genkantor.pas

November 27, 2017

Contents

1 genkantor

What follows is a program written in Vector Pascal. The text in roman font that follows are comments. The program code is generally in san-serif font, and the whole, is in the literate programming output format generated by the compiler.

program *genKantor* ;

* The programme has as its objective to generalise the algorithm of Kantorovich that he first presents in the context of excavators working on soils of different types, so that it will work for a general plan optimisation for the whole economy by his method of resolving multipliers. It starts out from the data provided in an i/o Table. In the table the *norms* for for different production techniques are shown in italics. In this context a norm means the expected output per year of the technique. In the case of Kantorovich's original algorithm the norms were represented as a matrix of outputs of different soils and machines. In an input output table a technique is conventionally represented by a column (Leontief notation) or a row (Sraffa notation) of a matrix.

We can convert the Kantorovich table

$$\begin{matrix} 105 & 107 & 64 \\ 56 & 66 & 38 \\ 56 & 83 & 53 \end{matrix}$$

Where each row corresponds to a soil type and each column to an excavator into an extended Sraffa format as

$$\begin{matrix} 105 & 0 & 0 & -1 & 0 & 0 \\ 107 & 0 & 0 & 0 & -1 & 0 \\ 64 & 0 & 0 & 0 & 0 & -1 \\ 0 & 56 & 0 & -1 & 0 & 0 \\ 0 & 66 & 0 & 0 & -1 & 0 \\ 0 & 38 & 0 & 0 & 0 & -1 \\ 0 & 0 & 56 & -1 & 0 & 0 \\ 0 & 0 & 83 & 0 & -1 & 0 \\ 0 & 0 & 53 & 0 & 0 & -1 \end{matrix}$$

where the first 3 columns correspond to the outputs of different soils and the next 3 to the outputs of excavators. The outputs of excavators are negative since a given process uses up an excavator.

The objective was given, by Kantorovich in a final column of his table, in our format we would have a row vector of final outputs, and a second row vector of resources available at the start of the period.

To copy Kantorovich target of 20000 meters of soil for each type of soil we would have a target row vector

$$\begin{matrix} 20000 & 20000 & 20000 & 0 & 0 & 0 \end{matrix}$$

And an initial resource vector of

0 0 0 1 1 1

indicating we have 1 machine of each type. Following the columns of the norms Kantorovich gives the optimal allocation of machine times to activities to minimise overall time taken to do the digging. The program will reproduce this result by applying his method of resolving multipliers or objectively determined valuations. We first introduce our domain of discourse : the types of soil, the types of machine and the units of measurement we are using.

```
type
    soil = ( I ,II ,III );
    excavator = ( A ,B ,C );
    units =( hr , meter );
```

Now we introduce the dimensions in which volume, time and norms are specified. For instance norms are real numbers denoting cubic meter per hour. The word pow in what follows means raised to the power, and * is the multiplication operator.

```
type
    volume = real of meter pow 3;
    duration = real of hr ;
    norm = real of meter pow 3 * hr pow -1;
const
    hour : duration =1.0;
    cubicmeter : volume =1.0;
    epsilon =0.001;
```

The production norms for the machines working on each kind of soil and targets for soil to be moved are copied from Kantrovichs Table 5 and stored in an appropriate matrix called norms, and a vector called targets.

```
const
    norms : array [soil ,excavator ] of norm =
    ( ( 105, 107, 64),
    ( 56, 66,38),
    ( 56, 83, 53));
    identity: array [soil ,soil ] of real =
    ( ( 1, 0, 0),
    ( 0, 1, 0),
    ( 0, 0, 1));
    target : volume = 20000 ;
```

We now introduce the variables of the problem: a matrix x which will encode the time each machine spends on each type of soil; L, a vector of objectively determined valuations of different soils. The standardised output of each machine for each soil type is obtained by applying resolving multipliers to the soil types.

```
var
    x: array [soil ,excavator ] of duration ;
    Let dx ∈ duration;
resolving multipliers
    L : array [soil ] of real ;
    standardisedoutput: array [ excavator ,soil ] of norm ;
    outputs : array [soil ] of norm ;
    totals: array [ soil ] of volume ;
    Let ok ∈ boolean;
    Let greatestsoil, leastsoil, deltam ∈ volume;
    Let best ∈ norm;
    Let e, Scoop ∈ excavator;
    Let s, least, m, j ∈ soil;
    Let λ, f ∈ real;
    Let count ∈ integer;
    equated: array [soil ] of real ;
procedure ComputeTotalsEtc ; (see Section 2 )
function marginalgain ( s :soil ; d :excavator ):volume ; (see Section 3 )
function mainSoilProducedBy ( e :excavator ):soil ; (see Section 4 )

function findScoop ( var s :soil ):excavator ; (see Section 5 )

Kantorovich algorithm
begin
initialise multipliers
    L← 1;
    ok← false;
    count← 0;
    f← 0.3;
```

Iterate the following steps until we have a satisfactory answer.

```
    while not ok do
    begin
default assumption
        x← 0 × hour;
```

Use the L to get a standardised performance for each machine.

```
        standardisedoutput← (norms × L T)T;
```

For each machine find the soil for which it has the best performance

```
for e← A to C do
begin
```

find the best performance of the machine on any soil

```
    outputs← standardisedoutput_e;
    best← \max outputs ;
```

set each machine to work on the soil it is best at

```
    for s← I to III do
        if standardisedoutput_{e,s} = best then x_{s,e}← hour;
end ;
totals← ∑ (norms × x) ;
greatestsoil← \max totals ;
leastsoil← \min totals ;
if leastsoil ≤ 0.0 × cubicmeter then
begin
```

check if any soil has a zero output and raise its value if it has

```
    for s← I to III do
        if totals_s < greatestsoil then
    end
    else ok← true;
end ;
count← 0;
```

At this point our estimate of the resolving multipliers is accurate enough to ensure that some of each soil is now being moved, but we have not yet met the requirement that the same amount of each soil must be moved. We now try to get a more precise estimate of the resolving multipliers and in the process we adjust the amounts of each soil being moved. It is important to note at this point that any further adjustments must come by de-specialising some of the excavators so that they move more than one soil type. The resolving multipliers have until now been used to weight the outputs of different soil types in order to assign each digger to the soil it is best suited to. If a machine is no longer specialised, that is if it moves more than one soil, then the weights must be such that it is no longer *best* at one particular soil type. The multipliers must be set so that the marginal weighted output of the excavator on any of the soils on which it is employed are the same. Thus if a machine k is employed on two soils i, j then standardisedoutput[k,j]=standarisedoutput[k,i].

In turn this implies that for any machine that is employed to move two soils the ratio of the resolving multipliers must be the inverse of the ratio of the norms.

The algorithm will work soil a time bringing ever more soil outputs into equality. We define the set of soils whose outputs has been brought into equality as the equated set.

For those soils in the equated set, the resolving multipliers of the soils will have been corrected so that for any machine moving more than one soil they stand in inverse ratio to that digger's norms.

computeTotalsEtc;
repeat

Find which machine non in the equated set is 2nd best at producing this soil under current resolving multipliers. Call this machine Scoop.

Scoop← *findScoop* (*least*);

m← *mainSoilProducedBy* (*Scoop*);

Adjust the resolving multiplier ratio between Scoops soil and the least produced soil to ratio of Scoops norms.

$L_m \leftarrow \frac{L_{least} \times norms_{least,scoop}}{norms_{m,scoop}}$;

It is now necessary to reduce the ouput of scoop on scoops soil and increase it on the least produced soil. It is necessary to compute how much to reduce scoops soil by. The resolving multipliers give us substitution ratios between different soil outputs. Suppose that we want to reduce output of soil m by one unit and increase the output of soil j, the increase in j we get is

$$\Delta_j = \frac{L_m}{L_j}$$

. If we want to reduce the output of i and we have two other soils j, k which we want to increase equally then we have

$$\Delta_k = \Delta_j$$

where Δ_x means change in x and

$$-\Delta_m L_m = \Delta_k L_k + \Delta_j L_j = \Delta_k (L_j + L_k)$$

. Thus

$$\Delta_m = -\Delta_k \frac{L_j + L_k}{L_m}$$

Let C_m, C_j, C_k be the current outputs of each soil; given that $C_j = C_k$ we have to chose the Δs so that

$$C_m + \Delta_m = C_j + \Delta_j = C_k + \Delta_k$$

It follows that

$$C_m - \Delta_k \frac{L_j + L_k}{L_m} = C_k + \Delta_k$$

and

$$C_m - C_k = \Delta_k \frac{L_j + L_k}{L_m} + \Delta_k = \Delta_k (1 + \frac{L_j + L_k}{L_m})$$

so

$$\Delta_j = \frac{C_m - C_j}{1 + \frac{L_j + L_k}{L_m}}$$

We next compute the reduction to be made in soil m from the formula

$$\Delta_m = -\Delta_k \frac{L_j + L_k}{L_m}$$

substitiuting we get

$$\Delta_m = -\frac{C_m - C_j}{1 + \frac{L_j + L_k}{L_m}} (\frac{L_j + L_k}{L_m})$$

Translating this to the variables used in the program we have:

$j \leftarrow least$;
$\lambda \leftarrow \frac{L.equated}{L_m}$;
$deltam \leftarrow \frac{(-1) \times \lambda \times (totals_m - totals_j)}{1 + \lambda}$;

Note that in the line above we are generalising the term $L_j + L_k$ to an aribtrary number of multipliers (1 or 2 in this program) by computing the inner product between the equated vector and the multipliers. This works because the equated vector has a 1 for all soils in the equated set. We now compute the change in duration that Scoop spends on its best soil (dx) by scaling deltam by Scoops norm for soil m.

$dx \leftarrow \frac{deltam}{norms_{m,Scoop}}$;
$x_{m,Scoop} \leftarrow x_{m,Scoop} + dx$;

reallocate this time to Scoops best soil in the equated set which we will now call j

$best \leftarrow norms_{I,Scoop} \times 0;$
$j \leftarrow I;$
for $s \leftarrow I$ **to** III **do**
  if $norms_{s,Scoop} \times equated_s > best$ **then**
  begin
    $best \leftarrow norms_{s,Scoop};$
    $j \leftarrow s;$
  end ;

  $x_{j,Scoop} \leftarrow x_{j,Scoop}$ - $dx;$
  $computeTotalsEtc;$
  $count \leftarrow count + 1;$
until $((\sum equated) = 3) \vee (count > 10);$
writeln('answer arrived at after ' , $count$, ' trys');
writeln(' allocation ' , $\frac{x \times target/totals_I}{hour}$);

Kantorovich algorithm

end .

2 ComputeTotalsEtc

procedure *ComputeTotalsEtc* ;

Work out how much is being produced, which soil is being produced least and which soils outputs are equals to this,

var
  d : **array** [*soil*] **of** *real* ;
begin
  $totals \leftarrow \sum (norms \times x)$;
  $leastsoil \leftarrow \backslash min \ totals$;

Find the soil that is least produced.

  for $s \leftarrow I$ **to** III **do**
    if $totals_S = leastsoil$ **then** $least \leftarrow s;$

Find the ones on the plan ray

  $d \leftarrow \frac{totals - leastsoil}{cubicmeter};$
  $equated \leftarrow \begin{cases} 1.0 & \text{if } (d < \epsilon) \wedge (d > \text{- } \epsilon) \\ 0.0 & \text{otherwise} \end{cases}$;
end ;

3 marginalgain

function *marginalgain (s* :*soil* ; *d* :*excavator*):*volume* ;

This computes the marginal gain, under the weighting imposed by the current resolving multipliers, of a small shift of the digger d's time to the specified soil type s. We compute the effect of multiplying all current time allocations to $1-\epsilon$ whilst increasing the allocation of time to soil s by ϵ. The assumption here is that for now each machine has only one hour to allocate.

```
const
    epsilon =0.001;
var
    Let currentoutput ∈ volume;
marginal gain
begin
    currentoutput← ∑ x_d × norms_{ι0,d} ;
    marginalgain← ((ϵ × hour) × norms_{s,d}) - ϵ × currentoutput;
end ;
```

4 mainSoilProducedBy

function *mainSoilProducedBy (e* :*excavator*):*soil* ;

This determines which soil excavator e produces the most of.

```
var
    Let v ∈ volume;
    Let j, s ∈ soil;
begin
    v← 0 × cubicmeter;
    s← i;
    for j← I to III do
    begin
        if v < norms_{j,e} × x_{j,e} then
        begin
            v← norms_{j,e} × x_{j,e};
            s← j;
        end ;
    end ;
    mainSoilProducedBy← s;
end ;
```

5 findScoop

function *findScoop (var s* :*soil*):*excavator* ;

Find which machine not currently fully commited is best at producing this soil. Call this machine Scoop.

find Scoop

```
var
    Let gain ∈ volume;
    Let j, m ∈ soil;
    Let d, Scoop ∈ excavator;
begin
    Scoop← A;
    gain← ( - maxint ) × cubicmeter;
    for d← A to C do
        for j← I to III do
            if equated_j > 0 then
            begin
                m← mainsoilproducedby (d);
                if marginalgain (j, d) > gain then
                begin
                    if equated_m < 1 then
                    begin
                        gain← marginalgain (j, d);
                        Scoop← d;
                        s← j;
                    end ;
                end ;
            end ;
            findScoop← Scoop;
end ;
```