Factorio Model, Version 3

Jaiden King 2/7/2025

This script shows how to use the Factorio Model. The model will balance the whole system from mining drills to assembling machines for a desired production rate.

The model inputs are:

- desired production rates.

The model outputs are:

- Net computed production rates (should match input)
- Gross production rates (what you'll see on in-game production stats)
- Tips for minimum transport belt requirements
- Construction suggestion (# of assembling machines, etc.)

The FactorioMath_03b.mlx document shows how to limit the scope of the problem for balancing smaller subsystems.

```
% Standard workspace clear
clear all
close all
clc
```

These external files populate the global variables used to run the model. They define all the resources, machines, and processes that exist in the game.

```
% Initialize model workspace variables
resources
machines
processes % Defines the variable A used below
```

The first thing the user does is define the production rates that they want to solve for. This is just a column vector of numbers, but the `new_rate_vector` function helps make the syntax more readable by letting you use item names. It's not uncommon to have multiple versions of the function call commented out as reference or for quick access.

```
% Set the desired output rate.
b = new_rate_vector({"electronic_circuit", 15}); % One transport belt worth of
green circuits
%b = new_rate_vector({"automation_science_pack", 15; "copper_plate", 30}); % One
transport belt worth of science and 2 extra belts of copper plate
```

After doing all the work to set up the problem in the form of $A\mathbf{x} = \mathbf{b}$, solving it is a single operation.

```
% Solve the reduced system
x = A\b;
```

```
x = x.*(abs(x)>0.0001); % Hack to suppress very small numbers in the result caused
by numerical error
```

We now compute the final production rate vector $A\mathbf{x}$. This will include the resources that were previously ignored during the solve. It's always good to do this to double check the output rather than assuming it resulted in the rates you asked for. We also compute the gross rates by supressing all the negative entries of the A matrix. These values would match what you see in the in-game produciton stats.

```
Ax=A*x;
Ax = Ax.*(abs(Ax)>0.0001); % Hack to suppress very small numbers in the result
caused by numerical error
Ax_gross = gross_rates(A,x);
```

So far $\mathbf{x}$ has been used to represent how many copies of each process were running, assuming it was all done with crafting speed 1. However, many processes can't run on a machine with crafting speed 1. The `assign_default_machines_to_unit_processes` function converts the $\mathbf{x}$ vector into a matrix which represents processes implemented on specific machines. This can then be used as a guide for what to build in-game.

```
x2 = assign_default_machines_to_unit_processes(x);
```

The final report summarizes our work.

```
display_results;
```

```
 --- Model Input ---

Target Production Rates:
electronic_circuit: 15

Resource Scope:
(All resources)

Process Scope:
(All processes)

 --- Model Results ---

Net Production Rates:
electronic_circuit: 15

Gross Production Rates:
iron_ore: 15
copper_ore: 22.5
iron_plate: 15
copper_plate: 22.5
copper_cable: 45
electronic_circuit: 15

Belt Requirements (Yellow Belt):
iron_ore: 1
copper_ore: 1.5
iron_plate: 1
copper_plate: 1.5
copper_cable: 3
electronic_circuit: 1
Note: Belt requirements depend on specific implementation details.
```

```
Unit Process Analysis:
Non-zero UNIT processes:
iron_mining    15
copper_mining    22.5
iron_plate_smelting    48
copper_plate_smelting    72
copper_cable_crafting    11.25
circuit_crafting    7.5

Construction Suggestion:
iron_mining    electric_mining_drill    30
copper_mining    electric_mining_drill    45
iron_plate_smelting    electric_furnace    24
copper_plate_smelting    electric_furnace    36
copper_cable_crafting    assembling_machine_3    9
circuit_crafting    assembling_machine_3    6
```

Misc Todo:

Function to make the machine matrix (`x2`) from string-string-int like the `new_rate_vector` function.

Make a "perfect balance" finder rather than solving for a specific production rate.