```
/* IV-regression functions */
ivreg(const mY,const mVariables,const mInstruments,const
mWeight)
{
  decl mInvXZX=invertgen((mVariables'mInstruments)*mWeight*
(mInstruments'mVariables));
  decl vIV;
  if(mInvXZX==0) vIV=constant(.NaN,columns(mVariables),1);
  else vIV=mInvXZX*(mVariables'mInstruments)*mWeight*
(mInstruments'mY);
  return vIV;
}

/* This function evaluates the idiosyntractic component of
utility */
value(const aMu,const vParam,const t)
{
  decl i;
  decl rowid=aProductID[t];
  decl mMu=new matrix[rows(rowid)][Sim];
  for(i=0;i<rows(vParam);i++) mMu+=vParam[i]*(aZ[t])[i];
  aMu[0]=exp(mMu);
  return 1;
}
demand(const mMu,const aShare,const aJac,const vDelta,const
t,const vParam)
{
  decl i;
  decl rowid=aProductID[t];
  decl eV=exp(vDelta[rowid]).*mMu;
  decl mS=eV./(1+sumc(eV));
  decl vShat=meanr(mS);
  if(aJac[0]) {
    decl mD=diag(meanr(mS.*(1-mS)))-setdiagonal(mS*mS'/Sim,0);
    aJac[0]=mD;
    }
  aShare[0]=vShat;
  return 1;
```

```
}
inverse(const aDelta, const vP,const eps1,const eps)
{
  decl vShat,vDelta=vDelta0;
  decl f=1000;
  decl mJacobian=1;
  decl rowid,t;
  decl it,maxit=1000;
  decl vIT=new matrix[T][1];
  decl mMu;
  decl time0=timer();
  decl mNorm=<>;
  decl maxT=T;
  if(iprint) maxT=1;
  parallel for(t=0;t<maxT;t++) /* Parallelize the inversion
across markets. When possible use Nb processors = T (31) */
    {
      time0=timer();
      /* Pre-compute the heterogeneity component of utility
(indepedent of delta) */
      value(&mMu,vP,t);
      rowid=aProductID[t];
      vIT[t]=0;
      f=1000;
      do{
        /* Evalute the demand without the jacobian matrix if the
norm is larger than 1  */
        if(norm(f)>eps1) {
          mJacobian=0;
          demand(mMu,&vShat,&mJacobian,vDelta,t,vP);
          f=log(vShare[rowid])-log(vShat); /* Zero function: Log
difference in shares */
          vDelta[rowid]=vDelta[rowid]+f; /* Contraction mapping
step */
        }
        /* Evalute the demand with the jacobian matrix if the
norm is less than 1 */
        else {
          mJacobian=1;
```

```
          demand(mMu,&vShat,&mJacobian,vDelta,t,vP);
          f=log(vShare[rowid])-log(vShat); /* Zero function: Log
difference in shares */
          vDelta[rowid]=vDelta[rowid]+invert(mJacobian./vShat)*f;
/* Newton step */
        }
        vIT[t]+=1;
        if(iprint==1 && t==0) {
          mNorm~=(norm(f)|vIT[t]);
          println("t = ",t," it ",vIT[t]," norm : ",norm(f));
        }
        //
      }while(norm(f)>eps && vIT[t]<maxit);
      if(norm(f)>eps) vDelta[rowid]=constant(.NaN,rowid);
    }
  if(iprint) {
    DrawXMatrix(0,mNorm[0][],"Zero function norm",mNorm[1]
[],"Iteration");
    ShowDrawWindow();
    if(eps1>0) SaveDrawWindow("Inverse_iteration_newton.pdf");
    else SaveDrawWindow("Inverse_iteration_contraction.pdf");
  }
  //println("cpu time: ",(timer()-time0)/100);
  aDelta[0]=vDelta;
  return 1;
}


/* GMM objective function */
gmm_obj(const vP, const adFunc, const avScore, const amHessian)
{
  /* Invert demand */
  decl eps1=1;
  decl eps=10^(-12);
  inverse(&vDelta0,vP,eps1,eps);
  /* Quality decomposition */
  decl vLParam=ivreg(vDelta0,mX,mIV,A);
  decl vXi=vDelta0-mX*vLParam;
  /* GMM objective function */
```

```
    mG=vXi.*mIV;
    decl scale=100;
    decl g=sumc(mG);
    if(isnan(vDelta0)==1) adFunc[0]=.NaN;
    else adFunc[0]=double(-g*A*g'/scale);
    return 1;
}
```