Returning to Kantorovich

Paul Cockshott

January 21, 2018

In this post I focus on recovering the work of Kantorovich the only Soviet Economic Nobelist, and showing that his work provided a fundamental theoretical response to the Austrian school economist von Mises. Kantorovich was an eminent mathematician, whose work went well beyond economics, but in this article we focus only on his economic contributions. In explaining these I reproduce in section 1 some of his original numerical examples drawn from his experience in Soviet heavy industry. I have avoided, however, giving any detailed presentation of the mathematical techniques (algorithms) that Kantorovich and Dantzig developed, both because we assume that the readership are not specialists in linear algebra, and secondly because these techniques have now been packaged up in open-source software that can be used by non-algebraists. I give in sections 1 and 1.1 what is essentially a tutorial introduction to using such package to solve planning problems. We summarise what these mathematical techniques mean in practical terms. What type of economic problem do they allow us to solve?

As illustrations I show how Kantorovich allows us to pose problems of national or continental environmental trade-offs.

In Section 3 and beyond I provide a detailed commented source listing of a programme that applies Kantorovich method. The programme is available in source and in Linux binary format from the directory pointed to by the following link:

https://drive.google.com/drive/folders/1CBkWwqdO1OT0y2kcZ04JI45oulgt-Pn6?usp=sharing

With this programme you can experiment with macroeconomic planning problems. You need an input output matrix for an economy and a set of initial resources and a vector of plan objectives - what Kantorovich called a planray. It will then print Kantrovich's objective valuations and the achievable gross output given the available resources.

The illustrations I give in the course of the post rely on the open source linear programming package lp-solve which is available for all Linux systems. I also provide my own programme in order to show the technique used by Kantorovich and to provide the Objective Valuations that he suggested be used in socialist calculation.

In the process of developing this programme I found that there were certain limitations to the Kantorovich's own technique for deriving ODVs when you applied them to data in input output table format. I have therefore introduced some changes to his technique which are described in section 2.1 .

Table 1: Kantorovich's first example.

Type of machine	# machines	output per machine		Total output	
		As	Bs	As	Bs
Milling machines	3	10	20	30	60
Turret lathes	3	20	30	60	90
Automatic turret lathes	1	30	80	30	80
Max total				120	230

1 Kantorovich's method

In the early 30s, no algorithmic techniques were known which would solve the general problem of socialist calculation where there can be joint production and multiple possible techniques to produce individual products. But in 1939 [Kan60] the Soviet mathematician V Kantorovich came up with a method which later came to be known as *linear programming* or *linear optimisation,* for which he was later awarded both Stalin and Nobel prizes. Describing his discovery he wrote:

> I discovered that a whole range of problems of the most diverse character relating to the scientific organization of production (questions of the optimum distribution of the work of machines and mechanisms, the minimization of scrap, the best utilization of raw materials and local materials, fuel, transportation, and so on) lead to the formulation of a single group of mathematical problems (extremal problems). These problems are not directly comparable to problems considered in mathematical analysis. It is more correct to say that they are formally similar, and even turn out to be formally very simple, but the process of solving them with which one is faced [i.e., by mathematical analysis] is practically completely unusable, since it requires the solution of tens of thousands or even millions of systems of equations for completion.
>
> I have succeeded in finding a comparatively simple general method of solving this group of problems which is applicable to all the problems I have mentioned, and is sufficiently simple and effective for their solution to be made completely achievable under practical conditions. ([Kan60], p. 368)

What was significant about Kantorovich's work was that he showed that it was possible, starting out from a description in purely physical terms of the various production techniques available, to use a determinate mathematical procedure to determine which combination of techniques will best meet plan targets. He indirectly challenged von Mises[1], both by proving that in-natura calculation is possible, and by showing that there can be a non monetary scalar objective function : the degree to which plan targets are met.

[1] There is no indication that he was aware of von Mises at the time.

Table 2: Kantorovich's examples of output assignments.

Type of machine	Simple solution		Best solution	
	As	Bs	As	Bs
Milling machines	20	20	26	6
Turret lathes	36	36	60	0
Automatic turret lathes	21	21	0	80
Total	77	77	86	86

The practical problems with which he was concerned came up whilst working in the plywood industry. He wanted to determine the most effective way of utilising a set of machines to maximise output. Suppose we are making a final product that requires two components, an A and a B. Altogether these must be supplied in equal numbers. We also have three types of machines whose productivities are shown in the Table 1.

Suppose we set each machine to produce equal numbers of As and Bs. The three milling machines can produce 30 As per hour or 60 Bs per hour. If the three machines produce As for 40 mins in the hour and Bs for 20 mins then they can produce 20 of each. Applying similar divisions of time we can produce 36 As and Bs on the Turret lathes and 21 As and Bs on the automatic turret lathe (Table 2).

But Kantorovich goes on to show that this assignment of machines is not the best. If we assign the automatic lathe to producing only Bs, the turret lathe to producing only As and split the time of the milling machines so that they spend 6 mins per hour producing Bs and the rest producing As, the total output per hour rises from 77 As and Bs to 86 As and Bs.

The key concept here is that each machine should be preferentially assigned to producing the part for which it is relatively most efficient. The relative efficiency of producing As/Bs of the three machines was milling machine $= \frac{1}{2}$, turret lathes $= \frac{2}{3}$, and automatic lathe $= \frac{3}{8}$. Clearly the turret lathe is relatively most efficient at producing As, the automatic lathe is relatively most efficient at producing Bs and the milling machine stands in between. Thus the automatic lathe is set to produce only Bs, the turret lathes to make only As and the time of the milling machines is split so as to ensure that an equal number of each product is turned out.

The decision process is shown diagrammatically in Figure 1. The key to the construction of the diagram, and to the decision algorithm is to rank the machines in order of their relative productivities. If one does this, one obtains a convex polygon whose line segments represent the different machines. The slopes of the line segments are the relative productivities of the machines. One starts out on the left with the machine that is relatively best at producing Bs, then move through the machines in descending order of relative productivity. Because relative productivity is monotonically decreasing one is guaranteed that the boundary will be convex. One then computes the intersection of the 45 degree line representing equal output of As and Bs with the boundary of this polygon. This intersection point is the optimal way of meeting the plan. The term linear programming stems from the fact that the production functions are represented by straight lines in the case of

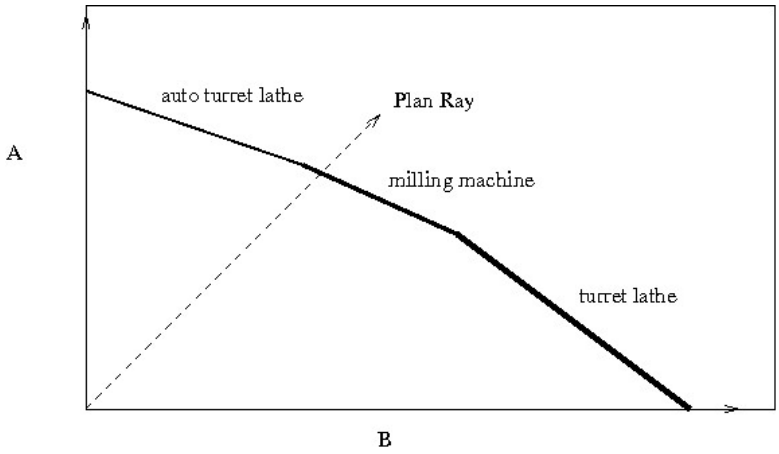


Figure 1: Kantorovich's example as a diagram. The plan ray is the locus all points where the output of As equals the output of Bs. The production possibility frontier is made of straight line segments whose slopes represent the relative productivities of the various machines for the two products. As a whole these make a polygon. The plan objective is best met where the plan ray intersects the boundary of this polygon.

2 products, planes for 3 products, and for the general higher dimensional case by linear functions. That is to say, functions in which variables only appear raised to the power 1.

The slope of the boundary where the plan ray intersects was called by Kantorovich the resolving ratio. Any machine whose slope is less than this should be assigned to produce Bs any machine whose slope is greater, should be assigned to produce As.

When there are only two products being considered, the method is easy and lends itself to diagrammatic representation. But it can handle problems of higher dimensions, involving 3 or more products. In these cases we can not use graphical solutions, but Kantorovich provided an algorithmic by which the resolving ratios for different pairs of outputs could be arrived at by successive approximations. Kantorovich's work was unknown outside of the USSR until the late 50s and prior to that Dantzig had independently developed a similar algorithm for solving linear programming problems, the so called simplex method [Dan63]. This has subsequently been incorporated into freely available software tools[2]. These packages allow you to enter the problem as a set of linear equations or linear inequalities which they then solve.

In the West, linear programming was used to optimise the use of production facilities operating within a capitalist market. This meant that the objective function that was maximised was not a fixed mix of outputs, in Kantorovich's first example equal numbers of parts A and B, but the money that would be obtained from

[2]For example `lp_solve` and `GLPK`.

```
A;

m1<=3;

m2<=3;

m3<=1;

A-B=0;

m1-0.1 x1a - 0.05 x1b=0;

m2-0.05 x2a - 0.033333 x2b=0;

m3- 0.033333 x3a - 0.0125 x3b=0;

x1a+x2a+x3a - A=0;

x1b+x2b+x3b -B =0;

int A;
```

Algorithm 1: Kantorovich's example as equations input to `lp_solve`..

selling the output: price A ×number of As + price B ×number of Bs. Manuals and textbooks produced in association with Western linear programming software assumes this sort of objective. However, as we shall see, one can readily formulate Kantorovich's problem using this sort of software by adding additional equations. We shall now show how you can use the package `lp_solve` to reproduce Kantorovich's solution to his problem.

The program requires that you input an expression to be maximised or minimised followed by a sequence of equations or inequalities. In Algorithm 1 we give Kantorovich's problem in the format that `lp_solve` requires. In this example we use the following variables:

variable	meaning
A	number of units of A produced
B	number of units of B produced
m1	number of milling machines used
m2	number of turret lathes used
m3	number of automatic turret lathes used
xij	number of units of j produced on machine i

Thus x1a means the output of As on milling machines.

The first line of input is the objective function to be maximised. We give this as A, meaning maximise the output of A's. The following lines give the constraints to which the maximisation process is to be subjected.

00.00.0000 [A-B=0]

This is another way of writing that A=B, or that equal quantities of A and B must be produced.

m1<=3

This means that the number of milling machines used must be less than or equal to 3. The characters '<' '=' are used because $\leq$ is not available on computer keyboards. Similar constraints are provided for the other machines.

m1-0.1x1a-0.05x1b=0

This specifies $m1 = 0.1x1a + 0.05x1b = \frac{1}{10}x1a + \frac{1}{20}x1b$ or in words, that allocating a milling machine to produce an A uses $\frac{1}{10}$ of a milling machine hour, and that allocating a milling machine to produce a unit of B uses $\frac{1}{20}$ of a milling machine hour. We provide similar production equations for the other machines.

x1a+x2a+x3a - A=0

This says that the total output of A is equal to the sum of the outputs of A from each of the machines. We provide a similar equation defining the output of B.

Note that all equations have to be provided with variables and constants on the left and a constant on the right. One can readily re-arrange the equations in this form. The last line specifies that the number of units of A produced should be an integer. When the equations are input to lp_solve it produces the answer:

```
Value of objective function: 86

Actual values of the variables:

A           86

B           86

x1a          26

x1b          6

x2a          60

x2b          0

x3a          0

x3b          80
```

```
m1          2.9

m2          3

m3          1
```

which exactly reproduces Kantorovich's own solution (Table 2) arrived at using his algorithm.

1.1 Generalising Kantorovich's approach

In his first example Kantorovich deals with a very simple problem, producing two goods in equal proportions using a small set of machines. He was aware, even in 1939 that the potential applications of mathematical planning were much wider. We will look at two issues that he considered which are important for the more general application of the method.

1. Producing outputs in a definite ratio rather than in strictly equal quantities.
2. Taking into account consumption of raw materials and other inputs.

Suppose that instead of wanting to produce one unit of A for every unit of B, as might be the case if we were matching car engines to car bodies, we want to produce 4 units of A for every unit of B, as would be the case if we were matching wheels to car engines (and ignoring spare wheels). Can Kantorovich's method deal with this as well. Consider Figure 1 again. In that the plan ray is shown at an angle of 45° a slope of 1 to 1. If we drew the plan ray at a slope of 4 to 1, the intersection with the production frontier would provide the solution. Since this geometric approach only works for two products, let us consider the algebraic implications.

You should now be convinced that it is possible to solve Kantorovich's original problem[3] by algebraic means. In Algorithm 1 we specified that $A - B = 0$ or in other words $A = B$, if one wanted 4 units of A for every B we would have to specify $A = 4B$ or, expressing it in the standard form used in linear optimisation, $A - 4B = 0$. Suppose A stands for engines, B stands for wheels. If we now say wheels come in packs of 4, then we can repose the problem in terms of producing equal numbers of packs of wheels and engines. Introduce a new variable $\beta = 4B$ to stand for packs of wheels, and rewrite the equations in terms of β and we can return to an equation specifying the output mix in the form $A - \beta = 0$, which we know to be soluble.

How do we deal with consumption of raw materials or intermediate products?

In our previous example we had variables like `x1b` which stood for the output of product B on machine 1. This was always a positive quantity. Suppose that there is a third good to be considered - electricity, and that each machine consumes electricity at different rate depending on what it is turning out. Call electricity C and introduce new variables `x1ac, x1bc` etc referring to how much electricity is consumed by machine 1 producing outputs A and B. Then add equations specifying

[3] Actually this was his "problem A"

how much electricity is consumed by each machine doing each task, and the model will specify the total amount of electricity consumed.

We now know how to :

1. Use Kantorovich's approach to specify that outputs must be produced in a definite ratios.

2. Use it to take into account consumption of raw materials and other inputs.

If we can do these two tasks, we can in principle perform *in-natura* calculations for an entire planned economy. Given a final output bundle of consumer and investment goods to maximise and given our current resources, a system of linear equations and inequalities can be solved to yield the structure of the plan. From simple beginnings, optimising the output of plywood on different machines, Kantorovich had come up with a mathematical approach which could be extended to the problem of optimising the operation of the economy as a whole.

1.2 A second example

Let us consider a more complicated example, where we have to draw up a plan for a simple economy. We imagine an economy that produces three outputs : energy, food, and machines. The production uses labour, wind and river power, and two types of land: fertile valley land, and poorer highlands. If we build dams to tap hydro power, some fertile land is flooded. Wind power on the other hand, can be produced on hilly land without compromising its use for agriculture. We want to draw up a plan that will make the most rational use of our scarce resources of people, rivers and land.

In order to plan rationally, we must know what the composition of the final output is to be - Kantorovich's ray. For simplicity we will assume that final consumption is to be made up of food and energy, and that we want to consume these in the ratio 3 units of food per unit of energy. We also need to provide equations relating to the productivities of our various technologies and the total resources available to us.

Valleys are more fertile. When we grow food in a valleys, each valley requires 10,000 workers and 1000 machines and 20,000 units of energy to produce 50,000 units of food. If we grow food on high land, then each area of high land produces only 20,000 units of food using 10,000 workers, 800 machines and 10,000 units of energy.

Electricity can be produced in two ways. A dam produces 60000 units of energy, using one valley and 100 workers and 80 machines. A windmill produces 500 units of electricity, using 4 workers and 6 machines, but the land on which it is sited can still be used for farming.

We will assume that machine production uses 20 units of electricity and 10 workers per machine produced.

Finally we are constrained by the total workforce, which we shall assume to be 104,000 people.

Table 3: Variables in the example economy

e	total energy output
e_c	household energy consumption
f	food
v	valleys
w	windmills
m	machines
d	dams
u	undammed valleys
h	highland
f_h	food produced on high land
f_v	food produced in valleys

Table 4: Resource constraints and productivities in our example economy

final output mix	$f = 3e_c$
number of valleys	$v = 4$
dams use valleys	$v - u = d$
valley food output	$f_v = 50000u$
valley farm labour	$l_v = 10000u$
valley energy use	$e_v = 20000u$
valley farm machines	$m_v = 1000u$
highland food prod	$f_h = 20000h$
highland farm labour	$l_h = 10000h$
highland energy use	$e_h = 10000h$
highland farm machines	$m_h = 800h$
energy production	$e = 500w + 60000d$
energy workers	$l_e = 100d + 4w$
machines in energy prod	$m_e = 80d + 6w$
workers making machines	$l_m = 10m$
energy used to make machines	$e_m = 20m$
energy consumption	$e_m + e_v + e_h + e_c \leq e$
machine use	$m_e + m_h + m_v \leq m$
total food prod	$f = f_h + f_v$
workforce	$l_m + l_e + l_v + l_h \leq 104000$

Table 5: Economic plan for the example economy using lp_solve

d (dams)	0
e	270500
f	200218
h	0.0108889
m	6172.71
u	4
v	4
w (windmills)	541
e_c	66739.3
e_h	108.889
e_m	123454
e_v	80000
f_h	217.778
f_v	200000
l_e	2164
l_h	108.889
l_m	61727.1
l_v	40000
m_e	2164
m_h	8.71111
m_v	4000

Tables 3 and 4 show how to express the constraints on the economy and the plan in equational form. If we feed these into `lp_solve` we obtain the plan shown in Table 5. The equation solver shows that the plan targets can best be met by building no dams, generating all electricity using 541 windmills, and devoting the river valleys to agriculture.

It also shows how labour should be best allocated between activities: 40000 people should be employed in agriculture in the valleys, 109 people should work as farmers in the highlands, 2164 people should work on energy production, and 61727 people should work building machines.

The results that we have obtained were by no means obvious at the outset. It was not initially clear that it would be better to use all the river valleys for agriculture rather than building dams on some of them. In fact, whether dams or windmills are preferred turns out to depend on the whole system, not just on their individual rates of producing electricity. We can illustrate this by considering what happens if we cut the labour supply in half to 52000 people?

If we feed this constraint in to the system of equations we find the optimal use of resources has changed. The plan now involves 1 dam and 159 windmills. Cut the working population slightly further, down to 50000 people and the optimal plan involves flooding two valleys with dams and building only 23 windmills. Why?

As the population is reduced, there are no longer enough people available to both farm the valleys and produce agricultural machinery. Under these circum-

stances the higher fertility of lowland valleys is of no importance, it is better to use one or more of them to generate electricity. By applying Kantorovich's approach it becomes possible for a socialist plan to do two things that von Mises had believed impossible:

1. It allows the plan to take into account natural resource constraints - in this case the shortage of land in river valleys which can be put to alternative uses.

2. It allows rational choices to be made between different technologies - in this case between windmills and hydro power and between lowland and highland agriculture.

Contrary to what von Mises claimed, the whole calculation can be done in physical units without any recourse to money or to prices.

2 Valuation

The core of Mises's argument relates to the use of prices to arrive at a rational use of intermediate or capital goods. Mises argues that, in practice, only money prices will do for this, but concedes that, in principle, other systems of valuation, such as labour values would also be applicable. Kantorovich too, was very concerned with the problem of relative valuation[Kan65], and developed what he called *objectively determined valuations* (ODV). These valuations differed from prices, since a price involves an exchange of commodities for money between two owners. In the USSR all factories and all products were owned by the state. When products moved from one factory to another, there was no sale or purchase involved. The ODVs were purely notional numbers, used in economic calculations, not selling prices.

He considered a situation where planners have to deal with several different types of factories ($A..E$) each able to produce products 1 and 2, and where the intended ratio of output of product 1 and 2 are fixed in the plan. Each class of factory $A..E$ has different relative productivities for the two products.

He next looked at the apparent profitability of producing products 1 and 2 under different relative valuations. Under some schemes of relative price, all factories would find product 1 to be unprofitable relative to product 2, under other the reverse would occur. Intermediate price schemes would allow both products to be produced, with some classes of factories specializing on 1 and others on 2. He gives the example of children's clothing as something which, under the administratively determined valuations then used in the USSR, were unprofitable to produce, and unless factories were specifically instructed to ignore local profitability, too few children's clothes would be made.

He asks if there exists a relative valuation structure which would allow factories to concentrate on the most valuable output, and at the same time meet the specified plan targets and arrives at certain conclusions:

1. That among the very large number of possible plans there is always an optimal one which maximises output of plan goals with current resources.

2. That in the optimal plan there exists a set *objectively determined valuations* (ODV) of goods which will ensure that each factory

   (a) produces the output which will contribute most to maximising the plan goals

   (b) each factory also finds that the output which contributes to maximising plan targets is also the output which is most profitable

3. With arbitrary valuations which differ from ODV, these conditions can not be met, and profit maximising factories will not specialise in a way that meets plan goals optimally.

It is important to understand that his ODVs are valuations that apply only for a plan which optimally meets a specific plan target. Kantorovich's procedure for arriving at an optimal plan involved successive adjustments to the ODVs and factory specialisation until both the appropriate mix of goods is reached, and at the same time each factory is producing its most profitable good. He actually gave several different mathematical procedures for arriving at such a plan and system of ODVs.

Although Kantorovich asserts that labour is ultimately the only source of value, his ODV's are short term valuations and differ from the classical labour theory of value, which gave valuations in terms of the long term labour reproduction costs of goods - including the reproduction costs of capital goods. Kantorovich, in contrast, is concerned with valuations which should apply with the current stock of means of production and labour resources. For example, he considers the situation of giving a valuation to electric power relative to labour. Instead of valuing it in terms of the labour required to produce electricity, he first assumes that the total electrical power available is fixed - ie, power-stations operating at full capacity, and then works out how many person hours of labour is saved by using an additional kilowatt hour of electricity. The assumption is made that in order to arrive at this objective valuation of electricity in terms of labour

1. The plan targets must be met

2. The plan must be optimal

Kantorovich's insistence on considering short term, very material constraints - so many megawatts of power, such and such a number of cutting machines, etc, gives his work an intensely practical and pragmatic character, quite different from that of most theoretical economists.

2.1 Alterations made to Kantorovich's algorithm

Besides the alterations to the input data structures that I will describe in Section 7, I found that there were 3 other areas that needed improvement.

1. Selection of a plan ray where the outputs produced are, unlike his soil example, unequal. This is easily dealt with by selecting units of measurement for the outputs so as to make all outputs equal. Thus if you were to need twice

as much of soil A as soil B, you would express your volume of soil A in units of 2 cubic meters not 1 cubic meter. This can be effected by an appropriate predivision of the resource vector and the technology matrix.

2. It was noticed that on certain problems the first phase of the algorithm to obtain resolving multipliers (Section 23), fails to terminate. This turns out to be because all resources are allocated to the technique for which they are 'best' under the valuation given by the resolving multipliers. This greedy algorithm can result in the sytem never terminating under some circumstances. When a given technique uses multiple resources, as is typical with IO table examples, the system can get trapped in an oscillation in which all resources are allocated alternately to two disjoint set of techniques, leaving some goods unproduced. I altered this rule to one in which half of the available resources were allocated to the 'best' line of production, leaving some free for other techniques.

3. Kantorovich's optimisation procedure works very well for his own examples, but fails to converge for some input output table examples. I provide an option to use a different optimizer that also works on gradient descent, but with simulated annealing added. This proves more robust than Kantorovich's method, albeit slower than his on the examples where his method works. You select which optimiser to use by setting the `optimiser` flag in the `kantor` library. For the programme `objectiveValuation` the flag is set to use my optimiser.

3 The programme objectiveValuations

program *objectiveValuation* ;
uses *kantor* ,*csvfilereader* ;

4 Purpose

The programme applies the Objective Valuation technique of the Soviet Nobelist Kantorovich to data from input output tables. Kantorovich derived a set of numbers, which he termed Objectively Determined Valuations, from a combination of

1. the technology available
2. the objectives of the socialist plan
3. the available stock of material resources

This programme can potentially be used in studies of the comparative efficiency of labour values, prices of production or Kantorovich valuations in economic planning.

5 Comand line interface

Usage: `objectiveValuation iotable.csv planray.csv resourcevec.csv > result.csv`

The files should be laid out with a first line and first column made up of text fields labeling the rows and columns. All other cells should be numeric. Discussion of matrix sizes in what follows refers exclusively to the rectangular subarray of numeric values.

5.1 iotable

The first file, the iotable one, should contain an N by M io table in standard column form, with a column corresponding to an industry so that cell at row i col j contains the amount of output from the ith industry used by the jth industry.

The last line must contain the outputs of each industry.

5.2 planray

This contains the target level of gross outputs in the next period to be obtained using the technology of the iotable. In line with Kantorovich practice this corresponds to a ray in N-1 dimensional space along which the outputs are supplied in fixed proportions. This is also sometimes called a Leontief demand function. The format, excluding headers, is a 1 by N-1 matrix, or N-1 element row vector. For a typical io table this implies that even though labour is not a produced input it should appear as a zero output target in the plan.

Headers should correspond to the column headers of the IO table.

5.3 reseourcevec

This is a N-1 row vector of available resources, the headers should correspond to the row titles of the IO table.

5.4 results

The output is a 2 by M matrix in csv format. The two rows encode the intensity with which each column technology in the IO table must be operated and the objective valuations associated with each output.

```
type
   pmat =^ matrix ;
   channel = record
      p : pcsv;
      r;
      m : ↑ matrix ;
   end ;
procedure rf ( var ch :channel ;param :integer ); (see Section 6 )
var
   Let iot, rv, pr, results ∈ channel;
   Let tech ∈ ^technologytable;
   Let odvs ∈ ^ODVvec;
   Let X ∈ ^intensityvec;
   Let target, startingresource ∈ ^resourcevec;
   Let outputs ∈ ^resourcevec;
   Let i, j ∈ integer;
begin
   rf (iot, 1);
   rf (pr, 2);
   rf (rv, 3);
```

omit last row and allow for the base of 2nd dim of techniques to be 0

```
   new ( tech ,iot .m ^ .cols ,iot .m ^ .rows -2);
   tech↑← 0;

   for i← 1 to tech↑.ω do
   begin
      for j← agriculture to tech↑.amusements do
```

```
    begin
      tech↑[i, src, j]← - iot.m↑[j + 1, i] ;
    end ;
    tech↑[i, dest, i - 1]← iot.m↑[iot.m↑.rows, i];
end ;

new ( target ,pr .m ^ .cols -1);
target↑← pr.m↑[1, 1 + ι0];
new ( startingresource ,rv .m ^ .cols -1);
new ( odvs ,rv .m ^ .cols -1);
new ( X ,iot .m ^ .cols );
startingresource↑← rv.m↑[1, 1 + ι0];
optimiser← pauls;
kantorovichMethod ( tech ^ ,target ^ ,
startingresource ^ ,
odvs ^ , X ^ );

for i← 1 to iot.m↑.cols do
  write ( ',' ,iot .c ^ [i ]^ .textual ^ );
writeln;
write( 'output' );
for i← 1 to iot.m↑.cols do
  write ( ',' ,x ^ [i ]* iot .m ^ [iot .m ^ .rows ,i ]:1:4);
writeln;
write( 'odv' );
for i← 1 to iot.m↑.cols do
  write ( ',' ,odvs ^ [i -1]:1:4);
writeln;

end .
```

inputs go in transposed with minus sign

outputs from last row of io matrix

there is to be an odv for each resource

there is an intensity for each production technique, thus 1 for each column in the io table

6 rf

```
procedure rf ( var ch :channel ;param :integer );
```

Read in one of the file parameters and extract the data from it

```
begin
  with ch do
  begin
    p← parsecsvfile (paramstr (param));
    if p = nil then
    begin
      writeln( 'error opening or parsing file ' , paramstr (param));
      halt (2);
    end
    else ;
    r← getrowheaders (p);
    c← getcolheaders (p);
    m← getdatamatrix (p);
  end ;
end ;
```

7 kantor

What follows is a library 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.

```
unit Kantor ;
```

* The library provides a procedure which generalises the algorithm of Kantorovich that he first presented 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 *techniques* for for different production techniques are shown in italics. In this context a technique means the expected output per year of the technique. In the case of Kantorovich's original algorithm the techniques 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.

In practice we split this into two distinct tables one the `dest` table which lists the outputs and the other the `src` table which lists the inputs. Thus:

`dest`

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

src

$$\begin{matrix} 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 \end{matrix}$$

Why reorganise like this?

Because the general economic planning problem involves the production and consumption of many of the same goods. Energy is both used and produced, for example. It is thus convenient to show the inputs and outputs in two different tables.

The pair composed of a destination row and the corresponding src row we term a technique.

In principle we could have a problem with joint production, but for now we are sticking to the simpler single product per technique problem. Thus each row of the destination column contains only one non zero element.

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

Kantorovich's algorithm demanded equal quantities of all outputs which is not the general case. We keep the Kantorovich approach within this library, which means that the units of measurement of all inputs must be scaled in order to result in an output that is the same, in these units of measurement, for all products.

Thus if we wanted to produce 2 cubic meters of soil type 1 relative to every cubic meter of soil type 2 and 3 we use 2 cubic meter as the standard unit for 1.

We handle this by dividing the appropriate column of the technology matrix by 2 and similarly scaling any corresponding entries for the available resources. More generally, given a targets vector we divide all other row vectors in the problem by this.

20000 20000 20000 0 0 0

And an initial resource vector of

0 0 0 1 1 1

indicating we have 1 machine of each type.

My first step in developing the programme in this new format was to take the programme that I had previously published in Intervention, and to reformulate it in this more general form. The aim is to check that given the new data format the algorithm still gives the same answers as Kantorovich. Following on from that we can provide a more general version that will work on arbitrary sized input output tables.

Following the columns of the techniques 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 resources, techniques and units of usage and output. We simply number the resources 0 to 5 with resources in this case including soil and machines.

```
interface
  const
    alpha =1;
```

```
    epsilon =0.001;
    agriculture =0;
    dest =0;
    src =1;
    growthconst =0.1;
    annealrate =0.99;
type
    optimizer =( kantorovichs ,pauls );
    flow = dest ..src ;
    resource = integer ;
    units =real ;
    intensity = real ;
    ODV = real ;

    technique = integer ;
    resourcevec(amusements:resource ) =  array [agriculture ..amusements ]  of  units ;
```

first entry in US IO table is agriculture

Kantorovich's Objectively Determined Valuations

last entry of US IO table is amusements

A technology is defined as a flow of src inputs to a flow of dest outputs, with each flow being a vector.

```
    technologytable(omega,amusements:integer ) =  array  [alpha .. omega ,dest ..src ,agriculture ..
    amusements ]  of  units ;
    ODVvec =resourcevec ;

    intensityvec =vector ;
    volume =real ;
var
    Let growthfactor ∈ real;
    Let optimiser ∈ optimizer;
procedure kantorovichMethod ( var techniques :technologytable ; var target ,startingresource :resourcevec ; var L :
```

The method must be supplied with a technology table and a vector of targets and available resources. It outputs a vector of resolving multipliers L and an vector of technique intensities X which specify how much each technique will be used.

7.1 Implementation

```
implementation
```

```
type
  productionIndex (omega:integer )= array [alpha .. omega ] of  resource ;
  pproductionIndex = ^ productionIndex ;

procedure IndexedMethod ( var produces :productionindex ;var techniques :technologytable ; var targets ,startingre



procedure kantorovichMethod ( var techniques :technologytable ; var target ,startingresource ,L :ODVvec ; var X :in

unit initialisation
begin
  growthfactor← growthconst;
  optimiser← kantorovichs;
unit kantor
end .
```

8 IndexedMethod

procedure *IndexedMethod (**var** produces :productionindex ;**var** techniques :technologytable ; **var** targets ,startingresource ,L :ODVvec ; **var** X :intensityvec);*

The indexed method is a nested procedure which will work using an index of what each technique produces. Many techniques may produce same output.

The algorithm manipulates a vector x which will encode the time for which each technique is operated, this corresponds in Kantorovich's case to the time a given type of machine spends on a given soil. Later it may have a more general impliation. Next L, a vector of objectively determined valuations of resources[4].

The standardised output of each technique is obtained by applying resolving multipliers to the outputs types it produces.

```
type
```

[4] Presumably Kantorovich labels his multipliers L because of their similarity to Lagrangian multipliers

outputmatrix(omega,amusements:*integer*) = **array** [*alpha .. omega ,agriculture .. amusements*] **of** *units* ;

var

Let *targetintensity* ∈ real;
Let *achievableintensities* ∈ ˆresourcevec;
Let *dx* ∈ intensity;
Let *standardisedoutput* ∈ ˆ outputmatrix;
Let *available* ∈ r̂esourcevec;
Let *outputs* ∈ întensityvec;
Let *totals* ∈ ôdvvec;
Let *ok* ∈ boolean;
Let *greatestProducedOutput, leastProducedOutput, deltam* ∈ real;
Let *best* ∈ real;
e , e2 ,Scoop ,

first input resource — Let *A* ∈ resource;
Let *r, s, j* ∈ resource;

Output weighted by L — Let *weightedTotalOutput* ∈ real;
Let *λ, f, t* ∈ real;
Let *least, m, jt, st* ∈ technique;
Let *count, k* ∈ integer;
Let *equated* ∈ ôdvvec;

function *findfirstinput* :*resource* ; (see Section 10)

procedure *claimHeapSpace* ; (see Section 11)
procedure *freeheapspace* ; (see Section 12)

function *marginalgain (s* :*technique* ; *d* :*resource*):*volume* ; (see Section 13)

function *mpp (s* :*technique* ; *d* :*resource*):*volume* ; (see Section 14)

function *mostProductiveTechniqueFor (e* :*resource*):*technique* ; (see Section 15)

procedure *ComputeTotalsEtc* ; (see Section 16)

function *somePositiveTargetsStillZero* :*boolean* ; (see Section 17)
function *numpostargets* :*integer* ; (see Section 18)
procedure *renormaliseMultipliers* ; (see Section 19)
procedure *LowerNonEquatedMultipliers* ; (see Section 20)
procedure *LowerNonEquatedIntensities* ; (see Section 21)
procedure *getavailability* ; (see Section 22)
procedure *RedistributeResources* ; (see Section 23)
procedure *rescaleall* ; (see Section 24)

procedure *PaulsOptimiser* ; (see Section 25)

```
procedure KantorovichsOptimiser ; (see Section 26 )
var
   Let av ∈ real;
Kantorovich algorithm
begin
   claimHeapSpace;

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
      available↑← startingresource;
default assumption
      x← 0;
      redistributeResources;
      totals↑← 0;
```

Compute the total output of each product in value terms

```
      for k← α to techniques.ω do
         totals↑[produces_k]← totals↑[produces_k] + (techniques_{k,dest,produces_k}) × x_k × L_{produces_k};
      greatestProducedOutput← \max totals↑ ;
      leastProducedOutput← \min totals↑ ;
      if somePositivetargetsStillZero then
      begin
```

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

```
         for s← agriculture to techniques.amusements do
s iterates over outputs
            if targets_S > 0 then
               if totals↑[s] ≤ 0 then L [s ]:=L [s ]*( 1.0+growthfactor *( random and 7)/8 + growthfactor );
         renormaliseMultipliers;
      end
      else ok← true;
      count← count + 1;
      {writeln(X,L/L[0],count);}
      growthfactor← growthfactor × annealrate;
      if count > techniques.omega × 10000 then
      begin
```

```
        writeln( 'after ' , count,  ' trys could not find initial resolving multipliers' );
        halt (999);
    end ;
end ;
{writeln('initial multipliers found after ',count,' trys');
writeln(L);}
count← 0;
```

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

In turn this implies that for any machine that is employed to make two resources the ratio of the resolving multipliers must be the inverse of the ratio of the techniques.

The algorithm will work output a time bringing ever more outputs into equality with their targets. We define the set of resources whose outputs has been brought into equality with target as the equated set.

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

```
   growthfactor← growthconst;
   computeTotalsEtc;
   repeat
      case optimiser of
         kantorovichs : KantorovichsOptimiser;
         pauls : PaulsOptimiser;
      end ;
      computeTotalsEtc;
      count← count + 1;

   until (( ∑ equated↑ ) = numpostargets) ∨ (count > (500)) ∨ (growthfactor < ϵ);
   { writeln('second pass terminated after ', count);}
   rescaleall;
   freeHeapSpace;
end ;
```

Indexed method

9 kantorovichMethod

```
procedure kantorovichMethod ( var techniques :technologytable ; var target ,startingresource ,L
:ODVvec ; var X :intensityvec );
var
   Let index ∈ ^productionindex;
   Let t ∈ technique;
   Let r ∈ resource;
   Let targets ∈ ^odvvec;
begin
   new ( index , techniques . omega );

   growthfactor← growthconst;
   new ( targets ,techniques .amusements );
   targets↑ ← { 0.0              if target = 0.0
              { target/target_0  otherwise        ;
   for t← α to techniques.ω do
      for r← agriculture to techniques.amusements do
         if techniques_{t,dest,r} > 0 then index↑[t]← r;
```

Kantorovich method

prescale by targets

$$techniques \leftarrow \begin{cases} techniques/targets\uparrow & \text{if } target \neq 0 \\ techniques & \text{otherwise} \end{cases};$$

$$startingresource \leftarrow \begin{cases} startingresource / targets \uparrow & \text{if } target \neq 0 \\ startingresource & \text{otherwise} \end{cases};$$

indexedMethod (*index*↑, *techniques*, *targets*↑, *startingresource*, *L*, *X*);

return scale of the input data to its original value

$$techniques \leftarrow \begin{cases} techniques \times targets \uparrow & \text{if } target \neq 0 \\ techniques & \text{otherwise} \end{cases};$$

$$startingresource \leftarrow \begin{cases} startingresource \times targets \uparrow & \text{if } target \neq 0 \\ startingresource & \text{otherwise} \end{cases};$$

dispose (*index*);
dispose (*targets*);
end ; Kantorovich method

10 findfirstinput

function *findfirstinput* :*resource* ;
var
Let $t \in$ technique;
Let $r, f \in$ resource;
begin
$f \leftarrow techniques.amusements$;
for $t \leftarrow \alpha$ **to** $techniques.\omega$ **do**
for $r \leftarrow f$ **downto** 0 **do**
if $techniques_{t,src,r} > 0$ **then** $f \leftarrow r$;
findfirstinput $\leftarrow f$
end ; Find first input

11 claimHeapSpace

procedure *claimHeapSpace* ;
var
Let *maxtechnique*, *maxresource* $\in$ integer;
begin claimHeapSpace;
$maxtechnique \leftarrow techniques.\omega$;
$maxresource \leftarrow techniques.amusements$;
new (*achievableintensities* ,*maxresource*);
new (*standardisedoutput* ,*maxtechnique* ,*maxresource*);
new (*available* ,*startingresource* .*amusements*);
new (*outputs* ,*maxtechnique*);

```
    new ( totals ,maxresource );
    new ( equated ,maxresource );
    A← findfirstinput;
claimHeapSpace
end ;
```

12 freeheapspace

```
procedure freeheapspace ;
begin
    dispose ( achievableintensities );
    dispose ( standardisedoutput );
    dispose ( available );
    dispose ( outputs );
    dispose ( totals );
    dispose ( equated );
end ;
```

13 marginalgain

```
function marginalgain ( s :technique ; d :resource ):volume ;
```

This computes the marginal gain, of a small shift of the resource d time to the specified technique type s. We compute the effect of multiplying all current time allocations to $1-\epsilon$ whilst increasing the allocation of time to resource s by ϵ.

```
var
    Let currentoutput, currentuse, shift, marginalIncreaseinS ∈ real;
    Let valueofincrease, valueofloss ∈ real;
    Let t ∈ technique;
marginal gain
begin
```

check if technique s uses resource d

```
    if techniques_{s,src,d} = 0 then
    begin
        marginalgain← - maxint ;
    end
    else
```

begin

get total current use of resource d

current use is negative !
$currentuse \leftarrow \sum x \times techniques_{t_0,src,d}$;

shift will be negative too
$shift \leftarrow \epsilon \times currentuse$;

marginal increase will be +ve, since we divide by a negative
$marginalIncreaseInS \leftarrow \frac{shift \times techniques_{s,dest,produces_s}}{techniques_{s,src,d}}$;

$valueofincrease \leftarrow marginalIncreaseInS \times L_{produces_s}$;

we next have to work out the total value of output currently produced in all sectors by d

$valueofloss \leftarrow 0$;
for $t \leftarrow \alpha$ **to** $techniques.\omega$ **do**
if it uses d
if $techniques_{t,src,d} \neq 0$ **then**
$valueofloss \leftarrow valueofloss + techniques_{t,dest,produces_t} \times x_t \times \epsilon \times L_{produces_t}$;

$marginalgain \leftarrow valueofincrease - valueofloss$
end
marginalgain
end ;

14 mpp

function *mpp (s :technique ; d :resource):volume ;*

This computes the marginal physical product or the output of s with respect to input d

mpp
begin

check if technique s uses resource d

if $techniques_{s,src,d} = 0$ **then**
begin
$mpp \leftarrow$ *- maxint* ;

```
    end
    else
    begin
        mpp← techniques_{s,dest,produces_s} / techniques_{s,src,d};
    end
end ;
```

mpp

15 mostProductiveTechniqueFor

function *mostProductiveTechniqueFor (e :resource):technique ;*

This determines within which technique resource e produces the most of. under current valuations.

```
var
    Let v ∈ volume;
    Let j, s ∈ technique;
begin
    v← 0.0;
    s← α;
    for j← α to techniques.ω do
        if techniques_{j,src,e} < 0 then
        begin
            if v < (L_{produces_j} × techniques_{j,dest,produces_j} × x_j) / (−techniques_{j,src,e}) then
            begin
                v← (L_{produces_j} × techniques_{j,dest,produces_j} × x_j) / (−techniques_{j,src,e});
                s← j;
            end ;
        end ;

        mostProductiveTechniqueFor← s;
end ;
```

it uses this input

mostProductiveTechniqueFor

16 ComputeTotalsEtc

procedure *ComputeTotalsEtc ;*

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

```
var
   Let i, j ∈ integer;
   Let least ∈ real;
begin
   totals↑← 0;
   for i← α to techniques.ω do
   begin
      totals↑[produces_i]← totals↑[produces_i] + (techniques_{i,dest,produces_i}) × x_i;
   end ;
```

totals now has a vector of how much of each resource is produced at the end relative to the target

```
   weightedTotalOutput← L.totals↑;
   leastProducedOutput← \min (totals↑) ;
```

Find the resource that is least produced or most in shortage

```
   for s← agriculture to techniques.amusements do
      if (totals↑[s]) = leastProducedOutput then least← s;
```

Find the ones on the plan ray

```
   equated↑← 0;
   least← maxint;
   for i← agriculture to techniques.amusements do
      if targets_i > 0 then
         if totals↑[i] < least then least← totals↑[i];

   for i← agriculture to techniques.amusements do
      if targets_1 > 0 then
         if (totals↑[i] - least) < ϵ then equated↑[i]← 1;


   equated↑← ord(((totals↑ - least ) < ϵ) ∧ ((totals↑ - least ) > - ϵ ));
   {writeln('t',totals);
   writeln('e',equated);writeln('x',x);
```

```
  writeln('least',least);
  {write('press return');readln;{}
```

end ;

17 somePositiveTargetsStillZero

function *somePositiveTargetsStillZero* :*boolean* ;

returns true if some products which should have positive output are still zero

begin
  somePositiveTargetsStillZero← \ or ((*totals*↑ ≤ 0) ∧ (*targets* > 0)) ;
end ;

18 numpostargets

function *numpostargets* :*integer* ;
begin
end ;

19 renormaliseMultipliers

procedure *renormaliseMultipliers* ;

In order to prevent uncontrolled growth or shrinkage in multipliers, express them with respect to their own average.

var
  Let *av* ∈ real;
begin
find the average $av \leftarrow \sum \frac{L}{1+techniques.amusements}$;
renormalise L $L \leftarrow \frac{L}{av}$;
end ;

20 LowerNonEquatedMultipliers

procedure *LowerNonEquatedMultipliers* ;
begin

those in the equated set are unchanged by this others are reduced

$L \leftarrow L \times (equated\uparrow \times growthfactor + 1 - growthfactor)$;

renormaliseMultipliers;
end ;

21 LowerNonEquatedIntensities

procedure *LowerNonEquatedIntensities* ;
begin
$X \leftarrow X \times (equated\uparrow[\text{produces}] \times growthfactor + 1 - growthfactor)$;
end ;

22 getavailability

procedure *getavailability* ;
var
Let $i \in$ integer;
begin

$available\uparrow \leftarrow startingresource$;
for $i \leftarrow \alpha$ **to** *techniques.omega* **do**
$available\uparrow \leftarrow available\uparrow + techniques_{i,src} \times X_i$;
end ;

23 RedistributeResources

procedure *RedistributeResources* ;

move the free resources to the place where they are most valuable

begin

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

```
standardisedoutput↑← (techniques_{ι0,dest} × L);
getavailability;
```

For each input find the technique for which it has the best performance

```
for e← agriculture to techniques.amusements do
```
test it is an input
```
  if available↑[e] > 0 then
  begin
```

find the best performance of the input on any technique

```
    outputs↑← 0;
```
try every technique
```
    for s← α to techniques. ω do
```
which uses resource e
```
      if techniques_{s,src,e} < 0 then
      begin
        t← 0;
        for r← agriculture to techniques.amusements do
          t← t + standardisedoutput↑[s, r];
        outputs↑[s]← t;
      end ;
      best← \max outputs↑ ;
```

allocate each resource to the technique it is best at

```
      for s← α to techniques. ω do
        if outputs↑[s] = best then
        begin
```

set this technique to have an intensity sufficient to use 50 percent of the resource e that is available subject to the availability of the other resources technique s needs. The achievable intensities specifies how much more output this technique could produce subject to the constraint of each type of resource's availability. Thus it is indexed by the available resource types.

```
          achievableintensities ^ :=if techniques [s ,src ]=0.0 then maxreal
          else  - available↑ / (techniques_{s,src}) ;

          targetintensity← 0.5 × \min achievableintensities↑ ;
```

We find the lowest of the intensities that can be achieved with remaining resources.

add it into x

$x_S \leftarrow x_S$ + *targetintensity*;

update the available resources , we add the source weighted by intensity, note that these numbers will be negative and will reduce availability

available↑$\leftarrow$ *available*↑ + (*targetintensity* × $techniques_{S,src}$) ;
end ;

end ;

end ;

24 rescaleall

procedure *rescaleall* ;
```
(* this, having found the best ratio of intensities, scales them all to
the
limit of the available resources by finding which is the rate limiting
resource *)
```
var
Let *usage* ∈ ^resourcevec;
Let *lowestsurplus* ∈ real;
begin
getavailability;
new (*usage* ,*techniques .amusements*);
usage↑$\leftarrow$ *startingresource* - *available*↑;
lowestsurplus$\leftarrow$ $\backslash min$ ($\frac{startingresource}{usage}$↑) ;
X$\leftarrow$ *X* × *lowestsurplus*;
dispose (*usage*);
end ;

25 PaulsOptimiser

procedure *PaulsOptimiser* ;
```
(* !  this is an alternative algorithmic step for finding the optimal
resolving multipliers
once an initial set has been found.

Its basic steps are:
```

```
\begin{enumerate}
\item Lower the resolving multipliers of the non equated outputs relative
to the equated ones.
\item Lower the intensities (X) of all processes producing non equated
outputs.
\item Redistribute the free resources to the processes whose outputs are
in the equated set.
\end{enumerate}
*)
```
begin

LowerNonEquatedMultipliers;
LowerNonEquatedIntensities;
RedistributeResources;
rescaleall;
growthfactor← *growthfactor* × *annealrate*;

end ;

26 KantorovichsOptimiser

procedure *KantorovichsOptimiser* ;

This optimising step is called repeatedly once an initial set of resolving multipliers has been found. It is the one originally proposed by Kantorovich himself.

function *findScoop* (**var** *s* :*technique*):*resource* ; (see Section 27)
begin

Find which input is 2nd best at producing output least in the non equated set under current resolving multipliers. Call this input Scoop. The parameter least is an output of the function call.

Scoop← *findScoop* (*least*);
m← *mostProductiveTechniqueFor* (*Scoop*);

Adjust the resolving multiplier ratio between Scoops resource and the least produced resource to ratio of Scoops techniques.

$$L_{produces_m} \leftarrow \frac{L_{produces_{least}} \times mpp(least,scoop)}{mpp(m,scoop)};$$

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

$j \leftarrow produces_{least}$;

$\lambda \leftarrow \frac{L.(equated\uparrow)}{L_{produces_m}}$;

we have used the cross product with the equated set, represented as a vector of 0 or 1 to sum over the equated set.

$deltam \leftarrow \frac{(-1)\times\lambda\times(totals\uparrow[produces_m]-totals\uparrow[j])}{1+\lambda}$;

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 resources in the equated set. We now compute the change in intensity that Scoop spends on its best resource (dx) by scaling deltam by Scoops technique output for resource m.

$dx \leftarrow \frac{deltam}{-mpp(m,scoop)}$;

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

```
best← - maxint ;
jt← α;

for st← α to techniques.ω do
   if mpp (st, scoop) × equated↑[produces_st] > best then
   begin
      best← mpp (st, scoop);
      jt← st;
   end ;

   x_m← x_m + dx;
   x_jt← x_jt - dx;
end ;
```

27 findScoop

function *findScoop* (**var** *s* :*technique*):*resource* ;

search all outputs which are not yet reaching target to find a pair of input and output = produces[s] where shifting input time to producing that output would produce the best result. Return the input as the function result and the technique to use in the parameter s

```
var
   Let gain ∈ volume;
   Let m ∈ resource;
   Let j, t ∈ technique;
   Let d, Scoop ∈ resource;
find Scoop
begin
   Scoop← A;
   m← 0;
   gain← - maxint ;
   s← α;
   for d← agriculture to techniques.amusements do
   begin
      if startingresource_d > 0 then
      begin
```

that is to say we iterate over all goods used as inputs

```
m stands for Main output of d
         m← produces_mostProductiveTechniqueFor(d);

         for j← α to techniques.ω do
            if equated↑[produces_j] > 0 then
```

find a technique j, whose output is doing ok that potentially could have resource reallocated to make d

```
            begin
               if marginalgain (j, d) > gain then
               begin
```

if this is the best marginal gain so far, record the technique j and the input d

```
                  if equated↑[m] < 1 then
```

```
                begin
                  gain← marginalgain (j, d);
                  { writeln('gain',gain,',j',j,',d',d,',m',m);}
                  Scoop← d;
                  s← j;
                end ;
              end ;
            end ;
        end ;
    end ;
    {writeln('best gain',gain,',s=',s,',scoop=',scoop,',best output from
    scoop',m);}
    findScoop← Scoop;
end ;
```

28 csvfilereader

unit *csvfilereader* ;

This parses csv files meeting the official UK standard for such files The following text is imported from that definition at https://www.ofgem.gov.uk/sites/default/files/docs/2013/01/csvfileformats

29 Introduction

29.1 Background

The comma separated values (CSV) format is a widely used text file format often used to exchange data between applications. It contains multiple records (one per line), and each field is delimited by a comma.

29.2 CSV File Format

The primary function of CSV file is to separate each field values by comma separated and transport text - based data to one or more target application. A source application is one which creates or appends to a CSV file and a target application is one which reads a CSV file

29.2.1 CSV File Structure

The CSV file structure use following two notations

FS (Field Separator) i.e. comma separated

FD (Field Delimiter) i.e. Always use a double - quote.

Each line feed in CSV file represents one record and each line is terminated by any valid NL (New line i.e. Carriage Return (CR) ASCII (13) and Line Feed (LF) ASCII (10)) feed. Each record contains one or more fields and the fields are separated by the FS character (i.e. Comma) A field is a string of text characters which will be delimited by the FD character (i.e. double - quote (")) Any field may be quoted (with double quotes).

Fields containing a line - break, double - quote, and/or commas should be quoted. (If they are not, the file will likely be impossible to process correctly).

The FS ch aracter (i.e. comma) may appear in a FD delimited field and in this case it is not treated as the field separator. If a field's value contains one or more commas, double - quotes, CR or LF characters, then it MUST be delimited by a pair of double - quotes (AS CII 0x22).

DO NOT apply double - quote protection where it is not required as applying double quotes on every field or on empty field would takes more file space If a field requires Excel protection, its value MUST be prefixed with a single tilde character .

See example below:

FS =,

FD ="

Data Record:

```
Test1,Test2„"Test3,Test4","Test5 ""Test6"" Test7","Test8,""",",Test9"
```

Indicat es the following four fields

`Test1`	5 characters
`Test2`	5 characters
	0 characters
`Test3,Test4`	11 characters
`Test5 "Test6" Test7`	20 characters
`Test8,"`	8 characters
`,Test9`	6 characters

30 CSV File Rules

- The file type extension MUST be set to .CSV
- The character set used by data contained in the file MUST be an 8 - bit (UTF - 8).
- No binary data should be transported in CSV file.
- A CSV file MUST contain at least one record.
- No limit to the number of data records
- The End of Record m ust be set to CR +LF (i.e. Carriage Return and Line Feed)
- Do not use whitespaces in the file name
- The EOR marker MUST NOT be taken as being part of the CSV record
- EOF (End of File) character indicates a logical EOF (SUB - ASCII 0x1A) and not the physical en d .
- A logical EOF marker cannot be double - quote protected.
- Any record appears after the EOF will be ignored

30.1 File Size

Maximum csv file size should be 30 MB.

30.2 CSV Records

A CSV record consists of two elements, a data record followed by an end - of - record marker (EOR). The EOR is a data record delivery marker and does not form part of the data delivered by the record

31 CSV Record Rules

Pls. note this rule applies to every CSV record including the last record in the file.

31.1 CSV Field Column Rules

- Each recor d within the same CSV file MUST contain the same number of field columns . The header record describes how many fields the application should expect to process.
- Field columns MUST be separated from each other by a single separation character
- A field column MUST NOT have leading or trailing whitespace

31.2 Header Record Rules

A header record allows the Ofgem IT systems to guard against the potential issues such as missing column or additional column that are not in scope

- The header record MUST be the first recor d in the file.
- A CSV file MUST contain one header record only .
- Header labels MUST NOT be blank.
- Use single word only
- Do not use spaces (Use _ if words needs to be separated)

```
interface
  const
    textlen =80;
  type
    pcsv =^ csvcell ;
    celltype =( linestart ,numeric ,alpha );
    textfield =textline ;
    csvcell = record
      right : pcsv;
      case  tag : celltype  of
        linestart : (down : pcsv);
        numeric : (number : real);
        α : (textual : pstring);
      end ;

      headervec ( max :integer ) =array [1..max ]of pcsv ;
      pheadervec = ↑ headervec ;
    procedure printcsv ( var f :text ;p :pcsv ); (see Section ?? )
    function parsecsvfile ( name :textline ):pcsv ; (see Section ?? )
    function rowcount ( p :pcsv ):integer ; (see Section ?? )
    function getdatamatrix ( p :pcsv ):^ matrix ; (see Section 32 )
    function getcell ( p :pcsv ;row ,col :integer ):pcsv ; (see Section ?? )
    function getrowheaders ( p :pcsv ):^ headervec ; (see Section ?? )
    function getcolheaders ( p :pcsv ):^ headervec ; (see Section ?? )
    function colcount ( p :pcsv ):integer ; (see Section ?? )
```

returns nil for file that can not be opened, otherwise returns pointer to tree of csv-cells.

```
    implementation
```

field delimitor

field separator

record separator

```
const
   FD = 34;
   FS = 44;
   RS = 10;
   EOI = $1a;
   CR = 13;
type
   token = (FDsym);
   tokenset = set of token ;
var
   categorisor:  array [byte] of token;

function getdatamatrix ( p :pcsv ):^ matrix ; (see Section 32 )
```

32 getdatamatrix

```
function getdatamatrix ( p :pcsv ):^ matrix ;
```

extract the column headers as a vector of strings

```
var
   m : ↑ matrix ;
procedure recursedown ( j :integer ;q :pcsv ); (see Section 33 )
```

33 recursedown

```
procedure recursedown ( j :integer ;q :pcsv );
procedure recurse ( i :integer ;q :pcsv ); (see Section 34 )
```

34 recurse

```
procedure recurse ( i :integer ;q :pcsv );
begin

   if  q ≠ nil  then
   begin
      if  i ≥ 1  then
      begin
         if  q↑.tag = numeric  then
```

```
            m↑[j, i]← q↑.number
          else m↑[j, i]← 0.0
        end ;
        recurse (i + 1, q↑.right);
    end
    ;
end ;

begin
    if q ≠ nil then
    begin
        recurse (0, q↑.right);
        recursedown (j + 1, q↑.down);
    end
end ;
begin

    if p = nil then getdatamatrix← nil
    else
    begin
        new ( m ,rowcount ( p )-1,colcount ( p )-1);
        recursedown (1, p↑.down);
        getdatamatrix← m;
    end ;
end ;
function getcolheaders ( p :pcsv ):^ headervec ; (see Section 35 )
```

35 getcolheaders

```
function getcolheaders ( p :pcsv ):^ headervec ;
```

extract the column headers

```
var
    M;
    h : ↑ headervec ;
procedure recurse ( i :integer ;q :pcsv ); (see Section 36 )
```

36 recurse

```
procedure recurse ( i :integer ;q :pcsv );
```

```
begin
   if  q ≠ nil  then
   begin

      if  i ≥ 1  then  h↑[i]← q;
      recurse (i + 1, q↑.right);
   end
end ;
begin

   if  p = nil  then  getcolheaders← nil
   else
   begin
      new ( h ,colcount ( p )-1);
      recurse (0, p↑.right);
      getcolheaders← h;
   end ;
end ;
function getrowheaders ( p :pcsv ):^ headervec ; (see Section 37 )
```

37 getrowheaders

```
function getrowheaders ( p :pcsv ):^ headervec ;
```

extract the rows headers

```
var
   M;
   h : ↑ headervec ;
procedure recurse ( i :integer ;q :pcsv ); (see Section 38 )
```

38 recurse

```
procedure recurse ( i :integer ;q :pcsv );
begin
   if  q ≠ nil  then
   begin
      h↑[i]← q↑.right;
      recurse (i + 1, q↑.down);
   end
end ;
```

```
begin
   if  p = nil  then  getrowheaders← nil
   else
   begin
      new ( h ,rowcount ( p )-1);
      recurse (1, p↑.down);
      getrowheaders← h;
   end ;
end ;
function colcount ( p :pcsv ):integer ; (see Section 39 )
```

39 colcount

```
function colcount ( p :pcsv ):integer ;
```

return the number of columns in the spreadsheet

```
begin
   if  p = nil  then  colcount← 0
   else
      case  p↑.tag  of
         linestart : colcount← colcount (p↑.right);
      end
end ;
function getcell ( p :pcsv ;row ,col :integer ):pcsv ; (see Section 40 )
```

40 getcell

```
function getcell ( p :pcsv ;row ,col :integer ):pcsv ;
```

return the cell at position row,col in the spredsheet

```
begin
   if  p = nil  then  getcell← nil
   else   if  row = 1  then
   begin
      else   if  col = 1  then  getcell← p
   end
end ;
```

procedure *removetrailingnull (var p :pcsv);* (see Section 41)

41 removetrailingnull

procedure *removetrailingnull (var p :pcsv);*
function *onlynulls (q :pcsv):boolean ;* (see Section 42)

42 onlynulls

```
function onlynulls ( q :pcsv ):boolean ;
begin
   if  q = nil  then  onlynulls← false  false
   else
      if  q↑.tag = α  then
      begin
      end
      else  onlynulls← false  false
end ;
begin
   if  p ≠ nil  then
      case  p↑.tag  of
         linestart :
         or ( ( p ^ .down =nil )and onlynulls ( p ^ .right )) then p :=nil
         else  removetrailingnull (p↑.down);
      end
end ;
function rowcount ( p :pcsv ):integer ; (see Section 43 )
```

43 rowcount

```
function rowcount ( p :pcsv ):integer ;
begin
   if  p = nil  then  rowcount← 0
   else
      case  p↑.tag  of
         linestart : rowcount← 1 + rowcount (p↑.down);
         numeric← 1
      end
end ;
```

function *isint (r :real):boolean* ; (see Section 44)

44 isint

```
function isint ( r :real ):boolean ;
var
   i : integer;
begin
   i← round(r);
   isint← (i × 1.0) = r
end ;
procedure printcsv ( var f :text ;p :pcsv ); (see Section 45 )
```

45 printcsv

```
procedure printcsv ( var f :text ;p :pcsv );
begin
   if p ≠ nil then
      with p↑ do
      begin
         if tag = linestart then
         begin
            printcsv (f, right);
            if down ≠ nil then
            begin
               writeln(f);
               printcsv (f, down);
            end ;
         end
         else
            if tag = numeric then
            begin
               else write(f, number : 1 : 6);
               if right ≠ nil then
               begin
                  write ( f , ',' );
               end

            end
            else
               if tag = α then
               begin
```

```
            if textual ≠ nil then write(f, '"' , textual↑, '"' ) else write(f, 'nil' );
            if right ≠ nil then
            begin

               write ( f , ',' );
            end

         end
      end
   end ;
   function parsecsvfile ( name :textfield ):pcsv ; (see Section 46 )
```

46 parsecsvfile

```
function parsecsvfile ( name :textfield ):pcsv ;
const
   megabyte = 1024 × 1024;
   maxbuf = 30 × megabyte;
type
   bytebuf = array [1..maxbuf ] of byte ;
var
   f : fileptr;
   bp : ↑ bytebuf ;
   fs;
   tokstart;
   firstfield;
function thetoken :token ; (see Section 47 )
```

47 thetoken

```
function thetoken :token ;
begin
   if currentchar ≤ fs then
      thetoken← categorisor bp↑[currentchar]
   else thetoken← EOFsym
end ;
function peek ( c :token ):boolean ; (see Section 48 )
```

48 peek

function *peek (c :token):boolean* ;

matches current char against the token c returns true if it matches.

```
begin
   peek← c = thetoken
end ;
function isoneof ( s :tokenset ):boolean ; (see Section 49 )
```

49 isoneof

```
function isoneof ( s :tokenset ):boolean ;
begin
   isoneof← thetoken ∈ s
end ;
procedure nextsymbol ; (see Section 50 )
```

50 nextsymbol

```
procedure nextsymbol ;
begin
   if currentchar ≤ fs then currentchar← currentchar + 1
end ;
function have ( c :token ):boolean ; (see Section 51 )
```

51 have

```
function have ( c :token ):boolean ;
begin
   if peek (c) then
   begin
      nextsymbol;
      have← true;

   end
   else
      have← false;
```

```
end ;
function haveoneof ( c :tokenset ):boolean ; (see Section 52 )
```

52 haveoneof

```
function haveoneof ( c :tokenset ):boolean ;
begin
   if isoneof (c) then
   begin
      nextsymbol;
      haveoneof← true;
   end
   else
      haveoneof← false;
end ;

procedure initialise ; (see Section 53 )
```

53 initialise

```
procedure initialise ;
begin
   firstfield← nil;
   lastfield← nil;
   firstrecord← nil;

end ;
procedure resolvealpha ; (see Section 54 )
```

54 resolvealpha

```
procedure resolvealpha ;
var
   i;
begin
   with lastfield↑ do
   begin
      tag← α;
      new ( textual );
      textual↑← '' ;
```

```
      l← tokend min(tokstart + textlen - 1) ;
      { copy field to string}
      for i← tokstart to l - 1 do
      begin
        textual↑← textual↑ + chr(bp↑[i]) ;
      end ;
    end ;
end ;

procedure resolvedigits ; (see Section 55 )
```

55 resolvedigits

```
procedure resolvedigits ;
var
  i;
  s : string;
begin
  with lastfield↑ do
  begin
    tag← numeric;
    new ( textual );
    s← '' ;
    l← tokend min(tokstart + textlen - 1) ;
    { copy field to a string }
    for i← tokstart to l do
    begin
      s← s + chr(bp↑[i]);
    end ;
    val (s, number, l);
  end ;
end ;
procedure resolvetoken ; (see Section 56 )
```

convert to binary

56 resolvetoken

```
procedure resolvetoken ;
begin
  if chr(bp↑[tokstart]) in [ '0' .. '9' ] then resolvedigits
  else resolvealpha
end ;
```

procedure *markbegin* ; (see Section 57)

57 markbegin

mark start of a field

```
procedure markbegin ;
begin
   tokstart← currentchar;
   new ( lastfield ↑ .right );
   lastfield← lastfield↑.right;
   lastfield↑.right← nil;
end ;
procedure markend ; (see Section 58 )
```

58 markend

marks the end of a field

```
procedure markend ;
begin
   tokend← currentchar;
   resolvetoken;
end ;
procedure setalpha ( s :textfield ); (see Section 59 )
```

59 setalpha

```
procedure setalpha ( s :textfield );
begin
   lastfield↑.tag← α;
   new ( lastfield ↑ .textual );
   lastfield↑.textual↑← s;
end ;
procedure emptyfield ; (see Section 60 )
```

60 emptyfield

```
procedure emptyfield ;
begin

   markbegin;
   setalpha ( '' );
```

```
end ;

procedure parsebarefield ; (see Section 61 )
```

61 parsebarefield

```
procedure parsebarefield ;
begin
   if isoneof ([RSsym, EOFsym, FSsym]) then emptyfield
else begin begin
      markbegin;
      while haveoneof ([any, space]) do   ;
skip over the field
      markend;
   end ;
end ;
procedure parsedelimitedfield ; (see Section 62 )
```

62 parsedelimitedfield

```
procedure parsedelimitedfield ;
```

parses a field nested between " chars converting escape chars as it goes

```
var
   s : textfield;
   i : integer;
   continue : boolean;
procedure appendcurrentchar ; (see Section 63 )
```

63 appendcurrentchar

```
procedure appendcurrentchar ;
begin
   s← s + chr(bp↑[currentchar]);
   nextsymbol;
end ;
begin
   markbegin;
```

```
    s← '' ;
    continue← true;
    repeat
      while isoneof ([FSsym..any]) do
      begin
        appendcurrentchar;
      end ;
eat what may be closing quotes
      have (FDsym);
      continue← peek (FDsym) ∧ (length (s) < textlen);
      if continue then appendcurrentchar;
    until (not continue );
    setalpha (s);
end ;
procedure parsefield ; (see Section 64 )
```

64 parsefield

```
procedure parsefield ;
begin
  if have (FDsym) then parsedelimitedfield
  else parsebarefield
end ;
procedure parserecord ; (see Section 65 )
```

65 parserecord

```
procedure parserecord ;
begin
  parsefield;
  while have (FSsym) do parsefield;
end ;
procedure parseheader ; (see Section 66 )
```

66 parseheader

```
procedure parseheader ;
begin
  { claim heap space for start of first line }
  new ( firstrecord );
  lastfield← firstrecord;
```

```
    firstfield← firstrecord;
    with firstrecord↑ do
    begin
      tag← linestart;
      down← nil;
      right← nil;
    end ;
    parserecord;
end ;
procedure parsewholefile ; (see Section 67 )
```

67 parsewholefile

```
procedure parsewholefile ;
begin
    parseheader;

    while have (RSsym) do
    begin
      { claim heap space for the start of the new line }
      new ( firstfield ^ .down );
      firstfield← firstfield↑.down;
      lastfield← firstfield;
      with firstfield↑ do
      begin
        tag← linestart;
        down← nil;
        right← nil;
      end ;
      parserecord;
    end ;
end ;
begin
    initialise;
    parsecsvfile← nil;

    assign (f, name);
    reset (f);
    if ioresult = 0 then
    begin
      fs← filesize (f);
      if fs < maxbuf then
      begin
        new ( bp );
```

the default case of failure

open file for reading

ioresult =0 if opened ok

```
        blockread (f, bp↑[1], fs, rc);
        if  rc = fs  then
        begin
          currentchar← 1;
```

We now have the csv file in memory - parse it

```
          parsewholefile;
          removetrailingnull (firstrecord);
          parsecsvfile← firstrecord;
        end ;
        dispose ( bp );
        close (f);
      end ;
  end ;
end ;
begin
  categorisor← any;
  categorisor_FD← FDsym;
  categorisor_FS← FSsym;
  categorisor_RS← RSsym;
  categorisor_EOI← EOFsym;
  categorisor_ord( ' ' )← space;
  categorisor_CR← space;
  {writeln('fs=',fs,'fd=',fd,'rs=',rs);
  writeln(categorisor);}
end .
```

Contents

1 Kantorovich's method **2**
1.1 Generalising Kantorovich's approach 7
1.2 A second example 8

2 Valuation **11**
2.1 Alterations made to Kantorovich's algorithm 12

3 The programme objectiveValuations **13**

4 Purpose **13**

5 Comand line interface **14**
5.1 iotable 14
5.2 planray 14
5.3 reseourcevec 15
5.4 results 15

6 rf **16**

7 kantor **17**
7.1 Implementation 21

8 IndexedMethod **22**

9 kantorovichMethod **26**

10 findfirstinput **27**

11 claimHeapSpace **27**

12 freeheapspace **28**

13 marginalgain **28**

14 mpp **29**

15 mostProductiveTechniqueFor **30**

16 ComputeTotalsEtc **30**

17 somePositiveTargetsStillZero **32**

18 numpostargets **32**

19 renormaliseMultipliers **32**

20 LowerNonEquatedMultipliers **33**

21 LowerNonEquatedIntensities **33**

22 getavailability **33**

23 RedistributeResources **33**

24 rescaleall **35**

25 PaulsOptimiser **35**

26 KantorovichsOptimiser **36**

27 findScoop **39**

28 csvfilereader **40**

29 Introduction **40**
29.1 Background 40
29.2 CSV File Format 40
29.2.1 CSV File Structure 41

30 CSV File Rules **43**
30.1 File Size 43
30.2 CSV Records 43

31 CSV Record Rules **43**
31.1 CSV Field Column Rules 43
31.2 Header Record Rules 44

32 getdatamatrix **45**

33 recursedown **45**

34 recurse **45**

35 getcolheaders **46**

36 recurse **46**

37 getrowheaders **47**

38 recurse **47**

39 colcount **48**

40 getcell **48**

41 removetrailingnull **49**

42 onlynulls **49**

43 rowcount **49**

44 isint **50**

45 printcsv **50**

46 parsecsvfile **51**

47 thetoken **51**

48 peek **52**

49 isoneof **52**

50 nextsymbol **52**

51 have **52**

52 haveoneof **53**

53 initialise **53**

54 resolvealpha **53**

55 resolvedigits **54**

56 resolvetoken **54**

57 markbegin **55**

58 markend **55**

59 setalpha **55**

60 emptyfield **55**

61 parsebarefield **56**

62 parsedelimitedfield **56**

63 appendcurrentchar **56**

64 parsefield **57**

65 parserecord **57**

66 parseheader **57**

67 parsewholefile **58**

References

[Dan63] G. Dantzig. *Linear Programming and Extensions.* Princeton University Press, Princeton, 1963.

[Kan60] L.V. Kantorovich. Mathematical Methods of Organizing and Planning Production. *Management Science*, 6(4):366–422, 1960.

[Kan65] L.V. Kantorovich. *The Best Use of Economic Resources.* Harvard University Press, 1965.