sumsw(ipw,ik,3) = 1
651 * if a (2,2) firm exists then sumsw(ipw,ik,4) = 1
652     if ((th0(thr)=1 and th1(thi)=1),
653         sumsw(ipw,ik,"1") = 1;
654     elseif (th0(thr)=1 and th1(thi)=2),
655         sumsw(ipw,ik,"2") = 1;
656     elseif (th0(thr)=2 and th1(thi)=1),
657         sumsw(ipw,ik,"3") = 1;
658     elseif (th0(thr)=2 and th1(thi)=2),
659         sumsw(ipw,ik,"4") = 1;
660     );
661 );
662 );
663 );
664 put "Summary of club distribution (order is same as above)" /;
665 put "Number of clubs   Membership" /;
666 * Important: this loop must be the same as the one above so that the orde
r
667 * is the same
668     loop((thi,thr),
669         loop(slab$(mprob.l(thr,thi,slab)>0),
670             zs = ord(thi)-1;
671             put mprob.l(thr,thi,slab):<>16:3, "         (" ,ord(thr):1:0," ,zs:1:0,»
) " /;
672     );
673 );
674 put #69;
675

```

```

676     pk = presk/presk;
677 * Now write decentralization info to output file.
678     put ////;
679     put "DECENTRALIZATION THAT SUPPORTS THE ABOVE PARETO OPTIMUM":<>64 /
680     "-----":<>64 //;
681     put "Prices:" /;
682     put "   capital                               ", pk:<>4:3 /;
683     count1 = 0; count2 = 0;
684     loop((thi,thr),
685         loop(slab$(mprob.l(thr,thi,slab)>0),
686             zs = ord(thi)-1;
687             if ((th1(thi)=0 and count1=0),
688                 put "   self-employment clubs   number   worker           value of »
firm" /;
689                 count1 = count1+1;
690             );
691             if ((th1(thi)=0 and count1>0),
692                 put "                               ",
693                     count1:<>6:0, prices(thr,thi,slab,"1"):<>6:3,
694                     voffirm(thr,thi,slab):>19:3 /;
695                 count1 = count1+1;
696             );
697             if ((th1(thi)>0 and count2=0),
698                 put "   supervisor-worker clubs   number   worker   super.   value of »
firm" /;
699                 count2 = count2+1;
700             );
701             if ((th1(thi)>0 and count2>0),
702                 put "                               ", count2:<>6:0,
703                     prices(thr,thi,slab,"1"):<>6:3, prices(thr,thi,slab,"2"):<>9:3,
704                     voffirm(thr,thi,slab):<>16:3 /»
);
705                 count2 = count2+1;
706             );
707         );
708     );
709     put /;
710     put "Resource constraint multipliers are mu_(c-q) = ", presc:<5:3,
711         " , mu_k = ",presk:<5:3 /;
712     put /;
713     expenditures(thr) = 0;
714     expenditures(thr) = sum((thi,slab),
715         mprob.l(thr,thi,slab)*prices(thr,thi,slab,"1"));
716     expenditures("1") = expenditures("1")
717         +sum((thr,slab),mprob.l(thr,"1",slab)*prices(thr,"1",slab,»
"2"));
718     expenditures("2") = expenditures("2")
719         +sum((thr,slab),mprob.l(thr,"2",slab)*prices(thr,"2",slab,»
"2"));
720     expper(th) =expenditures(th)/alp(th);
721     put "Expenditures:           TOTAL           PER CAPITA" /;
722     put "   type-1           ", expenditures("1"):<>12:3, expper("1"):<>»
16:3 /;
723     put "   type-2           ", expenditures("2"):<>12:3, expper("2"):<>»
16:3 /;
724     put "   Entire Population ", (expenditures("1")+expenditures("2")):<>»
12:3 /;
725     put /;
726     put "Endowments of Capital:   TOTAL           PER CAPITA" /;
727     put "   type-1           ", expenditures("1"):<>12:3, expper("1"):<>»
16:3 /;
728     put "   type-2           ", expenditures("2"):<>12:3, expper("2"):<>»
16:3 /;

```

```

730 put " Entire Population ", (expenditures("1")+expenditures("2")):<>>
12:3 /;
731 put /;
732 put "Utility of type-1 = ", autill:<6:4 /;
733 put "Utility of type-2 = ", autill2:<6:4 /;
734
735 putpage mfile;
736
737 * Finish the Pareto weight and capital endowment loops.
738 );
739 );
740 * Write the types of supervisor worker firms as a function of the
741 * pareto weights and capital endowment to a file for graphing purposes;
742 put mfile2;
743 loop(ipw,
744     put "pw(", ord(ipw), ") = ", ParW(ipw), ";" /;
745 );
746 loop(ik,
747     put "kap(", ord(ik), ") = ", kap(ik), ";" /;
748 );
749 loop((ipw,ik,j),
750     put "sumsw(", ord(ipw), ",", ord(ik), ",", ord(j), ") = ",
751         sumsw(ipw,ik,j), ";" /;
752 );

```