Kantorovich and beyond

Towards 21st Century Cyber Communist Economics

Paul Cockshott

In the 1920s the conservative economist von Mises claimed that in the absence of monetary calculation there was no feasible way of coordinating an economy.
At the time this argument was countered by market socialist economists like Lange and Dickinson who said that a socialist country, with state owned industry and commerce, could still use money and a consumer goods market to achieve economic coordination.

With the evident economic and technological success of the USSR during the 1950s and early 60s such criticisms as Mises were largely forgotten. It was only during the 1980s, when the USSR was evidently getting into difficulties that both Western and west influenced economists within the Soviet Union revived Mises argument. Nowadays, if you go to the website of the Mises Institute or other free market think tanks you will get the impression that Mises ideas were unchallenged. Insofar as any challenge is seen, it comes from Dengist market socialism, something which could be seen as similar to the Lange school.

What such accounts miss out is the work of Kantorovich, the brilliant Russian mathematician who won a Nobel prize for his invention of linear optimisation. His techniques, developed from the 1930s to the late 50s provided a definitive theoretical rebuttal to Mises, even though Kantorovich was initially unaware of Mises. Kantorovich's linear programming techniques allow planners to do what Mises claimed was impossible - compute an optimal plan for the whole economy without reference to money or prices. Instead Kantorovich requires a description in entirely physical units of what the production capacity of current plant and equipment is. In addition he assumes that the political authorities have specified an output target - again in material units. This could be in terms of quantities of different kinds of food to be supplied, sq meters of housing space, numbers of taxis to be built etc.

Using linear programming and this entirely physical information, Kantorovich provided a way of calculating how to maximally fulfill the plan targets. Thus he provides an in-principle refutation of Mises claims.

Kantorovich's basic mathematical techniques were subsequently independently invented by Danzig in the west and, vial this route, have been incorporated into open source software packages. One such package, lp-solve, is available on both Windows and Linux.

In order to popularise the work of Kantorovich I have released two software packages for macroeconomic planning. The first uses Kantorovich style linear programming, and the second uses an improved method that I have developed myself since the late 1980s.
The advantage of the new approach is that it is very much faster than Kantrovich's own one and as a result allows computers to be used to plan economies down to a much finer level of detail.

A full account of the new method is given in an article that I have submitted to the World Review of Political Economy. It is an extension of a technique, called Harmony Planning, that Allin and I originally described in our book Towards a New Socialism. What is new is that

1. I have released source code for anyone to use
2. The algorithm has been extended to work with multi-year plans, like the famous 5 year plans of the USSR.
3. Timings have been run on modern computers showing how much faster it is than Kantorovich's method. (Table 1 and 2)

Table 1: Projected time to compute large plans with lp-solve.

industries	1 year plan	5 year plan
500	0.24hr	17hr
5000	11 days	2.2 years
50000	33years	2417years

Table 2: Projected time to compute 5 year plans using Harmony algorithm.

products	1 thread	1000 threads
50000	9.4hr	
50000000	94hr	5.6 mins
200000000	377hrs	22 mins

The source can be downloaded from git hub using the site
https://github.com/wc22m/5yearplan

Let me first describe the part of the package that uses the Kantorovich approach.

Documentation for the Kantorovich style N-year plan solver

This package allows you to experiment with macro-economic planning. Using it you can compute multi-year plans for either toy economies, or in principle, it can be applied to compute sectoral plans for whole economies using input output tables.

It is intended to be run in a Linux environment, from the command line.

When you unpack the plancode.zip file you will find that the directory plancode contains the following files

```
cap.csv  documentation.odt  labtarg.csv
dep.csv  flows.csv          Makefile  planning
```

There are 3 program source files in the directory plancode/planning

```
csvfilereader.java
nyearplan.java
pcsv.java
```

The first is a utility class for reading in the popular Comma Separated Value spreadsheet file format. The second nyearplan.java is a program that analyses a collection of comma separated value files to produce a linear programming file which consists of a set of equations and inequalities that have to be met to satisfy the plan. The last, pcsv.java declares internal classes to be used for the in-memory representation of Comma Separated Value files.

The Makefile contains instructions to compile the software and make your nyear plan.

The details of the economy that you want to model are included in the files

```
cap.csv  dep.csv  flows.csv  labtarg.csv
```

As distributed these contain input output tables for a toy economy set out in a manner that it should be reasonably easy to convert published input output tables from actual economies into this format.

Steps

1. Before you test your system out check that you have the following packages installed in your Linux system:
   a) A Java development system or JDK
   b) the make tool
   c) the lp-solve package
2. Change directory to the planning directory that you got when you unpacked the zip file.
3. Type make
4. The source code will be compiled and a plan calculated. The plan will be placed in the file plan.txt

Interpreting the spreadsheets

Below I show what the spreadsheets look like if opened with a text editor, lower down I will show what they look like with a spreadsheet like LibreOffice Calc.

First comes a flow input/output table in standard column format called flows.csv

```
odroid@odroid:/media/odroid/UBUNTU 18_0/plancode$ cat flows.csv
headings,iron,coal,corn,bread
iron,0.1,0,0,0
coal,2,1,0.04,0.2
corn,0,0,1,1
bread,0,0,0,0
labour,0.3,1,2,0.1
output,10,5,10,1
```

Note that the csv file contains first a row of column headings separated by commas. Later lines all start with a row header which is followed by numbers separated by commas. The headings must start with a letter and contain no spaces. We are dealing with a simple economy that produces iron,coal,corn and bread.

It is easier to understand it if we load the file into LibreOffice. It then looks like this:

flows.csv

headings	iron	coal	corn	bread
iron	0.1	0	0	0
coal	2	1	0.04	0.2
corn	0	0	1	1
bread	0	0	0	0
labour	0.3	1	2	0.1
output	10	5	10	1

Allowing you to see the column layout better. This is a standard format, albeit tiny, input output table.

Read down the **iron** column. Right down at the bottom you see an output row. This says that at the start the economy produces 10 units of iron, and to make that iron it uses 0.1 unit of existing iron, and 2 units of coal plus 0.3 units of labour. Dont worry what the units are for the moment. In principle they can be anything, tons, kgs cubic feet etc, provided that each type of product is consistently measured in its own unit. So you could measure iron in million kg, coal in million lbs, bread in thousands of loaves etc, so long as each mention of iron in the tables is always in million kg, each mention of coal is in million lbs etc.

Similarly to produce 5 units of coal, the economy uses up 1 unit of labour and 1 unit of coal. For each industry you have a column describing how it is made, and each row represents intermediate uses of a given product.

All of these are flows of intermediate products. That is to say flows that are immediately used up as raw materials. The flows do not include flows to replace capital stock used up. For that we need two more tables. The next is the capital stock table, cap.csv, which I will show only in LibreOffice view:

cap.csv

headings	iron	coal	corn	bread
iron	10	7	1.4	1
coal	2	1	0.04	0.1
corn	0	0	1	0
bread	0	0	0	0

This says that to produce the current output of iron, we need a capital stock of 10 units of iron (think big machines), and a buffer stock of 2 units of coal – think piles of it in the ironworks. These are not flows, these are the stocks needed to continue production.

Note that there is no output or labour row for this table, since the labour and the output are assumed to be the same as in the flows.csv table.

However capital stocks wear out. And some stocks wear out more quickly than others. A van, for example, wears out faster than a railway locomotive. So a long term plan needs to know how fast each type of product wears out. This is specified in the depreciation table dep.csv.

dep.csv

headings	iron	coal	corn	bread
iron	0.07	0.07	0.07	0.06
coal	0.5	0.5	0.5	0.5
corn	0	0	0.5	0
bread	0	0	0	0

This table tells you the rate at which the stock of each type of means of production depreciates each year. So 0.07 or 7% of the iron used to make iron wears out each year, whereas the buffer stock of seed corn to make corn, deteriorates much faster, half of it is lost per year.

Finally we have a table labtarg.csv, that specifies the target outputs and available labour force each year of the plan.

headings	iron	coal	corn	bread	labour
year1	0.1	3	0	2	3
year2	0.1	3	0	2.01	3.01
year3	0.1	3	0	2.02	3.02
year4	0.1	3.1	0	2.1	3.03
year5	0.1	3.1	0	2.1	3.07

These specify the target levels of final consumption of each product each year. It is important to note that this is not the total output, since that would also include outputs of raw materials and replacement capital goods. Instead we specify the plan in terms of the output available for consumption. So in the first year we want 0.1 unit of iron for direct consumption – knives and forks, pots and pans, 3 units of coal for heating and cooking, and 2 units of bread. The labour force available to the economy will be 3.

The following year the labour force has grown by 0.01 to 3.01 and, with this larger labour force we aim to produce slightly more bread. And so on for each year. You can add or remove years from the plan by editing the table.

The planning algorithm seeks to maximize the fulfillment of these plan targets for each year of the n year plan.

The final plan

After executing the make command, the plan is in a file plan.txt. The documentation includes the version that will be produced from the example spreadsheets.

finalConsumptionOfbread1	0.318443
finalConsumptionOfbread2	3.51837
finalConsumptionOfbread3	3.54022
finalConsumptionOfbread4	3.60183
finalConsumptionOfbread5	4.11402

Note that in the first year, it is not possible to meet the plan target for bread consumption, but in subsequent years it is overfull-filled by a large margin.

If you look at the section of the plan (at the end) which summarises the overall fulfillment we see that it starts off very poor and then improves dramatically.

targetFulfillmentForYear1	0.159221
targetFulfillmentForYear2	1.75043
targetFulfillmentForYear3	1.75259
targetFulfillmentForYear4	1.71516
targetFulfillmentForYear5	1.95906

This is because the starting position specified in our initial input output tables is producing completely the wrong mix of net outputs to meet the plan targets. In order to get on track, there has to be a substantial re-arrangement of stocks of capital goods to enable the desired new mix of outputs to be made. During this drastic rearrangement, feasible production falls short of the target.

It is better to set the output mix in the first year to be closer to the net output implied by the flow and capital stock tables.

When working with real economy data this should not be a problem since you can look up what the real final consumption in year 1 was and make this, or some large percentage of it, the target. Changes in output mix should be gradual to avoid the plan falling drastically short at the point of change.

Use of the Harmony Planner

The harmony planner, based on the algorithmic techniques described in Towards a New Socialism has been written to have exactly the same input interface as the Kantorovich style planner.

To use it one issues on the command line the command

$ java -Xmx1512m planning.nyearHarmony testflow.csv testcap.csv testdep.csv testtarg.csv

The csv files are the same as before. It uses the same type of spreadsheets, specifying the same parameters of the national economy.

It is also written in Java and shares substantial bodies of code with the Kantorovich interface planner.

The input data format, using IO matrices, is very inefficient for large numbers of industries. The space taken up by the planning tables and by the printed results grows as order N^2 . At present this limits the number of industries worked with to under 1000 for practical purposes. Above that level one runs out of heap space in the java module that parses spreadsheets. From the standpoint of the underlying planning algorithm that is not a problem. Given data provided in a more compact form, for example from relational databases, space would not be a problem until much larger numbers of products or industries are encountered, since the storage used for the actual calculation grows only in proportion to the non-null entries in the production matrix.

The description of the harmony algorithm in Towards a New Socialism

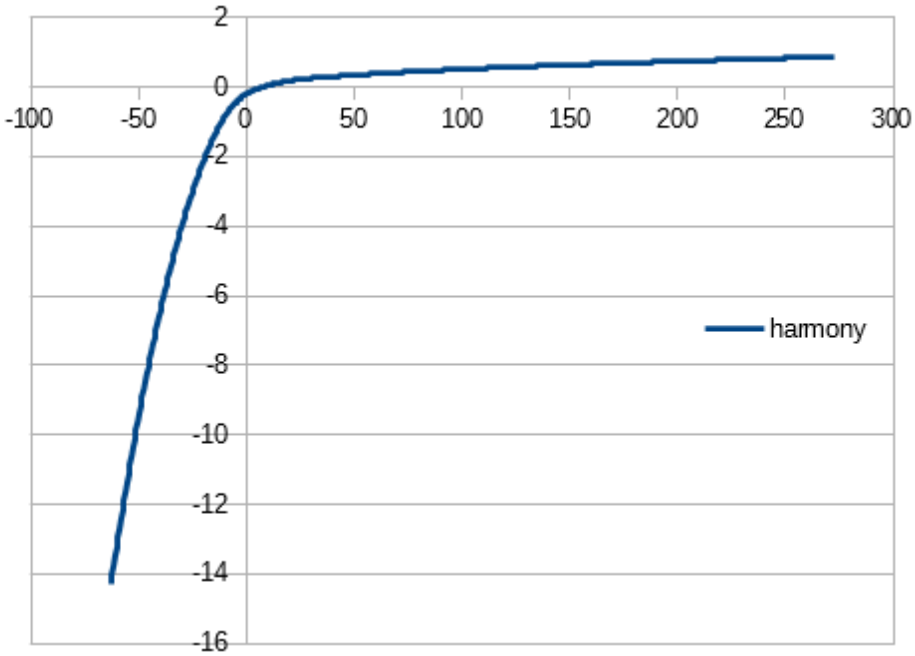
Because the full description of the new implementation of the algorithm is under review by a journal, I give an account here of the first version of the algorithm published in the 1990s. The new version contains some subtle extensions but the main thrust of it is gleanable from the earlier version.

Stages of the algorithm
(1) Randomly allocate resources to the industries. This is a notional random allocation, taking place within the computer. No real allocation of goods in the economy takes place at this stage. We specify a random allocation, since if our relaxation technique is valid then any starting point is as good as any other.
(2) For each industry determine which of the resources currently available to it is the rate-limiting factor, that is to say the resource that acts as a bottleneck on production.

(3) Each industry gives up non-critical resources (i.e. those which are surplus to requirements, given the rate-limiting factor) and allocates them to a common pool. This by definition does not reduce production and thus leaves harmony unaltered. Note that this reallocation takes place only within the computer's memory; there would be no reallocation in the real world until the whole algorithm terminated.
(4) Work out harmony of each industry.
(5) Work out the mean harmony for the whole economy.
(6) Sort industries in order of harmony.
(7) Starting with those industries with highest harmony, reduce their output until their production level is such that their harmony is equal to the mean harmony. This is easy to do since the harmony function is invertible (that is, we can work backwards from harmony level to the corresponding output just as easily as from output to the corresponding harmony). Resources thus released go into the common pool.
(8) Starting with the industries with lowest harmony, assign to them resources from the 'pool' and increase their output until they are producing at a level equal to the mean harmony.
(9) Work out the new mean harmony. If it is significantly different from before go back to stage 6.
Why does this work for multiple techniques in one industry?
Well the harmony function can be given as

```
function H(target,output: real) :real;
var excess:real;
begin
   excess:= (output -target)/target;
   if excess > 0 then H:= sqrt(excess)
   else H:= - (excess*excess)
end;
```

The excess of a good is therefore expressed in terms of fractions of the target output for that good.
The aim is to have a function that is symmetrically reflected around an axis of -45% passing through the plan target. It looks like this



Essentially it heavily penalizes falling short of the plan target and modestly rewards rising above the target. Allin pointed out to me later that it performs poorly in the range -1 to 1 because of a discontinuity of the derivative at 0, but for practical purposes, one can eliminate this problem by changing the definition of excess to
excess:= 10000000* (output -target)/target;
so that the excess will almost never fall in the range with nasty derivatives.
If one wants to work with multiple techniques per product, you slightly alter the algorithm so that where we refer to industries, we substitute techniques. It which now reads
(1) Randomly allocate resources to techniques. This is a notional random allocation, taking place within the computer. No real allocation of goods in the economy takes place at this stage. We specify a random allocation, since if our relaxation technique is valid then any starting point is as good as any other.
(2) For each technique determine which of the resources currently available to it is the rate-limiting factor, that is to say the resource that acts as a bottleneck on production.
(3) Each technique gives up non-critical resources (i.e. those which are surplus to requirements, given the rate-limiting factor) and allocates them to a common pool. This by definition does not reduce production and thus leaves harmony unaltered. Note that this reallocation takes place only within the computer's memory; there would be no reallocation in the real world until the whole algorithm terminated.
(4) Work out harmony of each product and each technique.
(5) Work out the mean harmony for the whole economy.
(6) Sort products in order of harmony.
(7) Starting with fraction of those products with highest harmony (say the top 20%), reduce their output until their production level is such that their harmony is equal to the mean harmony. This is easy to do since the harmony function is invertible (that is, we can work backwards from harmony level to the corresponding output just as easily as from output to the corresponding harmony). Resources thus released go into the common pool.
(8) Starting with the products with lowest harmony, assign to them resources
from the 'pool' and increase their output. If possible raise it until they are producing at a level equal to the mean harmony. In allocating resources, preferentially allocate them to the techniques within an industry that are capable of producing the greatest increase in harmony given the available pool of resources.
(9) Work out the new mean harmony. If it is significantly different from before
go back to stage 6.

The harmony function is defined on the net product of the economy, so when computing the harmony of a technique one forms a weighted sum of the harmony contribution of the output less the harmony used up by the inputs. In the case where there is just one technique per product, the harmony contribution of the output is simply the harmony of the product, in other cases, it has to be scaled by the amount of the product produced. One always scales the inputs used by their harmonies and the amount used when computing the final contribution of the technique.