Modeling Code for USITC Report 'Raspberries for Processing'

Read me:

This model accompanies the USITC report 'Raspberries for Processing: Conditions of Competition between U.S. and Foreign Suppliers, with a Focus on Washington State,' Inv. 332-577. The report includes a quantitative analysis of the impact of increased imports of raspberries for processing to the U.S. in recent years. Economic effects are modeled separately for Individually Quick Frozen (IQF) and non-IQF raspberries for processing. For each product, consumers choose between domestic and imported sources. The model is calibrated to yearly sales between 2010 and 2020. The model is then used to simulate a scenario were imported raspberries for processing did not rise as quickly in the early years of the period of analysis.

The simulation scenario was chosen by looking at year-over-year percentage change in the value of imports to the U.S. For each product, a single set of consecutive years are selected as a 'surge' based on having higher growth rates compared to the rest of the sample. The non-surge average growth rate is calculated using the non-surge years. The percentage changes of the surge years are then reduced by the difference between the surge and the non-surge average. The resulting sequence of yearly percentage changes has the non-surge average for the full 2010-2020 period while still allowing year-to-year changes. The model is then simulated with the no-surge scenario.

The two models share many features:

- CES demand between domestic and imports.
- Imports are aggregated, not differentiated by foreign source.
- Import supply is assumed to be fixed (perfectly inelastic).

In the IQF model:

- Domestic supply has asymptotic marginal costs.
- Pricing and production is approximated by the domestic market having some pricing power.

In the non-IQF model:

- Domestic supply has a constant price elasticity of supply.
- Pricing and production are solved by the competitive equilibrium.

This model file requires the data files 'rr_imports.xlsx', 'raspberries_census_revisions_by_district.xlsx', and '12182020_raspberry_pricing_wrrc.xlsx' to be installed in the same folder as this file.

This model file requires `data_processing.py`, `rr_outputs.py`, `pype_perfect.py`, and `helper.py` to be installed in the same folder as this file. These files contain functions that are called in this notebook.

Model Code

Import Modules

To run these Commission models you will need the following Python packages.

In [1]:
```
import numpy as np
from scipy import stats
import os
```

These additional Python scripts are required to perform various background functions for the model.

`data_processing` processes the import and domestic data and econometrically estimates the elastasticity of substitution parameter.

`rr_outputs` produces tables and figures based on the model results.

`helper` provides functions that help the other scripts call on each other. This script also calls on `pype_perfect` , which includes functions necessary to calibrate and solve perfectly competitive and monopoly partial equilibrium models.

In [2]:
```
import data_processing as dataproc
import helper as hlpr
import rr_outputs as outs
```

User provided variables

These are variables that guide how the models are solved. The report discusses the chosen values in more detail.

IQF model settings

Surge years were chosen to be 2012-2013 based on staff analysis of year-over-year changes in import values of IQF products.

In [3]:
```
surge_years_IQF = [2,3]
```

The maximum capacity constraint for the asymptotic domestic supply function is set at 110% of the domestic production for each year based on correspondence with an industry representative.

In [4]:
```
Qmax_multiplier_IQF = 1.1
```

The total elasticity of demand in the IQF market uses a standard value.

In [5]:
```
total_elasticity_IQF = -1.0
```

Non-IQF (Other) model settings

Surge years were chosen to be 2011-2014 based on staff analysis of year-over-year changes in import values of non-IQF products.

In [6]:
```
surge_years_Other = [1,2,3,4]
```

The price elasticity of the isoelastic domestic supply function and the total price elasticity of demand use standard values.

In [7]:
```
supply_elasticity_Other = 1.0
total_elasticity_Other = -1.0
```

Algorithm Settings

These variables determine how many Armington elasticities are used to produce the distribution of potential results and what percentiles are reported in tables and plots.

In [8]:
```
n_specifications = 103  # If this variable is 1, only produce a point estimat
e.
hi_percentile = 0.95    # Should be > 0.5 and < 1.0
lo_percentile = 0.05    # Should be > 0.0 and < 0.5
```

Output Location

Select a subfolder name for output tables and figures. If a subfolder with that name does not exist inside the current folder, the subfolder will be created.

In [9]:
```
folder_outputs = 'outputs'
```

Read and process domestic and imports data 2010-2020

Data is processed and cleaned using functions defined in the supporting scripts to put the data in a format that is consistent between domestic and imports.

Only years 2010-2020 are kept due to domestic data availability. For convenience, values are changed to millions of dollars.

In [10]:
```
# Open domestic and import data 2010-2020
df_domestic = dataproc.RRDomesticClean('12182020_raspberry_pricing_wrrc.xlsx')
df_import = dataproc.RRDataWebClean2('rr_imports.xlsx').reset_index()
df_import = df_import.astype({'Year':'int'})

# Divide to millions of dollars
df_agg = dataproc.RRAggregator3(df_import,df_domestic)
df_agg[['Value_import',
        'Value_domestic',
        'Total']] = df_agg[['Value_import', 'Value_domestic', 'Total']]/100000
0
```

Define lists to be used in calibration functions

These lines create lists of years and import values for IQF and non-IQF products.

In [11]:
```
year_list = list(df_agg.loc[df_agg['Total'].notnull(),'Year'].unique())

Qimports_IQF = list(df_agg.loc[(df_agg['Total'].notnull())
                               &(df_agg['IQF']==True),'Value_import'])
Qimports_Other = list(df_agg.loc[(df_agg['Total'].notnull())
                                 &(df_agg['IQF']==False),'Value_import'])
```

Estimate elasticities of substitution

These functions use trade data on value and trade costs to estimate the elasticity of substitution between different sources. This elasticity is assumed to be the same as the elasticity of substitution between imports and domestic goods.

In [12]:
```
tmp = df_import.copy().drop(['cv','ldpv'],
                            axis=1).rename({'cv_IQF':'cv',
                                            'ldpv_IQF':'ldpv'},
                                           axis=1)
arm_IQF_mean, arm_IQF_se = dataproc.ee6(tmp)
tmp = df_import.copy().drop(['cv','ldpv'],
                            axis=1).rename({'cv_Other':'cv',
                                            'ldpv_Other':'ldpv'},
                                           axis=1)
arm_Other_mean, arm_Other_se = dataproc.ee6(tmp)
```

The estimated elasticity parameters and the standard deviations of those parameters are used to generate a normally distributed set of possible elasticities. The set is used by applying the inverse cdf function of the estimated distribution to a range of numbers between 0 and 1 (exluding the endpoints, ignoring positive and negative infinity from the discretized distribution).

In [13]:
```
zeroone=np.linspace(0,1,n_specifications)[1:-1]
arm_IQF_dist = [1.0-stats.norm.ppf(val, loc=arm_IQF_mean, scale=arm_IQF_se)
                for val in zeroone]
arm_Other_dist = [1.0-stats.norm.ppf(val, loc=arm_Other_mean,
                                     scale=arm_Other_se) for val in zeroone]
```

Construct and solve model in DataFrame format

Model inputs and results are stored in a pandas DataFrame (similar to a spreadsheet). Each row represents a different version of the model, identified by product (IQF or non-IQF), year, and elasticity of substitution. Each column contains a variable relevant to the model.

The following code first constructs the DataFrame, starting with the combined domestic and imports data and adding columns for additional input variables. Next, the `apply()` function is called to run the model calibration and solver functions on each row. This is achieved by calling functions defined in the helper script that are designed to work with the dataframe row format. Those helper functions in turn call functions from the PyPE (Python Partial Equilibrium) script that actually calibrate or solve the model.

Set up the DataFrame

The DataFrame df will hold all model inputs and results.Each row represents a market (IQF or Other), year, and specification (elasticity of substitution).Each column represents an input or output variable.

Shorthand is used to keep column names for results short. Columns ending in '0' are based on the actual data while columns ending in '1' represent the simulation. Columns with last non-numeric character 'd' are domestic, with 'f' are imports.

The following code initializes the the DataFrame with the data and then adds columns for model inputs and the simulation imports.

In [14]:
```
df = df_agg.copy()

df = hlpr.include_supply_param(df,Qmax_multiplier_IQF,
                               supply_elasticity_Other)

df = hlpr.include_elasticities(df,arm_IQF_dist,arm_Other_dist,
                               total_elasticity_IQF,
                               total_elasticity_Other)

df = hlpr.simulation_imports(df,year_list,
                         Qimports_IQF,surge_years_IQF,
                         Qimports_Other,surge_years_Other)
```

Calibrate remaining parameters

This code calls the calibration helper function which simply calls PyPE calibration functions and applies them to each row of the DataFrame. The calibration assumes that each market is in equilibrium, normalizes prices to 1, and then backs out the implied supply parameters that would result in that equilibrium. The remaining lines add price and quantity columns for that equilibrium.

In [15]:
```
df[['sd0','sf0']] = df.apply(lambda row: hlpr.calibrate_supply_params(row),
                             result_type='expand',axis=1)
df[['pd0','pf0']] = df.apply(lambda row: hlpr.solve_prices(row, False),
                             result_type='expand',axis=1)
df[['qd0','qf0']] = df.apply(lambda row: hlpr.solve_prods(row, False),
                             result_type='expand',axis=1)
```

Solve simulations

This code calls the price and quantity solver functions again. Instead of using using the actual data, this section uses the simulation imports calculated earlier.

In [16]:
```
df[['pd1','pf1']] = df.apply(lambda row: hlpr.solve_prices(row, True),
                             result_type='expand',axis=1)
df[['qd1','qf1']] = df.apply(lambda row: hlpr.solve_prods(row, True),
                             result_type='expand',axis=1)
```

Calculate derived variables

With the base and simulation models solved, the remaining step is to calculate variables of interest that are derived from the equilibrium prices and quantities.

First, calculate expenditures.

In [17]:
```
df['xd0'] = df['pd0']*df['qd0']
df['xd1'] = df['pd1']*df['qd1']
df['xf0'] = df['pf0']*df['qf0']
df['xf1'] = df['pf1']*df['qf1']
df['x0'] = df['xd0']+df['xf0']
df['x1'] = df['xd1']+df['xf1']
```

Now, calculate simulation prices and quantities as a percentage over the actual values. Note that these columns end with '01' to show that they are comparing the actual ('0') to the simulation ('1').

In [18]:
```
df['pd01'] = 100*(df['pd1']-df['pd0'])/df['pd0']
df['qd01'] = 100*(df['qd1']-df['qd0'])/df['qd0']
```

Output tables and plots

The following code generates the tables and plots displayed in the report.

First, the code checks if the output subfolder exists. If not, the code creates it.

In [19]:
```
if not os.path.exists(folder_outputs):
    os.makedirs(folder_outputs)
```

Create and save tables with model results

This function saves model results to Excel workbooks. The function also outputs a DataFrame with only the median, 5th percentile, and 95 percentile results, which are used for some of the output figures.

In [20]:
```
df, df_short = outs.output_tables(df,lo_percentile,hi_percentile,
                                  folder_outputs)
```

Create figures

In [21]:
```
outs.generate_figures(df,df_short,year_list,
                      surge_years_IQF,Qimports_IQF,
                      surge_years_Other,Qimports_Other,
                      folder_outputs)
```

Show import visualizations

This snippet of code is not used in the modeling itself, but demonstrates how the data is processed and provides additional information about imports of raspberries for processing. The output folder is listed twice since these functions generate both figures and tables, which can be saved to two separate locations.

In [22]:
```
outs.vis_stackplots(df_domestic,df_import,folder_outputs,folder_outputs)
outs.vis_pieplots(df_import,folder_outputs,folder_outputs)
```