joint.pas

July 21, 2004

Contents

1 jointproduction

program *JointProduction* ;

1.1 Introduction

This document describes a simulation experiment to test an ideand put forward by Ajit Sinha concerning Sraffian prices and joint production. His suggestion is that if we have:

1. a system of joint production with n products

2. two possible numeraires m_1 and m_2
3. an initial technology matrix t_a and a slightly perturbed technology matrix t_b
4. then the Sraffian profit rate equalising rule gives us a tensor of prices

$$\begin{matrix} \mathbf{p}_{1,a} & \mathbf{p}_{1,b} \\ \mathbf{p}_{2,a} & \mathbf{p}_{2,b} \end{matrix}$$

where the $\mathbf{p}_{i,j}$ are price vectors.

It is clear that in such a configuration we would espect that in going from $\mathbf{p}_{i,a} \rightarrow \mathbf{p}_{i,b}$ we would see some prices rising and some falling. Ajit's hypothesis is that there will exist some prices which are rising in the transition $\mathbf{p}_{1,a} \rightarrow \mathbf{p}_{1,b}$ which will be falling in the transition $\mathbf{p}_{2,a} \rightarrow \mathbf{p}_{2,b}$. That is to say that a change in the numeraire will result in a change in the direction of price rises under technical change.
The first question is to ask whether a set of 2 price vectors expressed in a 3rd numeraire, which we assume for the moment is state money and thus not part of the n commodities, could be transformed into 4 price vectors in terms of the internal numeraires and which would have the traits that Ajit proposes.
Suppose we have the price vector in Euro $(1, 2, 3, 4)$ and the vector $(2, 1, 8, 4)$ and let us assume the commodities are (gold,silver, corn,iron). Then we have the four price vectors for corn and iron in terms of gold and silver:

$$\begin{matrix} & t_1 & t_2 \\ gold & (3, 4) & (4, 2) \\ silver & (1.5, 2) & (8, 4) \end{matrix}$$

If we express the directions of price change we have

$$\begin{matrix} (\uparrow, \downarrow) \\ (\uparrow, \uparrow) \end{matrix}$$

Which would appear to meet Ajit's initial requirement without our having to investigate the complexities of joint production. Let us call this weak reversing. If this is too trivial and what Ajit actually wants is a direction change like:

$$\begin{matrix} (\uparrow, \downarrow) \\ (\downarrow, \uparrow) \end{matrix}$$

Let us call this strong reversing. Can this exist if there is a single underlying price vector at each time step such that all of the price vectors in different numeraires agree up to a scalar transform?
Observe that we can always so chose out units of commodity measure to make the first Euro price vector degenerate, we simply chose units of gold, silver, corn and iron that cost Euro 1 in time step 1. This gives us (1,1,1,1) as our price vector. Let our second price vector be (g, s, c, i), we want to chose this such that:
$\frac{g}{c} > 1, \frac{g}{i} < 1, \frac{s}{c} < 1, \ \frac{s}{i} > 1$
Thus we have
$g > c$, (1)

$g < i$, (2)
$s < c$, (3)
$s > i$ (4)
Relations (1) and (3) imply $g > s$, but relations (2) and (4) imply that $g < s$, so we have a contradiction. It thus follows that strong reversing can not arise from the simple switch of numeraire to represent a situation in which economic calculation takes place in state currency.
At first sight it seems that there is another possibility. In the example so far we have assumed that the real unit of account is the Euro, and this is the unit that profit rates are calculated on. Now since the profit rate introduces a non linear principle into the equations determining prices, there seems the possibility that when profits are calculated in gold or silver, it may no longer be possible to construct our degenerate initial price vector. The vector might be degenerate if profits had been calculated in gold but not if profits had been calculated in silver. However one can see that this is not the case provided that technology has not changed for a long time prior to t_1, since in that case relative prices do not change over time. It follows that one can always chose a unit of weight of silver such that it will have the same exchange value as our putative gold coin, and that provided calculations are done in this unit of account, profit calculations in terms of gold and in terms of silver will be identical.
It follows that strong reversing is impossible all we have to look at is whether technical change can result in weak reversing. Our method will be to construct random i/o matrices and then see how frequently weak reversing occurs after a technical change.

```
const
    n =5;
    runs =100;
    numcommodities =n ;
    numindustries =n ;
```

We define n to be the number of commmodities in the system, and we will collect statistics over the specified number of runs.

```
type
    percent = 1..100;
    industry = 1..numindustries ;
    commodity = 1..numcommodities ;
    iomatrix = array [industry ,commodity ] of real ;
```

The type iomatrix will be used to describe both the input and the output matrices. The rows of the matrix represent production processes and the columns represent commoditities.

```
    pricevector = array [commodity ] of real ;
```

commodityvector = **array** [*commodity*] **of** *real* ;
industryvector = **array** [*industry*] **of** *real* ;

A price vector defines the prices of all commodities. It is indexed by commodities. In the case where an internal commodity is taken as the money commodity, at least one element of the price vector will =1. An industry vector specifies the amount of some resource used in each industry. It is indexec by industry.

type
numeraire = 1..2;

We will run using two different numeraires

var
Let *U*, *G*, *Net* ∈ iomatrix;
P,Q: **array** [*numeraire*] **of** *pricevector* ;
Let λ ∈ industryvector;
Let *w* ∈ commodityvector;
Let ρ ∈ real;
Let *m* ∈ commodity;

Our models will be defined in terms of a Use matrix U, whose element u_{ij} represents the amount of the jth commodity used in industry i. It is assumed that these quantities are in some sort of natural units. Conversely the Generate matix G, represents the outputs of the industries such that g_{ij} is the output of commodity j by the ith industry.

- P is the price vector before technical change, Q the price vector after technical change.
- w is the real wage represented as a vector of physical units of each commodity.
- ρ is the rate of profit which is assumed to be equal in all industries.
- λ specifies the labour usage of the economy.
- The variable m indicates which commodity is currently used as money.

We now introduce a group of utility functions that are useful for randomising the expanded matrices.

function *br (p* :*percent*):*boolean* ; (see Section 2)
function *fr* :*real* ; (see Section 3)

```
function ir ( top :integer ):integer ; (see Section 4 )
procedure setnet ; (see Section 5 )
procedure makeiomatrices ( Usparse ,Gsparse :percent ); (see Section 6 )
procedure deriveprices ( var p :pricevector ;numeraire :commodity ); (see Section 7 )
procedure removejointproduction ; (see Section 8 )
var
    Let tempuse ∈ commodityvector;
    Let matrices, pricevectors ∈ text;
procedure savematrices ( n :string [30]); (see Section 9 )
type
    direction =( up ,down );
procedure saveprices ( n :string [30]); (see Section 10 )
begin
    makeiomatrices (90, 90);

    removejointproduction;
    write(U, G);
    savematrices ( 'matrices.tex' );
```

The table below displays the matrices that we have built.

```
G
     0.61684     0.00000     0.00000     0.00000     0.00000
     0.00000     0.03318     0.00000     0.00000     0.00000
     0.00000     0.00000     0.55197     0.00000     0.00000
     0.00000     0.00000     0.00000     0.41343     0.00000
     0.00000     0.00000     0.00000     0.00000     0.38458

U
     0.01304     0.00519     0.05109     0.11997     0.19107
     0.00000     0.00000     0.02063     0.00000     0.00000
     0.14247     0.00000     0.14848     0.00000     0.00000
     0.05249     0.00000     0.00535     0.00000     0.00000
     0.10042     0.01140     0.05044     0.08674     0.00122

      lambda
     0.12921     0.00537     0.11651     0.01417     0.73474
   real wage
     0.15421     0.00829     0.13799     0.10336     0.09615
```

```
    deriveprices (p1, 1);
    deriveprices (p2, 2);
```

Now we change the technology by swapping two rows of the use matrix

$tempuse \leftarrow U_3$;
$U_3 \leftarrow U_4$;
$U_4 \leftarrow tempuse$;

This will not change the net product of the economy and thus we can assume that distribution will be unchanged. We recompute the net consumption matrix after this change.

setnet;
deriveprices (q_1, 1);
deriveprices (q_2, 2);

We now write the prices to a file and display it in table below.

```
before technical change
   1.00000   0.82186   0.80477   0.22116   1.50925
   1.21675   1.00000   0.97920   0.26909   1.83638

after technical change
   1.00000   0.27711   0.22892   0.65370   1.42767
   3.60868   1.00000   0.82609   2.35898   5.15202

change direction
      down      down      down        up      down
        up      down      down        up        up
```

Note that even this simple example exhibits weak reversal.

saveprices ('prices.tex');

change back to original conditions of production

$U_4 \leftarrow U_3$;
$U_3 \leftarrow tempuse$;

Now alter the output of industry 3 to be higher and adjust the wage as well, these changes are made in such a way as not to alter the distribution of income, since the increas in wage is by half of the net increase in production, which is the same ratio as the wage to the existing net product.

$G_{3,3} \leftarrow G_{3,3} + 0.5$;
$w_3 \leftarrow w_3 + 0.25$;
setnet;

```
    savematrices ( 'augmentedmatrices.tex' );
    deriveprices (q1, 1);
    deriveprices (q2, 2);
    saveprices ( 'augmentedprices.tex' );
end .
```

2 br

```
function br ( p :percent ):boolean ;
```

br makes a random boolean choice with p giving the percentage probability that the answer will be true. It uses the library function random which returns a random integer.

```
begin
    br← (abs (random) mod 100) ≤ p;
end ;
```

3 fr

```
function fr :real ;
```

fr returns a random floating point number in the range 0..1, again using the library function random.

```
const
    mask =$ffff;
begin
    fr← random∧mask / mask;
end ;
```

4 ir

```
function ir ( top :integer ):integer ;
```

This returns a random number in the range 1..top

```
begin
    ir← 1 + (random mod top);
end ;
```

5 setnet

procedure *setnet* ;

This computes the net production of each industry taking into account the consumption of its workers.

begin
Net$\leftarrow$ *G* - *U* - *w* $\times$ λ^{T};
end ;

6 makeiomatrices

procedure *makeiomatrices (Usparse* ,*Gsparse* :*percent*);

This procedure initialises the matrices U and G to form a consistent pair of production matrices created in such a way that the economy has a net positive product of all commodities. The parameter Usparse and Gsparse specify how sparse the Use and Generate matrices will be. Sparseness of 100% corresponds to a situation where each commodity is used/generated by each industry.

The procedure simultaneously constructs the labour input vector and initialises the rate of profit.

The approach taken to constructing the matrices is to start out with a simple pair of matrices containing a single commodity for $g_{1,1} > u_{1,1}$ and then carrying out a series of operations which grow the matrices whilst preserving the net input/output ratio of the system.

Essentially we start of with a simple 'corn economy' whose expansion ratio is well defined, we then divide the corn into two categories of commodity whilst preserving the same overall expansion ratio, and recursively apply the process. The aim is to generate an i/o table that approximates the structure of real i/o tables. These can be presented with successively greater degrees of disaggregation - thus at one level one might have a sector called timber products.

On disagregating this might divide into plywood, sawn timber, and fiber board products. The three sub-sectors will show substantial similarity in their input structure, one would certainly expect them to be more similar in terms of cost structure than any of them were to for example non-ferrous metal production. This genetic similarity of sibling industries is to be emulated by the procedure of successively spliting industries, represented by rows in the U and G matrices, into two daughter industries that are similar to but not identical to each other.

The two basic operations are to split the matrices along the columns to increase the number of products, and to split them along the rows to increase the number of industries.

For these purposes it is useful to introduce a new schematic type representing either a column of the matrix.

```
type
    column(length:integer )= array [0..length ,0..0] of real ;
```

The real work of splitting rows and columns is done by the following two procedures

```
procedure splitcolumn ( var src ,dest :column ;dfrac :real ); (see Section 11 )
procedure splitrow ( var src ,dest :commodityvector ;sparseness :percent ); (see Section 12 )
var
    Let i, j ∈ integer;
    Let f ∈ real;
```

The body of the procedure splits a randomly selected column and then a randomly selected row until we have the full square matrix constructed.

```
begin
```

Initialise the system to have a reproducible corn economy in which each seed of corn produces two seeds at harvest, and half of a seed has to be paid to the workers

$U_{1,1} \leftarrow 1;$

```
    G_{1,1} ← 2;
    w_1 ← 0.5;
    lambda_1 ← 1;
    ρ ← 0.5;
    for i ← 2 to n do
    begin
fraction to dest
        f ← fr;
        j ← ir (i - 1);
        splitcolumn ( U [][j ..j ],U [][i ..i ],f );
        splitcolumn ( G [][j ..j ],G [][i ..i ],f );
        { split real wage in same ratio }
        w_i ← w_j × f;
        w_j ← (1 - f) × w_j;
        f ← fr;
        j ← ir (i - 1);
        splitrow (U_j, U_i, Usparse);
        splitrow (G_j, G_i, Gsparse);
        f ← fr;
split labour entry too
        lambda_i ← f × lambda_j;
        lambda_j ← (1 - f) × lambda_j;

    end ;
    setnet;

end ;
```

7 deriveprices

procedure *deriveprices* (**var** *p* :*pricevector* ;*numeraire* :*commodity*);

This procedure attempts to search for a set of profit equalising prices. It uses an algorithm that assumes no joint production. At each iteration it computes a new price vector np by setting the prices of each product to the ones which would equalise profits. This is iterated several times to converge the result. After 20 iterations the results are found to be stable.

```
const
    iterations =20;
    adjust =0.5;
var
    Let i, j ∈ integer;
    Let costs, sales, effect ∈ commodityvector;
    Let totalsales, sad, pn, totalcapital ∈ real;
    Let moneywage ∈ real;
    Let np ∈ commodityvector;
```

```
    Let capital, profit, profitrate, dev ∈ industryvector;
begin
    np← 1;
    profit← 0;
    capital← 1;
    for i← 1 to iterations do
    begin
        p← np;
        pn← p_numeraire;
        p← p/pn;
        totalcapital← Σ capital ;
        ρ← Σ profit / totalcapital;

        moneywage← w.p;
        dev← profitrate - ρ;
        sad← Σ (abs (dev)) ;

        for j← 1 to numindustries do
        begin
            costs← U_j × p;
            sales← G_j × p;
            effect← Net_j × p;
            capital_j← Σ costs ;
            totalsales← Σ sales ;
            profit_j← Σ effect ;
            profitrate_j← profit_j / capital_j;
            np ← { np                                                  if effect < 0
                 { ((1 + ρ) × capital_j + lambda_j × moneywage)/G_j    otherwise      ;
        end ;
    end ;
end ;
```

8 removejointproduction

procedure *removejointproduction* ;

This takes the randomly constructed joint product matrix and diagonalises it into a single product matrix.

```
var
    Let outputs ∈ commodityvector;
begin
    outputs← Σ G^T ;
    G ← { outputs   if ι_0 = ι_1
        { 0.0       otherwise   ;
```

```
    setnet;
end ;
```

9 savematrices

```
procedure savematrices ( n :string [30]);
```

This outputs the conditions of production to a file formated in Latex Verbatim mode so that they can be re-imported to this document. They are saved in file called n

```
begin
    assign (matrices, n);
    rewrite (matrices);
    writeln(matrices, '\begin{verbatim}' );
    write(U);
    writeln(matrices, 'G' );
    write(matrices, G);
    writeln(matrices, 'U' );
    write(matrices, U);
    writeln(matrices, 'lambda' );
    write(matrices, λ);
    writeln(matrices, 'real wage' );
    write(matrices, w);
    writeln(matrices, '\end{verbatim}' );
    close (matrices);
end ;
```

10 saveprices

```
procedure saveprices ( n :string [30]);
var
    dir: array [numeraire ,commodity ] of direction ;
begin
    dir ← { up   if q > p
          { down otherwise ;
    write(q, p, dir);
    assign (pricevectors, n);
    rewrite (pricevectors);
    writeln(pricevectors, '\begin{verbatim}' );
    writeln(pricevectors, 'before technical change' );
    write(pricevectors, p : 10);
    writeln(pricevectors, 'after technical change' );
    write(pricevectors, q : 10);
    writeln(pricevectors, 'change direction' );
```

```
    write(pricevectors, dir : 10);
    writeln(pricevectors, '\end{verbatim}' );
    close (pricevectors);
end ;
```

11 splitcolumn

procedure *splitcolumn* (**var** *src* ,*dest* :*column* ;*dfrac* :*real*);

Given a source column, src and a destination column this transfers the fraction dfrac of the source to the destination, leaving the source with the fraction (1-dfrac) in it.

If the corresponding columns of the U and G matrices are split in the same way, we ensure that the net output of the two daughter commodities will be the same as the net output of the original commodity - assuming all are measured in units of mass kilos for example. We also ensure that if the commodity represented by the src column had a positive net reproduction condition applying, then it will apply to both the src and dest columns afterwards

```
begin
    dest← dfrac × src;
    src← (1 - dfrac) × src;
end ;
```

12 splitrow

procedure *splitrow* (**var** *src* ,*dest* :*commodityvector* ;*sparseness* :*percent*);

This operator splits an industry into two daughter industries in such a way that the net joint product of all commodities produced by the two industries remains unchanged.

It also ensures that the industries can become differentiated by using the sparseness operator to determine whether all of the two industries joint product or joint consumption goes into one industry or into both industries.

```
var
    Let i ∈ commodity;
    Let f ∈ real;
begin
    for i← 1 to numcommodities do
    begin
        if br (sparseness) then
share with both     begin
            f← fr;
            dest_i← f × src_i;
            src_i← (1 - f) × src_i;
        end
        else
            if br (50) then
src gets it all         dest_i← 0
            else
dest gets it all        begin
                dest_i← src_i;
                src_i← 0;
            end ;
    end ;
end ;
```