Tyche

Release 1.0

Aug 09, 2022

CONTENTS:

Risk and uncertainty are core characteristics of research and development (R&D) programs. Attempting to do what has not been done before will sometimes end in failure, just as it will sometimes lead to extraordinary success. The challenge is to identify an optimal mix of R&D investments in pathways that provide the highest returns while reducing the costs of failure. The goal of the R&D Pathway and Portfolio Analysis and Evaluation project is to develop systematic, scalable pathway and portfolio analysis and evaluation methodologies and tools that provide high value to the U.S. Department of Energy (DOE) and its Office of Energy Efficiency & Renewable Energy (EERE). This work aims to inform decision-making across R&D projects and programs by assisting analysts and decision makers to identify and evaluate, quantify and monitor, manage, document, and communicate energy technology R&D pathway and portfolio risks and benefits. The project-level risks typically considered are technology cost and performance (e.g., efficiency and environmental impact), while the portfolio level risks generally include market factors (e.g., competitiveness and consumer preference).

Quick Start Guide covers how to set up an R&D decision context using Tyche by creating the necessary input datasets and writing the technology model. *Technology Model Example* provides a simple example of developing a technology model, and *Analysis Example* provides an analysis example of decision support analysis. High-level information on the approach behind Tyche is given in *Approach* and details on the mathematical formulation used to represent technologies and analyze investment impact is given in *Mathematical Formulation*. *Optimization* gives information on built-in optimization algorithms. A guide to the Tyche user interface is given in *User Interface*, and a preliminary deployment plan in *Deployment Plan*. The complete Python API for the Tyche codebase, technology models provided with Tyche, and the user interface is in *Python API*.

CHAPTER

ONE

QUICK START GUIDE

The purpose of this quick start guide is to allow a new user to set up their first R&D decision context using Tyche, and to provide some examples of using Tyche for decision support analyses.

An R&D decision context involves one or more technologies that are subject to various R&D investments with the goal of changing the technology metrics and outcomes.

1.1 Set up Tyche package

To download and use the Tyche package on a personal computer:

- On Windows only, for users without a Python distribution: First download and install Anaconda.
- Download the source code.
- Navigate to the Tyche repository folder.
- Create the Tyche environment

On Windows:

```
conda env create --file conda\win.yml
conda activate tyche
```

On Mac:

```
conda env create --file conda/mac.yml
conda activate tyche
```

Note that the conda environment was created with the command:

```
conda create -n tyche -c conda-forge python=3.7 numpy scipy scikit-learn seaborn=0.10␣
↪matplotlib=3.3 quart hypercorn jupyter
```

- If you receive an HTTPS error, consider retrying the command with the *–insecure* flag added.
- See the conda documentation for additional information on installing and troubleshooting environments.

1.2 Directory Structure

The directory where users should store new technology models (.py files) and the accompanying datasets discussed below is indicated in blue in Fig. 1.1. We recommend that users create sub-directories for each

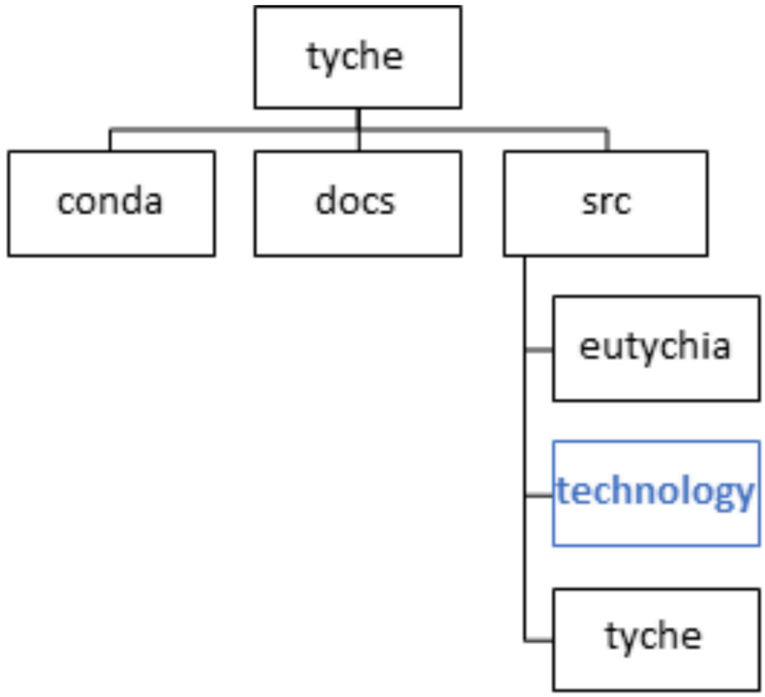


Fig. 1.1: Tyche repository directory structure. Users should not alter this structure. New technology models and data can be saved in sub-directories under the technology folder, indicated in blue.

new technology or decision context, to avoid confusing the various input datasets.

1.3 Defining Technologies

1.3.1 What is a "technology"?

In the R&D decision contexts represented and analyzed by Tyche, "technology" has a very broad definition. A technology converts input(s) to output(s) using capital with a defined lifetime and incurs fixed and/or variable costs in doing so. A technology may be a manufacturing process, a biorefinery, an agricultural process, a renewable energy technology component such as a silicon wafer, a renewable energy technology unit such as a wind turbine or solar panel, a renewable power plant system such as a concentrated solar power plant, and more. Within the R&D decision context, a technology is also subject to one or more research areas in which R&D investments can be made to change the technology and its economic, environmental, and other metrics of interest. Multiple technologies can be modeled and compared within the same decision context, provided the same metrics are calculable for each technology. Within Tyche, a technology is represented both physically and economically using a classic but simple and generalized techno-economic analysis (TEA). The TEA is based on a user defined technology model and accompanying datasets of technological and investment information.

1.4 Input Datasets

1.4.1 Designs Dataset

A *design* is one set of technology data that results from a specific R&D investment scenario. The *designs* dataset collects the technology versions that may result from all R&D investment scenarios being considered in a decision context.

The *designs* dataset contains information for one or more technologies being compared within an R&D investment decision context. There will be multiple sets of data for each technology; each set represents the technology data that results from a specific R&D investment scenario. Table 1.1 provides a data dictionary for the *designs* dataset. Additional information on mandatory Variables is provided in Table 1.2.

Table 1.1: Data dictionary for the *designs* dataset which defines various technology versions resulting from R&D investments.

Column Name	Data Type	Allowed Values	Description
Technology	String	Any	Name of the technology.
Scenario	String	Any names are allowed. There must be at least two scenarios defined.	R&D investment scenario that results in this technology design.
Variable	String	• Input • Input efficiency • Input price • Output efficiency • Output price • Lifetime • Scale	Variable types required by technology model and related functions.
Index	String	Any	Name of the elements within each Variable.
Value	• Float • Distribution • Mixture of distributions	• Set of real numbers • *scipy.stats* distributions • Mixture of *scipy.stats* distributions	Value for the R&D investment scenario. Example: st.triang(1,loc=5,scale=0.1)
Units	String	Any	User defined units for Variables. Not used by Tyche.
Notes	String	Any	Description provided by user. Not used by Tyche.

If there are no elements within a Variable for the technology under study, the Variable must still be included in the *designs* dataset: leaving out any of the Variables in this dataset will break the code. The Value for irrelevant Variables may be set to 0 or 1. Variables and their component Indexes are defined further in Table 1.2.

Table 1.2: Mandatory values for Variables in the *designs* dataset.

Variable	Description	Index Description
Input	Ideal input amounts that do not account for inefficiencies or losses.	Names of inputs to the technology.
Input efficiency	Input inefficiencies or losses, expressed as a number between 0 and 1.	Names of inputs to the technology: every input with an amount must also have an efficiency value, even if the efficiency is 1.
Input price	Purchase price for the input(s)	Names of inputs to the technology.
Output efficiency	Output efficiencies or losses, expressed as a number between 0 and 1.	Names of outputs from the technology. Every output must have an efficiency value, even if the efficiency is 1.
Output price	Sale price for the output(s).	Names of outputs from the technology. Every output must have a price, even if the price is irrelevant (in which case, set the price to 0).
Lifetime	Time that a piece of capital spends in use; time it takes for a piece of capital's value to depreciate to zero.	Names of the capital components of the technology.
Scale	Scale at which the technology operates (one value for the technology).	No index.

1.4.2 Parameters Dataset

The *parameters* dataset contains any ad hoc data, other than that contained in the *designs* dataset, that is required to calculate a technology's capital cost, fixed cost, production (actual output amount(s)), and metrics. If the information in the *designs* dataset completely defines the technology and its metrics of interest, then the *parameters* dataset can be left blank except for the column names. Identically to the *designs* dataset, the *parameters* dataset contains multiple sets of data corresponding to different R&D investment scenarios.

Table 1.3: Data dictionary for the *parameters* dataset, which contains ad-hoc technology parameters other than those in the *designs* dataset.

Column Name	Data type	Description
Technology	String	Name of the technology.
Scenario	String	Name of the R&D investment scenario that resulted in the corresponding parameter values or distributions.
Parameter	String	Name of the parameter.
Offset	String	Numerical location of the parameter.
Value	Float; Distribution; Mixture of distributions	Parameter value for the R&D investment scenario. Example: st.triang(1,loc=5,scale=0.1)
Units	String	Parameter units. User defined; not used or checked during Tyche calculations.
Notes	String	Any additional information defined by the user. Not used during Tyche calculations.

Including the Offset value in the *parameters* dataset creates a user reference that makes it easier to access parameter values when defining the technology model.

1.4.3 Technology Model

The technology model is a Python file (.py) which is user defined and contains methods for calculating capital cost, fixed cost, production (the actual output amount), and any metrics of interest, using the content of the *designs* and *parameters* datasets. Table 1.4 describes methods that must be included in the technology model Python file. The names of the methods are user-defined and must match the contents of the *functions* dataset, discussed below. Additional methods can be included in the technology model, if necessary, but the methods in Table 1.4 are required. All return values for the required methods must be formatted as Numpy stacks

Table 1.4: Methods required within the technology model Python file. Method names are user-defined and should match the contents of the functions dataset. Additional methods can be defined within the technology model as necessary.

Recommended Method Name	Parameters	Returns
capital_cost	scale, parameter	Capital cost(s) for each type of capital in the technology.
fixed_cost	scale, parameter	Annual fixed cost(s) of operating the technology.
production	scale, capital, lifetime, fixed, input, parameter	Calculated actual (not ideal) output amount(s).
metrics	scale, capital, lifetime, fixed, input_raw, input, input_price, output_raw, output, cost, parameter	Calculated technology metric value(s).

The production method can access the actual input amount, which is the ideal or raw input amount value multiplied by the input efficiency value (both defined in the *designs* dataset). In contrast, the metrics method can access both the ideal input amount (*input_raw*) and the actual input amount (*input*).

1.5 Defining R&D Investments

1.5.1 Tranches Dataset

A *tranche* is a discrete unit of R&D investment (dollar amount) in a specific research category. Tranches within the same research category are mutually exclusive: one cannot simultaneously invest $1M and $5M in a research category. A *scenario* is a combination of tranches that represents one option for making R&D investments.

The *tranches* dataset defines the allowed set of R&D investments across the research categories that are relevant to the technology under study. Tranches are combined into investment Scenarios – the same Scenarios found in the *designs* and *parameters* datasets. The impact of each Scenario on the technology is highly uncertain and is quantified using expert elicitation. A data dictionary for the *tranches* dataset is given in Table 1.5.

Table 1.5: Data dictionary for the *tranches* dataset.

Column Name	Data Type	Description
Category	String	Names of the R&D categories in which investment can be made to impact the technology or technologies being studied.
Tranche	String	Names of the tranches.
Scenario	String	Names of the R&D investment scenarios, which combine tranches across R&D categories. The names in this column must correspond to the Scenarios listed in the designs and parameters datasets.
Amount	Float; Distribution; Mixture of distributions	The R&D investment amount of the Tranche. The amount may be defined as a scalar, a probability distribution, or a mix of probability distributions.
Notes	String	Additional user-defined information. Not used by Tyche.

1.5.2 Investment Dataset

An *investment*, similar to a *scenario*, is a combination of tranches that represents a particular R&D strategy.

The *investments* dataset provides a separate way to look at making R&D investments. Combining individual tranches allows users to explore and optimize R&D investment amounts, but it may be the case that there are specific strategies that users wish to explore, without optimizing. In this case, the *investments* dataset is used to define specific combinations of tranches that are of interest. A data dictionary for the *investments* dataset is given in Table 1.6.

Table 1.6: Data dictionary for the *investments* dataset.

Column Name	Data Type	Description
Investment	String	Name of the R&D investment. Distinct from the Scenarios.
Category	String	Names of the R&D categories being invested in. Within each row, the Category must match the Tranche.
Tranche	String	Names of the tranches within the Investment. Within each row, the Tranche must match the Category
Notes	String Additional user-defined information. Not used by Tyche.	

1.6 Uncertainty in the Input Datasets

Tyche provides two general use cases for exploring the relationship between R&D investments and technological changes, both of which rely on expert elicitation to quantify inherent uncertainty. In the first and likely more common use case, a user knows what the R&D investment options are for a technology or set of technologies and is interested in determining what impact these investment options have on the technology(ies) in order to decide how to allocate an R&D budget. In other words, in this use case the user already knows the contents of the *tranches* and *investments* datasets, which are deterministic (fixed), and uses expert elicitation to fill in key values in the *designs* and *parameters* datasets with probability distributions.

In the second use case, a user knows what technological changes must be achieved with R&D investment and is interested in determining the investment amount that will be required to achieve these changes. In this case the user already knows the contents of the *designs* and *parameters* dataset, which are deterministic, and uses expert elicitation to fill in the investment amounts in the *tranches* dataset.

It is critical to note that these use cases are **mutually exclusive**. Tyche cannot be used to evaluate a scenario in which desired technological changes as well as the investment amounts are both uncertain. What this means for the user is that probability distributions, or mixtures of distributions, can be used to specify values either in the *designs* and *parameters* datasets or in the *tranches* dataset, but not both. If distributions are used in all three datasets, the code will break by design.

1.6.1 Defining values as probability distributions and mixtures

An uncertain value can be defined within a dataset using any of the built-in distributions of the scipy.stats package. A list of available distributions is provided at the hyperlink. Uncertain values can also be defined as a weighted average or mixture of probability distributions using the Tyche *mixture* method.

1.7 Additional Input Datasets

1.7.1 Indices Dataset

The *indices* dataset contains the numerical indexes (location within a list or array) used to access content in the other datasets. Table 1.7 describes the columns required for the indices table. Numerical locations for parameters should not be listed in this dataset.

Table 1.7: Data dictionary for the *indices* dataset.

Column Name	Data Type	Allowed Values	Description
Technology	String	Any	Name of the technology
Type	String	• Capital • Input • Output • Metric	Names of the Types defined within the designs dataset.
Index	String	Any	Name of the elements within each Type. For instance, names of the Input types.
Offset	Integer	geq 0	Numerical location of the Index within each Type.
Description	String	Any	Additional user-defined information, such as units. Not used during Tyche calculations.
Notes	String	Any	Additional user-defined information. Not used during Tyche calculations.

All four Types must be listed in the *indices* dataset. If a particular Type is not relevant to the technology under study, it still must be included in this dataset.

1.7.2 Relationship between *indices* and other datasets

A technology in the Tyche context is quantified using five sets of attribute values and one technology-level attribute value. The five sets of attribute values are Capital, Input, Output, Parameter, and Metric, and the technology-level attribute is Scale. Elements within each of the five sets are defined with an Index which simply names the element (for instance, Electricity might be one of the Index values within the Input set). Elements of Capital have an associated Lifetime. Elements of the Input set have an associated ideal amount (also called Input), an Input efficiency value, and an Input price. Elements of the Output set have only an Output efficiency and an Output price; the ideal output amounts are calculated from the technology model. Elements of the Metric set are named with an Index and are likewise calculated from the technology model. Elements of the Parameter set have only a value.

The *indices* dataset lists the elements of the Capital, Input, Output, and Metric sets, and contains an Offset column giving the numerical location of each element within its set. The *designs* dataset contains values for each element of the Capital, Input, Output, and Metric sets as well as the technology-level Scale value. The *parameters* dataset names and gives values for each element of the Parameter set.

1.7.3 Functions Dataset

The *functions* dataset is used internally by Tyche to locate the technology model file and identify the four required methods listed in Table 1.4. Table 1.8 provides a data dictionary for the *functions* dataset.

Table 1.8: Data dictionary for the *functions* dataset.

Column Name	Data Type	Allowed Values	Description
Technology	String	Any	Name of the technology.
Style	String	numpy	See below for explanation.
Module	String	Any	Filename of the technology model Python file, discussed below. Do not include the file extension.
Capital	String	Any	Name of the method within the technology model Python file that returns the calculated capital cost.
Fixed	String	Any	Name of the method within the technology model Python file that returns the calculated fixed cost.
Production	String	Any	Name of the method within the technology model Python file that returns the calculated output amount.
Metrics	String	Any	Name of the method within the technology model Python file that returns the calculated technology metrics.
Notes	String	Any	Any information that the user needs to record can go here. Not used during Tyche calculations.

The Style should remain *numpy* in Tyche 1.0. This indicates that inputs and outputs from the methods within the technology model Python file are treated as arrays rather than higher-dimensional (i.e., tensor) structures.

If only one technology model is used within a decision context, then the *functions* dataset will contain a single row.

1.7.4 Results Dataset

The *results* dataset lists the Tyche outcomes that are of interest within a decision context, organized into categories defined by the Variable column. This dataset is used internally by Tyche for organizing and labeling results tables for easier user comprehension. A data dictionary for the *results* dataset is given in Table 1.9.

Table 1.9: Data dictionary for the *results* dataset.

Column Name	Data Type	Allowed Values	Description
Technology	String	Any	Name of the technology.
Variable	String	• Cost • Output • Metric	Specific technology outcomes calculated by Tyche.
Index	String	Any	Names of the elements within each Variable.
Units	String	Any	User-defined units of the Index values. Not used or checked during Tyche calculations.
Notes	String	Any	Additional information defined by the user. Not used during Tyche calculations.

The Variable "Cost" is a technology-wide lifetime cost, and as such may not be relevant within all decision contexts. To fill in the Index values for the "Output" and "Metric" Variables, see the *designs* dataset.

CHAPTER

TWO

TECHNOLOGY MODEL EXAMPLE

Here is a very simple model for electrolysis of water. We just have water, electricity, a catalyst, and some lab space. We choose the fundamental unit of operation to be moles of H_2:

$H_2O \rightarrow H_2 + ½\ O_2$

For this example, we treat the catalyst as the capital that we use to transform inputs into outputs. Our inputs are water and electricity, and our outputs are oxygen and hydrogen. Our only fixed cost is the rent on the lab space at $1000/year. Using our past experience with electrolysis technology as well as some historical data, we estimate that we'll be able to produce 6650 mol/year of hydrogen and at this scale, our catalyst has a lifetime of about 3 years. The metrics we'd like to calculate for our electrolysis technology are cost, greenhouse gas (GHG) emissions, and jobs created. Based on this information, the *designs* dataset for the base case electrolysis technology is as shown in Table 2.1.

Table 2.1: *designs* dataset for the base case (without any R&D) of the simple electrolysis example technology.

Technology	Scenario	Variable	Index	Value	Units	Notes
Simple electrolysis	Base Electrolysis	Input	Water	19.04	g/mole	
Simple electrolysis	Base Electrolysis	Input efficiency	Water	0.95	1	Due to mass transport loss on input.
Simple electrolysis	Base Electrolysis	Input	Electricity	279	kJ/mole	
Simple electrolysis	Base Electrolysis	Input efficiency	Electricity	0.85	1	Due to ohmic losses on input.
Simple electrolysis	Base Electrolysis	Output efficiency	Oxygen	0.9	1	Due to mass transport loss on output.
Simple electrolysis	Base Electrolysis	Output efficiency	Hydrogen	0.9	1	Due to mass transport loss on output.
Simple electrolysis	Base Electrolysis	Lifetime	Catalyst	3	yr	Effective lifetime of Al-Ni catalyst.
Simple electrolysis	Base Electrolysis	Scale	n/a	6650	mole/yr	Rough estimate for a 50W setup.
Simple electrolysis	Base Electrolysis	Input price	Water	4.80E-03	USD/mole	
Simple electrolysis	Base Electrolysis	Input price	Electricity	3.33E-05	USD/kJ	
Simple electrolysis	Base Electrolysis	Output price	Oxygen	3.00E-03	USD/g	
Simple electrolysis	Base Electrolysis	Output price	Hydrogen	1.00E-02	USD/g	

Note that this is not the only way to model the electrolysis technology. We could choose to purchase lab space and equipment instead of renting, in which case we would have more types of capital, each with a particular lifetime. We could treat the oxygen output from our technology as waste instead of a coproduct and remove it from the model entirely. We could operate at a different scale and perhaps change our fixed or capital costs by doing so. Depending on where we operate this technology, our input and output prices will likely change. The Tyche framework offers great flexibility in representing technologies and technology systems; it is unlikely that there will only be a single correct way to model a decision context.

A key quantity that is not included in the *designs* dataset is our fixed cost, rent for the lab space. This quantity is included in the *parameters* dataset in Table 2.2, along with the necessary data to calculate our metrics of interest (cost, GHG, jobs).

Table 2.2: *parameters* dataset for the base case (without any R&D) of the simple electrolysis example technology.

Technology	Scenario	Parameter	Offset	Value	Units	Notes
Simple electrolysis	Base Electrolysis	Oxygen production	0	16	g	
Simple electrolysis	Base Electrolysis	Hydrogen production	1	2	g	
Simple electrolysis	Base Electrolysis	Water consumption	2	18.08	g	
Simple electrolysis	Base Electrolysis	Electricity consumption	3	237	kJ	
Simple electrolysis	Base Electrolysis	Jobs	4	1.50E-04	job/mole	
Simple electrolysis	Base Electrolysis	Reference scale	5	6650	mole/yr	
Simple electrolysis	Base Electrolysis	Reference capital cost for catalyst	6	0.63	USD	
Simple electrolysis	Base Electrolysis	Reference fixed cost for rent	7	1000	USD/yr	
Simple electrolysis	Base Electrolysis	GHG factor for water	8	0.00108	gCO2e/g	based on 244,956 gallons = 1 Mg CO2e
Simple electrolysis	Base Electrolysis	GHG factor for electricity	9	0.138	gCO2e/kJ	based on 1 kWh = 0.5 kg CO2e

Within our R&D decision context, we're interested in increasing the input and output efficiencies of this process so we can produce hydrogen as cheaply as possible. Experts could assess how much R&D to increase the various efficiencies η would cost. They could also suggest different catalysts, adding alkali, or replacing the process with PEM.

The `indices` table (see Table 2.3) simply describes the various indices available for the variables. The `Offset` column specifies the memory location in the argument for the production and metric functions.

Table 2.3: Example of the `indices` table.

Technology	Type	Index	Offset	Description	Notes
Simple electrolysis	Capital	Catalyst	0	Catalyst	
Simple electrolysis	Fixed	Rent	0	Rent	
Simple electrolysis	Input	Water	0	Water	
Simple electrolysis	Input	Electricity	1	Electricity	
Simple electrolysis	Output	Oxygen	0	Oxygen	
Simple electrolysis	Output	Hydrogen	1	Hydrogen	
Simple electrolysis	Metric	Cost	0	Cost	
Simple electrolysis	Metric	Jobs	1	Jobs	
Simple electrolysis	Metric	GHG	2	GHGs	

2.1 Production function (à la Leontief)

$$P_{\text{oxygen}} = (16.00\ \text{g}) \cdot \min\left\{\frac{I^*_{\text{water}}}{18.08\ \text{g}}, \frac{I^*_{\text{electricity}}}{237\ \text{kJ}}\right\}$$

$$P_{\text{hydrogen}} = (2.00\ \text{g}) \cdot \min\left\{\frac{I^*_{\text{water}}}{18.08\ \text{g}}, \frac{I^*_{\text{electricity}}}{237\ \text{kJ}}\right\}$$

2.2 Metric functions

$$M_{\text{cost}} = K/O_{\text{hydrogen}}$$

$$M_{\text{GHG}} = \left((0.00108\ \text{gCO2e/gH20})\, I_{\text{water}} + (0.138\ \text{gCO2e/kJ})\, I_{\text{electricity}}\right)/O_{\text{hydrogen}}$$

$$M_{\text{jobs}} = (0.00015\ \text{job/mole})\,/O_{\text{hydrogen}}$$

2.3 Performance of current design.

$K = 0.18$ USD/mole (i.e., not profitable since it is positive)

$O_{\text{oxygen}} = 14$ g/mole

$O_{\text{hydrogen}} = 1.8$ g/mole

$\mu_{\text{cost}} = 0.102$ USD/gH2

$\mu_{\text{GHG}} = 21.4$ gCO2e/gH2

$\mu_{\text{jobs}} = 0.000083$ job/gH2

2.4 Technology Model

Each technology design requires a Python file with a capital cost, a fixed cost, a production, and a metrics function. Listing 2.1 shows these functions for the simple electrolysis example.

Listing 2.1: Example technology-defining functions.

```
# Simple electrolysis.


# All of the computations must be vectorized, so use `numpy`.
import numpy as np


# Capital-cost function.
def capital_cost(
  scale,
  parameter
):

  # Scale the reference values.
  return np.stack([np.multiply(
    parameter[6], np.divide(scale, parameter[5])
  )])


# Fixed-cost function.
def fixed_cost(
  scale,
  parameter
):

  # Scale the reference values.
  return np.stack([np.multiply(
    parameter[7],
    np.divide(scale, parameter[5])
  )])


# Production function.
def production(
  capital,
  fixed,
  input,
  parameter
):

  # Moles of input.
  water       = np.divide(input[0], parameter[2])
  electricity = np.divide(input[1], parameter[3])

  # Moles of output.
  output = np.minimum(water, electricity)

  # Grams of output.
  oxygen   = np.multiply(output, parameter[0])
  hydrogen = np.multiply(output, parameter[1])
```

(continues on next page)

(continued from previous page)

```
    # Package results.
    return np.stack([oxygen, hydrogen])


# Metrics function.
def metrics(
    capital,
    fixed,
    input_raw,
    input,
    img/output_raw,
    output,
    cost,
    parameter
):

    # Hydrogen output.
    hydrogen = output[1]

    # Cost of hydrogen.
    cost1 = np.divide(cost, hydrogen)

    # Jobs normalized to hydrogen.
    jobs = np.divide(parameter[4], hydrogen)

    # GHGs associated with water and electricity.
    water       = np.multiply(input_raw[0], parameter[8])
    electricity = np.multiply(input_raw[1], parameter[9])
    co2e = np.divide(np.add(water, electricity), hydrogen)

    # Package results.
    return np.stack([cost1, jobs, co2e])
```

CHAPTER

THREE

ANALYSIS EXAMPLE

Multiple Objectives for Residential PV.

3.1 Import packages.

```
import os
import sys
sys.path.insert(0, os.path.abspath("../src"))
```

```
import numpy             as np
import matplotlib.pyplot as pl
import pandas            as pd
import seaborn           as sb
import tyche             as ty

from copy            import deepcopy
from IPython.display import Image
```

3.2 Load data.

The data should be stored in a set of comma-separated value files in a sub-directory of the technology folder, as shown in the directory structure diagram (Fig. 1.1)

```
designs = ty.Designs("data/pv_residential_simple")
```

```
investments = ty.Investments("data/pv_residential_simple")
```

Compile the production and metric functions for each technology in the dataset.

```
designs.compile()
```

3.3 Examine the data.

The `functions` table specifies where the Python code for each technology resides.

```
designs.functions
```

Right now, only the style `numpy` is supported.

The `indices` table defines the subscripts for variables.

```
designs.indices
```

The `designs` table contains the cost, input, efficiency, and price data for a scenario.

```
designs.designs
```

The `parameters` table contains additional techno-economic parameters for each technology.

```
designs.parameters
```

The `results` table specifies the units of measure for results of computations.

```
designs.results
```

The `tranches` table specifies multually exclusive possibilities for investments: only one `Tranch` may be selected for each `Category`.

```
investments.tranches
```

The `investments` table bundles a consistent set of tranches (one per category) into an overall investment.

```
investments.investments
```

3.4 Evaluate the scenarios in the dataset.

```
scenario_results = designs.evaluate_scenarios(sample_count=50)
```

```
scenario_results.xs(1, level="Sample", drop_level=False)
```

3.4.1 Save results.

```
scenario_results.to_csv("output/pv_residential_simple/example-scenario.csv")
```

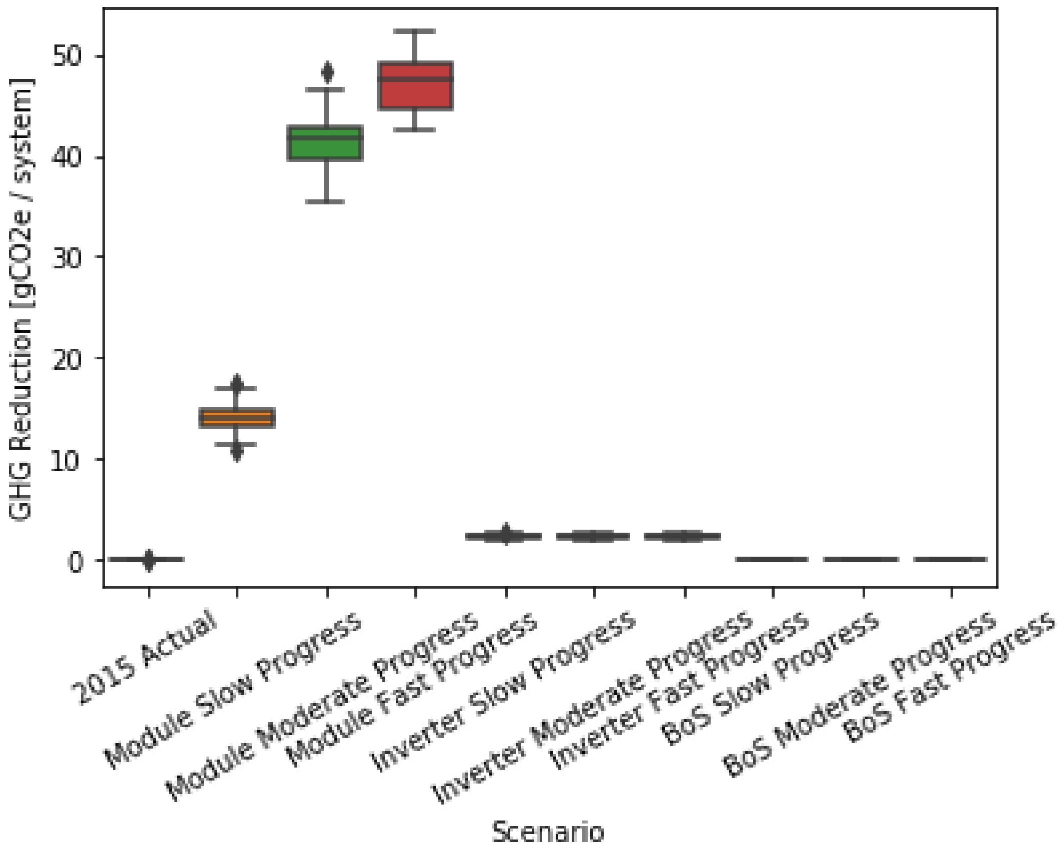
3.4.2 Plot GHG metric.

```
g = sb.boxplot(
    x="Scenario",
    y="Value",
    data=scenario_results.xs(
        ["Metric", "GHG"],
        level=["Variable", "Index"]
    ).reset_index()[["Scenario", "Value"]],
```

(continues on next page)

(continued from previous page)

```
    order=[
        "2015 Actual"              ,
        "Module Slow Progress"      ,
        "Module Moderate Progress"  ,
        "Module Fast Progress"      ,
        "Inverter Slow Progress"    ,
        "Inverter Moderate Progress",
        "Inverter Fast Progress"    ,
        "BoS Slow Progress"         ,
        "BoS Moderate Progress"     ,
        "BoS Fast Progress"         ,
    ]
)
g.set(ylabel="GHG Reduction [gCO2e / system]")
g.set_xticklabels(g.get_xticklabels(), rotation=30);
```



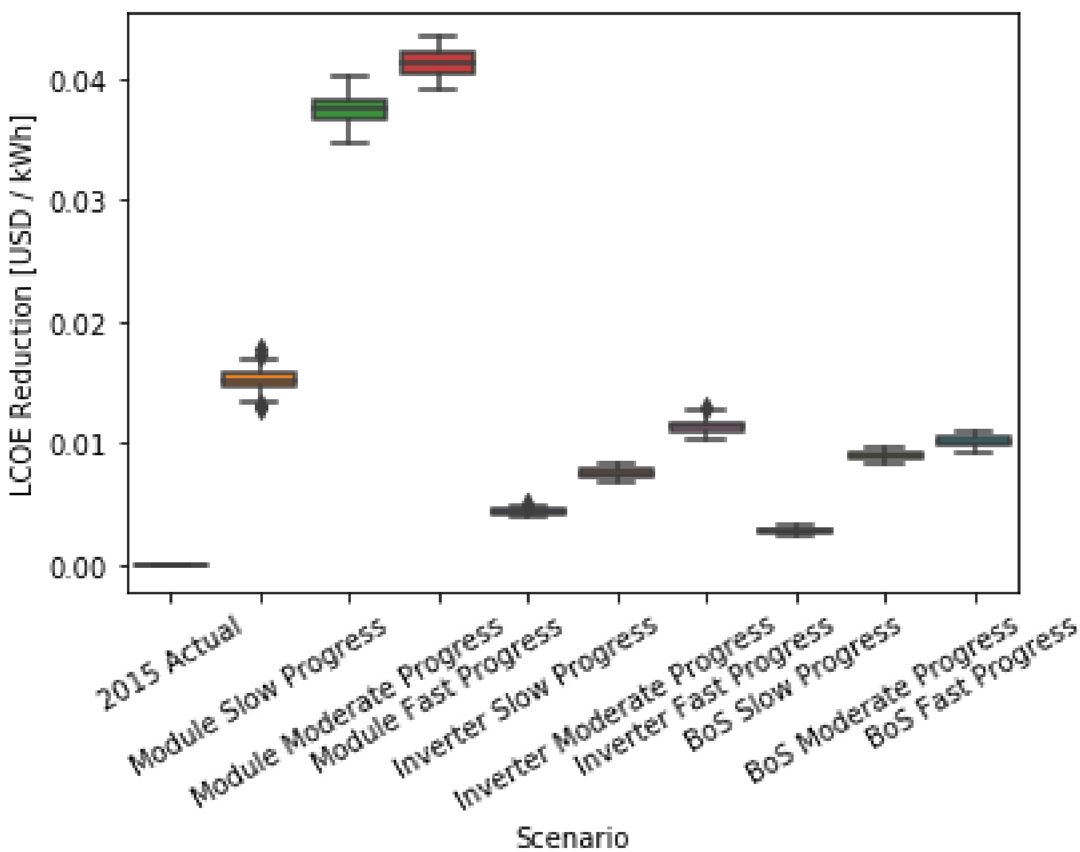
3.4.3 Plot LCOE metric.

```
g = sb.boxplot(
    x="Scenario",
    y="Value",
    data=scenario_results.xs(
```

(continues on next page)

(continued from previous page)

```
        ["Metric", "LCOE"],
        level=["Variable", "Index"]
    ).reset_index()[["Scenario", "Value"]],
    order=[
        "2015 Actual"                 ,
        "Module Slow Progress"         ,
        "Module Moderate Progress"     ,
        "Module Fast Progress"         ,
        "Inverter Slow Progress"       ,
        "Inverter Moderate Progress",
        "Inverter Fast Progress"       ,
        "BoS Slow Progress"            ,
        "BoS Moderate Progress"        ,
        "BoS Fast Progress"            ,
    ]
)
g.set(ylabel="LCOE Reduction [USD / kWh]")
g.set_xticklabels(g.get_xticklabels(), rotation=30);
```



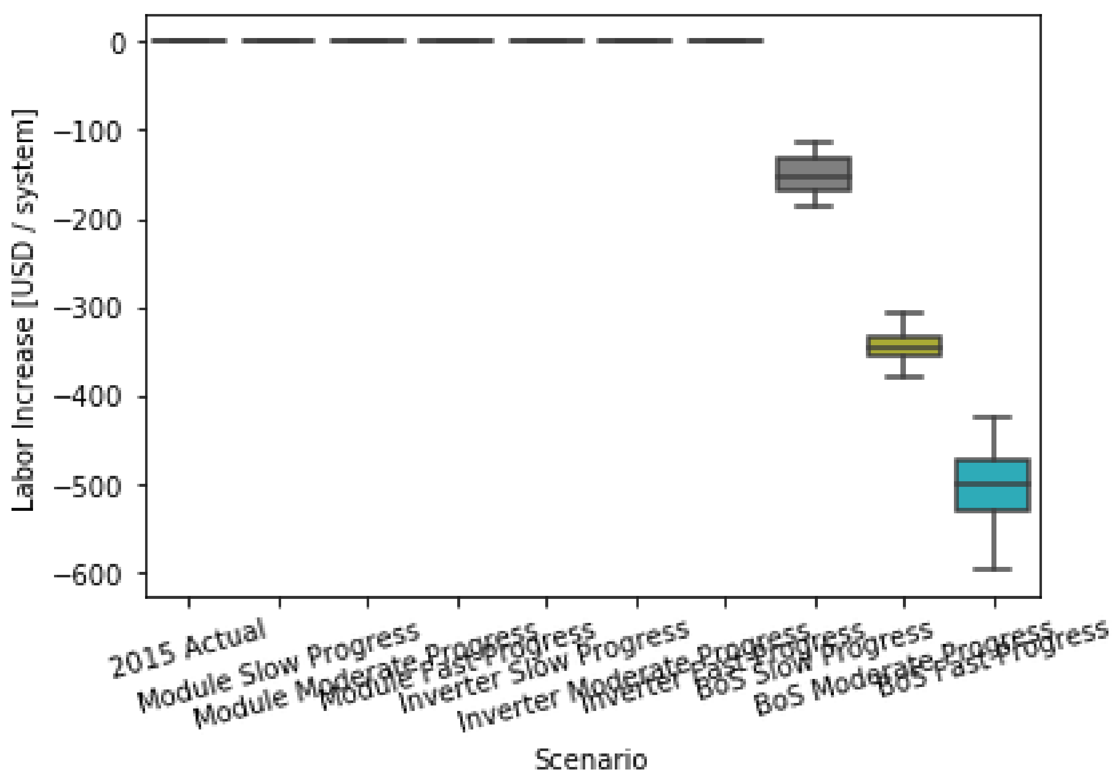
3.4.4 Plot labor metric.

```
g = sb.boxplot(
```

(continues on next page)

(continued from previous page)

```
    x="Scenario",
    y="Value",
    data=scenario_results.xs(
        ["Metric", "Labor"],
        level=["Variable", "Index"]
    ).reset_index()[["Scenario", "Value"]],
    order=[
        "2015 Actual"                ,
        "Module Slow Progress"        ,
        "Module Moderate Progress"    ,
        "Module Fast Progress"        ,
        "Inverter Slow Progress"      ,
        "Inverter Moderate Progress",
        "Inverter Fast Progress"      ,
        "BoS Slow Progress"           ,
        "BoS Moderate Progress"       ,
        "BoS Fast Progress"           ,
    ]
)
g.set(ylabel="Labor Increase [USD / system]")
g.set_xticklabels(g.get_xticklabels(), rotation=15);
```



3.5 Evaluate the investments in the dataset.

```
investment_results = investments.evaluate_investments(designs, sample_count=50)
```

3.5.1 Costs of investments.

```
investment_results.amounts
```

3.5.2 Benefits of investments.

```
investment_results.metrics.xs(1, level="Sample", drop_level=False)
```

```
investment_results.summary.xs(1, level="Sample", drop_level=False)
```

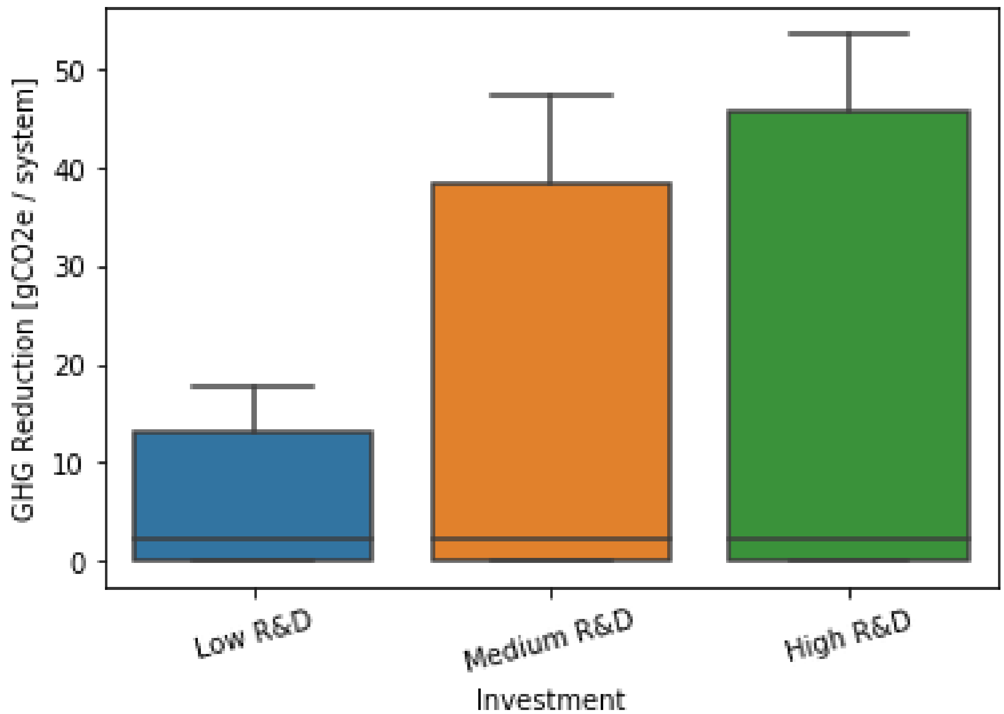
3.5.3 Save results.

```
investment_results.amounts.to_csv("output/pv_residential_simple/example-investment-
↪amounts.csv")
```

```
investment_results.metrics.to_csv("output/pv_residential_simple/example-investment-
↪metrics.csv")
```

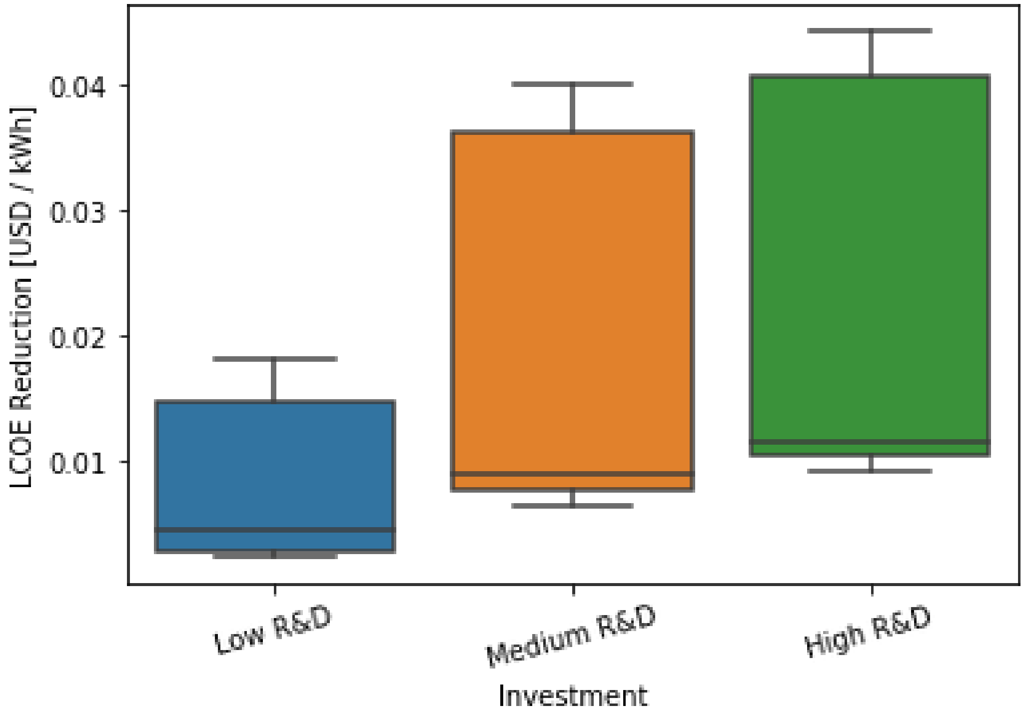
3.5.4 Plot GHG metric.

```
g = sb.boxplot(
    x="Investment",
    y="Value",
    data=investment_results.metrics.xs(
        "GHG",
        level="Index"
    ).reset_index()[["Investment", "Value"]],
    order=[
        "Low R&D"   ,
        "Medium R&D",
        "High R&D"  ,
    ]
)
g.set(ylabel="GHG Reduction [gCO2e / system]")
g.set_xticklabels(g.get_xticklabels(), rotation=15);
```



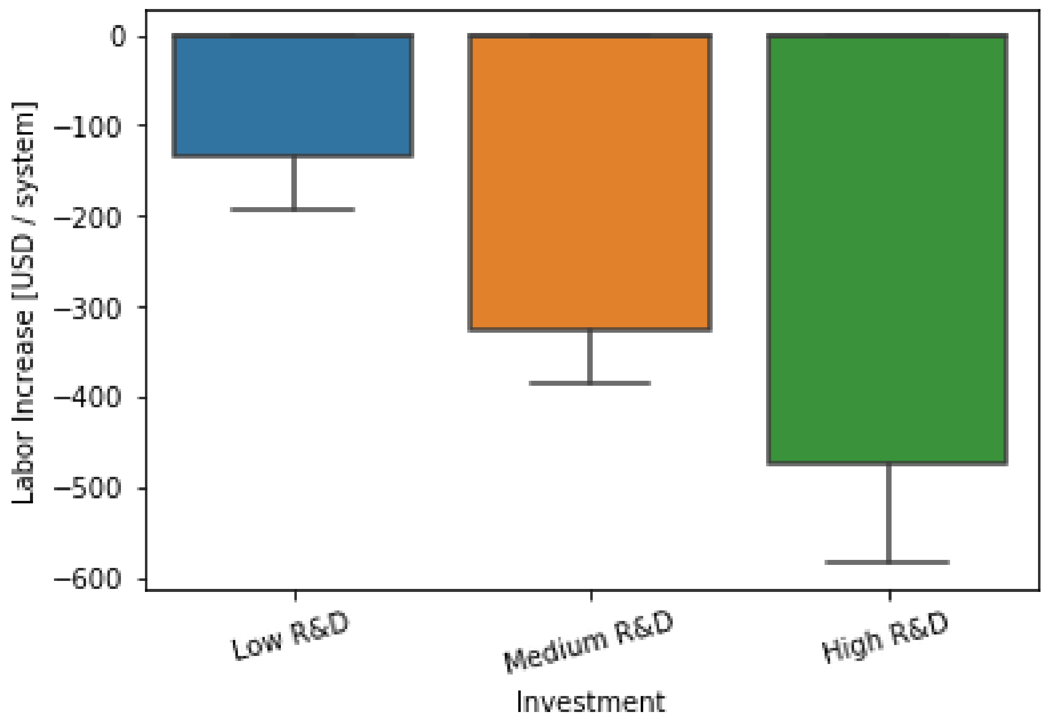
3.5.5 Plot LCOE metric.

```
g = sb.boxplot(
    x="Investment",
    y="Value",
    data=investment_results.metrics.xs(
        "LCOE",
        level="Index"
    ).reset_index()[["Investment", "Value"]],
    order=[
        "Low R&D"    ,
        "Medium R&D",
        "High R&D"   ,
    ]
)
g.set(ylabel="LCOE Reduction [USD / kWh]")
g.set_xticklabels(g.get_xticklabels(), rotation=15);
```



3.5.6 Plot labor metric.

```
g = sb.boxplot(
    x="Investment",
    y="Value",
    data=investment_results.metrics.xs(
        "Labor",
        level="Index"
    ).reset_index()[["Investment", "Value"]],
    order=[
        "Low R&D"    ,
        "Medium R&D",
        "High R&D"   ,
    ]
)
g.set(ylabel="Labor Increase [USD / system]")
g.set_xticklabels(g.get_xticklabels(), rotation=15);
```



3.6 Multi-objective decision analysis.

3.6.1 Compute costs and metrics for tranches.

Tranches are atomic units for building investment portfolios. Evaluate all of the tranches, so we can assemble them into investments (portfolios).

```
tranche_results = investments.evaluate_tranches(designs, sample_count=50)
```

Display the cost of each tranche.

```
tranche_results.amounts
```

Display the metrics for each tranche.

```
tranche_results.summary
```

Save the results.

```
tranche_results.amounts.to_csv("output/pv_residential_simple/example-tranche-amounts.csv
↪")
tranche_results.summary.to_csv("output/pv_residential_simple/example-tranche-summary.csv
↪")
```

3.6.2 Fit a response surface to the results.

The response surface interpolates between the discrete set of cases provided in the expert elicitation. This allows us to study funding levels intermediate between those scenarios.

```
evaluator = ty.Evaluator(investments.tranches, tranche_results.summary)
```

Here are the categories of investment and the maximum amount that could be invested in each:

```
evaluator.max_amount
```

Here are the metrics and their units of measure:

```
evaluator.units
```

Example interpolation.

Let's evaluate the case where each category is invested in at half of its maximum amount.

```
example_investments = evaluator.max_amount / 2
example_investments
```

```
evaluator.evaluate(example_investments)
```

```
Category    Index  Sample
BoS R&D     GHG    1          -0.0010586097518157094
                   2           7.493162517135921e-05
                   3            0.001253893601450784
                   4            -0.00398626797827717
                   5           -0.005572343870333896
                                            ...
Module R&D  Labor  46           0.014371009324918305
                   47           0.011128728287076228
                   48          0.0039832773605894545
                   49           0.006026680267950724
                   50           0.028844695933457842
Name: Value, Length: 450, dtype: object
```

Let's evaluate the mean instead of outputing the whole distribution.

```
evaluator.evaluate_statistic(example_investments, np.mean)
```

```
Index
GHG       30.156830
LCOE       0.038160
Labor   -246.843027
Name: Value, dtype: float64
```

Here is the standard deviation:

```
evaluator.evaluate_statistic(example_investments, np.std)
```

```
Index
GHG       1.410956
LCOE      0.000850
Labor    16.070395
Name: Value, dtype: float64
```

A risk-averse decision maker might be interested in the 10% percentile:

```
evaluator.evaluate_statistic(example_investments, lambda x: np.quantile(x, 0.1))
```

```
Index
GHG        28.573627
LCOE        0.037140
Labor    -268.059699
Name: Value, dtype: float64
```

3.6.3 ϵ-Constraint multiobjective optimization

```
optimizer = ty.EpsilonConstraintOptimizer(evaluator)
```

In order to meaningfully map the decision space, we need to know the maximum values for each of the metrics.

```
metric_max = optimizer.max_metrics()
metric_max
```

```
GHG      49.429976
LCOE      0.062818
Labor     0.049555
Name: Value, dtype: float64
```

Example optimization.

Limit spending to $3M.

```
investment_max = 3e6
```

Require that the GHG reduction be at least 40 gCO2e/system and that the Labor wages not decrease.

```
metric_min = pd.Series([40, 0], name = "Value", index = ["GHG", "Labor"])
metric_min
```

```
GHG      40
Labor     0
Name: Value, dtype: int64
```

Compute the ϵ-constrained maximum for the LCOE.

```
optimum = optimizer.maximize(
    "LCOE"                        ,
    total_amount = investment_max,
    min_metric   = metric_min    ,
    statistic    = np.mean       ,
)
optimum.exit_message
```

```
'Optimization terminated successfully.'
```

Here are the optimal spending levels:

```
np.round(optimum.amounts)
```

```
Category
BoS R&D                 0.0
Inverter R&D            0.0
Module R&D        3000000.0
Name: Amount, dtype: float64
```

Here are the three metrics at that optimum:

```
optimum.metrics
```

```
Index
GHG      41.627691
LCOE      0.037566
Labor     0.028691
Name: Value, dtype: float64
```

Thus, by putting all of the investment into Module R&D, we can expected to achieve a mean 3.75 ¢/kWh reduction in LCOE under the GHG and Labor constraints.

It turns out that there is no solution for these constraints if we evaluate the 10th percentile of the metrics, for a risk-averse decision maker.

```
optimum = optimizer.maximize(
    "LCOE"                        ,
    total_amount = investment_max,
    min_metric   = metric_min    ,
    statistic    = lambda x: np.quantile(x, 0.1),
)
optimum.exit_message
```

```
'Iteration limit exceeded'
```

Let's try again, but with a less stringent set of constraints, only constraining GHG somewhat but not Labor at all.

```
optimum = optimizer.maximize(
    "LCOE"                                                          ,
    total_amount = investment_max                                   ,
    min_metric   = pd.Series([30], name = "Value", index = ["GHG"]),
```

(continues on next page)

(continued from previous page)

```
    statistic    = lambda x: np.quantile(x, 0.1)                    ,
)
optimum.exit_message
```

```
'Optimization terminated successfully.'
```

```
np.round(optimum.amounts)
```

```
Category
BoS R&D             0.0
Inverter R&D        0.0
Module R&D    3000000.0
Name: Amount, dtype: float64
```

```
optimum.metrics
```

```
Index
GHG      39.046988
LCOE      0.036463
Labor    -0.019725
Name: Value, dtype: float64
```

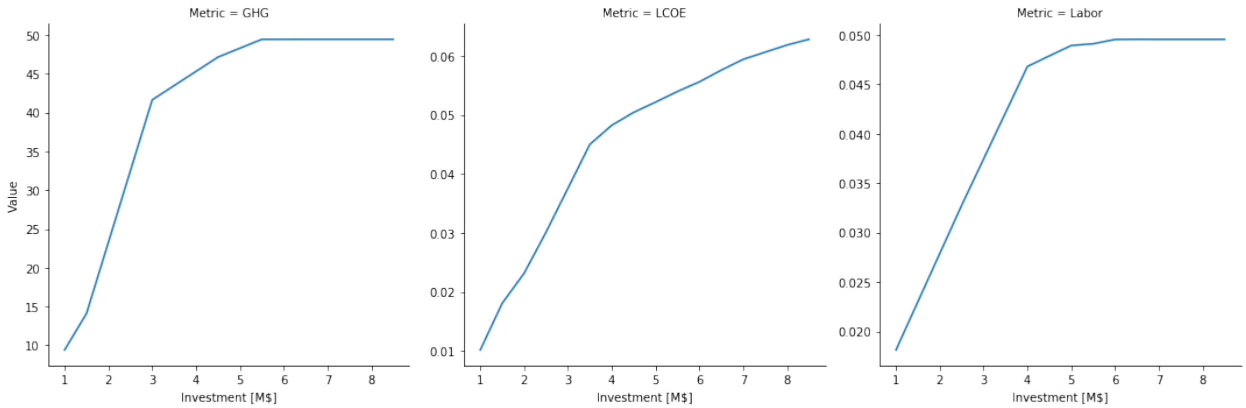
3.6.4 Pareto surfaces.

Metrics constrained by total investment.

```
pareto_amounts = None
for investment_max in np.arange(1e6, 9e6, 0.5e6):
    metrics = optimizer.max_metrics(total_amount = investment_max)
    pareto_amounts = pd.DataFrame(
        [metrics.values]                                        ,
        columns = metrics.index.values                          ,
        index   = pd.Index([investment_max / 1e6], name = "Investment [M$]"),
    ).append(pareto_amounts)
pareto_amounts
```

```
sb.relplot(
    x         = "Investment [M$]",
    y         = "Value"          ,
    col       = "Metric"         ,
    kind      = "line"           ,
    facet_kws = {'sharey': False},
    data      = pareto_amounts.reset_index().melt(id_vars = "Investment [M$]", var_name␣
↪= "Metric", value_name = "Value")
)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f9da11752b0>
```



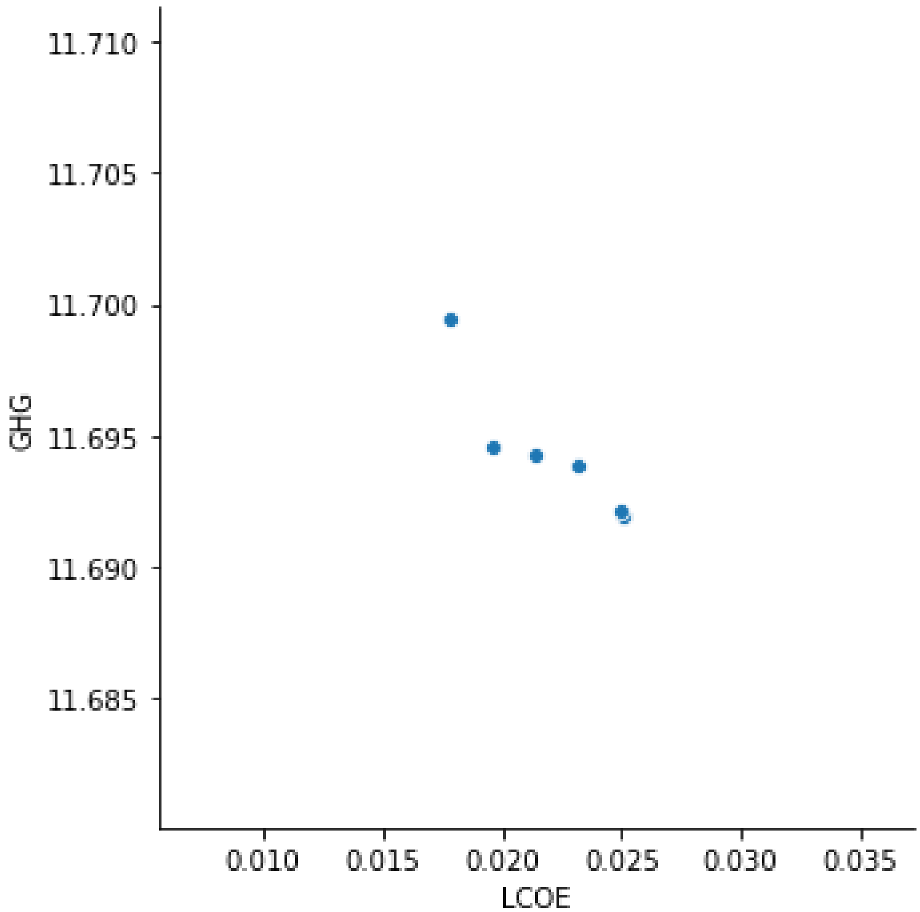
We see that the LCOE metric saturates more slowly than the GHG and Labor ones.

GHG vs LCOE, constrained by total investment.

```
investment_max = 3
pareto_ghg_lcoe = None
for lcoe_min in 0.95 * np.arange(0.5, 0.9, 0.05) * pareto_amounts.loc[investment_max, "LCOE"]:
    optimum = optimizer.maximize(
        "GHG",
        max_amount   = pd.Series([0.9e6, 3.0e6, 1.0e6], name = "Amount", index = ["BoS R&D", "Inverter R&D", "Module R&D"]),
        total_amount = investment_max * 1e6                                ,
        min_metric   = pd.Series([lcoe_min], name = "Value", index = ["LCOE"]),
    )
    pareto_ghg_lcoe = pd.DataFrame(
        [[investment_max, lcoe_min, optimum.metrics["LCOE"], optimum.metrics["GHG"], optimum.exit_message]],
        columns = ["Investment [M$]", "LCOE (min)", "LCOE", "GHG", "Result"]                  ,
    ).append(pareto_ghg_lcoe)
pareto_ghg_lcoe = pareto_ghg_lcoe.set_index(["Investment [M$]", "LCOE (min)"])
pareto_ghg_lcoe
```

```
sb.relplot(
    x = "LCOE",
    y = "GHG",
    kind = "scatter",
    data = pareto_ghg_lcoe#[pareto_ghg_lcoe.Result == "Optimization terminated successfully."]
)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f9da13ae630>
```



The three types of investment are too decoupled to make an interesting pareto frontier, and we also need a better solver if we want to push to lower right.

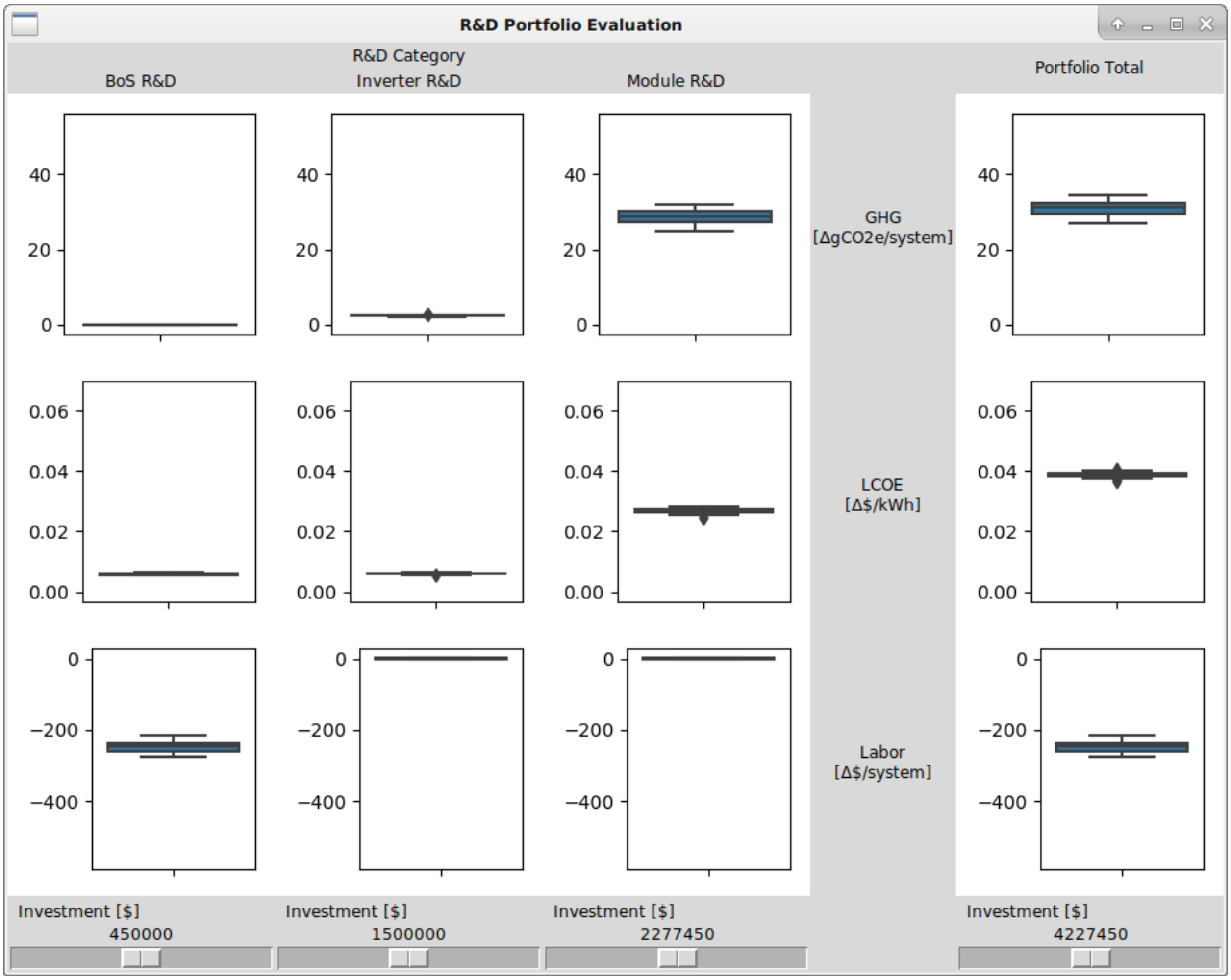
3.7 Run the interactive explorer for the decision space.

Make sure the the `tk` package is installed on your machine. Here is the Anaconda link: https://anaconda.org/anaconda/tk.

```
w = ty.DecisionWindow(evaluator)
w.mainloop()
```

A new window should open that looks like the image below. Moving the sliders will cause a recomputation of the boxplots.

```
Image("pv_residential_simple_gui.png")
```


R&D Portfolio Evaluation
R&D Category
BoS R&D
Inverter R&D
Module R&D
Portfolio Total
GHG
[ΔgCO2e/system]
LCOE
[Δ$/kWh]
Labor
[Δ$/system]
Investment [$]
450000
Investment [$]
1500000
Investment [$]
2277450
Investment [$]
4227450

CHAPTER FOUR

APPROACH

Our production-function approach to R&D portfolio evaluation is mathematically formulated as a stochastic multi-objective decision-optimization problem and is implemented in the Python programming language. The framework abstracts the technology-independent aspects of the problem into a generic computational schema and enables the modeler to specify the technology-dependent aspects in a set of data tables and Python functions. This approach not only minimizes the labor needed to add new technologies, but it also enforces uniformity of financial, mass-balance, and other assumptions in the analysis.

The framework is scalable, supporting rapid computation on laptops computer and large-ensemble studies on high-performance computers (HPC). The use of vectorized operations for the stochastic calculations and of response-surface fits for the portfolio evaluations minimizes the computational resources needed for complex multi-objective optimizations. The software handles parameterized studies such as tornado plots, Monte-Carlo sensitivity analyses, and a generalization of epsilon-constraint optimization.

All values in the data tables may be probability distributions, specified by Python expressions using a large library of standard distributions, or the values may be simple numbers. Expert opinion is encoded through these distributions. The opinions may be combined prior to simulator or subsequent to it.

Four example technologies have been implemented as examples illustrating framework's use: biorefineries, electrolysis, residential photovoltaics (PV), and utility-scale PV. A desktop user interface allows exploration of the cost-benefit trade-offs in portfolio decision problems.

4.1 Mock FOA Definition

4.1.1 Background

Understanding the FOA process is essential to designing an effective tool to make to make technically and analytically-based decisions. The "Mock FOA" process takes a service design approach to understanding the FOA-writing process as it stands. The Mock FOA began with interviews with five previous DOE detailees and seven senior DOE staff who have led multiple FOA development efforts. A major theme emerged: effective communication of analysis logic and results poses one of the largest challenges during the FOA process.

A decision-support tool could assist in the communication necessary to percolate this technical information up the chain. Interview findings were formalized in collaboration with the NREL service design team to understand where such a tool could make the greatest impact. The team considered the steps taken to issue a FOA, resources referenced, and decision-makers involved.

Phases

Interviews revealed that, while all FOA processes are unique and highly non-linear, specific actions must occur. These characterize phases of the FOA journey:

1. **Launch**. Decide to issue a FOA.
2. **Frame**. Formulate a plan to collect the information necessary to write the FOA
3. **Scope**. Investigate topic options.
4. **Draft**. Compile information into draft FOA.
5. **Refine**. Prepare FOA for distribution.

The specific needs of each phase inform the tool **content**.

Roles

The team considered that different staff members will interact with this information differently and prefer different methods of data communication. These roles were characterized by "personae" defined by level of involvement in the FOA-writing process.

- Technical analyst
- Technical lead
- FOA lead
- Approver

Decision makers in each of these roles will interact with tool output. The tool users determine how the tool will be used and how its content will be displayed, informing **interactions and data visualization**. For example, a user who will view the tool output in a presentation will need a static representation of the tool output.

4.1.2 Potential topics

Prototyping a tool requires content. The team referenced two previous FOAs to understand the breakdown of topic areas. We then extracted FOA topic/subtopic areas and metrics from 2016 budget request, combining hard/soft cost-focused FOAs to examine how to compete the two and avoid directly analyzing a specific past FOA. Following this process further informed the team's understanding of how decision-makers might decide what to input into the tool.

Topics under consideration might be assessed by the following metrics:

- $/$W_{DC}$
- $/kWh
- Strategic metal content (lifetime)
- Hazardous waste content (lifetime)
- Lifetime
- Reliability
- Emissions
- Labor

The following text details topic areas considered for Tyche tool development.

1. Crystaline silicon wafer design

- Wafer area
- Wafer thickness
- Wafer density
- Silicon utilization
- Production yield

2. Tandem thin-films

- Design parameters
- Architectures

3. Polysilicon module

- (many parameters)

4. Module design

- Module Capital
- Module Lifetime
- Module Efficiency
- Module Aperture
- Module O&M Fixed
- Module Degradation
- Module Soiling Loss

5. Inverter design

- Inverter Capital
- Inverter Lifetime
- Inverter Replacement
- Inverter Efficiency

6. Balance-of-system design

- Hardware Capital
- Direct Labor
- Permitting
- Customer Acquisition
- Installer Overhead & Profit

CHAPTER FIVE

MATHEMATICAL FORMULATION

We separate the financial and conversion-efficiency aspects of a production process, which are generic across all technologies, from the physical and technical aspects, which are necessarily specific to the particular process. The motivation for this is that the financial and waste computations can be done uniformly for any technology (even for disparate ones such as PV cells and biofuels) and that different experts may be required to assess the cost, waste, and techno-physical aspects of technological progress. Table 5.1 defines the indices that are used for the variables that are defined in Table 5.2.

Table 5.1: Definitions for set indices used for variable subscripts.

Set	Description	Examples
$c \in \mathcal{C}$	capital	equipment
$f \in \mathcal{F}$	fixed cost	rent, insurance
$i \in \mathcal{I}$	input	feedstock, labor
$o \in \mathcal{O}$	output	product, co-product, waste
$m \in \mathcal{M}$	metric	cost, jobs, carbon footprint, efficiency, lifetime
$p \in \mathcal{P}$	technical parameter	temperature, pressure
$\nu \in N$	technology type	electrolysis, PV cell
$\theta \in \Theta$	scenario	the result of a particular investment
$\chi \in X$	investment category	investment alternatives
$\phi \in \Phi_\chi$	investment	a particular investment
$\omega \in \Omega$	portfolio	a basket of investments

Table 5.2: Definitions for variables.

Variable	Type	Description	Units
K	calculated	unit cost	USD/unit
C_c	function	capital cost	USD
τ_c	cost	lifetime of capital	year
S	cost	scale of operation	unit/year
F_f	function	fixed cost	USD/year
I_i	input	input quantity	input/unit
I^*_i	calculated	ideal input quantity	input/unit
η_i	waste	input efficiency	input/input
p_i	cost	input price	USD/input
O_o	calculated	output quantity	output/unit
O^*_o	calculated	ideal output quantity	output/unit
η'_o	waste	output efficiency	output/output
p'_o	cost	output price (+/-)	USD/output
μ_m	calculated	metric	metric/unit

Continued on next page

Table 5.2 – continued from previous page

Variable	Type	Description	Units
P_o	function	production function	output/unit
M_m	function	metric function	metric/unit
α_p	parameter	technical parameter	(mixed)
ξ_θ	variable	scenario inputs	(mixed)
ζ_θ	variable	scenario outputs	(mixed)
ψ	function	scenario evaluation	(mixed)
σ_ϕ	function	scenario probability	1
q_ϕ	variable	investment cost	USD
ζ_ϕ	random variable	investment outcome	(mixed)
$\mathbf{Z}(\omega)$	random variable	portfolio outcome	(mixed)
$Q(\omega)$	calculated	portfolio cost	USD
Q^{min}	parameter	minimum portfolio cost	USD
Q^{max}	parameter	maximum portfolio cost	USD
q_ϕ^{min}	parameter	minimum category cost	USD
q_ϕ^{max}	parameter	maximum category cost	USD
Z^{min}	parameter	minimum output/metric	(mixed)
Z^{max}	parameter	maximum output/metric	(mixed)
$\mathbb{F}$, $\mathbb{G}$	operator	evaluate probabilities	(mixed)

5.1 Cost

The cost characterizations (capital and fixed costs) are represented as functions of the scale of operations and of the technical parameters in the design:

- Capital cost: $C_c(S, \alpha_p)$.
- Fixed cost: $F_f(S, \alpha_p)$.

The per-unit cost is computed using a simple levelization formula:

$K = \left(\sum_c C_c / \tau_c + \sum_f F_f \right) / S + \sum_i p_i \cdot I_i - \sum_o p'_o \cdot O_o$

5.2 Waste

The waste relative to the idealized production process is captured by the η parameters. Expert elicitation might estimate how the ηs would change in response to R&D investment.

- Waste of input: $I^*_i = \eta_i I_i$.
- Waste of output: $O_o = \eta'_o O^*_o$.

5.3 Production

The production function idealizes production by ignoring waste, but accounting for physical and technical processes (e.g., stoichiometry). This requires a technical model or a tabulation/fit of the results of technical modeling.

$O^*_o = P_o(S, C_c, \tau_c, F_f, I^*_i, \alpha_p)$

5.4 Metrics

Metrics such as efficiency, lifetime, or carbon footprint are also compute based on the physical and technical characteristics of the process. This requires a technical model or a tabulation/fit of the results of technical modeling. We use the convention that higher values are worse and lower values are better.

$\mu_m = M_m(S, C_c, \tau_c, F_f, I_i, I_i^*, O_o^*, O_o, K, \alpha_p)$

5.5 Scenarios

A *scenario* represents a state of affairs for a technology ν. If we denote the scenario as θ, we have the tuple of input variables

$\xi_\theta = (S, C_c, \tau_c, F_f, I_i, \eta_i, \eta_o', \alpha_p, p_i, p_o')|_\theta$

and the tuple of output variables

$\zeta_\theta = (K, I_i^*, O_o^*, O_o, \mu_m)|_\theta$

and their relationship

$\zeta_\theta = \psi_\nu\left(\xi_\theta\right)|_{\nu=\nu(\theta)}$

given the tuple of functions

$\psi_\nu = (P_o, M_m)|_\nu$

for the technology of the scenario.

5.6 Investments

An *investment* ϕ assigns a probability distribution to scenarios:

$\sigma_\phi(\theta) = P\left(\theta|\phi\right)$.

such that

$\int d\theta \sigma_\phi(\theta) = 1$ or $\sum_\theta \sigma_\phi(\theta) = 1$,

depending upon whether one is performing the computations discretely or continuously. Expectations and other measures on probability distributions can be computed from the $\sigma_\phi(\theta)$. We treat the outcome ζ_ϕ as a random variable for the outcomes ζ_θ according to the distribution $\sigma_\phi(\theta)$.

Because investment options may be mutually exclusive, as is the case for investing in the same R&D at different funding levels, we say Φ_χ is the set of mutually exclusive investments (i.e., only one can occur simultaneously) in investment category χ: investments in different categories χ can be combined arbitrarily, but just one investment from each Φ_χ may be chosen.

Thus the universe of all portfolios is $\Omega = \prod_\chi \Phi_\chi$, so a particular portfolio $\omega \in \Omega$ has components $\phi = \omega_\chi \in \Phi_\chi$. The overall outcome of a portfolio is a random variable:

$\mathbf{Z}(\omega) = \sum_\chi \zeta_\phi \mid_{\phi=\omega_\chi}$

The cost of an investment in one of the constituents ϕ is q_ϕ, so the cost of a porfolio is:

$Q(\omega) = \sum_\chi q_\phi \mid_{\phi=\omega_\chi}$

5.7 Decision problem

The multi-objective decision problem is

$\min_{\omega \in \Omega} \; \mathbb{F} \, \mathbf{Z}(\omega)$

such that

$Q^{\min} \leq Q(\omega) \leq Q^{\max}$,

$q_\phi^{\min} \leq q_{\phi=\omega_\chi} \leq q_\phi^{\max}$,

$Z^{\min} \leq \mathbb{G} \, \mathbf{Z}(\omega) \leq Z^{\max}$,

where $\mathbb{F}$ and $\mathbb{G}$ are the expectation operator $\mathbb{E}$, the value-at-risk, or another operator on probability spaces. Recall that $\mathbf{Z}$ is a vector with components for cost K and each metric μ_m, so this is a multi-objective problem.

The two-stage decision problem is a special case of the general problem outlined here: Each scenario θ can be considers as a composite of one or more stages.

5.8 Experts

Each expert elicitation takes the form of an assessment of the probability and range (e.g., 10th to 90th percentile) of change in the cost or waste parameters or the production or metric functions. In essence, the expert elicitation defines $\sigma_\phi(\theta)$ for each potential scenario θ of each investment ϕ.

CHAPTER

SIX

OPTIMIZATION

6.1 Nonlinear Programming Formulation

Three methods in the `EpsilonConstraintOptimizer` class, `opt_slsqp`, `opt_shgo` and `opt_diffev`, are wrappers for optimization algorithm calls. (The *tyche.EpsilonConstraints* section provides full parameter and return value information for these methods.) The optimization methods define the optimization problem according to each algorithm's requirements, call the algorithm, and provide either optimized results in a standard format for postprocessing, or an error messages if the optimization did not complete successfully. The SLSQP algorithm, which is not a global optimizer, is provided to assess problem feasibility and provide reasonable upper and lower bounds on metrics being optimized. Because the technology models within an R&D decision context may be arbitrarily complex, two global optimization algorithms were also implemented. The global algorithms were chosen according to the following criteria.

- Ability to perform constrained optimization with inequality constraints, for instance on metric values or investment amounts.
- Ability to optimize without specified Jacobian or Hessian functions (derivative-less optimization).
- Ability to specify bounds on individual decision variables, which allows constraints on single research areas.
- Ability to work on a variety of potentially non-convex and otherwise complex problems.

6.1.1 Algorithm Testing

As a point of comparison between algorithms, the *Simple Residential Photovoltaics* decision context was optimized for minimum levelized cost of electricity (LCOE) subject to an investment constraint and a metric constraint (GHG). The solutions are given in Table 6.1. The solve times listed are in addition to the time required to set up the problem and solve for the optimum metric values; this procedure currently uses the SLSQP algorithm by default. This setup time is between 10 and 15 seconds.

Table 6.1: Minimizing LCOE subject to a total investment amount of $3 MM USD and GHG being at least 40.

Algorithm	Objective Function Value	GHG Constraint Value	Solve Time (s)
Differential evolution	0.037567	41.699885	145
Differential evolution	0.037547	41.632867	589
SLSQP	0.037712	41.969348	~ 2
SHGO	None found	None found	None found

Additional details for each solution are given below under the section for the corresponding algorithm.

6.2 Sequential Least Squares Programming (SLSQP)

The Sequential Least Squares Programming algorithm uses a gradient search method to locate a possibly local optimum. [6] A complete list of parameters and options for the `fmin_slsqp` algorithm is available in the scipy.optimize.fmin_slsqp documentation.

Constraints for `fmin_slsqp` are defined either as a single function that takes as input a vector of decision variable values and returns an array containing the value of all constraints in the problem simultaneously. Both equality and inequality constraints can be defined, although they must be as separate functions and are provided to the `fmin_slsqp` algorithm under separate arguments.

6.2.1 SLSQP Solution to Simple Residential Photovoltaics

Solve time: 1.5 s

Table 6.2: Optimal decision variables found by the SLSQP algorithm.

Decision Variable	Optimized Value
BoS R&D	1.25 E-04
Inverter R&D	3.64 E-08
Module R&D	3.00 E+06

Table 6.3: Optimal system metrics found by the SLSQP algorithm.

System Metric	Optimized Value
GHG	41.97
LCOE	0.038
Labor	0.032

6.3 Differential Evolution

Differential evolution is one type of evolutionary algorithm that iteratively improves on an initial population, or set of potential solutions. [5] Differential evolution is well-suited to searching large solution spaces with multiple local minima, but does not guarantee convergence to the global minimum. Moreover, users may need to adjust the default solving parameters and options in order to obtain a solution and cut down on solve time. A complete list of parameters and options for the *differential_evolution* algorithm is available in the scipy.optimize.differential_evolution documentation.

Constraints for *differential_evolution* are defined by passing the same multi-valued function defined in *opt_slsqp* to the NonlinearConstraint method.

6.3.1 Differential Evolution Solutions to Simple Residential Photovoltaics

Differential evolution stochastically populates the initial set of potential solutions, and so the optimal solution and solve time may vary with the random seed used.

Solution 1

Starting with a random seed of 2, the solution time was 145 seconds.

Table 6.4: Optimal decision variables found by the differential evolution algorithm with a seed of 2.

Decision Variable	Optimized Value
BoS R&D	9.62 E+02
Inverter R&D	5.33 E+02
Module R&D	2.99 E+06

Table 6.5: Optimal system metrics found by the differential evolution algorithm with a seed of 2.

System Metric	Optimized Value
GHG	41.70
LCOE	0.038
Labor	-0.456

Solution 2

Starting with a random seed of 1, the solution time was 589 seconds.

Table 6.6: Optimal decision variables found by *differential_evolution* as called by *EpsilonConstraints.opt_diffev* with a seed of 1.

Decision Variable	Optimized Value
BoS R&D	4.70 E+03
Inverter R&D	3.71 E+02
Module R&D	2.99 E+06

Table 6.7: Optimal system metrics found by *differential_evolution* as called by *EpsilonConstraints.opt_diffev* with a seed of 1.

System Metric	Optimized Value
GHG	41.63
LCOE	0.037
Labor	-2.29

6.4 Simplicial Homology Global Optimization

The Simplicial Homology Global Optimization (SHGO) algorithm applies simplicial homology to general non-linear, low-dimensional optimization problems. [4] SHGO provides fast solutions using default parameters and options, but the optimum found may not be as precise as that found by the differential evolution algorithm. Constraints for *shgo* must be provided as a dictionary or sequence of dictionaries with the following format:

```
constraints = [ {'type': 'ineq', 'fun': g1(x)},
                {'type': 'ineq', 'fun': g2(x)},
                ...
                {'type': 'eq', 'fun': h1(x)},
                {'type': 'eq', 'fun': h2(x)},
                ... ]
```

Each of the constraint functions *g1(x)*, *h1(x)*, and so on are functions that take decision variable values as inputs and return the value of the constraint. Inequality constraints (*g1(x)* and *g2(x)* above) are formulated as $g(x) \geq 0$ and equality constraints (*h1(x)* and *h2(x)* above) are formulated as $h(x) = 0$. Each constraint in the optimization problem is defined as a separate function, with a separate dictionary giving the constraint type. With *shgo* it is not possible to use one function that returns a vector of constraint values.

6.5 Piecewise Linear (MILP) Formulation

6.5.1 Notation

Table 6.8: Index definitions for the MILP formulation.

Index	Description
I	Number of elicited data points (investment levels and metrics)
J	Number of investment categories
K	Number of metrics

Table 6.9: Data definitions for the MILP formulation.

Data	Notation	Information
Investment amounts	$c_{ij}, i \in \{1, ..., I\}$	c_i is a point in J-dimensional space
Metric value	$q_{ik}, i \in \{1, ..., I\}, k \in \{1, ..., K\}$	One metric will form the objective function, leaving up to $K-1$ metrics for constraints

Table 6.10: Variable definitions for the MILP formulation.

Variable	Notation	Information
Binary variables	$y_{ii'}, i, i' \in \{1, ..., I\}, i' > i$	Number of linear intervals between elicited data points.
Combination variables	$\lambda_i, i \in \{1, ..., I\}$	Used to construct linear combinations of elicited data points. $\lambda_i \geq 0 \forall i$

Each metric and investment amount can be written as a linear combination of elicited data points and the newly introduced variables λ_i and $y_{ii'}$. Additional constraints on $y_{ii'}$ and λ_i take care of the piecewise linearity by ensuring that the corners used to calculate q_k reflect the interval that c_i is in. There will be a total of $\binom{I}{2}$ binary y variables, which reduces to $\frac{I(I-1)}{2}$ binary variables.

6.5.2 One-Investment-Category, One-Metric Example

Suppose we have an elicited data set for one metric ($K = 1$) and one investment category ($J = 1$) with three possible investment levels ($I = 3$). We can write the total investment amount as a linear combination of the three investment levels $c_{i1}, i \in \{1, 2, 3\}$, using the λ variables:

$\lambda_1 c_{11} + \lambda_2 c_{21} + \lambda_{13} c_{31} = \sum_i \lambda_i c_{i1}$

We can likewise write the metric as a linear combination of q_{1i} and the λ variables:

$\lambda_1 q_{11} + \lambda_2 q_{21} + \lambda_3 q_{31} = \sum_i \lambda_i q_{i1}$

We have the additional constraint on the λ variables that

$\sum_i \lambda_i = 1$

These equations, combined with the integer variables $y_{ii'} = \{y_{12}, y_{13}, y_{23}\}$, can be used to construct a mixed-integer linear optimization problem.

The MILP that uses this formulation to minimize a technology metric subject to a investment budget B is as follows:

$\min_{y,\lambda} \lambda_1 q_{11} + \lambda_2 q_{21} + \lambda_3 q_{31}$

subject to

$\lambda_1 c_{11} + \lambda_2 c_{21} + \lambda_3 c_{31} \leq B$, (1) Total budget constraint

$\lambda_1 + \lambda_2 + \lambda_3 = 1$, (2)

$y_{12} + y_{23} + y_{13} = 1$, (3)

$y_{12} \leq \lambda_1 + \lambda_2$, (4)

$y_{23} \leq \lambda_2 + \lambda_3$, (5)

$y_{13} \leq \lambda_1 + \lambda_3$, (6)

$0 \leq \lambda_1, \lambda_2, \lambda_3 \leq 1$, (7)

$y_{12}, y_{23}, y_{13} \in \{0, 1\}$, (8)

(We've effectively removed the investments and the metrics as variables, replacing them with the elicited data points and the new λ and y variables.)

6.5.3 Extension to N x N Problem

Note: k' indicates the metric which is being constrained. $k*$ indicates the metric being optimized. J' indicates the set of investment categories which have a budget limit (there may be more than one budget-constrained category in a problem).

No metric constraint or investment category-specific budget constraint

$\min_{y,\lambda} \sum_i \lambda_i q_{ik*}$

subject to

$\sum_i \sum_j \lambda_i c_{ij} \leq B$, (1) Total budget constraint

$\sum_i \lambda_i = 1$, (2)

$\sum_{i,i'} y_{ii'} = 1$, (3)

$y_{ii'} \leq \lambda_i + \lambda_{i'} \forall i, i'$, (4)

$0 \leq \lambda_i \leq 1 \forall i$, (5)

$y_{ii'} \in \{0, 1\} \forall i, i'$, (6)

With investment category-specific budget constraint

$\min_{y,\lambda} \sum_i \lambda_i q_{ik*}$

subject to

$\sum_i \sum_j \lambda_i c_{ij} \leq B$, (1) Total budget constraint

$\sum_i \lambda_i c_{ij'} \leq B_{j'} \forall j' \in J'$, (2) Investment category budget constraint(s)

$\sum_i \lambda_i = 1$, (3)

$\sum_{i,i'} y_{ii'} = 1$, (4)

$y_{ii'} \leq \lambda_i + \lambda_{i'} \forall i, i'$, (5)

$0 \leq \lambda_i \leq 1 \forall i$, (6)

$y_{ii'} \in \{0, 1\} \forall i, i'$, (7)

With metric constraint and investment category-specific budget constraint

$\min_{y,\lambda} \sum_i \lambda_i q_{ik*}$

subject to

$\sum_i \sum_j \lambda_i c_{ij} \leq B$, (1) Total budget constraint

$\sum_i \lambda_i c_{ij'} \leq B_{j'} \forall j' \in J'$ (2) Investment category budget constraint(s)

$\sum_i \lambda_i q_{ik'} \leq M_{k'}$, (3) Metric constraint

$\sum_i \lambda_i = 1$, (4)

$\sum_{i,i'} y_{ii'} = 1$, (5)

$y_{ii'} \leq \lambda_i + \lambda_{i'} \forall i, i'$, (6)

$0 \leq \lambda_i \leq 1 \forall i$, (7)

$y_{ii'} \in \{0, 1\} \forall i, i'$, (8)

Problem Size

In general, I is the number of rows in the dataset of elicited data. In the case that all investment categories have elicited data at the same number of levels (not necessarily the same levels themselves), I can also be calculated as l^J where l is the number of investment levels.

The problem will involve $\frac{I(I-1)}{2}$ binary variables and I continuous (λ) variables.

6.6 References

1. `scipy.optimize.shgo` SciPy v1.5.4 Reference Guide: Optimization and root finding (`scipy.optimize`) URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.shgo.html#rb2e152d227b3-1 Last accessed 12/28/2020.
2. `scipy.optimize.differential_evolution` SciPy v1.5.4 Reference Guide: Optimization and root finding (`scipy.optimize`) URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.differential_evolution.html Last accessed 12/28/2020.
3. `scipy.optimize.fmin_slsqp` SciPy v1.5.4 Reference Guide: Optimization and root finding (`scipy.optimize`) URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.fmin_slsqp.html Last accessed 12/28/2020.
4. Endres, SC, Sandrock, C, Focke, WW. (2018) "A simplicial homology algorithm for Lipschitz optimisation", Journal of Global Optimization (72): 181-217. URL: https://link.springer.com/article/10.1007/s10898-018-0645-y
5. Storn, R and Price, K. (1997) "Differential Evolution - a Simple and Efficient Heuristic for Global Optimization over Continuous Spaces", Journal of Global Optimization (11): 341 - 359. URL: https://link.springer.com/article/10.1023/A:1008202821328
6. Kraft D (1988) A software package for sequential quadratic programming. Tech. Rep. DFVLR-FB 88-28, DLR German Aerospace Center — Institute for Flight Mechanics, Koln, Germany.
7. `scipy.optimize.NonlinearConstraint` SciPy v1.5.4 Reference Guide: Optimization and root finding (`scipy.optimize`) URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.NonlinearConstraint.html Last accessed 12/29/2020.

CHAPTER SEVEN

USER INTERFACE

The Eutychia interface is a user's portal to interact with the Tyche decision support tool. Users can make decisions to change investments and the metrics by which they will be assessed (as described in the following sections). Eutychia aims to aide in formalizing funding processes to make technically and analytically-based decisions, through modeling possible scenarios and generating visualizations to communicate these results. Tool output aims to aide decision-makers in

1. **Focused analysis** comparing investment scenarios to examine impact across metrics when exploring options during the decision-making process and
2. **Broader communication** of Office goals externally, such as through the dissemination of a funding opportunity announcement.

Feedback is appreciated to enhance the interface to best meet user needs.

7.1 User Input

Investment Categories A user can suggest research foci by selecting **investment categories** and **investment levels** ($) in each topic area and/or across the investment portfolio. In the current iteration of the Eutychia prototype, users have the option to select a budget for each of the following investment categories:

1. Balance of System R&D
2. Inverter R&D
3. Module R&D

Later-stage iterations of the prototype will include as many categories as the user selects for which data is available.

Metrics A user can also select up to three metrics to impact through R&D on these selected investment categories and specify goals that must be met. The current options include:

1. Greenhouse gas emissions (ΔgCO2e/system)
2. Labor (Δ\$/system)
3. Levelized cost of energy (Δ\$/kWh)

7.2 Modes

The Eutychia interface operates in two modes:

1. **Explore Mode**, checked by default,

2. **Optimize Mode**, which can be enabled by deselecting "explore." Entering Optimize Mode allows users to update optimization parameters.

The selected mode determines which user inputs can be edited. The following table summarizes the parameters that can be updated, the corresponding `optimizer` parameter name, and the widget (currently) used to make this change.

	Parameter	Widget	Explore Mode	Optimize Mode
Investment level (USD) by category	`max_amount`	slider	X	X
Total portfolio investment (USD)	`total_amount`	slider		X
Metric constraint	`min_metric`	slider		X
Optimization metric to maximize	`metric`	dropdown		X

In either mode, changes made to investment level(s) by category will be reflected immediately in the output visualizations. In Optimize Mode, once satisfied with the selected metrics, the user can click "optimize" to model the chosen scenario.

7.3 Visualizations

Users are presented with the option to interact with the data in varying levels of detail. These options are enabled to suit the needs of users, from those who prefer a snapshot of the bigger picture for quick analysis to those who would like to study the distributional probability of achieving each metric. Plots are generated using the Seaborn 0.11.0 package.[1] The available visualizations in order of increasing level of detail include:

1. **Heatmaps** (`heatmap`) with metric scaled to percent of the maximum possible improvement,
2. **Annotated heatmaps** with metric values overlayed, and
3. **Distributions** with the probability of ahieving each metric based on the number of samples. At this stage of development, these results can be viewed in columns (by metric) or in a grid. The user can select from the following options:
   - Box plots (`boxplot`)
   - Probability distributions (`kdeplot`)
   - Violin plots (`violinplot`)

A user can toggle between their visualization options using the links (heatmap, column, grid) at the top left-hand corner of the screen. By default, Eutychia opens to the grid layout.

7.4 Running the Server

Visit the folder `src/eutychia/` and start the server in debug mode

```
cd src\eutychia
debug.cmd
```

on Windows, or on Mac

```
cd src/eutychia
./debug.sh
```

[1] Michael Waskom, Olga Botvinnik, Maoz Gelbart, Joel Ostblom, Paul Hobson, Saulius Lukauskas, David C Gemperline, et al. 2020. Mwaskom/Seaborn: V0.11.0 (Sepetmber 2020). Zenodo. https://doi.org/10.5281/zenodo.4019146.

or in production mode

```
cd src\eutychia
run.cmd
```

on Windows, or on Mac

```
cd src/eutychia
./run.sh
```

and then visit http://127.0.0.1:5000/.

7.5 References

CHAPTER

EIGHT

DEPLOYMENT PLAN

8.1 Objectives

1. Securely house all potentially sensitive data within on DOE servers within the DOE intranet.
2. Minimize the deployment and maintenance burden at DOE.
3. Assure the quality of software and data updates.
4. Enable DOE personnel and contractors to contribute technology models and data.

8.2 Components and Activities

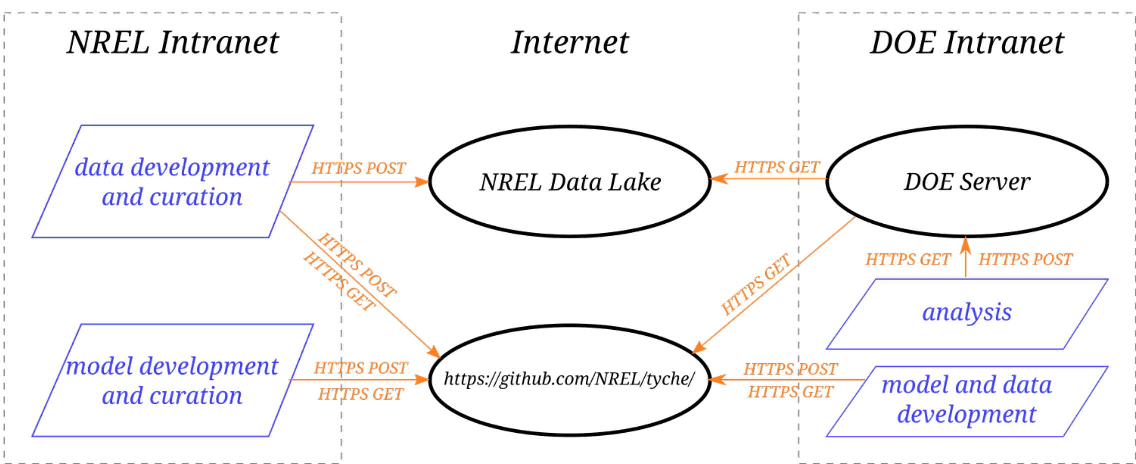


Fig. 8.1: Deployment of services and activities.

8.3 Activities

Analysts at DOE will connect to Tyche server within the DOE intranet using their web browsers to run and analyze scenarios using Tyche. The server will have the capability to record scenarios for sharing within DOE, but that data will never leave the DOE intranet.

Analysts developing data and technology models at DOE, NREL, and elsewhere can post that data and software to a branch of the GitHub Software Repository. Those contributions will be reviewed, vetted, and tested before they are pushed to the NREL Data Lake (in the case of datasets) or to the `production` branch of the GitHub repository (in the case of technology models).

NREL will perform quality assurance and periodically update the production version of the data and software, both of which can be fetched by DOE on a regular basis.

8.4 Components

8.4.1 DOE Server

The DOE server for Tyche resides within the DOE intranet. It fetches software updates from the GitHub Software Repository and fetches data updates from the NREL Data Lake. (Because data volumes are small, it could perform these automatically on a daily or weekly basis during off hours.) It runs a Quart HTTP server within a Conda environment. Requirements for this server are as follows:

1. Linux (preferred) or Windows.
2. Four to 16 CPU cores and at least 32 GB of memory.
3. An up-to-date installation (version 4.8.3 or later) of the Conda software package manager.
4. Installation of the Tyche environment within Conda. (This will install the correct version of Python and the other required software, so those need not be installed individually.) See the attachment conda-environment.yml.
5. Running a shell script for the Quart HTTP server.
6. Open outgoing HTTPS ports for `GET` requests to the NREL Data Lake and GitHub.com.
7. An open HTTP incoming port from client web browsers withing the DOE intranet.
8. A folder on disk that is regularly backed up.

8.4.2 NREL Data Lake

The NREL Data Lake, which is housed on Amazon Web Services (AWS), contains all of the non-sensitive data, such as the parameters for technology models and the results of expert elicitations. NREL curates the data that is pushed to the data lake.

8.4.3 GitHub Software Repository

The Tyche software resides on the NREL GitHub software repository <https://github.com/NREL/tyche/>. The `production` branch contains the latest deployable version of the software. Other branches contain work in progress, contributions from DOE and its subcontractors, and the `development` (pre-release) version of the software.

8.5 Security Considerations

1. NREL has authority to operate (ATO) with non-sensitive software and data on its Data Lake and on GitHub.com.

2. Sensitive data (in the form of scenario definitions and results) may reside on the DOE server and on the laptops of DOE users.
3. The Tyche service only makes HTTPS `GET` requests outside of the DOE intranet, and these only consist of fetching non-sensitive datasets and technology models. Thus, the firewall for the Tyche server should be configured at follows:
    1. Block all incoming traffic from outside the DOE intranet.
    2. Allow incoming HTTP traffic from inside the DOE intranet.
    3. Allow outgoing HTTPS traffic to NREL Data Lake and GitHub.com.
    4. Block all other outgoing traffic.
4. Ideally, the Tyche software (and its library dependencies) and its updates should undergo a security audit.

CHAPTER

NINE

PYTHON API

The module *tyche* contains defines and solves multi-objective R&D optimization problems, which the module *eutychia* provides a server of a web-based user interface. The module *technology* definies individual R&D technologies.

9.1 Tyche Module

9.1.1 tyche.DecisionGUI

Interactive exploration of a technology.

class tyche.DecisionGUI.DecisionWindow(*evaluator*)

Bases: `object`

Class for displaying an interactive interface to explore cost-benefit tradeoffs for a technology.

create_figure(*i*, *j*) → matplotlib.figure.Figure

mainloop()

Run the interactive interface.

reevaluate(*next=<function DecisionWindow.<lambda>>*, *delay=200*)

Recalculate the results after a delay.

Parameters

- **next** (*function*) – The operation to perform after completing the recalculation.
- **delay** (*int*) – The number of milliseconds to delay before the recalculation.

reevaluate_immediate(*next=<function DecisionWindow.<lambda>>*)

Recalculate the results immediately.

Parameters **next** (*function*) – The operation to perform after completing the recalculation.

refresh()

Refresh the graphics after a delay.

refresh_immediate()

Refresh the graphics immediately.

9.1.2 tyche.Designs

Designs for technologies.

class tyche.Designs.**Designs**(*path=None*, *uncertain=True*, *indices='indices.csv'*, *functions='functions.csv'*, *designs='designs.csv'*, *parameters='parameters.csv'*, *results='results.csv'*)

Bases: `object`

Designs for a technology.

indices
The *indices* table.

Type DataFrame

functions
The *functions* table.

Type DataFrame

designs
The *designs* table.

Type DataFrame

parameters
The *parameters* table.

Type DataFrame

results
The *results* table.

Type DataFrame

compile()
Compile the production and metrics functions.

evaluate(*technology*, *sample_count=1*)
Evaluate the performance of a technology.

Parameters

- **technology** (*str*) – The name of the technology.
- **sample_count** (*int*) – The number of random samples.

evaluate_scenarios(*sample_count=1*)
Evaluate scenarios.

Parameters **sample_count** (*int*) – The number of random samples.

vectorize_designs(*technology*, *scenario_count*, *sample_count=1*)
Make an array of designs.

vectorize_indices(*technology*)
Make an array of indices.

vectorize_parameters(*technology*, *scenario_count*, *sample_count=1*)
Make an array of parameters.

vectorize_scenarios(*technology*)
Make an array of scenarios.

vectorize_technologies()
Make an array of technologies.

tyche.Designs.**sampler**(*x*, *sample_count*)
Sample from an array.

Parameters

- **x** (*array*) – The array.
- **sample_count** (*int*) – The sample size.

9.1.3 tyche.Distributions

Utilities for probability distributions.

`tyche.Distributions.choice(a, size=None, replace=True, p=None)`

Generates a random sample from a given 1-D array

New in version 1.7.0.

Note: New code should use the `choice` method of a `default_rng()` instance instead; please see the random-quick-start.

Parameters

- **a** (*1-D array-like or int*) – If an ndarray, a random sample is generated from its elements. If an int, the random sample is generated as if a were np.arange(a)
- **size** (*int or tuple of ints, optional*) – Output shape. If the given shape is, e.g., `(m, n, k)`, then `m * n * k` samples are drawn. Default is None, in which case a single value is returned.
- **replace** (*boolean, optional*) – Whether the sample is with or without replacement
- **p** (*1-D array-like, optional*) – The probabilities associated with each entry in a. If not given the sample assumes a uniform distribution over all entries in a.

Returns samples – The generated random samples

Return type single item or ndarray

Raises `ValueError` – If a is an int and less than zero, if a or p are not 1-dimensional, if a is an array-like of size 0, if p is not a vector of probabilities, if a and p have different lengths, or if replace=False and the sample size is greater than the population size

See also:

`randint()`, `shuffle()`, `permutation()`

`Generator.choice()` which should be used in new code

Notes

Sampling random rows from a 2-D array is not possible with this function, but is possible with *Generator.choice* through its `axis` keyword.

Examples

Generate a uniform random sample from np.arange(5) of size 3:

```
>>> np.random.choice(5, 3)
array([0, 3, 4]) # random
>>> #This is equivalent to np.random.randint(0,5,3)
```

Generate a non-uniform random sample from np.arange(5) of size 3:

```
>>> np.random.choice(5, 3, p=[0.1, 0, 0.3, 0.6, 0])
array([3, 3, 0]) # random
```

Generate a uniform random sample from np.arange(5) of size 3 without replacement:

```
>>> np.random.choice(5, 3, replace=False)
array([3,1,0]) # random
>>> #This is equivalent to np.random.permutation(np.arange(5))[:3]
```

Generate a non-uniform random sample from np.arange(5) of size 3 without replacement:

```
>>> np.random.choice(5, 3, replace=False, p=[0.1, 0, 0.3, 0.6, 0])
array([2, 3, 0]) # random
```

Any of the above can be repeated with an arbitrary array-like instead of just integers. For instance:

```
>>> aa_milne_arr = ['pooh', 'rabbit', 'piglet', 'Christopher']
>>> np.random.choice(aa_milne_arr, 5, p=[0.5, 0.1, 0.1, 0.3])
array(['pooh', 'pooh', 'pooh', 'Christopher', 'piglet'], # random
      dtype='<U11')
```

`tyche.Distributions.`**`constant`**`(`*value*`)`

The constant distribution.

Parameters **`value`** (*`float`*) – The constant value.

`tyche.Distributions.`**`mixture`**`(`*weights*, *distributions*`)`

A mixture of two distributions.

Parameters

- **`weights`** (*`array of float`*) – The weights of the distributions to be mixed.
- **`distributions`** (*`array of distributions`*) – The distributions to be mixed.

`tyche.Distributions.`**`parse_distribution`**`(`*text*`)`

Make the Python object for the distribution, if any is specified.

Parameters **`text`** (*`str`*) – The Python expression for the distribution, or plain text.

9.1.4 tyche.EpsilonConstraints

Epsilon-constraint optimization.

class `tyche.EpsilonConstraints.`**`EpsilonConstraintOptimizer`**`(`*evaluator*, *scale=1000000.0*`)`

Bases: `object`

An epsilon-constration multi-objective optimizer.

`evaluator`

The technology evaluator.

Type tyche.Evaluator

`scale`
The scaling factor for output.

Type float

`opt_diffev`(*metric*, *sense=None*, *max_amount=None*, *total_amount=None*, *eps_metric=None*, *statistic=<function mean>*, *strategy='best1bin'*, *seed=2*, *tol=0.01*, *maxiter=75*, *init='latinhypercube'*, *verbose=0*)
Maximize the objective function using the differential_evoluaion algorithm.

Parameters

- **metric** (*str*) – Name of metric to maximize. The objective function.
- **sense** (*str*) –

  Optimization sense ('min' or 'max'). If no value is provided to this method, the sense value used to create the EpsilonConstraintOptimizer object is used instead.
- **max_amount** (*DataFrame*) – Maximum investment amounts by R&D category (defined in investments data) and maximum metric values
- **total_amount** (*float*) – Upper limit on total investments summed across all R&D categories.
- **eps_metric** (*Dict*) – RHS of the epsilon constraint(s) on one or more metrics. Keys are metric names, and the values are dictionaries of the form {'limit': float, 'sense': str}. The sense defines whether the epsilon constraint is a lower or an upper bound, and the value must be either 'upper' or 'lower'.
- **statistic** (*function*) – Summary statistic used on the sample evaluations; the metric measure that is fed to the optimizer.
- **strategy** (*str*) – Which differential evolution strategy to use. 'best1bin' is the default. See algorithm docs for full list.
- **seed** (*int*) – Sets the random seed for optimization by creating a new *RandomState* instance. Defaults to 1. Not setting this parameter means the solutions will not be reproducible.
- **init** (*str or array-like*) – Type of population initialization. Default is Latin hypercube; alternatives are 'random' or specifying every member of the initial population in an array of shape (popsize, len(variables)).
- **tol** (*float*) – Relative tolerance for convergence
- **maxiter** (*int*) – Upper limit on generations of evolution (analogous to algorithm iterations)
- **verbose** (*int*) – Verbosity level returned by this outer function and the differential_evolution algorithm. verbose = 0 No messages verbose = 1 Objective function value at every algorithm iteration verbose = 2 Investment constraint status, metric constraint status, and objective function value verbose = 3 Decision variable values, investment constraint status, metric constraint status, and objective function value verbose > 3 All metric values, decision variable values, investment constraint status, metric constraint status, and objective function value

`opt_milp`(*metric*, *sense=None*, *max_amount=None*, *total_amount=None*, *eps_metric=None*, *statistic=<function mean>*, *sizelimit=1000000.0*, *verbose=0*)
Maximize the objective function using a piecewise linear representation to create a mixed integer linear program.

Parameters

- **metric** (*str*) – Name of metric to maximize
- **sense** (*str*) – Optimization sense ('min' or 'max'). If no value is provided to this method, the sense value used to create the EpsilonConstraintOptimizer object is used instead.
- **max_amount** (*DataFrame*) – Maximum investment amounts by R&D category (defined in investments data) and maximum metric values
- **total_amount** (*float*) – Upper limit on total investments summed across all R&D categories.
- **eps_metric** (*Dict*) – RHS of the epsilon constraint(s) on one or more metrics. Keys are metric names, and the values are dictionaries of the form {'limit': float, 'sense': str}. The sense defines whether the epsilon constraint is a lower or an upper bound, and the value must be either 'upper' or 'lower'.
- **statistic** (*function*) – Summary statistic (metric measure) fed to evaluator_corners_wide method in Evaluator
- **total_amount** – Upper limit on total investments summed across all R&D categories
- **sizelimit** (*int*) – Maximum allowed number of binary variables. If the problem size exceeds this limit, pwlinear_milp will exit before building or solving the model.
- **verbose** (*int*) – A value greater than zero will save the optimization model as a .lp file A value greater than 1 will print out status messages

Returns Optimum – exit_code exit_message amounts (None, if no solution found) metrics (None, if no solution found) solve_time opt_sense

Return type NamedTuple

opt_shgo(*metric*, *sense=None*, *max_amount=None*, *total_amount=None*, *eps_metric=None*, *statistic=<function mean>*, *tol=0.01*, *maxiter=None*, *sampling_method='simplicial'*, *verbose=0*)

Maximize the objective function using the shgo global optimization algorithm.

Parameters

- **metric** (*str*) – Name of metric to maximize.
- **sense** (*str*) – Optimization sense ('min' or 'max'). If no value is provided to this method, the sense value used to create the EpsilonConstraintOptimizer object is used instead.
- **max_amount** (*DataFrame*) – Maximum investment amounts by R&D category (defined in investments data) and maximum metric values
- **total_amount** (*float*) – Upper metric_limit on total investments summed across all R&D categories.
- **eps_metric** (*Dict*) – RHS of the epsilon constraint(s) on one or more metrics. Keys are metric names, and the values are dictionaries of the form {'limit': float, 'sense': str}. The sense defines whether the epsilon constraint is a lower or an upper bound, and the value must be either 'upper' or 'lower'.
- **statistic** (*function*) – Summary metric_statistic used on the sample evaluations; the metric measure that is fed to the optimizer.
- **tol** (*float*) – Objective function tolerance in stopping criterion.

- **maxiter** (*int*) – Upper metric_limit on iterations that can be performed. Defaults to None. Specifying this parameter can cause shgo to stall out instead of solving.
- **sampling_method** (*str*) – Allowable values are 'sobol and 'simplicial'. Simplicial is default, uses less memory, and guarantees convergence (theoretically). Sobol is faster, uses more memory and does not guarantee convergence. Per documentation, Sobol is better for "easier" problems.
- **verbose** (*int*) – Verbosity level returned by this outer function and the SHGO algorithm. verbose = 0 No messages verbose = 1 Convergence messages from SHGO algorithm verbose = 2 Investment constraint status, metric constraint status, and convergence messages verbose = 3 Decision variable values, investment constraint status, metric constraint status, and convergence messages verbose > 3 All metric values, decision variable values, investment constraint status, metric constraint status, and convergence messages

opt_slsqp(*metric*, *sense=None*, *max_amount=None*, *total_amount=None*, *eps_metric=None*, *statistic=<function mean>*, *initial=None*, *tol=1e-08*, *maxiter=50*, *verbose=0*)

Optimize the objective function using the fmin_slsqp algorithm.

Parameters

- **metric** (*str*) – Name of metric to maximize.
- **sense** (*str*) – Optimization sense ('min' or 'max'). If no value is provided to this method, the sense value used to create the EpsilonConstraintOptimizer object is used instead.
- **max_amount** (*DataFrame*) – Maximum investment amounts by R&D category (defined in investments data) and maximum metric values
- **total_amount** (*float*) – Upper limit on total investments summed across all R&D categories.
- **eps_metric** (*Dict*) – RHS of the epsilon constraint(s) on one or more metrics. Keys are metric names, and the values are dictionaries of the form {'limit': float, 'sense': str}. The sense defines whether the epsilon constraint is a lower or an upper bound, and the value must be either 'upper' or 'lower'.
- **statistic** (*function*) – Summary statistic used on the sample evaluations; the metric measure that is fed to the optimizer.
- **initial** (*array of float*) – Initial value of decision variable(s) fed to the optimizer.
- **tol** (*float*) – Search tolerance fed to the optimizer.
- **maxiter** (*int*) – Maximum number of iterations the optimizer is permitted to execute.
- **verbose** (*int*) – Verbosity level returned by the optimizer and this outer function. Defaults to 0. verbose = 0 No messages verbose = 1 Summary message when fmin_slsqp completes verbose = 2 Status of each algorithm iteration and summary message verbose = 3 Investment constraint status, metric constraint status, status of each algorithm iteration, and summary message verbose > 3 All metric values, decision variable values, investment constraint status, metric constraint status, status of each algorithm iteration, and summary message

optimum_metrics(*max_amount=None*, *total_amount=None*, *sense=None*, *statistic=<function mean>*, *tol=1e-08*, *maxiter=50*, *verbose=0*)

Maximum value of metrics.

Parameters

- **max_amount** (*DataFrame*) – The maximum amounts that can be invested in each category.
- **total_amount** (*float*) – The maximum amount that can be invested *in toto*.
- **sense** (*Dict or str*) – Optimization sense for each metric. Must be 'min' or 'max'. If None, then the sense provided to the EpsilonConstraintOptimizer class is used for all metrics. If string, the sense is used for all metrics.
- **statistic** (*function*) – The statistic used on the sample evaluations.
- **tol** (*float*) – The search tolerance.
- **maxiter** (*int*) – The maximum iterations for the search.
- **verbose** (*int*) – Verbosity level.

9.1.5 tyche.Evaluator

Fast evaluation of technology investments.

class tyche.Evaluator.**Evaluator**(*tranches*)

Bases: object

Evalutate technology investments using a response surface.

amounts

Cost of tranches.

Type DataFrame

categories

Categories of investment.

Type DataFrame

metrics

Metrics for technologies.

Type DataFrame

units

Units of measure for metrics.

Type DataFrame

interpolators

Interpolation functions for technology metrics.

Type DataFrame

evaluate(*amounts*)

Sample the distribution for an investment.

Parameters **amounts** (*DataFrame*) – The investment levels.

evaluate_corners_semilong(*statistic=<function mean>*)

Return a dataframe indexed my investment amounts in each category, with columns for each metric.

Parameters **statistic** (*function*) – The statistic to evaluate.

evaluate_corners_wide(*statistic=<function mean>*)
Return a dataframe indexed my investment amounts in each category, with columns for each metric.

Parameters **statistic** (*function*) – The statistic to evaluate.

evaluate_statistic(*amounts*, *statistic=<function mean>*)
Evaluate a statistic for an investment.

Parameters

- **amounts** (*DataFrame*) – The investment levels.
- **statistic** (*function*) – The statistic to evaluate.

make_statistic_evaluator(*statistic=<function mean>*)
Return a function that evaluates a statistic for an investment.

Parameters **statistic** (*function*) – The statistic to evaluate.

9.1.6 tyche.IO

I/O utilities for Tyche.

tyche.IO.**make_table**(*dtypes*, *index*)
Make a data frame from column types and an index.

Parameters

- **dtypes** (*array*) – The column types.
- **index** (*array*) – The index.

tyche.IO.**read_table**(*path*, *name*, *dtypes*, *index*)
Read a data table from a file.

Parameters

- **path** (*str*) – The path to the folder.
- **name** (*str*) – The filename for the table.
- **dtypes** (*array*) – The column types.
- **index** (*array*) – The index.

9.1.7 tyche.Investments

Investments in technologies.

class tyche.Investments.**Investments**(*path=None*, *uncertain=False*, *tranches='tranches.csv'*, *investments='investments.csv'*)
Bases: object

Investments in a technology.

tranches
The *tranches* table.

Type DataFrame

investments
The *investments* table.

Type DataFrame

compile()
Parse any probability distributions in the tranches.

evaluate_investments(*designs*, *tranche_results=None*, *sample_count=1*)
Evaluate the investments for a design.

Parameters

- **designs** (*tyche.Designs*) – The designs.
- **tranche_results** (*tyche.Evaluations*) – Output of evaluate_tranches method. Necessary only if the investment amounts contain uncertainty.
- **sample_count** (*int*) – The number of random samples.

evaluate_tranches(*designs*, *sample_count=1*)
Evaluate the tranches of investment for a design.

Parameters

- **designs** (*tyche.Designs*) – The designs.
- **sample_count** (*int*) – The number of random samples.

9.1.8 tyche.Types

Data types for Tyche.

class tyche.Types.**Evaluations**(*amounts*, *metrics*, *summary*, *uncertain*)
Bases: `tuple`

Named tuple type for rows in the *evaluations* table.

amounts
Alias for field number 0

metrics
Alias for field number 1

summary
Alias for field number 2

uncertain
Alias for field number 3

class tyche.Types.**FakeDistribution**(*rvs*)
Bases: `tuple`

Named tuple type for a fake distribution.

rvs
Alias for field number 0

class tyche.Types.**Functions**(*style*, *capital*, *fixed*, *production*, *metric*)
Bases: `tuple`

Name tuple type for rows in the *functions* table.

capital
Alias for field number 1

fixed
Alias for field number 2

metric
Alias for field number 4

production
Alias for field number 3

style
Alias for field number 0

class tyche.Types.**Indices**(*capital*, *input*, *output*, *metric*)
Bases: tuple

Name tuple type for rows in the *indices* table.

capital
Alias for field number 0

input
Alias for field number 1

metric
Alias for field number 3

output
Alias for field number 2

class tyche.Types.**Inputs**(*lifetime*, *scale*, *input*, *input_efficiency*, *input_price*, *output_efficiency*, *output_price*)
Bases: tuple

Named tuple type for rows in the *inputs* table.

input
Alias for field number 2

input_efficiency
Alias for field number 3

input_price
Alias for field number 4

lifetime
Alias for field number 0

output_efficiency
Alias for field number 5

output_price
Alias for field number 6

scale
Alias for field number 1

class tyche.Types.**Optimum**(*exit_code*, *exit_message*, *amounts*, *metrics*, *solve_time*, *opt_sense*)
Bases: tuple

Named tuple type for optimization results.

amounts
Alias for field number 2

exit_code
Alias for field number 0

exit_message
Alias for field number 1

metrics
Alias for field number 3

opt_sense
Alias for field number 5

solve_time
Alias for field number 4

class tyche.Types.Results(*cost*, *output*, *metric*)
Bases: `tuple`

Named tuple type for rows in the *results* table.

cost
Alias for field number 0

metric
Alias for field number 2

output
Alias for field number 1

9.1.9 Module contents

Tyche: a Python package for R&D pathways analysis and evaluation.

9.2 Eutychia (User Interface) Module

9.2.1 Submodules

9.2.2 eutychia.example module

Example script for multiple objective optimization of residential PV.

9.2.3 eutychia.main module

9.2.4 Module contents

Eutychia: user interface for a Python package for R&D pathways analysis and evaluation.

9.3 Technology Module

9.3.1 Pre-Built Technology Models and Datasets

Residential Photovoltaics

Generic model for residential PV.

This PV model tracks components, technologies, critical materials, and hazardous waste.

Table 9.1: Elements of `capital` arrays.

Index	Description	Units
0	module capital cost	$/system
1	inverter capital cost	$/system
2	balance capital cost	$/system

Table 9.2: Elements of `fixed` arrays.

Index	Description	Units
0	fixed cost	$/system

Table 9.3: Elements of `input` arrays.

Index	Description	Units
0	strategic metals	g/system

Table 9.4: Elements of `output` arrays.

Index	Description	Units
0	lifetime energy production	kWh/system
1	lifecycle hazardous waste	g/system
2	lifetime greenhouse gas production	gCO2e/system

Table 9.5: Elements of `metric` arrays.

Index	Description	Units
0	system cost	$/Wdc
1	levelized energy cost	$/kWh
2	greenhouse gas	gCO2e/kWh
3	strategic metal	g/kWh
4	hazardous waste	g/kWh
5	specific yield	hr/yr
6	module efficiency	%/100
7	module lifetime	yr

Table 9.6: Elements of `parameter` arrays.

Index	Description	Units
0	discount rate	1/yr
1	insolation	W/m^2
2	system size	m^2
3	module capital cost	$/m^2
4	module lifetime	yr
5	module efficiency	%/100
6	module aperture	%/100
7	module fixed cost	$/kW/yr
8	module degradation rate	1/yr
9	location capacity factor	%/100
10	module soiling loss	%/100
11	inverter capital cost	$/W
12	inverter lifetime	yr
13	inverter replacement cost	%/100
14	inverter efficiency	%/100
15	hardware capital cost	$/m^2
16	installation labor cost	$/system
17	permitting cost	$/system
18	customer acquisition cost	$/system
19	installer overhead cost	%/100
20	hazardous waste content	g/m^2
21	greenhouse gas offset	gCO2e/kWh
22	benchmark LCOC	$/Wdc
23	benchmark LCOE	$/kWh

`technology.pv_residential_large.`**`capital_cost`**`(`*scale*, *parameter*`)`
Capital cost function.

Parameters

- **`scale`** (*float*) – The scale of operation.
- **`parameter`** (*array*) – The technological parameterization.

`technology.pv_residential_large.`**`discount`**`(`*rate*, *time*`)`
Discount factor over a time period.

Parameters

- **`rate`** (*float*) – The discount rate per time period.
- **`time`** (*int*) – The number of time periods.

`technology.pv_residential_large.`**`fixed_cost`**`(`*scale*, *parameter*`)`
Fixed cost function.

Parameters

- **`scale`** (*float*) – The scale of operation.
- **`parameter`** (*array*) – The technological parameterization.

`technology.pv_residential_large.`**`metrics`**`(`*scale*, *capital*, *lifetime*, *fixed*, *input_raw*, *input*, *input_price*, *output_raw*, *output*, *cost*, *parameter*`)`
Metrics function.

Parameters

- **scale** (*float*) – The scale of operation.
- **capital** (*array*) – Capital costs.
- **lifetime** (*float*) – Technology lifetime.
- **fixed** (*array*) – Fixed costs.
- **input_raw** (*array*) – Raw input quantities (before losses).
- **input** (*array*) – Input quantities.
- **output_raw** (*array*) – Raw output quantities (before losses).
- **output** (*array*) – Output quantities.
- **cost** (*array*) – Costs.
- **parameter** (*array*) – The technological parameterization.

`technology.pv_residential_large.module_power(parameter)`
Nominal module energy production.

Parameters **parameter** (*array*) – The technological parameterization.

`technology.pv_residential_large.npv(rate, time)`
Net present value of constant cash flow.

Parameters

- **rate** (*float*) – The discount rate per time period.
- **time** (*int*) – The number of time periods.

`technology.pv_residential_large.performance_ratio(parameter)`
Performance ratio for the system.

Parameters **parameter** (*array*) – The technological parameterization.

`technology.pv_residential_large.production(scale, capital, lifetime, fixed, input, parameter)`
Production function.

Parameters

- **scale** (*float*) – The scale of operation.
- **capital** (*array*) – Capital costs.
- **lifetime** (*float*) – Technology lifetime.
- **fixed** (*array*) – Fixed costs.
- **input** (*array*) – Input quantities.
- **parameter** (*array*) – The technological parameterization.

`technology.pv_residential_large.specific_yield(parameter)`
Specific yield for the system.

Parameters **parameter** (*array*) – The technological parameterization.

Simple Residential Photovoltaics

Simple residential PV.

`technology.pv_residential_simple.capital_cost(scale, parameter)`
Capital cost function.

Parameters

- **scale** (*float*) – The scale of operation.
- **parameter** (*array*) – The technological parameterization.

`technology.pv_residential_simple.discount(rate, time)`
Discount factor over a time period.

Parameters

- **rate** (*float*) – The discount rate per time period.
- **time** (*int*) – The number of time periods.

`technology.pv_residential_simple.fixed_cost(scale, parameter)`
Fixed cost function.

Parameters

- **scale** (*float*) – The scale of operation.
- **parameter** (*array*) – The technological parameterization.

`technology.pv_residential_simple.metrics(scale, capital, lifetime, fixed, input_raw, input, input_price, output_raw, output, cost, parameter)`
Metrics function.

Parameters

- **scale** (*float*) – The scale of operation.
- **capital** (*array*) – Capital costs.
- **lifetime** (*float*) – Technology lifetime.
- **fixed** (*array*) – Fixed costs.
- **input_raw** (*array*) – Raw input quantities (before losses).
- **input** (*array*) – Input quantities.
- **output_raw** (*array*) – Raw output quantities (before losses).
- **output** (*array*) – Output quantities.
- **cost** (*array*) – Costs.
- **parameter** (*array*) – The technological parameterization.

`technology.pv_residential_simple.npv(rate, time)`
Net present value of constant cash flow.

Parameters

- **rate** (*float*) – The discount rate per time period.
- **time** (*int*) – The number of time periods.

`technology.pv_residential_simple.production(scale, capital, lifetime, fixed, input, parameter)`
Production function.

Parameters

- **scale** (*float*) – The scale of operation.
- **capital** (*array*) – Capital costs.
- **lifetime** (*float*) – Technology lifetime.

- **fixed** (*array*) – Fixed costs.
- **input** (*array*) – Input quantities.
- **parameter** (*array*) – The technological parameterization.

Utility-Scale Photovoltaics

Simple pv utility-scale module example. Inspired by Kavlak et al. Energy Policy 123 (2018) 700–710.

`technology.utility_pv.capital_cost(scale, parameter)`
Capital cost function.

Parameters

- **scale** (*float*) – The scale of operation.
- **parameter** (*array*) – The technological parameterization.

`technology.utility_pv.fixed_cost(scale, parameter)`
Fixed cost function.

Parameters

- **scale** (*float*) – The scale of operation.
- **parameter** (*array*) – The technological parameterization.

`technology.utility_pv.metrics(scale, capital, lifetime, fixed, input_raw, input, input_price, output_raw, output, cost, parameter)`
Metrics function.

Parameters

- **scale** (*float*) – The scale of operation.
- **capital** (*array*) – Capital costs.
- **lifetime** (*float*) – Technology lifetime.
- **fixed** (*array*) – Fixed costs.
- **input_raw** (*array*) – Raw input quantities (before losses).
- **input** (*array*) – Input quantities.
- **output_raw** (*array*) – Raw output quantities (before losses).
- **output** (*array*) – Output quantities.
- **cost** (*array*) – Costs.
- **parameter** (*array*) – The technological parameterization.

`technology.utility_pv.production(scale, capital, lifetime, fixed, input, parameter)`
Production function.

Parameters

- **scale** (*float*) – The scale of operation.
- **capital** (*array*) – Capital costs.
- **lifetime** (*float*) – Technology lifetime.
- **fixed** (*array*) – Fixed costs.
- **input** (*array*) – Input quantities.

- **parameter** (*array*) – The technological parameterization.

Transportation

Phase-1 model to estimate the cost, energy, and emissions associated with a particular vehicle/transport technology.

`technology.transport_model.capital_cost(scale, parameter)`
Capital cost function.

Parameters

- **scale** (*float*) – The scale of operation.
- **parameter** (*array*) – The technological parameterization.

`technology.transport_model.fixed_cost(scale, parameter)`
Capital cost function.

Parameters

- **scale** (*float*) – The scale of operation.
- **parameter** (*array*) – The technological parameterization.

`technology.transport_model.metrics(scale, capital, lifetime, fixed, input_raw, input, input_price, output_raw, output, cost, parameter)`
Metrics function.

Parameters

- **scale** (*float*) – The scale of operation.
- **capital** (*array*) – Capital costs.
- **lifetime** (*float*) – Technology lifetime.
- **fixed** (*array*) – Fixed costs.
- **input_raw** (*array*) – Raw input quantities (before losses).
- **input** (*array*) – Input quantities.
- **output_raw** (*array*) – Raw output quantities (before losses).
- **output** (*array*) – Output quantities.
- **cost** (*array*) – Costs.
- **parameter** (*array*) – The technological parameterization.

`technology.transport_model.production(scale, capital, lifetime, fixed, input, parameter)`
Production function.

Parameters

- **scale** (*float*) – The scale of operation.
- **capital** (*array*) – Capital costs.
- **lifetime** (*float*) – Technology lifetime.
- **fixed** (*array*) – Fixed costs.
- **input** (*array*) – Input quantities.
- **parameter** (*array*) – The technological parameterization.

9.3.2 Tutorial Technologies

The technology models in this section are for exploratory and learning purposes only.

Simple Electrolysis

Simple electrolysis.

`technology.simple_electrolysis.capital_cost(scale, parameter)`
Capital cost function.

Parameters

- **scale** (*float*) – The scale of operation.
- **parameter** (*array*) – The technological parameterization.

`technology.simple_electrolysis.fixed_cost(scale, parameter)`
Fixed cost function.

Parameters

- **scale** (*float*) – The scale of operation.
- **parameter** (*array*) – The technological parameterization.

`technology.simple_electrolysis.metrics(scale, capital, lifetime, fixed, input_raw, input, input_price, output_raw, output, cost, parameter)`
Metrics function.

Parameters

- **scale** (*float*) – The scale of operation.
- **capital** (*array*) – Capital costs.
- **lifetime** (*float*) – Technology lifetime.
- **fixed** (*array*) – Fixed costs.
- **input_raw** (*array*) – Raw input quantities (before losses).
- **input** (*array*) – Input quantities.
- **output_raw** (*array*) – Raw output quantities (before losses).
- **output** (*array*) – Output quantities.
- **cost** (*array*) – Costs.
- **parameter** (*array*) – The technological parameterization.

`technology.simple_electrolysis.production(scale, capital, lifetime, fixed, input, parameter)`
Production function.

Parameters

- **scale** (*float*) – The scale of operation.
- **capital** (*array*) – Capital costs.
- **lifetime** (*float*) – Technology lifetime.
- **fixed** (*array*) – Fixed costs.
- **input** (*array*) – Input quantities.

- **parameter** (*array*) – The technological parameterization.

Toy Biorefinery

Biorefinery model with four processing steps.

`technology.tutorial_biorefinery.`**`capital_cost`**(*scale*, *parameter*)
Capital cost function.

Parameters

- **scale** (*float*) – The scale of operation.
- **parameter** (*array*) – The technological parameterization.

Returns

Return type Total capital cost for one biorefinery (USD/biorefinery)

`technology.tutorial_biorefinery.`**`fixed_cost`**(*scale*, *parameter*)
Fixed cost function.

Parameters

- **scale** (*float [Unused]*) – The scale of operation.
- **parameter** (*array*) – The technological parameterization.

Returns

Return type total fixed costs for one biorefinery (USD/year)

`technology.tutorial_biorefinery.`**`metrics`**(*scale*, *capital*, *lifetime*, *fixed*, *input_raw*, *input*, *input_price*, *output_raw*, *output*, *cost*, *parameter*)
Metrics function.

Parameters

- **scale** (*float*) – The scale of operation. Unitless
- **capital** (*array*) – Capital costs. Units: USD/biorefinery
- **lifetime** (*float*) – Technology lifetime. Units: year
- **fixed** (*array*) – Fixed costs. Units: USD/year
- **input_raw** (*array*) – Raw input quantities (before losses). Units: metric ton feedstock/year
- **input** (*array*) – Input quantities. Units: metric ton feedstock/year
- **input_price** (*array`*) – Array of input prices. Various units.
- **output_raw** (*array*) – Raw output quantities (before losses). Units: gal biofuel/year
- **output** (*array*) – Output quantities. Units: gal biofuel/year
- **cost** (*array*) – Costs.
- **parameter** (*array*) – The technological parameterization. Units vary; given in comments below

`technology.tutorial_biorefinery.`**`production`**(*scale*, *capital*, *lifetime*, *fixed*, *input*, *parameter*)
Production function.

Parameters

- **scale** (*float*) – The scale of operation.
- **capital** (*array*) – Capital costs.
- **lifetime** (*float*) – Technology lifetime.
- **fixed** (*array*) – Fixed costs.
- **input** (*array*) – Input quantities.
- **parameter** (*array*) – The technological parameterization.

Returns Ideal/theoretical production of each technology output: biofuel at gals/year

Return type output_raw

Onshore Wind Turbines

Template for technology functions.

`technology.tutorial_basic.capital_cost`(*scale*, *parameter*)
Capital cost function.

Parameters

- **scale** (*float*) – The scale of operation.
- **parameter** (*array*) – The technological parameterization.

`technology.tutorial_basic.fixed_cost`(*scale*, *parameter*)
Capital cost function.

Parameters

- **scale** (*float*) – The scale of operation.
- **parameter** (*array*) – The technological parameterization.

`technology.tutorial_basic.metrics`(*scale*, *capital*, *lifetime*, *fixed*, *input_raw*, *input*, *input_price*, *output_raw*, *output*, *cost*, *parameter*)
Metrics function.

Parameters

- **scale** (*float*) – The scale of operation.
- **capital** (*array*) – Capital costs.
- **lifetime** (*float*) – Technology lifetime.
- **fixed** (*array*) – Fixed costs.
- **input_raw** (*array*) – Raw input quantities (before losses).
- **input** (*array*) – Input quantities.
- **output_raw** (*array*) – Raw output quantities (before losses).
- **output** (*array*) – Output quantities.
- **cost** (*array*) – Costs.
- **parameter** (*array*) – The technological parameterization.

`technology.tutorial_basic.production`(*scale*, *capital*, *lifetime*, *fixed*, *input*, *parameter*)
Production function.

Parameters

- **scale** (*float*) – The scale of operation.
- **capital** (*array*) – Capital costs.
- **lifetime** (*float*) – Technology lifetime.
- **fixed** (*array*) – Fixed costs.
- **input** (*array*) – Input quantities.
- **parameter** (*array*) – The technological parameterization.

9.3.3 Module contents

PYTHON MODULE INDEX