```
#include <oxstd.h>

/* ================================== */
/* Auxiliary class for OLS estimation */
/* ================================== */

class OLS
{
  decl Y;
  decl X;
  decl N;
  decl K;
  decl B;
  decl u;
  decl R2;

  OLS (Y, X);
  getBeta ();
  getRSquared ();
}

OLS::OLS (Y, X)
{
  this.Y = Y;
  this.X = X;
  this.N = rows (Y);
  this.K = columns (X) + 1;
  decl W = ones (N, 1) ~ X;
  olsc (Y, W, &B);
  u = Y - W * B;
  decl y = Y - meanc(Y);
  R2 = double (1.0 - sumsqrc(u) / sumsqrc(y));
}


OLS::getBeta ()
{
  return B;
}

OLS::getRSquared ()
{
  return R2;
}



/* ================================== */
/* Evaluates qs and Omega functions */
/* ================================== */

class Omega
{

  decl A;
  decl B;
  decl C;
  decl N;
  decl rho;
  decl delta;
  decl z;
  decl nlo;
  decl nup;
```

```
  Omega (N, rho, delta, A, B, C, z);
  evalQs (n);
  evalOmega (n);
  solve ();
  getNLow ();
  getNUp ();
}

Omega::Omega (N, rho, delta, A, B, C, z)
{
  this.N = N;
  this.rho = rho;
  this.delta = delta;
  this.A = A;
  this.B = B;
  this.C = C;
  this.z = z;

  /* Eqs. (9) and (10) */
  decl k = (delta * B + 2 * delta * C * (N + rho * z));
  this.nlo = 1.0 / k;
  this.nup = (k - sqrt (k * k - 8 * delta * C)) / (4 * delta * C);
}

Omega::evalQs (n)
{
  if (n <= nlo)
      return 1.0;
  if (n >= nup)
      return 0;
  return ((1 - n * delta * B) / (2 * delta * C * n) - N + n - rho *
z) / n;
}


Omega::evalOmega (n)
{
  decl qs = evalQs (n);
  decl qsn1z = evalQs (n - 1);

  return (qs - 1 - delta * (B + 2 * C * (rho * z + N - n)) *
              (n * qs - (n - 1) * qsn1z) -
              delta * C * (n ^ 2 * qs ^ 2 - (n - 1) ^ 2 * qsn1z ^ 2)
+
              2 * delta * C * (n - 1) * qsn1z +
              2 * delta * C * (rho * z + N - n) + delta * (B + C));
}

Omega::solve ()
{
  decl nstar = -1;
  decl bestOmega;
  for (decl i = N; i >= nlo; i--) {
      decl omega = evalOmega (i);
      if (omega == 0.0)
        return i;
      if (nstar == -1) {
        if (omega > 0.0) {
            nstar = i;
            bestOmega = omega;
        }
      } else {
        if (omega > 0.0 && omega < bestOmega) {
```

```
            nstar = i;
            bestOmega = omega;
          }
        }
  }
  return nstar;
}

Omega::getNLow ()
{
  return nlo;
}


Omega::getNUp ()
{
  return nup;
}


/* ================================ */
/* Algorithm */
/* ================================ */

class Alg
{
  decl delta;
  decl rho;
  decl N;
  decl gamma;
  decl A;
  decl B;
  decl C;
  decl z;
  decl nstar;
  decl qs;
  decl nlo;
  decl nup;

  Alg (N, delta, rho, gamma);
  solve ();
  simulate (period);
  startingValues ();
  checkValues ();
  getA ();
  getB();
  getC();
  getZ ();
  getNStar ();
  getQs ();
  getNLow ();
  getNUp ();
}

Alg::Alg (N, delta, rho, gamma) {
  this.delta = delta;
  this.rho = rho;
  this.N = N;
  this.gamma = gamma;
  startingValues ();
}

Alg::startingValues ()
{
```

```
  /* Starting values for A, B, and C */
  /* Eq. (3) */
  A = 1.0 /(1 - delta) - (delta * gamma * N * N * (1 + delta * rho))
/
        ((1 - delta * rho) * (1 - delta * rho * rho) * (1 - delta));
  /* Eq. (4) */
  B = (2 * delta * rho * gamma * N) /
        ((1 - delta * rho) * (1 - delta * rho * rho));
  /* Eq. (5) */
  C = gamma / (1 - delta * rho * rho);


  println ("\n    Starting values: ");
  println ("    A = ", A,  ", B = ", B,  ", C = ", C);
}

Alg::checkValues ()
{
  decl cond6 = FALSE;
  decl cond7 = FALSE;

  while (!cond6 || !cond7) {
      /* Check condition (6) */
      cond6 = delta * (B + (2 * C * N) / (1 - rho)) <= 1;
      /* Check condition (7) */
      cond7 = 1.0 / (delta * N) < B;

      println ("    A = ", A, ", B = ", B, ", C = ", C);
      println ("    Cond. 6 -> ", cond6? "TRUE" : "FALSE",
                  ", Cond. 7 -> ", cond7? "TRUE" : "FALSE");

      if (!cond6  || !cond7) {
        /* Correct B and/or C */
        if (!cond6 && cond7) { /* 7.1.1 */
            // B = 1.0 / delta - (2 * N * C) / (1 - rho);
            //if (B <= 1.0 / (delta * N)) { /* 7.1.1.2 */
              decl Cmax = ((1 - rho ) * (1 - delta * B)) / (2 * N *
delta);
              C = double (ranu (1, 1) * Cmax);
              //}
        } else if (cond6 && !cond7) {
            if (B <= 1.0 / (delta * N)) { /* 7.1.1.2 */
              C = (1 * (N - 1) * (1 - rho)) / (20 * delta * N ^ 2);
              B = 5.0 / delta - (2 * N * C) / (1 - rho);
            }
        } else { /* !cond6 && !cond7 */
              C = (1 * (N - 1) * (1 - rho)) / (20 * delta * N ^ 2);
              B = 5.0 / delta - (2 * N * C) / (1 - rho);
        }
      }
  }
}


Alg::solve () {
  println ("delta = ", "%g", delta,
               ", rho = ", "%g", rho,
               ", gamma = ", "%g", gamma,
               ", N = ", "%d", N);



  /* Boolean flag for convergence */
  decl conv = FALSE;
```

```
  /* Iterations count */
  decl iter = 0;

  println ("\n    Updating A, B, and C");
  while (conv == FALSE) {
      if (iter > 100) {
        println ("Convergence not achieved after 5000 iterations");
        return FALSE;
      }

      checkValues ();

      /* Store values needed when checking convergence */
      decl oldA = A;
      decl oldB = B;
      decl oldC = C;

      /* 7.2 */
      decl h = 200;
      decl z = zeros (h + 1, 1);
      decl y = zeros (h + 1, 1);
      for (decl j = 0; j <= h; j++) {
        z[j] = (j * N)/ (h * (1 - rho));
        decl omega = new Omega (N, rho, delta, A, B, C, z[j]);
        decl nstar = omega.solve ();
        if (nstar == -1) {
            println ("No positive value of Omega(n) for n in [",
omega.getNLow (),
                        ", ", N , "]");
            return FALSE;
        }
        y[j] = nstar - nstar * omega.evalQs (nstar);
      }

      /* Linear regression */
      decl ols = new OLS (y, z);
      decl Bols = ols.getBeta ();

      decl alpha = Bols[0];
      decl beta = Bols[1];

      /* Update A, B, and C */
      C = gamma / (1 - delta * (rho - beta)^2);
      B = (beta / N + 2 * delta * C * (rho - beta) * (N - alpha)) /
            (1 - delta * (rho - beta));
      A = ((N - alpha) / N - delta * (N - alpha) * B
             - delta * (N - alpha)^2 * C) / (1 - delta);


      // Convergence check
      decl tol = 0.05;
      decl critA = (A - oldA) / oldA;
      decl critB = (B - oldB) / oldB;
      decl critC = (C - oldC) / oldC;

      if (fabs (critA) <= tol && fabs (critB) <= tol && fabs (critC)
<= tol)
        conv = TRUE;

      println ("    ",
                  "Iter = ", ++iter, ", ",
                  "A = ", A, " (", "%.2f%%", critA * 100, "), ",
                  "B = ", B, " (", "%.2f%%", critB * 100, "), ",
```

```
                  "C = ", C, " (", "%.2f%%", critC * 100, ") ");
  }


  println ("");
  println ("    Convergence achieved in ", "%d", iter, "
iterations.");
  println ("    A = ", A,  ", B = ", B,  ", C = ", C);
  return TRUE;
}


Alg::simulate (periods)
{
  z = zeros (periods + 1, 1);
  nstar = zeros (periods + 1, 1);
  nlo = zeros (periods + 1, 1);
  nup = zeros (periods + 1, 1);
  qs = zeros (periods + 1, 1);


  /* Starting value */
  z[0] = 0;
  decl omega = new Omega (N, rho, delta, A, B, C, z[0]);
  nstar[0] = omega.solve ();
  if (nstar[0] == -1)
      return FALSE;
  nlo[0] = omega.getNLow ();
  nup[0] = omega.getNUp ();
  qs[0] = omega.evalQs (nstar[0]);

  /* Simulation */
  for (decl t = 1; t <= periods; t++) {
      z[t] = rho * z[t - 1] + N - nstar[t - 1] + nstar[t - 1] * qs[t
- 1];
      omega = new Omega (N, rho, delta, A, B, C, z[t]);
      nstar[t] = omega.solve ();
      if (nstar[0] == -1)
        return FALSE;
      nlo[t] = omega.getNLow ();
      nup[t] = omega.getNUp ();
      qs[t] = omega.evalQs (nstar[t]);
  }

  return TRUE;
}

Alg::getA ()
{
  return A;
}

Alg::getB ()
{
  return B;
}

Alg::getC ()
{
  return C;
}

Alg::getZ ()
{
```

```
  return z;
}

Alg::getQs ()
{
  return qs;
}

Alg::getNUp ()
{
  return nup;
}

Alg::getNLow ()
{
  return nlo;
}

Alg::getNStar ()
{
  return nstar;
}

main ()
{
  decl logfile = fopen ("log.txt", "l");


  /* Values for rho */
  decl vrho = < 0.70, 0.75, 0.80, 0.85, 0.90, 0.95 >;

  /* Values for delta */
  decl vdelta = < 0.90, 0.92, 0.94, 0.96, 0.98, 0.99 >;

  /* Values for N */
  decl vN = < 75, 80, 85, 90, 95, 100 >;


  decl summarycorr, summaryg, summaryqs;

  summarycorr  =  fopen ("summarycorr.txt", "w");
  summaryg = fopen ("summaryg.txt", "w");
  summaryqs = fopen ("summaryqs.txt", "w");
  fprintln (summarycorr,
                  "%15s", "N",
                  "%15s", "delta",
                  "%15s", "rho",
                  "%15s", "rgn0",
                  "%15s", "rgn150",
                  "%15s", "rgGp");

  fprintln (summaryg,
                  "%15s", "N",
                  "%15s", "delta",
                  "%15s", "rho",
                  "%15s", "gamma",
                  "%15s", "igamma",
                  "%15s", "alpha",
                  "%15s", "Gf",
                  "%15s", "Gp",
                  "%15s", "rzn");

  decl minGp = 5000;
  decl minGpFile;
```

```
  decl maxGp = -1;
  decl maxGpFile;
  decl avgGp = 0;

  for (decl irho = 0; irho < columns (vrho); irho++) {
      decl rho = vrho[irho];
      for (decl idelta = 0; idelta < columns (vdelta); idelta++) {
        decl delta = vdelta[idelta];
        for (decl iN = 0; iN < columns (vN); iN++) {
            decl N = vN[iN];
            /* Eq (1) */
            decl upgamma = (1 - rho * delta) * (1 - rho) / (2 * delta
* N);

            /* Eq (2) */
            decl logamma = (1 - rho * delta) / (2 * delta * N * N);
            decl vgamma = zeros (10, 1);
            decl vn0 = zeros (10, 1);
            decl vn150 = zeros (10, 1);
            decl vGp = zeros (10, 1);

            for (decl i = 1; i <= 10; i++) {

              /* Values for gamma */
              decl gamma = logamma + i * (upgamma - logamma) / 10;
              vgamma[i - 1] = gamma;
              decl alg = new Alg (N, delta, rho, gamma);
              decl initA = alg.getA ();
              if (alg.solve() == TRUE) {
                  decl periods = 150;
                  if (alg.simulate (periods)) {
                    decl t = range(0, periods)';
                    decl z = alg.getZ ();
                    decl qs = alg.getQs ();
                    decl nstar = alg.getNStar ();
                    decl nlo = alg.getNLow ();
                    decl nup = alg.getNUp ();
                    decl filename = sprint ("n", "%2d", N,
                                                    "r", "%2d",
rho * 100,
                                                    "d", "%2d",
delta * 100,
                                                    "g", "%02d",
i,
                                                    ".txt");
                    decl file = fopen (filename, "w");
                    decl data = (t ~ z ~ nstar ~ qs ~ nlo ~ nup);
                    fprintln (file, "%c",
                                    {"t", "z", "nstar", "qs", "nlo",
"nup" },
                                    "%cf",
                                    { "%12.0f", "%12.4g", "%12.4g",
"%12.4g",
                                          "%12.4g", "%12.4g" },
                                    data);
                    fclose (file);

                    decl idx = qs .!= 0;
                    if (rows (idx) > 0) {
                        fprintln (summaryqs, filename);
                        fprintln (summaryqs, "%c",
                                    {"t", "z", "nstar", "qs", "nlo",
"nup" },
                                    "%cf",
```

```
                                    { "%12.0f", "%12.4g", "%12.4g",
"%12.4g",
                                          "%12.4g", "%12.4g" },
                                      selectifr(data, idx));
                    }

                    /* Summary */
                    decl alpha = ((1 - delta * rho) ^2) /
                                       (4 * delta * gamma * N ^ 2 *
(1 - delta));
                    decl Gf = alpha - initA;
                    decl Gp = (alg.getA () - initA ) / Gf;

                    decl r = correlation (z ~ nstar)[1][0];

                    if (Gp < minGp) {
                        minGp = Gp;
                        minGpFile = filename;
                    }

                    if (Gp > maxGp) {
                        maxGp = Gp;
                        maxGpFile = filename;
                    }

                    avgGp += Gp;

                    fprintln (summaryg,
                                    "%15g", N,
                                    "%15g", delta,
                                    "%15g", rho,
                                    "%15g", gamma,
                                    "%15g", i,
                                    "%15g", alpha,
                                    "%15g", Gf,
                                    "%15g", Gp,
                                    "%15g", r);


                    vn0[i - 1] = nstar[0];
                    vn150[i - 1] = nstar[periods];
                    vGp[i - 1] = Gp;


                    if (r >= 0) {
                        decl bad = fopen ("BAD.txt", "a");
                        fprintln (bad,
                                    "N = ", N, ", ",
                                    "rho = ", rho, ", ",
                                    "delta = ", delta, ", ",
                                    "gamma = ", gamma, ", ",
                                    "gammai = ", i, ", ",
                                    "corr = ", r);
                        fclose (bad);
                    }
                }
            }
        }

        decl rgn0 = correlation (vgamma ~ vn0)[0][1];
        decl rgn150 = correlation (vgamma ~ vn150)[0][1];
        decl rGp = correlation (vgamma ~ vGp)[0][1];

        if (rgn0 >= 0 || rgn150 >= 0 || rGp >= 0)
```

```
                fprintln (summarycorr,
                                "%15g", N,
                                "%15g", delta,
                                "%15g", rho,
                                "%15g", rgn0,
                                "%15g", rgn150,
                                "%15g", rGp);
          }
        }
    }

    fclose (summaryqs);
    fclose (summaryg);
    fclose (summarycorr);
    avgGp = avgGp / (10 * columns (vN) * columns (vdelta) * columns
(vrho));
    println ("Min Gp = ", minGp, " -> ", minGpFile);
    println ("Max Gp = ", maxGp, " -> ", maxGpFile);
    println ("Avg Gp = ", avgGp);
    //  fclose (logfile);
}
```