1 Introduction

The aim of this Notebook is to present a *Godley tables* translator from different formats (either .txt, .ods or .xls) to a .mo file executable in *OpenModelica*. To do so, a simple monetary model will be built in *Minsky Software*, and then the different possible files that the translator can return will be examinated.

1.1 Godley tables

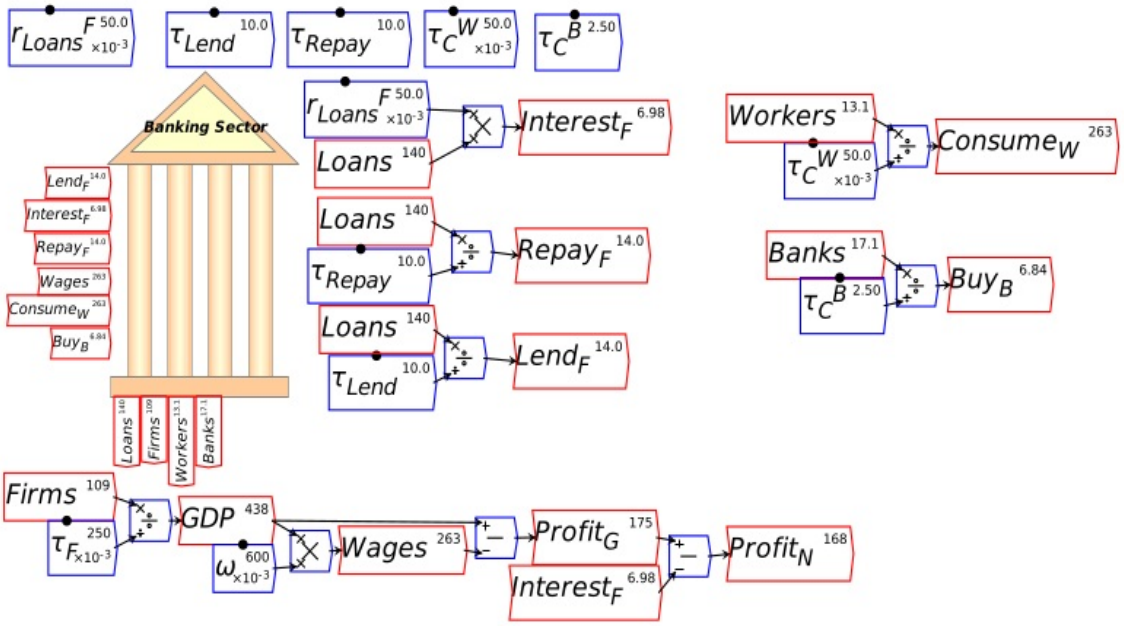
Godley Tables are double-entry bookkepping tables, that allow to preserve stock-flows consistency. Each column represents a stock (it can be an Asset, a Liability or Equity); each row represents a flow. So each position *ij* from the table represents **a flow modifying a stock**. The main advantages of this way of expressing transactions are the preservation of the stock-clow consistency and of the fundamental law *Assets - Liabilities - Equity = 0*.

1.2 The model

The simple monetary model is based on the following *Godley table:*

	Asset	Liability		Equity	A-L-E
Flows ↓ / Stock Vars →	*Loans* ▼	*Firms* ▼	*Workers* ▼	*Banks*	*0*
Initial Conditions	110	90	10	10	*0*
Firm borrowing	$Lend_F$	$Lend_F$			*0*
Interest payments		$-Interest_F$		$Interest_F$	*0*
Debt repayment	$-Repay_F$	$-Repay_F$			*0*
Hire workers		*-Wages*	*Wages*		*0*
Workers consume		$Consume_W$	$-Consume_W$		*0*
Bank purchases		Buy_B		$-Buy_B$	*0*

And is complemented by the following variables



Notice how the Godley Table takes different flows and allows them to be used to define the different stocks. Many of the flows that modify the stocks depend themselves on them;

therefore, a cycle of interdependence between the variables is formed.

2 Building of the model in OpenModelica

To begin, it is necessary to replicate the Godley table in a .txt, .ods or .xlsx file (these last two are exactly the same) with the following format:

```
Stock type,Asset,Liability,Liability,Equity
Stock name, Loans,Firms,Workers,Banks
Initial Conditions,110,90,10,10
Firm borrowing,Lend_F,Lend_F,0,0
Interest payments,0,-Interest_F,0,Interest_F
Debt repayment,-Repay_F,-Repay_F,0,0
Hire workers,0,-Wages,Wages,0
Workers consume,0,Consume_W,-Consume_W,0
Bank purchases,0,Buy_B,0,-Buy_B
```

Stock type	Asset	Liability	Liability	Equity
Stock name	Loans	Firms	Workers	Equity
Initial Conditions	110	90	10	10
Firm borrowing	Lend_F	Lend_F	0	0
Interest payments	0	-Interest_F	0	Interest_F
Debt repayment	-Repay_F	-Repay_F	0	0
Hire workers	0	-Wages	Wages	0
Workers consume	0	Consume_W	-Consume_W	0
Bank purchases	0	Buy_B	0	-Buy_B

Once the source code is already downloaded, the written table must be saved in the *files* directory, in folder corresponding to the chosen format. Once this is done, a terminal must be oppened at the *csv-to-modelica-translator* directory and the following command should be executed:

python3 main.py

After this, the chosen format must be stated; then, the name of the file (without the extension) must be written; finally, two options are presented: a model with or without **Connectors.** We will see now what these two really mean.

2.1 Model without Connectors

If this option is chosen, the script will create an .mo file that will be saved at "*files/modelica"*, named *chosen_name.mo*. In the chosen example, the .mo file looks like this:

```
model Godley_table
output Real  Loans(start = 110, fixed = true);
output Real Firms(start = 90, fixed = true);
output Real Workers(start = 10, fixed = true);
output Real Banks(start = 10, fixed = true);
input Real Buy_B;
input Real Lend_F;
input Real Interest_F;
input Real Consume_W;
input Real Repay_F;
input Real Wages;
equation
der(Loans) = (Lend_F) + (-Repay_F);
der(Firms) = (Lend_F) + (-Interest_F) + (-Repay_F) + (-Wages) +
(Consume_W) + (Buy_B);
der(Workers) = (Wages) + (-Consume_W);
der(Banks) = (Interest_F) + (-Buy_B);
end Godley_table;
```

```
{Godley_table}
```

The structure of the model is the following: firstly, there is a series of **output** variables, representing stocks; each one of them has a starting value, which in the "hand-written" table was defined in the *Initial Conditions* row. Secondly, **input** variables are added, the flows; none of them is defined, because this is not specified in the table. Lastly, in the **equation** section, corresponding to the code in which behaiviour of each variable is defined, the different derivatives of each of the stocks are presented.
As it was alreay explained, the flows behaiviour is yet to be defined. To do so, a new model in OpenModelica is created.

```
model Monetary_model
equation

end Monetary_model;
```

```
{Monetary_model}
```

With the Godley table defined, now it is time to define the complete model by creating a new class named *Monetary_model*. The first thing to do is **instantiating** the previosuly defined class *Godley_table,* which we are naming *godley_table_1*.

```
model Monetary_model
Godley_table godley_table_1;
equation

end Monetary_model;
```

```
{Monetary_model}
```

godley_table variable_1 is then a instance of the class *Godley_table.* Therefore, many different godley tables can be created in a single bigger model. Their intern variables are also accesible to be modified. Particularily, the current goal is to define the behaiviour of the following flows: *Buy_B, Lend_F, Interest_F, Consume_W, Repay_F y Wages.* However, before this, some necessary parameters are to be defined: *tau_Repay, tau_Lend, tau_CB, tau_CW y r_Loans* (in *Minsky SW*, these are the blue coloured blocks). We define parameters like this:

```
model Monetary_model
Godley_table godley_table_1;
parameter Real tau_Repay = 10;
parameter Real tau_Lend = 10;
parameter Real tau_CW = 0.05;
parameter Real tau_CB = 2.5;
parameter Real r_Loans = 0.05;

equation

end Monetary_model;
```

```
{Monetary_model}
```

Now, it is necessary to determine the behaviour of the flows; to access an intern variable of another variable in Modelica, we write *variable.intern_variable.* All of this is defined in the **equation** section.

```
model Monetary_model
Godley_table godley_table_1;
parameter Real tau_Repay = 10;
```

```
parameter Real tau_Lend = 10;
parameter Real tau_CW = 0.05;
parameter Real tau_CB = 2.5;
parameter Real r_Loans = 0.05;

equation
godley_table_1.Interest_F = r_Loans*godley_table_1.Loans;
godley_table_1.Repay_F = godley_table_1.Loans / tau_Repay;
godley_table_1.Lend_F = godley_table_1.Loans / tau_Lend;
godley_table_1.Consume_W = godley_table_1.Workers / tau_CW;
godley_table_1.Buy_B = godley_table_1.Banks / tau_CB;

end Monetary_model;
```

```
{Monetary_model}
```

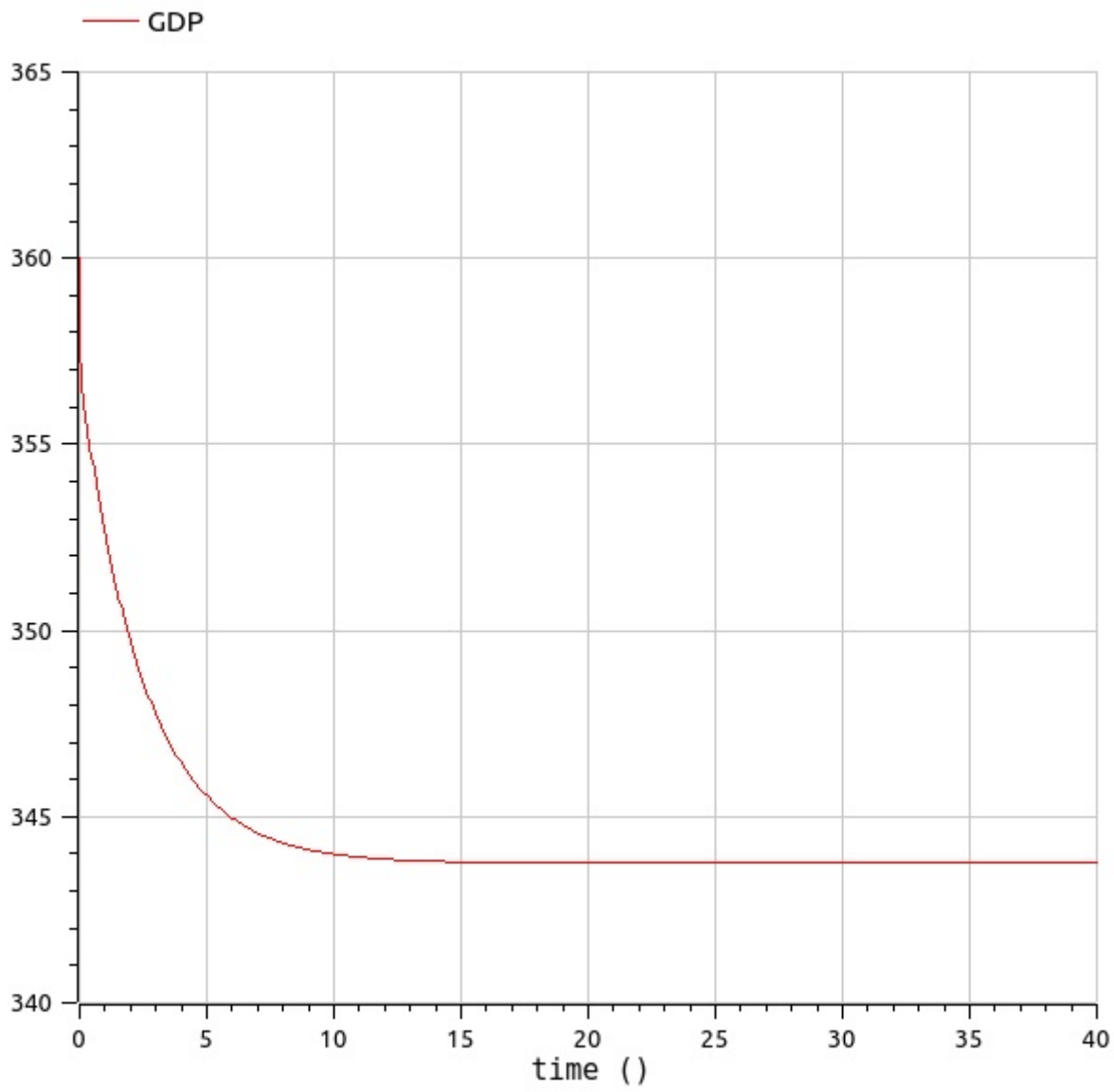
In the original model, *Wages* depends on another variable *GDP*, which is not in the Godley table. Besides, this variable is affected by some new paratemers, *omega* and *tau_F*. It is therefore time, to define all of them:

```
model Monetary_model
Godley_table godley_table_1;
parameter Real tau_Repay = 10;
parameter Real tau_Lend = 10;
parameter Real tau_CW = 0.05;
parameter Real tau_CB = 2.5;
parameter Real r_Loans = 0.05;
parameter Real omega = 0.6;
parameter Real tau_F = 0.25;
output Real GDP;

equation
godley_table_1.Interest_F = r_Loans*godley_table_1.Loans;
godley_table_1.Repay_F = godley_table_1.Loans / tau_Repay;
godley_table_1.Lend_F = godley_table_1.Loans / tau_Lend;
godley_table_1.Consume_W = godley_table_1.Workers / tau_CW;
godley_table_1.Buy_B = godley_table_1.Banks / tau_CB;
godley_table_1.Wages = GDP * omega;
GDP = godley_table_1.Firms / tau_F;

end Monetary_model;
```

```
{Monetary_model}
```

```
simulate(Monetary_model, stopTime=40);
plot(GDP);
```

```
[done]
```



OpenModelica allows to modify parameters discretely. To do so, the *sample* function allows to take certain actions given a certain interval of time and a starting time point. For example, to increase in 10 units the parameter *tau_Repay* every 10 units of time, the following code can be written:

```
model Monetary_model
Godley_table godley_table_1;
Real tau_Repay (start = 10, fixed = true);
parameter Real tau_Lend = 10;
parameter Real tau_CW = 0.05;
parameter Real tau_CB = 2.5;
parameter Real r_Loans = 0.05;
parameter Real omega = 0.6;
parameter Real tau_F = 0.25;
output Real GDP;

equation

when sample(0, 10) then
  tau_Repay = pre(tau_Repay) + 10;
end when;
```

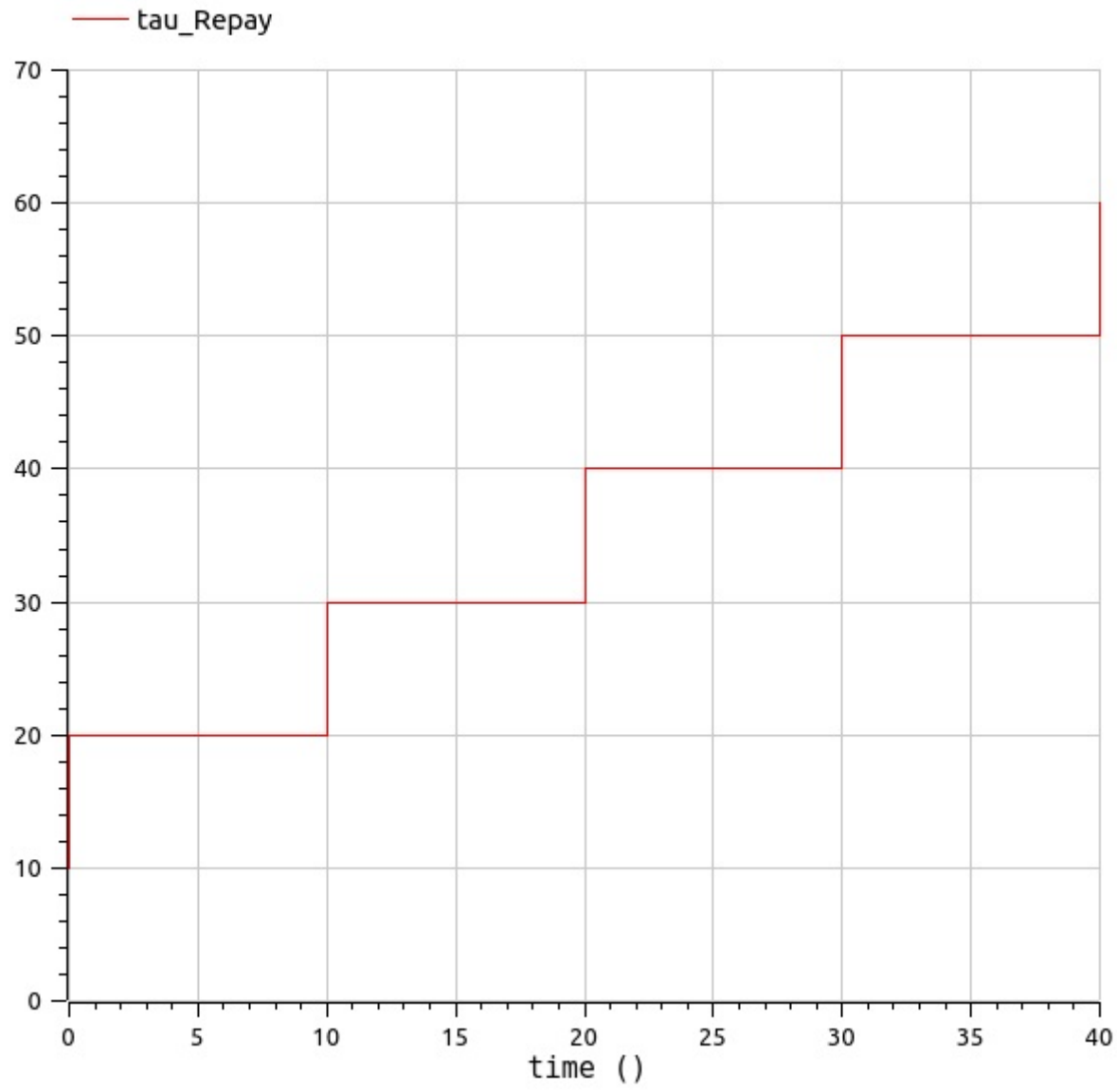
```
godley_table_1.Interest_F = r_Loans*godley_table_1.Loans;
godley_table_1.Repay_F = godley_table_1.Loans / tau_Repay;
godley_table_1.Lend_F = godley_table_1.Loans / tau_Lend;
godley_table_1.Consume_W = godley_table_1.Workers / tau_CW;
godley_table_1.Buy_B = godley_table_1.Banks / tau_CB;
godley_table_1.Wages = GDP * omega;
GDP = godley_table_1.Firms / tau_F;

end Monetary_model;
```

```
{Monetary_model}
```

It is important to know that these changes cannot be made if the variable is defined as **parameter.**

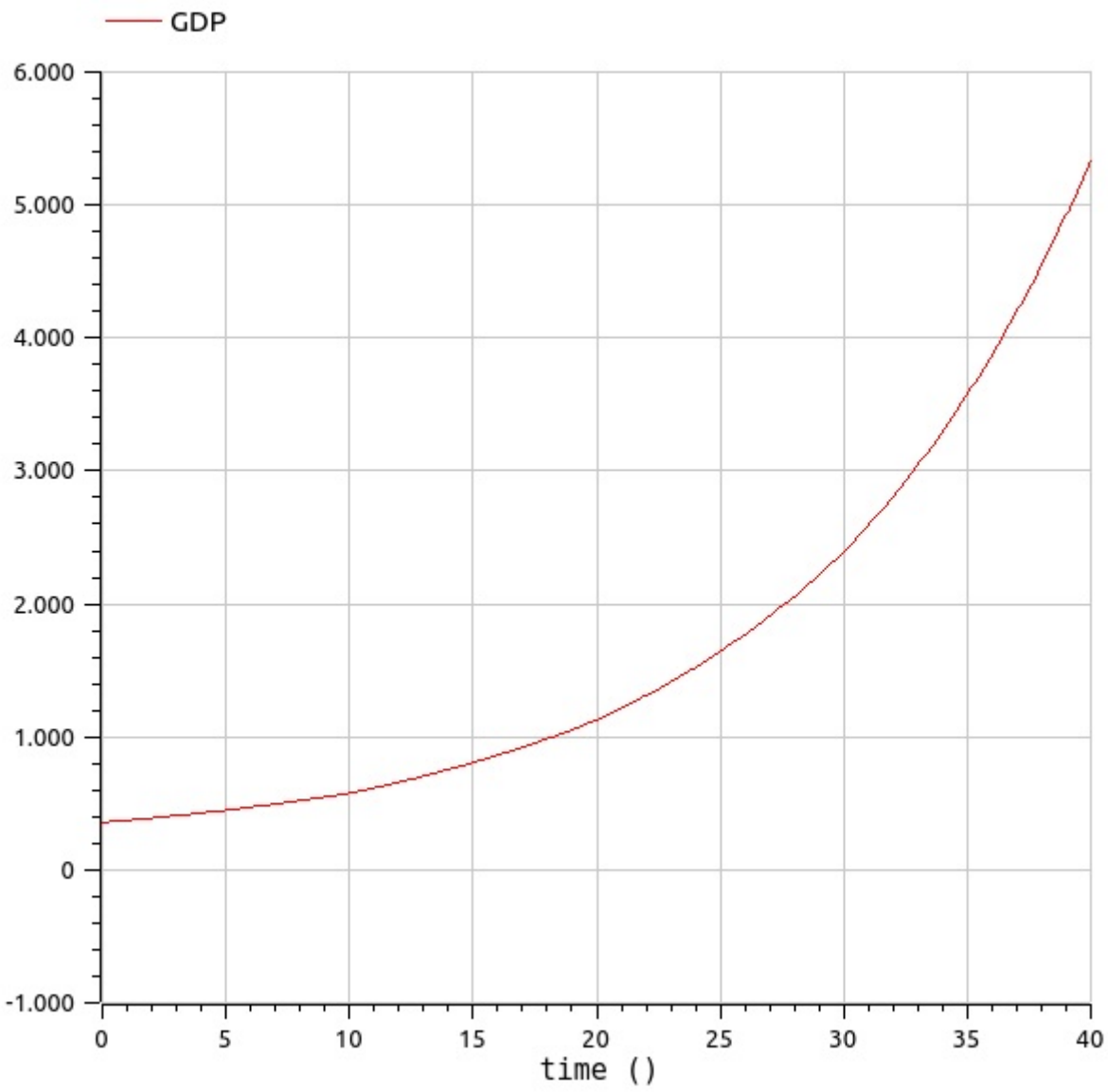
```
simulate(Monetary_model,stopTime = 40);
plot(tau_Repay);
```

```
[done]
```




```
plot(GDP);
```

```
[done]
```



Notice how, because of the increase in the *tau_Repay* parameter, the *GDP* start to grow, contrasting with the previously obtained results.

Some other interesting variations can be defined for the parameters:

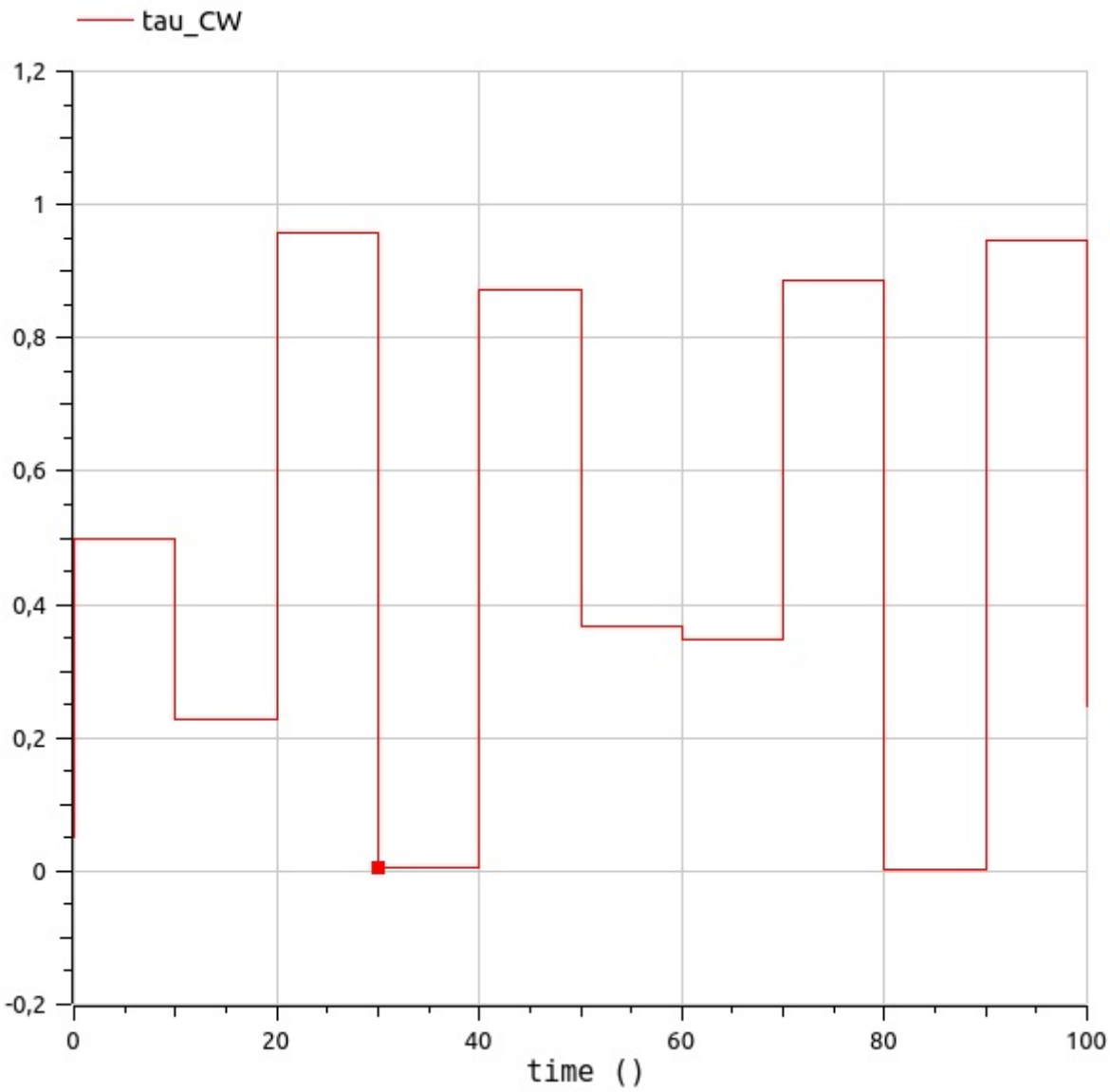
```
model Monetary_model
Godley_table godley_table_1;
parameter Real tau_Repay = 10;
parameter Real tau_Lend = 10;
Real tau_CW (start = 0.05, fixed = true);
parameter Real tau_CB = 2.5;
parameter Real r_Loans = 0.05;
parameter Real omega = 0.6;
parameter Real tau_F = 0.25;
output Real GDP;

equation

when sample(0, 10) then
  tau_CW = (sin(time) + 1) / 2;
end when;
```

```
godley_table_1.Interest_F = r_Loans*godley_table_1.Loans;
godley_table_1.Repay_F = godley_table_1.Loans / tau_Repay;
godley_table_1.Lend_F = godley_table_1.Loans / tau_Lend;
godley_table_1.Consume_W = godley_table_1.Workers / tau_CW;
godley_table_1.Buy_B = godley_table_1.Banks / tau_CB;
godley_table_1.Wages = GDP * omega;
GDP = godley_table_1.Firms / tau_F;

end Monetary_model;
```

```
{Monetary_model}
```

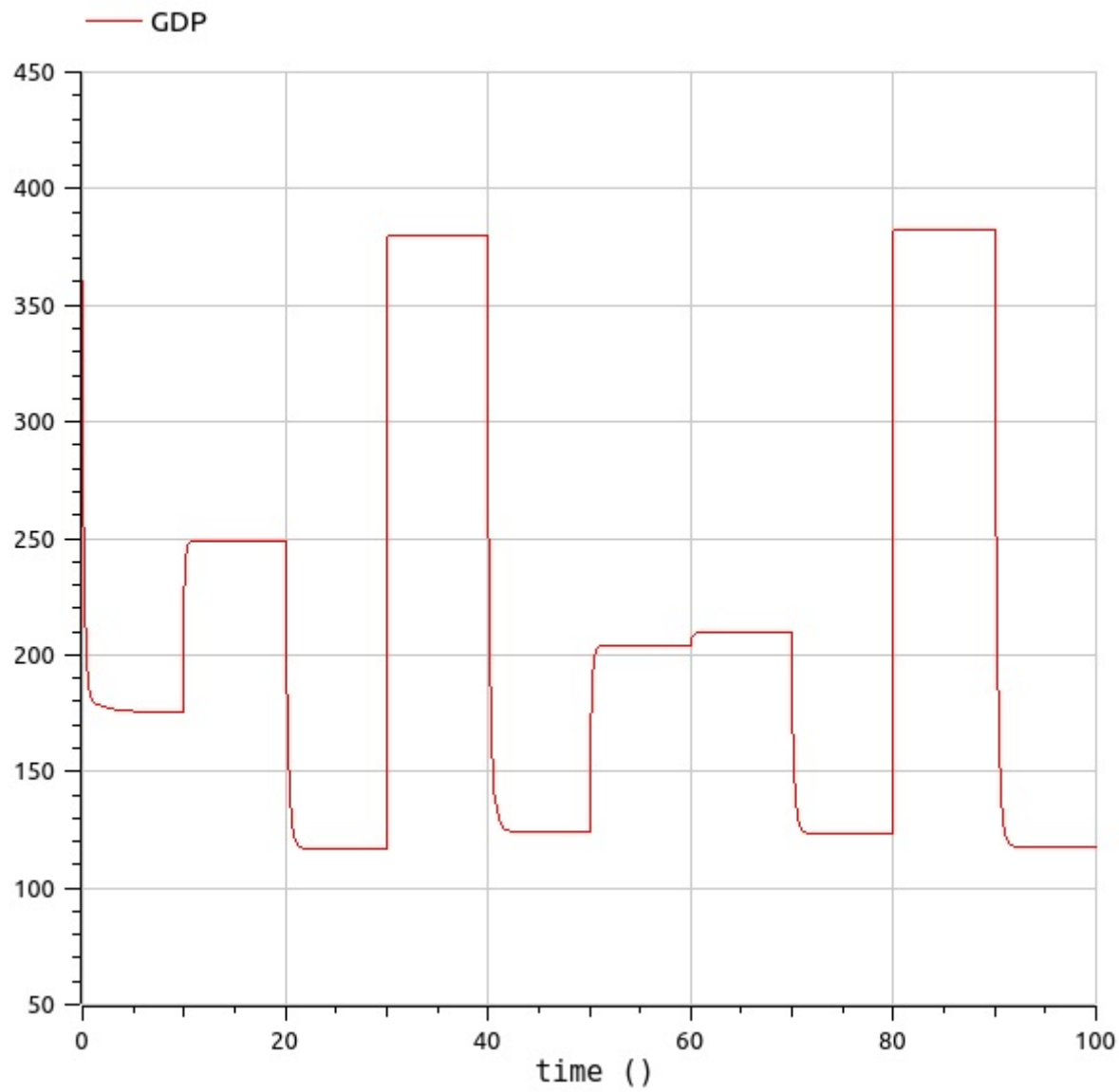
```
simulate(Monetary_model, stopTime = 100);
plot(tau_CW);
```

```
[done]
```




```
plot(GDP);
```

```
[done]
```



Different functions and lengths of intervals can be then defined, so that the value of a parameter changes according to both of them. Notice how the starting point of change can also be modified.

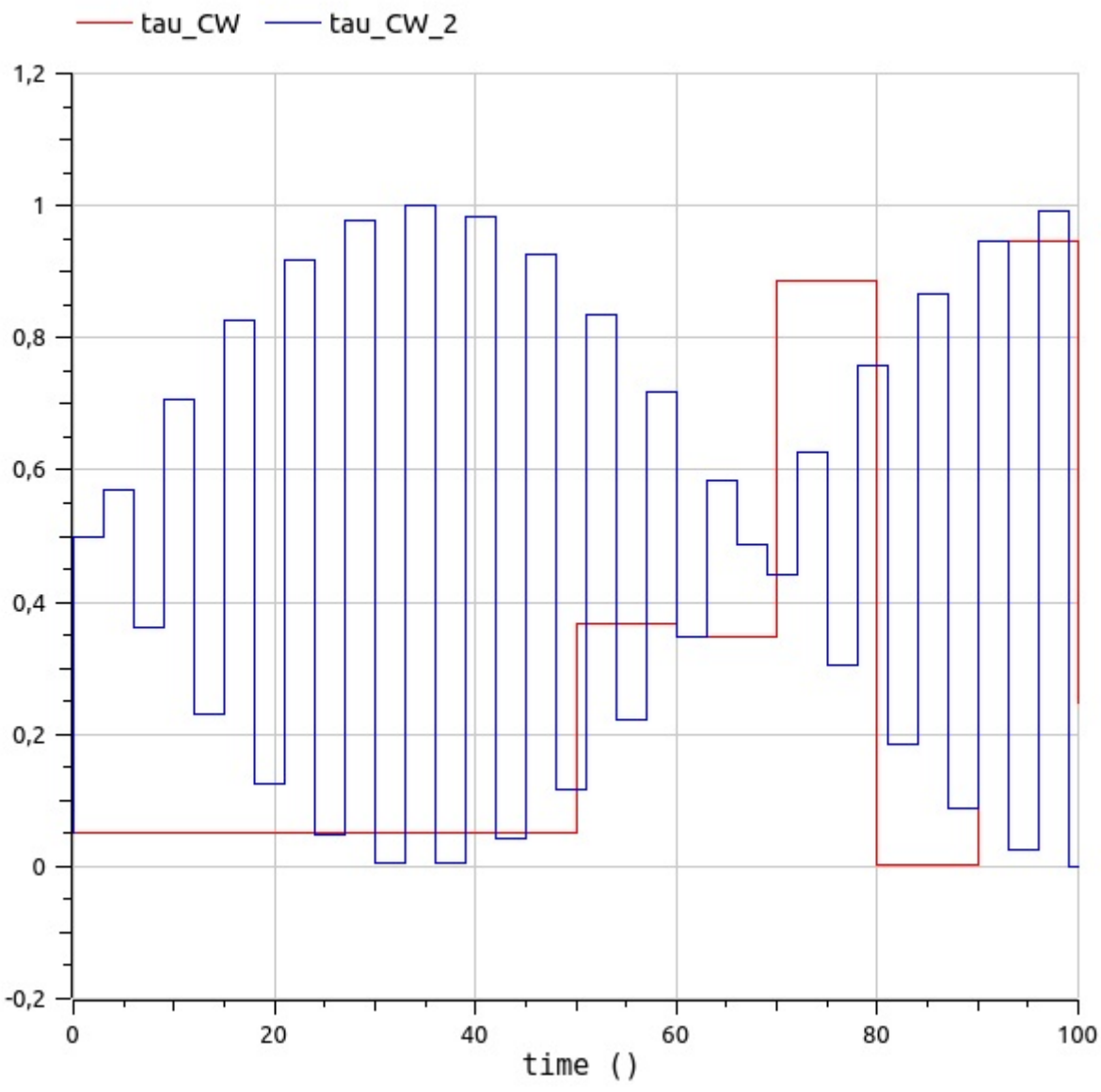
```
model Monetary_model
Godley_table godley_table_1;
parameter Real tau_Repay = 10;
parameter Real tau_Lend = 10;
Real tau_CW (start = 0.05, fixed = true);
Real tau_CW_2(start = 0.05, fixed = true);
parameter Real tau_CB = 2.5;
parameter Real r_Loans = 0.05;
parameter Real omega = 0.6;
parameter Real tau_F = 0.25;
output Real GDP;

equation

when sample(50, 10) then
  tau_CW = (sin(time) + 1) / 2;
end when;
```

```
when sample(0,3) then
  tau_CW_2 = (sin(time) + 1) / 2;
end when;

godley_table_1.Interest_F = r_Loans*godley_table_1.Loans;
godley_table_1.Repay_F = godley_table_1.Loans / tau_Repay;
godley_table_1.Lend_F = godley_table_1.Loans / tau_Lend;
godley_table_1.Consume_W = godley_table_1.Workers / tau_CW;
godley_table_1.Buy_B = godley_table_1.Banks / tau_CB;
godley_table_1.Wages = GDP * omega;
GDP = godley_table_1.Firms / tau_F;

end Monetary_model;
```

```
{Monetary_model}
```

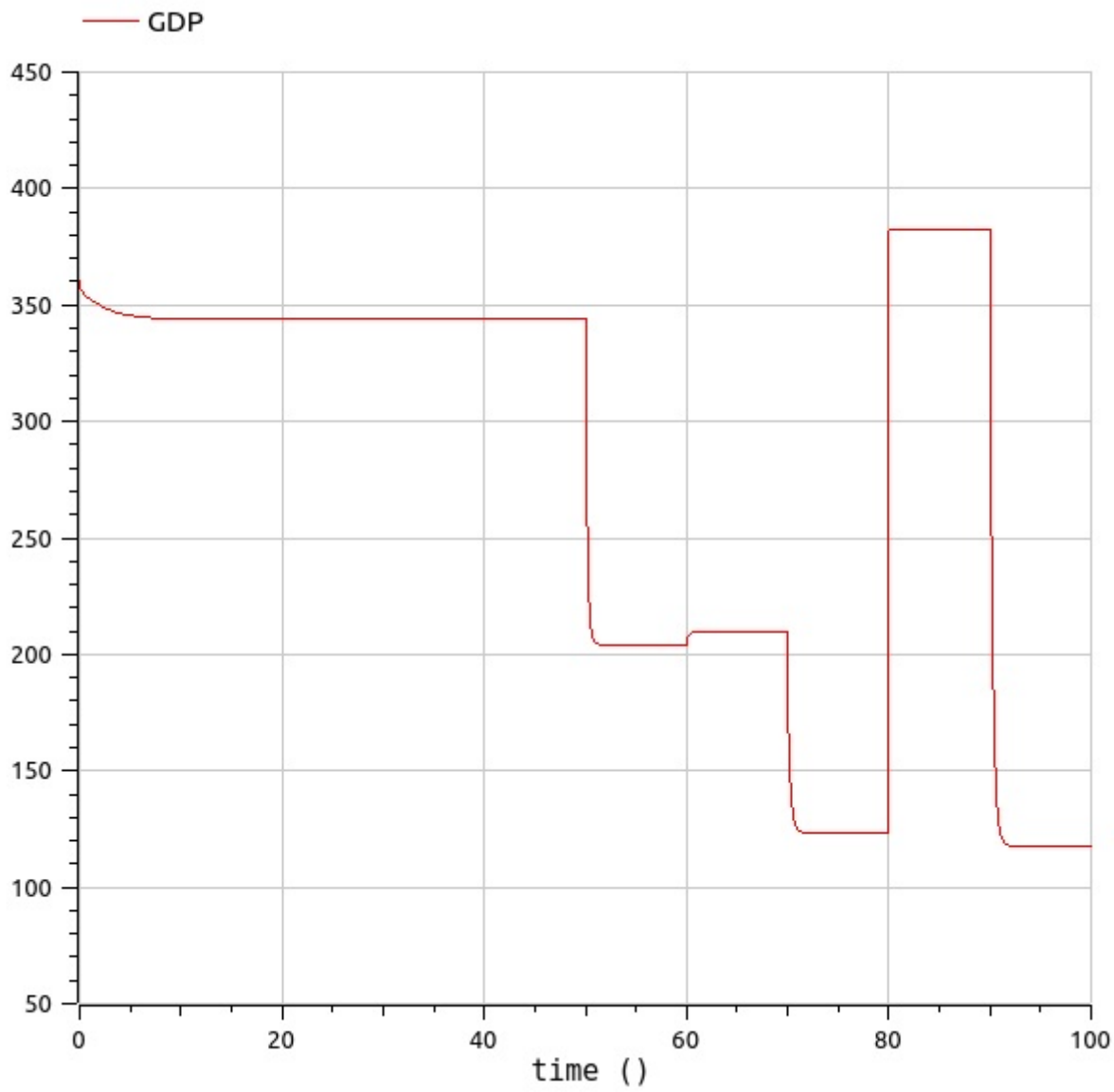
```
simulate(Monetary_model, stopTime = 100);
plot({tau_CW,tau_CW_2});
```

```
[done]
```



```
plot(GDP);
```

```
[done]
```



2.2 Model with Connectors

If this option is chosen, the script will generate a file named *name_of_chosen_filie_block_.mo* and will save it at *files/modelica* directory. For the chosen example, the generated code is the following:

```
model Connected_Godley_table
Modelica.Blocks.Interfaces.RealOutput Loans(start = 110);
Modelica.Blocks.Interfaces.RealOutput Firms(start = 90);
Modelica.Blocks.Interfaces.RealOutput Workers(start = 10);
Modelica.Blocks.Interfaces.RealOutput Banks(start = 10);
Modelica.Blocks.Interfaces.RealInput Interest_F;
Modelica.Blocks.Interfaces.RealInput Lend_F;
Modelica.Blocks.Interfaces.RealInput Buy_B;
Modelica.Blocks.Interfaces.RealInput Repay_F;
Modelica.Blocks.Interfaces.RealInput Wages;
Modelica.Blocks.Interfaces.RealInput Consume_W;
```

```
equation

der(Loans) = (Lend_F) + (-Repay_F);
der(Firms) = (Lend_F) + (-Interest_F) + (-Repay_F) + (-Wages) +
(Consume_W) + (Buy_B);
der(Workers) = (Wages) + (-Consume_W);
der(Banks) = (Interest_F) + (-Buy_B);
end Connected_Godley_table;
```

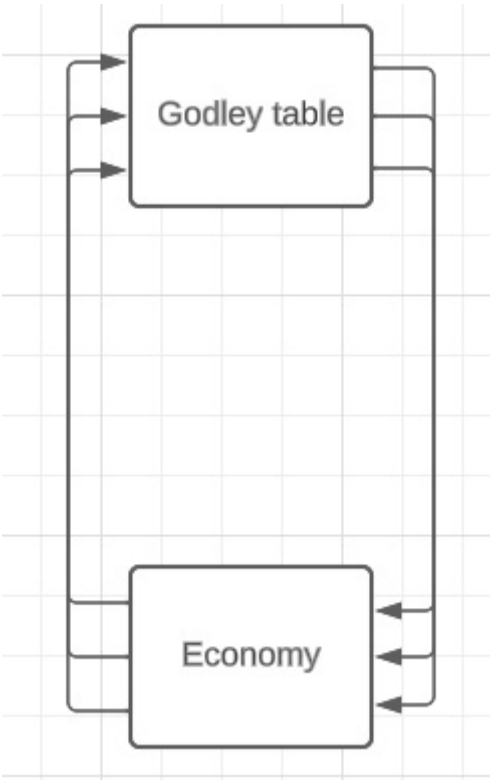
```
{Connected_Godley_table}
```

Notice how the general structure of the code is similar to the previous case; however, the variable **types** are now ***connectors*****,** instead of simply inputs and outputs. Connectors are classes whose instances can be "joint" with each other; this way, isntances can be thought as blocks that are interconnected through input/output ports. To do this, it is necessary to define another class that defines all the remaining variables:

```
model Economy
Modelica.Blocks.Interfaces.RealInput Loans;
Modelica.Blocks.Interfaces.RealInput Firms;
Modelica.Blocks.Interfaces.RealInput Workers;
Modelica.Blocks.Interfaces.RealInput Banks;

Modelica.Blocks.Interfaces.RealOutput Interest_F;
Modelica.Blocks.Interfaces.RealOutput Lend_F;
Modelica.Blocks.Interfaces.RealOutput Buy_B;
Modelica.Blocks.Interfaces.RealOutput Repay_F;
Modelica.Blocks.Interfaces.RealOutput Wages;
Modelica.Blocks.Interfaces.RealOutput Consume_W;
parameter Real tau_Repay = 10;
parameter Real tau_Lend = 10;
parameter Real tau_CW = 0.05;
parameter Real tau_CB = 2.5;
parameter Real r_Loans = 0.05;
parameter Real omega = 0.6;
parameter Real tau_F = 0.25;
output Real GDP;

equation
Interest_F = r_Loans*Loans;
Repay_F = Loans / tau_Repay;
Lend_F = Loans / tau_Lend;
Consume_W = Workers / tau_CW;
Buy_B = Banks / tau_CB;
Wages = GDP * omega;
GDP = Firms / tau_F;

end Economy;
```

```
{Economy}
```

To understand all of this better, this classes can be thought as a "black box" that, given certain inputs, returns certain outputs. If inputs of an instance of this class are connected with ouputs of some instance of the class *Connected_Godley_table* and viceversa, a "circuit" is built representing the whole monetary model.



Finally, it is necessary to actually "connect" all connectors of each of the classes. To do so, the following class is defined:

```
model Monetary_model
Connected_Godley_table godley_table_1;
Economy economic_system;

equation
connect(economic_system.Loans, godley_table.Loans);
connect(economic_system.Firms, godley_table.Firms);
connect(economic_system.Workers, godley_table.Workers);
connect(economic_system.Banks, godley_table.Banks);

connect(godley_table_1.Interest_F, complementary_model.Interest_F);
connect(godley_table_1.Lend_F, complementary_model.Lend_F);
connect(godley_table_1.Buy_B, complementary_model.Buy_B);
connect(godley_table_1.Repay_F, complementary_model.Repay_F);
connect(godley_table_1.Wages, complementary_model.Wages);
connect(godley_table_1  .Consume_W, complementary_model.Consume_W);
end Monetary_model;
```

```
{Monetary_model}
```

Firstly, two variables are created (*godley_table* and *complementary_model*), which are instances of *Godley_table* and *Connecte_extra* and- trhough the *connect* function, the joint between connectors of both instances is performed. Results are exactly the same as the model without connectors.