MFMod models in Python with ModelFlow

Andrew Burns and Ib Hansen

Apr 02, 2023

CONTENTS

I Introduction 3

1 Introduction 5
- 1.1 Background 5
- 1.2 Early steps to bring the MFMod system to the broader economics community 5
- 1.3 Moving the framework to an open-source footing 6

II Macrostructural Models 7

2 Macrostructural models 9
- 2.1 A system of equations 9
- 2.2 Behavioural equations 10

3 Modelflow and the MFMod models of the World Bank 11
- 3.1 A brief history of ModelFlow 11

III Installation of modelflow 13

4 Installation of Modelflow 15
- 4.1 Installation of Python 15

5 Installation of Modelflow 17
- 5.1 Installation of `modelflow` under Anaconda 17

6 Testing your installation of modelflow 19
- 6.1 Specifying the model 19
- 6.2 Create some data 20
- 6.3 Putting it together 21
- 6.4 Create a scenario and run again 22
- 6.5 Inspect results 22

IV Features 27

7 Useful model instance properties and methods 29
- 7.1 Import the model class 29
- 7.2 Class methods to help in Jupyter Notebook 30
- 7.3 .modelload Load a pre-cooked model, data and descriptions 30
- 7.4 Create a scenario 31
- 7.5 () Simulate on a dataframe 31

7.6 Access results 31
7.7 .current_per, The time frame operations are performed on 33
7.8 Using the index operator [ ] to select and visualize variables. 34
7.9 .plot chart the selected and transformed variables 39
7.10 Plotting inspiration 39
7.11 .draw() Graphical presentation of relationships between variables 40
7.12 dekomp() Attrribution of right hand side variables to change in result. 41
7.13 Bespoken plots using matplotlib (or plotly -later) (should go to a separate plot book 44
7.14 Plot four separate plots of multiple series in grid 44

V More **47**

Bibliography **49**

Andrew Burns and Ib hansen

Part I

Introduction

CHAPTER

ONE

INTRODUCTION

Warning: This Jupyter Book is work in progress.

This paper describes the implementation of the World Bank's MacroFiscalModel (MFMod) [BCJ+19] in the open source solution program ModelFlow (Hansen, 2023).

1.1 Background

The impetus for this paper and the work that it summarizes was to make available to a wider constituency the work that the Bank has done over the past several decades to disseminate Macro-structural models[1] – notably those that form part of its MFMod (MacroFiscalModel) framework.

MFMod is the World Bank's work-horse macro-structural economic modelling framework. It exists as a linked system of 184 country specific models that can be solved either independently or as a larger system. The MFMod system replaced the Bank's RMSIM-X model {cite}p:`addison_world_2019` **not in the bib file** and evolved from earlier macro-structural models developed by the Bank during the 2000s to strengthen the basis for the forecasts produced by the World Bank.

Some examples of these models were released on the World Bank's isimulate platform early in 2010 along with several CGE models dating from this period.

Beginning in 2015, the core model that was developed for the isimulate platform was developed and extended substantially into the main MFMod (MacroFiscalModel) model that is used today for the World Bank's twice annual forecasting exercise The Macro Poverty Outlook. This model continues to evolve and be used as the workhorse forecasting and policy simulation model of the World Bank.

1.2 Early steps to bring the MFMod system to the broader economics community

Bank staff were quick to recognize that the models built for its own needs could be of use to the broader economics community. An initial project `isimulate` in 2007 made several versions of this earlier model available for simulation on the isimulate platform, and these models continue to be available there. The isimulate platform housed (and continues to house) public access to earlier versions of the MFMod system, and allows simulation of these and other models – but does not give researchers access to the code or the ability to construct complex simulations.

[1] Economic modelling has a long tradition at the World Bank. The Bank has had a long-standing involvement in CGE modeling is the World Bank [DJ13], indeed the popular mathematics package GAMS, which is widely used to solve CGE and Linear Programming models, started out as a project begun at the World Bank in the 1970s.

In another effort to make models widely available a large number (more than 60 as of June 2023) customized stand-alone models (collectively known as called MFModSA - MacroFiscalModel StandAlones) have been developed from the main model. Typically developed for a country-client (Ministry of Finance, Economy or Planning or Central Bank), these Stand Alones extend the standard model by incorporating additional details not in the standard model that are of specific import to different economies and the country-clients for whom they were built, including: a more detailed breakdown of the sectoral make up of an economy, more detailed fiscal and monetary accounts, and other economically important features of the economy that may exist only inside the aggregates of the standard model.

Training and dissemination around these customized versions of MFMod have been ongoing since 2013. In addition to making customized models available to client governments, Bank teams have run technical assistance program designed to train government officials in the use of these models and their maintenance, modification and revision.

1.2.1 Climate change and the MFMod system

Most recently, the Bank has extended the standard MFMod framework to incorporate the main features of climate change ([BJS21a])– both in terms of the impact of the economy on climate (principally through green-house gas emissions, like $CO_2, N_2O, CH_4, ...$) and the impact of the changing climate on the economy (higher temperatures, changes in rainfall quantity and variability, increased incidence of extreme weather) and their impacts on the economy (agricultural output, labor productivity, physical damages due to extreme weather events, sea-level rises etc.).

These climate enhanced versions of MFMod serve as one of the two main modelling systems (along with the Bank's MANAGE CGE system) in the World Bank's Country Climate Development Reports

1.3 Moving the framework to an open-source footing

Models in the MFMod family are normally built using the proprietary EViews econometric and modelling package. While offering many advantages for model development and maintenance, its cost may be a barrier to clients in developing countries. As a result, the World Bank joined with Ib Hansen, a Danish economist formerly with the European Central Bank and the Danish Central Bank, who over the years has developed `modelflow` a generalized solution engine written in Python for economic models. Together with World Bank, Hansen has worked to extend `modelflow` so that MFMod models can be ported and run in the framework.

This paper reports on the results of these efforts. In particular, it provides step by step instructions on how to install the `modelflow` framework, import a World Bank macrostructural model, perform simulations with that model and report results using the many analytical and reporting tools that have been built into `modelflow`. It is not a manual for `modelflow`, such a manual can be found here nor is it documentation for the MFMod system, such documentation can be found here Burns *et al.* and here [BJS21b], [BJS21a]). Nor is it documentation for the specific models described and worked with below.

Part II

Macrostructural Models

CHAPTER

TWO

MACROSTRUCTURAL MODELS

The economics profession uses a wide range of models for different purposes. Macro-structural models (also known as semi-structural or Macro-econometric models) are a class of models that seek to summarize the most important interconnections and determinants of an economy. Computable General Equilibrium (CGE), and Dynamic Stochastic General Equilibrium (DSGE) models are other classes of models that also seek, using somewhat different methodologies, to capture the main economic channels by which the actions of agents (firms, households, governments) interact and help determine the structure, level and rate of growth of economic activity in an economy. Olivier Blanchard, former Chief Economist at the International Monetary Fund, in a series of articles published between 2016 and 2018 that were summarized in [Bla18]. In these articles he lays out his views on the relative strengths and weaknesses of each of these systems, concluding that each has a role to play in helping economists analyze the macro-economy.

Typically organizations, including the World Bank, use all of these tools, privileging one or the other for specific purposes. Macrostructural models like the MFMod framework are widely used by Central Banks, Ministries of Finance; and professional forecasters both for the purposes of generating forecasts and policy analysis.

2.1 A system of equations

Macro-structural models are a system of equations comprised of two kinds of equations and three kinds of variables.

- `Identities` are variables that are determined by a well defined accounting rule that always holds. The famous GDP Identity Y=C+I+G+(X-M) is one such identity, that indicates that GDP at market prices is definitionally equal to Consumption plus Investment plus Government spending plus Exports less Imports.
- `Behavioural` variables are determined by equations that typically attempt to summarize an economic (vs accounting) relationship. Thus, the equation that says real C = f(Disposable Income,the price level, and animal spirits) is a behavioural equation – where the relationship is drawn from economic theory. Because these equations do not fully explain the variation in the dependent variable and the sensitivities of variables to the changes in other variables are uncertain, these equations and their parameters are typically estimated econometrically and are subject to error.
- `Exogenous` variables are not determined by the model. Typically there are set either by assumption or from data external to the model. For an individual country model, would often be set as an exogenous variable because the level of activity of the economy itself is unlikely to affect the world price of oil.

In a fully general form it can be written as:

$$\begin{aligned}
y_t^1 &= f^1(y_{t+u}^1..., y_{t+u}^n..., y_t^2..., y_t^n...y_{t-r}^1..., y_{t-r}^n, x_t^1...x_t^k, ...x_{t-s}^1..., x_{t-s}^k) \\
y_t^2 &= f^2(y_{t+u}^1..., y_{t+u}^n..., y_t^1..., y_t^n...y_{t-r}^1..., y_{t-r}^n, x_t^1...x_t^k, ...x_{t-s}^1..., x_{t-s}^k) \\
&\vdots \\
y_t^n &= f^n(y_{t+u}^1..., y_{t+u}^n..., y_t^1..., y_t^{n-1}...y_{t-r}^1..., y_{t-r}^n, x_t^1...x_t^r, x..._{t-s}^1..., x_{t-s}^k)
\end{aligned}$$

where y_t^1 is one of n endogenous variables and x_t^1 is an exogenous variable and there are as many equations as there are unknown (endogenous variables).

Rewritten for our GDP identity and substituting the variable mnemonics Y,C,I,G,X,M we could write a simple model as a system of 6 equations in 6 unknowns:

$$\begin{aligned}
Y_t &= C_t + I_t + G + t + (X_t - M_t) \\
C_t &= c_t(C_{t-1}, C_{t-2}, I_t, G_t, X_t, M_t, P_t) \\
I_t &= c_t(I_{t-1}, I_{t-2}, C_t, G_t, X_t, M_t, P_t) \\
G_t &= c_t(G_{t-1}, G_{t-2}, C_t, I_t, X_t, M_t, P_t) \\
X_t &= c_t(X_{t-1}, X_{t-2}, C_t, I_t, G_t, M_t, P_t, P_t^f) \\
M_t &= c_t(M_{t-1}, M_{t-2}, C_t, I_t, G_t, X_t, P_t, P_t^f)
\end{aligned}$$

and where P_t, P_t^f domestic and foreign prices respectively are exogenous in this simple model.

2.2 Behavioural equations

Behavioural equations in a macrostructural equation are typically estimated. In MFMod they are often expressed in Error Correction form. In this approach the behaviour of the dependent variable (say Consumption) is assumed to be the joint product of a long-term economic relationship – usually drawn from economic theory, and various short-run influences which can be more ad hoc in nature. The ECM formulation has the advantage of tieing down the long run behavior of the economy to economic theory, while allowing its short-run dynamics (where short-run can in some cases be 5 or more years) to reflect the way the economy actually operates (not how textbooks say it should behave).

For the consumption equation, utility maximization subject to a budget constraint might lead us to define a long run relationship like this economic theory might lead us to something like this:

$$C_t = \alpha + \beta_1 \frac{rK_t + WB_t + GTR_t(1 - \tau^{Direct})}{PC_t} - \beta_3(r_t - \dot{p}_t) + \eta_t$$

Where in the long run consumption (C_t) for a given interest rate is a stable share of real disposable income ($\frac{rK_t+WB_t+GTR_t}{PC_t}$), implying a constant savings rate. And where real disposable income is given by interest earned on capital (rK_t) plus earnings from labour (WB_t) + Government transfers to households (GTR_t) multiplied by 1 less the direct rate (τ^{Direct}). The final term reflects the influence of real interest rates on final consumption, such that as real interest rates rise consumption as a share of disposable income declines (the savings rate rises).

Replacing the expression following β with Y_t^{disp}, the above simplifies and can be rewritten as:

$$C_t = (\alpha + \beta_1 Y_t^{disp} - \beta_3(r_t - \dot{p}_t))$$

and dividing both sides by Y_t^{disp} gives:

$$\frac{C_t}{Y_t^{disp}} = \beta_1 - \beta_3 \frac{r_t - \dot{p}_t}{Y_t^{disp}}$$

or in logarithms

$$c_{t-1} - y_{t-1}^{disp} - ln(\beta_1) + \beta_3 ln(r_{t-1} - \dot{p}_{t-1} - y_{t-1}^{disp}) = 0$$

we can then write our ECM equation as

$$\Delta c_t = -\lambda(\eta_{t-1}) + SR_t$$

Substituting the LR expression for the error term in t-1 we get

$$\Delta c_t = -\lambda(c_{t-1} - y_{t-1}^{disp} - ln(\beta_1) + \beta_3 ln(r_{t-1} - \dot{p}_{t-1} - y_{t-1}^{disp})) + \beta_{SR1} y_t^{disp} - \beta_{SR2} ln(r_t - \dot{p}_t - y_t^{disp})$$

where β_{SR1} is the short run elasticity of consumption to disposable income; β_{SR2} is the short run real interest rate elasticity of consumption and λ is the speed of adjustment (the rate at which past errors are corrected in each period).

Burns *et al.* provides more complete derivations of the functional forms for most of the behavioural equations in MFmod.

CHAPTER

THREE

MODELFLOW AND THE MFMOD MODELS OF THE WORLD BANK

At the World Bank models built using the MFMod framework are developed in EViews. When disseminated to clients, the models are operated in a World Bank customized EViews environment. But as a systems of equations and associated data the models can be solved, and operated under any system capable of solving a system of simultaneous equations – as long as the equations and data can be transferred from EViews to the secondary system. `Modelflow` is such a system and offers a wide range of features that permit not only solving the model, but also provide a rich and powerful suite of tools for analyzing the model and reporting results.

3.1 A brief history of ModelFlow

Modelflow is a python library that was developed by Ib Hansen over several years while working at the Danish Central Bank and the European Central Bank. The framework has been used both to port the U.S. Federal Reserve's macro-structural model to python, but also been used to bring several stress-testing models developed by European Central Banks and the European Central Bank into a python environment.

Beginning in 2019, Hansen has worked with the World Bank to develop additional features that facilitate working with models built using the Bank's MFMod Framework, with the objective of creating an open source platform through which the Bank's models can be made available to the public.

This paper, and the models that accompany it, are the product of this collaboration.

Part III

Installation of modelflow

CHAPTER

FOUR

INSTALLATION OF MODELFLOW

Modelflow is a python package that defines the `model` class, its methods and a number of other functions that extend and combine pre-existing python functions to allow the easy solution of complex systems of equations including macro-structural models like MFMod. To work with `modelflow`, a user needs to first install python (preferably the Anaconda variant), several supporting packages, and of course the `modelflow` package itself. While `modelflow` can be run directly from the python command-line or IDEs (Interactive Development Environments) like `Spyder` or Microsoft's `Visual Code`, it is suggested that users also install the Jupyter notebook system. Jupyter Notebook facilitates an interactive approach to building python programs, annotating them and ultimately doing simulations using MFMod under `modelflow`. This entire manual and the examples in it were all written and executed in the Jupyter Notebook environment.

4.1 Installation of Python

Python is an extremely powerful and versatile and extensible open-source language. It is widely used for artificial intelligence application, interactive web sites, and scientific processing. As of 14 November 2022, the Python Package Index (PyPI), the official repository for third-party Python software, contained over 415,000 packages that extend its functionality[1]. Modelflow is one of these packages.

Python comes in many flavors and `modelflow` will work with any of them. However, it is strongly suggested that you use the Anaconda version of Python. The remainder of this section points to instructions on how to install the Anaconda version of python (under Windows, MacOS and under Linux). Modelflow works equally well under all three.

This is followed by section that describes the steps necessary to create an anaconda environment with all the necessary packages to run `modelflow`.

4.1.1 Installation of Anaconda under Windows

The definitive source for installing Anaconda under windows can be found here.

It is strongly advised that Anaconda be installed for a single user (Just Me) This is much easier to maintain over time. Installing "For all users on this computer" will substabitally increase the complexity of maintaining python on your computer.

[1] Wikipedia article on python

4.1.2 Installation of Python under macOS

The definitive source for installing Anaconda under macOS can be found here.

4.1.3 Installation of Python under Linux

The definitive source for installing Anaconda under Linux can be found here.

Once Anaconda is fully installed, you can then go to the following instruction on how to install `modelflow` and the packages on which it depends.

CHAPTER

FIVE

INSTALLATION OF MODELFLOW

Warning: The following instructions concern the installation of `modelflow` within an Anaconda installation of python. Different flavors of Python may require slight changes to this recipe, but are not covered here.

`Modelflow` is built and tested using the anaconda python environment. It is strongly recommended to use Anaconda with ````modelflow```.

If you have not already installed Anaconda following the instructions in the preceding chapter, please do so **Now**.

`Modelflow` is a python package that defines the modelflow class `model` among others. `Modelflow1` has many dependencies. Installing the class the first time can take some time depending on your internet connection and computer speed. It is essential that you follow all of the steps outlined below to ensure that your version of `modelflow` operates as expected.

5.1 Installation of `modelflow` under Anaconda

1. Open the anaconda command prompt
2. Execute the following commands by copying and pasting them – either line by line or as a single mult-line step
3. Press enter

```
conda create -n ModelFlow -c ibh -c  conda-forge modelflow_pinned_developement_test -y
conda activate ModelFlow
pip install dash_interactive_graphviz
conda install pyeviews -c conda-forge -y
jupyter contrib nbextension install --user
jupyter nbextension enable hide_input_all/main
jupyter nbextension enable splitcell/splitcellcd
jupyter nbextension enable toc2/main
```

Depending on the speed of your computer and of your internet connection installation could take as little as 10 minutes or more than 1/2 an hour.

At the end of the process you will have a new conda environment ModelFlow, and this will have been activated.

Once modelflow is installed you are ready to work with it. The following sections give a brief introduction to Jupyter notebook, which is a flexible tool that allows us to execute python code, interact with the modelflow class and World Bank Models and annotate what we have done for future replication.

Note: NB: The next time you want to work with modelflow, you will need to activate the `modelflow` environment by

1) Opening the Anaconda command prompt window
2) Activate the ModelFlow environment we just created by executing the folling command

```
conda activate modelflow
```

CHAPTER

SIX

TESTING YOUR INSTALLATION OF MODELFLOW

To test that the installation of modelflow has worked properly, we will build a model using the modelflow framework and then simulate it. A simple model that illustrates many of the functions of modelflow is the Solow growth model.

The code below first sets up the python environment by importing the modelflow and pandas classes. The initial two lines of code and the final two lines just set up the environment for optimal display and are not required.

To test the installation on your system you can copy this code into a Jupyter notebook and execute it.

6.1 Specifying the model

Having loaded the model class from the modelflow library, we can start constructing the model.

The first step is to define the equations of the model, using `modelflow`'s Business Logic Language.

Business Logic Language

More on how to specify models here

The below code segment defines a string fsolow that contains the equations for the solow model, where:

- GDP is defined as a simple Cobb-Douglas production function as the product of TFP, Capital (raised to the share of capital in total income) and Labour (raised to the share of labor in total income)
- Investment is equal to GDP less consumption
- The change in capital is equal to investment this period less the depreciation of the capital stock from the previous period
- Labor grows at the rate of growth of the variable `Labor_growth`
- a pure reporting identity Capital_intensity the ratio of the Capital Stock to the Labor input

We thus have a system of 6 equations with 6 unknowns (GDP, Consumption, Investment, Change in the Capital stock, and change in Labor supply, and the capital_intensity) and exogenous variables (TFP, alfa,savings_rate,Depreciation_rate and Labor_growth).

The equations for Labor and Capital have been entered as difference equations. The `modelflow` object will automatically normalize them, generating an internal representation of `Labour=Labour(t-1)*(1+Labor_growth)` and `Capital=Capital(t-1)*(1-Depreciation_rate)+Investment`

```
fsolow = '''\
GDP            = TFP  * Capital**alfa * Labor **(1-alfa)
Consumption      = (1-saving_rate)  * GDP
Investment       = GDP - Consumption
diff(Capital)    = Investment-Depreciation_rate * Capital(-1)
diff(Labor)      = Labor_growth * Labor(-1)
Capital_intensity = Capital/Labor
'''
```

To create the model we instantiate (create) a variable `msolow` (which will ultimately contain both the equations and data for the model) using the `.from_eq()` method of the `modelflow` class – submitting to it the equations in string form, and giving it the name "Solow model".

```
msolow = model.from_eq(fsolow,modelname='Solow model')
```

The internal representation of the normalized equations can be displayed in normalized business language with the `modelflow` method `.print_model`:

```
msolow.print_model
```

```
FRML <> GDP            = TFP  * CAPITAL**ALFA * LABOR **(1-ALFA)  $
FRML <> CONSUMPTION      = (1-SAVING_RATE)  * GDP  $
FRML <> INVESTMENT       = GDP - CONSUMPTION    $
FRML <> CAPITAL=CAPITAL(-1)+(INVESTMENT-DEPRECIATION_RATE * CAPITAL(-1))$
FRML <> LABOR=LABOR(-1)+(LABOR_GROWTH * LABOR(-1))$
FRML <> CAPITAL_INTENSITY = CAPITAL/LABOR  $
```

6.2 Create some data

For the moment `msolow` has a mathematical representation of a system of equations but no data.

To add data we create a pandas dataframe with initial values for our exogenous variables. Technically capital and labor are endogenous in the Solow model, but because they are specified as change equations their initial values are exogenous and need to be initialized.

The code below instantiates (creates) a panda dataframe `df` and fills it with the variables for our model, initializing these with a series of values over 300 datapoints. The final command displays the first ten rows of the dataframe.

Note:

```
Pandas data frames is a foundational class of python.  There are thousands of web␣
↪sites dedicated to understanding pandas.  Some notable ones include:
```

```
N = 300
df = pd.DataFrame({'LABOR':[100]*N,
                   'CAPITAL':[100]*N,
                   'ALFA':[0.5]*N,
                   'TFP': [1]*N,
                   'DEPRECIATION_RATE': [0.05]*N,
                   'LABOR_GROWTH': [0.01]*N,
```

(continues on next page)

(continued from previous page)

```
                     'SAVING_RATE':[0.05]*N},index=[v for v in range(2000,2300)])
df.head() #this prints out the first 5 rows of the dataframe
```

```
      LABOR  CAPITAL  ALFA  TFP  DEPRECIATION_RATE  LABOR_GROWTH  SAVING_RATE
2000    100      100   0.5    1               0.05          0.01         0.05
2001    100      100   0.5    1               0.05          0.01         0.05
2002    100      100   0.5    1               0.05          0.01         0.05
2003    100      100   0.5    1               0.05          0.01         0.05
2004    100      100   0.5    1               0.05          0.01         0.05
```

6.3 Putting it together

Having defined an initial data set for all the exogenous variables, we can combine these with the equations and solve the model.

The command below solves the model `msolow` on the data contained in the dataframe `df` and stores the output in a new dataframe called `result`.

The last line displays the values of the simulated model, which now includes results for the endogenous variables, and different values for the Labor and Capital variables reflecting their endogeneity for periods 2 through 300.

```
result = msolow(df,keep='Baseline')
# The model is simulated for all years possible

result.head(10)
```

```
           LABOR     CAPITAL  ALFA  TFP  DEPRECIATION_RATE  LABOR_GROWTH  \
2000  100.000000  100.000000   0.5  1.0               0.05          0.01
2001  101.000000  100.025580   0.5  1.0               0.05          0.01
2002  102.010000  100.076226   0.5  1.0               0.05          0.01
2003  103.030100  100.151443   0.5  1.0               0.05          0.01
2004  104.060401  100.250762   0.5  1.0               0.05          0.01
2005  105.101005  100.373733   0.5  1.0               0.05          0.01
2006  106.152015  100.519926   0.5  1.0               0.05          0.01
2007  107.213535  100.688931   0.5  1.0               0.05          0.01
2008  108.285671  100.880357   0.5  1.0               0.05          0.01
2009  109.368527  101.093830   0.5  1.0               0.05          0.01

      SAVING_RATE         GDP  INVESTMENT  CONSUMPTION  CAPITAL_INTENSITY
2000         0.05    0.000000    0.000000     0.000000           0.000000
2001         0.05  100.511609    5.025580    95.486029           0.990352
2002         0.05  101.038487    5.051924    95.986562           0.981043
2003         0.05  101.580575    5.079029    96.501546           0.972060
2004         0.05  102.137821    5.106891    97.030930           0.963390
2005         0.05  102.710176    5.135509    97.574667           0.955022
2006         0.05  103.297593    5.164880    98.132713           0.946943
2007         0.05  103.900030    5.195002    98.705029           0.939144
2008         0.05  104.517449    5.225872    99.291576           0.931613
2009         0.05  105.149813    5.257491    99.892323           0.924341
```

6.4 Create a scenario and run again

dataframe.upd

When importing modelclass all pandas dataframes are enriched with a a handy way to create a new pandas dataframe as a copy of an existing one but with one or more series updated.

In this case df.upd will create a a new dataframe `dfscenaario` with updated LABOR_GROWTH

For more detail on the `.upd` method look here here

```
dfscenario = df.mfcalc('<2023 2200> LABOR_GROWTH = LABOR_GROWTH + 0.002')  # create a␣
↪new dataframe, increase LABOR_GROWTH by 0.002
scenario   = msolow(dfscenario,keep='Higher labor growth ') # simulate the model
```

6.5 Inspect results

`Modelflow` includes a range of methods to view data and results, either as graphs or as tables. Some of these are part of standard python, others are additional features that `modelflow` makes available.

Scenario results can be inspected either by referring to the scenario name given in the (optional) `keep` statement when the model was solved, by referring to the `basedf` and the `lastdf`.

- `basedf` is a dataframe that is automatically generated when the model is solved and contains a copy of the initial conditions of the model prior to the shock.
- `lastdf`is a dataframe that is automatically generated when the model is solved and contains a copy of the results from the simulation. Several built in display functions use these functions to display results.

Finally one could also look at the dataframe to which the results of the simulation were assigned `scenario` in the example above.

Below is a small sub-set of the visualization options available.

6.5.1 Graphical representations of results

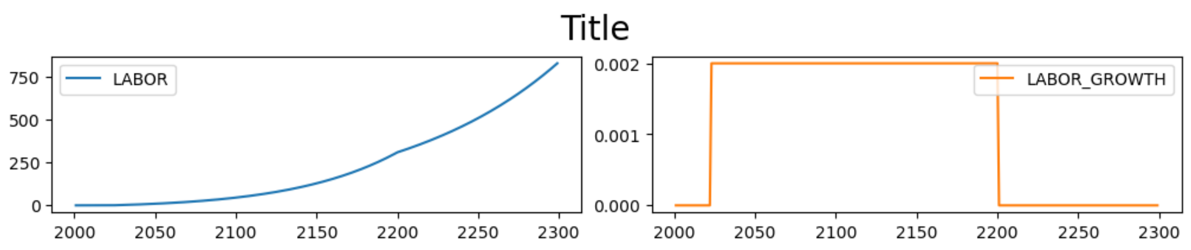
The .dif.plot() method

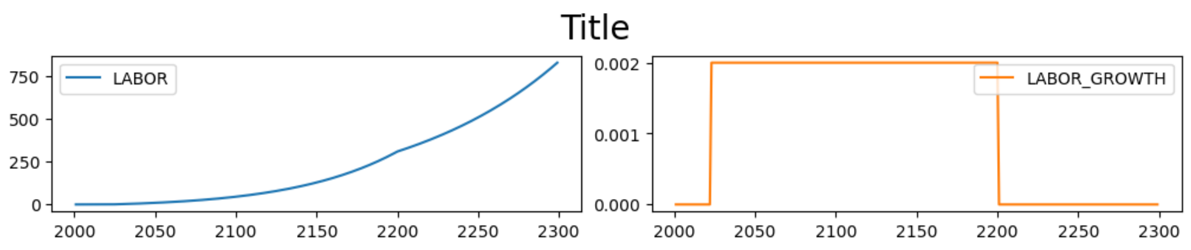
The `.dif.plot` method will plot the change in the level of requested variables. Requested variables can be selected either directly by name or using wildcards.

In this example, a wild card specification is used, requesting the display of all variables that begin with the text ‘labor’. Note that the selector is not case sensitive.

In this case we are displaying changes into the labor and labor growth variables due to the shock when we increased the growth rate of labor by .0002

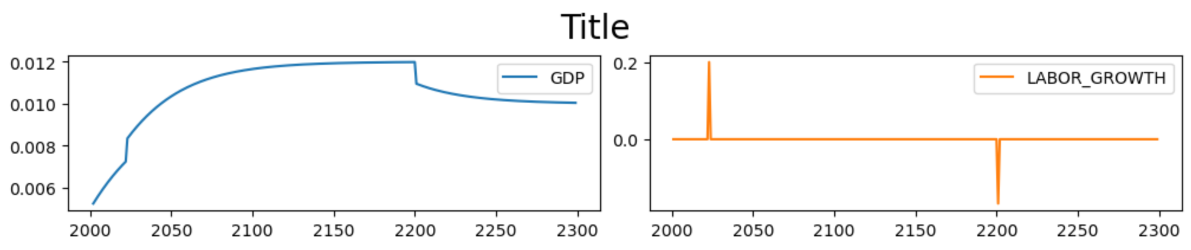
```
msolow['labor*'].dif.plot()
```





In this example, instead of using a wild card selector we requested a variable explicitly by name.

```
msolow['GDP LABOR_GROWTH'].pct.plot()
```



Using the kept solutions

Because the keyword `keep` was used when running the simulations, we can refer to the scenarios by their names – or produce graphs from multiple scenarios – not just the first and last.

```
msolow.keep_plot('GDP')
```

```
{'GDP': <Figure size 1000x600 with 1 Axes>}
```

6.5.2 Textual and tabular display of results

Standard pandas syntax can be used to display data in the results dataframes.

Here we use the standard pandas `.loc` method to display every 10th data point for consumption from the results dataframe, beginning from observation 50 through 100.

```
msolow.lastdf.loc[50:100:10,'CONSUMPTION']
```

```
Series([], Name: CONSUMPTION, dtype: float64)
```

The `.dif.df` method

The `.dif.df` method prints out the changes in variables, i.e. eh difference between the level of specified variables in the `lastdf` dataframe vs the `basedf` dataframe.

```
msolow['GDP CONSUMPTION'].dif.df
```

```
             GDP  CONSUMPTION
2001    0.000000     0.000000
2002    0.000000     0.000000
2003    0.000000     0.000000
2004    0.000000     0.000000
2005    0.000000     0.000000
...          ...          ...
2295  665.334581   632.067852
2296  672.097592   638.492713
2297  678.925939   644.979642
2298  685.820324   651.529308
2299  692.781453   658.142380

[299 rows x 2 columns]
```

The `.difpct.df` method

The `.dif.pct.df` method express the changes between the last simulation and base simulation results as a percent differences in the level ($\frac{\Delta X_t}{X_{t-1}^{basedf}}$). In the example below the mul100 method multiplies the result by 100.

```
msolow['GDP CONSUMPTION'].difpct.mul100.df
```

```
           GDP  CONSUMPTION
2001       NaN          NaN
2002  0.000000     0.000000
2003  0.000000     0.000000
2004  0.000000     0.000000
2005  0.000000     0.000000
...        ...          ...
2295  0.005047     0.005047
2296  0.004892     0.004892
2297  0.004742     0.004742
2298  0.004596     0.004596
2299  0.004456     0.004456

[299 rows x 2 columns]
```

6.5.3 Interactive display of impacts

When working within Jupyter notebook the [ ] command will produce (without the .df termination) will generate a widget with the results expressed as level differences, percent differences, differences in the growth rate – both graphically and in table form.

Please consult here for a fuller presentation of the display routines built into `modelflfow`.

```
msolow['GDP CONSUMPTION']
```

```
Tab(children=(Tab(children=(HTML(value='<?xml version="1.0" encoding="utf-8"␣
↪standalone="no"?>\n<!DOCTYPE svg …
```

Part IV

Features

CHAPTER

SEVEN

USEFUL MODEL INSTANCE PROPERTIES AND METHODS

The focus of this chapter is to introduce some properties and methods of the model instance.

First a model and data is loaded, then a scenario is run. Then we have some content to use.

A model instance gives the user access to a number of properties and methods which helps in managing the model and its results.

If `mmodel` is a model instance `mmodel.<property>` will return a property. Some properties can also be assigned by the user just by:

> mmodel.property = something

The model class itself also have a few properties. These are simple accessed by `model.<property>`.

Enjoy

7.1 Import the model class

This class incorporates most of the methods used to manage a model.

Assuming the ModelFlow library has been installed on your machine, the following imports set up your notebook so that you can run the cells in this notebook.

In order to manipulate plots later on matplotlib.pyplot is also imported.

```
#%matplotlib notebook
%matplotlib inline
```

```
from modelclass import model
```

```
import matplotlib.pyplot as plt # To manipulate plots
```

7.2 Class methods to help in Jupyter Notebook

7.2.1 .widescreen() use Jupyter Notebook in widescreen

Enables the whole viewing area of the browser.

```
model.widescreen()
```

```
<IPython.core.display.HTML object>
```

7.2.2 .scroll_off() Turn off scroll cells in Jupyter Notebook

Can be useful

```
model.scroll_off()
```

7.3 .modelload Load a pre-cooked model, data and descriptions

In this notebook, we will be using a pre-existing model of Pakistan.

The file 'pak.pcim' has been created from a Eviews workspace. It contains all that is needed to run the model:

- Model equations
- Data
- Simulation options
- Variable descriptions

Using the 'modelload' method of the 'model' class, a model instance 'mpak' and a 'result' DataFrame is created.

```
mpak,baseline = model.modelload('../models/pak.pcim',run=1,silent=1,keep='Baseline')
```

```
file read:  C:\modelflow manual\papers\mfbook\content\models\pak.pcim
```

mpak The *modelload* method processes the file and initiates the model, that we call 'mpak' (m for model and pak for Pakistan) with both equations and the data.

'mpak' is an **instance** of the model object with which we will work.

baseline 'result' is a Pandas dataframe containing the data that was loaded.

run=1 the model is simulated. The simulation timeframe and options from the time the file where dumped will be used. The two objects **mpak.basedf** and **mpak.lastdf** will contain the simulation result. If run=0 the model will not be simulated.

silent=1 if silent is set to 0 information regarding the simulation will be displayed.

keep='Baseline' This saves the result in a dictionary mpak.keep_solutions.

7.4 Create a scenario

Many objects relates to comparison of different scenarios. So first a scenario is created by updating some exogenous variables. In this case the carbon tax rates for gas, oil and coal are all set to 29 from 2023 to 2100. Then the scenario is simulated. Now the mpak object contains a number of useful properties and methods.

You can find more on this experiment here

```
scenario_exo  =  baseline.upd("<2020 2100> PAKGGREVCO2CER PAKGGREVCO2GER␣
↪PAKGGREVCO2OER = 29")
```

7.5 () Simulate on a dataframe

When calling the model instance like `mpak(dataframe,start, end)` the model will be simulated for the time frame `start to end` using the dataframe. Just above we created a dataframe `scenario_exo` where the tax variables are updated. Now the `mpak` can be simulated. We simulate from 2020 to 2100.

```
scenario = mpak(scenario_exo,2020,2100,keep=f'Coal, Oil and Gastax : 29') # runs the␣
↪simulation
```

7.6 Access results

Now we have two dataframes with results `baseline` and `scenario`. These dataframes can be manipulated and visualized with the tools provided by the **pandas** library and other like **Matplotlib** and **Plotly**. However to make things easy the first and latest simulation result is also in the mpak object:

- **mpak.basedf**: Dataframe with the values for baseline
- **mpak.lastdf**: Dataframe with the values for alternative

This means that .basedf and .lastdf will contain the same result after the first simulation. If new scenarios are simulated the data in .lastdf will then be replaced with the latest results.

These dataframes are used by a number of model instance methods as you will see later.

The user can assign dataframes to both .basedf and .lastdf. This is useful for comparing simulations which are not the first and last.

```
print(f'mpak.basedf: Dataframe: with {mpak.basedf.shape[0]} years and {mpak.basedf.
↪shape[1]} variables')
print(f'mpak.lastdf: Dataframe: with {mpak.lastdf.shape[0]} years and {mpak.lastdf.
↪shape[1]} variables')
```

```
mpak.basedf: Dataframe: with 121 years and 1290 variables
mpak.lastdf: Dataframe: with 121 years and 1290 variables
```

7.6.1 .keep_solutions, A dictionary of dataframes with results

Create a dictionary of dataframes with .keep_solutions. Sometimes we want to be able to compare more than two scenarios. Using `keep='some description'` the dataframe with results can be saved into a dictionary with the description as key and the dataframe as value.

In our example we have created two scenarios. A baseline and a scenario with the tax set to 29. So mpak.keep_solutions looks like this:

```
print('mpak.keep_solutions contains:')
for key,value in mpak.keep_solutions.items():
    print(f'key = {key:25}|Dataframe: {value.shape[0]} years and {value.shape[1]}␣
↪variables')
```

```
mpak.keep_solutions contains:
key = Baseline                 |Dataframe: 121 years and 1290 variables
key = Coal, Oil and Gastax : 29|Dataframe: 121 years and 1290 variables
```

Sometime it can be useful to reset the `.keep_solutions`, so that a new set of solutions can be inspected. This is done by replacing it with an empty dictionary. Two methods can be used:

mpak.keep_solutions = { }

or in the simulation call:

mpak(,,keep=”)

7.6.2 More on manipulating keep_solution:

Here

7.6.3 .oldkwargs, Options in the simulation call is persistent between calls

When simulating a model the parameters are persistent. So the user just have to provide the solution options once. These persistent parameters are located in the property .oldkwargs.

In this case the persistent parameters are:

```
mpak.oldkwargs
```

```
{'silent': 1, 'alfa': 0.7, 'ldumpvar': 0, 'keep': 'Coal, Oil and Gastax : 29'}
```

The user may have to reset the parameters, this is done like this:

To reset the options just do:

mpak.oldkwargs = { }

7.7 .current_per, The time frame operations are performed on

Most operations on a model class instance operates on the current time frame. It is a subset of the row index of the dataframe which is simulated.

In this case it is:

```
mpak.current_per
```

```
Int64Index([2020, 2021, 2022, 2023, 2024, 2025, 2026, 2027, 2028, 2029, 2030,
            2031, 2032, 2033, 2034, 2035, 2036, 2037, 2038, 2039, 2040, 2041,
            2042, 2043, 2044, 2045, 2046, 2047, 2048, 2049, 2050, 2051, 2052,
            2053, 2054, 2055, 2056, 2057, 2058, 2059, 2060, 2061, 2062, 2063,
            2064, 2065, 2066, 2067, 2068, 2069, 2070, 2071, 2072, 2073, 2074,
            2075, 2076, 2077, 2078, 2079, 2080, 2081, 2082, 2083, 2084, 2085,
            2086, 2087, 2088, 2089, 2090, 2091, 2092, 2093, 2094, 2095, 2096,
            2097, 2098, 2099, 2100],
           dtype='int64')
```

The possible times in the dataframe is contained in the `<dataframe>.index` property.

```
scenario.index   # the index of the dataframe
```

```
Int64Index([1980, 1981, 1982, 1983, 1984, 1985, 1986, 1987, 1988, 1989,
            ...
            2091, 2092, 2093, 2094, 2095, 2096, 2097, 2098, 2099, 2100],
           dtype='int64', length=121)
```

7.7.1 .smpl, Set time frame

The time frame can be set like this:

```
mpak.smpl(2020,2025)
mpak.current_per
```

```
Int64Index([2020, 2021, 2022, 2023, 2024, 2025], dtype='int64')
```

7.7.2 .set_smpl, Set timeframe for a local scope

For many operations it can be useful to apply the operations for a shorter time frame, but retain the global time frame after the operation. This can be done with a `with` statement like this.

```
print(f'Global time  before    {mpak.current_per}')
with mpak.set_smpl(2022,2023):
    print(f'Local time frame       {mpak.current_per}')
print(f'Unchanged global time {mpak.current_per}')
```

```
Global time  before   Int64Index([2020, 2021, 2022, 2023, 2024, 2025], dtype='int64
↪')
Local time frame      Int64Index([2022, 2023], dtype='int64')
Unchanged global time Int64Index([2020, 2021, 2022, 2023, 2024, 2025], dtype='int64
↪')
```

7.7.3 .set_smpl_relative Set relative timeframe for a local scope

When creating a script it can be useful to set the time frame relative to the current time.

Like this:

```
print(f'Global time  before   {mpak.current_per}')
with mpak.set_smpl_relative (-1,0):
    print(f'Local time frame      {mpak.current_per}')
print(f'Unchanged global time {mpak.current_per}')
```

```
Global time  before   Int64Index([2020, 2021, 2022, 2023, 2024, 2025], dtype='int64
↪')
Local time frame      Int64Index([2019, 2020, 2021, 2022, 2023, 2024, 2025], dtype=
↪'int64')
Unchanged global time Int64Index([2020, 2021, 2022, 2023, 2024, 2025], dtype='int64
↪')
```

7.8 Using the index operator [ ] to select and visualize variables.

The index operator [ ] can be used to select variables and then process the values for quick analysis.

To select variables the method accept patterns which defines variable names. Wildcards:

- `\*` matches everything
- `?` matches any single character
- `\[seq]` matches any character in seq
- `\[!seq]` matches any character not in seq

For more how wildcards can be used, the specification can be found here (https://docs.python.org/3/library/fnmatch.html)

In the following example we are selecting the results of mpak['PAKNYGDPMKTPKN']

This call will return a special class (called `vis`). It implements a number of methods and properties which comes in handy for quick analyses.

Several properties and methods can be chained. An example:

```
with mpak.set_smpl(2020,2100):
    mpak['PAKNYGDPMKTPKN'].difpctlevel.mul100.rename().plot(colrow=1,
                title='Difference to baseline in percent',top=0.8);
```

But first some basic information

7.8.1 model['#ENDO']

Use '#ENDO' to access all endogenous variables in your model instance.

For the sake of space, the result is saved in the variable 'allendo' and not printed.

```
allendo = mpak['#ENDO']
# allendo.show
```

7.8.2 Access values in .lastdf and .basedf

To limit the output printed, we set the time frame to 2020 to 2023.

```
mpak.smpl(2020,2023);
```

To access the values of 'PAKNYGDPMKTPKN' and 'PAKNECONPRVTKN' from the latest simulation a small widget is displayed.

```
mpak['PAKNYGDPMKTPKN PAKNECONPRVTKN']
```

```
Tab(children=(Tab(children=(HTML(value='<?xml version="1.0" encoding="utf-8"␣
↪standalone="no"?>\n<!DOCTYPE svg …
```

To access the values of 'PAKNYGDPMKTPKN' and 'PAKNECONPRVTKN' from the base dataframe, specify .base

```
mpak['PAKNYGDPMKTPKN PAKNECONPRVTKN'].base.df
```

```
      PAKNYGDPMKTPKN  PAKNECONPRVTKN
2020    2.627394e+07    2.367289e+07
2021    2.651137e+07    2.397282e+07
2022    2.668514e+07    2.416413e+07
2023    2.696308e+07    2.442786e+07
```

7.8.3 .df Pandas dataframe

Sometime you need to perform additional operations on the values. Therefor the .df will return a dataframe with the selected variables.

```
mpak['PAKNYGDPMKTPKN PAKNECONPRVTKN'].df
```

```
      PAKNYGDPMKTPKN  PAKNECONPRVTKN
2020    2.647002e+07    2.344055e+07
2021    2.676493e+07    2.366076e+07
2022    2.688965e+07    2.376966e+07
2023    2.708904e+07    2.395330e+07
```

7.8.4 .show as a html table with tooltips

If you want the variable descriptions use this

```
mpak['PAKNYGDPMKTPKN PAKNECONPRVTKN'].show
```

```
Tab(children=(Tab(children=(HTML(value='<?xml version="1.0" encoding="utf-8"␣
↪standalone="no"?>\n<!DOCTYPE svg …
```

7.8.5 .names Variable names

If you select variables using wildcards, then you can access the names that correspond to your query.

```
mpak['PAKNYGDP??????'].names
```

```
['PAKNYGDPDISCCN',
 'PAKNYGDPDISCKN',
 'PAKNYGDPFCSTCN',
 'PAKNYGDPFCSTKN',
 'PAKNYGDPFCSTXN',
 'PAKNYGDPMKTPCD',
 'PAKNYGDPMKTPCN',
 'PAKNYGDPMKTPKD',
 'PAKNYGDPMKTPKN',
 'PAKNYGDPMKTPXN',
 'PAKNYGDPPOTLKN']
```

7.8.6 .frml The formulas

Use .frml to access all the equations for the endogenous variables.

```
mpak['PAKNYGDPMKTPKN PAKNECONPRVTKN'].frml
```

```
PAKNYGDPMKTPKN : FRML <> PAKNYGDPMKTPKN =␣
↪PAKNECONPRVTKN+PAKNECONGOVTKN+PAKNEGDIFTOTKN+PAKNEGDISTKBKN+PAKNEEXPGNFSKN-
↪PAKNEIMPGNFSKN+PAKNYGDPDISCKN+PAKADAP*PAKDISPREPKN $
PAKNECONPRVTKN : FRML <Z,EXO> PAKNECONPRVTKN = (PAKNECONPRVTKN(-
↪1)*EXP(PAKNECONPRVTKN_A+ (-0.2*(LOG(PAKNECONPRVTKN(-1))-LOG(1.21203101101442)-
↪LOG((((PAKBXFSTREMTCD(-1)-PAKBMFSTREMTCD(-1))*PAKPANUSATLS(-1))+PAKGGEXPTRNSCN(-
↪1)+PAKNYYWBTOTLCN(-1)*(1-PAKGGREVDRCTXN(-1)/100))/PAKNECONPRVTXN(-1)))+0.
↪763938860758873*((LOG((((PAKBXFSTREMTCD-
↪PAKBMFSTREMTCD)*PAKPANUSATLS)+PAKGGEXPTRNSCN+PAKNYYWBTOTLCN*(1-PAKGGREVDRCTXN/
↪100))/PAKNECONPRVTXN))-(LOG((((PAKBXFSTREMTCD(-1)-PAKBMFSTREMTCD(-
↪1))*PAKPANUSATLS(-1))+PAKGGEXPTRNSCN(-1)+PAKNYYWBTOTLCN(-1)*(1-PAKGGREVDRCTXN(-
↪1)/100))/PAKNECONPRVTXN(-1))))-0.0634474791568939*DURING_2009-0.
↪3*(PAKFMLBLPOLYXN/100-((LOG(PAKNECONPRVTXN))-(LOG(PAKNECONPRVTXN(-1)))))) )) *␣
↪(1-PAKNECONPRVTKN_D)+ PAKNECONPRVTKN_X*PAKNECONPRVTKN_D $
```

7.8.7 .rename() Rename variables to descriptions

Use .rename() to assign variable descriptions as variable names.

Handy when plotting!

```
mpak['PAKNYGDPMKTPKN PAKNECONPRVTKN'].rename().df
```

```
          Real GDP   HH. Cons Real
2020  2.647002e+07   2.344055e+07
2021  2.676493e+07   2.366076e+07
2022  2.688965e+07   2.376966e+07
2023  2.708904e+07   2.395330e+07
```

7.8.8 Transformations of solution results

When the variables has been selected through the index operator a number of standard data transformations can be performed.

Transformation	Meaning	expression
pct	Growth rates	$\left(\frac{this_t}{this_{t-1}} - 1\right)$
dif	Difference in level	$l - b$
difpct	Differens in growth rate	$\left(\frac{l_t}{l_{t-1}} - 1\right) - \left(\frac{b_t}{b_{t-1}} - 1\right)$
difpctlevel	differens in level in pct of baseline	$\left(\frac{l_t - b_t}{b_t}\right)$
mul100	multiply by 100	$this_t \times 100$

- $this$ is the chained value. Default lastdf but if preseeded by .base the values from .basedf will be used
- b is the values from .basedf
- l is the values from .lastdf

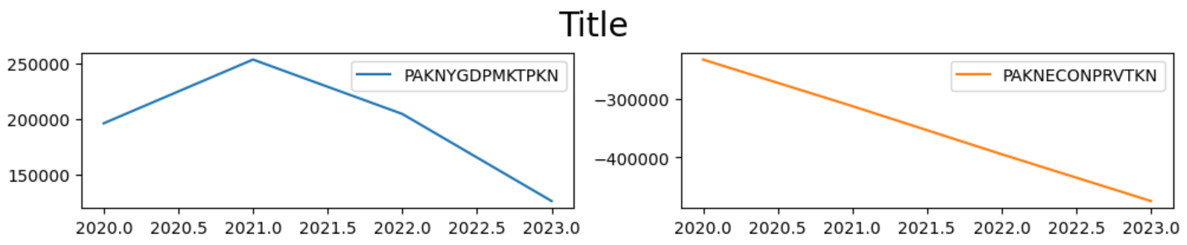
7.8.9 .dif Difference in level

The 'dif' command displays the difference in levels of the latest and previous solutions.

$l - b$

where l is the variable from the .lastdf and b is the variable from .basedf.

```
mpak['PAKNYGDPMKTPKN PAKNECONPRVTKN'].dif.plot()
```



7.8.10 .pct Growthrates

Display growth rates

$$\left(\frac{l_t}{l_{t-1}} - 1\right)$$

```
mpak['PAKNYGDPMKTPKN PAKNECONPRVTKN'].pct.plot();
```

7.8.11 .difpct property difference in growthrate

The difference in the growth rates between the last and base dataframe.

$$\left(\frac{l_t}{l_{t-1}} - 1\right) - \left(\frac{b_t}{b_{t-1}} - 1\right)$$

```
mpak['PAKNYGDPMKTPKN PAKNECONPRVTKN'].difpct.plot() ;
```

7.8.12 .difpctlevel percent difference of levels

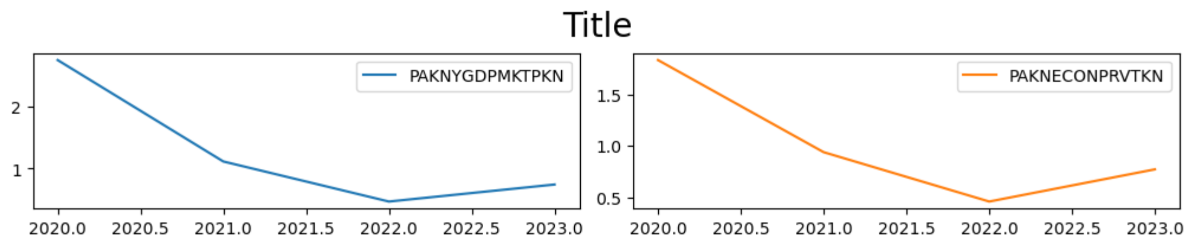
$$\left(\frac{l_t - b_t}{b_t}\right)$$

```
mpak['PAKNYGDPMKTPKN PAKNECONPRVTKN'].difpctlevel.plot();
```

7.8.13 mul100 multiply by 100

multiply growth rate by 100.

```
mpak['PAKNYGDPMKTPKN PAKNECONPRVTKN'].pct.mul100.plot()
```



7.9 .plot chart the selected and transformed variables

After the varaibles has been selected and transformed, they can be plotted. The .plot() method plots the selected variables separately

```
mpak.smpl(2020,2100);

mpak['PAKNYGDP??????'].rename().plot();
```

```
C:\Users\ibhan\miniconda3\envs\mfbooknew\lib\site-packages\pandas\plotting\_
↪matplotlib\tools.py:227: RuntimeWarning: More than 20 figures have been opened.␣
↪Figures created through the pyplot interface (`matplotlib.pyplot.figure`) are␣
↪retained until explicitly closed and may consume too much memory. (To control␣
↪this warning, see the rcParam `figure.max_open_warning`). Consider using␣
↪`matplotlib.pyplot.close()`.
  fig = plt.figure(**fig_kw)
```

7.9.1 Options to plot()

Common:

- title (optional): title. Defaults to ‘’.
- colrow (TYPE, optional): Columns per row . Defaults to 2.
- sharey (TYPE, optional): Share y axis between plots. Defaults to False.
- top (TYPE, optional): Relative position of the title. Defaults to 0.90.

More excotic:

- splitchar (TYPE, optional): If the name should be split . Defaults to ‘__’.
- savefig (TYPE, optional): Save figure. Defaults to ‘’.
- xsize (TYPE, optional): x size default to 10
- ysize (TYPE, optional): y size per row, defaults to 2
- ppos (optional): # of position to use if split. Defaults to -1.
- kind (TYPE, optional): Matplotlib kind . Defaults to ‘line’.

```
mpak['PAKNYGDP??????'].difpct.mul100.rename().plot(title='GDP growth ',top = 0.92);
```

7.10 Plotting inspiration

The following graph shows the components of GDP using the values of the baseline dataframe.

```
mpak['PAKNYGDPMKTPKN PAKNECONPRVTKN PAKNEGDIFTOTKN'].\
difpctlevel.mul100.rename().\
plot(title='Components of GDP in pct of baseline',colrow=1,top=0.90,kind='bar') ;
```

7.10.1 Heatmaps

For some model types heatmaps can be helpful, and they come out of the box. This feature was developed for use by bank stress test models.

```
with mpak.set_smpl(2020,2030):
    heatmap = mpak['PAKNYGDPMKTPKN PAKNECONPRVTKN'].pct.rename().mul100.heat(title=
↪'Growth rates',annot=True,dec=1,size=(10,3))
```

7.10.2 Violin and boxplots,

Not obvious for macro models, but useful for stress test models with many banks.

```
with mpak.set_smpl(2020,2030):
    mpak['PAKNYGDPMKTPKN PAKNECONPRVTKN'].difpct.box()
    mpak['PAKNYGDPMKTPKN PAKNECONPRVTKN'].difpct.violin()
```

7.10.3 Plot baseline vs alternative

A raw routine, only showing levels. To make it really useful it should be expanded.

```
mpak['PAKNYGDPMKTPKN PAKNECONPRVTKN'].plot_alt() ;
```

7.11 .draw() Graphical presentation of relationships between variables

.draw() helps you understand the relationship between variables in your model better.

The thickness the arrow reflect the attribution of the the upstream variable to the impact on the downstream variable.

7.11.1 .draw(up = level, down = level)

You can specify how many levels up and down you want in your graphical presentation (Needs more explanation).

In this example all variables that depend directly upon GDP and consumption as well as those that are determined by them, are displayed. This means one step upstream in the model logic and one step downstream.

More on the how to visualize the logic structure here

```
mpak['PAKNYGDPMKTPKN PAKNECONPRVTKN'].draw(up=1,down=1)   # diagram of all direct␣
↪dependencies
```

```
<IPython.core.display.SVG object>
```

```
<IPython.core.display.SVG object>
```

7.11.2 .draw(filter =<minimal impact>)

By specifying filter= only links where the minimal impact is more than <minimal impact> are show. In this case 20%

```
mpak['PAKNECONPRVTKN'].draw(up=3,down=1,filter=20)
```

```
<IPython.core.display.SVG object>
```

7.12 dekomp() Attrribution of right hand side variables to change in result.

For more information on attribution look here

The dekomp command decomposes the contributions of the right hand side variables to the observed change in the left hand side variables.

```
with mpak.set_smpl(2021,2025):
    mpak['PAKNYGDPMKTPKN PAKNECONPRVTKN'].dekomp()   # frml attribution
```

```
Formula        : FRML  <> PAKNYGDPMKTPKN =␣
 ↪PAKNECONPRVTKN+PAKNECONGOVTKN+PAKNEGDIFTOTKN+PAKNEGDISTKBKN+PAKNEEXPGNFSKN-
 ↪PAKNEIMPGNFSKN+PAKNYGDPDISCKN+PAKADAP*PAKDISPREPKN $

                       2021        2022        2023        2024        2025
Variable    lag
Base        0   26511370.41 26685141.87 26963077.57 27393200.36 27963231.53
Alternative 0   26764926.87 26889649.52 27089036.50 27454422.35 27979057.19
Difference  0     253556.46   204507.65   125958.93    61221.99    15825.66
Percent     0          0.96        0.77        0.47        0.22        0.06

 Contributions to differende for  PAKNYGDPMKTPKN
                         2021       2022       2023       2024       2025
Variable       lag
PAKNECONPRVTKN 0   -312052.97 -394466.14 -474558.93 -531755.17 -563616.01
PAKNECONGOVTKN 0    303335.99  268694.45  232506.87  209988.19  197439.80
PAKNEGDIFTOTKN 0    188565.48  188222.74  177226.36  163571.78  148739.93
PAKNEGDISTKBKN 0        -0.02      -0.01      -0.02      -0.02      -0.05
PAKNEEXPGNFSKN 0     -2911.23   -5414.50   -7960.34  -10272.64  -12204.84
PAKNEIMPGNFSKN 0     76619.12  147471.06  198744.89  229689.74  245466.56
PAKNYGDPDISCKN 0        -0.02      -0.01      -0.02      -0.02      -0.05
PAKADAP        0        -0.02      -0.01      -0.02      -0.02      -0.05
PAKDISPREPKN   0        -0.02      -0.01      -0.02      -0.02      -0.05

 Share of contributions to differende for  PAKNYGDPMKTPKN
                          2021        2022        2023        2024        2025
Variable       lag
PAKNEIMPGNFSKN 0           30%         72%        158%        375%       1551%
PAKNECONGOVTKN 0          120%        131%        185%        343%       1248%
PAKNEGDIFTOTKN 0           74%         92%        141%        267%        940%
PAKNEGDISTKBKN 0           -0%         -0%         -0%         -0%         -0%
PAKNYGDPDISCKN 0           -0%         -0%         -0%         -0%         -0%
PAKADAP        0           -0%         -0%         -0%         -0%         -0%
```

(continues on next page)

(continued from previous page)

```
PAKDISPREPKN   0          -0%         -0%         -0%         -0%         -0%
PAKNEEXPGNFSKN 0          -1%         -3%         -6%        -17%        -77%
PAKNECONPRVTKN 0        -123%       -193%       -377%       -869%      -3561%
Total          0         100%        100%        100%        100%        100%
Residual       0          -0%         -0%         -0%         -0%         -0%

 Contribution to growth rate PAKNYGDPMKTPKN
                         2021        2022        2023        2024        2025
Variable       lag
PAKNECONPRVTKN 0        -0.0%       -0.0%       -0.0%       -0.0%       -0.0%
PAKNECONGOVTKN 0         0.0%        0.0%        0.0%        0.0%        0.0%
PAKNEGDIFTOTKN 0         0.0%        0.0%        0.0%        0.0%        0.0%
PAKNEGDISTKBKN 0        -0.0%       -0.0%       -0.0%       -0.0%       -0.0%
PAKNEEXPGNFSKN 0        -0.0%       -0.0%       -0.0%       -0.0%       -0.0%
PAKNEIMPGNFSKN 0         0.0%        0.0%        0.0%        0.0%        0.0%
PAKNYGDPDISCKN 0        -0.0%       -0.0%       -0.0%       -0.0%       -0.0%
PAKADAP        0        -0.0%       -0.0%       -0.0%       -0.0%       -0.0%
PAKDISPREPKN   0        -0.0%       -0.0%       -0.0%       -0.0%       -0.0%

Formula        : FRML <Z,EXO> PAKNECONPRVTKN = (PAKNECONPRVTKN(-
 ↪1)*EXP(PAKNECONPRVTKN_A+ (-0.2*(LOG(PAKNECONPRVTKN(-1))-LOG(1.21203101101442)-
 ↪LOG((((PAKBXFSTREMTCD(-1)-PAKBMFSTREMTCD(-1))*PAKPANUSATLS(-1))+PAKGGEXPTRNSCN(-
 ↪1)+PAKNYYWBTOTLCN(-1)*(1-PAKGGREVDRCTXN(-1)/100))/PAKNECONPRVTXN(-1)))+0.
 ↪763938860758873*((LOG((((PAKBXFSTREMTCD-
 ↪PAKBMFSTREMTCD)*PAKPANUSATLS)+PAKGGEXPTRNSCN+PAKNYYWBTOTLCN*(1-PAKGGREVDRCTXN/
 ↪100))/PAKNECONPRVTXN))-(LOG((((PAKBXFSTREMTCD(-1)-PAKBMFSTREMTCD(-
 ↪1))*PAKPANUSATLS(-1))+PAKGGEXPTRNSCN(-1)+PAKNYYWBTOTLCN(-1)*(1-PAKGGREVDRCTXN(-
 ↪1)/100))/PAKNECONPRVTXN(-1))))-0.0634474791568939*DURING_2009-0.
 ↪3*(PAKFMLBLPOLYXN/100-((LOG(PAKNECONPRVTXN))-(LOG(PAKNECONPRVTXN(-1)))))) ))  *␣
 ↪(1-PAKNECONPRVTKN_D)+ PAKNECONPRVTKN_X*PAKNECONPRVTKN_D  $

                      2021        2022        2023        2024        2025
Variable    lag
Base        0   23972815.36 24164128.02 24427863.05 24818524.47 25323255.17
Alternative 0   23660762.40 23769661.89 23953304.14 24286769.32 24759639.22
Difference  0    -312052.95  -394466.13  -474558.91  -531755.15  -563615.95
Percent     0         -1.30       -1.63       -1.94       -2.14       -2.23

 Contributions to differende for  PAKNECONPRVTKN
                         2021       2022       2023       2024       2025
Variable         lag
PAKNECONPRVTKN   -1  -187434.07 -250462.33 -317486.72 -384175.74 -432745.55
PAKNECONPRVTKN_A  0       -0.01      -0.01      -0.01      -0.01      -0.04
PAKBXFSTREMTCD   -1   -38694.42  -49412.27  -52084.76  -50817.15  -48170.52
PAKBMFSTREMTCD   -1      120.58     140.33     135.37     121.42     106.31
PAKPANUSATLS     -1     3137.57    3566.27    3817.26    3916.63    3901.04
PAKGGEXPTRNSCN   -1    -2382.89   -4372.94   -5966.91   -7223.04   -8206.86
PAKNYYWBTOTLCN   -1   -78794.18 -120093.04 -145773.43 -156461.75 -167189.37
PAKGGREVDRCTXN   -1       -0.01      -0.01      -0.01      -0.01      -0.04
PAKNECONPRVTXN   -1   204247.65  231199.67  249025.22  258789.48  262005.59
PAKBXFSTREMTCD    0    66466.87   69836.78   67727.29   63861.51   59511.44
PAKBMFSTREMTCD    0     -189.25    -182.00    -162.26    -141.32    -122.29
PAKPANUSATLS      0    -4809.78   -5132.40   -5233.84   -5184.63   -5043.02
PAKGGEXPTRNSCN    0     5895.35    8018.71    9646.92   10900.77   11850.22
PAKNYYWBTOTLCN    0   160980.65  194563.25  207466.32  220404.39  237003.52
PAKGGREVDRCTXN    0       -0.01      -0.01      -0.01      -0.01      -0.04
```

(continues on next page)

(continued from previous page)

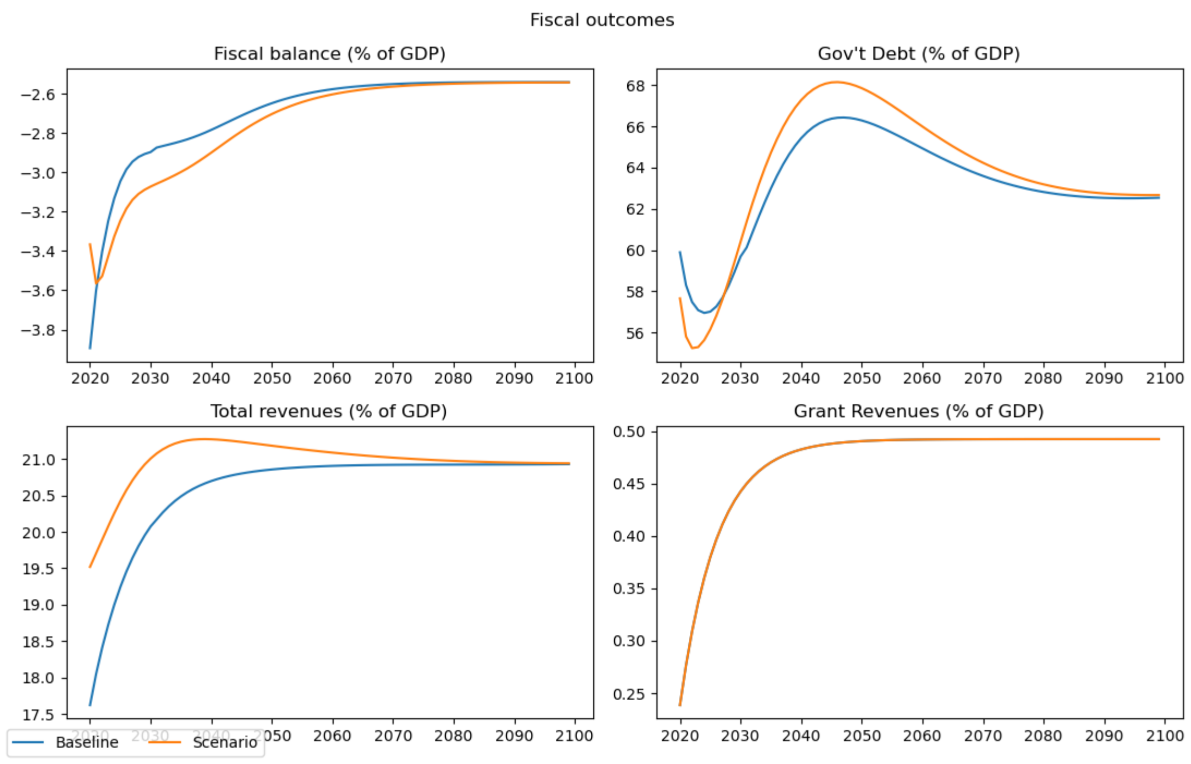
```
PAKNECONPRVTXN    0  -410022.04 -440677.37 -455322.70 -458425.12 -453478.88
DURING_2009       0       -0.01      -0.01      -0.01      -0.01      -0.04
PAKFMLBLPOLYXN    0   -34994.35  -36409.85  -35203.57  -32189.52  -28049.75
PAKNECONPRVTKN_D  0       -0.01      -0.01      -0.01      -0.01      -0.04
PAKNECONPRVTKN_X  0       -0.01      -0.01      -0.01      -0.01      -0.04

 Share of contributions to differende for  PAKNECONPRVTKN
                            2021        2022        2023        2024        2025
Variable         lag
PAKNECONPRVTXN    0         131%        112%         96%         86%         80%
PAKNECONPRVTKN   -1          60%         63%         67%         72%         77%
PAKNYYWBTOTLCN   -1          25%         30%         31%         29%         30%
PAKBXFSTREMTCD   -1          12%         13%         11%         10%          9%
PAKFMLBLPOLYXN    0          11%          9%          7%          6%          5%
PAKGGEXPTRNSCN   -1           1%          1%          1%          1%          1%
PAKPANUSATLS      0           2%          1%          1%          1%          1%
PAKBMFSTREMTCD    0           0%          0%          0%          0%          0%
PAKNECONPRVTKN_A  0           0%          0%          0%          0%          0%
PAKGGREVDRCTXN   -1           0%          0%          0%          0%          0%
                  0           0%          0%          0%          0%          0%
DURING_2009       0           0%          0%          0%          0%          0%
PAKNECONPRVTKN_D  0           0%          0%          0%          0%          0%
PAKNECONPRVTKN_X  0           0%          0%          0%          0%          0%
PAKBMFSTREMTCD   -1          -0%         -0%         -0%         -0%         -0%
PAKPANUSATLS     -1          -1%         -1%         -1%         -1%         -1%
PAKGGEXPTRNSCN    0          -2%         -2%         -2%         -2%         -2%
PAKBXFSTREMTCD    0         -21%        -18%        -14%        -12%        -11%
PAKNYYWBTOTLCN    0         -52%        -49%        -44%        -41%        -42%
PAKNECONPRVTXN   -1         -65%        -59%        -52%        -49%        -46%
Total             0         101%        101%        101%        101%        101%
Residual          0           1%          1%          1%          1%          1%

 Contribution to growth rate PAKNECONPRVTKN
                            2021        2022        2023        2024        2025
Variable         lag
PAKNECONPRVTKN   -1         0.0%        0.0%        0.0%        0.0%        0.0%
PAKNECONPRVTKN_A  0        -0.0%       -0.0%       -0.0%       -0.0%       -0.0%
PAKBXFSTREMTCD   -1        -0.0%       -0.0%       -0.0%       -0.0%       -0.0%
PAKBMFSTREMTCD   -1         0.0%        0.0%        0.0%        0.0%        0.0%
PAKPANUSATLS     -1         0.0%        0.0%        0.0%        0.0%        0.0%
PAKGGEXPTRNSCN   -1        -0.0%       -0.0%       -0.0%       -0.0%       -0.0%
PAKNYYWBTOTLCN   -1        -0.0%       -0.0%       -0.0%       -0.0%       -0.0%
PAKGGREVDRCTXN   -1        -0.0%       -0.0%       -0.0%       -0.0%       -0.0%
PAKNECONPRVTXN   -1         0.0%        0.0%        0.0%        0.0%        0.0%
PAKBXFSTREMTCD    0         0.0%        0.0%        0.0%        0.0%        0.0%
PAKBMFSTREMTCD    0        -0.0%       -0.0%       -0.0%       -0.0%       -0.0%
PAKPANUSATLS      0        -0.0%       -0.0%       -0.0%       -0.0%       -0.0%
PAKGGEXPTRNSCN    0         0.0%        0.0%        0.0%        0.0%        0.0%
PAKNYYWBTOTLCN    0         0.0%        0.0%        0.0%        0.0%        0.0%
PAKGGREVDRCTXN    0        -0.0%       -0.0%       -0.0%       -0.0%       -0.0%
PAKNECONPRVTXN    0        -0.0%       -0.0%       -0.0%       -0.0%       -0.0%
DURING_2009       0        -0.0%       -0.0%       -0.0%       -0.0%       -0.0%
PAKFMLBLPOLYXN    0        -0.0%       -0.0%       -0.0%       -0.0%       -0.0%
PAKNECONPRVTKN_D  0        -0.0%       -0.0%       -0.0%       -0.0%       -0.0%
PAKNECONPRVTKN_X  0        -0.0%       -0.0%       -0.0%       -0.0%       -0.0%
```

7.13 Bespoken plots using matplotlib (or plotly -later) (should go to a separate plot book

The predefined plots are not necessary created for presentation purpose. To create bespoken plots the they can be constructed directly in python scripts. The two main libraries are matplotlib, plotly but any ther python plotting library can be used. Here is an example using matplotlib.

7.14 Plot four separate plots of multiple series in grid

```
figure,axs= plt.subplots(2,2,figsize=(11, 7))
axs[0,0].plot(mpak.basedf.loc[2020:2099,'PAKGGBALOVRLCN_'],label='Baseline')
axs[0,0].plot(mpak.lastdf.loc[2020:2099,'PAKGGBALOVRLCN_'],label='Scenario')
#axs[0,0].legend()

axs[0,1].plot(mpak.basedf.loc[2020:2099,'PAKGGDBTTOTLCN_'],label='Baseline')
axs[0,1].plot(mpak.lastdf.loc[2020:2099,'PAKGGDBTTOTLCN_'],label='Scenario')

axs[1,0].plot(mpak.basedf.loc[2020:2099,'PAKGGREVTOTLCN']/mpak.basedf.loc[2020:2099,
 ↪'PAKNYGDPMKTPCN']*100,label='Baseline')
axs[1,0].plot(mpak.lastdf.loc[2020:2099,'PAKGGREVTOTLCN']/mpak.lastdf.loc[2020:2099,
 ↪'PAKNYGDPMKTPCN']*100,label='Scenario')

axs[1,1].plot(mpak.basedf.loc[2020:2099,'PAKGGREVGRNTCN']/mpak.basedf.loc[2020:2099,
 ↪'PAKNYGDPMKTPCN']*100,label='Baseline')
axs[1,1].plot(mpak.lastdf.loc[2020:2099,'PAKGGREVGRNTCN']/mpak.lastdf.loc[2020:2099,
 ↪'PAKNYGDPMKTPCN']*100,label='Scenario')
#axs2[4].plot(mpak.lastdf.loc[2000:2099,'PAKGGREVGRNTCN']/mpak.basedf.loc[2000:2099,
 ↪'PAKNYGDPMKTPCN']*100,label='Scenario')

axs[0,0].title.set_text("Fiscal balance (% of GDP)")
axs[0,1].title.set_text("Gov't Debt (% of GDP)")
axs[1,0].title.set_text("Total revenues (% of GDP)")
axs[1,1].title.set_text("Grant Revenues (% of GDP)")
figure.suptitle("Fiscal outcomes")

plt.figlegend(['Baseline','Scenario'],loc='lower left',ncol=5)
figure.tight_layout(pad=2.3) #Ensures legend does not overlap dates
figure
```


Fiscal outcomes
Fiscal balance (% of GDP)
Gov't Debt (% of GDP)
Total revenues (% of GDP)
Grant Revenues (% of GDP)
Baseline
Scenario

Part V

More

BIBLIOGRAPHY

[Bla18] Olivier Blanchard. On the future of Macroeconomic models. *Oxford Review of Economic Policy*, 34(1-2):43–54, 2018. URL: https://academic.oup.com/oxrep/article/34/1-2/43/4781808, doi:https://doi.org/10.1093/oxrep/grx045.

[BCJ+19] Andrew Burns, Benoit Campagne, Charl Jooste, David Stephan, and Thi Thanh Bui. *The World Bank Macro-Fiscal Model Technical Description*. Number 8965 in Policy Research Working Papers. World Bank, Washington DC., 2019. URL: https://openknowledge.worldbank.org/handle/10986/32217.

[BJS21a] Andrew Burns, Charl Jooste, and Gregor Schwerhoff. *Climate Modeling for Macroeconomic Policy : A Case Study for Pakistan*. Number 9780 in Policy Research Working Papers. World Bank, Washington, DC, 2021. URL: https://openknowledge.worldbank.org/bitstream/handle/10986/36307/Climate-Modeling-for-Macroeconomic-Policy-A-Case-Study-for-Pakistan.pdf?sequence=1&isAllowed=y.

[BJS21b] Andrew Burns, Charl Jooste, and Gregor Schwerhoff. *Macroeconomic Modeling of Managing Hurricane Damage in the Caribbean: The Case of Jamaica*. Volume 9505 of Policy Research Working Paper. World Bank, Washington DC., 2021. URL: https://documents1.worldbank.org/curated/en/593351609776234361/pdf/Macroeconomic-Modeling-of-Managing-Hurricane-Damage-in-the-Caribbean-The-Case-of-Jamaica.pdf.

[DJ13] Peter B Dixon and DFale W. Jorgenson. *Handbook of Computable General Equilibrium Modelling*. Volume 1A. Elsevier B.V., 2013. ISBN ISSN: 2211-6885. URL: https://www.sciencedirect.com/handbook/handbook-of-computable-general-equilibrium-modeling/.