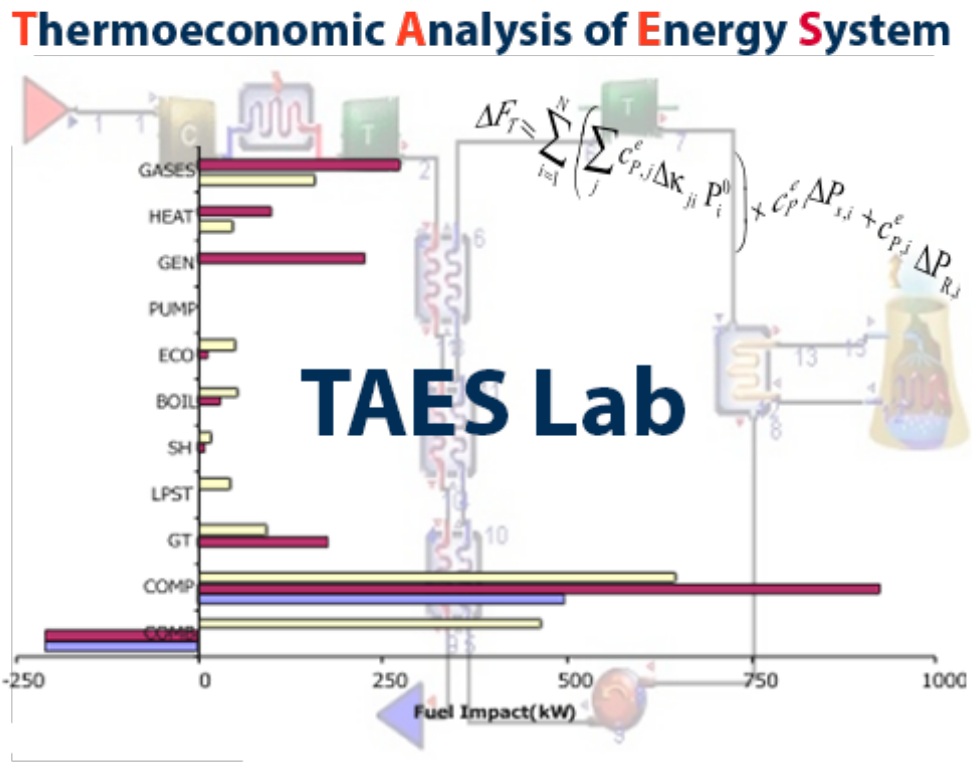
TaesLab

Reference Guide

Version: 26 May 2023



Instituto Universitario de Investigación Mixto
CIRCE
UniversidadZaragoza

Universidad
Zaragoza
1542

Table of Contents

1 Introduction

TaesLab, an acronym for Thermoeconomic Analysis of Energy Systems, is a MATLAB package tool designed to perform the thermoeconomic analysis of any energy system from its thermodynamic model and productive structure definition.

It is based on the works on Thermoeconomics developed in the CIRCE Institute and the Dpt. of Mechanical Engineering of the University of Zaragoza (Spain) since 1986. Their main characteristics are:

- MATLAB base applications and functions.
- Octave (GNU version of MATLAB) compatible
- Export and Import data and results in different formats, including Excel.
- Direct and Generalised Exergy Cost Calculation.
- Automatic generation of the Fuel-Product Table.
- Cost decomposition and formation process of products and waste analysis.
- Recycling Analysis.
- Thermoeconomic Diagnosis.

This manual describes the available functions and explains the basic handling functions with examples.

TaesLab is an evolution of TAESS 2.0 for Excel, with implements last advanced in Circular Thermoeconomics.

The functions are organised in four levels:

- Read and Check DataModel files.
- Create DataModel object
- Thermoeconomic Analysis computations.
- Show and Export Results.

2 TaesLab Package Installation

The TaesLab software is distributed in four different ways:

- Full package for Matlab: TaesLab.zip
- Full package for Octave: TaesLab_Octave.zip
- Matlab app: TaesLab.mlappinstal
- Standalone installation: TaesLab_install.exe

These files can be downladed at https://www.exergoecology.com/TaesLab

2.1 Full Package for MATLAB

This package includes all the files required to work in interactive form, examples, and documentation.

The folders are:

- Apps: Include the apps: TaesLab and ViewModelResults
- Base: Include the Base functions
- Classes: Include all the classes of the package
- Functions: Include additional functions required by the package
- Examples: Data model files of plant examples and templates to build the data files
- Documentation: Documentation of TaesLab
- LiveScripts: LiveScripts examples for how to use the base functions.

To install it:

- Unzip the file in the desired place.
- Open MATLAB, and in MATLAB's file explorer, go to the TaesLab folder.
- Right-click on the following folders, and select Add to Path > Folders and Subfolders:
  - Apps
  - Base
  - Classes
  - Functions
  - LiveScripts
- In the main toolbar, select ENVIRONMENT->Set Path, check that the desired folders have been included and press Button SAVE.

2.2 Full Package for Octave

This package includes all the files required to work in interactive form, examples, and documentation.

The folders are the same as for MATLAB except for Apps and LiveScripts, whose files do not work with Octave. It includes instead the folder Scripts with scripts utilities to work with TaesLab in Octave.

To install it:

- Unzip the file in the desired place.
- Open Octave, and in Octave's file explorer, go to the TaesLab folder.
- Execute the following commands in the console:

```
>> addpath C:\Documents\Termoeconomics\TaesLab
>> addpath C:\Documents\Termoeconomics\TaesLab\Scripts
>> addpath C:\Documents\Termoeconomics\TaesLab\Functions
>> addpath C:\Documents\Termoeconomics\TaesLab\Classes
>> addpath C:\Documents\Termoeconomics\TaesLab\Base
>> pkg load io
>> savepath
```

Assuming we have unzipped the files in `C:\Documents\Thermoeconomics\TaessLab`, these commands register the folders and install the additional io package, which allows it to read and write files in different formats.

2.3 TaesLab app

In this case, only the TaesLab app is installed.

To install it, double-click on the file `TaesLab.mlappinstall` and follow the instructions.

To run the app, go to APP and double-click on the TaesLab icon.

2.4 TaesLab Standalone program

This option does not require to have installed MATLAB.

To install it, double-click on the file `TaesLab_install.exe`

During the installation, the MATLAB runtime Library is installed. The program will be installed in the place you indicate and will create an icon on the desktop, if you will.

3 Data Model Definition

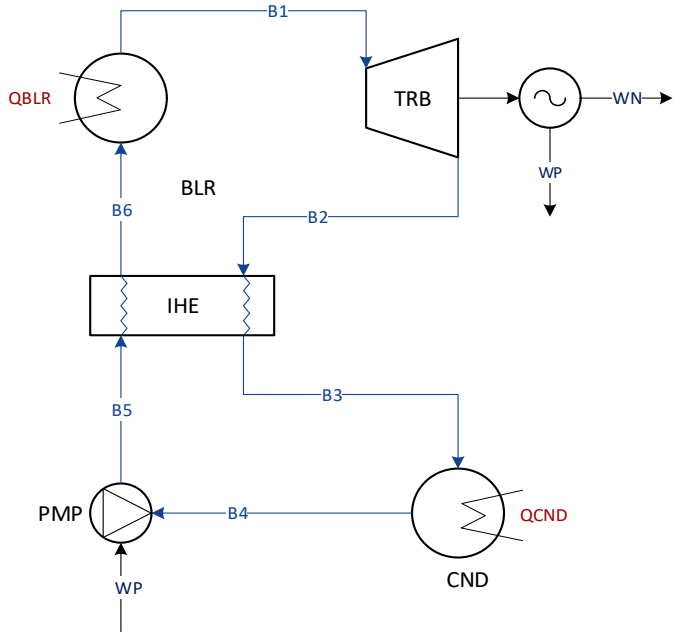
In this section, we describe the data model required for a thermoeconomic analysis of a plant.

The information is provided in files, which could be organised in two types of formats:

- Tabular format: Data is prepared in several tables. XLSX Spreadsheets and CSV text files are used.
- Structured format: Data is prepared in a structured file using JSON or XML formats.

ReadDataModel and *CheckDataModel* functions read and check these files and return the internal Data Model object to be used with other functions.

An Organic Rankine Cycle (ORC) is used as an example to illustrate the application's functionality. It uses waste heat to produce electricity. Part of the heat dissipated in the condenser could be recycled and used for air conditioning in another plant process. The data model files of this example can be found in the folder: `Examples/rorc`



Parameter	Value
ORC fluid	R245fa
Generator pressure P_{blr} (bar)	20
Inlet waste gases temperature T_G (ºC)	160
Generator pinch point ΔT_{blr} (ºC)	20
Condenser outlet temperature T_{cnd} (ºC)	30
Turbine isentropic efficiency η_{trb} (%)	80
Pump isentropic efficiency η_{pnp}(%)	85
IHE efficiency ε_{ihe}(%)	80
Net Power (kW)	50
Reference temperature T_0 (ºC)	20
Reference pressure P_0 (bar)	1.013

Figure 2.1. Physical diagram and design parameters of the ORC plant

3.1 Excel format

The Excel (XLSX) data model is a spreadsheet with a set of mandatory and optional sheets, which will be processed if they exist. Additional working sheets could be included with support information, like plant diagram thermodynamic properties of flows, …

The mandatory and optional sheets could use formulas and references.

There is a template file called *plant_model.xlsx* in the examples folder.

Flows sheet

This sheet contains the definition of the system's flows. There is one row per flow, and it is mandatory.

Fields

key – Name by which the flow is identified. This name must begin with a letter and may only contain letters and numbers. It is recommended that the length of this field does not exceed four characters.

type - Indicates how the flow is connected to the different processes in the plant.

There are four types of flows:

- INTERNAL - Connects two internal plant processes.
- RESOURCE - This is a flow of input resources to the plant.
- OUTPUT - It is the final product of the plant.
- WASTE - This is waste leaving the plant.

You could use additional fields or columns to include a more detailed description, such as the kind of flow (material, heat, work, …). The application does not process these fields.

Table 2.1 *Flows* sheet of the rorc_model.xlsx data model file.

key	type
B1	INTERNAL
B2	INTERNAL
B3	INTERNAL
B4	INTERNAL
B5	INTERNAL
B6	INTERNAL
QBLR	RESOURCE
WP	INTERNAL
WN	OUTPUT
QCND	WASTE

Processes sheet

This sheet contains the process description and its efficiency definition. One row per process and it is mandatory. Columns with additional information can be added (as description)

Fields

key - Name by which the process is identified. It is recommended that the length of this field does not exceed six characters.

fuel - Describes which flows constitute the fuel for the process. The flow names must be defined in the key field of the *Flows* table.

product - Describes which flows constitute the product of the process.

type - Indicates the function of the process within the plant. There are two types of processes.

- PRODUCTIVE – Its function is obtaining functional products. Its products are either internal plant flows or final products.

- DISSIPATIVE - Its function is to remove waste from the plant. Its product must be a WASTE type flow.

Table 2.2 *Processes* sheet of the rorc_model.xlsx data model file.

key	fuel	product	type	description
BLR	QBLR	B1-B6	PRODUCTIVE	Steam Generator
TRB	B1-B2	WN+WP	PRODUCTIVE	Turbine
IHE	B2-B3	B6-B5	PRODUCTIVE	Internal Heat Exchanger
PMP	WP	B5-B4	PRODUCTIVE	Pump
CND	B3-B4	QCND	DISSIPATIVE	Condenser

Exergy sheet

This sheet contains the values of the plant flow exergies for the different states or simulations we want to analyse. It is a double-entry table.Flow/State, where we indicate in each cell the flow exergy for each state or simulation of the plant

Fields

key – Keys of the flows, as defined in the *Flows* table.

<state> - Flow exergies for each plant state. <state> is a string of characters without blanks used as a column name identifying it. The length of this string shall not exceed eight characters. It is used in the application as *State* property.

Therefore, the number of rows of this table must be the same that the table *Flows*, and the number of columns is the number of states plus one (use for key).

Table 2.3.- *Exergy* sheet of the data model file rorc_model.xlsx

Key	TCND30	TCND45	ETAT75	PBLR15	NoIHE	N-Butane
B1	82.99	106.70	88.81	84.14	82.99	103.5
B2	17.32	40.68	18.92	18.96	17.32	37.2
B3	15.25	37.42	16.36	17.02	17.32	35.03
B4	5.71	9.71	6.11	6.39	5.711	24.87
B5	7.79	12.31	8.34	8.09	7.79	27.41
B6	9.30	15.13	10.17	9.51	7.79	29.02
QBLR	99.53	119.10	105.80	102.80	106.4	100.8
WP	2.43	3.02	2.60	1.98	2.432	2.967
WN	50.00	50.00	50.00	50.00	50	50
QCND	9.54	27.71	10.25	10.63	11.609	10.16

In this case, the model has six states defined. The column TCND30 contains the exergy values for the reference state of the plant. The column TCND45 contains the exergy values for the simulation of the plant working at a condensing temperature of 45ºC. Column ETAT75 contains the exergy values when the isentropic efficiency of the turbine is 75%. Column TCND45 contains the exergy values for the simulation of the plant working at an evaporation pressure of 15 bar. NoIHE simulates the plant working without IHE, and N-Butane simulates the plant using N-Butane instead of R245fa as the working fluid.

Format

This sheet contains the format and physical/economics units for the result tables of the thermoeconomic model. This sheet is mandatory.

Fields

key – Identifies the type of results to which the format applies. These values are the same for all the data models.

width - The number of digits with which the value is presented.

precision - The number of decimals with which the value is shown.

unit - Physical or economic unit of the value shown.

Table 2.4. *Format* sheet for the data model file rorc_model.xlsx

key	width	precision	unit
EXERGY	10	3	(kW)
EXERGY_COST	10	3	(kW)
EXERGY_UNIT_COST	10	4	(J/J)
GENERALIZED_COST	10	3	(c/h)
GENERALIZED_UNIT_COST	10	4	(c/kWh)
DIAGNOSIS	10	4	(kW)

According to this table, the EXERGY values will be shown with three decimals and kW as the unit, and the GENERALIZED_UNIT_COST with four decimals and its unit are c/kWh.

Note that the units must be consistent. EXERGY and EXERGY_COST must have the same units, and EXERGY_UNIT_COST must be dimensionless. EXERGY and GENERALIZED_COST must be consistent with GENERALIZED_UNIT_COST.

DIAGNOSIS indicate the values of diagnosis tables, and the unit must be the same as EXERGY.

WasteDefinition

It is optional. This sheet contains information on how thermoeconomic analysis algorithms manage waste cost allocation. If the sheet is omitted, the default algorithm is used. If the data model has not waste defined, this sheet is not processed.

Fields

flow – key of the waste flow. This flow must be defined as waste in *Flows* sheet. If a waste flow is not included in this sheet, the default algorithm is used for this flow.

type – type of algorithm used for waste cost allocation. There is following waste cost allocation algorithm defined:

- RESOURCE: Allocate the waste cost to the resource flows. It is the default algorithm.
- EXERGY: Allocate the waste cost to the process generated proportionally to its exergy.
- COST: The same that the previous method but proportionally to its direct exergy cost
- IRREVERSIBILITY: Allocate the waste cost to the processes proportionally to the contribution of its irreversibilities to the waste cost.
- HYBRID: Combined EXERGY and IRREVERSIBILITY methods
- MANUAL: Waste cost allocation values are provided in the sheet *WasteAllocation*

recycle – Number between 0 and 1, indicating the portion of waste recycled. If it is not provided, a zero value is used.

Table 2.5.- *WasteDefinition* sheet for the data model file rorc_model.xlsx.

flow	type	recycle
QCND	IRREVERSIBILITY	0

This plant has a waste flow QCND (heat dissipated in the condenser). The values of the Irreversibility-Cost table for this flow are used to allocate its cost to the productive processes of the plant. In this case, no waste is recycled. These fields could be modified for each state using the ThermoeconomicTool functions.

WasteAllocation

This table must be provided if the manual type is chosen in the table above for some of the waste streams. It is a double-entry table Process/WasteFlow, where we indicate, in each cell, the waste stream cost that is allocated to a process.

The table has as many rows as productive processes and as many columns as waste flows.

Fields

key – key of the process where the waste cost is allocated.

<waste flow> - Indicate the portion of waste cost allocated to each process. The name of each column must be the same that the waste flow key.

Table 2.6.- *WasteAllocation* sheet for the data model file rorc_model.xlsx.

key	QCND
BLR	93.15
TRB	2.57
IHE	2.64
PMP	1.64

ResourcesCost sheet

This sheet contains the resource cost values of the plant: Resources Flows and Processes. We can define one or several samples. It is a double-entry table.Key/Sample, where we indicate, in each cell, the resource cost for each sample defined.

Fields

key – Name by which the resources is identified. It could be a flow of type RESOURCE or a process.

type – Indicates if the key is a FLOW or a PROCESS

<sample> - Each column contains the resource cost values. If it is a resource FLOW the units must be indicated in the *Format* sheet as GENERALIZED_UNIT_COST. If it is PROCESS the units must be indicated in the *Format* sheet as GENERALISED_ COST.

<sample> is a string of characters without blanks used as column name which identifies it. The length of this string shall not exceed eight characters. It is used in the application as ResourceSample property.

Table 2.7. *ResourcesCost* sheet for the data model file rorc_model.xlsx.

key	type	Internal	External
QBLR	FLOW	0.00	1.5
BLR	PROCESS	6.51	6.51
TRB	PROCESS	151.75	151.75
IHE	PROCESS	2.56	2.56
PMP	PROCESS	2.11	2.11
CND	PROCESS	10.75	10.75

In this example, we have two resource cost samples: Internal and External. In the first sample, the cost of waste heat used in the boiler is zero because it is produced in another part of the plant. In the second sample, the waste heat cost is 1.5 c€/kWh, considering that it is purchased from an external supplier. The processes' investment, maintenance and operation cost are the same in both cases and expressed in c€/h.

3.2 CSV format

The data model could be created using comma-separated value format (CSV) as an alternative to Excel spreadsheet. The tabular structure is the same as Excel. The data model consists of a file with the CSV extension and a folder containing the files, one per table defined. The name of the files must be the same as the Excel sheets. The file only includes the name of the folder.

3.3 Structured text format

The data model of a plant can be stored in text files with structured data formats such as JSON or XML. In the folder Examples/Templates are the JSON data schema files: plant_model.json, where the data's structure, format and validation are defined.

There are specialised text editors, such as Visual Studio Code, which simplify the editing and validation of this type of file.

Structured files are usually more efficient to read than tabular files, although they can be more complex to edit. TaesLab provides the SaveDataModel function that allows saving the tabular data model in JSON or XML format.

3.4 MATLAB files

Finally, the data model obtained with the *ReadDataModel* or *CheckDataModel* functions can be saved in MATLAB internal format (mat files) for subsequent work sessions.

These files can also be obtained from the *TaesLab* and *ThermoeconomicPanel* applications or the interactive *ThermoeconomicTool*, using the *SaveDataModel* function with the MAT extension.

These files can be used as data model files, like the other file types, XLSX or JSON. They can also be loaded as a *cReadModel* object with the standard MATLAB *load* function.

4 Base Functions

TaesLab provides a set of functions to make the thermoeconomic analysis of a plant, and to save the results, for further analysis.

4.1 Read Data Model Functions

CheckDataModel

This function reads the data model file and checks if it is correct. Perform the following checks:

- Check if all required information is available.
- Check the consistency of flows and processes key names.
- Check if the productive structure is correct, including graph connectivity.
- Check if the exergy balances are correct for all defined states.
- Check if the format definition is correct.
- Check if the waste definition and allocation are correct.
- Check if the resource costs are correct.

Usage:

```
data = CheckDataModel(filename);
```

Input Arguments:

filename - Name of the file with the data model information. It admits the following file extension types: XLSX, CSV, JSON, XML, MAT

Output Argument:

data – cReadModel object containing all the information of the data model.

Example:

```
data = CheckDataModel('rorc_model.xlsx');
```

```
INFO: cStatusLogger. Productive Structure is valid
INFO: cStatusLogger. Format Configuration is valid
INFO: cStatusLogger. Exergy values [TCND30] are valid
INFO: cStatusLogger. Exergy values [TCND45] are valid
INFO: cStatusLogger. Exergy values [ETAT75] are valid
INFO: cStatusLogger. Exergy values [PBLR15] are valid
INFO: cStatusLogger. Exergy values [NoIHE] are valid
INFO: cStatusLogger. Exergy values [N_Butane] are valid
INFO: cStatusLogger. Resources Cost sample [Base] is valid
INFO: cStatusLogger. Waste definition is valid
INFO: cReadModelXLS. Data Model rorc_model.xlsx is valid
```

ReadDataModel

This function reads the file containing the data model, but does not make a complete check, only:

- Check if all required information is available.
- Check the consistency of flows and processes key names.
- Check if the productive structure is correct, including graph connectivity.

The function does not show any message if the data model is correct. If it has some error, it is shown in the console.

Usage:

```
data = ReadDataModel(filename);
```

Input Arguments:

filename - Name of the file with the data model information. It admit the following file extension types: *.xlsx, *.csv, *.json, *.xml, *.mat

Output Argument:

data – cReadModel object containing all the information of the data model.

Example:

Read the data model file `rorc_model.xlsx`, and display the information:

```
data=ReadDataModel('rorc_model.xlsx');
disp(data)
```

```
           NrOfFlows: 10
       NrOfProcesses: 5
          NrOfWastes: 1
          NrOfStates: 6
              States: {'TCND30'  'TCND45'  'ETAT75'  'PBLR15'  'NoIHE'  'N_Butane'}
NrOfResourceSamples: 2
    ResourceSamples: {'Internal'  'External'}
             isWaste: 1
            isFormat: 1
     isResourceCost: 1
        isDiagnosis: 1
       isTableModel: 1
ProductiveStructure: [1×1 cProductiveStructure]
          ModelFile: '.\TaesLab\Examples\rorc\rorc_model.xlsx'
          ModelName: 'rorc_model'
          ModelData: [1×1 cModelData]
             status: 1
```

4.2 Thermoeconomic Analysis Functions

These functions make different kinds of thermoeconomic analysis and return an object with the results. These results could be viewed and saved in different ways, using the cResultInfo functions explained in the next sections.

ProductiveStructure

Give the information and tables of the productive structure of the plant.

Usage:

```
res = ProductiveStructure(data);
```

Input Arguments:

data – cReadModel object with the data model

Output Argument:

res – *cResultInfo* object containing all the information related to the Productive Structure of the system (`cType.ResultId.PRODUCTIVE_STRUCTURE`)

Example:

Print the Productive Structure tables of the plant

```
res=ProductiveStructure(data);
printResults(res);
```

```
Flows Table

  Id  Key   From    To      Type
-------------------------------------
   1  B1    BLR_P1  TRB_F1  INTERNAL
   2  B2    TRB_F1  IHE_F1  INTERNAL
   3  B3    IHE_F1  CND_F1  INTERNAL
   4  B4    CND_F1  PMP_P1  INTERNAL
   5  B5    PMP_P1  IHE_P1  INTERNAL
   6  B6    IHE_P1  BLR_P1  INTERNAL
   7  QBLR  ENV_R1  BLR_F1  RESOURCE
   8  WP    TRB_P1  PMP_F1  INTERNAL
   9  WN    TRB_P1  ENV_O1  OUTPUT
  10  QCND  CND_P1  ENV_W1  WASTE

Processes Table

  Id  Key  Description              Fuel   Product  Type
-----------------------------------------------------------------
   1  BLR  Steam Generator          QBLR   B1-B6    PRODUCTIVE
   2  TRB  Turbine                  B1-B2  WN+WP    PRODUCTIVE
   3  IHE  Internal Heat Exchanger  B2-B3  B6-B5    PRODUCTIVE
   4  PMP  Pump                     WP     B5-B4    PRODUCTIVE
   5  CND  Condenser                B3-B4  QCND     DISSIPATIVE
```

ProductiveDiagram

Give the information and tables of the productive diagram of the plant. The productive diagram info contains the adjacency tables of the productive structure. This information could be saved in Excel format and used with Graph software like yED to obtain the diagrams.

Usage:

```
res = ProductiveDiagram(data);
```

Input Arguments:

data – cReadModel object with the data model

Output Argument:

res – *cResultInfo* object containing all the information related to the Productive Structure of the system (`cType.ResultId.PRODUCTIVE_DIAGRAM`)

Example:

Export the productive adjacency tables into an Excel spreadsheet.

```
res=ProductiveDiagram(data);
SaveResults(res);
```

```
INFO: cStatusLogger. Productive Diagram available in file rorc_diagram.xlsx
```

ThermoeconomicState

Give the information and the tables about the exergy model for and the state of the plant

Usage:

```
res = ThermoeconomicState(data, state);
```

Input Arguments:

data – cReadModel object with the data model

State – Vector of characters containing the state's name to show the info.

Output Argument:

res – *cResultInfo* object containing all the information related to the Thermoeconomic State of the system (`cType.ResultId.THERMOECONOMIC_STATE`)

Example:

Print the process exergy table, for the State *TCND30*

```
res=ThermoeconomicState(data, 'TCND30');
res.printTable(cType.Tables.EXERGY_PROCESSES);
```

```
Processes Exergy Table

 Key       F(kW)      P(kW)      I(kW)     k(J/J)
-------------------------------------------------
 BLR      91.790     72.150     19.640     1.2722
 TRB      64.290     52.381     11.909     1.2274
 IHE       2.030      1.473      0.557     1.3781
 PMP       2.381      2.036      0.345     1.1694
 CND       9.339      9.339      0.000     1.0000
 ENV      91.790     50.000     41.790     1.8358
```

DiagramFP

Give the information and the tables about the Diagram FP

Usage:

```
res = DiagramFP(data, options);
```

Input Arguments:

data – cReadModel object with the data model

options – pairs of Name-Value arguments with the function options

Options:

State - Character vector specifying the state's name for which we want to make the calculations. This name must be defined as *State* in the data model. If the parameter is not specified *State* takes the value of the first State value.

Table – Character vector specifying the type of FP Table to obtain:

cType.Tables.TABLE_FP

cTypes.Tables.COST_TABLE_FP

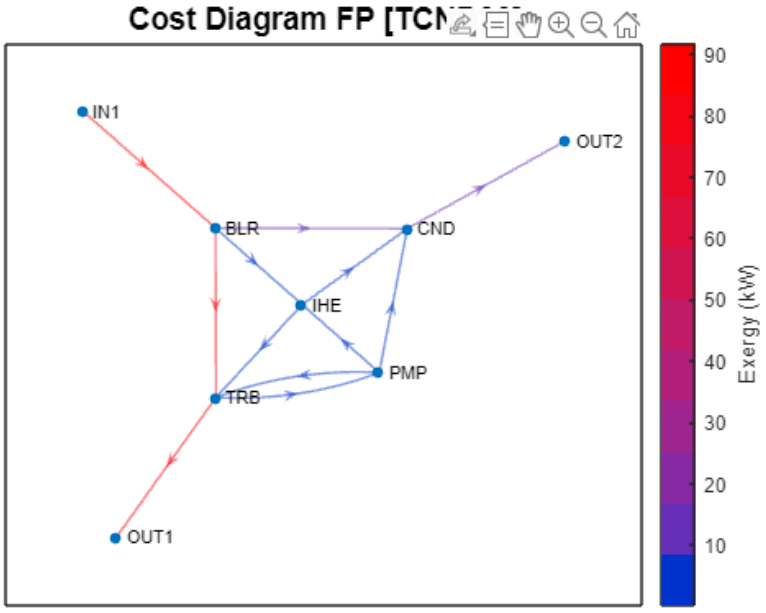
Output Argument:

res – *cResultInfo* object containing all the information related to Diagram FP of the system (`cType.ResultId.DIAGRAM_FP`)

Example:

Plot the Cost Diagram FP for the reference State

```
res=DiagramFP(data,'Table',cType.Tables.COST_TABLE_FP);
res.showDiagramFP;
```



ExergyCostCalculator

Give the information and the tables about the exergy cost for the current state of the plant. This function uses the Structural Theory algorithms.

Usage:

```
res = ExergyCostCalculator(data, options);
```

Input Arguments:

data – cReadModel object with the data model

options – pairs of Name-Value arguments with the function options

Options:

State - Character vector specifying the state's name, for which we want to make the calculations. This name must be defined as *State* in the data model. If the parameter is not specified *State* takes the value of the first State value.

ResourceSample: character vector specifying the name of the sample for which we want to make the calculations. This name must be defined as *ResourceSample* in the data model. If the parameter is not specified *ResourceSample* takes the value of the first ResourceSample value

CostTable: character vector specifying the type of tables we like to obtain in a thermoeconomic analysis. The valid values are: `'DIRECT' | 'GENERALIZED' | 'ALL'.` If the parameter is not specified, `'DIRECT'` is chosen.

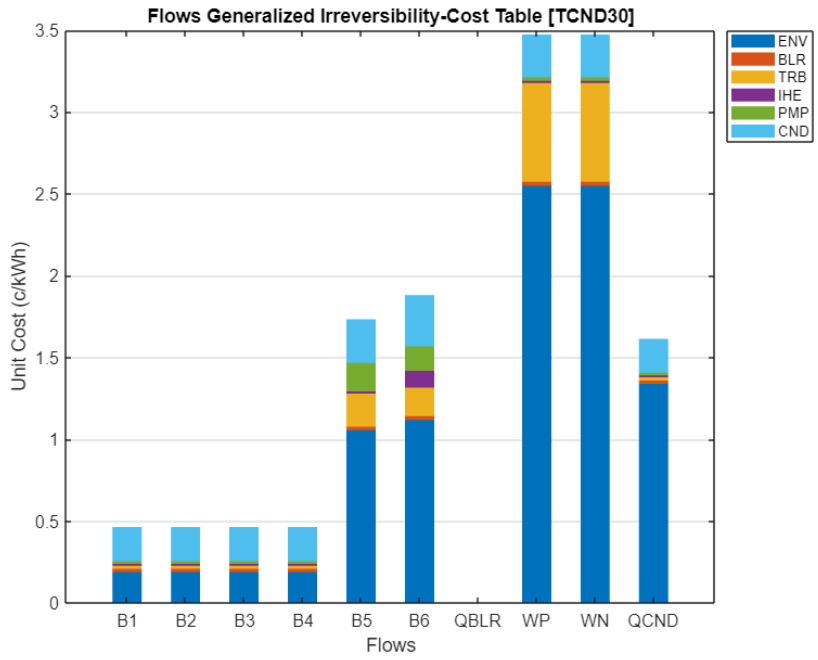
Output Argument:

res – *cResultInfo* object containing all the information and tables related to the exergy cost model `(cType.ResultId.EXERGY_COST)`

Example:

Shows the generalised irreversibility-cost graph for the flows of the reference state of the system

```
res=ExergyCostCalculator(data,'CostTables','ALL');
res.graphCost(cType.Tables.FLOW_GENERALIZED_ICT);
```



ThermoeconomicAnalysis

Give the information and the tables about the exergy cost for a state of the plant. This function uses the Model FPR algorithms.

Usage:

```
res = ThermoeconomicAnalysis(data, options);
```

Input Arguments:

data – cReadModel object with the data model

options – pairs of Name-Value arguments with the function options

Options:

State - Character vector specifying the state's name for which we want to make the calculations. This name must be defined as *State* in the data model. If the parameter is not specified *State* takes the value of the first State value.

ResourceSample: character vector specifying the name of the sample for which we want to make the calculations. This name must be defined as *ResourceSample* in the data model. If the parameter is not specified *ResourceSample* takes the value of the first ResourceSample value

CostTable: character vector specifying the type of tables we like to obtain in a thermoeconomic analysis. The valid values are: `'DIRECT' | 'GENERALIZED' | 'ALL'.` If the parameter is not specified, `'DIRECT'` is chosen.

Output Argument:

res – *cResultInfo* object containing all the information and tables related to the exergy cost model `(cType.ResultId.THERMOECONOMIC_ANALYSIS)`

Example:

Print generalised cost Fuel-Product Table for the State noIHE

```
res=ThermoeconomicAnalysis(data,'CostTables','ALL','State','NoIHE');
res.printTable(cType.Tables.GENERALIZED_COST_TABLE_FP);
```

```
FP Generalized Exergy Cost Table (c/h)

 Key          BLR       TRB       IHE       PMP       CND       ENV     Total
-------------------------------------------------------------------------------
 BLR        0.000    18.292     0.000     0.000     3.235     0.000    21.527
 TRB        0.000     0.000     0.000     8.149     0.000   171.120   179.269
 IHE        0.000     0.000     0.000     0.000     0.000     0.000     0.000
 PMP        0.000     8.899     0.000     0.000     1.574     0.000    10.473
 CND       15.016     0.330     0.000     0.210     0.000     0.000    15.555
 ENV        6.511   151.747     0.000     2.115    10.747     0.000   171.120
-------------------------------------------------------------------------------
 Total     21.527   179.269     0.000    10.473    15.555   171.120
```

ThermoeconomicDiagnosis

Compare two plant states through thermoeconomic diagnosis, and return the corresponding tables and info.

Usage:

```
res = ThermoeconomicDiagnosis(data, options);
```

Input Arguments:

data – cReadModel object with the data model

options – pairs of Name-Value arguments with the function options

Options:

State - Character vector specifying the state's name, for which we want to make the calculations. This name must be defined as *State* in the data model. State takes the second value on the state's list if the parameter is not specified.

ReferenceState: character vector specifying the state's name used as *Reference* in a thermoeconomic diagnosis. This name must be defined as *State* in the data model. If the parameter is not specified *ReferenceState* takes the value of the first State value.

DiagnosisMethod: character vector specifying the method applied in a thermoeconomic diagnosis. The valid values are: 'NONE' | 'WASTE_OUPUT' | 'WASTE_INTERNAL'. If the parameter is not specified or the system has no wastes' WASTE_OUTPUT' is selected. If NONE is chosen, no diagnosis is performed.

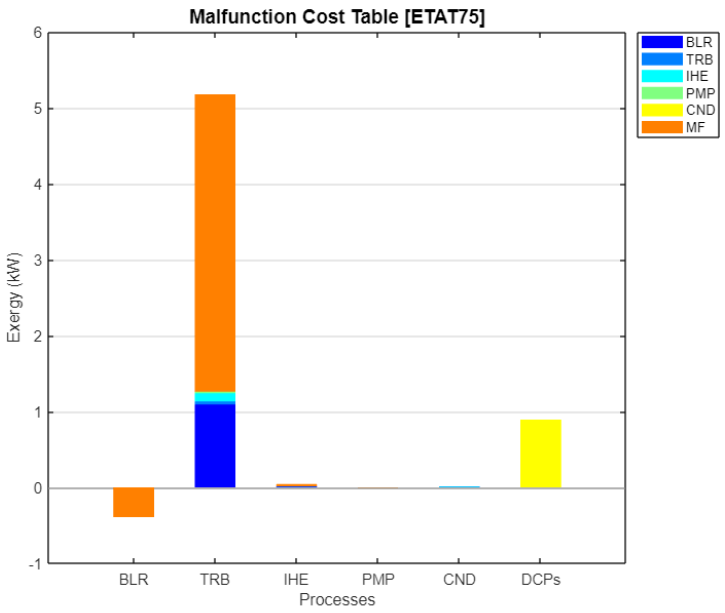
Output Argument:

res – *cResultInfo* object containing all the information and tables related to the thermoeconomic diagnosis `(cType.ResultId.THERMOECONOMIC_ANALYSIS)`

Example:

Compare the reference and ETAT75 states, print the diagnosis summary and show malfunction cost graph. The default method WASTE_OUTPUT is used,

```
res=ThermoeconomicDiagnosis(data,'State','ETAT75');
res.graphDiagnosis(cType.Tables.MALFUNCTION_COST);
res.fuelImpact;
```




```
Fuel Impact:    5.7300 (kW)
```

ResultsSummary

Obtain the summary tables of a thermoeconomic model comparing the costs of the different states of the plant.

Usage:

```
res = RecyclingAnalysis(data, options);
```

Input Arguments:

data – cReadModel object with the data model

Output Argument:

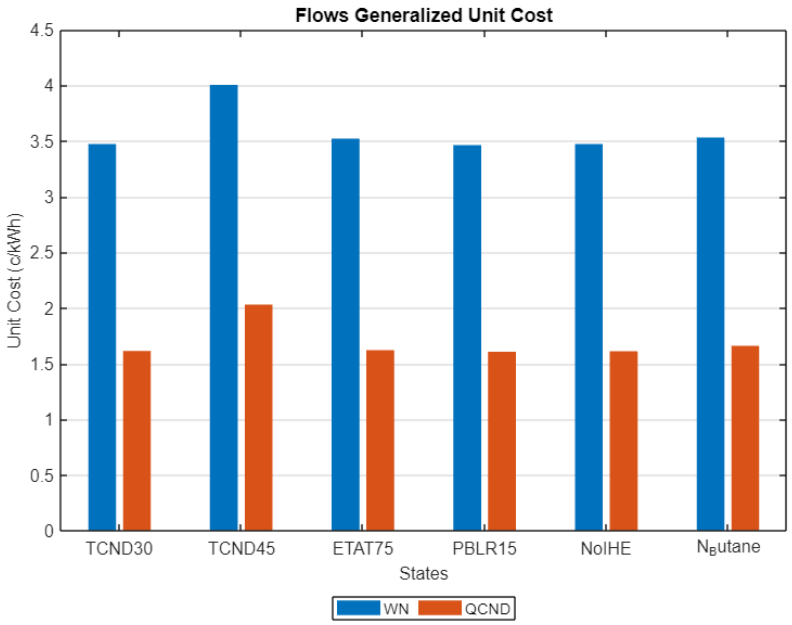
res – cResultInfo object containing all the summary results and tables

Example:

Get the unit consumption of the processes for all the states of the plant, and plot a comparative graph of the generalised unit cost of the system flows

```
res=SummaryResults(data);
res.printResults(cType.SummaryTables.UNIT_CONSUMPTION)
res.graphSummary(cType.SummaryTables.FLOW_GENERAL_UNIT_COST);
```

```
Process Unit Consumption (J/J)

 Key      TCND30     TCND45     ETAT75     PBLR15      NoIHE   N_Butane
-------------------------------------------------------------------------
 BLR      1.2722     1.2234     1.2668     1.2810     1.3350     1.2656
 TRB      1.2274     1.2195     1.3021     1.2288     1.2274     1.2266
 IHE      1.3781     1.1558     1.3991     1.3581     1.0000     1.3418
 PMP      1.1694     1.1618     1.1689     1.1691     1.1694     1.1710
 CND      1.0000     1.0000     1.0000     1.0000     1.0000     1.0000
```



RecyclingAnalysis

Analyse the cost saving of a system state when recycling a waste flow. Calculates the cost of output flows (final products and wastes) when recycled from 0 to %100, and returns the corresponding tables and information.

Usage:

```
res = RecyclingAnalysis(data, options);
```

Input Arguments:

data – cReadModel object with the data model

options – pairs of Name-Value arguments with the function options

Options:

WasteFlow - Character vector specifying the waste flow key for which we want to make the calculations. This name must be defined as a waste flow. If the parameter is not specified, *WasteFlow* takes the name of the first waste flow defined in the Flows table.

State - Character vector specifying the state's name for which we want to make the calculations. This name must be defined as *State* in the data model. If the parameter is not specified *State* takes the value of the first State value.

ResourceSample: character vector specifying the sample name for which we want to make the calculations. This name must be defined as *ResourceSample* in the data model. If the parameter is not specified *ResourceSample* takes the value of the first ResourceSample value

CostTable: character vector specifying the type of tables we like to obtain in a thermoeconomic analysis. The valid values are: `'DIRECT' | 'GENERALIZED' | 'ALL'`. If the parameter is not specified, `'DIRECT'` is chosen.

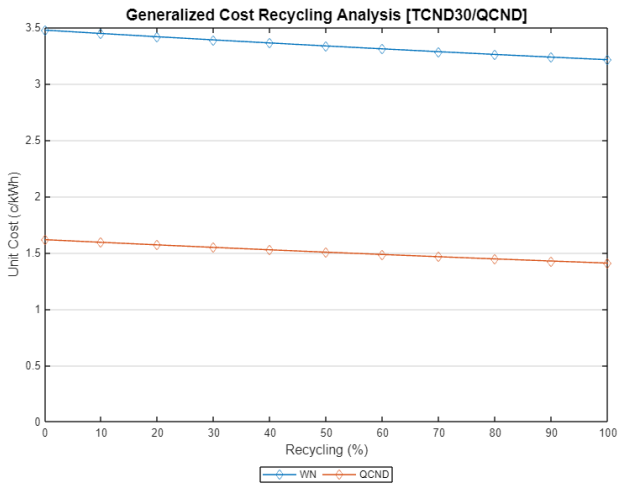
Output Argument:

res – *cResultInfo* object containing all the information and tables related to the thermoeconomic diagnosis `(cType.ResultId.RECYCLING_ANALYSIS)`

Example:

Get the *Recycling Analysis* result info and plot the generalised cost.

```
param.CostTables='ALL';
param.WasteFlow='QCND';
res=RecyclingAnalysis(data,param);
res.graphRecycling (cType.Tables.WASTE_RECYCLING_GENERAL);
```



4.3 Save Results Functions

Depending on the file extension, these functions save the model's results into files.

The valid file extensions are:

- xlsx – Excel format. One table per sheet
- csv – Comma-Separated Values. Create a folder with the file's name, and save a file per table.
- txt – Save all the tables in a text file in the same format that appears in the console.
- mat – Save the *cResultInfo* object in a file, which can be restored with the load command.

These functions show a log message indicating the status of file saving. If this message is not needed, use the corresponding functions of the *cResultInfo* or *cThermoeconomicModel* objects

SaveResults

Save the tables of a cResultInfo object into a file.

Usage:

```
SaveResults(res, filename)
```

Input Parameters:

res – cResultInfo object to save.

filename – char array containing the filename with extension.

SaveResultsModel

Save all the tables of a thermoeconomic model for the current state. (See ThermoeconomicTool)

Usage:

```
SaveResultsModel(model, filename)
```

Input Parameters:

model – cThermoeconomicModel object.

filename – char array containing the filename with extension.

SaveSummaryResults

Save the summary tables of a thermoeconomic model.

Usage:

```
SaveSummaryResults(model, filename)
```

Input Parameters:

model – cThermoeconomicModel object.

filename – char array containing the filename with extension.

SaveProductiveDiagram

Save the productive diagram adjacency tables of the model.

Usage:

```
SaveDiagramFP(model, filename)
```

Input Parameters:

model – cThermoeconomicModel object.

filename – char array containing the filename with extension.

SaveDiagramFP

Save the diagram FP adjacency tables for the current state of the model.

Usage:

```
SaveDiagramFP(model, filename, option)
```

Input Parameters:

model – cThermoeconomicModel object.

filename – char array containing the filename with extension.

option – Optional parameter indicating the type of FP table.

```
cType.Tables.TABLE_FP (default)
cType.Tables.COST_TABLE_FP
```

SaveDataModel

Save the data model tables. This function permits to save the data model file in different formats. In this case JSON and XML are also valid extensions.

Usage:

```
SaveDataModel(data, filename);
```

Input Parameters:

data – cReadModel or cThermoeconomicModel objects containing the data model

filename – char array with the name of the file.

4.4 LiveScripts Demos

The LiveScripts folder contains demos as a tutorial to explain the use and functionality of TaesLab's base functions. It Includes:

- ProductiveStructureDemo
- ThermoeconomicStateDemo
- ThermoeconomicAnalysisDemo
- ThermoeconomicDiagnosisDemo
- ExergyCostCalculatorDemo
- RecyclingAnalysisDemo
- WasteAnalysisDemo
- ShowDiagramFpDemo
- ThermoeconomicToolDemo.

4.5 Examples

The full MATLAB and Octave distributions include a folder with examples of simple thermodynamic cycles that are easy to use with the base functions, applications, and scripts.

Each folder contains the thermoeconomic model data of the plant in XLSX and JSON format, as well as the thermodynamic data of the different states or simulations considered.

A Templates folder is also included with a template of the XLSX file and the schemas for JSON and XML.

The examples included are:

- cgam: Gas turbine cycle with a heat recovery steam generator for cogeneration. It includes economic data. This example is used in the TaesLab application guide.

- cicloComb: Combined cycle for electricity production. It includes economic data.

- gturbo: Gas turbine cycle with several stages. The heat from the gases can be used (recycled) in other systems.

- orcvcr: Organic Rankine cycle whose electrical output feeds the refrigeration cycle compressor. It includes economic data.

- rorc: Organic Rankine Cycle used in the examples in the manual.

- powerplant: Water vapour Rankine cycle, with several turbine bodies and heaters.

5 Result Information

Most of the Base Functions of the TaesLab package return a *cResultInfo* object containing the results of the different calculations performed.

The *cResultInfo* is an object container which stores the result tables and the calculation object with internal information as the production operator, characteristic matrix …

```
res=ProductiveStructure(data);
display(res)
```

```
cResultInfo with properties:
          Info: [1×1 cProductiveStructure]
        Tables: [1×1 struct]
    NrOfTables: 3
     ModelName: 'rorc_model'
         State: 'SUMMARY'
      ResultId: 1
    ResultName: 'Productive Structure'
        status: 1
```

5.1 Tables

We will show all the available tables on TaesLab group by its *ResultId,* Indicating if it had a graph associated.

Each table has an identifier (Id), defined as `cType.Tables.Id`

Productive Structure: `cType.ResultId.PRODUCTIVE_STRUCTURE`

Id	Name	Description	Graph
FLOW_TABLE	flows	Flows Table	
PROCESS_TABLE	processes	Processes Table	
STREAM_TABLE	streams	Streams Table	

Thermoeconomic State: `cType.ResultId.THERMOECONOMIC_STATE`

Id	Name	Description	Graph
FLOW_EXERGY	eflows	Flows Exergy Table	
STREAM_EXERGY	estreams	Streams Exergy Table	
PROCESS_EXERGY	eprocesses	Processes Exergy Table	
TABLE_FP	tfp	FP Table	✓

Thermoeconomic Analysis: `cType.ResultId.THERMOECONOMIC_ANALYSIS`

Id	Name	Description	
PROCESS_COST	dcost	Process Exergy Cost	
PROCESS_UNIT_COST	ducost	Process Unit Exergy Cost	
FLOW_COST	dfcost	Flows Exergy Cost	
COST_TABLE_FP	dcfp	FP Exergy Cost Table	✓
COST_TABLE_FPR	dcfpr	FPR Exergy Cost Table	
PROCESS_ICT	dict	Irreversibility-Cost Table	✓
FLOW_ICT	dfict	Flows Irreversibility-Cost Table	✓
WASTE_ALLOCATION	wa	Waste Allocation Table	✓
PROCESS_GENERAL_COST	gcost	Process Generalised Cost	
PROCESS_GENERAL_UNIT_COST	gucost	Process Generalised Unit Cost	
FLOW_GENERAL_COST	gfcost	Flows Generalised Cost	
PROCESS_GENERAL_ICT	gict	Generalised Irreversibility-Cost Table	✓
FLOW_GENERAL_ICT	gfict	Flows Generalised Irreversibility-Cost Table	✓
GENERAL_COST_TABLE_FP	gcfp	FP Generalized Exergy Cost Table	

Thermoeconomic Diagnosis: `cType.ResultId.THERMOECONOMIC_Diagnosis`

Id	Name	Description	Graph
DIAGNOSIS	dgn	Diagnosis Summary	
MALFUNCTION	mf	Malfunction Table	✓
MALFUNCTION_COST	mfc	Malfunction Cost Table	✓
IRREVERSIBILITY_VARIATION	dit	Irreversibility Variation Table	✓

ExergyCostCalculator: `cType.ResultId.EXERGY_COST_CALCULATOR`

Id	Name	Description	Graph
PROCESS_COST	dcost	Process Exergy Cost	
PROCESS_UNIT_COST	ducost	Process Unit Exergy Cost	
FLOW_COST	dfcost	Flows Exergy Cost	
PROCESS_ICT	dict	Irreversibility-Cost Table	✓
FLOW_ICT	dfict	Flows Irreversibility-Cost Table	✓
WASTE_ALLOCATION	wa	Waste Allocation Table	
PROCESS_GENERAL_COST	gcost	Process Generalised Cost	
PROCESS_GENERAL_UNIT_COST	gucost	Process Generalised Unit Cost	
FLOW_GENERAL_COST	gfcost	Flows Generalised Cost	
PROCESS_GENERAL_ICT	gict	Generalised Irreversibility-Cost Table	✓
FLOW_GENERAL_ICT	gfict	Flows Generalised Irreversibility-Cost Table	✓

Waste Analysis: `cType.ResultId.WASTE_ANALYSIS`

Id	Name	Description	Graph
WASTE_DEFINITION	wd	Waste Definition Table	✓
WASTE_ALLOCATION	wa	Waste Allocation Table	✓

Diagram FP: `cType.ResultId.DIAGRAM_FP`

Id	Name	Description	Graph
TABLE_FP	tfp	Table FP	✓
DIAGRAM_FP	atfp	Adjacency table FP	✓

Productive Diagram: `cType.ResultId.PRODUCTIVE_DIAGRAM`

Id	Name	Description	Graph
FLOW_DIAGRAM	tfp	Flows Adjacency Table	✓
PRODUCTIVE_DIAGRAM	atfp	Productive Adjacency Table	✓

Recycling Analysis: `cType.ResultId.RECYCLING_ANALYSIS`

Id	Name	Description	Graph
WASTE_RECYCLED_DIRECT	rad	Recycled Analysis Direct Cost	✓
WASTE_RECYCLED_GENERAL	rad	Recycled Analysis Generalised Cost	✓

Summary Results: `cType.ResultId.SUMMARY_RESULTS`

Id	Name	Description	Graph
EXERGY	exergy	Flows Exergy	
UNIT_CONSUMPTION	pku	Process Unit Consumption	✓
PROCESS_COST	dpc	Processes Direct Exergy Cost	
PROCESS_UNIT_COST	dpuc	Processes Direct Unit Exergy Cost	✓
FLOW_COST	dfc	Flows Direct Exergy Cost	
FLOW_UNIT_COST	dfuc	Flows Direct Unit Exergy Cost	✓
PROCESS_GENERAL_COST	gpc	Processes Generalised Cost	
PROCESS_GENERAL_UNIT_COST	gpuc	Processes Generalised Unit Cost	✓
FLOW_GENERAL_COST	gfc	Flows Generalised Cost	
FLOW_GENERAL_UNIT_COST	gfuc	Flows Generalised Unit Cost	✓

5.2 Functions

The following functions apply to the cResultInfo objects:

printResults

Print on console the tables of a cResultInfo object

Usage:
```
printResults(res)
```

Input Parameters:

res – cResultInfo object

printTable

Print on console a table of a cResultInfo object

Usage:
```
printTable(res, table)
```

Input Parameters:

res – cResultInfo object

table – table name char array

printIndexTable

Print on console the index table of a cResultInfo object

Usage:
```
printIndexTable(res)
```

Input Parameters:

res – cResultInfo object

viewTable

view a table of a cResultInfo object, in a iutable GUI

Usage:
```
viewTable(res, table)
```

Input Parameters:

res – cResultInfo object

table – table name char array

saveResults

Save into a file the tables of a cResultInfo object

Usage:
```
saveResults(res, filename)
```

Input Parameters:

res – cResultInfo object

filename – file name char array

getResultTables

get a structure with the tables of the cResultInfo

Usage:
```
sol = getResultTables(res, mode, format)
```

Input Parameters:

res –cResultInfo object

mode – Select the output mode. It could take the following values:

`cType.VarMode.NONE`: Return a struct with the cTable objects

`cType.VarMode.CELL`: Return a struct with cell values

`cType.VarMode.STRUCT`: Return a struct with structured array values

`cType.VarModel.TABLE`: Return a struct of Matlab tables

format – true/false value, indicating if the table format is applied to the output. By default, the value is false.

Output Parameters:

sol – Result tables in the selected format

fuelImpact

Print on console the fuel Impact value and units of a diagnosis analysis

Usage:
```
fuelImpact(res)
```

Input Parameters:

res – Thermoeconomic Diagnosis *cResultInfo*

graphCost

Show a barplot with the irreversibility cost table values of a given state.

Usage:
```
graphCost(res, table)
```

Input Parameters:

res – Thermoeconomic Analysis or Exergy Cost Calculator *cResultInfo*
table – table name char array. If it is omitted, the values of table `cType.Tables.PROCESS_ICT` are shown

graphDiagnosis

Show a barplot with the diagnosis table values for a given state.

Usage:
```
graphDiagnosis(res, table)
```

Input Parameters:

res – Thermoeconomic Diagnosis *cResultInfo*
table – table name char array. If it is omitted, the values of table `cType.Tables.MALFUNCTION_COST` are shown

graphSummary

Show a barplot group with the summary table values for the given variables.

Usage:
```
graphSummary(res, table, var)
```

Input Parameters:

res – Summary Results *cResultInfo*

table – table name char array. If it is omitted cType.SummaryTables.FLOW_UNIT_COST is selected

var – Cell array containing the keys of the variables to plot. If it is a flow table, and *var* is omitted, then the output flows are shown. If it is a process table, the values are required.

graphRecycling

Show a plot with the unit cost of the output flows varying the recycling ratio from 0 to 100%.

Usage:
```
graphRecycling(res, table)
```

Input Parameters:

res – Recycling Analysis *cResultInfo*

table – table name char array, indicating if the table (direct or generalised cost). If it is omitted the values of the table cType.Tables.RECYCLING_DIRECT are shown

showDiagramFP

Show the Diagram FP graph (only Matlab)

Usage:
```
showDiagramFP(res)
```

Input Parameters:

res – *cResultInfo* object. It must be a DiagramFP, ThermoeconomomicState or ThermoeconomicDiagnosis Result Info.

showFlowsDiagram

Show the Flows Diagram graph (only Matlab)

Usage:
```
showFlowsDiagram(res)
```

Input Parameters:

res – Productive Diagram *cResultInfo*

6 ThermoeconomicTool

ThermoeconomicTool is an interactive tool that integrates all the thermoeconomic analysis functions explained above. It is designed to use in LiveScript or in the console.

By default, ThermoeconomicTool first makes the thermoeconomic analysis of the reference state.

If you want to analyse another state, use the *setState* function to define the current state

If the *Reference State* is different from the Operation State and the DiagnosisMethod is activated (different to NONE), then the current state's thermoeconomic diagnosis results are obtained.

6.1 Create the Thermoeconononomic Model

The ThermoeconomicTool function creates a *cThermoeconomicModel* object with all the information to make the thermoeconomic analysis of a plant given its data model.

Usage:

```
model = ThermoeconomicTool(filename, properties);
```

Input Arguments:

filename - Name of the file with the data model information. It admits the following file extension types: XLSX, CSV, JSON, XML and MAT

properties – pairs of Name-Value arguments or structure with the property values of the model.

Output Argument:

model – cThermoeconomicModel object containing all the information required to perform a complete thermoeconomic analysis of the plant.

Properties

State - Character vector specifying the state's name, for which we want to make the calculations. This name must be defined as <state> in the data model. If the parameter is not specified *State* takes the value of the first state name.

ReferenceState: character vector specifying the name of the state used as *Reference* in a thermoeconomic diagnosis. This name must be defined as *State* in the data model. If the parameter is not specified *ReferenceState* takes the value of the first *State* value.

ResourceSample: character vector specifying the name of the sample for which we want to make the calculations. This name must be defined as *ResourceSample* in the data model. If the parameter is not specified *ResourceSample* takes the value of the first ResourceSample value

CostTable: character vector specifying the type of tables we like to obtain in a thermoeconomic analysis. The valid values are: `DIRECT | GENERALISED | ALL.` If the parameter is not specified `'DIRECT'` is chosen.

DiagnosisMethod: character vector specifying the type of method will be applied in a thermoeconomic diagnosis. The valid values are `NONE | WASTE_OUPUT | WASTE_INTERNAL`. If the parameter is not specified or the system has no waste `WASTE_OUTPUT` is selected. If `NONE` is chosen, no diagnosis is performed.

Summary: `true | false (default)`. Activate the generation of the Summary Results.

Debug: `true (default) | false`. Activate the debug information.

ThermoeconomicTool function returns a cThermoeconomicModel object which contains the model results for and state and the summary results. After the *model* object is created, more information and additional actions can get by using the object functions described below.

6.2 Set Functions

Let to assign values to the ThermeconomicTool properties.

setState

Define the current state of the model object and make the calculations.

Usage:

```
model.setState(value)
model.State=value;
```

Input Arguments:

value – char array with a valid state name

setReferenceState

Define the reference state of the model.

Usage:

```
model.setReferenceState(value)
model.ReferenceState=value;
```

Input Arguments:

value – char array with a valid state name

setResourceSample

Define the current resource sample for thermoeconomic analysis calculations.

Usage:

```
model.setResourceSample(value)
model.ResourceSample=value;
```

Input Arguments:

value – char array, with a valid resource sample name

setCostTables

Define the cost tables that will be obtained for thermoeconomic analysis calculations.

Usage:

```
model.setCostTables(value)
model.CostTables=value;
```

Input Arguments:

value – char array, with one of these values: `DIRECT | GENERALIZED | ALL`

setDiagnosisMethod

Define the diagnosis method that will be used in thermoeconomic diagnosis calculations.

Usage:

```
model.setDiagnosis(value)
model.DiagnosisMethod=value;
```

Input Arguments:

value – char array, with one of these values: `NONE | WASTE_OUTPUT | WASTE_INTERNAL`

setSummary

Activate or deactivate the Summary Results calculations.

Usage:

```
model.setSummary(value)
model.Summary =value;
```

Input Arguments:

value – true | false

6.3 Model Info Functions

These functions get information about the model.

showProperties

Show on console the model properties values and other helpful information.

Usage:

```
model.showProperties;
```

getStateNames

Get a cell array with the state names of the plant

Usage:

```
res=model.getStateNames;
res=getStateNames(model);
```

getResourceSamples

Get a cell array with the resource sample names of the plant.

Usage:

```
res=model.getResourceSamples;
res=getResourceSamples(model);
```

isResourceCost

Indicate is the model have resource cost defined.

Usage:

```
res=model.isResourceCost;
```

isGeneralCost

Indicate if the model has the Generalized Cost activated.

Usage:

```
res=model.isGeneralCost;
```

isDiagnosis

Indicate if the model has the diagnosis activated.

Usage:

```
res=model.isDiagnosis;
```

isWaste

Indicate if the model has waste defined.

Usage:

```
res=model.isWaste;
```

getWasteFlows

Get a cell array with the defined waste flows.

Usage:

```
res=model.getWasteFlows;
res=getWasteFlows(model)
```

setDebug

Activate or deactivate the model debug information.

Usage:

```
model.setDebug(value)
```

6.4 Print Functions

Print the result tables on the console

printResults

Print the result tables on the current state.

Usage:

```
model.printResults
printResults(model);
```

printTable

Print a specific result table of the model on the current state

Usage:

```
model.printTable(table)
```

Input Parameters:

table – char array with the name of the table to show.

printSummary

Print the summary tables on the current state.

Usage:

```
model.printSummary(table)
```

Input Parameters:

table – char array with the table's name to show (optional). If it is omitted, all the summary tables are shown in the console.

```
model.printTable(table)
```

viewResultTable

Show in a uitable a specific result table of the model on the current state.

Usage:

```
model.viewResultTable(table)
```

Input Parameters:

table – char array with the name of the table to show.

viewSummaryTable

Show in a uitable a specific summary table of the model.

Usage:

```
model.viewResultTable(table)
```

Input Parameters:

table – char array with the name of the table to show.

6.5 Save Functions

Save the results into a file. File extensions XLSX, CSV and TXT are valid.

saveResultsModel

Save the results model tables of the current state.

Usage:

```
log=model.saveResultsModel(filename);
SaveResultsModel(model,filename);
```

Input Parameters:

filename– char array with name of the file.

saveSummary

Save the summary tables of the model.

Usage:

```
log=model.saveSummary(filename);
SaveSummary(model,filename);
```

Input Parameters:

filename– char array with name of the file.

saveDiagramFP

Save the Diagram FP tables of the model in the current state.

Usage:

```
log=model.saveDiagramFP(filename, option);
SaveDiagramFP(model,filename,option);
```

Input Parameters:

filename – char array with name of the file.

option – Optional parameter indicating the type of FP table.

```
cType.Tables.TABLE_FP (default)
cType.Tables.COST_TABLE_FP
```

saveProductiveDiagram

Save the Productive Diagram tables of the model.

Usage:

```
log=model.saveDiagramFP(filename);
SaveDiagramFP(model,filename);
```

Input Parameters:

filename– char array with the name of the file.

saveDataModel

Save the data model tables. XLSX, CSV, MAT, JSON and XML extensions are valid. This function permits to save the data model file in a different format.

Usage:

```
log=model.saveDataModel(filename);
SaveDataModel(model,filename);
```

Input Parameters:

filename– char array with the name of the file.

6.6 Graph Functions

Show the graph associated with a table.

graphCost

It shows a barplot with the values of the ICT tables.

Usage:

```
log=model.graphCost(table)
```

Input Parameters:

Table – char array with the name of the table to plot. If the value is not provided, the table: `cType.Tables.PROCESS_ict` is plot

graphDiagnosis

It shows a barplot with the values of the diagnosis tables.

Usage:

```
log=model.graphDiagnosis(table)
```

Input Parameters:

table – char array with the name of the table to plot. If the value is not provided,

the table: `cType.Tables.MALFUNCTION_COST` is plot

graphSummary

It shows a barplot with the values of the summary.

Usage:

```
log=model.graphSummary(table)
```

Input Parameters:

table – char array with the name of the table to plot. If the value is not provided, the table: `cType.Tables.FLOW_UNIT_COST` is plot

graphRecycling

It shows a graph of the output flows cost of the system depending on the recycled exergy of the selected waste.

Usage:

```
log=model.graphRecycling(wkey, table)
```

Input Parameters:

wkey – char array with the name of the waste flow to analyse.

table – char array with the name of the table to plot. If the value is not provided, the table `cType.Tables.WASTE_RECYCLED_DIRECT` is plotted.

graphWasteAllocation

Show a pie chart with the waste allocation values of the selected waste

Usage:

```
log=model.graphWasteAllocation(wkey)
```

Input Parameters:

wkey – char array with the name of the waste flow to analyse. If omitted, the first waste flow is selected

showDiagramFP

Show the diagram FP graph.

Usage:

```
log=model.showDiagramFP(table)
```

Input Parameters:

table – char array with the name of the table FP to graph. If omitted, the table: `cType.Tables.TABLE_FP` is plot

showFlowDiagram

Show the flow diagram graph of the model.

Usage:

```
log=model.showFlowDiagram
```

6.7 Waste Functions

Get and set waste parameters of the model

wasteAllocation

Show the waste definition and allocation tables on the console

Usage:

```
res=model.wasteAllocation;
```

Output Parameters

res – Waste Allocation cResultInfo with waste information.

setWasteType

Set the waste type of the selected waste flow, and recalculate the cost tables

Usage:

```
model.setWasteType(key, value);
```

Input Parameters

key – char array with the name of the waste flow

value – waste cost allocation type. Valid values:

```
cType.WasteAllocation.RESOURCE
cType.WasteAllocation.EXERGY
cType.WasteAllocation.COST
cType.WasteAllocation.IRREVERSIBILITY
cType.WasteAllocation.HYBRID
```

setWasteValues

Set the waste allocation values of the selected waste flow, and recalculate the cost tables

Usage:

```
model.setWasteValues(key, value);
```

Input Parameters

key – char array with the name of the waste flow

value – array with the allocation values of the processes:

setWasteRecycled

Set the waste recycled value of the selected waste flow, and recalculate the cost tables

Usage:

```
model.setWasteRecycled(key, value);
```

Input Parameters

key – char array with the name of the waste flow

value – value between 0 and 1:

6.8 Resources Functions

Get and set Resources Cost values.

getResourcesCost

Get the resource cost values.

Usage:

```
res = model.getResourcesCost;
```

Output Parameters

res – cReadResources object with the resources cost values

setFlowResources

Set the unit cost values of the flow resources.

Usage:

```
model.setFlowResources(value);
```

Input Parameters:

value – array with the resource unit cost values of the flows:

setProcessResources

Set the cost values of the process resources.

Usage:

```
model.setProcessResources(value);
```

Input Parameters

value – array with the resource cost values of the processes:

setResourceFlowValue

Set the unit cost values of a resource.

Usage:

```
model.setResourceFlowValue(key,value);
```

Input Parameters:

key – char array with the flow key

value – unit cost value of the resource flow:

6.9 Result Info Functions

Get a *cResultInfo* object containing the tables and the properties of the results.

productiveStructure

Get information about the productive structure of the model.

Usage:

```
res = model.productiveStructure;
```

Output Parameters

res – Productive Structure cResultInfo object

productiveDiagram

Get the information about the Productive Diagram of the model

Usage:

```
res = model.productiveDiagram;
```

Output Parameters

res – Productive Structure cResultInfo object

thermoeconomicState

Get the information about the exergy values of the current state of the model.

Usage:

```
res = model.thermoeconomicState;
```

Output Parameters

res – Thermoeconomic State cResultInfo object

thermoeconomicAnalysis

Get the information about the thermoeconomic analysis of the actual state of the model.

Usage:

```
res = model.thermoeconomicAnalysis;
```

Output Parameters

res – Thermoeconomic Analysis cResultInfo object

thermoeconomicDiagnosis

Get the information about the thermoeconomic diagnosis of the current state of the model. It is activated, if the *ReferenceState* is different to the Operation *State*, and *DiagnosisMethod* is activated (different to NONE)

Usage:

```
res = model.thermoeconomicDiagnosis;
```

Output Parameters

res – Thermoeconomic Diagnosis *cResultInfo* object

recyclingAnalysis

Get the information about the recycling analysis of the current state of the model for one of the waste flows.

Usage:

```
res = model.recyclingAnalysis(wkey);
```

Input Parameters

wkey – char array with the name of the waste flow to analyse

Output Parameters

res – Recycling Analysis *cResultInfo* object

diagramFP

Get the information about the diagram FP of the current state of the model

Usage:

```
res = model.diagramFP(option);
```

Input Parameters:

option – char array indicating the diagram FP to obtain. Valid Values:

```
cType.Tables.TABLE_FP
cType.Tables.COST_TABLE_FP
```

If option is missing the first option is selected.

Output Parameters

res – DiagramFP cResultInfo object

summaryResults

Get the comparative results info of the cost of flows and processes for the different states of the plant

Usage:

```
res = model.summaryResults
```

Output Parameters

res –Summary Results *cResultInfo* object

Example:

A LiveScript `ThermoeconomicToolDemo.mlx` is show as example of all the functionality.

Check the data model file, create the thermoeconomic model and set some properties

```
model=ThermoeconomicTool('rorc_model.xlsx',...
    'CostTables','ALL',...
    'DiagnosisMethod','NONE',...
    'debug',false);
```

```
INFO: cStatusLogger. Productive Structure is valid
INFO: cStatusLogger. Format Configuration is valid
INFO: cStatusLogger. Exergy values [TCND30] are valid
INFO: cStatusLogger. Exergy values [TCND45] are valid
INFO: cStatusLogger. Exergy values [ETAT75] are valid
INFO: cStatusLogger. Exergy values [PBLR15] are valid
INFO: cStatusLogger. Exergy values [NoIHE] are valid
INFO: cStatusLogger. Exergy values [N_Butane] are valid
INFO: cStatusLogger. Resources Cost sample [Base] is valid
INFO: cStatusLogger. Waste definition is valid
INFO: cReadModelXLS. Data Model rorc_model.xlsx is valid
```

Print the current property values of the model.

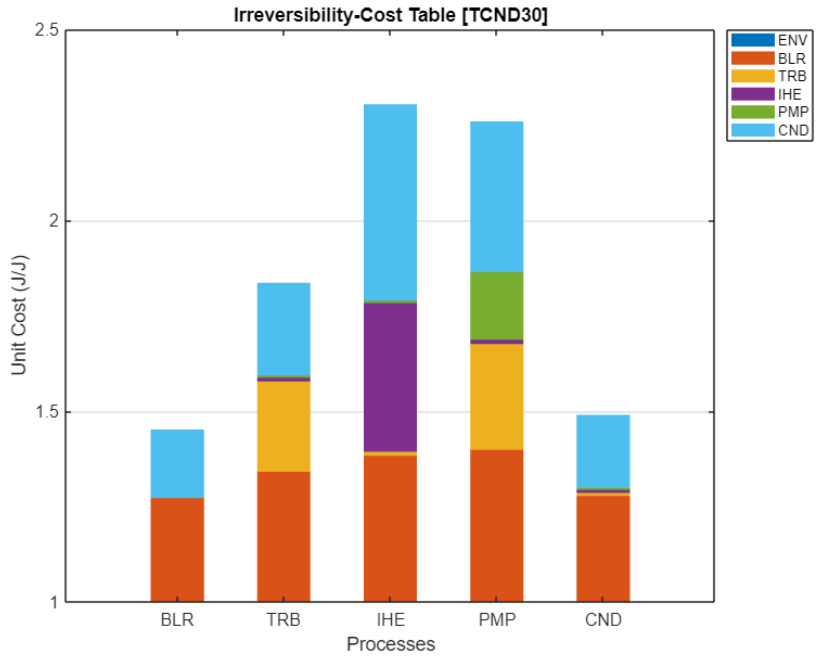
```
showProperties(model);
```

```
             State: 'TCND30'
    ReferenceState: 'TCND30'
    ResourceSample: 'Base'
        CostTables: 'ALL'
    IsResourceCost: 'true'
       IsDiagnosis: 'false'
   DiagnosisMethod: 'NONE'
           Summary: 'false'
             Debug: 'false'
```

Print some results for State TCND30

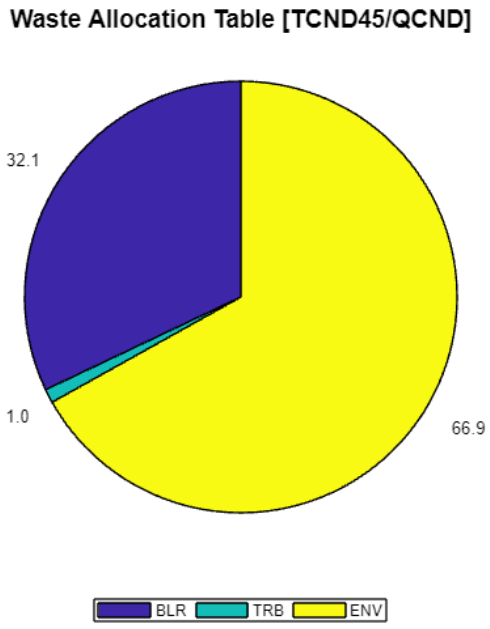
```
model.printTable('tfp');
model.printTable('ducost');
model.graphCost;
```

```
FP Table (kW)

 Key          BLR       TRB       IHE       PMP       CND       ENV      Total
-------------------------------------------------------------------------------
 BLR        0.000    61.308     1.936     0.000     8.906     0.000    72.150
 TRB        0.000     0.000     0.000     2.381     0.000    50.000    52.381
 IHE        0.000     1.252     0.040     0.000     0.182     0.000     1.473
 PMP        0.000     1.730     0.055     0.000     0.251     0.000     2.036
 CND        0.000     0.000     0.000     0.000     0.000     9.339     9.339
 ENV       91.790     0.000     0.000     0.000     0.000     0.000    91.790
-------------------------------------------------------------------------------
 Total     91.790    64.290     2.030     2.381     9.339    59.339

Process Unit Exergy Cost

 Key    kP*(J/J)  kPe*(J/J)  kPr*(J/J)   kF*(J/J)   kR*(J/J)     k(J/J)
-------------------------------------------------------------------------
 BLR      1.4519     1.2722     0.1797     1.0000     0.1797     1.2722
 TRB      1.8358     1.5933     0.2425     1.4902     0.0068     1.2274
 IHE      2.3033     1.7891     0.5142     1.4902     0.2496     1.3781
 PMP      2.2587     1.8633     0.3954     1.8358     0.1119     1.1694
 CND      1.4902     1.2982     0.1920     1.4902     0.0000     1.0000
```



Select state TCND45, define the recycled waste, show waste allocation piechart and print the direct cost table

```
model.State='TCND45';
model.setWasteRecycled('QCND',0.66);
model.graphWasteAllocation;
model.printTable('dcost');
```

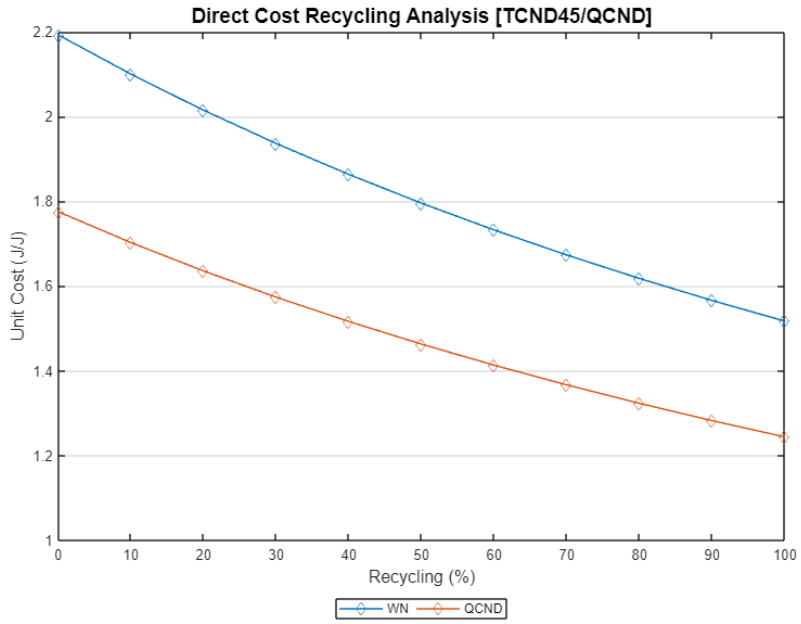


Process Unit Exergy Cost

Key	kP*(J/J)	kPe*(J/J)	kPr*(J/J)	kF*(J/J)	kR*(J/J)	k(J/J)
BLR	1.3560	1.2234	0.1327	1.0000	0.1327	1.2234
TRB	1.6961	1.5171	0.1790	1.3849	0.0073	1.2195
IHE	1.6932	1.4379	0.2553	1.3849	0.0926	1.1558
PMP	2.0667	1.7626	0.3041	1.6961	0.0961	1.1618
CND	1.3849	1.2441	0.1408	1.3849	0.0000	1.0000

Plot the recycling analysis graph

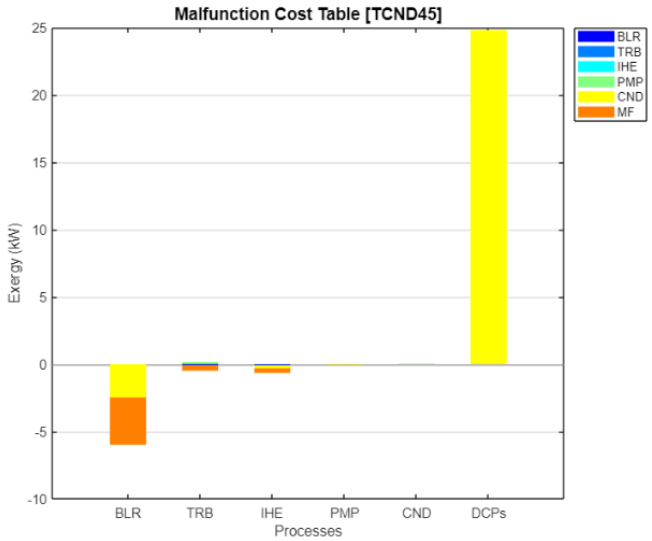
```
model.graphRecycling('QCND',cType.Tables.WASTE_RECYCLING_DIRECT);
```



Compare both states using the thermoeconomicDiagnosis function.

```
model.DiagnosisMethod='WASTE_INTERNAL';
model.fuelImpact;
model.printTable(cType.Tables.DIAGNOSIS_SUMMARY);
```

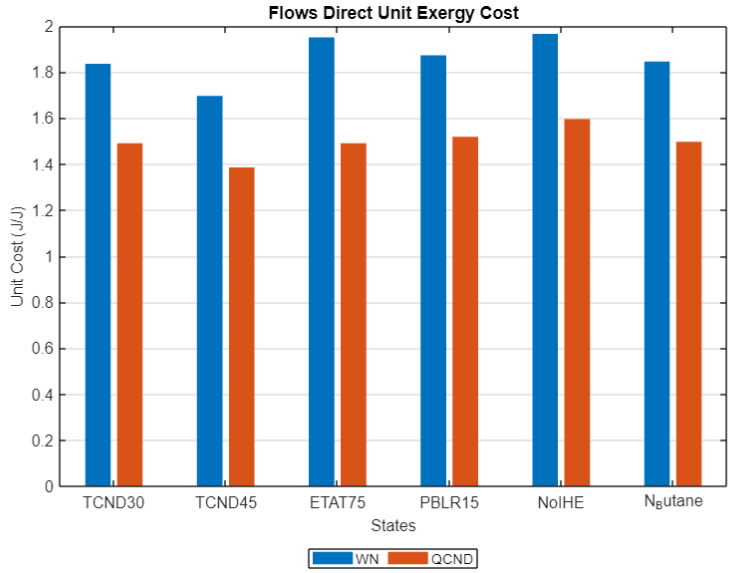
```
Fuel Impact:   17.8100 (kW)

Diagnosis Summary

 Key       MF(kW)      DI(kW)     DPs(kW)     MF*(kW)     MR*(kW)    DPs*(kW)
-------------------------------------------------------------------------------
 BLR      -3.5252      0.3700      0.0000     -3.5252     -2.4752      0.0000
 TRB      -0.4136     -0.2870      0.0000     -0.3848      0.0660      0.0000
 IHE      -0.3275     -0.1270      0.0000     -0.4033     -0.2491      0.0000
 PMP      -0.0155      0.0670      0.0000     -0.0236     -0.0187      0.0000
 CND       0.0000      0.0000     17.7870      0.0189      0.0109     24.7942
 ENV      -4.2819      0.0230     17.7870     -4.3180     -2.6662     24.7942
```



Show Summary Tables and Graphs

```
model.printSummary(cType.SummaryTables.FLOW_DIRECT_UNIT_COST)
model.graphSummary;
```

```
Flows Direct Unit Exergy Cost (J/J)

 Key       TCND30    TCND45    ETAT75    PBLR15     NoIHE   N_Butane
-------------------------------------------------------------------
 B1        1.4902    1.3849    1.4904    1.5185    1.5957    1.4970
 B2        1.4902    1.3849    1.4904    1.5185    1.5957    1.4970
 B3        1.4902    1.3849    1.4904    1.5185    1.5957    1.4970
 B4        1.4902    1.3849    1.4904    1.5185    1.5957    1.4970
 B5        1.6953    1.5290    1.7312    1.6844    1.8155    1.5695
 B6        1.7937    1.5596    1.8430    1.7802    1.8155    1.6074
 QBLR      1.0000    1.0000    1.0000    1.0000    1.0000    1.0000
 WP        1.8358    1.6961    1.9504    1.8722    1.9658    1.8450
 WN        1.8358    1.6961    1.9504    1.8722    1.9658    1.8450
 QCND      1.4902    1.3849    1.4904    1.5185    1.5957    1.4970
```



7 GUI Applications

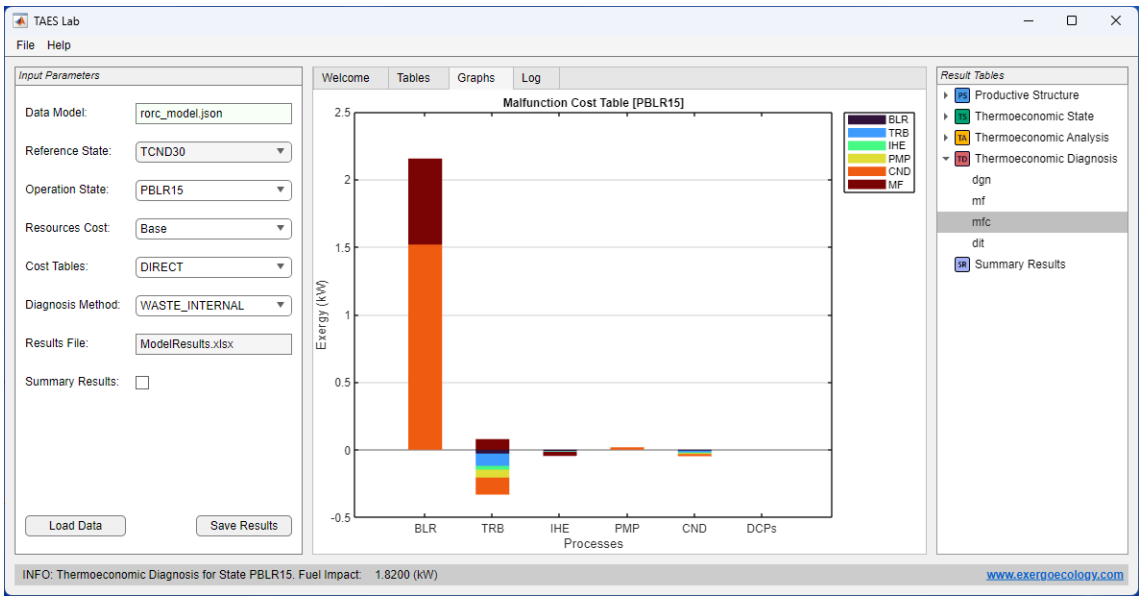
TaesLab provides some additional apps or graphic interface functions.

7.1 TaesLab

It is the main application using the MATLAB environment. It provides most of the features of the ThermoeconomicTool in graphic mode. It lets load a data model file, select the parameters to make the thermoeconomic analysis of the plant, including diagnosis, and visualise the tables and graph in the main panel. A full description of this application can be found in TaesLab documentation file TaesLab.pptx.

Usage:

```
TaesLab;
```



7.2 ViewModelResults

This function lets to show the result tables and graphs of a *cResultInfo* or *cThermoeconomicModel* objects obtained by the functions of the TaesLab package in a panel like the TaesLab application.

In the left panel appears the Results Info tree panel to select the table to view, and the main panel has two tabs, one to show the tables and the other to show graphs associated with the table.

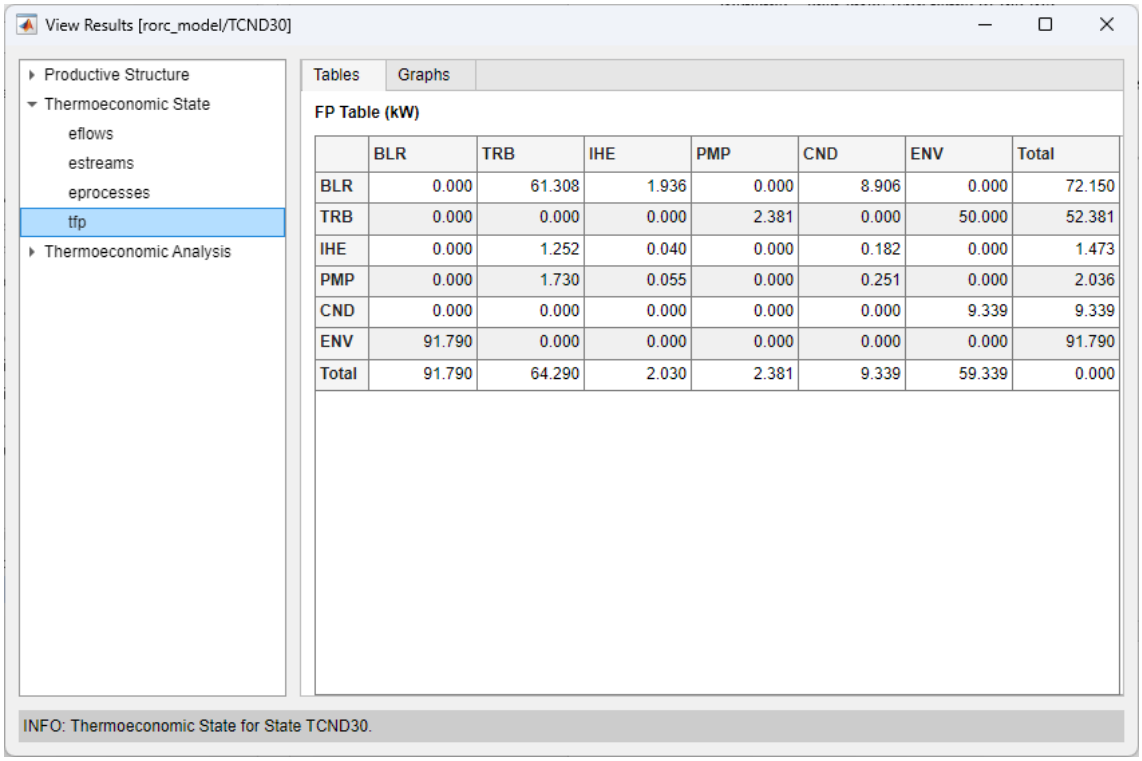
It works only in the MATLAB environment.

Usage:

```
ViewModelResults(res);
```

Input Parameters:

res – cResultInfo or cThermoeconomicModel objects containing the model results



	BLR	TRB	IHE	PMP	CND	ENV	Total
BLR	0.000	61.308	1.936	0.000	8.906	0.000	72.150
TRB	0.000	0.000	0.000	2.381	0.000	50.000	52.381
IHE	0.000	1.252	0.040	0.000	0.182	0.000	1.473
PMP	0.000	1.730	0.055	0.000	0.251	0.000	2.036
CND	0.000	0.000	0.000	0.000	0.000	9.339	9.339
ENV	91.790	0.000	0.000	0.000	0.000	0.000	91.790
Total	91.790	64.290	2.030	2.381	9.339	59.339	0.000

7.3 ThermoeconomicPanel

ThermoeconomicPanel is a Graphic Tool used as an interface to the *cThermoeconomicModel* class, which permits the selection of the different parameters and options to manage a thermoeconomic model. It performs the following functions:

- Select the data model.
- Save the result information.
- Export the results to the MATLAB workspace.

It works both in Octave and MATLAB environments. In the case of MATLAB it is possible to dock the panel in the MATLAB window.

Usage:

```
app=ThermoeconomicPanel;
```

Output Parameters:

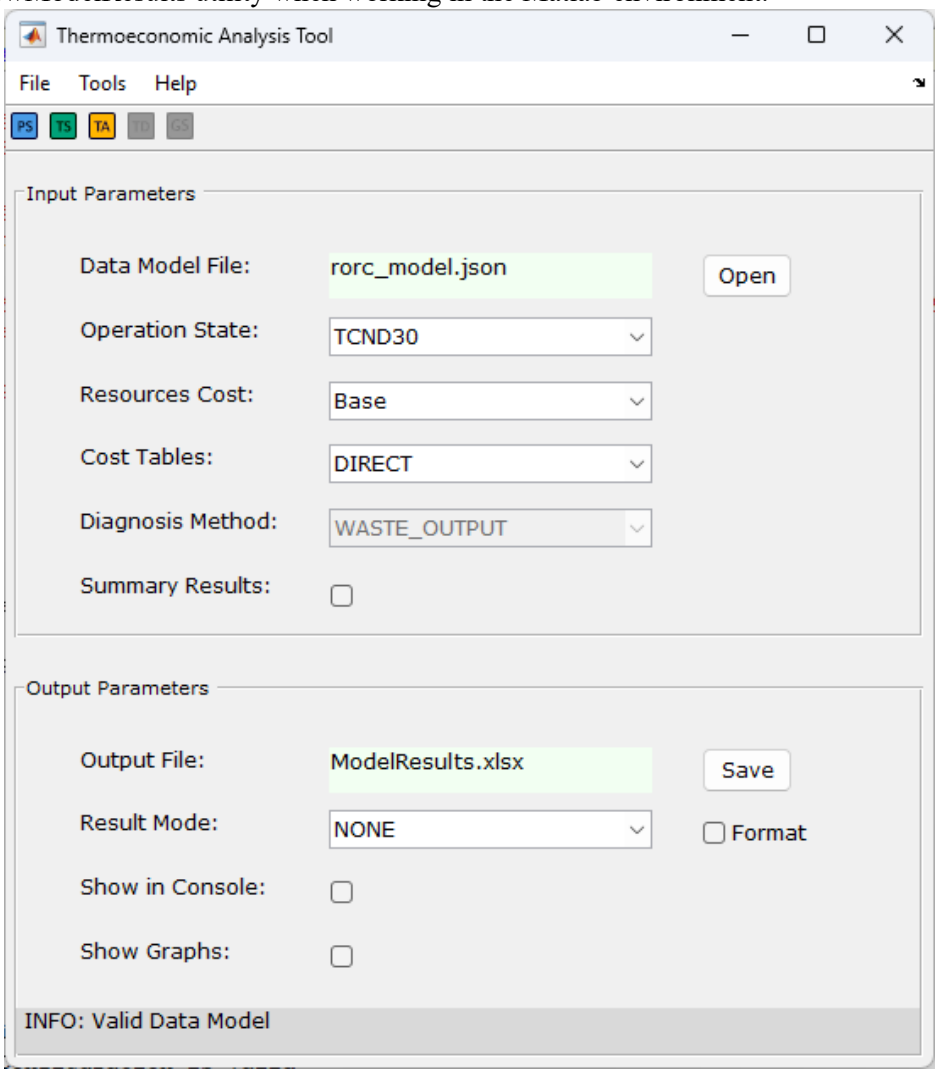
app – ThermoeconomicPanel object, which lets access to the internal cThermoeconomicModel object.

app. model – The current *cThermoeconomicModel* object

app.Options – The current output options

app.Results – The current *cResultInfo* objects of the model.

The interface allows the convenience of a graphical interface, avoiding errors when typing commands, with the flexibility of the interactive commands of the ThermoeconomicTool function. The results can be viewed with the ViewModelResults utility when working in the Matlab environment.

- 

The panel have the following work areas:

- Input Parameters: It ss equivalent to the left panel of *TaesLab* app.
- Output Parameters: Permit to select the outputs.
- ToolBar: Permit to select the Result Info to manage.
- Menu File: Open the data model and save the results.
- Menu Tools: Like ToolBar, permit operation with the Result Info.

Output Parameters

In this area, we can manage the Result Info output

- *Output File*: Show the file name where the results are saved.
- *Save Button*: Activate the results save.
- *Result Mode*: Indicate how the Result Info will be saved into the workspace.
    - NONE: Save as *cResultInfo* object
    - CELL: Save the tables as a cell array
    - STRUCT: Save the tables as structured arrays
    - TABLE: Save as MATLAB tables (only MATLAB)

When pressing the corresponding Result Info in Tool Bar or Menu Tool the Results are saved into the workspace in the indicated mode. The modes CELL and STRUCT are designed to use with Octave workspace.

- *Format*: Use the format definition to save results (applies to CELL and STRUCT modes)
- *Show in console*: print the results in the console.
- *Show Graphs*: When MATLAB is used, it opens the ViewModelResults application to see the selected results. In the case of Octave, it shows the default graph or table for each result group.

Menu File

This menu permits to make the file operations:

- Open: Open a file dialog to select the data model file
- Save: Open a file dialog to save the current results.
    - Save Model Results.
    - Save Data Model.
    - Save Summary
    - Save Diagram FP information.

Tool Bar and Menu Tool

The results information of the selected group are shown by pressing the corresponding buttons:

- Productive Structure
- Thermoeconomic State
- Thermoeconomic Analysis
- Thermoeconomic Diagnosis
- Summary Results.

Results are saved into the workspace, printed on the console, or visualised in the ViewModelResult application.it

The options Save Model Results save the model into the workspace and visualise all the result blocks in the app in the case of MATLAB.