D5_LCA

April 26, 2022

This notebook conducts a life cycle assessment (LCA) based on a theoretical case study, which analyses the environmental impact of different scenarios for the replacement of the fully glazed façade of an office building located in Brussels. It is part of the doctoral dissertation entitled *Glazing Beyond Energy Efficiency*, and refers to its **Chapter 4, "The Uncertainties of Efficiency."** As such, it should be read in concert with that chapter, which presents the conceptual and methodological framework (Section 5.1) and discusses the results (Sections 5.2 to 5.4).

This notebook relies on hypotheses and scenarios presented in the Excel file named **"D1_BEM_LCA_Hypotheses.xlsx"**. It also uses life cycle inventory (LCI) datasets available in Excel format in the folder **"D2_lci."** To these inventory data are added the results of the energy flows over the use phase, coming from the building energy simulations carried out in the notebook **"D4_BEM."** The LCI is completed by data from **Biosphere 3** (included in the Brightway2 package) and **Ecoinvent** (see: www.ecoinvent.org/).

To process the data, conduct the life cycle impact assessment, and perform the uncertainty analysis, the script relies on the LCA framework called **Brightway2**. As such, to run the script, Brightway2 should be installed (open source, see: https://brightway.dev/).

This notebook is structured in 13 parts: 1. The setup steps needed to run the script. 2. The definition of scenario lists and run batches. 3. The import of LCI datasets, including Ecoinvent, Biosphere 3 and the LCI datasets defined in the framework of this PhD research and available in Excel format. 4. Specification of LCA parameters. 5. Definition of the LCIA methods (LCAM). 6. Presentation of the weighting method and its factors. 7. A cradle-to-gate comparative analysis of glazing. 8. A comparative analysis from cradle-to-gate of curtain wall systems. 9. Import of the results obtained from building energy simulations for the use phase inventory. 10. Full life cycle analysis of different façade replacement strategies. 11. Post-processing of the data to plot the environmental trajectory over 40 years of service life. 12. Discussion of the results and sensitivity analysis. 13. Sensitivity analysis according to the electricity mix.

The script exports in CSV format to the "outputs" folder the main results.

To cite: Souviron, Jean. 2022. "Glazing Beyond Energy Efficiency: An Environmental Analysis of the Socio-Technical Trajectory of Architectural Glass." PhD diss., Université Libre de Bruxelles

1 Setup

First, import modules and codes from modules to run this notebook:

[1]:
```
from IPython.display import display
```

```python
from brightway2 import *
import bw2analyzer as bwa
import brightway2 as bw
from bw2data.parameters import *
from support.lci_to_bw2 import *
from bw2data.project import ProjectManager
from bw2data.parameters import (ActivityParameter, DatabaseParameter,
                                ProjectParameter, Group)

import pandas as pd
import numpy as np

import math
from decimal import *

import pathlib

import sqlite3

import os

import seaborn as sns

import matplotlib as mpl
import matplotlib.pyplot as plt
from matplotlib.ticker import FuncFormatter
%matplotlib inline
```

Defining a few global parameters:

[2]:
```python
# Defining the directory with datasets:
ROOT_DIR = "D2_lci"
```

[3]:
```python
# Defining the size of figures:
mpl.rcParams['figure.figsize'] = (16, 10)
pd.options.display.max_rows = 200
```

[4]:
```python
# Defining the path where to save figures:
path_img = os.path.abspath(os.path.join('outputs', 'fig_lca'))
if not os.path.exists(path_img):
    os.makedirs(path_img)
print(f'Images will be saved in {path_img}')
```

```
Images will be saved in C:\Users\souvi\Documents\These\90_PresentationsAndWritti
ng\90_Manuscript\5_Appendices\D\outputs\fig_lca
```

[5]:
```python
# Defining seaborn main parameters:
sns.set_style("ticks")
```

```
sns.color_palette("colorblind")
sns.set_context("paper", font_scale=1.25,
                rc={"axes.titlesize": 12, "lines.linewidth": 1,
                    "legend.fontsize": 10, "legend.title_fontsize": 10})
```

[6]:
```
# A function used to define the thickness of x and y axis:
def style_ax(ax):
    for axis in ['top', 'bottom', 'left', 'right']:
        ax.spines[axis].set_linewidth(0.5)
        ax.tick_params(width=0.5)
        ax.set_xlabel(None)
    return ax
```

[7]:
```
# Listing the available Brightway2 projects:
bw.projects
```

[7]:
```
Brightway2 projects manager with 5 objects:
	LCA_Glazing
	LCA_Glazing_0
	LCOPT_Setup
	default
	test
Use `projects.report()` to get a report on all projects.
```

[8]:
```
# Creating a new project or accessing an existing one:
bw.projects.set_current("LCA_Glazing")

# Locating the current project:
bw.projects.dir
```

[8]:
```
'C:\\Users\\souvi\\AppData\\Local\\pylca\\Brightway3\\LCA_Glazing.d2e1ffa0d7e38b
337d42880125eeaeab'
```

[9]:
```
# A boolean to export or not the graphs:
export = False
```

2 List of Scenarios with their Parameters

All scenarios and their parameters for the LCA are defined in the Excel file called lca_scenarios. Here it is imported.

[10]:
```
lca_scenarios = pd.ExcelFile(os.path.join(ROOT_DIR, "lca_scenarios.xlsx"))

# Printng the list of sheets in the Excel file:
print("lca_scenarios, sheet names = \n {}\n".format(lca_scenarios.sheet_names))
```

```
lca_scenarios, sheet names =
 ['Scenarios', 'Step1', 'Step2', 'Step3', 'Step4', 'Step5', 'Step6', 'Step7',
```

```
'Step8', 'Step9', 'Step10', 'Step11', 'Step12', 'Step13', 'Step14', 'Step15',
'Step16']
```

Creating a set of DataFrames. One for each calculation step, which corresponds to a batch of simulations defined by a specific building configuration with different types of IGUs:

[11]:
```python
# Creating one DataFrame per step:
df_step1 = lca_scenarios.parse('Step1').set_index('name')
df_step2 = lca_scenarios.parse('Step2').set_index('name')
df_step3 = lca_scenarios.parse('Step3').set_index('name')
df_step4 = lca_scenarios.parse('Step4').set_index('name')
df_step5 = lca_scenarios.parse('Step5').set_index('name')
df_step6 = lca_scenarios.parse('Step6').set_index('name')
df_step7 = lca_scenarios.parse('Step7').set_index('name')
df_step8 = lca_scenarios.parse('Step8').set_index('name')
df_step9 = lca_scenarios.parse('Step9').set_index('name')
df_step10 = lca_scenarios.parse('Step10').set_index('name')
df_step11 = lca_scenarios.parse('Step11').set_index('name')
df_step12 = lca_scenarios.parse('Step12').set_index('name')
df_step13 = lca_scenarios.parse('Step13').set_index('name')
df_step14 = lca_scenarios.parse('Step14').set_index('name')
df_step15 = lca_scenarios.parse('Step15').set_index('name')
df_step16 = lca_scenarios.parse('Step16').set_index('name')
```

3 Import of LCA Databases

Importing databases that include LCIA methods, global life cycle inventories (Ecoinvent and Biosphere 3) and those that are specific to this study (saved as Excel files in the subfolder "files").

[12]:
```python
# Print the databases already available in the current project:
bw.databases
```

[12]:
```
Databases dictionary with 8 object(s):
        biosphere3
        ecoinvent 3.7 cut-off
        exldb_alu
        exldb_cw
        exldb_cw_eol
        exldb_igu
        exldb_sand
        exldb_spacers
```

3.1 Ecoinvent and Biosphere 3

Importing Biosphere 3:

Biosphere 3 is the default biosphere database with all the resource and emission flows from the ecoinvent database, version 2.

```
[13]: # Importing elementary flows, LCIA methods and some other data
bw.bw2setup()
```

```
Biosphere database already present!!! No setup is needed
```

Importing Ecoinvent 3.7, cut-off system model:

For more information about the system models in ecoinvent, and especially the cut-off one, read this.

```
[14]: # Importing the ecoinvent 3.7 cut-off database, saved locally:
ei37cutdir = (r"C:
↪\Users\souvi\Documents\These\80_Calculations\06_LCA_SystemDiagrams\02_Dataset\ecoinvent␣
↪3.7_cutoff_ecoSpold02\datasets")

if 'ecoinvent 3.7 cut-off' in databases:
    print("Database has already been imported!")
else:
    ei37cut = bw.SingleOutputEcospold2Importer(
        ei37cutdir, 'ecoinvent 3.7 cut-off')
    ei37cut.apply_strategies()
    ei37cut.statistics()
    ei37cut.write_database()
```

```
Database has already been imported!
```

3.2 Excel Datasets

Import of the life cycle inventory specific to this case study and saved in the Excel files.

But first, a boolean variable to specify if importing (or updating) the inventory is necessary:

```
[15]: import_exldb = True
```

Importing the Excel dataset relating to aluminium production, regionalised for the case study:

```
[16]: if import_exldb:
    imp = bw.ExcelImporter(os.path.join(ROOT_DIR, "lci_alu.xlsx"))
    imp.apply_strategies()
    imp.match_database(fields=('name', 'unit', 'location'))
    imp.match_database("ecoinvent 3.7 cut-off",
                       fields=('name', 'unit', 'location', 'input'))
    imp.statistics()

    # Checking whether the import went as expected.
    # Creating an Excel sheet with process data:
    imp.write_excel()

    # Writing the data to a database to save it:
```

```
    imp.write_database()
```

```
Extracted 2 worksheets in 0.02 seconds
Applying strategy: csv_restore_tuples
Applying strategy: csv_restore_booleans
Applying strategy: csv_numerize
Applying strategy: csv_drop_unknown
Applying strategy: csv_add_missing_exchanges_section
Applying strategy: normalize_units
Applying strategy: normalize_biosphere_categories
Applying strategy: normalize_biosphere_names
Applying strategy: strip_biosphere_exc_locations
Applying strategy: set_code_by_activity_hash
Applying strategy: link_iterable_by_fields
Applying strategy: assign_only_product_as_production
Applying strategy: link_technosphere_by_activity_hash
Applying strategy: drop_falsey_uncertainty_fields_but_keep_zeros
Applying strategy: convert_uncertainty_types_to_integers
Applying strategy: convert_activity_parameters_to_list
Applied 16 strategies in 0.25 seconds
Applying strategy: link_iterable_by_fields
Applying strategy: link_iterable_by_fields

Writing activities to SQLite3 database:

2 datasets
12 exchanges
0 unlinked exchanges

Wrote matching file to:
C:\Users\souvi\AppData\Local\pylca\Brightway3\LCA_Glazing.d2e1ffa0d7e38b337d4288
0125eeaeab\output\db-matching-exldb_alu.xlsx

0% [##] 100% | ETA: 00:00:00
Total time elapsed: 00:00:00

Title: Writing activities to SQLite3 database:
  Started: 04/26/2022 12:12:29
  Finished: 04/26/2022 12:12:29
  Total time elapsed: 00:00:00
  CPU %: 0.00
  Memory %: 1.25
Created database: exldb_alu
```

Importing the Excel dataset relating to silica sand production, regionalised for the case study:

[17]:
```
if import_exldb:
    imp = bw.ExcelImporter(os.path.join(ROOT_DIR, "lci_silica_sand.xlsx"))
    imp.apply_strategies()
```

```python
    imp.match_database(fields=('name', 'unit', 'location'))
    imp.match_database("ecoinvent 3.7 cut-off",
                       fields=('name', 'unit', 'location', 'input'))
    imp.statistics()
    imp.write_excel()
    imp.write_database()
```

```
Extracted 2 worksheets in 0.03 seconds
Applying strategy: csv_restore_tuples
Applying strategy: csv_restore_booleans
Applying strategy: csv_numerize
Applying strategy: csv_drop_unknown
Applying strategy: csv_add_missing_exchanges_section
Applying strategy: normalize_units
Applying strategy: normalize_biosphere_categories
Applying strategy: normalize_biosphere_names
Applying strategy: strip_biosphere_exc_locations
Applying strategy: set_code_by_activity_hash
Applying strategy: link_iterable_by_fields
Applying strategy: assign_only_product_as_production
Applying strategy: link_technosphere_by_activity_hash
Applying strategy: drop_falsey_uncertainty_fields_but_keep_zeros
Applying strategy: convert_uncertainty_types_to_integers
Applying strategy: convert_activity_parameters_to_list
Applied 16 strategies in 0.19 seconds
Applying strategy: link_iterable_by_fields
Applying strategy: link_iterable_by_fields
```

```
Writing activities to SQLite3 database:
```

```
2 datasets
29 exchanges
0 unlinked exchanges

Wrote matching file to:
C:\Users\souvi\AppData\Local\pylca\Brightway3\LCA_Glazing.d2e1ffa0d7e38b337d4288
0125eeaeab\output\db-matching-exldb_sand.xlsx
```

```
0% [##] 100% | ETA: 00:00:00
Total time elapsed: 00:00:00
```

```
Title: Writing activities to SQLite3 database:
  Started: 04/26/2022 12:12:31
  Finished: 04/26/2022 12:12:31
  Total time elapsed: 00:00:00
  CPU %: 0.00
  Memory %: 1.32
Created database: exldb_sand
```

Importing the Excel dataset relating to the insulating glass units:

[18]:
```python
if import_exldb:
    imp = bw.ExcelImporter(os.path.join(ROOT_DIR, "lci_igu.xlsx"))
    imp.apply_strategies()
    imp.match_database(fields=('name', 'unit', 'location'))
    imp.match_database("ecoinvent 3.7 cut-off",
                       fields=('name', 'unit', 'location'))
    imp.match_database("exldb_alu",
                       fields=('name', 'unit', 'location', 'input'))
    imp.match_database("exldb_sand",
                       fields=('name', 'unit', 'location', 'input'))
    imp.statistics()
    imp.write_excel()

    # Adding the project-level parameters:
    imp.write_project_parameters()

    # Writing the data to a database to save it:
    imp.write_database()
```

```
Extracted 44 worksheets in 0.33 seconds
Applying strategy: csv_restore_tuples
Applying strategy: csv_restore_booleans
Applying strategy: csv_numerize
Applying strategy: csv_drop_unknown
Applying strategy: csv_add_missing_exchanges_section
Applying strategy: normalize_units
Applying strategy: normalize_biosphere_categories
Applying strategy: normalize_biosphere_names
Applying strategy: strip_biosphere_exc_locations
Applying strategy: set_code_by_activity_hash
Applying strategy: link_iterable_by_fields
Applying strategy: assign_only_product_as_production
Applying strategy: link_technosphere_by_activity_hash
Applying strategy: drop_falsey_uncertainty_fields_but_keep_zeros
Applying strategy: convert_uncertainty_types_to_integers
Applying strategy: convert_activity_parameters_to_list
Applied 16 strategies in 0.21 seconds
Applying strategy: link_iterable_by_fields
Applying strategy: link_iterable_by_fields

Writing activities to SQLite3 database:

Applying strategy: link_iterable_by_fields
Applying strategy: link_iterable_by_fields
44 datasets
379 exchanges
0 unlinked exchanges

Wrote matching file to:
```

```
C:\Users\souvi\AppData\Local\pylca\Brightway3\LCA_Glazing.d2e1ffa0d7e38b337d4288
0125eeaeab\output\db-matching-exldb_igu.xlsx

0% [##############################] 100% | ETA: 00:00:00
Total time elapsed: 00:00:00

Title: Writing activities to SQLite3 database:
  Started: 04/26/2022 12:12:34
  Finished: 04/26/2022 12:12:34
  Total time elapsed: 00:00:00
  CPU %: 80.10
  Memory %: 1.43
Created database: exldb_igu
```

Importing the Excel dataset relating to double glazing w/ different types of spacers:

[19]:
```python
if import_exldb:
    imp = bw.ExcelImporter(os.path.join(ROOT_DIR, "lci_spacers.xlsx"))
    imp.apply_strategies()
    imp.match_database(fields=('name', 'unit', 'location'))
    imp.match_database("ecoinvent 3.7 cut-off",
                       fields=('name', 'unit', 'location'))
    imp.match_database("exldb_igu",
                       fields=('name', 'unit', 'location', 'input'))
    imp.statistics()
    imp.write_excel()
    imp.write_database()
```

```
Extracted 13 worksheets in 0.10 seconds
Applying strategy: csv_restore_tuples
Applying strategy: csv_restore_booleans
Applying strategy: csv_numerize
Applying strategy: csv_drop_unknown
Applying strategy: csv_add_missing_exchanges_section
Applying strategy: normalize_units
Applying strategy: normalize_biosphere_categories
Applying strategy: normalize_biosphere_names
Applying strategy: strip_biosphere_exc_locations
Applying strategy: set_code_by_activity_hash
Applying strategy: link_iterable_by_fields
Applying strategy: assign_only_product_as_production
Applying strategy: link_technosphere_by_activity_hash
Applying strategy: drop_falsey_uncertainty_fields_but_keep_zeros
Applying strategy: convert_uncertainty_types_to_integers
Applying strategy: convert_activity_parameters_to_list
Applied 16 strategies in 0.20 seconds
Applying strategy: link_iterable_by_fields
Applying strategy: link_iterable_by_fields

Writing activities to SQLite3 database:
```

```
Applying strategy: link_iterable_by_fields
13 datasets
178 exchanges
0 unlinked exchanges

Wrote matching file to:
C:\Users\souvi\AppData\Local\pylca\Brightway3\LCA_Glazing.d2e1ffa0d7e38b337d4288
0125eeaeab\output\db-matching-exldb_spacers.xlsx

0% [#############] 100% | ETA: 00:00:00
Total time elapsed: 00:00:00

Title: Writing activities to SQLite3 database:
  Started: 04/26/2022 12:12:36
  Finished: 04/26/2022 12:12:36
  Total time elapsed: 00:00:00
  CPU %: 100.80
  Memory %: 1.39
Created database: exldb_spacers
```

Importing the Excel dataset relating to the end-of-life phase of curtain wall façades:

[20]:
```
if import_exldb:
    imp = bw.ExcelImporter(os.path.join(ROOT_DIR, "lci_cw_eol.xlsx"))
    imp.apply_strategies()
    imp.match_database(fields=('name', 'unit', 'location'))
    imp.match_database("ecoinvent 3.7 cut-off",
                       fields=('name', 'unit', 'location'))
    imp.statistics()
    imp.write_excel()
    imp.write_database()
```

```
Extracted 28 worksheets in 0.17 seconds
Applying strategy: csv_restore_tuples
Applying strategy: csv_restore_booleans
Applying strategy: csv_numerize
Applying strategy: csv_drop_unknown
Applying strategy: csv_add_missing_exchanges_section
Applying strategy: normalize_units
Applying strategy: normalize_biosphere_categories
Applying strategy: normalize_biosphere_names
Applying strategy: strip_biosphere_exc_locations
Applying strategy: set_code_by_activity_hash
Applying strategy: link_iterable_by_fields
Applying strategy: assign_only_product_as_production
Applying strategy: link_technosphere_by_activity_hash
Applying strategy: drop_falsey_uncertainty_fields_but_keep_zeros
Applying strategy: convert_uncertainty_types_to_integers
Applying strategy: convert_activity_parameters_to_list
Applied 16 strategies in 0.20 seconds
```

```
Applying strategy: link_iterable_by_fields
Applying strategy: link_iterable_by_fields

Writing activities to SQLite3 database:

28 datasets
108 exchanges
0 unlinked exchanges

Wrote matching file to:
C:\Users\souvi\AppData\Local\pylca\Brightway3\LCA_Glazing.d2e1ffa0d7e38b337d4288
0125eeaeab\output\db-matching-exldb_cw_eol.xlsx

0% [############################] 100% | ETA: 00:00:00
Total time elapsed: 00:00:00

Title: Writing activities to SQLite3 database:
  Started: 04/26/2022 12:12:39
  Finished: 04/26/2022 12:12:39
  Total time elapsed: 00:00:00
  CPU %: 208.30
  Memory %: 1.41
Created database: exldb_cw_eol
```

Importing the Excel dataset relating to the production and use of curtain wall façades:

[21]:
```
if import_exldb:
    imp = bw.ExcelImporter(os.path.join(ROOT_DIR, "lci_cw.xlsx"))
    imp.apply_strategies()
    imp.match_database(fields=('name', 'unit', 'location'))
    imp.match_database("ecoinvent 3.7 cut-off",
                       fields=('name', 'unit', 'location'))
    imp.match_database("exldb_igu",
                       fields=('name', 'unit', 'location', 'input'))
    imp.match_database("exldb_alu",
                       fields=('name', 'unit', 'location', 'input'))
    imp.match_database("exldb_cw_eol",
                       fields=('name', 'unit', 'location', 'input'))
    imp.statistics()
    imp.write_excel()
    imp.write_database()
```

```
Extracted 48 worksheets in 0.24 seconds
Applying strategy: csv_restore_tuples
Applying strategy: csv_restore_booleans
Applying strategy: csv_numerize
Applying strategy: csv_drop_unknown
Applying strategy: csv_add_missing_exchanges_section
Applying strategy: normalize_units
Applying strategy: normalize_biosphere_categories
Applying strategy: normalize_biosphere_names
```

```
Applying strategy: strip_biosphere_exc_locations
Applying strategy: set_code_by_activity_hash
Applying strategy: link_iterable_by_fields
Applying strategy: assign_only_product_as_production
Applying strategy: link_technosphere_by_activity_hash
Applying strategy: drop_falsey_uncertainty_fields_but_keep_zeros
Applying strategy: convert_uncertainty_types_to_integers
Applying strategy: convert_activity_parameters_to_list
Applied 16 strategies in 0.22 seconds
Applying strategy: link_iterable_by_fields
Applying strategy: link_iterable_by_fields

Writing activities to SQLite3 database:

Applying strategy: link_iterable_by_fields
Applying strategy: link_iterable_by_fields
Applying strategy: link_iterable_by_fields
48 datasets
245 exchanges
0 unlinked exchanges

Wrote matching file to:
C:\Users\souvi\AppData\Local\pylca\Brightway3\LCA_Glazing.d2e1ffa0d7e38b337d4288
0125eeaeab\output\db-matching-exldb_cw.xlsx

0% [##############################] 100% | ETA: 00:00:00
Total time elapsed: 00:00:00

Title: Writing activities to SQLite3 database:
  Started: 04/26/2022 12:12:42
  Finished: 04/26/2022 12:12:42
  Total time elapsed: 00:00:00
  CPU %: 133.00
  Memory %: 1.45
Created database: exldb_cw
```

Checking if the imports went well:

List databases:

[22]:
```
bw.databases
```

[22]:
```
Databases dictionary with 8 object(s):
        biosphere3
        ecoinvent 3.7 cut-off
        exldb_alu
        exldb_cw
        exldb_cw_eol
        exldb_igu
        exldb_sand
```

```
exldb_spacers
```

Checking Excel database:

Deleting a database, if needed:

3.3 Navigating through the Databases

Assigning a variable to each database to ease their use:

[23]:
```
eib3db = bw.Database('biosphere3')

eicutdb = bw.Database('ecoinvent 3.7 cut-off')

exldb_alu = bw.Database('exldb_alu')
exldb_igu = bw.Database('exldb_igu')
exldb_cw = bw.Database('exldb_cw')
exldb_spacers = bw.Database('exldb_spacers')
exldb_cw_eol = bw.Database('exldb_cw_eol')
```

Searching for a specific activity:

4 Defining the Parameters

4.1 Overview

Checking the total number of parameters:

[24]:
```
len(parameters)
```

[24]:
```
62
```

Listing the parameters:

[25]:
```
if len(ProjectParameter.select()) != 0:
    print("\033[1m", "Project parameters:", "\033[0m")
    for p in ProjectParameter.select():
        print(p.name, ":", p.amount)

print("------")
print("\033[1m", "Database parameters:", "\033[0m")
for p in DatabaseParameter.select():
    print(p.database, " > ", p.name, ":", round(p.amount, 2))
```

```
 Project parameters:
param_g_density : 2.5
param_t_lsg : 10.0
param_t_tsg : 10.0
param_n_pvb : 2.0
param_d1 : 125.0
```

```
param_t_g_ext : 8.0
param_t_g_mid_tg : 6.0
param_t_g_uncoated_int : 8.0
------
```

Database parameters:

```
exldb_cw_eol  >  param_g_density : 2.5
exldb_cw_eol  >  param_t_lsg : 10.0
exldb_cw_eol  >  param_t_tsg : 10.0
exldb_cw_eol  >  param_n_pvb : 2.0
exldb_cw_eol  >  param_d1 : 125.0
exldb_cw_eol  >  param_t_g_ext : 8.0
exldb_cw_eol  >  param_t_g_mid_tg : 6.0
exldb_cw_eol  >  param_t_g_uncoated_int : 8.0
exldb_cw_eol  >  param_m_sg_g : 25.0
exldb_cw_eol  >  param_m_sg_alu : 3.31
exldb_cw_eol  >  param_m_sg_low_wood : 0.09
exldb_cw_eol  >  param_m_sg_low_silicone : 0.15
exldb_cw_eol  >  param_m_sg_high_epdm : 0.55
exldb_cw_eol  >  param_m_dg_g : 45.0
exldb_cw_eol  >  param_m_dg_alu : 3.47
exldb_cw_eol  >  param_m_dg_low_wood : 0.09
exldb_cw_eol  >  param_m_dg_low_silicone : 0.15
exldb_cw_eol  >  param_m_dg_high_epdm : 0.67
exldb_cw_eol  >  param_m_tg_g : 60.0
exldb_cw_eol  >  param_m_tg_alu : 3.79
exldb_cw_eol  >  param_m_tg_epdm : 0.78
exldb_cw_eol  >  param_m_ccf_g : 70.0
exldb_cw_eol  >  param_m_ccf_alu : 13.22
exldb_cw_eol  >  param_m_ccf_epdm : 1.95
exldb_cw_eol  >  param_m_vacuum_g : 45.0
exldb_cw_eol  >  param_m_smart_g : 45.0
exldb_cw_eol  >  param_m_smart_elec : 0.94
exldb_cw_eol  >  param_m_dsf_g : 70.0
exldb_cw_eol  >  param_m_dsf_alu : 6.79
exldb_cw_eol  >  param_m_dsf_epdm : 1.22
exldb_cw_eol  >  param_d2 : 130.0
exldb_cw_eol  >  param_d3 : 50.0
exldb_cw  >  param_natural_gas : 0.0
exldb_cw  >  param_elec_use : 0.0
exldb_cw  >  param_servicelife : 1.0
exldb_cw  >  param_lifespan : 40.0
exldb_cw  >  param_ext_shdg_device : 0.0
exldb_cw  >  param_int_shdg_device : 0.0
exldb_cw  >  param_thermal_curtain : 0.0
exldb_cw  >  param_sg : 0.0
exldb_cw  >  param_sg_coated : 0.0
exldb_cw  >  param_dg : 0.0
exldb_cw  >  param_dg_coated : 0.0
```

```
exldb_cw  >  param_dg_coated_krypton : 0.0
exldb_cw  >  param_dg_2coatings : 0.0
exldb_cw  >  param_tg_coated : 0.0
exldb_cw  >  param_tg_2coatings : 0.0
exldb_cw  >  param_tg_2coatings_krypton : 0.0
exldb_cw  >  param_tg_2coatings_xenon : 0.0
exldb_cw  >  param_ccf : 0.0
exldb_cw  >  param_dg_vacuum : 0.0
exldb_cw  >  param_dg_smart : 0.0
exldb_cw  >  param_dsf : 0.0
```

4.2 Activating the Parameters

This step consists in asking Brightway2 to activate the exchanges and their formulas, when the latter rely on parameters:

[26]:
```
# Including formula-defined exchanges of activities to a new group,
# for igu production:
for act in exldb_igu:
    parameters.add_exchanges_to_group("igu_param_group", act)
```

[27]:
```
# Initialising a list of activity data from the exldb_cw_eol database:
ls_act_data_cw_eol = []

n_code = 0
for obj in DatabaseParameter.select().where(
        DatabaseParameter.database == "exldb_cw_eol"):
    ls_act_data_cw_eol.append({'name': obj.name, 'amount': obj.amount,
                               'formula': obj.formula, 'database': obj.database,
                              'code': "p_eol_"+str(n_code)})
    n_code += 1

# Entering multiple parameters and overwriting the existing ones
# in the parameter group:
parameters.new_activity_parameters(
    ls_act_data_cw_eol, "cw_eol_param_group", overwrite=True)

# Including formula-defined exchanges of activities to a new group,
# for the end-of-life dataset:
for act in exldb_cw_eol:
    parameters.add_exchanges_to_group("cw_eol_param_group", act)
```

[28]:
```
# Same action as previously, but fot the curtain wall database:
ls_act_data_cw = []

n_code = 0
for obj in DatabaseParameter.select().where(
        DatabaseParameter.database == "exldb_cw"):
```

```
    ls_act_data_cw.append({'name': obj.name, 'amount': obj.amount,
                           'formula': obj.formula, 'database': obj.database,
                           'code': "p_"+str(n_code)})
    n_code += 1

parameters.new_activity_parameters(
    ls_act_data_cw, "cw_use_param_group", overwrite=True)

for act in exldb_cw:
    parameters.add_exchanges_to_group("cw_use_param_group", act)
```

And finally, the exchanges are recalculated on the basis of the "activated" formula:

[29]:
```
ActivityParameter.recalculate_exchanges("igu_param_group")
ActivityParameter.recalculate_exchanges("cw_use_param_group")
ActivityParameter.recalculate_exchanges("cw_eol_param_group")
```

If needed, delete the parameters:

5 LCIA Methods

This section defines the LCIA methods. They are all based on ILCD 2.0 2018 midpoint, a version by Ecoinvent of the Environmental Footprint (EF) midpoint method. Three groups are created according to the number of impact indicators included: only global warming potential, nine, or sixteen.

For further information regarding the EF midpoint method: Fazio et al., 2018. 'Supporting Information to the Characterisation Factors of Recommended EF Life Cycle Impact Assessment Methods: New Methods and Differences with ILCD.' Luxembourg: The European Commission and the Joint Research Centre. http://publications.europa.eu/publication/manifestation_identifier/PUB_KJNA28888ENN.

Creating list of methods:

[30]:
```
method_ilcd_gwp = (
    'ILCD 2.0 2018 midpoint', 'climate change', 'climate change total')
```

[31]:
```
ls_method_small = [
    ('ILCD 2.0 2018 midpoint', 'climate change', 'climate change total'),
    ('ILCD 2.0 2018 midpoint', 'ecosystem quality', 'freshwater ecotoxicity'),
    ('ILCD 2.0 2018 midpoint', 'ecosystem quality',
     'freshwater and terrestrial acidification'),
    ('ILCD 2.0 2018 midpoint', 'ecosystem quality', 'freshwater␣
↪eutrophication'),
    ('ILCD 2.0 2018 midpoint', 'ecosystem quality', 'terrestrial␣
↪eutrophication'),
    ('ILCD 2.0 2018 midpoint', 'human health', 'ozone layer depletion'),
    ('ILCD 2.0 2018 midpoint', 'human health', 'photochemical ozone creation'),
    ('ILCD 2.0 2018 midpoint', 'resources', 'fossils'),
```

```
    ('ILCD 2.0 2018 midpoint', 'resources', 'land use')
]
```

[32]:
```
ls_method_full = [
    ('ILCD 2.0 2018 midpoint', 'climate change', 'climate change total'),
    ('ILCD 2.0 2018 midpoint', 'ecosystem quality', 'freshwater ecotoxicity'),
    ('ILCD 2.0 2018 midpoint', 'ecosystem quality',
     'freshwater and terrestrial acidification'),
    ('ILCD 2.0 2018 midpoint', 'ecosystem quality', 'freshwater␣
↪eutrophication'),
    ('ILCD 2.0 2018 midpoint', 'ecosystem quality', 'marine eutrophication'),
    ('ILCD 2.0 2018 midpoint', 'ecosystem quality', 'terrestrial␣
↪eutrophication'),
    ('ILCD 2.0 2018 midpoint', 'human health', 'non-carcinogenic effects'),
    ('ILCD 2.0 2018 midpoint', 'human health', 'carcinogenic effects'),
    ('ILCD 2.0 2018 midpoint', 'human health', 'ionising radiation'),
    ('ILCD 2.0 2018 midpoint', 'human health', 'ozone layer depletion'),
    ('ILCD 2.0 2018 midpoint', 'human health', 'photochemical ozone creation'),
    ('ILCD 2.0 2018 midpoint', 'human health', 'respiratory effects,␣
↪inorganics'),
    ('ILCD 2.0 2018 midpoint', 'resources', 'minerals and metals'),
    ('ILCD 2.0 2018 midpoint', 'resources', 'dissipated water'),
    ('ILCD 2.0 2018 midpoint', 'resources', 'fossils'),
    ('ILCD 2.0 2018 midpoint', 'resources', 'land use')
]
```

6 Weighting

The weighting step follows the European Environmental Footprint Methodology, which define weighting factors for each of the mid-point impact categories. See the following report:

Sala, Serenella, Alessandro Kim Cerutti, and Rana Pant. 'Development of a Weighting Approach for the Environmental Footprint'. Luxembourg: Publications Office of the European Union, 2018. https://ec.europa.eu/environment/eussd/smgp/documents/2018_JRC_Weighting_EF.pdf.

[33]:
```
# Matching impact labels between the Ecovinvent ILCD 2018 method
# and the report by Sala et al.:
dict_ilcd_to_weight = {
    ('climate change', 'climate change total'): (
        "Climate change"),
    ('human health', 'ozone layer depletion'): (
        "Ozone depletion"),
    ('human health', 'carcinogenic effects'): (
        "Human toxicity, cancer effects"),
    ('human health', 'non-carcinogenic effects'): (
        "Human toxicity, non-cancer effects"),
    ('human health', 'respiratory effects, inorganics'): (
```

```
        "Particulate matter"),
    ('human health', 'ionising radiation'): (
        "Ionizing radiation, human health"),
    ('human health', 'photochemical ozone creation'): (
        "Photochemical ozone formation, human health"),
    ('ecosystem quality', 'freshwater and terrestrial acidification'): (
        "Acidification"),
    ('ecosystem quality', 'terrestrial eutrophication'): (
        "Eutrophication, terrestrial"),
    ('ecosystem quality', 'freshwater eutrophication'): (
        "Eutrophication, freshwater"),
    ('ecosystem quality', 'marine eutrophication'): (
        "Eutrophication, marine"),
    ('ecosystem quality', 'freshwater ecotoxicity'): (
        "Ecotoxicity freshwater"),
    ('resources', 'land use'): (
        "Land use"),
    ('resources', 'dissipated water'): (
        "Water use"),
    ('resources', 'minerals and metals'): (
        "Resource use, minerals and metals"),
    ('resources', 'fossils'): (
        "Resource use, fossils")
}
```

[34]:
```
# List of weighting factors by impact category:
dict_weighting = {"Climate change": 21.06,
                  "Ozone depletion": 6.31,
                  "Human toxicity, cancer effects": 2.13,
                  "Human toxicity, non-cancer effects": 1.84,
                  "Particulate matter": 8.96,
                  "Ionizing radiation, human health": 5.01,
                  "Photochemical ozone formation, human health": 4.78,
                  "Acidification": 6.20,
                  "Eutrophication, terrestrial": 3.71,
                  "Eutrophication, freshwater": 2.80,
                  "Eutrophication, marine": 2.96,
                  "Ecotoxicity freshwater": 1.92,
                  "Land use": 7.94,
                  "Water use": 8.51,
                  "Resource use, minerals and metals": 7.55,
                  "Resource use, fossils": 8.32
                  }
```

[35]:
```
# Creating a DataFrame with the weighting factors:
df_weighting = pd.DataFrame.from_dict(dict_ilcd_to_weight, orient='index',
                                      columns=['Weighting factor'])
```

```python
for key, value in dict_weighting.items():
    df_weighting.loc[df_weighting['Weighting factor'] == key,
                     'Weighting factor'] = value

    df_weighting.index = pd.MultiIndex.from_tuples(
        df_weighting.index, names=['Category', 'Subcategory']
    )
```

[36]:
```python
df_weighting
```

[36]:
```
                                                            Weighting factor
Category          Subcategory
climate change    climate change total                                 21.06
human health      ozone layer depletion                                 6.31
                  carcinogenic effects                                  2.13
                  non-carcinogenic effects                              1.84
                  respiratory effects, inorganics                       8.96
                  ionising radiation                                    5.01
                  photochemical ozone creation                          4.78
ecosystem quality freshwater and terrestrial acidification               6.2
                  terrestrial eutrophication                            3.71
                  freshwater eutrophication                              2.8
                  marine eutrophication                                 2.96
                  freshwater ecotoxicity                                1.92
resources         land use                                              7.94
                  dissipated water                                      8.51
                  minerals and metals                                   7.55
                  fossils                                               8.32
```

7 A Comparative LCA of Flat Glass Panes and IGUs, Cradle-to-Gate

This section studies different types of flat glass and insulating glass units, comparing the main components and different designs to understand their contribution to environmental impact.

7.1 Flat Glass Production

A first LCA study focusing on the production of flat glass and its processing (laminated, toughened, coated…).

Listing the activities studied in this LCA:

[37]:
```python
# Unprocessed flat glass:
inv_fg = [act for act in exldb_igu
          if 'market for flat glass' in act['name']
          ]
```

```
inv_fg = sorted(inv_fg,
                key=lambda k: k['name']
                )
```

[38]:
```
# Processed flat glass:
ls_fg_processed = ['market for laminated safety glass',
                   'market for tempered safety glass',
                   'market for smart glass'
                   ]

inv_fg_processed = [act for act in exldb_igu
                    for n in ls_fg_processed
                    if n in act['name']
                    and "glazing" not in act['name']
                    ]

inv_fg_processed = sorted(inv_fg_processed,
                          key=lambda k: k['name']
                          )
```

[39]:
```
print("\033[1m",
      "List of activities related to flat glass production:", "\033[0m"
      )

for fg in (inv_fg and inv_fg_processed):
    print(fg['name'])
```

```
 List of activities related to flat glass production:
market for laminated safety glass
market for laminated safety glass, coated
market for smart glass
market for tempered safety glass
market for tempered safety glass, coated
```

Defining the functional unit per glass type:

[40]:
```
# Defining the functional unit for unprocessed flat glass,
# i.e., 25kg of glass to obtain a thickness of 10mm for 1m²:
fu_fg = 25

# Defining the functional unit for processed flat glass,
# i.e., 1m² with a thickness already defined as 10mm:
fu_fg_processed = 1
```

Conducting the LCIA:

[41]:
```python
# Creating a list where results will be saved:
impact_fg = []

# Calculating:
for act in inv_fg:
    lca = bw.LCA({act: fu_fg})
    lca.lci()
    for method in ls_method_small:
        lca.switch_method(method)
        lca.lcia()
        impact_fg.append((act["name"], act["location"],
                          method[1], lca.score,
                          bw.methods.get(method).get('unit')))

for act in inv_fg_processed:
    lca = bw.LCA({act: fu_fg_processed})
    lca.lci()
    for method in ls_method_small:
        lca.switch_method(method)
        lca.lcia()
        impact_fg.append((act["name"], act["location"],
                          method[1], lca.score,
                          bw.methods.get(method).get('unit')))
```

Creating a DataFrame with the LCIA results:

[42]:
```python
df_impact_fg = pd.DataFrame(impact_fg, columns=["Name",
                                                "Location",
                                                "Method",
                                                "Score",
                                                "Unit"]
                            )

df_impact_fg = (pd.pivot_table(df_impact_fg, index=["Name"],
                               columns=["Method", "Unit"],
                               values="Score"
                               )
                ).sort_values(("climate change", "kg CO2-Eq"), ascending=True)

df_impact_fg.index = df_impact_fg.index.str.replace('market for ', '')

df_impact_fg.round(2)
```

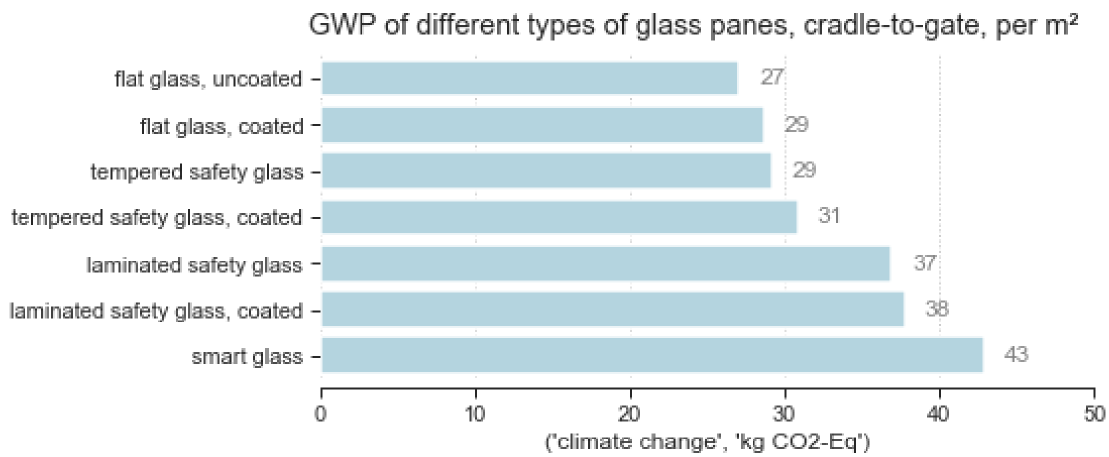
[42]:
```
Method                         climate change ecosystem quality          \
Unit                                kg CO2-Eq               CTU kg P-Eq
Name
flat glass, uncoated                    27.11              7.08    0.00
flat glass, coated                      28.71              9.15    0.00
```

```
tempered safety glass                     29.22            7.31    0.00
tempered safety glass, coated             30.86            9.41    0.00
laminated safety glass                    36.93           12.35    0.01
laminated safety glass, coated            37.75           13.40    0.01
smart glass                               42.92           16.68    0.01

Method                                              human health            \
Unit                          mol H+-Eq mol N-Eq   kg CFC-11. kg NMVOC-.
Name
flat glass, uncoated               0.22     0.60          0.0        0.14
flat glass, coated                 0.24     0.63          0.0        0.15
tempered safety glass              0.23     0.62          0.0        0.15
tempered safety glass, coated      0.24     0.65          0.0        0.15
laminated safety glass             0.26     0.69          0.0        0.17
laminated safety glass, coated     0.27     0.70          0.0        0.17
smart glass                        0.28     0.72          0.0        0.18

Method                         resources
Unit                           megajoule  points
Name
flat glass, uncoated              344.41  104.18
flat glass, coated                377.57  146.02
tempered safety glass             377.22  106.57
tempered safety glass, coated     411.04  149.25
laminated safety glass            623.93  196.82
laminated safety glass, coated    640.85  218.16
smart glass                       837.53  273.79
```

Displaying a bar chart showing the climate change potential of the different flat glass products:

[43]:
```python
fig, ax = plt.subplots(figsize=(7, 3))

g = sns.barplot(data=df_impact_fg,
                x=("climate change", "kg CO2-Eq"),
                y=df_impact_fg.index,
                color="lightblue", linewidth=1.5)

g.bar_label(g.containers[0], fmt="%.0f", padding=10, c='grey')

ax.yaxis.label.set_visible(False)
ax.grid(which='major', axis='x', linestyle=':', linewidth=1)

ax.set_xlim(0, 50)
plt.xticks(np.arange(0, 51, 10))

fig.suptitle(
    'GWP of different types of glass panes, cradle-to-gate, per m²')
```

```
sns.despine(left=True, offset=5)

plt.show()
```



Creating a DataFrame where the LCIA results are normalised to the highest value per impact category (i.e., $I_{max} = 1$):

[44]:
```
df_norm_impact_fg = df_impact_fg / df_impact_fg.max()
df_norm_impact_fg.round(2)
```

[44]:
```
Method                          climate change ecosystem quality          \
Unit                                 kg CO2-Eq                CTU kg P-Eq
Name
flat glass, uncoated                      0.63               0.42    0.30
flat glass, coated                        0.67               0.55    0.37
tempered safety glass                     0.68               0.44    0.30
tempered safety glass, coated             0.72               0.56    0.38
laminated safety glass                    0.86               0.74    0.59
laminated safety glass, coated            0.88               0.80    0.63
smart glass                               1.00               1.00    1.00

Method                                                 human health           \
Unit                            mol H+-Eq mol N-Eq   kg CFC-11. kg NMVOC-.
Name
flat glass, uncoated                 0.81     0.83         0.22        0.80
flat glass, coated                   0.86     0.88         0.23        0.84
tempered safety glass                0.83     0.86         0.24        0.83
tempered safety glass, coated        0.88     0.90         0.25        0.87
laminated safety glass               0.94     0.95         0.44        0.94
laminated safety glass, coated       0.96     0.97         0.45        0.96
smart glass                          1.00     1.00         1.00        1.00
```

```
Method                           resources
Unit                             megajoule points
Name
flat glass, uncoated                  0.41   0.38
flat glass, coated                    0.45   0.53
tempered safety glass                 0.45   0.39
tempered safety glass, coated         0.49   0.55
laminated safety glass                0.74   0.72
laminated safety glass, coated        0.77   0.80
smart glass                           1.00   1.00
```

[45]:
```python
# Normalised results, but without smart glass:
df_norm_impact_wo_smartg = (
    df_impact_fg.drop("smart glass", axis=0) /
    df_impact_fg.drop("smart glass", axis=0).max()
)
df_norm_impact_wo_smartg.round(2)
```

[45]:
```
Method                           climate change ecosystem quality          \
Unit                                  kg CO2-Eq               CTU kg P-Eq
Name
flat glass, uncoated                       0.72              0.53    0.47
flat glass, coated                         0.76              0.68    0.59
tempered safety glass                      0.77              0.55    0.48
tempered safety glass, coated              0.82              0.70    0.60
laminated safety glass                     0.98              0.92    0.94
laminated safety glass, coated             1.00              1.00    1.00

Method                                                 human health          \
Unit                             mol H+-Eq mol N-Eq   kg CFC-11. kg NMVOC-.
Name
flat glass, uncoated                  0.84     0.86         0.49        0.83
flat glass, coated                    0.89     0.90         0.51        0.88
tempered safety glass                 0.86     0.88         0.54        0.86
tempered safety glass, coated         0.91     0.92         0.56        0.90
laminated safety glass                0.97     0.98         0.99        0.98
laminated safety glass, coated        1.00     1.00         1.00        1.00

Method                           resources
Unit                             megajoule points
Name
flat glass, uncoated                  0.54   0.48
flat glass, coated                    0.59   0.67
tempered safety glass                 0.59   0.49
tempered safety glass, coated         0.64   0.68
laminated safety glass                0.97   0.90
```

```
laminated safety glass, coated      1.00   1.00
```

Now, same calculation, but using the MultiLCA class with the full list of impact categories, i.e., the 16 indicators from the ILCD midpoint method:

[46]:
```python
# Defining the system with the same activities and functional units as above:
mlca_syst_fg = []

for act in inv_fg:
    mlca_syst_fg.append({act.key: fu_fg})

for act in inv_fg_processed:
    mlca_syst_fg.append({act.key: fu_fg_processed})
```

Conducting the LCIA:

[47]:
```python
bw.calculation_setups['calculation_setup'] = {'inv': mlca_syst_fg,
                                              'ia': ls_method_full}


mlca = bw.MultiLCA('calculation_setup')

# Saving the results in a DataFrame:
df_impact_mlca_fg = pd.DataFrame(data=mlca.results, columns=mlca.methods)
```

Reorganisating a bit the DataFrame:

[48]:
```python
# Listing the activities concerned:
activities = [(get_activity(key), amount)
              for dct in mlca.func_units
              for key, amount in dct.items()
              ]

# Creating a DataFrame with activities info:
df_fu = pd.DataFrame([(x['name'], x['database'], x['code'],
                       x['location'], x['unit'], y)
                      for x, y in activities],
                     columns=('Database', 'Code', 'Name',
                              'Location', 'Unit', 'Amount')
                     )

# Merging activities info and LCIA results:
df_impact_mlca_fg = pd.concat([df_fu, df_impact_mlca_fg], axis=1
                             ).set_index("Name").drop(
    ["Database", "Code", "Location", "Unit", "Amount"], axis=1
)

# Renaming the columns with multi-index, according to LCIA method:
df_impact_mlca_fg.columns = pd.MultiIndex.from_tuples(
    df_impact_mlca_fg.columns, names=(
```

```
        'Method', 'Category', 'Subcategory')
)

# Sorting results:
df_impact_mlca_fg = df_impact_mlca_fg.sort_values(
    ('ILCD 2.0 2018 midpoint', 'climate change', 'climate change total'),
    ascending=True)
```

[49]:
```
with pd.option_context("display.max_rows", None,
                       "display.max_columns", None,
                       "display.float_format", '{:12.1e}'.format):
    display(df_impact_mlca_fg["ILCD 2.0 2018 midpoint"])
```

```
Category                      climate change      ecosystem quality  \
Subcategory             climate change total freshwater ecotoxicity
Name
market_glass_uncoated                2.7e+01                7.1e+00
market_glass_coated                  2.9e+01                9.1e+00
market_tsg                           2.9e+01                7.3e+00
market_tsg_coated                    3.1e+01                9.4e+00
market_lsg                           3.7e+01                1.2e+01
market_lsg_coated                    3.8e+01                1.3e+01
market_smartglass                    4.3e+01                1.7e+01

Category                                                         \
Subcategory             freshwater and terrestrial acidification
Name
market_glass_uncoated                                    2.2e-01
market_glass_coated                                      2.4e-01
market_tsg                                               2.3e-01
market_tsg_coated                                        2.4e-01
market_lsg                                               2.6e-01
market_lsg_coated                                        2.7e-01
market_smartglass                                        2.8e-01

Category                                                                \
Subcategory             freshwater eutrophication marine eutrophication
Name
market_glass_uncoated                     2.5e-03               5.2e-02
market_glass_coated                       3.2e-03               5.4e-02
market_tsg                                2.6e-03               5.3e-02
market_tsg_coated                         3.2e-03               5.6e-02
market_lsg                                5.0e-03               6.0e-02
market_lsg_coated                         5.4e-03               6.2e-02
market_smartglass                         8.5e-03               6.5e-02

Category                                                      human health  \
Subcategory             terrestrial eutrophication non-carcinogenic effects
```

```
Name
market_glass_uncoated                    6.0e-01                  1.0e-06
market_glass_coated                      6.3e-01                  1.4e-06
market_tsg                               6.2e-01                  1.0e-06
market_tsg_coated                        6.5e-01                  1.4e-06
market_lsg                               6.9e-01                  1.9e-06
market_lsg_coated                        7.0e-01                  2.1e-06
market_smartglass                        7.2e-01                  3.0e-06

Category                                                        \
Subcategory           carcinogenic effects ionising radiation
Name
market_glass_uncoated              2.4e-07            3.1e+00
market_glass_coated                3.4e-07            3.9e+00
market_tsg                         2.5e-07            3.2e+00
market_tsg_coated                  3.5e-07            4.0e+00
market_lsg                         4.3e-07            9.2e+00
market_lsg_coated                  4.9e-07            9.6e+00
market_smartglass                  5.8e-07            1.7e+01

Category                                                                   \
Subcategory           ozone layer depletion photochemical ozone creation
Name
market_glass_uncoated               2.7e-06                      1.4e-01
market_glass_coated                 2.9e-06                      1.5e-01
market_tsg                          3.0e-06                      1.5e-01
market_tsg_coated                   3.2e-06                      1.5e-01
market_lsg                          5.6e-06                      1.7e-01
market_lsg_coated                   5.6e-06                      1.7e-01
market_smartglass                   1.3e-05                      1.8e-01

Category                                                         resources  \
Subcategory           respiratory effects, inorganics minerals and metals
Name
market_glass_uncoated                         2.2e-06            6.8e-04
market_glass_coated                           2.4e-06            1.0e-03
market_tsg                                    2.3e-06            7.0e-04
market_tsg_coated                             2.4e-06            1.0e-03
market_lsg                                    2.6e-06            9.3e-04
market_lsg_coated                             2.6e-06            1.1e-03
market_smartglass                             2.6e-06            4.3e-03

Category
Subcategory           dissipated water      fossils     land use
Name
market_glass_uncoated          6.0e+00      3.4e+02      1.0e+02
market_glass_coated            6.8e+00      3.8e+02      1.5e+02
market_tsg                     6.2e+00      3.8e+02      1.1e+02
```

```
market_tsg_coated                6.9e+00      4.1e+02      1.5e+02
market_lsg                       1.2e+01      6.2e+02      2.0e+02
market_lsg_coated                1.2e+01      6.4e+02      2.2e+02
market_smartglass                1.4e+01      8.4e+02      2.7e+02
```

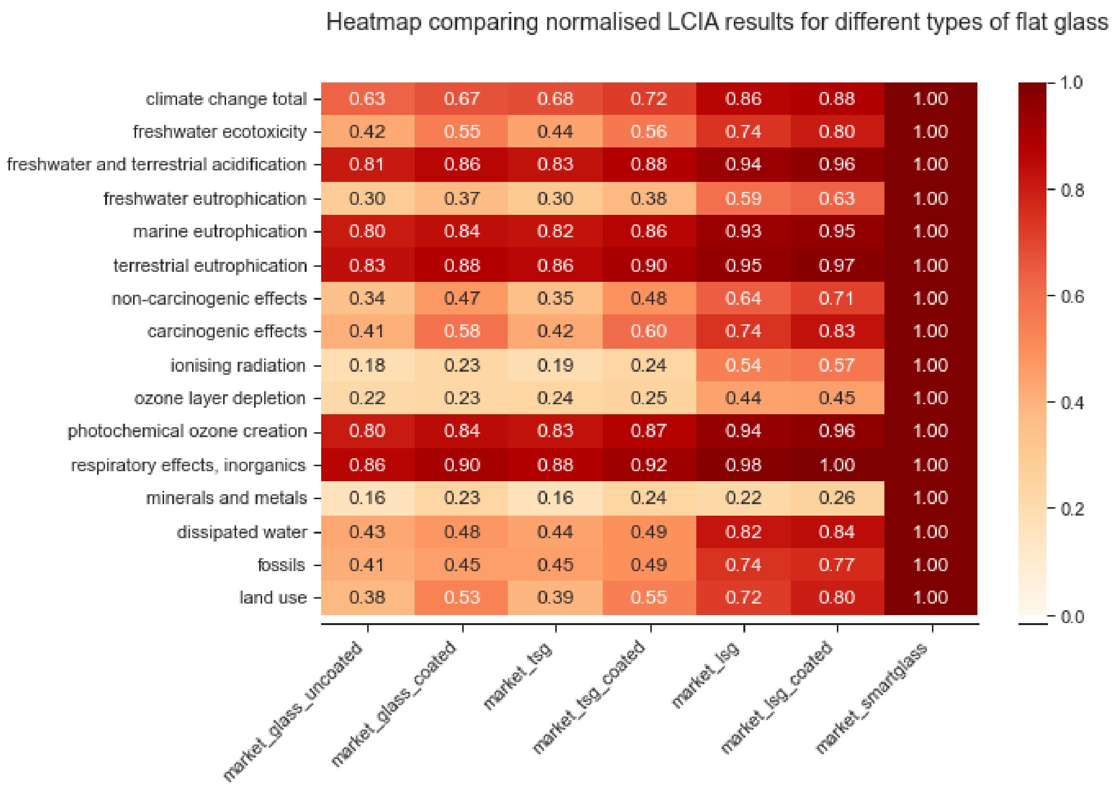
[50]:
```
df_impact_mlca_fg.to_csv('outputs\lca_table\df_impact_mlca_fg.csv')
```

Creating a DataFrame where the LCIA results are normalised to the highest value per impact category (i.e., $I_{max} = 1$):

[51]:
```
df_norm_impact_mlca_fg = df_impact_mlca_fg / df_impact_mlca_fg.max()

# Reorganising the DataFrame columns:
df_norm_impact_mlca_fg.columns = (
    df_norm_impact_mlca_fg.columns.droplevel([0, 1])
)
```

Displaying a heatmap with the normalised results (1 = maximum impact):

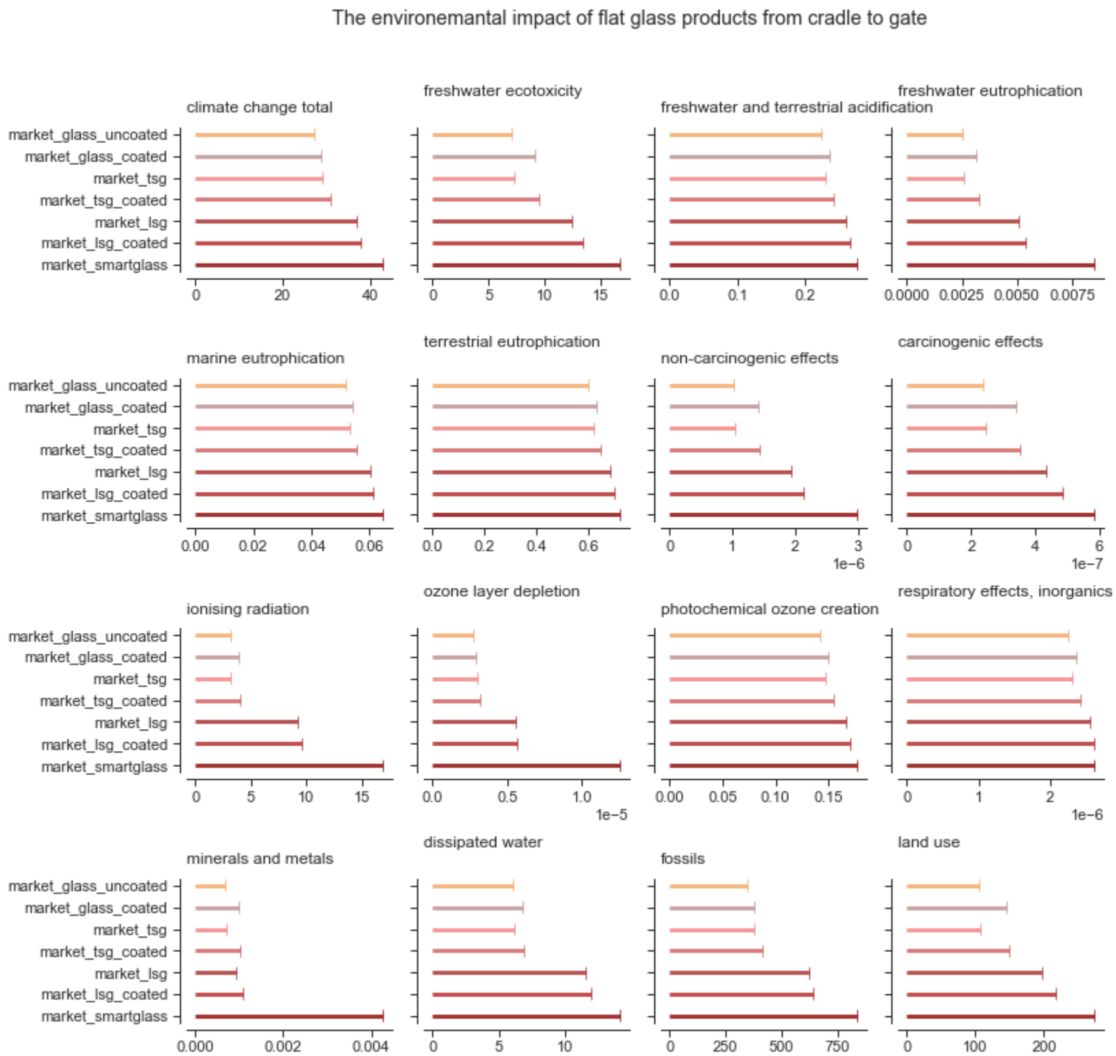
[52]:
```
fig, ax = plt.subplots(figsize=(9, 6))

df_plot = df_norm_impact_mlca_fg.T

ax = sns.heatmap(df_plot, cmap="OrRd", vmin=0, vmax=1, annot=True, fmt='.2f')

ax.yaxis.label.set_visible(False)
ax.xaxis.label.set_visible(False)

fig.suptitle(
    'Heatmap comparing normalised LCIA results'
    ' for different types of flat glass')

sns.despine(left=True, offset=5)

for tick in ax.get_xticklabels():
    tick.set_rotation(45)
    tick.set_ha('right')

plt.show()
```



Displaying a chart giving an overview of the environmental impact of each flat glass product according to each of the 16 indicators:

[53]:
```
fig, axes = plt.subplots(nrows=4, ncols=4,
                         sharex=False, sharey=True,
                         figsize=(12, 12))

df_plot = df_impact_mlca_fg.copy()
df_plot.columns = (df_impact_mlca_fg.columns.droplevel([0, 1]))

c = ["sandybrown", "rosybrown", "lightcoral", "indianred",
     "brown", "firebrick", "darkred"]

n = 0

for row in range(4):
    for col in range(4):
        col_name = df_plot.columns[n]
        ax = axes[row][col]

        ax.hlines(y=df_plot.index, xmin=0, xmax=df_plot[col_name],
                  linewidth=3, colors=c, alpha=0.8)
```

```
        sns.scatterplot(y=df_plot.index, x=df_plot[col_name],
                        hue=df_plot.index, s=80, marker="|",
                        palette=c, ax=ax)

        if (n % 2) == 0:
            ax.set_title(col_name, y=1.05, x=0,
                         ha='left', multialignment='left')
        else:
            ax.set_title(col_name, y=1.17, x=0,
                         ha='left', multialignment='left')

        ax.xaxis.label.set_visible(False)
        ax.yaxis.label.set_visible(False)

        ax.get_legend().remove()

        n += 1

fig.subplots_adjust(wspace=0.15, hspace=0.75)

fig.suptitle(
    'The environemantal impact of flat glass products from cradle to gate'
)
sns.despine(offset=5)

if export:
    # Save image:
    fig.savefig(os.path.join(path_img, 'FlatGlass_FullLCIA.png'),
                dpi=600, bbox_inches='tight')
    fig.savefig(os.path.join(path_img, 'FlatGlass_FullLCIA.pdf'),
                bbox_inches='tight')

plt.show()
```



Weighted environmental impact:

Comparing the different types of glass pane according to a single indicator calculated using the PEF weighting factors:

[54]:
```
# Defining a new DataFrame with the weighted values,
# i.e., multiplication of the impacts by df_weighting:
df_weighted_fg = pd.DataFrame(
    (df_impact_mlca_fg["ILCD 2.0 2018 midpoint"]
     .multiply(df_weighting["Weighting factor"].T, axis=1) / 100
     ).sum(axis=1), columns=['Weighted impact']
)

df_weighted_fg = df_weighted_fg.sort_values("Weighted impact",
                                            ascending=True
```

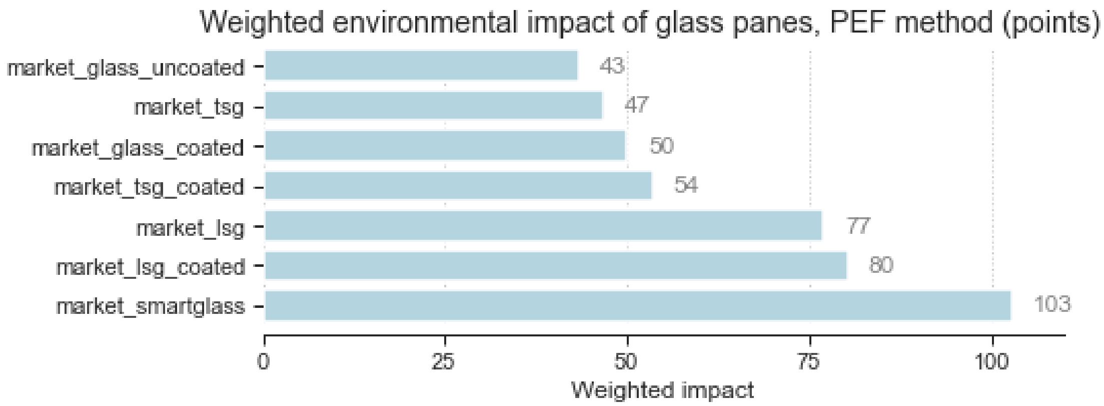
```python
                                          )

df_weighted_fg
```

[54]:

```
                         Weighted impact
Name
market_glass_uncoated          43.487143
market_tsg                     46.874804
market_glass_coated            50.047313
market_tsg_coated              53.566178
market_lsg                     77.051885
market_lsg_coated              80.397571
market_smartglass             102.887264
```

[55]:

```python
# Displaying a barplot figure with the weighted results:
fig, ax = plt.subplots(figsize=(7, 2.5))

g = sns.barplot(data=df_weighted_fg,
                x="Weighted impact",
                y=df_weighted_fg.index,
                color="lightblue", linewidth=1.5)

g.bar_label(g.containers[0], fmt="%.0f", padding=10, c='grey')

ax.yaxis.label.set_visible(False)
ax.grid(which='major', axis='x', linestyle=':', linewidth=1)

ax.set_xlim(0, 110)
plt.xticks(np.arange(0, 101, 25))

fig.suptitle('Weighted environmental impact of glass panes,'
             ' PEF method (points)')
sns.despine(left=True, offset=5)

if export:
    # Save image:
    fig.savefig(os.path.join(path_img, 'FlatGlass_WeightedLCIA.png'),
                dpi=600, bbox_inches='tight')
    fig.savefig(os.path.join(path_img, 'FlatGlass_WeightedLCIA.pdf'),
                bbox_inches='tight')

plt.show()
```



7.2 A Comparative Analysis of Spacers, Sealants and Insulating Gases, Cradle-to-Gate

7.2.1 Comparative Analysis of Spacers

Selecting the activities and defining the functional unit:

[56]:
```
# List of IGUs (production activities) with different types of spacer and␣
↪sealant:
inv_spacers = [act for act in bw.Database("exldb_spacers")
               if 'krypton' not in act['name']
               and 'xenon' not in act['name']
               and 'air' not in act['name']]

# 1 m² of IGU:
fu_spacers = [{igu: 1} for igu in inv_spacers]
```

[57]:
```
print("\033[1m", "List of the activities assessed:", "\033[0m")

for fu in fu_spacers:
    for key, value in fu.items():
        print(key["name"])
```

```
 List of the activities assessed:
double glazing production, single-seal aluminium, argon
double glazing production, dual-seal composite plastic, argon
double glazing production, composite with corrugated metal, argon
double glazing production, thermoplastic PIB, argon
double glazing production, thermally broken aluminium, argon
double glazing production, without spacer, argon
double glazing production, dual-seal steel, argon
double glazing production, dual-seal aluminium, argon
double glazing production, slicone foam, argon
double glazing production, epdm foam, argon
```

Conducting the LCIA:

[58]:
```
impact_spacers = []

for igu in inv_spacers:
    lca = bw.LCA({igu: 1})
    lca.lci()
    for method in ls_method_full:
        lca.switch_method(method)
        lca.lcia()
        impact_spacers.append((igu["name"], igu["location"],
                               method[1], method[2], lca.score,
                               bw.methods.get(method).get('unit'))
                              )
```

Creating a DataFrame with the LCIA results:

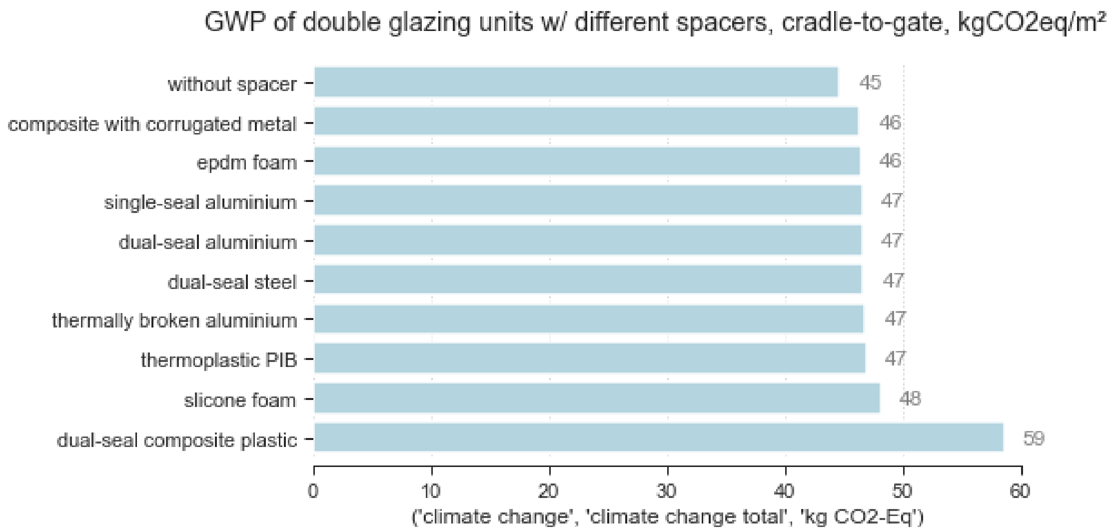
[59]:
```
# Creating the DataFrame:
df_impact_spacers = pd.DataFrame(
    impact_spacers,
    columns=["Name", "Location", "Category", "Subcategory", "Score", "Unit"]
)

# And reorganising it:
df_impact_spacers = pd.pivot_table(
    df_impact_spacers, index=["Name"],
    columns=["Category", "Subcategory", "Unit"], values="Score"
)

df_impact_spacers = df_impact_spacers.sort_values(
    ("climate change", "climate change total", "kg CO2-Eq"), ascending=True
)

# Simplifying the index:
df_impact_spacers.index = (df_impact_spacers.index
                           .str.replace('double glazing production, ', '')
                           .str.replace(', argon', '')
                           )
```

[61]:
```
df_impact_spacers.to_csv('outputs\lca_table\df_impact_spacers.csv')
```

Displaying a bar chart showing the climate change potential of the different flat glass products:

[62]:
```
fig, ax = plt.subplots(figsize=(7, 4))

g = sns.barplot(data=df_impact_spacers,
                x=("climate change", "climate change total", "kg CO2-Eq"),
                y=df_impact_spacers.index,
                color="lightblue", linewidth=1.5)
```

```
g.bar_label(g.containers[0], fmt="%.0f", padding=10, c='grey')

ax.yaxis.label.set_visible(False)
ax.grid(which='major', axis='x', linestyle=':', linewidth=1)

ax.set_xlim(0, 60)
plt.xticks(np.arange(0, 61, 10))

fig.suptitle(
    'GWP of double glazing units w/ different spacers,'
    ' cradle-to-gate, kgCO2eq/m²')
sns.despine(left=True, offset=5)
plt.show()
```



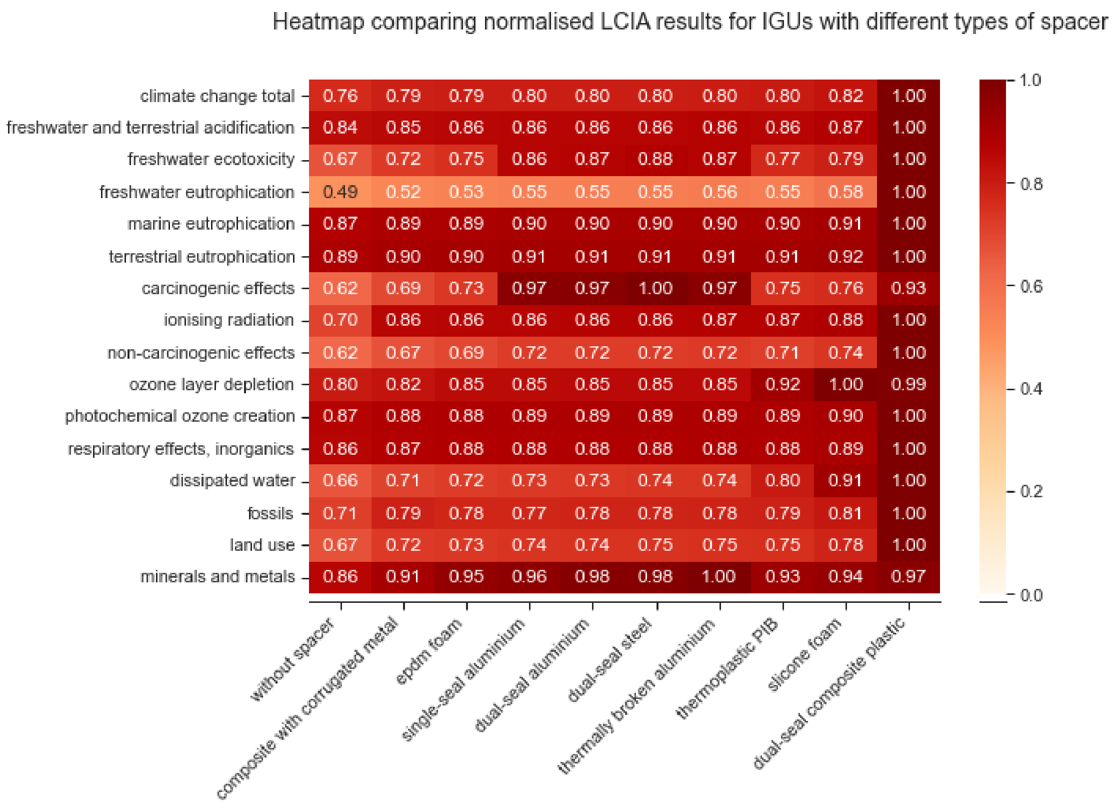
Normalising the results according to the highest value:

```
[63]: df_norm_impact_spacers = df_impact_spacers / df_impact_spacers.max()
```

Displaying a heatmap with the normalised results (1 = maximum impact):

```
[64]: fig, ax = plt.subplots(figsize=(9, 6))

y_axis_labels = []
for label in df_norm_impact_spacers.columns:
    y_axis_labels.append(label[1])

df_plot = df_norm_impact_spacers.T

ax = sns.heatmap(df_plot, cmap="OrRd", vmin=0, vmax=1, annot=True, fmt='.2f',
```

```
                 yticklabels=y_axis_labels)

ax.yaxis.label.set_visible(False)
ax.xaxis.label.set_visible(False)

fig.suptitle(
    'Heatmap comparing normalised LCIA results'
    ' for IGUs with different types of spacer')
sns.despine(left=True, offset=5)

for tick in ax.get_xticklabels():
    tick.set_rotation(45)
    tick.set_ha('right')

plt.show()
```

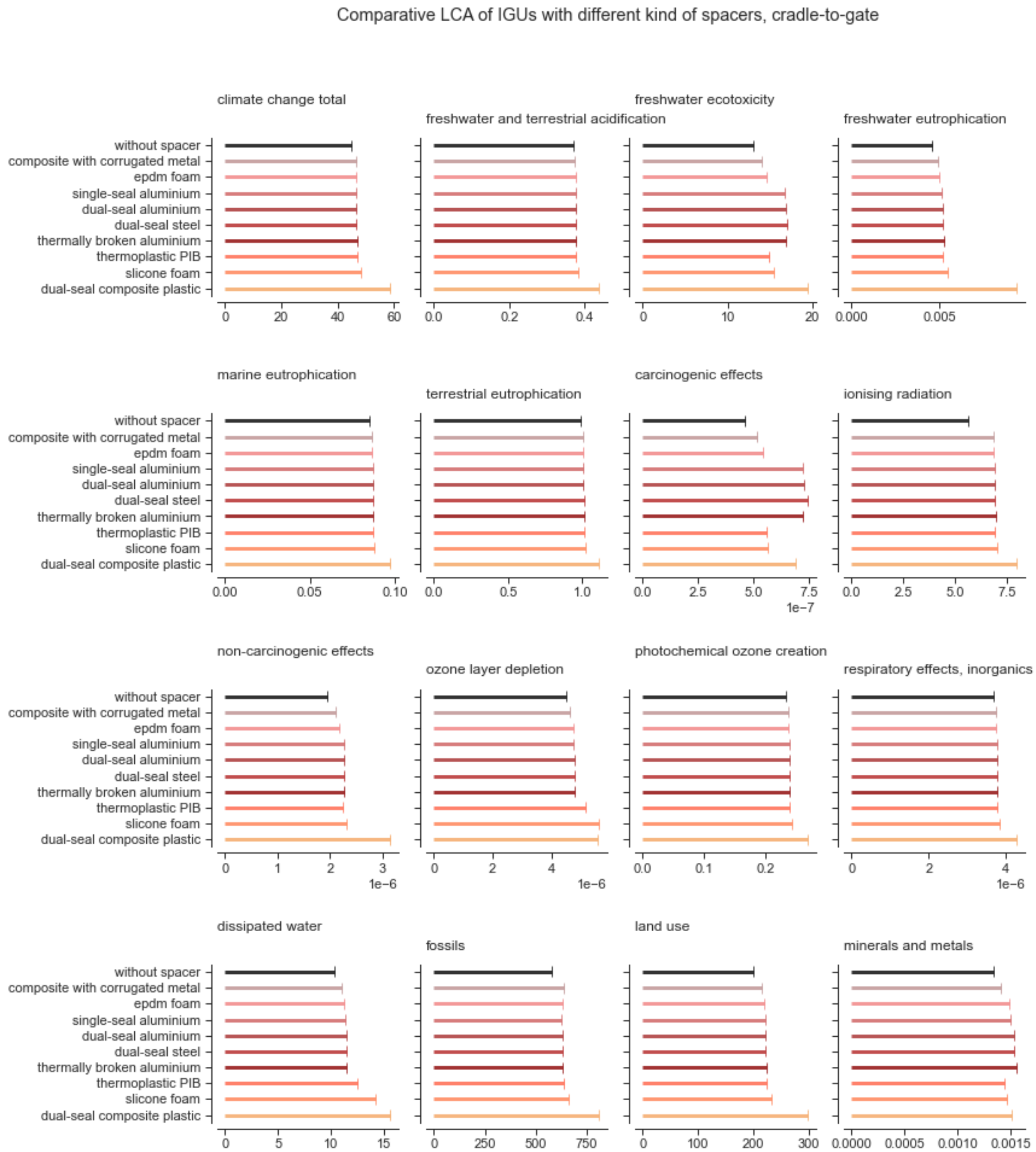


Displaying the full LCIA results:

```
[65]: fig, axes = plt.subplots(nrows=4, ncols=4,
                               sharex=False, sharey=True,
                               figsize=(12, 15))
```

```
c = ["black", "rosybrown", "lightcoral", "indianred",
     "brown", "firebrick", "darkred",
     "tomato", "coral", "sandybrown"]

n = 0

for row in range(4):
    for col in range(4):
        col_name = df_impact_spacers.columns[n]
        ax = axes[row][col]

        ax.hlines(y=df_impact_spacers.index,
                  xmin=0, xmax=df_impact_spacers[col_name],
                  linewidth=3, colors=c, alpha=0.8
                  )

        sns.scatterplot(y=df_impact_spacers.index,
                        x=df_impact_spacers[col_name],
                        hue=df_impact_spacers.index,
                        s=80, marker="|", palette=c, ax=ax
                        )

        if (n % 2) == 0:
            ax.set_title(col_name[1], y=1.17, x=0,
                         ha='left', multialignment='left')
        else:
            ax.set_title(col_name[1], y=1.05, x=0,
                         ha='left', multialignment='left')

        ax.xaxis.label.set_visible(False)
        ax.yaxis.label.set_visible(False)

        ax.get_legend().remove()

        n += 1

fig.subplots_adjust(wspace=0.15, hspace=0.75)

fig.suptitle(
    'Comparative LCA of IGUs with different kind of spacers, cradle-to-gate')
sns.despine(offset=5)

if export:
    # Save image:
    fig.savefig(os.path.join(path_img, 'IGU_Spacers_FullLCIA.png'),
                dpi=600, bbox_inches='tight')
    fig.savefig(os.path.join(path_img, 'IGU_Spacers_FullLCIA.pdf'),
```

```
                 bbox_inches='tight')

plt.show()
```



Weighted environmental impact:

Weighting the LCIA results according to the PEF weighting factors:

[66]:
```
# First, dropping the unit row index to ease the calculation:
df_weighted_spacers = df_impact_spacers.copy()
```

```python
df_weighted_spacers.columns = df_weighted_spacers.columns.droplevel(2)

# Defining a new DataFrame with the weighted values,
# i.e., multiplication of the impacts by df_weighting:
df_weighted_spacers = pd.DataFrame(
    (df_weighted_spacers.multiply(
        df_weighting["Weighting factor"].T, axis=1) / 100
     ).sum(axis=1), columns=['Weighted impact']
)

df_weighted_spacers = df_weighted_spacers.sort_values("Weighted impact",
                                                      ascending=True
                                                      )


df_weighted_spacers
```

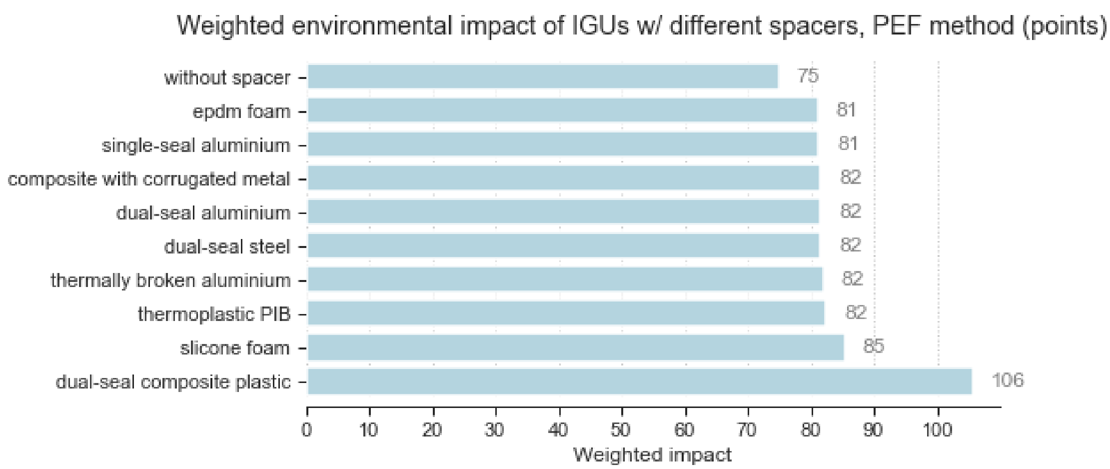
[66]:

```
                                  Weighted impact
Name
without spacer                          74.934996
epdm foam                               81.205578
single-seal aluminium                   81.340000
composite with corrugated metal         81.558729
dual-seal aluminium                     81.623840
dual-seal steel                         81.657879
thermally broken aluminium              82.080301
thermoplastic PIB                       82.420927
slicone foam                            85.408988
dual-seal composite plastic            105.655952
```

[67]:

```python
# Displaying a barplot figure with the weighted results:
fig, ax = plt.subplots(figsize=(7, 3.5))

g = sns.barplot(data=df_weighted_spacers,
                x="Weighted impact",
                y=df_weighted_spacers.index,
                color="lightblue", linewidth=1.5)

g.bar_label(g.containers[0], fmt="%.0f", padding=10, c='grey')

ax.yaxis.label.set_visible(False)
ax.grid(which='major', axis='x', linestyle=':', linewidth=1)

ax.set_xlim(0, 110)
plt.xticks(np.arange(0, 101, 10))

fig.suptitle('Weighted environmental impact of IGUs w/ different spacers,'
             ' PEF method (points)')
```

```
sns.despine(left=True, offset=5)

if export:
    # Save image:
    fig.savefig(os.path.join(path_img, 'IGU_Spacers_WeightedLCIA.png'),
                dpi=600, bbox_inches='tight')
    fig.savefig(os.path.join(path_img, 'IGU_Spacers_WeightedLCIA.pdf'),
                bbox_inches='tight')

plt.show()
```



7.2.2 Comparative Analysis of Insulating Gases

Listing the activities and defining the functional unit

[68]:
```
# List of the production activities of similar IGU,
# w/ different insulating gases:
inv_gas = [act for act in bw.Database("exldb_spacers")
           if 'thermally broken aluminium' in act['name']
           ]

# 1 m² of IGU:
fu_gas = [{igu: 1} for igu in inv_gas]
```

[69]:
```
print("\033[1m", "List of the activities assessed:", "\033[0m")

for fu in fu_gas:
    for key, value in fu.items():
        print(key["name"])
```

List of the activities assessed:
double glazing production, thermally broken aluminium, argon

```
double glazing production, thermally broken aluminium, krypton
double glazing production, thermally broken aluminium, xenon
double glazing production, thermally broken aluminium, air
```

Conducting the LCIA:

[70]:
```python
# Creating a list where results will be saved:
impact_gas = []

for igu in inv_gas:
    lca = bw.LCA({igu: 1})
    lca.lci()
    for method in ls_method_full:
        lca.switch_method(method)
        lca.lcia()
        impact_gas.append((igu["name"], igu["location"],
                           method[1], method[2], lca.score,
                           bw.methods.get(method).get('unit')))
```

Organising the results in a DataFrame:

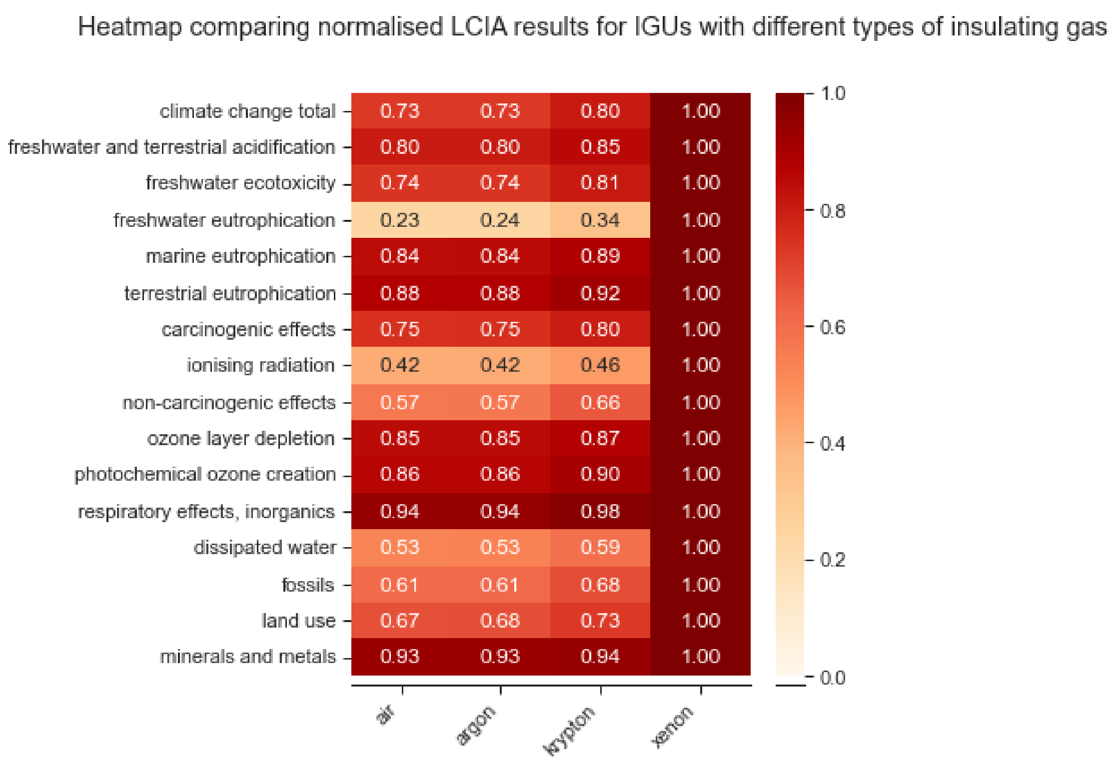
[71]:
```python
# Creating a DataFrame:
df_impact_gas = pd.DataFrame(
    impact_gas,
    columns=["Name", "Location", "Category", "Subcategory", "Score", "Unit"]
)

# Reorganising it:
df_impact_gas = pd.pivot_table(
    df_impact_gas, index=["Name"],
    columns=["Category", "Subcategory", "Unit"], values="Score"
)

# Sorting the values:
df_impact_gas = df_impact_gas.sort_values(
    ("climate change", "climate change total", "kg CO2-Eq"), ascending=True
)

# Simplifying the index:
df_impact_gas.index = (df_impact_gas.index
                       .str.replace('double glazing production, ', '')
                       .str.replace('thermally broken aluminium, ', '')
                       )
```

Normalising the results according to the highest value:

[73]:
```python
df_norm_impact_gas = df_impact_gas / df_impact_gas.max()
```

Displaying a heatmap with the normalised results (1 = maximum impact):

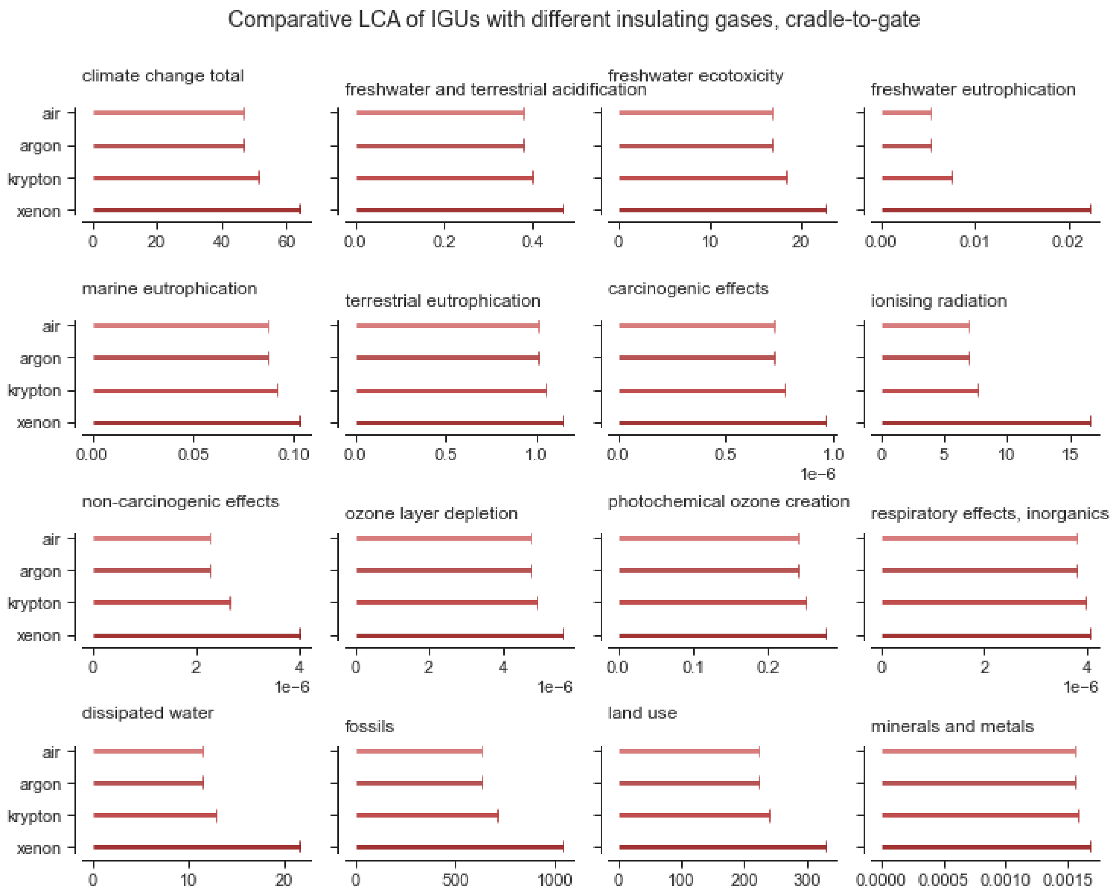
[74]:
```python
fig, ax = plt.subplots(figsize=(5, 6))

y_axis_labels = []
for label in df_norm_impact_gas.columns:
    y_axis_labels.append(label[1])

df_plot = df_norm_impact_gas.T

ax = sns.heatmap(df_plot, cmap="OrRd", vmin=0, vmax=1, annot=True, fmt='.2f',
                 yticklabels=y_axis_labels)

ax.yaxis.label.set_visible(False)
ax.xaxis.label.set_visible(False)

fig.suptitle(
    'Heatmap comparing normalised LCIA results'
    ' for IGUs with different types of insulating gas')
sns.despine(left=True, offset=5)

for tick in ax.get_xticklabels():
    tick.set_rotation(45)
    tick.set_ha('right')

plt.show()
```



Displaying the full LCIA results:

[75]:
```
fig, axes = plt.subplots(nrows=4, ncols=4,
                         sharex=False, sharey=True,
                         figsize=(12, 9))

c = ["indianred", "brown", "firebrick", "darkred"]

n = 0

for row in range(4):
    for col in range(4):
        col_name = df_impact_gas.columns[n]
        ax = axes[row][col]

        ax.hlines(y=df_impact_gas.index,
                  xmin=0, xmax=df_impact_gas[col_name],
                  linewidth=3, colors=c, alpha=0.8)

        sns.scatterplot(y=df_impact_gas.index,
                        x=df_impact_gas[col_name],
                        hue=df_impact_gas.index, s=80, marker="|",
                        palette=c, ax=ax)

        if (n % 2) == 0:
            ax.set_title(col_name[1], y=1.17, x=0,
                         ha='left', multialignment='left')
        else:
            ax.set_title(col_name[1], y=1.05, x=0,
                         ha='left', multialignment='left')

        ax.xaxis.label.set_visible(False)
        ax.yaxis.label.set_visible(False)

        ax.get_legend().remove()

        n += 1

fig.subplots_adjust(wspace=0.15, hspace=1)

fig.suptitle(
    'Comparative LCA of IGUs with different insulating gases, cradle-to-gate'
)
sns.despine(offset=5)

if export:
```

```
    # Save image:
    fig.savefig(os.path.join(path_img, 'IGU_Gas_FullLCIA.png'),
                dpi=600, bbox_inches='tight')
    fig.savefig(os.path.join(path_img, 'IGU_Gas_FullLCIA.pdf'),
                bbox_inches='tight')

plt.show()
```



Weighted environmental impact:

Weighting the LCIA results according to the PEF weighting factors:

[76]:
```
# Dropping the unit row index to ease the calculation:
df_weighted_gas = df_impact_gas.copy()
df_weighted_gas.columns = df_weighted_gas.columns.droplevel(2)

# Defining a new DataFrame with the weighted values,
# i.e., multiplication of the impacts by df_weighting:
df_weighted_gas = pd.DataFrame(
    (df_weighted_gas.multiply(
```

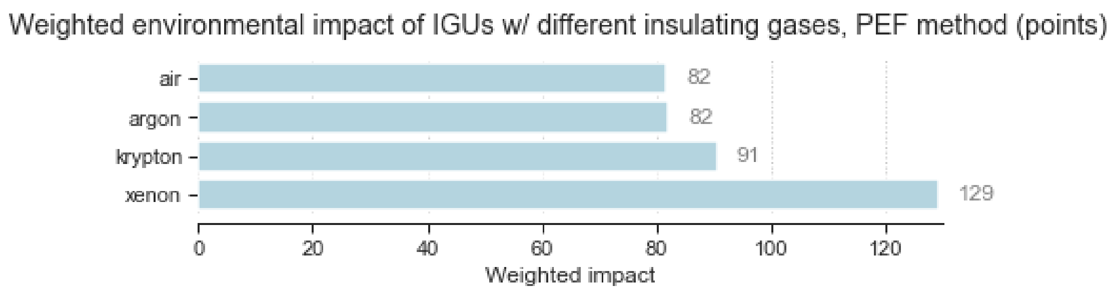
```
        df_weighting["Weighting factor"].T, axis=1) / 100
     ).sum(axis=1), columns=['Weighted impact']
)

df_weighted_gas = df_weighted_gas.sort_values("Weighted impact",
                                              ascending=True
                                              )

df_weighted_gas
```

[76]:

```
         Weighted impact
Name
air            81.844345
argon          82.080301
krypton        90.650686
xenon         129.406049
```

[77]:

```
# Displaying a barplot figure with the weighted results:
fig, ax = plt.subplots(figsize=(7, 1.5))

g = sns.barplot(data=df_weighted_gas,
                x="Weighted impact",
                y=df_weighted_gas.index,
                color="lightblue", linewidth=1.5)

g.bar_label(g.containers[0], fmt="%.0f", padding=10, c='grey')

ax.yaxis.label.set_visible(False)
ax.grid(which='major', axis='x', linestyle=':', linewidth=1)

ax.set_xlim(0, 130)
plt.xticks(np.arange(0, 121, 20))

fig.suptitle('Weighted environmental impact of IGUs '
             'w/ different insulating gases,'
             ' PEF method (points)', y=1.1)
sns.despine(left=True, offset=5)

if export:
    # Save image:
    fig.savefig(os.path.join(path_img, 'IGU_Gas_WeightedLCIA.png'),
                dpi=600, bbox_inches='tight')
    fig.savefig(os.path.join(path_img, 'IGU_Gas_WeightedLCIA.pdf'),
                bbox_inches='tight')

plt.show()
```



7.3 From Single to Triple Glazing: A Comparative LCA of IGUs, Cradle-to-Gate

Listing the IGUs (market activities) and the functional units:

[78]:
```
# List of the market actitivities relating to the glazing products studied:
inv_igus = [act for act in bw.Database("exldb_igu")
            if 'market' in act['name']
            and ('glazing' in act['name']
            or 'vacuum' in act['name'])
            ]

# 1 m² of IGU:
fu_igus = [{igu: 1} for igu in inv_igus]
```

[79]:
```
print("\033[1m", "List of the activities assessed:", "\033[0m")

for fu in fu_igus:
    for key, value in fu.items():
        print(key["name"])
```

```
 List of the activities assessed:
market for single glazing, lsg, coated
market for single glazing, lsg
market for double glazing, lsg, vacuum
market for triple glazing, lsg, two coatings
market for double glazing, lsg, two coatings
market for double glazing, coated
market for double glazing, lsg, coated, krypton
market for triple glazing, lsg, two coatings, krypton
market for double glazing, lsg, two coatings, xenon
market for triple glazing, lsg, two coatings, xenon
market for triple glazing, coated
market for triple glazing, lsg, coated
market for double glazing, lsg
market for smart glass, double glazing
market for double glazing, lsg, coated
```

Conducting the LCIA:

[80]:
```
impact_igus = []

for igu in inv_igus:
    lca = bw.LCA({igu: 1})
    lca.lci()
    for method in ls_method_full:
        lca.switch_method(method)
        lca.lcia()
        impact_igus.append((igu["name"], igu["location"],
                            method[1], method[2], lca.score,
                            bw.methods.get(method).get('unit')))
```

Creating a DataFrame with the LCIA results:

[81]:
```
# Creating a new DataFrame from the impact list:
df_impact_igus = pd.DataFrame(
    impact_igus,
    columns=["Name", "Location", "Category", "Subcategory", "Score", "Unit"]
)

# Reorganising it:
df_impact_igus = pd.pivot_table(
    df_impact_igus, index=["Name"],
    columns=["Category", "Subcategory", "Unit"], values="Score"
)

# Sorting the values:
df_impact_igus = df_impact_igus.sort_values(
    ("climate change", "climate change total", "kg CO2-Eq"), ascending=True
)

# Simplifying the index:
df_impact_igus.index = df_impact_igus.index.str.replace('market for ', '')
```

Normalising the results according to the highest value:

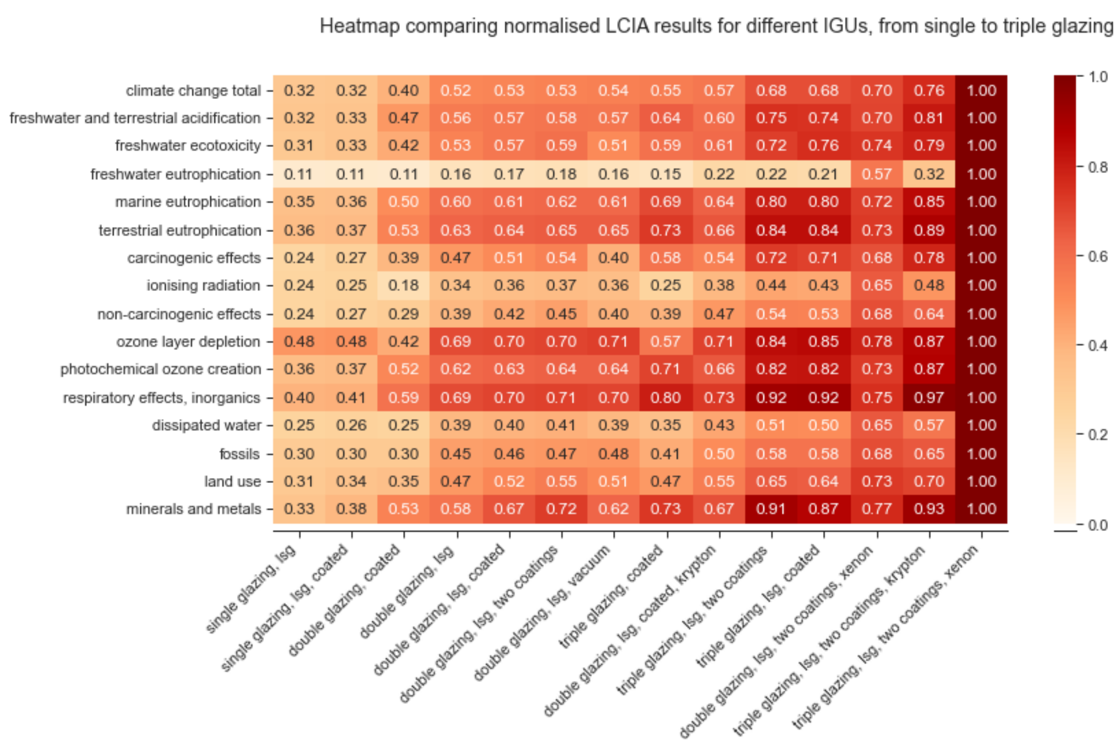
[83]:
```
# With all the IGUs:
df_norm_impact_igus = df_impact_igus / df_impact_igus.max()
```

[84]:
```
# ... and without the smart double glazing:
df_norm_impact_igus_wo_smartg = (
    df_impact_igus.drop("smart glass, double glazing", axis=0) /
    df_impact_igus.drop("smart glass, double glazing", axis=0).max()
)
```

Displaying a heatmap with the normalised results (1 = maximum impact):

[85]:

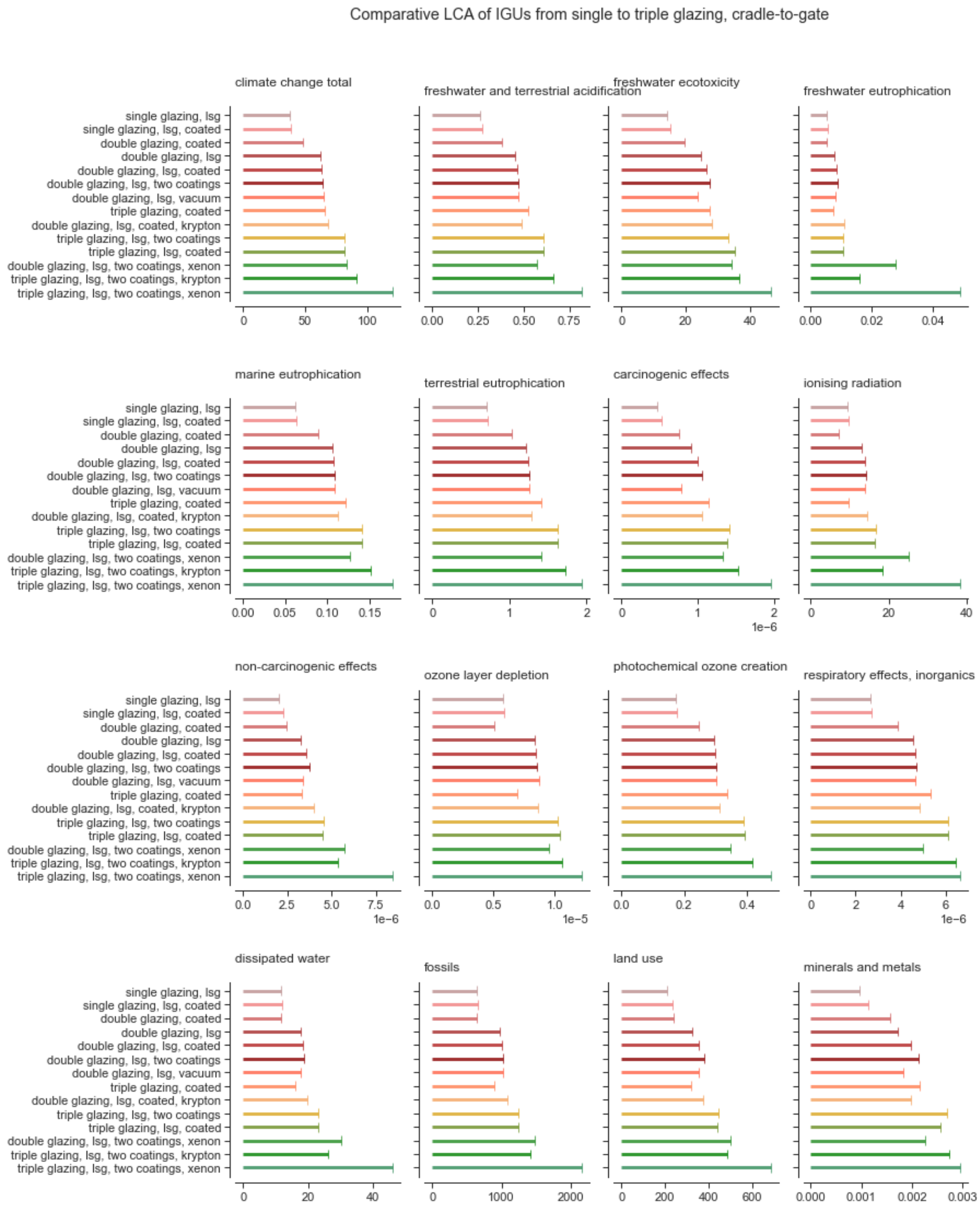
```python
fig, ax = plt.subplots(figsize=(12, 6))

y_axis_labels = []
for label in df_norm_impact_igus_wo_smartg.columns:
    y_axis_labels.append(label[1])

df_plot = df_norm_impact_igus_wo_smartg.T

ax = sns.heatmap(df_plot, cmap="OrRd", vmin=0, vmax=1, annot=True, fmt='.2f',
                 yticklabels=y_axis_labels)

ax.yaxis.label.set_visible(False)
ax.xaxis.label.set_visible(False)

fig.suptitle(
    'Heatmap comparing normalised LCIA results'
    ' for different IGUs, from single to triple glazing')
sns.despine(left=True, offset=5)

for tick in ax.get_xticklabels():
    tick.set_rotation(45)
    tick.set_ha('right')

plt.show()
```



Displaying the full LCIA results:

[86]:
```
fig, axes = plt.subplots(nrows=4, ncols=4,
                         sharex=False, sharey=True,
                         figsize=(12, 18))

df_plot = df_impact_igus.drop("smart glass, double glazing")

c = ["rosybrown", "lightcoral", "indianred",
     "brown", "firebrick", "darkred",
     "tomato", "coral", "sandybrown",
     "goldenrod", "olivedrab", "forestgreen",
     "green", "seagreen"
     ]

n = 0

for row in range(4):
    for col in range(4):
        col_name = df_plot.columns[n]
        ax = axes[row][col]

        ax.hlines(y=df_plot.index, xmin=0, xmax=df_plot[col_name],
                  linewidth=3, colors=c, alpha=0.8)

        sns.scatterplot(y=df_plot.index, x=df_plot[col_name],
                        hue=df_plot.index, s=80, marker="|",
                        palette=c, ax=ax)

        if (n % 2) == 0:
            ax.set_title(col_name[1], y=1.07, x=0,
                         ha='left', multialignment='left')
        else:
            ax.set_title(col_name[1], y=1.025, x=0,
                         ha='left', multialignment='left')

        ax.xaxis.label.set_visible(False)
        ax.yaxis.label.set_visible(False)

        ax.get_legend().remove()

        n += 1

fig.subplots_adjust(wspace=0.15, hspace=0.5)

fig.suptitle(
```

```
    'Comparative LCA of IGUs from single to triple glazing, cradle-to-gate',
    y=0.95
)
sns.despine(offset=5)

if export:
    # Save image:
    fig.savefig(os.path.join(path_img, 'IGU_FullLCIA.png'),
                dpi=600, bbox_inches='tight')
    fig.savefig(os.path.join(path_img, 'IGU_FullLCIA.pdf'),
                bbox_inches='tight')

plt.show()
```



Weighted environmental impact:

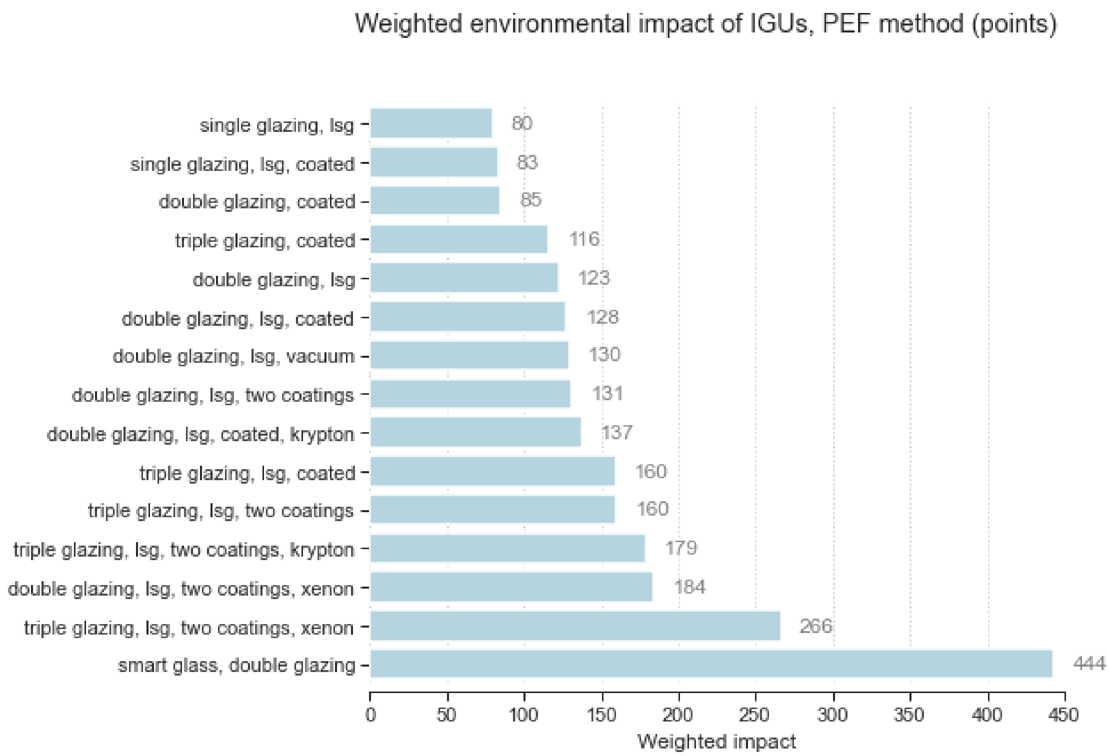
Comparing different types of IGUs according to a single indicator calculated using PEF weighting factors:

[87]:

```python
# Dropping the unit row index to ease the calculation:
df_weighted_igus = df_impact_igus.copy()
df_weighted_igus.columns = df_weighted_igus.columns.droplevel(2)

# Defining a new DataFrame with the weighted values,
# i.e., multiplication of the impacts by df_weighting:
df_weighted_igus = pd.DataFrame(
    (df_weighted_igus.multiply(
        df_weighting["Weighting factor"].T, axis=1) / 100
     ).sum(axis=1), columns=['Weighted impact']
)

df_weighted_igus = df_weighted_igus.sort_values("Weighted impact",
                                                ascending=True
                                                )
```

[88]:

```python
# Displaying a barplot figure with the weighted results:
fig, ax = plt.subplots(figsize=(7, 6))

g = sns.barplot(data=df_weighted_igus,
                x="Weighted impact",
                y=df_weighted_igus.index,
                color="lightblue", linewidth=1.5)

g.bar_label(g.containers[0], fmt="%.0f", padding=10, c='grey')

ax.yaxis.label.set_visible(False)
ax.grid(which='major', axis='x', linestyle=':', linewidth=1)

ax.set_xlim(0, 450)
plt.xticks(np.arange(0, 451, 50))

fig.suptitle('Weighted environmental impact of IGUs,'
             ' PEF method (points)', y=1)
sns.despine(left=True, offset=5)

if export:
    # Save image:
    fig.savefig(os.path.join(path_img, 'IGU_WeightedLCIA.png'),
                dpi=600, bbox_inches='tight')
    fig.savefig(os.path.join(path_img, 'IGU_WeightedLCIA.pdf'),
                bbox_inches='tight')

plt.show()
```



8 LCA of Curtain Wall Systems, from Cradle to Gate

In this section, the glazing units are integrated into curtain walls. The latter range from the classic mullion and transom system, to a unitised system (closed cavity façade, CCF) and a double skin façade (DSF). The scope of the LCA is still from cradle to gate, while an uncertainty analysis is conducted.

8.1 Environmental Impact of Curtain Wall Systems

Selecting first the activities and defining the functional unit:

[89]:
```
# List of market activities relating to the production of curtain walls:
inv_cw = [act for act in bw.Database("exldb_cw")
          if 'market for curtain wall' in act['name']
          # and 'xenon' not in act['name']
          # and 'air' not in act['name']
          ]

# 1 m² of façade:
fu_cw = [{cw: 1} for cw in inv_cw]
```

[90]:
```
print("\033[1m", "List of the activities assessed:", "\033[0m")
```

```python
for fu in fu_cw:
    for key, value in fu.items():
        print(key["name"])
```

```
 List of the activities assessed:
market for curtain wall, double glazing, two coatings, high perf alu frame
market for curtain wall, double glazing, coated, high perf alu frame
market for curtain wall, triple glazing, two coatings, xenon, high perf alu
frame
market for curtain wall, triple glazing, coated, high perf alu frame
market for curtain wall, triple glazing, two coatings, krypton, high perf alu
frame
market for curtain wall, vacuum double glazing, coated, high perf alu frame
market for curtain wall, double skin facade
market for curtain wall, single glazing, low perf alu frame
market for curtain wall, double glazing, low perf alu frame
market for curtain wall, triple glazing, two coatings, high perf alu frame
market for curtain wall, ccf
market for curtain wall, double glazing, coated, krypton, high perf alu frame
market for curtain wall, smart glazing, high perf alu frame
market for curtain wall, single glazing, coated, low perf alu frame
```

Conducting the LCIA:

[91]:
```python
impact_cw = []

for cw in inv_cw:
    lca = bw.LCA({cw: 1})
    lca.lci()
    for method in ls_method_full:
        lca.switch_method(method)
        lca.lcia()
        impact_cw.append((cw["name"], cw["location"],
                          method[1], method[2], lca.score,
                          bw.methods.get(method).get('unit')))
```

Organising the results in a DataFrame:

[92]:
```python
# Creating the DataFrame:
df_impact_cw = pd.DataFrame(
    impact_cw,
    columns=["Name", "Location", "Category", "Subcategory", "Score", "Unit"]
)

# Reorganising it:
df_impact_cw = pd.pivot_table(
    df_impact_cw, index=["Name"],
    columns=["Category", "Subcategory", "Unit"], values="Score"
)
```

```
# Sorting the values:
df_impact_cw = df_impact_cw.sort_values(
    ("climate change", "climate change total", "kg CO2-Eq"), ascending=True
)

# Simplifying the index:
df_impact_cw.index = (df_impact_cw.index
                      .str.replace('market for curtain wall, ', '')
                      )
```

[94]:
```
# Simplifying again the index to print the graph as clearly as possible:
df_impact_cw.index = (df_impact_cw.index
                      .str.replace(', high perf alu frame', '')
                      .str.replace(', low perf alu frame', '')
                      )
```

Normalising the results according to the highest value:

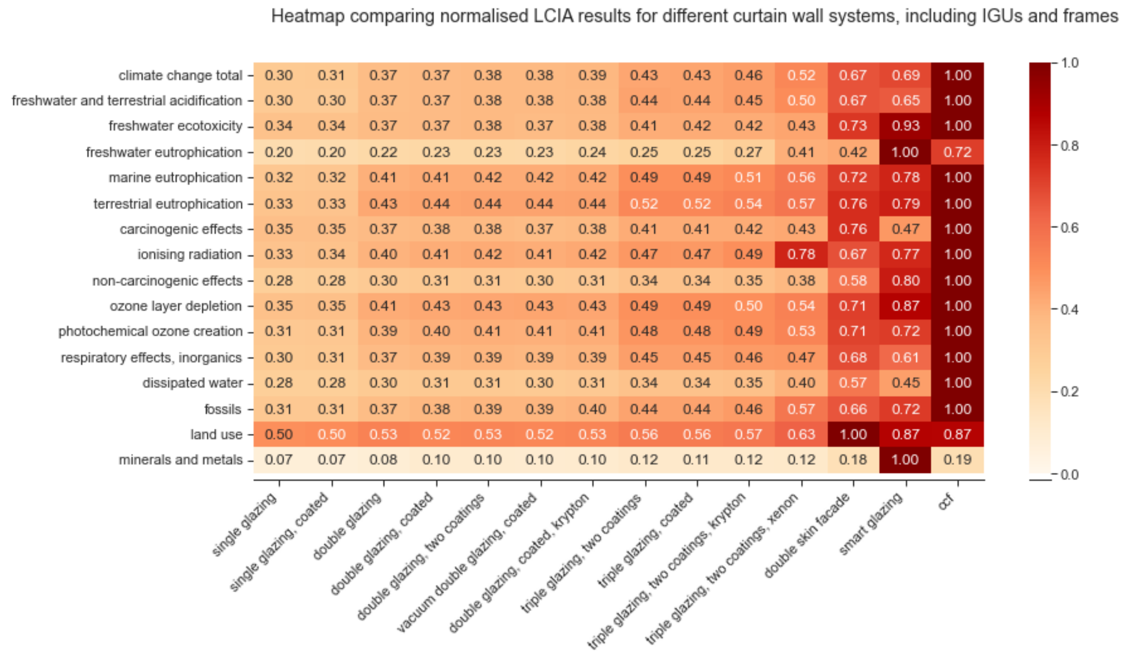
[95]:
```
# With each curtain wall system:
df_norm_impact_cw = df_impact_cw / df_impact_cw.max()
```

[96]:
```
# ... and without the smart double glazing:
df_norm_impact_cw_wo_smartg = (
    df_impact_cw.drop("smart glazing", axis=0) /
    df_impact_cw.drop("smart glazing", axis=0).max()
)
```

Displaying a heatmap with the normalised results (1 = maximum impact):

[97]:
```
fig, ax = plt.subplots(figsize=(13, 6))

y_axis_labels = []
for label in df_norm_impact_cw.columns:
    y_axis_labels.append(label[1])

df_plot = df_norm_impact_cw.T

ax = sns.heatmap(df_plot, cmap="OrRd", vmin=0, vmax=1, annot=True, fmt='.2f',
                 yticklabels=y_axis_labels)

ax.yaxis.label.set_visible(False)
ax.xaxis.label.set_visible(False)

fig.suptitle('Heatmap comparing normalised LCIA results'
             ' for different curtain wall systems, including IGUs and frames'
             )
```

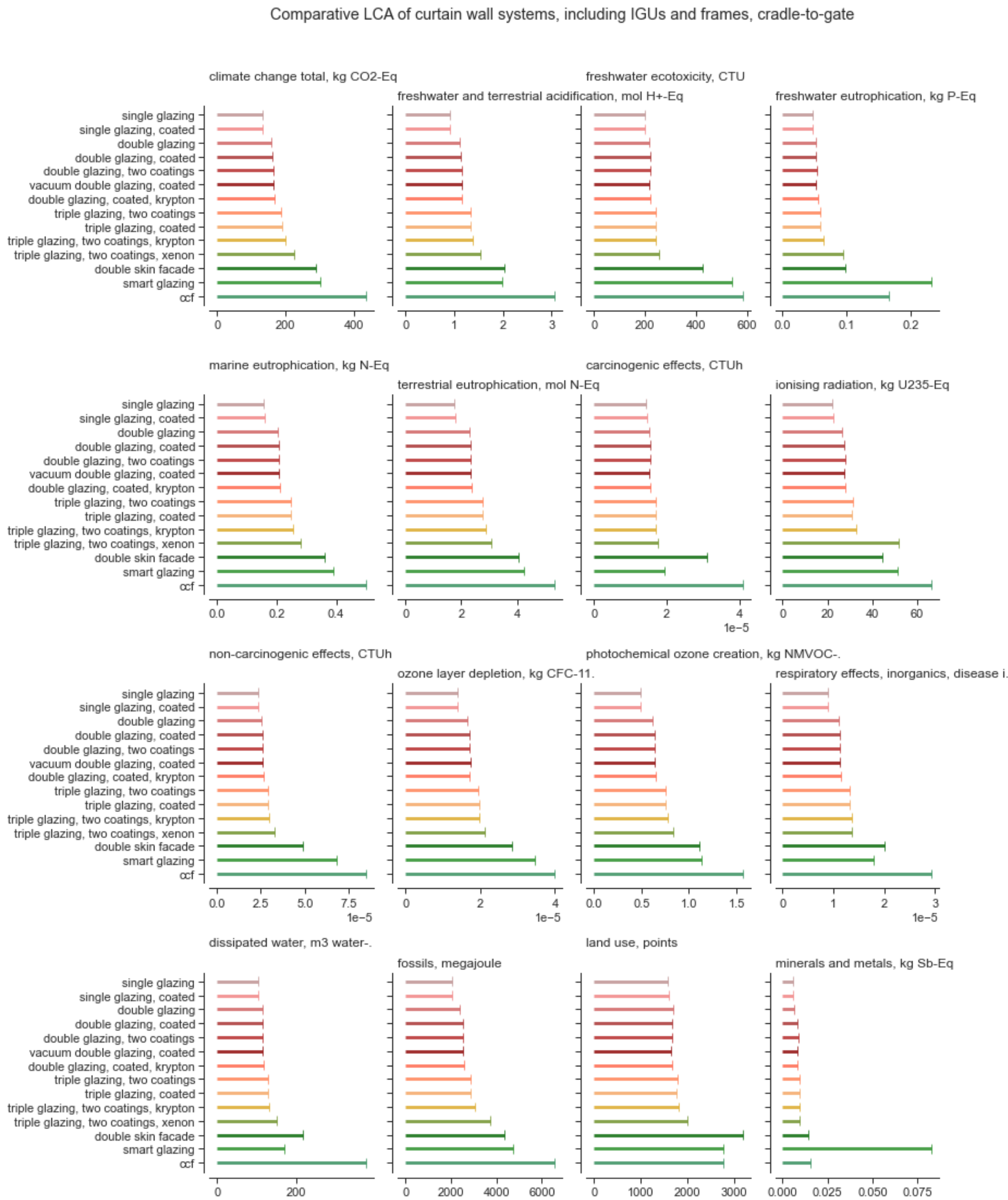
```python
sns.despine(left=True, offset=5)

for tick in ax.get_xticklabels():
    tick.set_rotation(45)
    tick.set_ha('right')

plt.show()
```



Displaying the full LCIA results:

[98]:
```python
fig, axes = plt.subplots(nrows=4, ncols=4,
                         sharex=False, sharey=True,
                         figsize=(12, 18))

c = ["rosybrown", "lightcoral", "indianred",
     "brown", "firebrick", "darkred",
     "tomato", "coral", "sandybrown",
     "goldenrod", "olivedrab", "darkgreen",
     "forestgreen", "seagreen"
     ]

n = 0

for row in range(4):
    for col in range(4):
        col_name = df_impact_cw.columns[n]
```

```
        ax = axes[row][col]

        ax.hlines(y=df_impact_cw.index, xmin=0, xmax=df_impact_cw[col_name],
                  linewidth=3, colors=c, alpha=0.8)

        sns.scatterplot(y=df_impact_cw.index, x=df_impact_cw[col_name],
                        hue=df_impact_cw.index, s=80, marker="|",
                        palette=c, ax=ax)

        if (n % 2) == 0:
            ax.set_title(f"{col_name[1]}, {col_name[2]}", y=1.1, x=0,
                         ha='left', multialignment='left')
        else:
            ax.set_title(f"{col_name[1]}, {col_name[2]}", y=1, x=0,
                         ha='left', multialignment='left')

        ax.xaxis.label.set_visible(False)
        ax.yaxis.label.set_visible(False)

        ax.get_legend().remove()

        n += 1

fig.subplots_adjust(wspace=0.15, hspace=0.45)

fig.suptitle('Comparative LCA of curtain wall systems, '
             'including IGUs and frames, cradle-to-gate',
             y=0.95
             )

sns.despine(offset=5)

if export:
    # Save image:
    fig.savefig(os.path.join(path_img, 'CW_fullLCIA.png'),
                dpi=600, bbox_inches='tight')
    fig.savefig(os.path.join(path_img, 'CW_fullLCIA.pdf'),
                bbox_inches='tight')

plt.show()
```



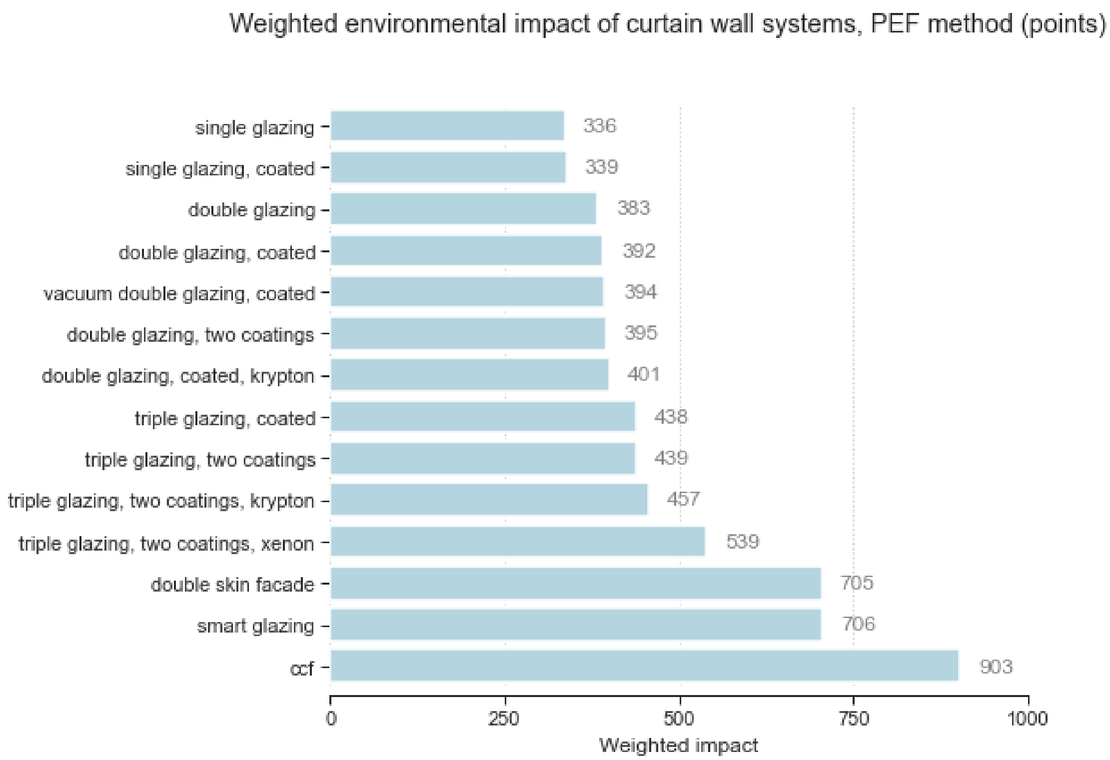
Weighted environmental impact:

Comparing different types of curatin wall systems according to a single indicator calculated using PEF weighting factors:

[99]:
```
# Dropping the unit row index to ease the calculation:
df_weighted_cw = df_impact_cw.copy()
df_weighted_cw.columns = df_weighted_cw.columns.droplevel(2)
```

```python
# Defining a new DataFrame with the weighted values,
# i.e., multiplication of the impacts by df_weighting:
df_weighted_cw = pd.DataFrame(
    (df_weighted_cw.multiply(
        df_weighting["Weighting factor"].T, axis=1) / 100
     ).sum(axis=1), columns=['Weighted impact']
)

df_weighted_cw = df_weighted_cw.sort_values("Weighted impact",
                                            ascending=True
                                            )
```

[100]:
```python
# Displaying a barplot figure with the weighted results:
fig, ax = plt.subplots(figsize=(7, 6))

g = sns.barplot(data=df_weighted_cw,
                x="Weighted impact",
                y=df_weighted_cw.index,
                color="lightblue", linewidth=1.5)

g.bar_label(g.containers[0], fmt="%.0f", padding=10, c='grey')

ax.yaxis.label.set_visible(False)
ax.grid(which='major', axis='x', linestyle=':', linewidth=1)

ax.set_xlim(0, 1000)
plt.xticks(np.arange(0, 1001, 250))

fig.suptitle('Weighted environmental impact of curtain wall systems,'
             ' PEF method (points)', y=1)
sns.despine(left=True, offset=5)

if export:
    # Save image:
    fig.savefig(os.path.join(path_img, 'CW_weightedLCIA.png'),
                dpi=600, bbox_inches='tight')
    fig.savefig(os.path.join(path_img, 'CW_weightedLCIA.pdf'),
                bbox_inches='tight')

plt.show()
```



Contribution analysis: considering the share of glazing in the climate change potential of the curtain walls:

Creating a new DataFrame with the results specific to the climate change potential:

[101]:
```python
# Data relating to the impact of IGUs:
df_contribution_igu = (
    df_impact_igus["climate change", "climate change total"].copy()
)

for igu in df_contribution_igu.index:
    if "lsg" not in igu and "smart" not in igu:
        df_contribution_igu = df_contribution_igu.drop(igu)

df_contribution_igu.index = (df_contribution_igu.index
                             .str.replace(", lsg", "")
                             .str.replace("double glazing, vacuum",
                                          "vacuum double glazing, coated")
                             .str.replace("smart glass, double glazing",
                                          "smart glazing")
                             .str.replace("", "")
                             )
```

```
df_contribution_igu.columns = pd.MultiIndex.from_tuples(
    [("IGU", "kg CO2-Eq")], names=['Scope', 'Unit']
)
```

[102]:
```
# Data relating to the impact of curtain walls:
df_cw_gwp = df_impact_cw["climate change", "climate change total"].copy()

df_cw_gwp.columns = pd.MultiIndex.from_tuples(
    [("CW", "kg CO2-Eq")], names=['Scope', 'Unit']
)
```

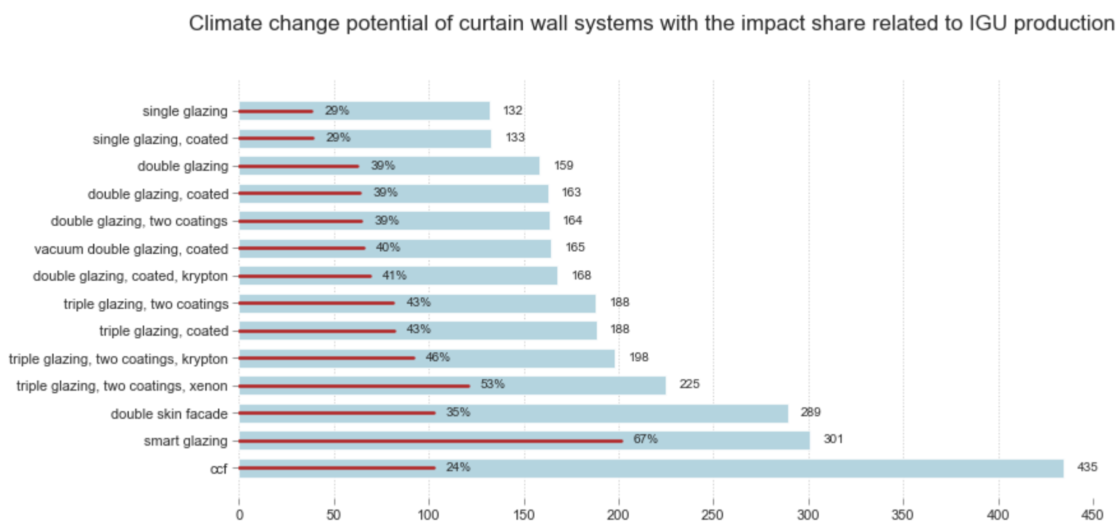
[103]:
```
# Merging the two DataFrames in a new one:
df_contribution_gwp = pd.concat([df_cw_gwp, df_contribution_igu], axis=1)

df_contribution_gwp[("IGU", "kg CO2-Eq")]["double skin facade"] = (
    df_contribution_gwp[("IGU", "kg CO2-Eq")]["single glazing, coated"]
    + df_contribution_gwp[("IGU", "kg CO2-Eq")]["double glazing, coated"]
)

df_contribution_gwp[("IGU", "kg CO2-Eq")]["ccf"] = (
    df_contribution_gwp[("IGU", "kg CO2-Eq")]["single glazing, coated"]
    + df_contribution_gwp[("IGU", "kg CO2-Eq")]["double glazing, coated"]
)

df_contribution_gwp = df_contribution_gwp.drop(
    ["double glazing, two coatings, xenon"]
)
```

Displaying the climate change impact of the different curtain wall configurations, with the share relating to glazing production:

[104]:
```
fig, ax = plt.subplots(figsize=(12, 6))

sns.barplot(x=df_contribution_gwp[("CW", "kg CO2-Eq")],
            y=df_contribution_gwp.index,
            color="lightblue", ax=ax
            )

ls_plot = df_contribution_gwp.index

# Plot an indicator line for IGU contribution:
for ix, a in enumerate(ax.patches):
    y_start = a.get_y()
    height = a.get_height()

    x1 = (df_contribution_gwp[("IGU", "kg CO2-Eq")]
          .loc[ls_plot[ix]]
          )
```

```
    ax.plot([0, x1],
            [y_start+height/2, y_start+height/2],
            '-', c='firebrick', linewidth=2.5)

    ax.set(xlim=(0, 450), ylabel="", xlabel="kg CO2-Eq.")

    # Write total impact:
    x_text = (df_contribution_gwp[("CW", "kg CO2-Eq")]
              .loc[ls_plot[ix]]
              )
    y_text = y_start+(height/2)

    s = str("%.0f" % (df_contribution_gwp[("CW", "kg CO2-Eq")]
                      .loc[ls_plot[ix]])
            )

    ax.text(x_text+7, y_text+0.1, s, fontsize=10)

    # Write IGU contribution:
    x_text_igu = (df_contribution_gwp[("IGU", "kg CO2-Eq")]
                  .loc[ls_plot[ix]]
                  )
    y_text_igu = y_start+(height/2)

    s_igu = str("%.0f" % (df_contribution_gwp[("IGU", "kg CO2-Eq")]
                          .loc[ls_plot[ix]]
                          / df_contribution_gwp[("CW", "kg CO2-Eq")]
                          .loc[ls_plot[ix]] * 100
                          ) + "%"
                )

    ax.text(x_text_igu+7, y_text_igu+0.1, s_igu, fontsize=10)

ax.grid(which='major', axis='x', linestyle=':', linewidth=1)

style_ax(ax)

# Adjust the width of the bars:
for patch in ax.patches:
    value = 0.7
    current_height = patch.get_height()
    diff = current_height - value
    patch.set_height(value)
    # recenter the bar:
    patch.set_y(patch.get_y() + diff*0.5)
```

```
fig.subplots_adjust(wspace=0.15, hspace=0.5)

fig.suptitle("Climate change potential of curtain wall systems"
             " with the impact share related to IGU production",
             fontsize=17, y=1)
sns.despine(left=True, bottom=True, offset=5)

if export:
    # Save image:
    fig.savefig(os.path.join(path_img, 'CW_contribution_GWP.png'),
                dpi=600, bbox_inches='tight')
    fig.savefig(os.path.join(path_img, 'CW_contribution_GWP.pdf'),
                bbox_inches='tight')

plt.show()
```



The same contribution analysis but for the other indicators:

[105]:
```
# Selecting the indicator within the column index:
i = 2

# Defining the index for displaying the chart below:
i_1 = df_impact_cw.columns[i][0]
i_2 = df_impact_cw.columns[i][1]
i_3 = df_impact_cw.columns[i][2]

print(i_1, ", ", i_2, ", ", i_3)
```

```
ecosystem quality ,  freshwater ecotoxicity ,  CTU
```

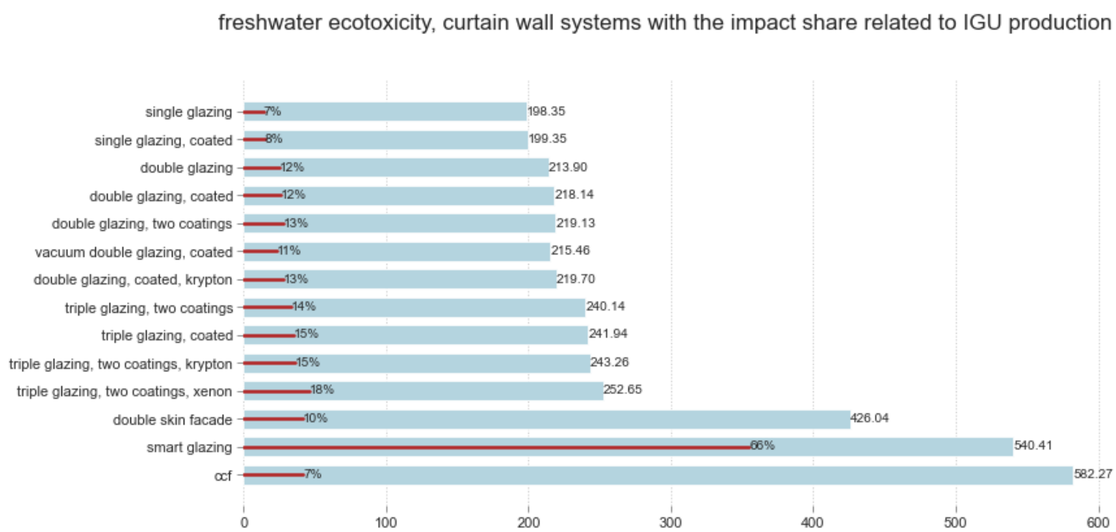
[106]:

```python
# Data relating to the impact of IGUs:
df_contribution_igu = (
    df_impact_igus[i_1, i_2].copy()
)

for igu in df_contribution_igu.index:
    if "lsg" not in igu and "smart" not in igu:
        df_contribution_igu = df_contribution_igu.drop(igu)

df_contribution_igu.index = (df_contribution_igu.index
                              .str.replace(", lsg", "")
                              .str.replace("double glazing, vacuum",
                                           "vacuum double glazing, coated")
                              .str.replace("smart glass, double glazing",
                                           "smart glazing")
                              .str.replace("", "")
                              )

df_contribution_igu.columns = pd.MultiIndex.from_tuples(
    [("IGU", i_3)], names=['Scope', 'Unit']
)

# Data relating to the impact of curtain walls:
df_contribution_cw = df_impact_cw[i_1, i_2].copy()

df_contribution_cw.columns = pd.MultiIndex.from_tuples(
    [("CW", i_3)], names=['Scope', 'Unit']
)

# Merging the two DataFrames in a new one:
df_contribution = pd.concat([df_contribution_cw, df_contribution_igu],
                            axis=1
                            )

df_contribution[("IGU", i_3)]["double skin facade"] = (
    df_contribution[("IGU", i_3)]["single glazing, coated"]
    + df_contribution[("IGU", i_3)]["double glazing, coated"]
)

df_contribution[("IGU", i_3)]["ccf"] = (
    df_contribution[("IGU", i_3)]["single glazing, coated"]
    + df_contribution[("IGU", i_3)]["double glazing, coated"]
)

df_contribution = df_contribution.drop(
    ["double glazing, two coatings, xenon"]
)
```

Displaying the climate change impact of the different curtain wall configurations, with the share relating to glazing production:

[107]:
```
fig, ax = plt.subplots(figsize=(12, 6))

sns.barplot(x=df_contribution[("CW", i_3)],
            y=df_contribution.index,
            color="lightblue", ax=ax
            )

ls_plot = df_contribution.index

# Plot an indicator line for IGU contribution:
for ix, a in enumerate(ax.patches):
    y_start = a.get_y()
    height = a.get_height()

    x1 = (df_contribution[("IGU", i_3)]
          .loc[ls_plot[ix]]
          )

    ax.plot([0, x1],
            [y_start+height/2, y_start+height/2],
            '-', c='firebrick', linewidth=2.5)

    ax.set(ylabel="", xlabel=i_3)
    ax.set_xlim(xmin=0)

    # Write total impact:
    x_text = (df_contribution[("CW", i_3)]
              .loc[ls_plot[ix]]
              )
    y_text = y_start+(height/2)

    s = str("%.2f" % (df_contribution[("CW", i_3)]
                      .loc[ls_plot[ix]])
            )

    ax.text(x_text+0.0008, y_text+0.1, s, fontsize=10)

    # Write IGU contribution:
    x_text_igu = (df_contribution[("IGU", i_3)]
                  .loc[ls_plot[ix]]
                  )
    y_text_igu = y_start+(height/2)

    s_igu = str("%.0f" % (df_contribution[("IGU", i_3)]
```

```
                          .loc[ls_plot[ix]]
                          / df_contribution[("CW", i_3)]
                          .loc[ls_plot[ix]] * 100
                          ) + "%"
                )

    ax.text(x_text_igu+0.0008, y_text_igu+0.1, s_igu, fontsize=10)

ax.grid(which='major', axis='x', linestyle=':', linewidth=1)

style_ax(ax)

# Adjust the width of the bars:
for patch in ax.patches:
    value = 0.7
    current_height = patch.get_height()
    diff = current_height - value
    patch.set_height(value)
    # recenter the bar:
    patch.set_y(patch.get_y() + diff*0.5)

fig.subplots_adjust(wspace=0.15, hspace=0.5)

fig.suptitle(f"{i_2}, curtain wall systems"
             " with the impact share related to IGU production",
             fontsize=17, y=1)
sns.despine(left=True, bottom=True, offset=5)

if export:
    # Save image:
    fig.savefig(os.path.join(path_img, 'CW_contribution_FW.png'),
                dpi=600, bbox_inches='tight')
    fig.savefig(os.path.join(path_img, 'CW_contribution_FW.pdf'),
                bbox_inches='tight')

plt.show()
```



8.2 Uncertainty Analysis through Monte Carlo Simulation

Defining the number of iterations:

[108]:
```
n_runs = 500
```

8.2.1 Monte Carlo Simulations, Single Activity and Single Indicator

Defining the activity:

[109]:
```
act = "market for curtain wall, double glazing, coated, high perf alu frame"

for fu in fu_cw:
    for key, value in fu.items():
        if act in str(key):
            print(key)
            mc_fu = fu
```

```
'market for curtain wall, double glazing, coated, high perf alu frame' (square
meter, BE, ('building components', 'windows'))
```

Conducting the Monte Carlo simulations for the ILCD climate change indicator:

[110]:
```
mc = MonteCarloLCA(mc_fu, method_ilcd_gwp)
ls_mc_results = [next(mc) for n in range(n_runs)]
```

Analysing the results:

[111]:
```
pd.DataFrame(ls_mc_results).describe()
```

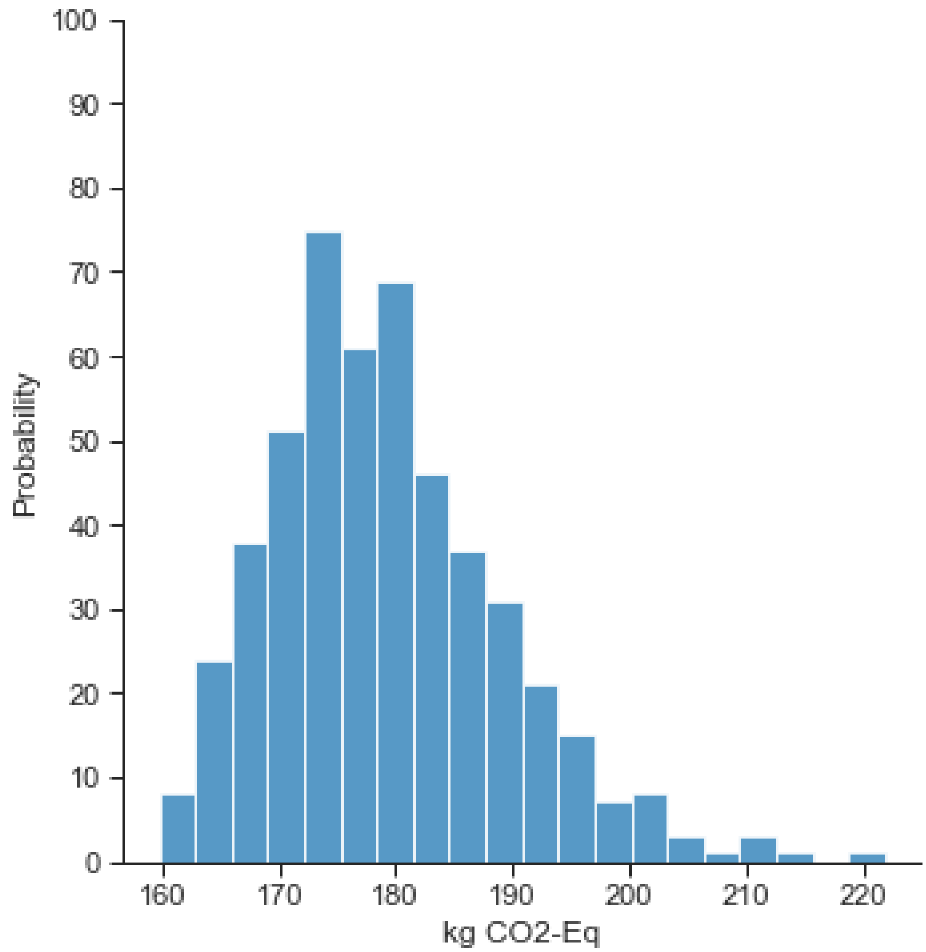
[111]:

	0
count	500.000000
mean	179.082698
std	9.777196
min	159.747156
25%	172.311025
50%	178.082988
75%	184.833467
max	221.776787

Displaying a bar chart with the results:

[112]:
```
sns.displot(data=ls_mc_results)

plt.ylabel("Probability")
plt.xlabel(methods[method_ilcd_gwp]["unit"])

plt.yticks(np.arange(0, 101, 10))

for key, value in mc_fu.items():
    print(key["name"])
    print(value, "m²")
```

```
market for curtain wall, double glazing, coated, high perf alu frame
1 m²
```



8.2.2 Monte Carlo Simulations of Different IGUs

Defining a boolean value to conduct or not the Monte Carlo Simulations. According to run number, it can take a some time. If False is chosen, data from the csv file are retrived (if the file exists):

[113]:
```
mc_bool = False
```

Checking which activity and method is analysed here:

[114]:
```
mc_fu
```

[114]:
```
{'market for curtain wall, double glazing, coated, high perf alu frame' (square
meter, BE, ('building components', 'windows')): 1}
```

[115]:
```
mc = MonteCarloLCA(mc_fu, ls_method_full[0])
# method_ilcd_gwp
# ls_method_small
# ls_method_full
```

[116]:
```
ls_method_full[0]
```

[116]: ('ILCD 2.0 2018 midpoint', 'climate change', 'climate change total')

Conducting the Monte Carlo simulations using the ILCD midpoint method for climate change potential:

[117]:
```
if mc_bool:

    simulations = []
    ls_col = []

    for n in range(n_runs):
        next(mc)
        ls_mcresults = []
        for fu in fu_cw:
            mc.redo_lcia(fu)
            ls_mcresults.append(mc.score)

        simulations.append(ls_mcresults)

    for fu in fu_cw:
        a = [label for label, q in fu.items()]
        ls_col.append(a[0]["name"])

    df_mc_result_gwp = pd.DataFrame(simulations, columns=ls_col)

    df_mc_result_gwp.to_csv('outputs\lca\mc_results_cw_gwp.csv')

else:
    # Retrieve the DataFrame from results already saved in csv file:
    if os.path.isfile('outputs\lca\mc_results_cw_gwp.csv'):
        df_mc_result_gwp_csv = (
            pd.read_csv('outputs\lca\mc_results_cw_gwp.csv'))

        df_mc_result_gwp_csv = df_mc_result_gwp_csv.rename(
            columns={"Unnamed: 0": "Iteration"}
        ).set_index("Iteration")

        df_mc_result_gwp = df_mc_result_gwp_csv
        print("MonteCarlo simulation data retrieved from the csv file!")

    else:
        print("MonteCarlo DataFrame is empty!")
```

MonteCarlo simulation data retrieved from the csv file!

Conducting the Monte Carlo simulations using the ILCD midpoint method for ozone layer depletion potential:

[118]:
```
mc = MonteCarloLCA(mc_fu, ls_method_full[9])
# method_ilcd_gwp
# ls_method_small
# ls_method_full
```

[119]:
```
ls_method_full[9]
```

[119]: ('ILCD 2.0 2018 midpoint', 'human health', 'ozone layer depletion')

[120]:
```
if mc_bool:

    simulations = []
    ls_col = []

    for n in range(n_runs):
        next(mc)
        ls_mcresults = []
        for fu in fu_cw:
            mc.redo_lcia(fu)
            ls_mcresults.append(mc.score)

        simulations.append(ls_mcresults)

    for fu in fu_cw:
        a = [label for label, q in fu.items()]
        ls_col.append(a[0]["name"])

    df_mc_result_odp = pd.DataFrame(simulations, columns=ls_col)

    df_mc_result_odp.to_csv('outputs\lca\mc_results_cw_odp.csv')

else:
    # Retrieve the DataFrame from results already saved in csv file:
    if os.path.isfile('outputs\lca\mc_results_cw_odp.csv'):
        df_mc_result_odp_csv = (
            pd.read_csv('outputs\lca\mc_results_cw_odp.csv'))

        df_mc_result_odp_csv = df_mc_result_odp_csv.rename(
            columns={"Unnamed: 0": "Iteration"}
        ).set_index("Iteration")

        df_mc_result_odp = df_mc_result_odp_csv
        print("MonteCarlo simulation data retrieved from the csv file!")

    else:
        print("MonteCarlo DataFrame is empty!")
```

MonteCarlo simulation data retrieved from the csv file!

[121]:
```
# Simplifying the column names:
df_mc_result_gwp.columns = (df_mc_result_gwp.columns
                            .str.replace('market for curtain wall, ', '')
                            )

df_mc_result_odp.columns = (df_mc_result_odp.columns
                            .str.replace('market for curtain wall, ', '')
                            )
```

Describing the results relating to ozone layer depletion potential:

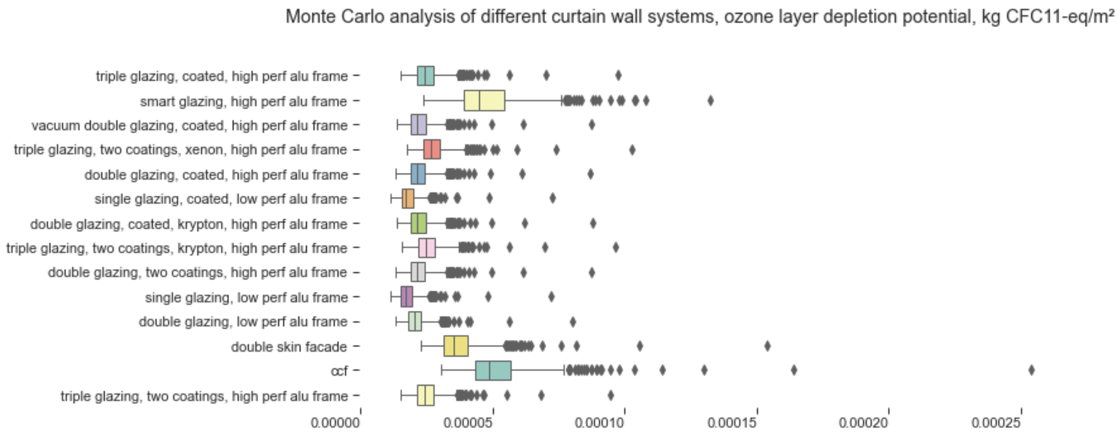
[122]:
```
with pd.option_context("display.max_rows", None,
                       "display.max_columns", None,
                       "display.float_format", '{:5.2e}'.format):
    display(df_mc_result_odp.describe().T
            .sort_values("mean", ascending=True)
            )
```

```
                                                      count     mean      std  \
single glazing, low perf alu frame                 5.00e+02 1.79e-05 4.70e-06
single glazing, coated, low perf alu frame         5.00e+02 1.81e-05 4.73e-06
double glazing, low perf alu frame                 5.00e+02 2.12e-05 5.25e-06
double glazing, coated, high perf alu frame        5.00e+02 2.22e-05 5.94e-06
double glazing, two coatings, high perf alu frame  5.00e+02 2.23e-05 5.97e-06
double glazing, coated, krypton, high perf alu … 5.00e+02 2.25e-05 6.00e-06
vacuum double glazing, coated, high perf alu frame 5.00e+02 2.25e-05 5.93e-06
triple glazing, two coatings, high perf alu frame  5.00e+02 2.52e-05 6.49e-06
triple glazing, coated, high perf alu frame        5.00e+02 2.54e-05 6.66e-06
triple glazing, two coatings, krypton, high per… 5.00e+02 2.56e-05 6.61e-06
triple glazing, two coatings, xenon, high perf … 5.00e+02 2.76e-05 6.95e-06
double skin facade                                 5.00e+02 3.74e-05 1.03e-05
smart glazing, high perf alu frame                 5.00e+02 4.81e-05 1.38e-05
ccf                                                5.00e+02 5.20e-05 1.64e-05

                                                        min      25%      50%  \
single glazing, low perf alu frame                 1.13e-05 1.53e-05 1.70e-05
single glazing, coated, low perf alu frame         1.13e-05 1.54e-05 1.71e-05
double glazing, low perf alu frame                 1.34e-05 1.82e-05 2.03e-05
double glazing, coated, high perf alu frame        1.33e-05 1.88e-05 2.12e-05
double glazing, two coatings, high perf alu frame  1.34e-05 1.89e-05 2.13e-05
double glazing, coated, krypton, high perf alu … 1.35e-05 1.90e-05 2.14e-05
vacuum double glazing, coated, high perf alu frame 1.36e-05 1.90e-05 2.14e-05
triple glazing, two coatings, high perf alu frame  1.53e-05 2.13e-05 2.41e-05
triple glazing, coated, high perf alu frame        1.53e-05 2.15e-05 2.43e-05
triple glazing, two coatings, krypton, high per… 1.56e-05 2.17e-05 2.45e-05
triple glazing, two coatings, xenon, high perf … 1.74e-05 2.35e-05 2.64e-05
double skin facade                                 2.27e-05 3.13e-05 3.54e-05
smart glazing, high perf alu frame                 2.36e-05 3.88e-05 4.50e-05
```

```
ccf                                                 3.03e-05 4.31e-05 4.87e-05

                                                         75%      max
single glazing, low perf alu frame                  1.96e-05 7.20e-05
single glazing, coated, low perf alu frame          1.97e-05 7.24e-05
double glazing, low perf alu frame                  2.30e-05 8.03e-05
double glazing, coated, high perf alu frame         2.42e-05 8.71e-05
double glazing, two coatings, high perf alu frame   2.43e-05 8.75e-05
double glazing, coated, krypton, high perf alu … 2.45e-05 8.80e-05
vacuum double glazing, coated, high perf alu frame 2.45e-05 8.71e-05
triple glazing, two coatings, high perf alu frame   2.74e-05 9.47e-05
triple glazing, coated, high perf alu frame         2.76e-05 9.76e-05
triple glazing, two coatings, krypton, high per… 2.79e-05 9.64e-05
triple glazing, two coatings, xenon, high perf … 3.00e-05 1.02e-04
double skin facade                                  4.07e-05 1.54e-04
smart glazing, high perf alu frame                  5.41e-05 1.32e-04
ccf                                                 5.68e-05 2.54e-04
```

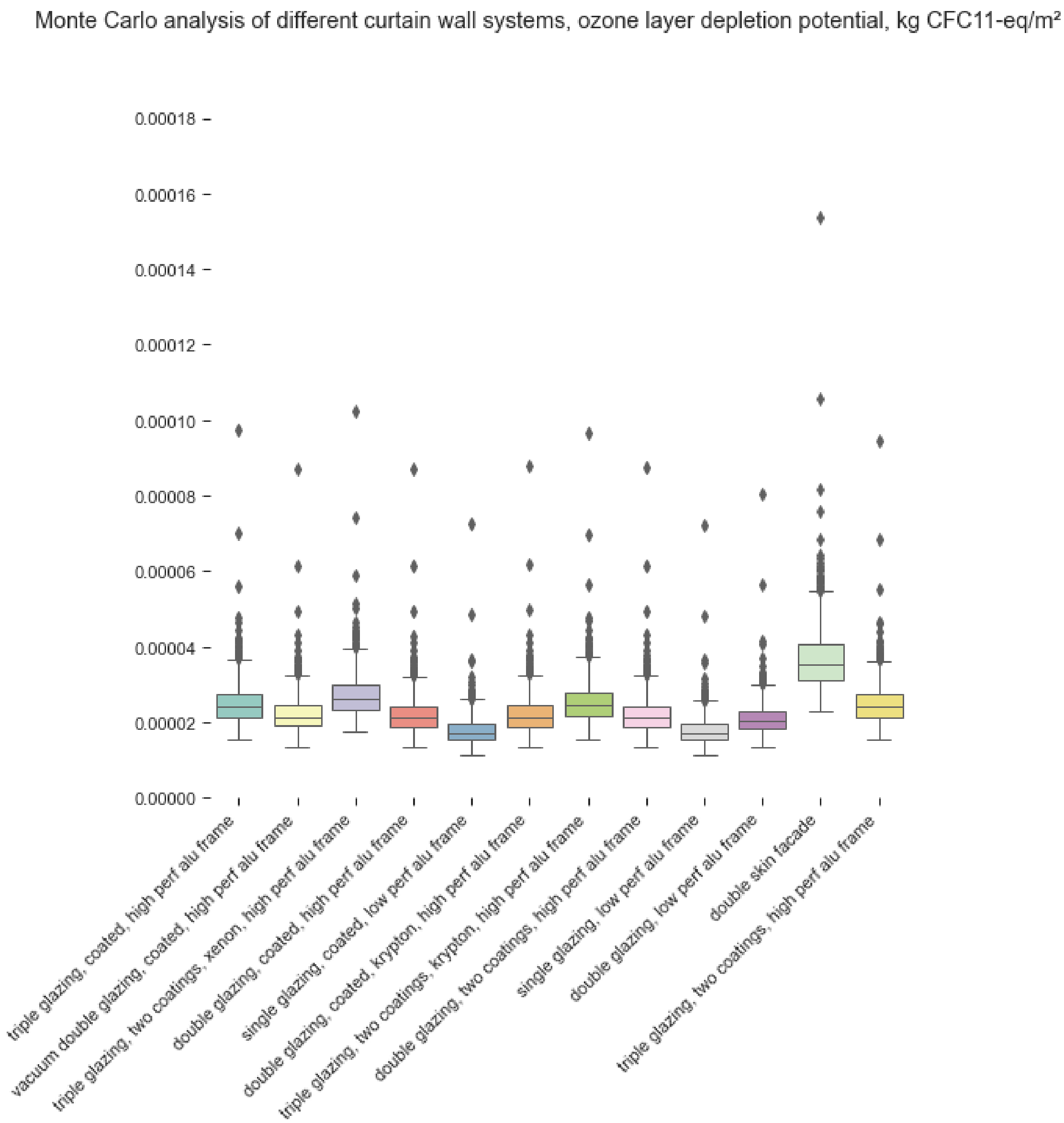
[123]:

```
fig, ax = plt.subplots(figsize=(10, 5))

ax = sns.boxplot(data=df_mc_result_odp, palette="Set3", orient="h")

ax.set(ylabel="", xlabel="")
plt.xticks(np.arange(0, 0.00026, 0.00005))

sns.despine(left=True, bottom=True, offset=5)

fig.suptitle(
    'Monte Carlo analysis of different curtain wall systems, '
    'ozone layer depletion potential, kg CFC11-eq/m²', y=1
)

for tick in ax.get_xticklabels():
    tick.set_rotation(0)
    tick.set_ha('right')

if export:
    # Save image:
    fig.savefig(os.path.join(path_img, 'MC_ODP_CW.png'),
                dpi=600, bbox_inches='tight')
    fig.savefig(os.path.join(path_img, 'MC_ODP_CW.pdf'),
                bbox_inches='tight')
```



Displaying the same graph, but without the CCF and curtain wall with smart glazing:

[124]:
```
fig, ax = plt.subplots(figsize=(8, 8))

df_plot = df_mc_result_odp[[
    x for x in df_mc_result_odp.columns if 'smart' not in x]]
df_plot = df_plot[[x for x in df_plot.columns if 'ccf' not in x]]

ax = sns.boxplot(
    data=df_plot, palette="Set3"
)

ax.set(ylabel="", xlabel="")
plt.yticks(np.arange(0, 0.0002, 0.00002))

sns.despine(left=True, bottom=True, offset=5)

fig.suptitle(
    'Monte Carlo analysis of different curtain wall systems, '
    'ozone layer depletion potential, kg CFC11-eq/m²', y=1
)

for tick in ax.get_xticklabels():
    tick.set_rotation(45)
    tick.set_ha('right')
```



Describing the results relating to climate change potential:

[125]:
```
df_mc_result_gwp.describe().round(1).T.sort_values("mean", ascending=True)
```

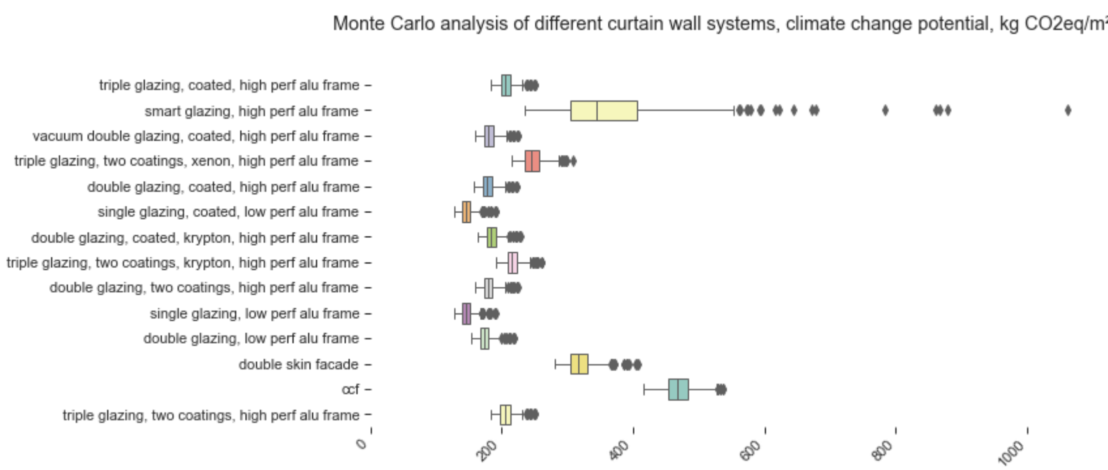
[125]:
```
                                                    count   mean   std    min  \
single glazing, low perf alu frame                  500.0  146.3  10.0  127.3
single glazing, coated, low perf alu frame          500.0  147.2  10.0  128.1
double glazing, low perf alu frame                  500.0  174.4  10.4  153.2
double glazing, coated, high perf alu frame         500.0  179.3  10.6  158.5
double glazing, two coatings, high perf alu frame   500.0  180.3  10.6  159.2
vacuum double glazing, coated, high perf alu frame  500.0  181.1  10.6  159.7
double glazing, coated, krypton, high perf alu …  500.0  184.8  10.7  162.7
```

```
triple glazing, two coatings, high perf alu frame   500.0  206.4  11.1  183.3
triple glazing, coated, high perf alu frame         500.0  206.6  11.1  183.6
triple glazing, two coatings, krypton, high per…  500.0  217.2  11.5  191.7
triple glazing, two coatings, xenon, high perf …  500.0  247.6  15.6  214.3
double skin facade                                  500.0  319.3  20.3  280.7
smart glazing, high perf alu frame                  500.0  366.5  95.8  234.5
ccf                                                 500.0  469.3  21.6  415.7

                                                      25%    50%    75%  \
single glazing, low perf alu frame                  139.3  145.2  151.2
single glazing, coated, low perf alu frame          140.2  146.1  152.1
double glazing, low perf alu frame                  166.9  173.5  180.1
double glazing, coated, high perf alu frame         171.7  178.4  185.2
double glazing, two coatings, high perf alu frame   172.6  179.3  186.2
vacuum double glazing, coated, high perf alu frame  173.5  180.3  187.1
double glazing, coated, krypton, high perf alu …  177.0  183.7  190.6
triple glazing, two coatings, high perf alu frame   198.0  205.3  212.9
triple glazing, coated, high perf alu frame         198.5  205.5  212.9
triple glazing, two coatings, krypton, high per…  208.7  216.1  223.7
triple glazing, two coatings, xenon, high perf …  236.1  245.4  257.3
double skin facade                                  305.3  317.2  329.8
smart glazing, high perf alu frame                  304.3  344.0  406.6
ccf                                                 453.7  468.3  482.8

                                                       max
single glazing, low perf alu frame                   190.7
single glazing, coated, low perf alu frame           191.4
double glazing, low perf alu frame                   218.8
double glazing, coated, high perf alu frame          223.5
double glazing, two coatings, high perf alu frame    224.2
vacuum double glazing, coated, high perf alu frame   225.1
double glazing, coated, krypton, high perf alu …   228.4
triple glazing, two coatings, high perf alu frame    250.7
triple glazing, coated, high perf alu frame          251.3
triple glazing, two coatings, krypton, high per…   261.4
triple glazing, two coatings, xenon, high perf …   309.5
double skin facade                                   408.3
smart glazing, high perf alu frame                  1061.6
ccf                                                  537.4
```

Displaying a boxplot graph with the results of the Monte Carlo analysis:

[126]:
```
fig, ax = plt.subplots(figsize=(10, 5))

ax = sns.boxplot(data=df_mc_result_gwp, palette="Set3", orient="h")

ax.set(ylabel="", xlabel="")
```

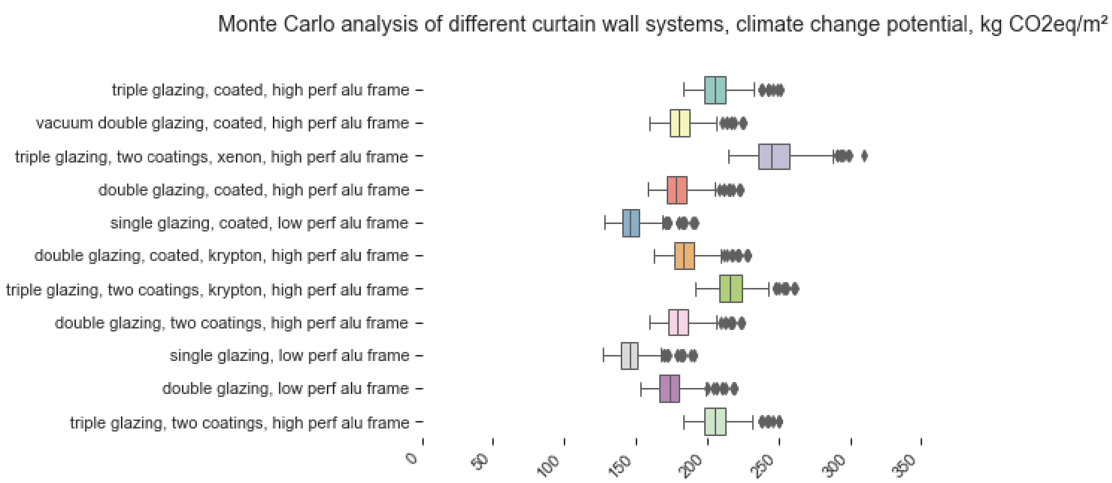
```
plt.xticks(np.arange(0, 1001, 200))

sns.despine(left=True, bottom=True, offset=5)

fig.suptitle(
    'Monte Carlo analysis of different curtain wall systems, '
    'climate change potential, kg CO2eq/m²', y=1
)

for tick in ax.get_xticklabels():
    tick.set_rotation(45)
    tick.set_ha('right')

if export:
    # Save image:
    fig.savefig(os.path.join(path_img, 'MC_GWP_CW.png'),
                dpi=600, bbox_inches='tight')
    fig.savefig(os.path.join(path_img, 'MC_GWP_CW.pdf'),
                bbox_inches='tight')
```



Displaying the same graph, but without the curtain wall with smart glazing:

[127]:
```
fig, ax = plt.subplots(figsize=(6, 4.5))

ax = sns.boxplot(
    data=df_mc_result_gwp[
        [x for x in df_mc_result_gwp.columns if ('smart glazing' not in x)
         and ('ccf' not in x) and ('double skin facade' not in x)]
    ], palette="Set3", orient="h"
)
```

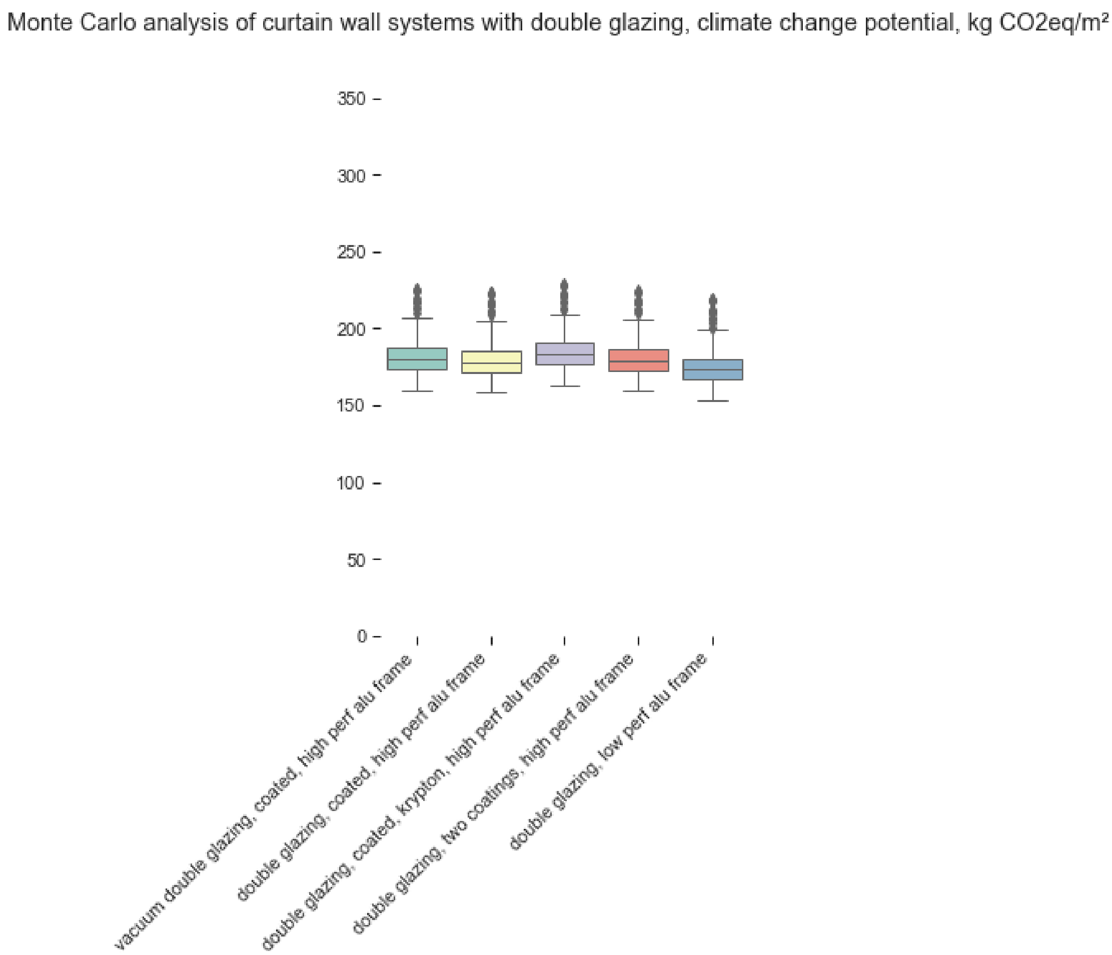
```
ax.set(ylabel="", xlabel="")
plt.xticks(np.arange(0, 351, 50))

sns.despine(left=True, bottom=True, offset=5)

fig.suptitle(
    'Monte Carlo analysis of different curtain wall systems, '
    'climate change potential, kg CO2eq/m²', y=1
)

for tick in ax.get_xticklabels():
    tick.set_rotation(45)
    tick.set_ha('right')

if export:
    # Save image:
    fig.savefig(os.path.join(path_img, 'MC_GWP_CW_DG_TG.png'),
                dpi=600, bbox_inches='tight')
    fig.savefig(os.path.join(path_img, 'MC_GWP_CW_DG_TG.pdf'),
                bbox_inches='tight')
```



Displaying a boxplot graph, but only for curtain wall systems with double glazing:

[128]:
```
fig, ax = plt.subplots(figsize=(4, 6))

ax = sns.boxplot(
    data=df_mc_result_gwp[
        [x for x in df_mc_result_gwp.columns if 'double glazing' in x]
    ], palette="Set3"
)
```

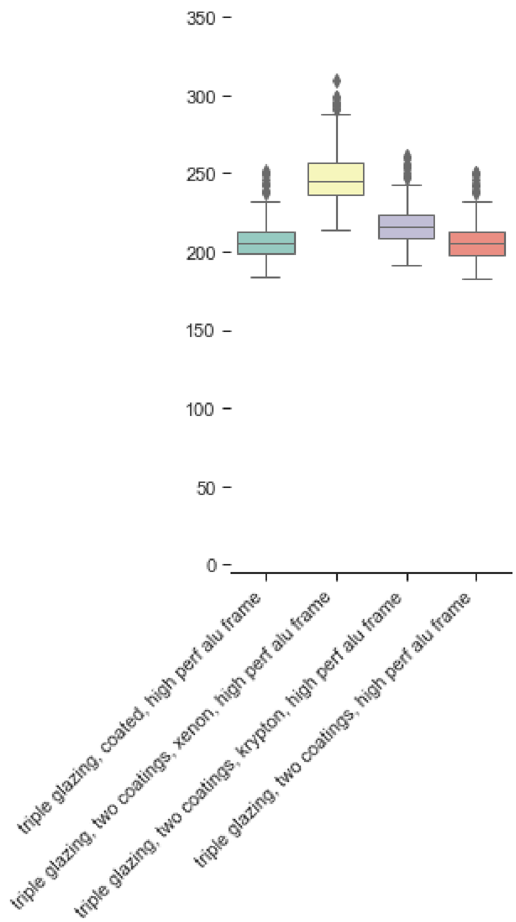
```
ax.set(ylabel="", xlabel="")
plt.yticks(np.arange(0, 351, 50))

sns.despine(left=True, bottom=True, offset=5)

fig.suptitle(
    'Monte Carlo analysis of curtain wall systems with double glazing, '
    'climate change potential, kg CO2eq/m²', y=1
)

for tick in ax.get_xticklabels():
    tick.set_rotation(45)
    tick.set_ha('right')
```



Displaying a boxplot graph, but only for curtain wall systems with triple glazing:

[129]:
```python
fig, ax = plt.subplots(figsize=(3, 6))

ax = sns.boxplot(
    data=df_mc_result_gwp[
        [x for x in df_mc_result_gwp.columns if 'triple glazing' in x]
    ], palette="Set3"
)

ax.set(ylabel="", xlabel="")
plt.yticks(np.arange(0, 351, 50))

sns.despine(left=True, offset=5)

fig.suptitle(
    'Monte Carlo analysis of curtain wall systems with triple glazing, '
    'climate change potential, kg CO2eq/m²', y=1
)

for tick in ax.get_xticklabels():
    tick.set_rotation(45)
    tick.set_ha('right')
```

Monte Carlo analysis of curtain wall systems with triple glazing, climate change potential, kg CO2eq/m²



8.2.3 Monte Carlo Analysis for Multiple Impact Categories

Extending the Monte Carlo analysis to all impact categories defined by the ILCD midpoint method:

[130]:
```
# Defining a function to conduct the MC simulations:
def multiImpactMonteCarloLCA(fu, ls_methods, nruns):

    mc_lca = bw.MonteCarloLCA(fu)
    mc_lca.lci()

    c_matrices = {}

    for method in ls_methods:
        mc_lca.switch_method(method)
        c_matrices[method] = mc_lca.characterization_matrix
```

```
    results = np.empty((len(ls_methods), nruns))

    for iteration in range(nruns):
        next(mc_lca)
        for method_index, method in enumerate(ls_methods):
            results[method_index, iteration] = (
                c_matrices[method]*mc_lca.inventory).sum()

    return results
```

[131]:
```
act_mc_multi_impact = (
    "market for curtain wall, double glazing, coated, high perf alu frame"
)

for fu in fu_cw:
    for key, value in fu.items():
        if act_mc_multi_impact in str(key):
            print(key["name"])
            mc_multi_impact_fu = fu
```

```
market for curtain wall, double glazing, coated, high perf alu frame
```

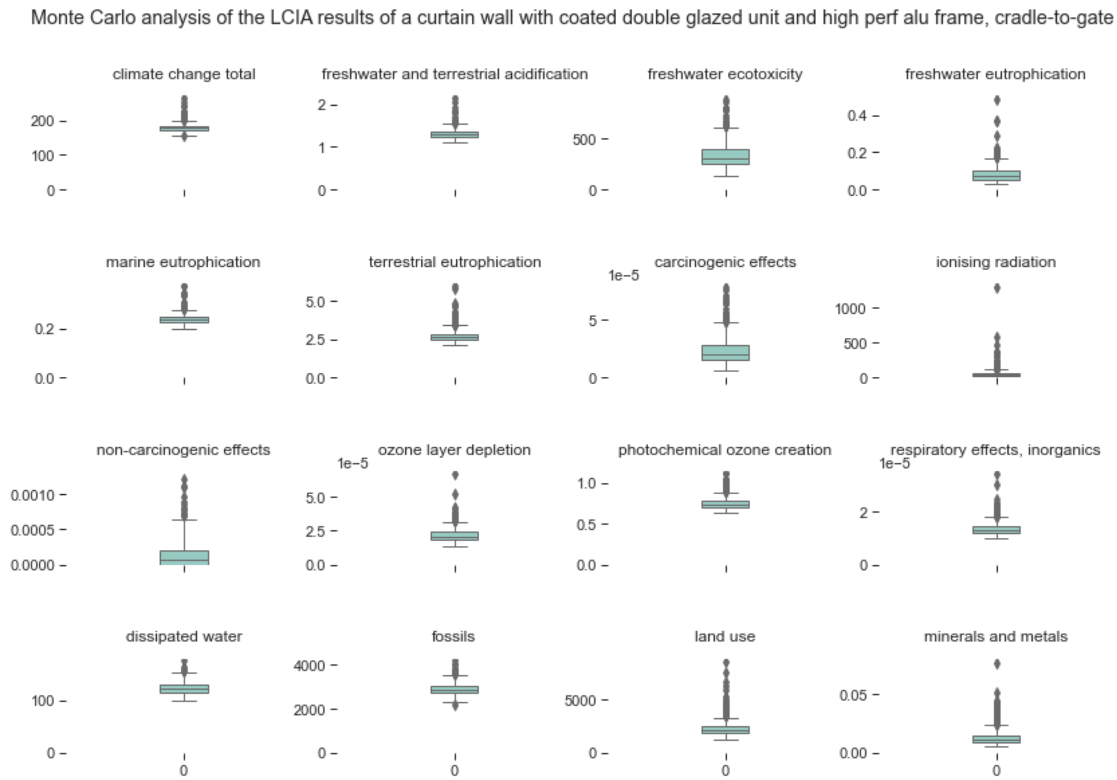
[132]:
```
if mc_bool:
    mc_results = multiImpactMonteCarloLCA(mc_multi_impact_fu,
                                          ls_method_full,
                                          n_runs
                                          )

    df_multiimpact_mc_results = pd.DataFrame(data=mc_results,
                                             index=ls_method_full).T

    df_multiimpact_mc_results.index.name = 'Iteration'

    df_multiimpact_mc_results.columns = pd.MultiIndex.from_tuples(
        df_multiimpact_mc_results.columns, names=['Method',
                                                  'Category',
                                                  'Subcategory']
    )

    df_multiimpact_mc_results.unstack().to_csv(
        'outputs\lca\multiimpact_mc_results.csv', index=True
    )

else:
    if os.path.isfile('outputs\lca\multiimpact_mc_results.csv'):
        df_multiimpact_mc_results_csv = (
            pd.read_csv('outputs\lca\multiimpact_mc_results.csv'))
        df_multiimpact_mc_results_csv = (df_multiimpact_mc_results_csv
```

```python
                                        .pivot_table(
                                            values='0',
                                            index=['Iteration'],
                                            columns=['Method',
                                                     'Category',
                                                     'Subcategory']
                                        )
                                        )

        df_multiimpact_mc_results = df_multiimpact_mc_results_csv
        print("MonteCarlo simulation data retrieved from the csv file!")

    else:
        print("MonteCarlo DataFrame is empty!")
```

```
MonteCarlo simulation data retrieved from the csv file!
```

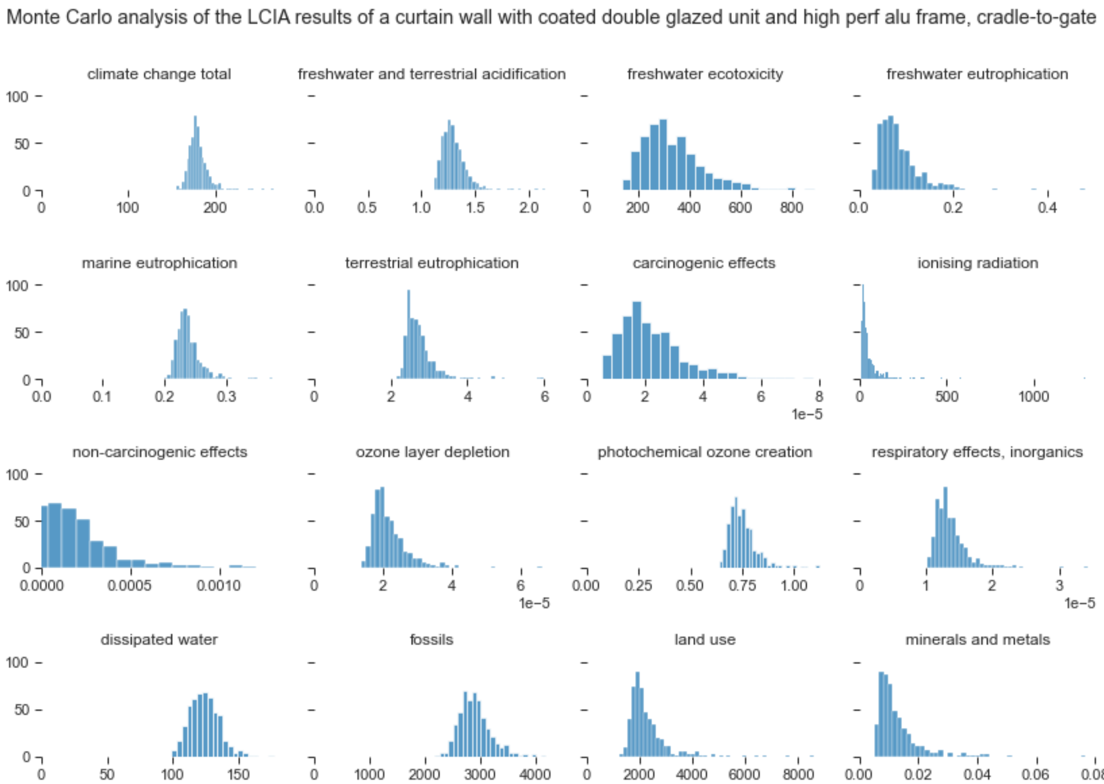
[134]:

```python
fig, axes = plt.subplots(nrows=4, ncols=4,
                         sharex=True, sharey=False,
                         figsize=(14, 9))

df_plot = df_multiimpact_mc_results['ILCD 2.0 2018 midpoint']

n = 0

for row in range(4):
    for col in range(4):
        ax = axes[row][col]
        i = df_plot.columns[n]
        sns.boxplot(data=df_plot[i], palette="Set3", width=0.2, ax=ax)

        n += 1

        ax.set(xlabel="", ylabel="")
        ax.set_ylim(ymin=0)
        ax.set_title(i[1], y=1.1)

fig.subplots_adjust(wspace=0.15, hspace=1)

fig.suptitle(
    'Monte Carlo analysis of the LCIA results of a curtain wall with '
    'coated double glazed unit and high perf alu frame, cradle-to-gate'
)
sns.despine(left=True, bottom=True, offset=5)
plt.show()
```



[135]:
```
fig, axes = plt.subplots(nrows=4, ncols=4,
                         sharex=False, sharey=True,
                         figsize=(14, 9))

df_plot = df_multiimpact_mc_results['ILCD 2.0 2018 midpoint']

n = 0

for row in range(4):
    for col in range(4):
        ax = axes[row][col]
        i = df_plot.columns[n]
        sns.histplot(data=df_plot[i], ax=ax)

        n += 1

        ax.set(xlabel="", ylabel="")
        ax.set_ylim(ymin=0, ymax=100)
        ax.set_xlim(xmin=0)
        ax.set_title(i[1], y=1.1)

fig.subplots_adjust(wspace=0.15, hspace=1)
```

```python
fig.suptitle(
    'Monte Carlo analysis of the LCIA results of a curtain wall with '
    'coated double glazed unit and high perf alu frame, cradle-to-gate'
)
sns.despine(left=True, bottom=True, offset=5)
plt.show()
```



9 Import Results from the BEM

This section imports the results of the building energy modelling carried out on the case study concerning the replacement of the curtain wall of an office building in Brussels. The data on energy use serve to conduct the LCA on the use phase of the previously analysed facades.

For more information regarding the building energy modelling, see the notebook “01_BEM” in that same project folder.

9.1 Function to Retrieve Data from the CSV Files Saved during Previous Simulations

[136]:
```
# Open the df_end_use_allsteps from the csv file:
# Avoid re-running energy simulations (time consuming):
if os.path.isfile('outputs\steps_dir\df_end_use_allsteps.csv'):
    df_end_use_allsteps_csv = (
        pd.read_csv('outputs\steps_dir\df_end_use_allsteps.csv'))
    df_end_use_allsteps_csv = df_end_use_allsteps_csv.pivot_table(
        values='0', index=['EndUse'], columns=['Run name', 'FuelType'])

    df_end_use_allsteps = df_end_use_allsteps_csv
```

A function to retrieve the df_step dataframes saved as csv, i.e. DataFrame with the main assumptions and results (natural gas and electricity) specific to each simulation run:

To assess the indirect impact of glazing replacement on energy use in the building, the natural gas and electricity use results for each scenario are subtracted by the initial scenario, where the exact same glazing is kept.

[137]:
```
def retrieve_df_step(n_step, df_step):
    """
    If a df_step.csv exists, retrieve the data and create a dataframe
        wich replace the one currently in use in the notebook.
        Avoid re-running energy simulation (time consuming).

    Parameters
    ----------
    n_step: number of the step
    df_step: a dataframe. followed by a number (e.g. step4),
    identify the step with simulation runs and main results

    Returns
    -------
    df_step: update with csv data or exactly the same as the one in the input

    """

    # Does the csv exist
    # and check if the existing df_step includes simulation results:
    if os.path.isfile(f"outputs\steps_dir\df_step"+str(n_step)+".csv"):
        df_step = (
            pd.read_csv(f"outputs\steps_dir\df_step"+str(n_step)+".csv")
            .set_index(['name']))

        print("df_step ", n_step, "updated with csv data")
    else:
        print("existing df_step ", n_step, "kept in place")
```

```python
    return df_step
```

9.2 Post-Process Data from the Building Energy Simulations

As a reminder: df_step units are MJ/m² of glazed façade for natural gas, and kWh/m² for electricity use.

[138]:
```python
df_step1 = retrieve_df_step(1, df_step1)
df_step1.name = "df_step1"
df_step2 = retrieve_df_step(2, df_step2)
df_step2.name = "df_step2"
df_step3 = retrieve_df_step(3, df_step3)
df_step3.name = "df_step3"
df_step4 = retrieve_df_step(4, df_step4)
df_step4.name = "df_step4"
df_step5 = retrieve_df_step(5, df_step5)
df_step5.name = "df_step5"
df_step6 = retrieve_df_step(6, df_step6)
df_step6.name = "df_step6"
df_step7 = retrieve_df_step(7, df_step7)
df_step7.name = "df_step7"
df_step8 = retrieve_df_step(8, df_step8)
df_step8.name = "df_step8"
df_step9 = retrieve_df_step(9, df_step9)
df_step9.name = "df_step9"
df_step10 = retrieve_df_step(10, df_step10)
df_step10.name = "df_step10"
df_step11 = retrieve_df_step(11, df_step11)
df_step11.name = "df_step11"
df_step12 = retrieve_df_step(12, df_step12)
df_step12.name = "df_step12"
df_step13 = retrieve_df_step(13, df_step13)
df_step13.name = "df_step13"
df_step14 = retrieve_df_step(14, df_step14)
df_step14.name = "df_step14"
df_step15 = retrieve_df_step(15, df_step15)
df_step15.name = "df_step15"
df_step16 = retrieve_df_step(16, df_step16)
df_step16.name = "df_step16"
```

```
df_step  1 updated with csv data
df_step  2 updated with csv data
df_step  3 updated with csv data
df_step  4 updated with csv data
df_step  5 updated with csv data
df_step  6 updated with csv data
df_step  7 updated with csv data
df_step  8 updated with csv data
```

```
df_step  9 updated with csv data
df_step  10 updated with csv data
df_step  11 updated with csv data
df_step  12 updated with csv data
df_step  13 updated with csv data
df_step  14 updated with csv data
df_step  15 updated with csv data
df_step  16 updated with csv data
```

As explained in the chapter dedicated to the methodological framework of this LCA, the contribution of each square metre of glazed façade under study is estimated following a consequential approach. This means that the initial state of the building, defined by an old double glazing called "dg_init_bronze" and an inefficient aluminium frame, is the reference state from which the impact of the new façade is calculated. For each given building configuration, the energy use in the initial state is substracted from the energy use of the whole building.

(energy use with glazing x) - (energy use with the old glazing) = (energy flow for the life cycle inventory of glazing x)

[139]:
```python
# Initial HVAC configuration, "inefficient" fan coils (steps 1, 2, 3),
# Subtraction of energy use by that in the initial scenario:
if not df_step1.loc[df_step1["glazing"] == "dg_init_bronze"].empty:
    i_gas = float(
        df_step1.loc[df_step1["glazing"] == "dg_init_bronze", "natural_gas"])
    i_elec = float(
        df_step1.loc[df_step1["glazing"] == "dg_init_bronze", "elec_use"])

    for df_step in [df_step1, df_step2, df_step3]:
        df_step["natural_gas"] = (df_step["natural_gas"] - i_gas)
        df_step["elec_use"] = (df_step["elec_use"] - i_elec)

else:
    print("DG_init not in step 1! energy use not substracted by dg_init!")
```

[140]:
```python
# Second building configuration w/ optimised VAV system,
# (steps 4, 5, 8, 9, 10-16),
# Subtraction of energy use by that in the initial scenario:
if not df_step4.loc[df_step4["glazing"] == "dg_init_bronze"].empty:
    i_gas = float(
        df_step4.loc[df_step4["glazing"] == "dg_init_bronze", "natural_gas"])
    i_elec = float(
        df_step4.loc[df_step4["glazing"] == "dg_init_bronze", "elec_use"])

    for df_step in [df_step4, df_step5, df_step8, df_step9,
                    df_step10, df_step11, df_step12, df_step13,
                    df_step14, df_step15, df_step16]:
        df_step["natural_gas"] = (df_step["natural_gas"] - i_gas)
        df_step["elec_use"] = (df_step["elec_use"] - i_elec)
```

```
else:
    print("DG_init not in step 4! energy use not substracted by dg_init!")
```

[141]:
```
# Third building configuration w/ a fully electrified VRF system (steps 6, 7):
# Subtraction of energy use by that in the initial scenario:
if not df_step6.loc[df_step6["glazing"] == "dg_init_bronze"].empty:
    i_gas = float(
        df_step6.loc[df_step6["glazing"] == "dg_init_bronze", "natural_gas"])
    i_elec = float(
        df_step6.loc[df_step6["glazing"] == "dg_init_bronze", "elec_use"])

    for df_step in [df_step6, df_step7]:
        df_step["natural_gas"] = (df_step["natural_gas"] - i_gas)
        df_step["elec_use"] = (df_step["elec_use"] - i_elec)

else:
    print("DG_init not in step 6! energy use not substracted by dg_init!")
```

10 Analysis of the Whole Life Cycle of Curtain Wall Retrofitting Scenarios

10.1 Setup of the LCA

Defining first the activity of dismantling, and thus disposal, of the existing curtain wall:

[142]:
```
out_old_cw = exldb_cw.get('dismantling_cw_old_dg')
# Check:
print('My activity is:\n', out_old_cw)
```

```
My activity is:
 'curtain wall, dismantling, old double glazing' (square meter, BE, ('building
components', 'windows'))
```

Defining then the production activity of the new curtain wall:

[143]:
```
prod_cw = exldb_cw.get('production_cw')
# Check:
print('My activity is:\n', prod_cw)
```

```
My activity is:
 'curtain wall, production' (square meter, BE, ('building components',
'windows'))
```

And the use phase activity (not linked to production):

[144]:
```
use_bldg_w_cw = exldb_cw.get('use_glazed_office_bldg')
# Check:
print('My activity is:\n', use_bldg_w_cw)
```

```
My activity is:
 'use of glazed office building, hvac and lighting' (square meter, BE,
('building components', 'windows'))
```

And another use phase activity, but linked to the production phase:

[145]:
```python
prod_and_use_cw = exldb_cw.get('use_cw')
# Check:
print('My activity is:\n', prod_and_use_cw)
```

```
My activity is:
 'use of curtain wall' (square meter, BE, ('building components', 'windows'))
```

Defining a maintenance activity:

[146]:
```python
repair_cw = exldb_cw.get('maintenance_cw')
# Check:
print('My activity is:\n', repair_cw)
```

```
My activity is:
 'curtain wall, maintenance' (square meter, BE, ('building components',
'windows'))
```

And finally, the end-of-life activity:

[147]:
```python
eol_cw = exldb_cw.get('eol_cw')
# Check:
print('My activity is:\n', eol_cw)
```

```
My activity is:
 'curtain wall, end of life' (square meter, BE, ('building components',
'windows'))
```

Checking the parameter related to the lifespan (years):

This parameter is a multiplier of the amount of energy reported in the use phase of the building. Thus, if the curtain wall is used for 40 years, the LCIA is first conducted for one year of use. The resulting impact values will then be multiplied by the number of years.

[148]:
```python
for p in DatabaseParameter.select():
    if p.name == 'param_servicelife':
        print(p.amount)
```

```
1.0
```

[149]:
```python
lifespan = 40
```

10.2 Functions to Perform the LCAs

The following functions conduct the LCIA according to activities and parameters set in the df_step DataFrame, i.e., including the type of glazing, the use or not of shading devices and contribution of each glazing to the building energy use:

[150]:
```python
def lca_cw_gwp(df_step, act, fu):
    """
    Perform a simple lca for different scenarios
    according to parameters defined in df_step

    Parameters
    ----------
    df_step: DataFrame with list of parameters and their values
    act: activity to assess
    fu: functional unit

    Returns
    -------
    ls_results: list of values for IPCC GWP
    """

    # A list to save the results:
    ls_results = []

    # Defining a new dataframe only with parameters useful for the LCIA:
    df_param = df_step.drop(['glazing',
                             'heating_setpoint',
                             'cooling_setpoint'], axis=1
                           )

    # Converting the dataframe in a numpy array:
    val_np = df_param.to_numpy()

    n_scenario = 0

    for v in val_np:
        name_scenario = df_param.index[n_scenario]
        n_scenario += 1

        for param_name in df_param.columns:
            # Change parameters according to column name:
            n = df_param.columns.get_loc(param_name)

            (ActivityParameter.update(amount=v[n])
             .where(ActivityParameter.name == f'param_{param_name}').execute())

        ActivityParameter.recalculate_exchanges("cw_use_param_group")
        ActivityParameter.recalculate_exchanges("cw_eol_param_group")

        # Conducting the LCIA:
        lca = LCA({act: fu}, method_ilcd_gwp)
        lca.lci()
```

```python
        lca.lcia()
        ls_results.append({'run': name_scenario, 'result': lca.score})

    return ls_results
```

The next function performs a multi_method LCIA, with the impact categories listed in "ls_method_small", according to activities and parameter set:

[151]:
```python
# Reminder of the small list of impact categories:
for method in ls_method_small:
    print(method[1], ": ", method[2])
```

```
climate change :  climate change total
ecosystem quality :  freshwater ecotoxicity
ecosystem quality :  freshwater and terrestrial acidification
ecosystem quality :  freshwater eutrophication
ecosystem quality :  terrestrial eutrophication
human health :  ozone layer depletion
human health :  photochemical ozone creation
resources :  fossils
resources :  land use
```

[152]:
```python
def lca_cw_mlca_small(df_step, act, fu):
    """
    Perform a multi-method lca for different scenarios
    according to parameters defined in df_step.

    Parameters
    ----------
    df_step: DataFrame with list of parameters and their values
    act: activity to assess
    fu: functional unit

    Returns
    -------
    ls_mlca_small_results: list of values
    """

    # A list to save the results:
    ls_mlca_small_results = []

    # Defining a new dataframe only with parameters useful for the LCIA:
    df_param = df_step.drop(['glazing',
                             'heating_setpoint',
                             'cooling_setpoint'], axis=1
                           )

    # Converting dataframe in a numpy array:
```

```
    val_np = df_param.to_numpy()

    n_scenario = 0

    for v in val_np:
        name_scenario = df_param.index[n_scenario]
        n_scenario += 1

        for param_name in df_param.columns:
            # Change parameters according to column name:
            n = df_param.columns.get_loc(param_name)

            (ActivityParameter.update(amount=v[n])
             .where(ActivityParameter.name == f'param_{param_name}')
             .execute()
             )

        ActivityParameter.recalculate_exchanges("cw_use_param_group")
        ActivityParameter.recalculate_exchanges("cw_eol_param_group")

        lca = LCA({act: fu})
        lca.lci()

        # Conducting the LCIA:
        for method in ls_method_small:
            lca.switch_method(method)
            lca.lcia()
            ls_mlca_small_results.append((name_scenario,
                                          method[1], method[2],
                                          lca.score,
                                          bw.methods.get(method).get('unit'))
                                          )

    return ls_mlca_small_results
```

The next function performs a multi_method LCIA, with the impact categories listed in "ls_method_full", according to activities and parameter set:

[153]:
```
# Reminder of the full list of impact categories:
for method in ls_method_full:
    print(method[1], ": ", method[2])
```

```
climate change :  climate change total
ecosystem quality :  freshwater ecotoxicity
ecosystem quality :  freshwater and terrestrial acidification
ecosystem quality :  freshwater eutrophication
ecosystem quality :  marine eutrophication
ecosystem quality :  terrestrial eutrophication
```

```
human health :  non-carcinogenic effects
human health :  carcinogenic effects
human health :  ionising radiation
human health :  ozone layer depletion
human health :  photochemical ozone creation
human health :  respiratory effects, inorganics
resources :  minerals and metals
resources :  dissipated water
resources :  fossils
resources :  land use
```

[154]:
```
def lca_cw_mlca_full(df_step, act, fu):
    """
    Perform a multi-method lca for different scenarios
    according to parameters defined in df_step.
    Methods= ReCiPe: GWP100, ODPinf, PMFP, POFP

    Parameters
    ----------
    df_step: DataFrame with list of parameters and their values
    act: activity to assess
    fu: functional unit

    Returns
    -------
    ls_mlca_full_results: list of values
    """

    # A list to save the results:
    ls_mlca_full_results = []

    # Defining a new dataframe only with parameters useful for the LCIA:
    df_param = df_step.drop(['glazing', 'heating_setpoint',
                             'cooling_setpoint'], axis=1)

    # Converting dataframe in a numpy array:
    val_np = df_param.to_numpy()

    n_scenario = 0

    for v in val_np:
        name_scenario = df_param.index[n_scenario]
        n_scenario += 1

        for param_name in df_param.columns:
            n = df_param.columns.get_loc(param_name)
```

```
            (ActivityParameter.update(amount=v[n])
             .where(ActivityParameter.name == f'param_{param_name}').execute())

        ActivityParameter.recalculate_exchanges("cw_use_param_group")
        ActivityParameter.recalculate_exchanges("cw_eol_param_group")

        lca = LCA({act: fu})
        lca.lci()

        # Conducting the LCIA:
        for method in ls_method_full:
            lca.switch_method(method)
            lca.lcia()
            ls_mlca_full_results.append((name_scenario,
                                         method[1], method[2],
                                         lca.score,
                                         bw.methods.get(method).get('unit'))
                                         )

    return ls_mlca_full_results
```

A little function to transform a list of mlca_results into a DataFrame:

[155]:
```
def ls_to_df_mlca(ls):
    """
    A little function to transform the ls_mlca_results
    in a readable DataFrame

    Parameters
    ----------
    ls: the list

    Returns
    -------
    df: the DataFrame
    """

    # DataFrame to then work w/ results:
    df = pd.DataFrame(ls,
                      columns=["Name",
                               "Category",
                               "Subcategory",
                               "Score",
                               "Unit"
                               ]
                      )
```

```
    df = pd.pivot_table(df,
                        index=["Name"],
                        columns=["Category",
                                 "Subcategory",
                                 "Unit"
                                 ],
                        values="Score"
                        )

    return df
```

A function to save mlca_full_results in a DataFrame, for each simulation run and LCA phase:

[156]:
```
def ls_to_df_mlca_full(step, ls, act, df_results):
    """
    A function to append a list of mlca results in a DataFrame,
        with values organised per simulation run (index),
        and LCA phase (columns).

    Parameters
    ----------
    step: correspond to the batch of simulation runs: step_1, 2...
    ls: the list of results.
    df_results: a DataFrame where LCA results will be saved.
    act: activity for which the LCA has been done.

    Returns
    -------
    df_results
    """

    # New DataFrame from list of results:
    df_temp = pd.DataFrame(ls,
                           columns=["Name",
                                    "Category",
                                    "Subcategory",
                                    "Score",
                                    "Unit"
                                    ]
                           )

    # Add information regarding the step:
    df_temp["Step"] = step
    # Add information regarding the LCA phase:
    df_temp["LCA Phase"] = str(act["name"])

    # Pivot the DataFrame:
```

```
    df_temp = pd.pivot_table(df_temp,
                             index=["Step",
                                    "Name"
                                    ],
                             columns=["LCA Phase",
                                      "Category",
                                      "Subcategory",
                                      "Unit"
                                      ],
                             values="Score"
                             )
    # Merge with existing results:
    if df_results.empty:
        df_results = df_temp
        print("empty, df_results replaced")
    else:
        # Merge by columns_to_use:
        df_results = pd.concat(
            [df_results, df_temp[~df_temp.index.isin(df_results.index)]]
        )
        df_results.update(df_temp)

    return df_results
```

A function to conduct a multi-LCA per activity and save the results in a DataFrame, for each simulation run and each LCA phase:

[157]:
```
def full_lca_to_df(step, df_step, df_results, fu):
    """
    A function to conduct an LCA, using the function lca_cw_mlca_full,
        and append a list of mlca results in a DataFrame, using the
        function: ls_to_df_mlca_full,
        with values organised per simulation run (index),
        and LCA phase (columns).

    Parameters
    ----------
    step: correspond to the batch of simulation runs: step_1, 2...
    df_step: DataFrame with list of parameters and their values

    Returns
    -------
    df_results: a DataFrame where LCA results are saved,
        simulation run as index, LCA phase and impact indicators as columns.
    """

    for act in [prod_cw, use_bldg_w_cw, repair_cw, eol_cw]:
```

```
        ls = lca_cw_mlca_full(df_step, act, fu)
        df_results = ls_to_df_mlca_full(step, ls, act, df_results)

    return df_results
```

10.3 Life Cycle Impact Assessment of Curtain Wall Systems, Cradle-to-Grave

10.3.1 Calculation

This section conducts the full life cycle assessment according to the ILCD midpoint method. The impact is assessed for each configuration and life cycle phase and then saved.

The following boolean defines whether the LCIA if conducted (True) or if the csv file, where previous results are stored, is directly imported (False).

If the calculation is undertaken, be patient, it takes time!

[158]:
```
# Conducting the LCIA?
calc_lcia = False
```

[159]:
```
if calc_lcia:
    # Initialise a DataFrame:
    df_mlca_full_raw_results = pd.DataFrame()

    # LCIA calculation:
    ls_df_step = [
        df_step1, df_step2, df_step3, df_step4,
        df_step5, df_step6, df_step7, df_step8,
        df_step9, df_step10, df_step11, df_step12,
        df_step13, df_step14, df_step15, df_step16
    ]

    n = 1
    for df in ls_df_step:
        step = "step_"+str(n)
        df_mlca_full_raw_results = full_lca_to_df(step,
                                                   df,
                                                   df_mlca_full_raw_results,
                                                   1
                                                   )
        n += 1

    # Save df_mlca_full_raw_results to csv:
    df_mlca_full_raw_results.unstack([0, 1]).to_csv(
        'outputs\lca\df_mlca_full_raw_results.csv', index=True)

else:
    # Open the csv file, to avoid recalculating the impacts:
    if os.path.isfile('outputs\lca\df_mlca_full_raw_results.csv'):
```

```
        with pd.option_context('display.precision', 10):
            df_mlca_full_raw_results = (
                pd.read_csv(
                    'outputs\lca\df_mlca_full_raw_results.csv',
                    float_precision=None)
            )
        df_mlca_full_raw_results = df_mlca_full_raw_results.pivot_table(
            values='0',
            index=['Step', 'Name'],
            columns=['LCA Phase', 'Category', 'Subcategory', 'Unit']
        )

    else:
        print("df_mlca_full_raw_results does not exist!")
```

10.3.2 Navigating trough the LCIA Result DataFrame

Listing the impact categories:

[160]:
```
df_ilcd_methods = pd.DataFrame()
n = 0
ls_n = []
ls_ic = []
ls_ic_details = []
ls_u = []
for method in ls_method_full:
    ls_n.append(n + 1)
    ls_ic.append(method[1])
    ls_ic_details.append(method[2])
    ls_u.append(bw.methods.get(method).get('unit'))
    n += 1

df_ilcd_methods["Category"] = ls_ic
df_ilcd_methods["Subcategory"] = ls_ic_details
df_ilcd_methods["Unit"] = ls_u
df_ilcd_methods["#"] = ls_n

df_ilcd_methods = df_ilcd_methods.set_index(["Category", "#"])

df_ilcd_methods
```

[160]:
```
                                                       Subcategory        Unit
Category          #
climate change    1                           climate change total   kg CO2-Eq
ecosystem quality 2                         freshwater ecotoxicity         CTU
                  3   freshwater and terrestrial acidification   mol H+-Eq
                  4                       freshwater eutrophication     kg P-Eq
```

```
                   5                  marine eutrophication     kg N-Eq
                   6             terrestrial eutrophication    mol N-Eq
human health       7              non-carcinogenic effects        CTUh
                   8                  carcinogenic effects        CTUh
                   9                    ionising radiation  kg U235-Eq
                   10                 ozone layer depletion  kg CFC-11.
                   11         photochemical ozone creation  kg NMVOC-.
                   12        respiratory effects, inorganics  disease i.
resources          13                  minerals and metals    kg Sb-Eq
                   14                      dissipated water  m3 water-.
                   15                               fossils   megajoule
                   16                              land use      points
```

Selecting the impact category to display:

[161]:
```python
# Define the rank of the impact category (#):
n = 3
```

Selecting the analysis step, or simulation batch, with series of simulation runs:

[162]:
```python
step = "step_16"
```

Displaying the LCIA results:

[163]:
```python
ic = df_ilcd_methods.xs(n, level=1)["Subcategory"][0]
df_mlca_full_raw_results.loc[step].xs(ic, axis=1,
                                      level=2, drop_level=False
                                      )
```

[163]:
```
LCA Phase                        curtain wall, end of life  \
Category                                   ecosystem quality
Subcategory         freshwater and terrestrial acidification
Unit                                              mol H+-Eq
Name
p_a_1927_dg_init_cc                                0.011261
p_b_1927_dg0_cc                                    0.011261
p_c_1927_dg4_cc                                    0.011508
p_d_1927_dg5_cc                                    0.011508
p_e_1927_dg6_cc                                    0.011508
p_f_1927_tg4_cc                                    0.014659
p_g_1927_tg5_cc                                    0.014659
p_h_1927_tg6_cc                                    0.014659

LCA Phase                        curtain wall, maintenance  \
Category                                   ecosystem quality
Subcategory         freshwater and terrestrial acidification
Unit                                              mol H+-Eq
Name
p_a_1927_dg_init_cc                                0.009725
```

```
p_b_1927_dg0_cc                                              0.009725
p_c_1927_dg4_cc                                              0.009725
p_d_1927_dg5_cc                                              0.009725
p_e_1927_dg6_cc                                              0.009725
p_f_1927_tg4_cc                                              0.009725
p_g_1927_tg5_cc                                              0.009725
p_h_1927_tg6_cc                                              0.009725

LCA Phase                              curtain wall, production  \
Category                                        ecosystem quality
Subcategory         freshwater and terrestrial acidification
Unit                                                    mol H+-Eq
Name
p_a_1927_dg_init_cc                                      2.693631
p_b_1927_dg0_cc                                          2.693631
p_c_1927_dg4_cc                                          2.719942
p_d_1927_dg5_cc                                          2.719942
p_e_1927_dg6_cc                                          2.726383
p_f_1927_tg4_cc                                          2.915474
p_g_1927_tg5_cc                                          2.915474
p_h_1927_tg6_cc                                          2.915474

LCA Phase           use of glazed office building, hvac and lighting
Category                                                ecosystem quality
Subcategory                 freshwater and terrestrial acidification
Unit                                                            mol H+-Eq
Name
p_a_1927_dg_init_cc                                             -0.034002
p_b_1927_dg0_cc                                                 -0.034215
p_c_1927_dg4_cc                                                 -0.034402
p_d_1927_dg5_cc                                                 -0.035207
p_e_1927_dg6_cc                                                 -0.033394
p_f_1927_tg4_cc                                                 -0.027958
p_g_1927_tg5_cc                                                 -0.031667
p_h_1927_tg6_cc                                                 -0.029403
```

10.3.3 LCIA of the Disposal Phase of the Existing Curtain Wall

The case study focuses on the replacement of a curtain wall. The first step therefore consists of disassembling the existing sturcture, its glazing and frame, and disposing of them.

[164]:
```python
# Conducting the LCIA of the disposal phase ("out_old_cw" activity):
ls_mlca_oldcw_results = []

lca = bw.LCA({out_old_cw: 1})
lca.lci()
for method in ls_method_full:
```

```
    lca.switch_method(method)
    lca.lcia()
    ls_mlca_oldcw_results.append((method[1], method[2],
                                  lca.score,
                                  bw.methods.get(method).get('unit')))
```

[165]:
```
# Organising the DataFrame with the LCIA results:
df_mlca_oldcw_results = pd.DataFrame(ls_mlca_oldcw_results,
                                     columns=["Category",
                                              "Subcategory",
                                              "Score",
                                              "Unit"]
                                     )

df_mlca_oldcw_results = pd.pivot_table(df_mlca_oldcw_results,
                                       columns=["Category",
                                                "Subcategory",
                                                "Unit"],
                                       values="Score"
                                       )

df_mlca_oldcw_results
```

[165]:
```
Category          climate change                          ecosystem quality  \
Subcategory climate change total freshwater and terrestrial acidification
Unit                   kg CO2-Eq                                mol H+-Eq
Score                   2.217209                                 0.010066

Category                                                           \
Subcategory freshwater ecotoxicity freshwater eutrophication
Unit                          CTU                     kg P-Eq
Score                    4.239562                    0.000168

Category                                                           \
Subcategory marine eutrophication terrestrial eutrophication
Unit                     kg N-Eq                     mol N-Eq
Score                   0.003623                     0.039469

Category            human health                                          \
Subcategory carcinogenic effects ionising radiation non-carcinogenic effects
Unit                        CTUh         kg U235-Eq                     CTUh
Score                4.432451e-08          0.154242             2.605557e-07

Category                                                             \
Subcategory ozone layer depletion photochemical ozone creation
Unit                   kg CFC-11.                   kg NMVOC-.
Score                3.954436e-07                     0.011283
```

```
Category                                                   resources               \
Subcategory respiratory effects, inorganics dissipated water    fossils
Unit                                  disease i.       m3 water-.  megajoule
Score                               1.674613e-07         0.287208  27.764659

Category
Subcategory   land use minerals and metals
Unit            points              kg Sb-Eq
Score        39.309361              0.000039
```

11 Post-Processing the LCIA Results over 40 years of Service Life

11.1 Calculating the Evolution of the Environmental Impact over 40 years

In this section, the LCIA results are combined to construct the environmental trajectory of each scenario over 40 years of service life. This means that the existing curtain wall is first dismantled, then a new one is produced, installed and used for 40 years with maintenance steps every 10 years, and finally comes its end of life. The following code therefore describes each of these steps in a new DataFrame.

[166]:
```
df_lca_lifespan = pd.DataFrame(
    {'Year': np.arange(lifespan+2),
     'Step': 'ref',
     'Scenario': 'no_retrofit',
     'Category': 'All',
     'Subcategory': 'All',
     'Unit': 'None',
     'Score': 0
     }
)

df_lca_lifespan = df_lca_lifespan.pivot(index='Year',
                                        columns=['Step',
                                                 'Scenario',
                                                 'Category',
                                                 'Subcategory',
                                                 'Unit'
                                                 ],
                                        values='Score'
                                        )
```

[167]:
```
# Defining the columns, one for each simulation run:
for run in df_mlca_full_raw_results.reset_index(level=0).index:
    n_step = df_mlca_full_raw_results.reset_index(level=0)["Step"].loc[run]
    for key in df_ilcd_methods.index:
        i = key[0]
```

```
        ic = df_ilcd_methods.loc[key]["Subcategory"]
        u = df_ilcd_methods.loc[key]["Unit"]
        # define a new column:
        df_lca_lifespan[n_step, run, i, ic, u] = 0.0

df_lca_lifespan = df_lca_lifespan.drop("ref", axis=1)
```

```
C:\Users\souvi\AppData\Local\Temp/ipykernel_3832/2198440881.py:9:
PerformanceWarning: DataFrame is highly fragmented.  This is usually the result
of calling `frame.insert` many times, which has poor performance.  Consider
joining all columns at once using pd.concat(axis=1) instead.  To get a de-
fragmented frame, use `newframe = frame.copy()`
  df_lca_lifespan[n_step, run, i, ic, u] = 0.0
```

[168]:
```
# LCIA over the 40 years of the service life of the curtain wall:
for step, run, i, ic, u in df_lca_lifespan.columns:
    # First phase of the LCA, disposal of the existing curtain wall
    # and production/construction of the new curtain wall:
    df_lca_lifespan.loc[0][step, run, i, ic, u] = (
        df_mlca_oldcw_results[i, ic, u]
        + df_mlca_full_raw_results.reset_index(level=0).loc[run][
            "curtain wall, production", i, ic, u
        ]
    )

    # Second phase, use of the curtain wall, indirect energy use impacts:
    for y in range(1, 41):
        df_lca_lifespan.loc[y][step, run, i, ic, u] = (
            df_lca_lifespan.loc[y-1][step, run, i, ic, u] +
            df_mlca_full_raw_results.reset_index(level=0).loc[run][
                "use of glazed office building, hvac and lighting", i, ic, u
            ]
        )

        if (y == 12 or y == 22 or y == 32):
            # Impacts relating to maintenance, every 10y:
            df_lca_lifespan.loc[y][step, run, i, ic, u] += (
                df_mlca_full_raw_results.reset_index(level=0).loc[run][
                    "curtain wall, maintenance", i, ic, u
                ]
            )
        if y == 25:
            if run == "i_g_2126_dg_smart":
                df_lca_lifespan.loc[y][step, run, i, ic, u] += (0.25 *
                                                                        ␣
 ↪df_mlca_full_raw_results.reset_index(level=0).loc[run][
```

```
                                                                    "curtain␣
↪wall, production", i, ic, u
                                                                ]
                                                                )


    # Last phase, end-of-life of the new curtain wall:
    df_lca_lifespan.loc[41][step, run, i, ic, u] = (
        df_lca_lifespan.loc[40][step, run, i, ic, u] +
        df_mlca_full_raw_results.reset_index(level=0).loc[run][
            "curtain wall, end of life", i, ic, u
        ]
    )
```

Post-processing the LCA results to take into consideration climate change:

[169]:
```
# Names for the simulations run in 2020 which corresponds
# to the same configuration as the one integrating climate change,
# step_14: step_5; step_15: step_10; step_16: step_11:
run_cc = {
    "n_a_2126_dg_init_cc": "e_a_2126_dg_init_vav_int",
    "n_b_2126_dg0_cc": "e_b_2126_dg0_vav_int",
    "n_c_2126_dg4_cc": "e_h_2126_dg4_vav_int",
    "n_d_2126_dg5_cc": "e_i_2126_dg5_vav_int",
    "n_e_2126_dg6_cc": "e_j_2126_dg6_vav_int",
    "n_f_2126_tg4_cc": "e_n_2126_tg4_vav_int",
    "n_g_2126_tg5_cc": "e_o_2126_tg5_vav_int",
    "n_h_2126_tg6_cc": "e_p_2126_tg6_vav_int",
    "o_a_2124_dg_init_cc": "j_a_2124_dg_init",
    "o_b_2124_dg0_cc": "j_b_2124_dg0",
    "o_c_2124_dg4_cc": "j_c_2124_dg4",
    "o_d_2124_dg5_cc": "j_d_2124_dg5",
    "o_e_2124_dg6_cc": "j_e_2124_dg6",
    "o_f_2124_tg4_cc": "j_f_2124_tg4",
    "o_g_2124_tg5_cc": "j_g_2124_tg5",
    "o_h_2124_tg6_cc": "j_h_2124_tg6",
    "p_a_1927_dg_init_cc": "k_a_1927_dg_init_ext",
    "p_b_1927_dg0_cc": "k_b_1927_dg0_ext",
    "p_c_1927_dg4_cc": "k_c_1927_dg4_ext",
    "p_d_1927_dg5_cc": "k_d_1927_dg5_ext",
    "p_e_1927_dg6_cc": "k_e_1927_dg6_ext",
    "p_f_1927_tg4_cc": "k_f_1927_tg4_ext",
    "p_g_1927_tg5_cc": "k_g_1927_tg5_ext",
    "p_h_1927_tg6_cc": "k_h_1927_tg6_ext"
}
```

[170]:
```
# Corresponding simulations:
steps_cc = {"step_14": "step_5",
```

```
            "step_15": "step_10",
            "step_16": "step_11"
            }
```

Two weather files are used: the first one corresponds to the average data of our time, the other one to the RCP 8.5 scenario in 2060, i.e., in 40 years. Thus, two results for each impact indicator bound the service life for each scenario. Between them, the data is linearly interpolated:

[171]:
```
# LCIA over 40years for climate change scenario:
for step, run, i, ic, u in df_lca_lifespan.columns:
    if step in steps_cc.keys():
        # Modification of the first year in use:
        df_lca_lifespan.loc[1][step, run, i, ic, u] = (
            df_lca_lifespan.loc[1][steps_cc[step], run_cc[run], i, ic, u]
        )

        # Delete data between year 1 and year 40:
        for y in range(2, 40):
            df_lca_lifespan.loc[y][step, run, i, ic, u] = np.nan

        # Interpolate between year 1 and year 40:
        df_lca_lifespan[step, run, i, ic, u] = (
            df_lca_lifespan[step, run, i, ic, u].interpolate(
                method='linear')
        )

        # Last phase, end-of-life of the new curtain wall:
        df_lca_lifespan.loc[41][step, run, i, ic, u] += (
            df_mlca_full_raw_results.reset_index(level=0).loc[run][
                "curtain wall, end of life", i, ic, u
            ]
        )
```

11.2 Setup to Create the Graphs

Defining a function to divide/multiply the y-axis by a thousand, if needed:

[173]:
```
def thousand_divide(x, pos):
    # The two args are the value and tick position
    return '%1.1f' % (x*1e-3)


def thousand_multiply(x, pos):
    # The two args are the value and tick position
    return '%1.1f' % (x*1e+3)
```

[174]:
```
formatter = FuncFormatter(thousand_divide)
```

Reorganising the DataFrame with the LCIA results for the 40 years of service life. Integrating the type of IGU for each simulation run:

[175]:
```
# Add a row index to sort by step (column indexing):
df_lca_lifespan = df_lca_lifespan.T

# Add a row to sort by IGU type (column indexing):
ls_igu = []
for code in df_lca_lifespan.index.get_level_values('Scenario'):
    if "dg_init" in code:
        ls_igu.append("dg_init")
    if "dg0" in code:
        ls_igu.append("dg0")
    if "sg" in code:
        ls_igu.append("sg")
    if (("dg1" in code) or ("dg2" in code) or ("dg3" in code)
            or ("dg4" in code) or ("dg5" in code) or ("dg6" in code)):
        ls_igu.append("dg")
    if (("tg1" in code) or ("tg2" in code) or ("tg3" in code)
            or ("tg4" in code) or ("tg5" in code) or ("tg6" in code)):
        ls_igu.append("tg")
    if "dg_vacuum" in code:
        ls_igu.append("dg_vacuum")
    if "dg_smart" in code:
        ls_igu.append("dg_smart")
    if "dsf" in code:
        ls_igu.append("dsf")
    if "ccf" in code:
        ls_igu.append("ccf")

df_lca_lifespan.loc[:, ('IGU')] = ls_igu

df_lca_lifespan = df_lca_lifespan.reset_index().set_index(
    ["Step", "Scenario", "IGU", "Category", "Subcategory", "Unit"]
).T
```

An overview of the resulting DataFrame:

Defining a function to display the evolution of the environmental impact over 40 years according to a specific indicator:

[177]:
```
def plot_lca_40(step_lines, impact_cat, ls_lineplot,
                ylim_min, ylim_max, ylim_gap
                ):
    """
    Plot the curves according to the data defined in ls_lineplot
        (i.e. sg, dg, dg_init, tg).
        x = 40 years of service life. 41 year corresponds to the end
```

```
        of timeline including 1 year of deconstruction.

    Parameters
    ----------
    step_lines: a string, step where the simulations and LCA results
        are taken from, plot as curves
    impact_cat: an int which correspond to the reference number
        in df_ilcd_methods
    ls_lineplot: list of simulation runs to plot, according to simplified
        igu name (i.e. sg, dg, dg_init, tg)
    ylim_min: an integer, minimum value of the y-axis.
    ylim_max: an integer, maximum value of the y-axis.
    ylim_gap: gap between each ytick.

    Returns
    -------
    df: the DataFrame with impact values at year = 41
    """

    ls_impact_eol = {}

    # Defining the LCA results to plot:
    ic = df_ilcd_methods.xs(impact_cat, level=1)["Subcategory"][0]
    i = df_ilcd_methods.index[df_ilcd_methods['Subcategory'] == ic][0][0]
    unit = df_ilcd_methods.xs(impact_cat, level=1)["Unit"][0]
    df_lca_step = df_lca_lifespan[step_lines].xs(ic,
                                                 axis=1, level=3,
                                                 drop_level=False
                                                 )

    print(step)
    print(i, ", ", ic)
    print("Unit is:", unit)

    fig, ax = plt.subplots(figsize=(9, 3))

    # Evolution of the GWP over 40 years:
    for run, igu, i, ic, u in df_lca_step.columns:
        # Then, we plot the curves:
        if ('dg_init' in run) and ('dg_init' in ls_lineplot):
            sns.lineplot(x=df_lca_step.index,
                         y=df_lca_step[run, igu, i, ic, u].tolist(),
                         color='black',
                         ax=ax
                         )

        elif ('dg0' in run) and ('dg0' in ls_lineplot):
```

```
            sns.lineplot(x=df_lca_step.index,
                         y=df_lca_step[run, igu, i, ic, u].tolist(),
                         color='black',
                         linestyle='--',
                         ax=ax
                         )

        elif ('sg' in run) and ('sg' in ls_lineplot):
            sns.lineplot(x=df_lca_step.index,
                         y=df_lca_step[run, igu, i, ic, u].tolist(),
                         color='lightsalmon',
                         ax=ax
                         )

        elif ('dg' in run) and ('dg' in ls_lineplot):
            sns.lineplot(x=df_lca_step.index,
                         y=df_lca_step[run, igu, i, ic, u].tolist(),
                         color='darksalmon',
                         ax=ax
                         )

        elif ('tg' in run) and ('tg' in ls_lineplot):
            sns.lineplot(x=df_lca_step.index,
                         y=df_lca_step[run, igu, i, ic, u].tolist(),
                         color='firebrick',
                         ax=ax
                         )

        elif ('dg_vacuum' in run) and ('dg_vacuum' in ls_lineplot):
            sns.lineplot(x=df_lca_step.index,
                         y=df_lca_step[run, igu, i, ic, u].tolist(),
                         color='cornflowerblue',
                         ax=ax
                         )

        elif ('dg_smart' in run) and ('dg_smart' in ls_lineplot):
            sns.lineplot(x=df_lca_step.index,
                         y=df_lca_step[run, igu, i, ic, u].tolist(),
                         color='royalblue',
                         ax=ax
                         )

        elif ('dsf' in run) and ('dsf' in ls_lineplot):
            sns.lineplot(x=df_lca_step.index,
                         y=df_lca_step[run, igu, i, ic, u].tolist(),
                         color='darkgreen',
                         ax=ax
```

```
                        )

        elif ('ccf' in run) and ('ccf' in ls_lineplot):
            sns.lineplot(x=df_lca_step.index,
                        y=df_lca_step[run, igu, i, ic, u].tolist(),
                        color='midnightblue',
                        ax=ax
                        )
        else:
            continue

        # Update the dictionary with value at year = 41:
        ls_impact_eol[run] = df_lca_step.loc[41][run, igu, i, ic, u]

    ax.set_xlim(0, 41)
    ax.axhline(y=0, c='grey', linestyle='-', linewidth=0.75)

    ax.set_ylim(ylim_min, ylim_max)
    plt.yticks(np.arange(ylim_min, ylim_max+1, ylim_gap))
    ax.grid(which='major', axis='y', linestyle=':', linewidth=1)

    ax.xaxis.label.set_visible(False)
    ax.yaxis.label.set_visible(False)

    style_ax(ax)

    sns.despine(offset=5, bottom=True, left=True)
    plt.show()

    df = pd.DataFrame.from_dict(ls_impact_eol, orient='index',
                                columns=['impact at year = 41, '+str(u)])

    return df
```

Same function as above, but displaying one chart per category of glazin (single, double, triple…):

[178]:
```
def plot_multilca_40(step, impact_cat, var,
                     ylim_min, ylim_max, ylim_gap):
    """
    Plot the curves of the specific step over 40 years.
        x = 40 years of service life. 41 year corresponds to the end
        of timeline including 1 year of deconstruction.

    Parameters
    ----------
    step: a string, step where the simulations and LCA results
        are taken from, plot as curves
```

```
    impact_cat: an int which correspond to the reference number
        in df_ilcd_methods
    var: string, "Scenario" or "IGU".
    ylim_min: an integer, minimum value of the y-axis.
    ylim_max: an integer, maximum value of the y-axis.
    ylim_gap: gap between each ytick.

    Returns
    -------
    df: the DataFrame with impact values at year = 41
    """

    ls_impact_eol = {}

    # Defining the LCA results to plot:
    ic = df_ilcd_methods.xs(impact_cat, level=1)["Subcategory"][0]
    i = df_ilcd_methods.index[df_ilcd_methods['Subcategory'] == ic][0][0]
    unit = df_ilcd_methods.xs(impact_cat, level=1)["Unit"][0]

    print(step)
    print(i, ",", ic)
    print("Unit is:", unit)

    df_lca_step = df_lca_lifespan[step].xs(ic,
                                           axis=1,
                                           level=3,
                                           drop_level=False
                                           )

    df_lca_step.columns = df_lca_step.columns.droplevel([2, 3])
    df_plot = df_lca_step.stack(level=[0, 1])

    col_name = df_plot.columns[0]
    df_plot = df_plot.reset_index()

    # Plot each year's time series in its own facet
    g = sns.relplot(
        data=df_plot,
        x="Year", y=col_name, col=var, hue="Scenario",
        kind="line", palette="crest", linewidth=1.5, zorder=5,
        col_wrap=2, height=2.5, aspect=1.5, legend=False,
    )

    # Iterate over each subplot to customize further
    for year, ax in g.axes_dict.items():

        # Add the title as an annotation within the plot
```

```
        ax.text(.1, .95, year, transform=ax.transAxes,
                fontweight="bold", fontsize=12)

        # Plot every year's time series in the background
        sns.lineplot(
            data=df_plot, x="Year", y=col_name, units="Scenario",
            estimator=None, color=".7", linewidth=0.5, ax=ax,
        )

        ax.axhline(y=0, c='salmon', linestyle='-', linewidth=0.75)

        style_ax(ax)

        ax.set_ylim(ylim_min, ylim_max)
        plt.yticks(np.arange(ylim_min, (ylim_max+1), ylim_gap))

        ax.set_xlim(0, 41)
        plt.xticks(np.arange(0, 42, 10))

    g.set_titles("")
    g.set_axis_labels("", "")
    g.tight_layout()

    for run in df_lca_step.columns:
        if df_lca_step.loc[41][run] < 0:
            a, b = df_lca_step.iloc[(df_lca_step[run]-0)
                                    .abs().argsort()[:2]].index
        else:
            a = "na"
            b = "na"
        ls_impact_eol[run[0]] = (df_lca_step.loc[41][run], max(a, b))

    df = pd.DataFrame.from_dict(ls_impact_eol, orient='index',
                                columns=['impact at year = 41, '+str(u),
                                         "year at net-zero"
                                         ]
                                )

    if export:
        # Save image:
        g.savefig(os.path.join(path_img, 'PayBack_LCA_'+str(step)+'.png'),
                  dpi=600, bbox_inches='tight')
        g.savefig(os.path.join(path_img, 'PayBack_LCA_'+str(step)+'.pdf'),
                  bbox_inches='tight')

    return df
```

Reminder of the impact categories and their reference numbers:

[179]:
```
df_ilcd_methods
```

[179]:

Category	#	Subcategory	Unit
climate change	1	climate change total	kg CO2-Eq
ecosystem quality	2	freshwater ecotoxicity	CTU
	3	freshwater and terrestrial acidification	mol H+-Eq
	4	freshwater eutrophication	kg P-Eq
	5	marine eutrophication	kg N-Eq
	6	terrestrial eutrophication	mol N-Eq
human health	7	non-carcinogenic effects	CTUh
	8	carcinogenic effects	CTUh
	9	ionising radiation	kg U235-Eq
	10	ozone layer depletion	kg CFC-11.
	11	photochemical ozone creation	kg NMVOC-.
	12	respiratory effects, inorganics	disease i.
resources	13	minerals and metals	kg Sb-Eq
	14	dissipated water	m3 water-.
	15	fossils	megajoule
	16	land use	points

11.3 Weighting Stage

Weighting the LCIA results according to the PEF weighting factors:

[180]:
```
df_weighted = pd.DataFrame()

for step, run, igu, i, ic, unit in df_lca_lifespan.columns:
    ref = (step, run, igu)
    if ref not in df_weighted.columns:
        df_to_weight = df_lca_lifespan.xs(
            run, axis=1, level=1, drop_level=False).loc[[41]]

        df_to_weight.columns = df_to_weight.columns.droplevel([0, 1, 2])

        # Defining a new DataFrame with the weighted values,
        # i.e., multiplication of the impacts by df_weighting:
        df_weighted = df_weighted.append(pd.DataFrame(
            (df_to_weight.multiply(
                df_weighting["Weighting factor"].T, axis=1) / 100
             ).sum(axis=1), columns=[ref]
        ))

df_weighted.columns = df_weighted.columns.rename("Step", level=0)
df_weighted.columns = df_weighted.columns.rename("Run", level=1)
df_weighted.columns = df_weighted.columns.rename("IGU", level=2)

df_weighted = df_weighted.rename(index={41: 'Score'})
```

```
df_weighted = df_weighted.groupby(level=0).max().sort_index(axis=1, level=0)
```

12 Data Analysis

Reminder of the list of glazing units and their code name, as used in the data analysis below:

[181]:
```
dict_sg = {
    "sg_1": ("55.2, clear", 5.6, 0.8, 0.9, ""),
    "sg_2": ("|55.2, coated clear", 3.1, 0.4, 0.7, "")
}

df_sg = pd.DataFrame.from_dict(dict_sg, orient='index',
                               columns=['single glazing', 'U-value',
                                        'SHGC', 'VT', 'Overview']
                               )

dict_dg = {
    "dg_init": ("8-10Air-55.2_bronze", 2.7, 0.5, 0.4, "old one"),
    "dg_0": ("8-10Air-55.2_clear", 2.7, 0.7, 0.8, "low-perf clear"),
    "dg_1": ("8-18Arg-|55.2", 1.1, 0.6, 0.8, "high SHG, high LT"),
    "dg_2": ("8|-18Arg-55.2", 1.0, 0.4, 0.6, "mid SHG, mid LT"),
    "dg_3": ("8|-18Arg-55.2", 1.1, 0.4, 0.7, "mid SHG, high LT"),
    "dg_4": ("8|-18Arg-|55.2", 1.0, 0.2, 0.4, "low SHG, low LT"),
    "dg_5": ("8|-18Arg-55.2", 1.0, 0.3, 0.6, "low SHG, mid LT"),
    "dg_6": ("8|-18Arg-55.2", 1.0, 0.4, 0.7, "low SHG, high LT")
}

df_dg = pd.DataFrame.from_dict(dict_dg, orient='index',
                               columns=['double glazing', 'U-value',
                                        'SHGC', 'VT', 'Overview']
                               )

dict_tg = {
    "tg_1": ("8|-14Arg-6-14Arg-|55.2", 0.6, 0.5, 0.8, "high SHG, high LT"),
    "tg_2": ("8|-14Arg-6-14Arg-|55.2", 0.7, 0.4, 0.5, "mid SHG, mid LT"),
    "tg_3": ("8|-14Arg-6-14Arg-|55.2", 0.7, 0.4, 0.6, "mid SHG, high LT"),
    "tg_4": ("8|-14Arg-6-14Arg-|55.2", 0.6, 0.2, 0.3, "low SHG, lowLT"),
    "tg_5": ("8|-14Arg-6-14Arg-55.2", 0.6, 0.2, 0.5, "low SHG, mid LT"),
    "tg_6": ("8|-14Arg-6-14Arg-55.2", 0.6, 0.3, 0.6, "low SHG, high LT"),
}

df_tg = pd.DataFrame.from_dict(dict_tg, orient='index',
                               columns=['triple glazing', 'U-value',
                                        'SHGC', 'VT', 'Overview']
                               )
```

[182]:
```python
df_sg
```

[182]:
```
             single glazing  U-value  SHGC   VT Overview
sg_1           55.2, clear      5.6   0.8  0.9
sg_2   |55.2, coated clear      3.1   0.4  0.7
```

[183]:
```python
df_dg
```

[183]:
```
                 double glazing  U-value  SHGC   VT           Overview
dg_init  8-10Air-55.2_bronze       2.7   0.5  0.4            old one
dg_0      8-10Air-55.2_clear       2.7   0.7  0.8     low-perf clear
dg_1            8-18Arg-|55.2       1.1   0.6  0.8  high SHG, high LT
dg_2            8|-18Arg-55.2       1.0   0.4  0.6    mid SHG, mid LT
dg_3            8|-18Arg-55.2       1.1   0.4  0.7   mid SHG, high LT
dg_4           8|-18Arg-|55.2       1.0   0.2  0.4    low SHG, low LT
dg_5            8|-18Arg-55.2       1.0   0.3  0.6    low SHG, mid LT
dg_6            8|-18Arg-55.2       1.0   0.4  0.7   low SHG, high LT
```

[184]:
```python
df_tg
```

[184]:
```
                 triple glazing  U-value  SHGC   VT           Overview
tg_1  8|-14Arg-6-14Arg-|55.2       0.6   0.5  0.8  high SHG, high LT
tg_2  8|-14Arg-6-14Arg-|55.2       0.7   0.4  0.5    mid SHG, mid LT
tg_3  8|-14Arg-6-14Arg-|55.2       0.7   0.4  0.6   mid SHG, high LT
tg_4  8|-14Arg-6-14Arg-|55.2       0.6   0.2  0.3     low SHG, lowLT
tg_5   8|-14Arg-6-14Arg-55.2       0.6   0.2  0.5    low SHG, mid LT
tg_6   8|-14Arg-6-14Arg-55.2       0.6   0.3  0.6   low SHG, high LT
```

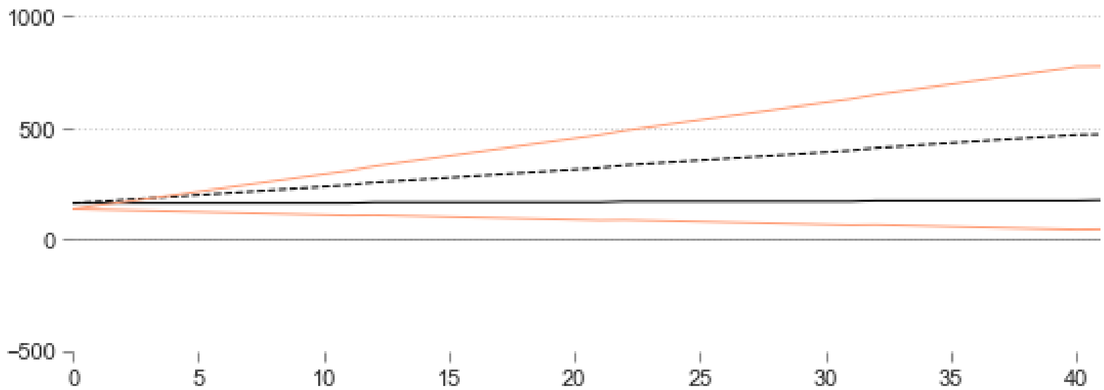
12.1 Steps 1-3: Glazing Performance and Façade Design, a Scenario Analysis

Initial configuration with fan coil chiller and natural gas boiler

12.1.1 Step 1: Without Shading Devices

[185]:
```python
# Define the simulation step, plot as curves:
step = "step_1"
# Define the rank of the impact category (#):
impact_cat = 1
# Simulation runs to print:
ls_igu = ['dg_init', 'sg', "dg0"]

# plot:
plot_lca_40(step, impact_cat, ls_igu, -500, 1000, 500)
```

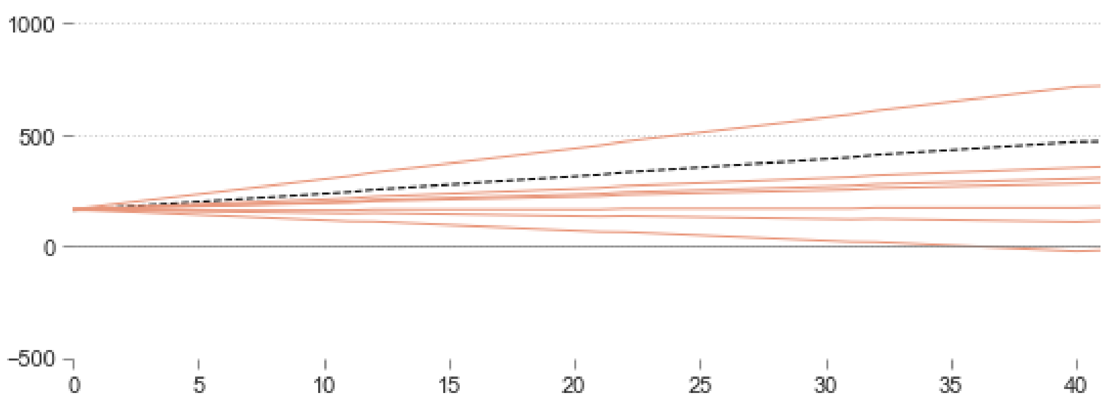
```
step_1
climate change ,  climate change total
Unit is: kg CO2-Eq
```



[185]:
```
                         impact at year = 41, kg CO2-Eq
a_a_2126_dg_init                             174.707163
a_b_2126_dg0                                 470.161355
a_c_2126_sg1                                 775.094163
a_d_2126_sg2                                  43.437485
```

[186]:
```python
ls_igu = ['dg', "dg0"]

# plot:
plot_lca_40(step, impact_cat, ls_igu,
            -500, 1000, 500
            )
```

```
step_1
climate change ,  climate change total
Unit is: kg CO2-Eq
```



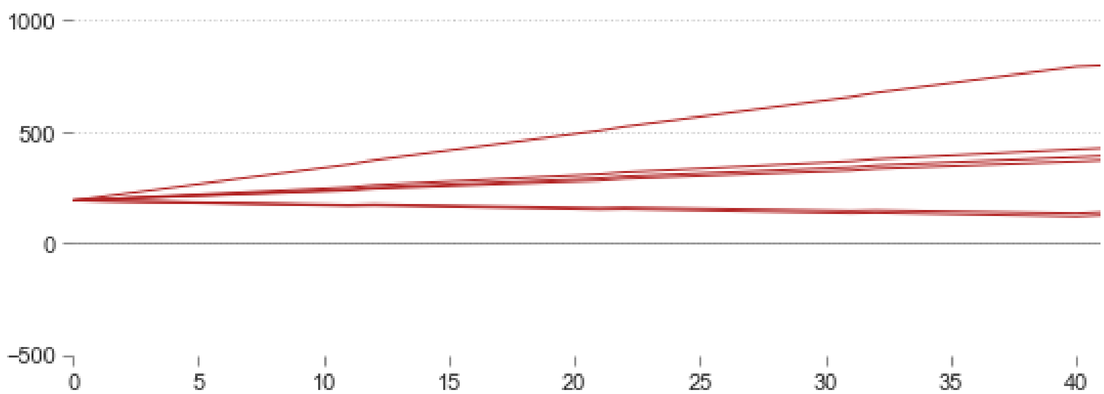
[186]:

	impact at year = 41, kg CO2-Eq
a_a_2126_dg_init	174.707163
a_b_2126_dg0	470.161355
a_e_2126_dg1	719.060213
a_f_2126_dg2	305.831126
a_g_2126_dg3	356.178657
a_h_2126_dg4	-18.826401
a_i_2126_dg5	111.617073
a_j_2126_dg6	283.923207

[187]:

```
ls_igu = ['tg']

# plot:
plot_lca_40(step, impact_cat, ls_igu,
            -500, 1000, 500
            )
```

```
step_1
climate change ,  climate change total
Unit is: kg CO2-Eq
```



[187]:

	impact at year = 41, kg CO2-Eq
a_k_2126_tg1	797.190097
a_l_2126_tg2	425.133957
a_m_2126_tg3	370.971449
a_n_2126_tg4	123.647686
a_o_2126_tg5	138.323590
a_p_2126_tg6	390.402460

Another kind of plot:

[188]:

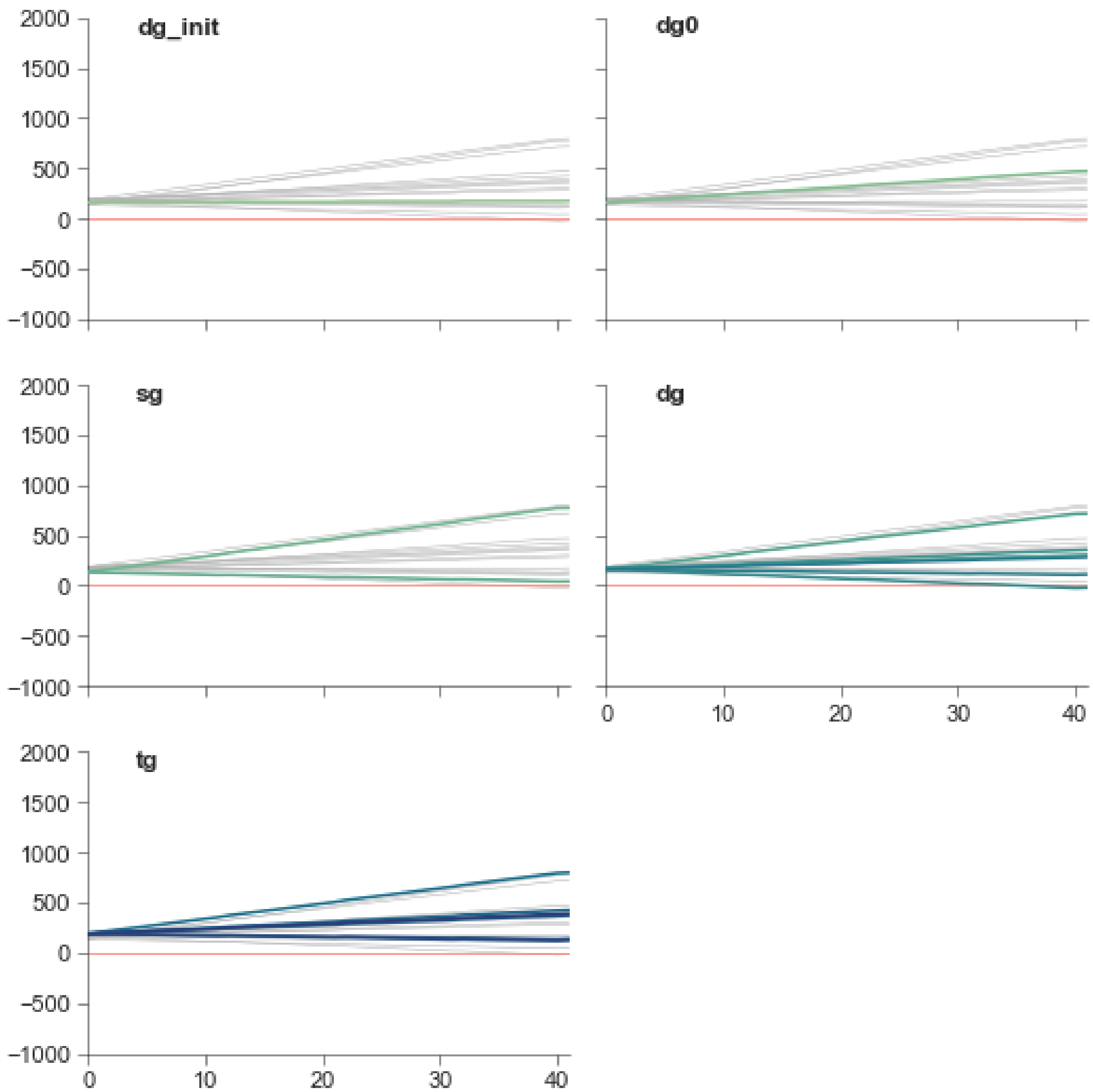
```
# Define the rank of the impact category (#):
impact_cat = 1
var = "IGU"
```

```
step = "step_1"

plot_multilca_40(step, impact_cat, var, -1000, 2000, 500)
```

```
step_1
climate change , climate change total
Unit is: kg CO2-Eq
```

[188]:

	impact at year = 41, points	year at net-zero
a_a_2126_dg_init	174.707163	na
a_b_2126_dg0	470.161355	na
a_c_2126_sg1	775.094163	na
a_d_2126_sg2	43.437485	na
a_e_2126_dg1	719.060213	na
a_f_2126_dg2	305.831126	na
a_g_2126_dg3	356.178657	na
a_h_2126_dg4	-18.826401	36
a_i_2126_dg5	111.617073	na
a_j_2126_dg6	283.923207	na
a_k_2126_tg1	797.190097	na
a_l_2126_tg2	425.133957	na
a_m_2126_tg3	370.971449	na
a_n_2126_tg4	123.647686	na
a_o_2126_tg5	138.323590	na
a_p_2126_tg6	390.402460	na



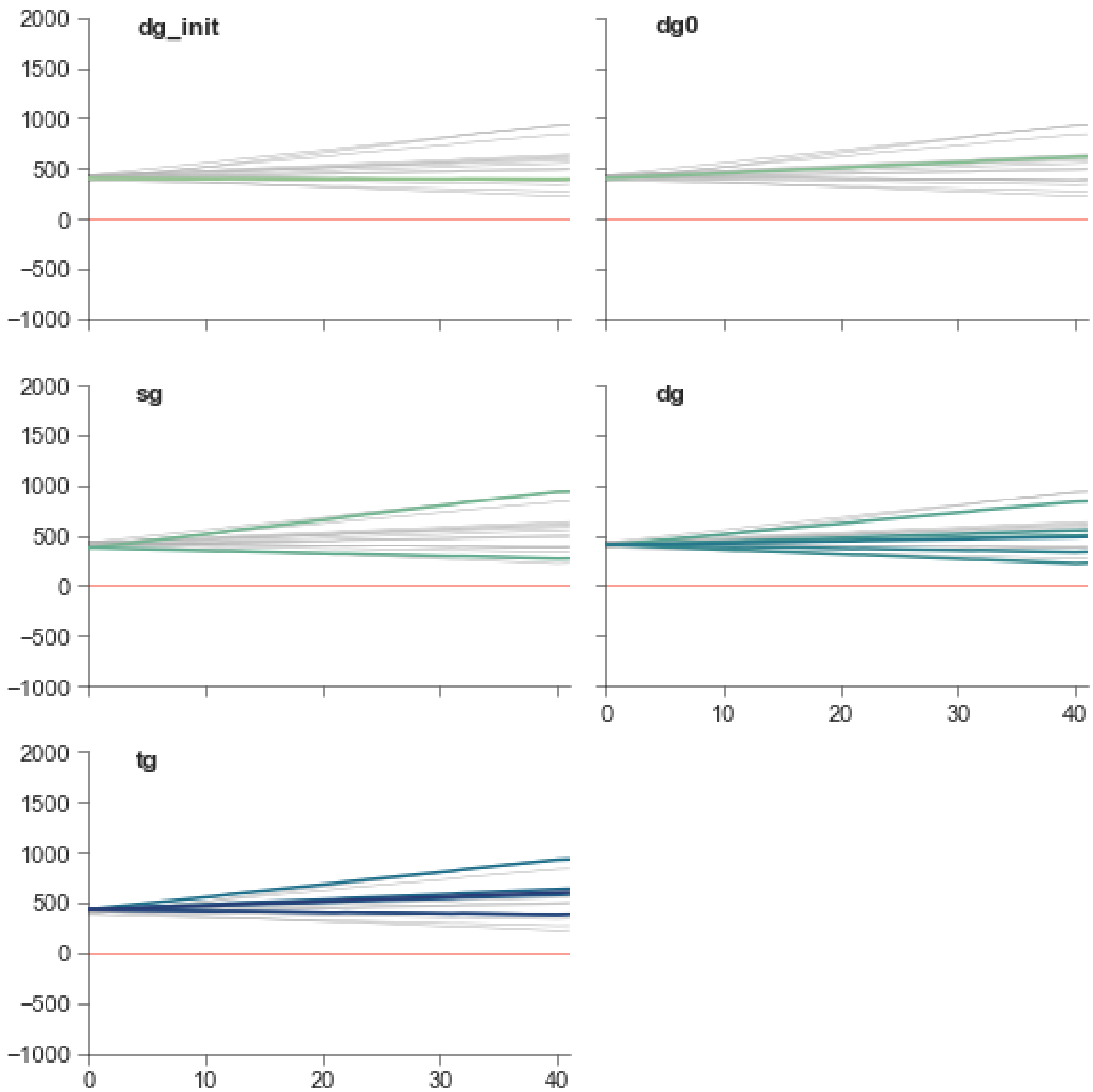
12.1.2 Step 2: With Interior Shading Devices

[189]:
```
# Define the rank of the impact category (#):
impact_cat = 1
var = "IGU"
step = "step_2"

plot_multilca_40(step, impact_cat, var, -1000, 2000, 500)
```

```
step_2
climate change , climate change total
Unit is: kg CO2-Eq
```

[189]:

	impact at year = 41, points	year at net-zero
b_a_2126_dg_init_int	391.550381	na
b_b_2126_dg0_int	616.217497	na
b_c_2126_sg1_int	936.925609	na
b_d_2126_sg2_int	268.748396	na
b_e_2126_dg1_int	838.327841	na
b_f_2126_dg2_int	501.500446	na
b_g_2126_dg3_int	552.672578	na
b_h_2126_dg4_int	223.546318	na
b_i_2126_dg5_int	336.443486	na
b_j_2126_dg6_int	487.885643	na
b_k_2126_tg1_int	932.520806	na
b_l_2126_tg2_int	635.148984	na
b_m_2126_tg3_int	581.636476	na
b_n_2126_tg4_int	379.152909	na
b_o_2126_tg5_int	379.114862	na
b_p_2126_tg6_int	601.892079	na



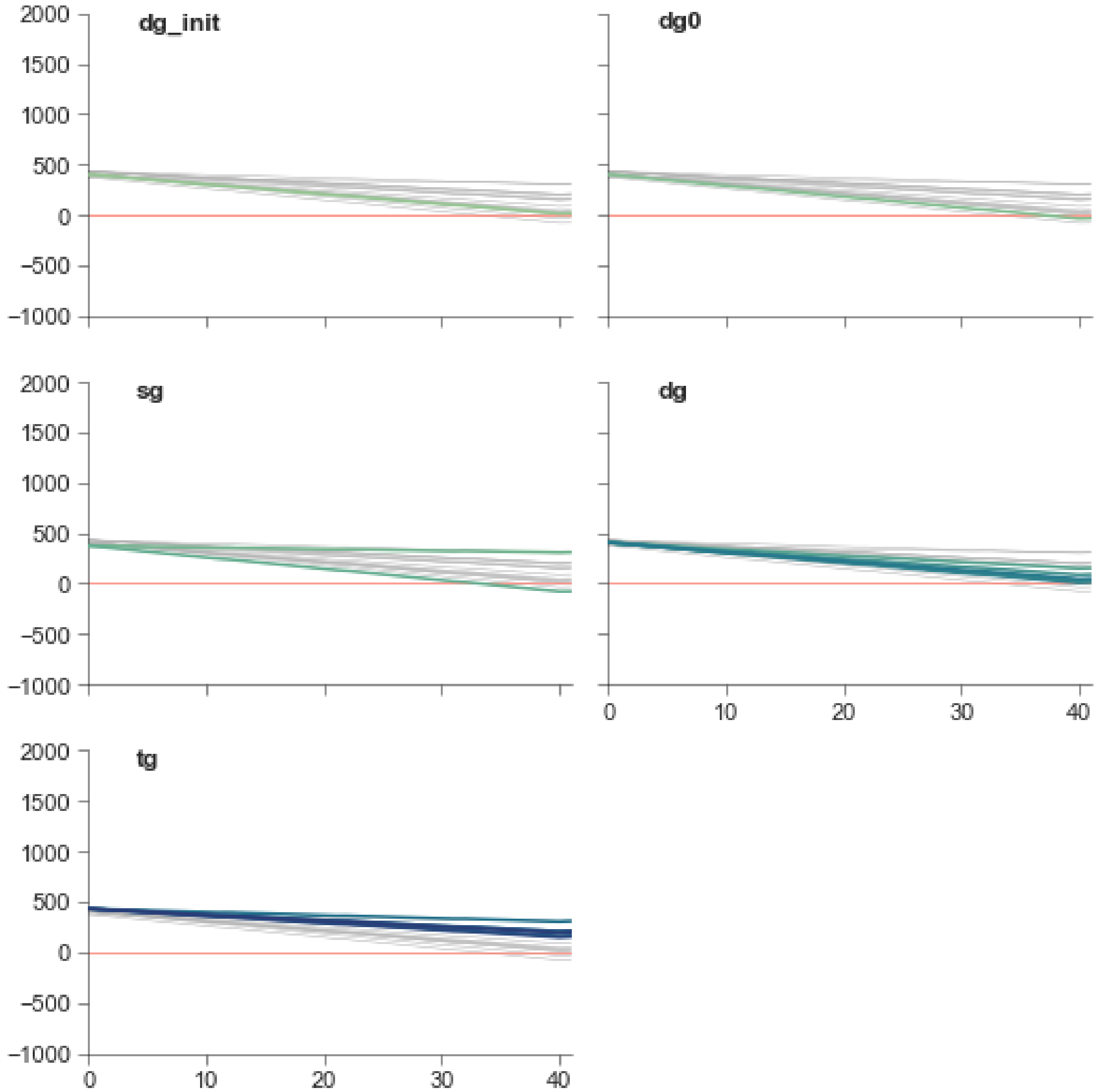
12.1.3 Step 3: With Exterior Shading Devices

[190]:
```
# Define the rank of the impact category (#):
impact_cat = 1
var = "IGU"
step = "step_3"

plot_multilca_40(step, impact_cat, var, -1000, 2000, 500)
```

```
step_3
climate change , climate change total
Unit is: kg CO2-Eq
```

[190]:

	impact at year = 41, points	year at net-zero
c_a_2126_dg_init_ext	14.811388	na
c_b_2126_dg0_ext	-32.991959	37
c_c_2126_sg1_ext	310.723091	na
c_d_2126_sg2_ext	-77.377148	34
c_e_2126_dg1_ext	155.109475	na
c_f_2126_dg2_ext	37.091148	na
c_g_2126_dg3_ext	91.193071	na
c_h_2126_dg4_ext	29.954722	na
c_i_2126_dg5_ext	9.795424	na
c_j_2126_dg6_ext	50.053384	na
c_k_2126_tg1_ext	308.713337	na
c_l_2126_tg2_ext	210.517698	na
c_m_2126_tg3_ext	168.430768	na
c_n_2126_tg4_ext	209.538461	na
c_o_2126_tg5_ext	151.565761	na
c_p_2126_tg6_ext	197.310010	na



12.1.4 Overall Impact

Comparative Analysis

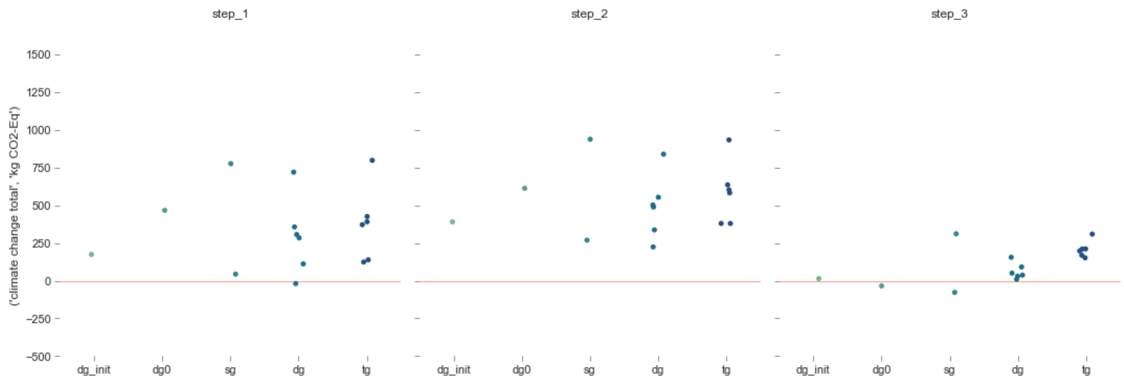
[191]:
```
df_ilcd_methods
```

[191]:

Category	#	Subcategory	Unit
climate change	1	climate change total	kg CO2-Eq
ecosystem quality	2	freshwater ecotoxicity	CTU
	3	freshwater and terrestrial acidification	mol H+-Eq
	4	freshwater eutrophication	kg P-Eq
	5	marine eutrophication	kg N-Eq

```
                 6          terrestrial eutrophication   mol N-Eq
human health     7           non-carcinogenic effects        CTUh
                 8               carcinogenic effects        CTUh
                 9                 ionising radiation  kg U235-Eq
                 10             ozone layer depletion  kg CFC-11.
                 11        photochemical ozone creation  kg NMVOC-.
                 12       respiratory effects, inorganics  disease i.
resources        13               minerals and metals    kg Sb-Eq
                 14                   dissipated water  m3 water-.
                 15                            fossils   megajoule
                 16                           land use      points
```

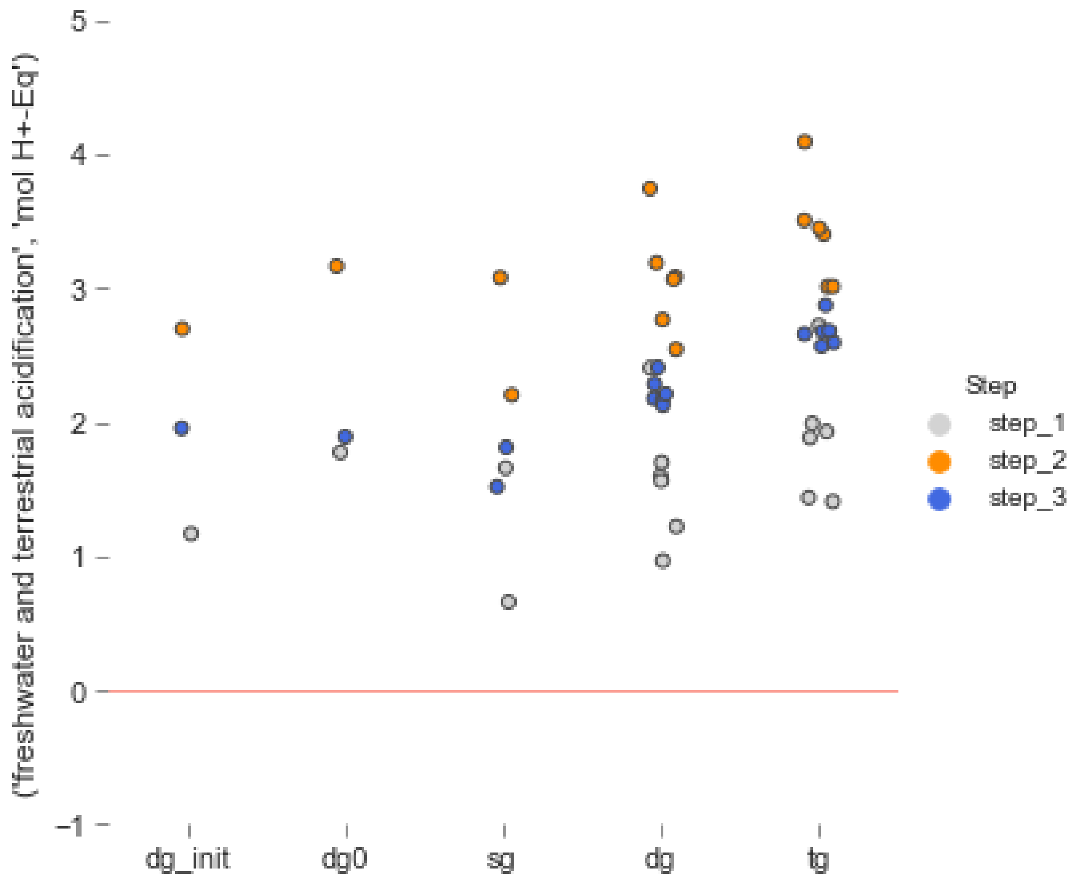
[192]:
```python
# Define the impact category:
n = 1

ic = df_ilcd_methods.xs(n, level=1)["Subcategory"][0]
i = df_ilcd_methods.index[df_ilcd_methods['Subcategory'] == ic][0][0]
ic_unit = df_ilcd_methods.xs(n, level=1)["Unit"][0]


# Keep the lca impact results at the end of life:
df_plot = df_lca_lifespan[['step_1', 'step_2', 'step_3']].xs(
    ic, axis=1, level=4, drop_level=False).loc[[41]]

# Transpose:
df_plot = df_plot.T.unstack(level=(4, 5))[41].reset_index()

# Category plot:
g = sns.catplot(data=df_plot, x="IGU",
                y=(ic, ic_unit),
                hue="IGU", col="Step",
                palette="crest", height=5, aspect=1
                )

for ax in g.axes.flat:
    # ax.yaxis.set_major_formatter(FuncFormatter(thousand_divide))
    style_ax(ax)
    ax.axhline(y=0, c='salmon', linestyle='-', linewidth=0.75)

(g.set_titles("{col_name}", fontsize=17, y=1.1)
 .set(ylim=(-500, 1500))
 .despine(left=True, bottom=True, offset=5)
 )

plt.show()
```



[193]:

```
# Define the impact category:
n = 3

ic = df_ilcd_methods.xs(n, level=1)["Subcategory"][0]
i = df_ilcd_methods.index[df_ilcd_methods['Subcategory'] == ic][0][0]
ic_unit = df_ilcd_methods.xs(n, level=1)["Unit"][0]


# Keep the lca impact results at the end of life:
df_plot = df_lca_lifespan[['step_1', 'step_2', 'step_3']].xs(
    ic, axis=1, level=4, drop_level=False).loc[[41]]

# Transpose:
df_plot = df_plot.T.unstack(level=(4, 5))[41].reset_index()

mycolors = sns.color_palette(['lightgrey', 'darkorange', 'royalblue'])

# Category plot:
g = sns.catplot(data=df_plot, x="IGU",
                y=(ic, ic_unit),
                hue="Step", linewidth=1,
                palette=mycolors, height=5, aspect=1
                )

for ax in g.axes.flat:
    # ax.yaxis.set_major_formatter(FuncFormatter(thousand_divide))
    style_ax(ax)
    ax.axhline(y=0, c='salmon', linestyle='-', linewidth=0.75)

(g.set_titles("", fontsize=17, y=1.1)
 .set(ylim=(-1, 5))
 .despine(left=True, bottom=True, offset=5)
 )
```

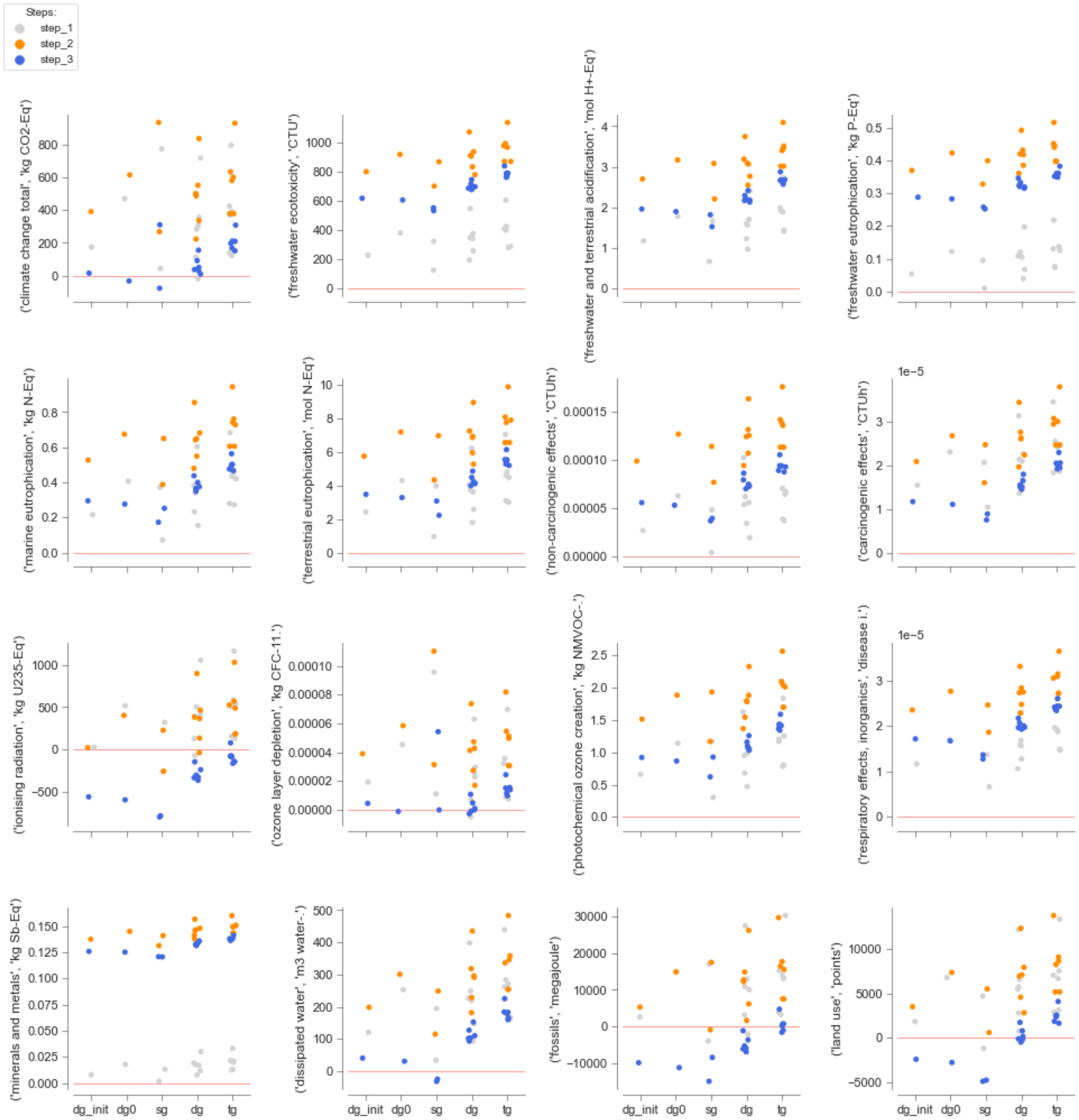
```
plt.show()
```



Overview of the full LCIA:

[194]:
```
fig, axes = plt.subplots(nrows=4, ncols=4,
                         sharex=True, sharey=False,
                         figsize=(16, 16))

n = 1

mycolors = sns.color_palette(['lightgrey', 'darkorange', 'royalblue'])

for row in range(4):
    for col in range(4):

        ax = axes[row][col]

        ic = df_ilcd_methods.xs(n, level=1)["Subcategory"][0]
```

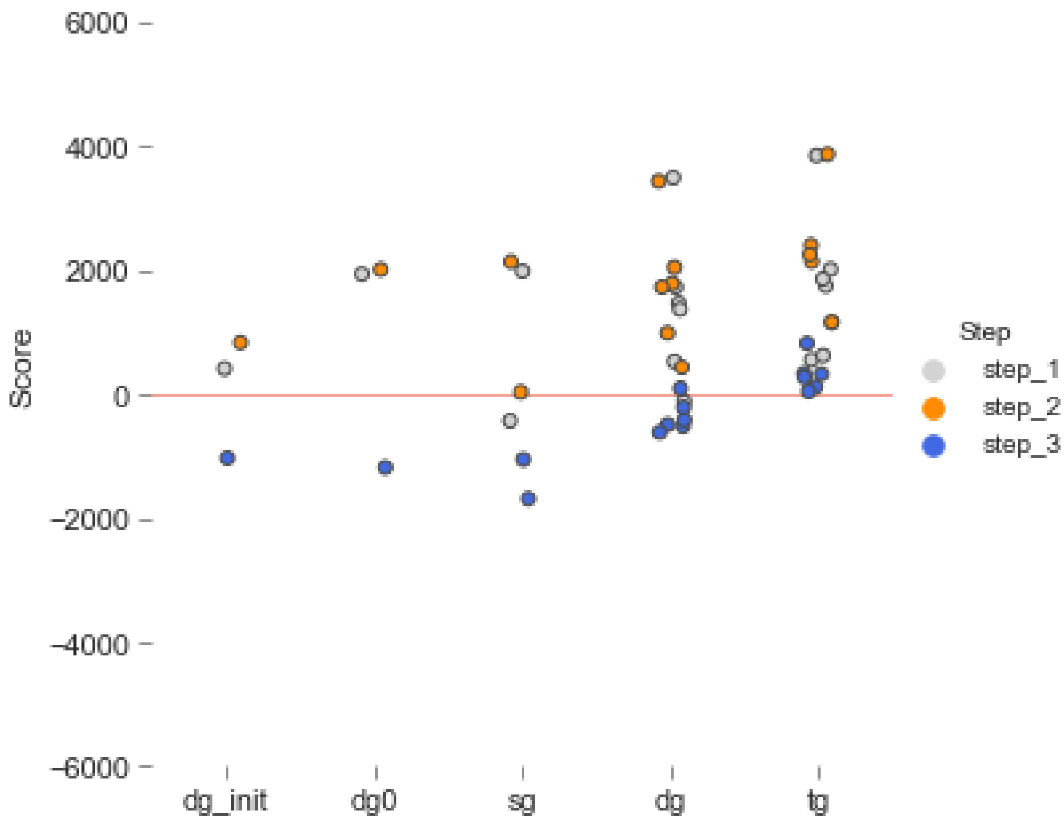
```
        i = df_ilcd_methods.index[df_ilcd_methods['Subcategory'] == ic][0][0]
        ic_unit = df_ilcd_methods.xs(n, level=1)["Unit"][0]

        # Keep the lca impact results at the end of life:
        df_plot = df_lca_lifespan[['step_1', 'step_2', 'step_3']].xs(
            ic, axis=1, level=4, drop_level=False).loc[[41]]

        # Transpose:
        df_plot = df_plot.T.unstack(level=(4, 5))[41].reset_index()

        # mycolors=sns.color_palette(['firebrick', 'lightcoral', 'royalblue'])

        # Category plot:
        sns.stripplot(data=df_plot, x="IGU",
                      y=(ic, ic_unit),
                      hue="Step",
                      palette=mycolors, ax=ax
                      )

        ax.axhline(y=0, c='salmon', linestyle='-', linewidth=0.75)

        ax.get_legend().remove()
        style_ax(ax)

        n += 1

fig.subplots_adjust(wspace=0.55, hspace=0.45)

# Add legend:
handles, labels = ax.get_legend_handles_labels()
fig.legend(handles, labels, loc='center', ncol=1,
           title='Steps:',
           bbox_to_anchor=(0.1, 0.94))

fig.suptitle('', y=0.95)

sns.despine(offset=5)

if export:
    # Save image:
    fig.savefig(os.path.join(path_img, 'FullLCIA_Step1-3.png'),
                dpi=600, bbox_inches='tight')
    fig.savefig(os.path.join(path_img, 'FullLCIA_Step1-3.pdf'),
                bbox_inches='tight')

plt.show()
```



Analysis of the weighted impact:

[195]:
```python
# Keep the lca impact results at the end of life:
df_plot = df_weighted[['step_1', 'step_2', 'step_3']]

# Transpose:
df_plot = df_plot.T.reset_index()

mycolors = sns.color_palette(['lightgrey', 'darkorange', 'royalblue'])

# Category plot:
g = sns.catplot(data=df_plot, x="IGU",
```

```
                y="Score",
                hue="Step", linewidth=1,
                palette=mycolors, height=5, aspect=1
                )

for ax in g.axes.flat:
    # ax.yaxis.set_major_formatter(FuncFormatter(thousand_divide))
    style_ax(ax)
    ax.axhline(y=0, c='salmon', linestyle='-', linewidth=0.75)

(g.set_titles("", fontsize=17, y=1.1)
 .set(ylim=(-6000, 7000))
 .despine(left=True, bottom=True, offset=5)
 )

if export:
    # Save image:
    g.savefig(os.path.join(path_img, 'WeightedLCIA_Step1-3.png'),
              dpi=600, bbox_inches='tight')
    g.savefig(os.path.join(path_img, 'WeightedLCIA_Step1-3.pdf'),
              bbox_inches='tight')

plt.show()
```



12.2 Steps 4-7: Glazing Performance and HVAC Systems, A Sensitivity Analysis

12.2.1 Step 4: Efficient VAV HVAC System, w/o Shading Devices

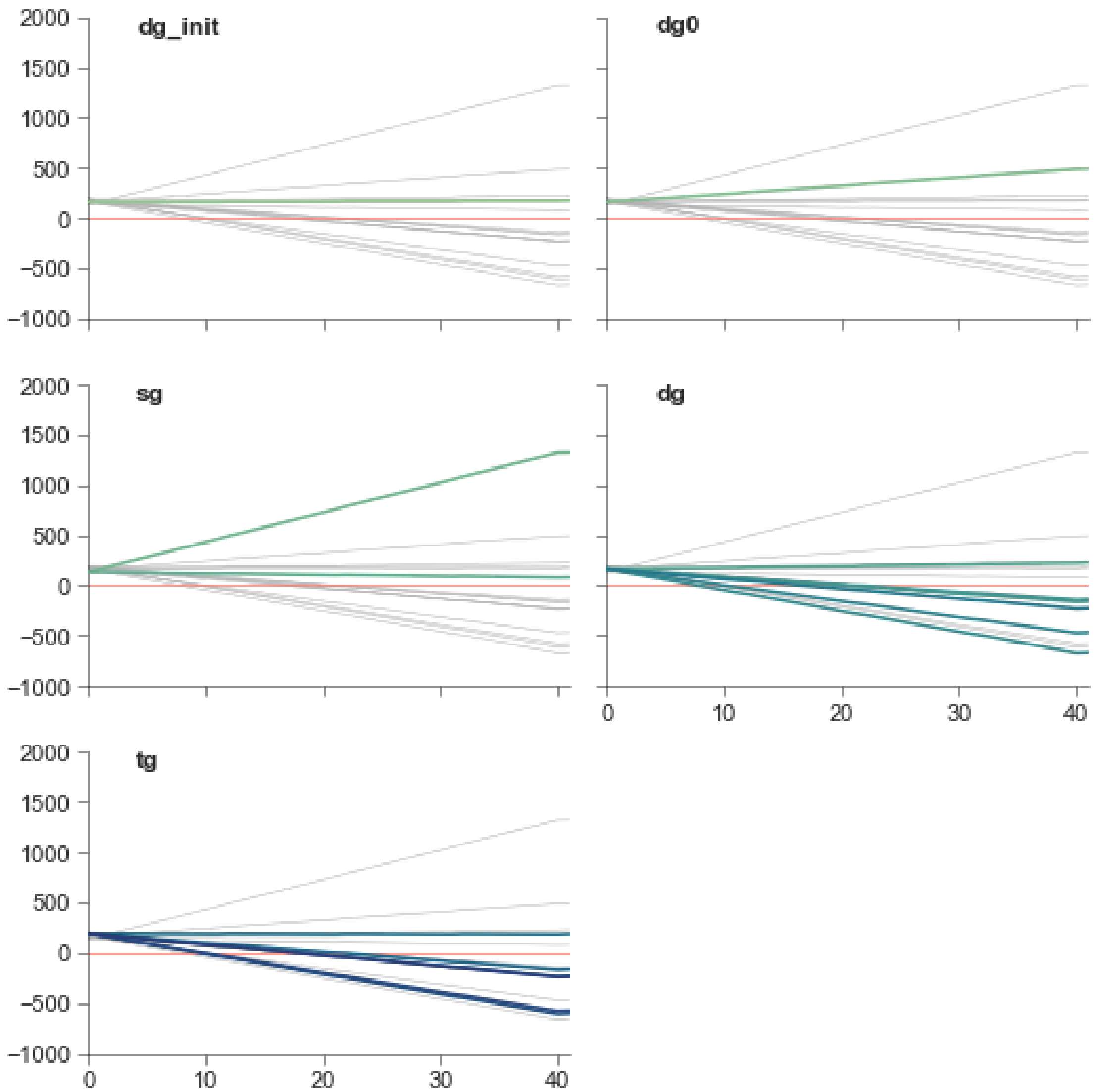
[196]:
```
# Define the rank of the impact category (#):
impact_cat = 1
var = "IGU"
step = "step_4"

plot_multilca_40(step, impact_cat, var, -1000, 2000, 500)
```

```
step_4
climate change , climate change total
Unit is: kg CO2-Eq
```

[196]:

	impact at year = 41, points	year at net-zero
d_a_2126_dg_init_vav	174.707163	na
d_b_2126_dg0_vav	488.920418	na
d_c_2126_sg1_vav	1321.051119	na

d_d_2126_sg2_vav	82.030242	na
d_e_2126_dg1_vav	226.450733	na
d_f_2126_dg2_vav	-157.384412	20
d_g_2126_dg3_vav	-138.397083	22
d_h_2126_dg4_vav	-665.857675	8
d_i_2126_dg5_vav	-470.927942	11
d_j_2126_dg6_vav	-226.233505	17
d_k_2126_tg1_vav	184.908873	na
d_l_2126_tg2_vav	-164.253647	22
d_m_2126_tg3_vav	-231.039398	18
d_n_2126_tg4_vav	-608.100456	10
d_o_2126_tg5_vav	-579.143174	10
d_p_2126_tg6_vav	-236.140428	18



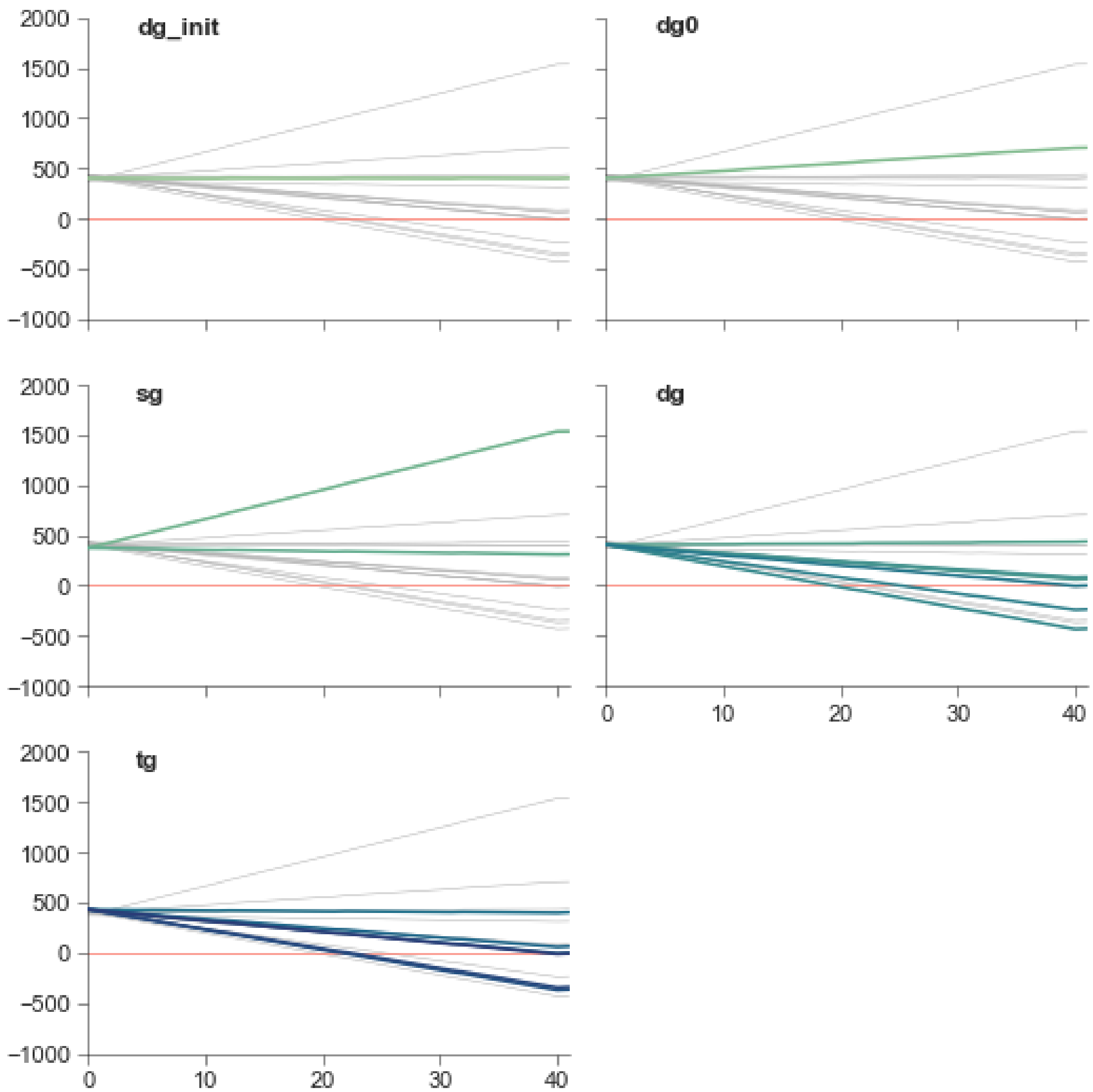
12.2.2 Step 5: Efficient VAV HVAC System, with Interior Shading Devices

[197]:
```python
# Define the rank of the impact category (#):
impact_cat = 1
var = "IGU"
step = "step_5"

plot_multilca_40(step, impact_cat, var, -1000, 2000, 500)
```

```
step_5
climate change , climate change total
Unit is: kg CO2-Eq
```

[197]:

	impact at year = 41, points	year at net-zero
e_a_2126_dg_init_vav_int	402.515952	na
e_b_2126_dg0_vav_int	706.181456	na
e_c_2126_sg1_vav_int	1537.366613	na
e_d_2126_sg2_vav_int	312.705846	na
e_e_2126_dg1_vav_int	440.337777	na
e_f_2126_dg2_vav_int	66.864249	na
e_g_2126_dg3_vav_int	85.729935	na
e_h_2126_dg4_vav_int	-429.017570	20
e_i_2126_dg5_vav_int	-237.636621	26
e_j_2126_dg6_vav_int	0.944366	na
e_k_2126_tg1_vav_int	402.563400	na
e_l_2126_tg2_vav_int	62.554237	na
e_m_2126_tg3_vav_int	-1.235628	41
e_n_2126_tg4_vav_int	-366.875088	22
e_o_2126_tg5_vav_int	-341.395766	23
e_p_2126_tg6_vav_int	-9.027442	39



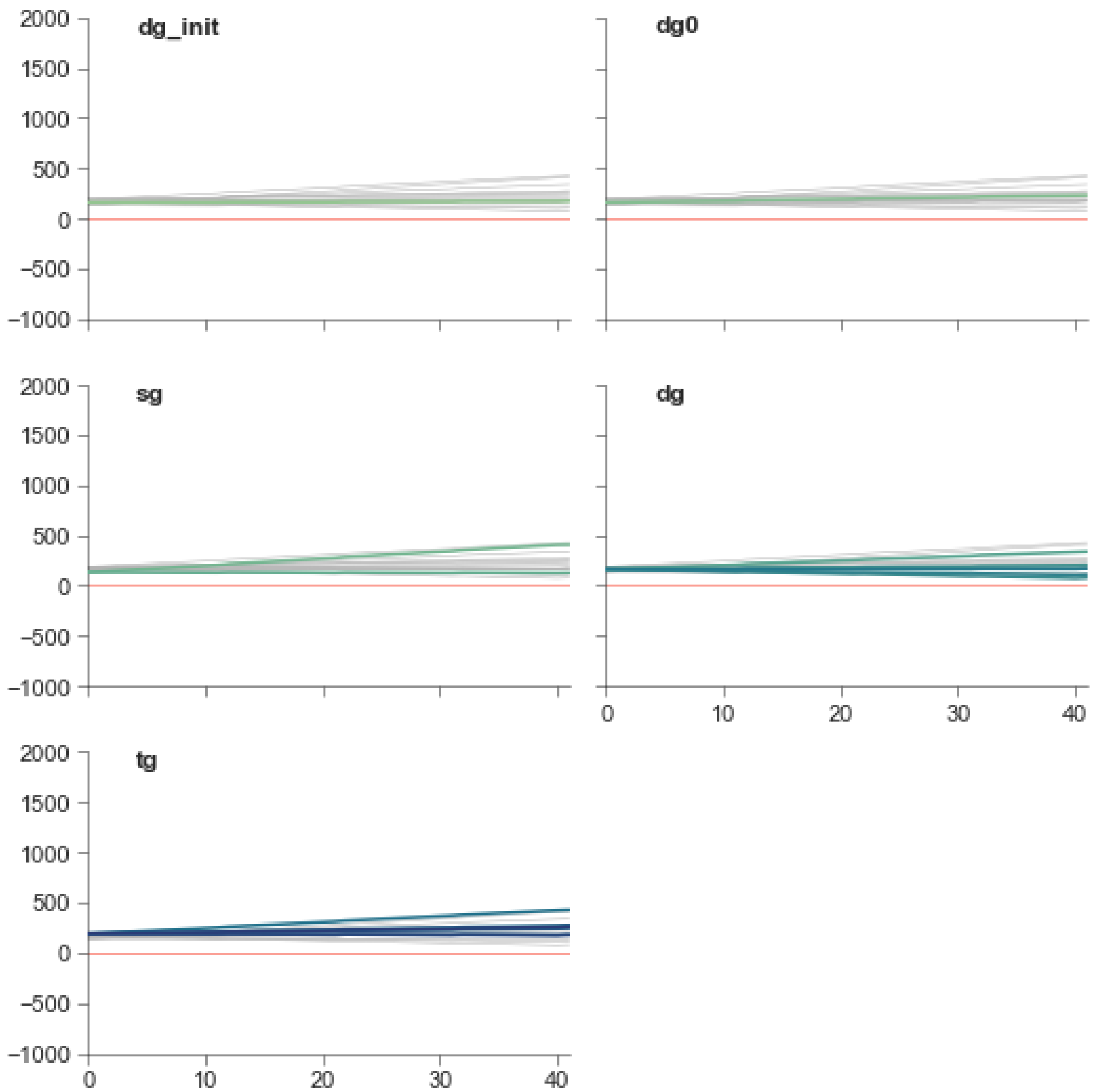
12.2.3 Step 6: Efficient VRF HVAC System, w/o Shading Devices

[198]:
```
# Define the rank of the impact category (#):
impact_cat = 1
var = "IGU"
step = "step_6"

plot_multilca_40(step, impact_cat, var, -1000, 2000, 500)
```

```
step_6
climate change , climate change total
Unit is: kg CO2-Eq
```

[198]:

	impact at year = 41, points	year at net-zero
f_a_2126_dg_init_vrf	174.707163	na
f_b_2126_dg0_vrf	229.877198	na
f_c_2126_sg1_vrf	407.838957	na
f_d_2126_sg2_vrf	127.916735	na
f_e_2126_dg1_vrf	338.319916	na
f_f_2126_dg2_vrf	174.429604	na
f_g_2126_dg3_vrf	207.155290	na
f_h_2126_dg4_vrf	77.843242	na
f_i_2126_dg5_vrf	110.229452	na
f_j_2126_dg6_vrf	172.924702	na
f_k_2126_tg1_vrf	427.078553	na
f_l_2126_tg2_vrf	274.468603	na
f_m_2126_tg3_vrf	246.359818	na
f_n_2126_tg4_vrf	181.882413	na
f_o_2126_tg5_vrf	172.949289	na
f_p_2126_tg6_vrf	261.232989	na



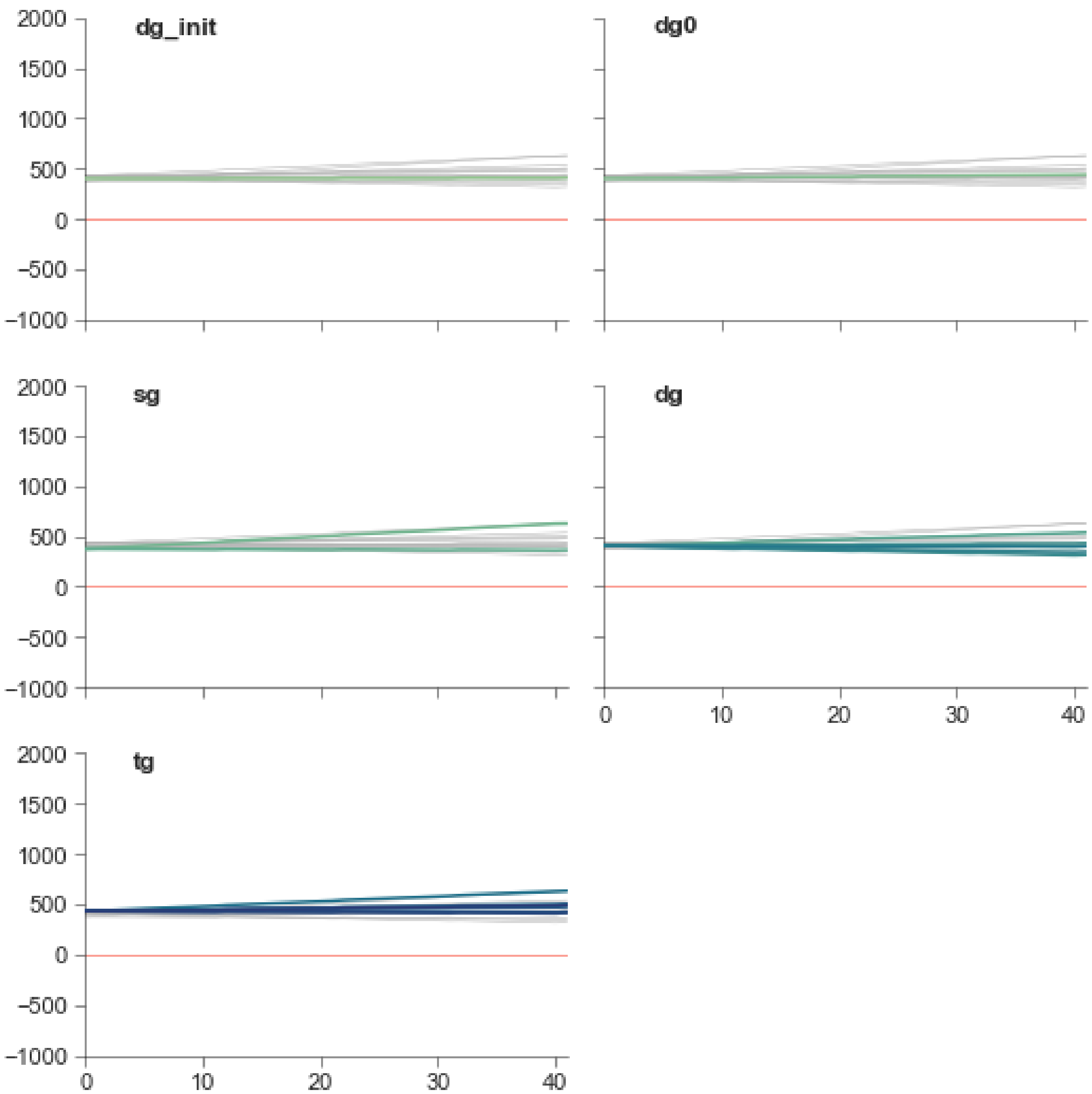
12.2.4 Step 7: Efficient VRF HVAC System, with Interior Shading Devices

[199]:
```
# Define the rank of the impact category (#):
impact_cat = 1
var = "IGU"
step = "step_7"

plot_multilca_40(step, impact_cat, var, -1000, 2000, 500)
```

```
step_7
climate change , climate change total
Unit is: kg CO2-Eq
```

[199]:

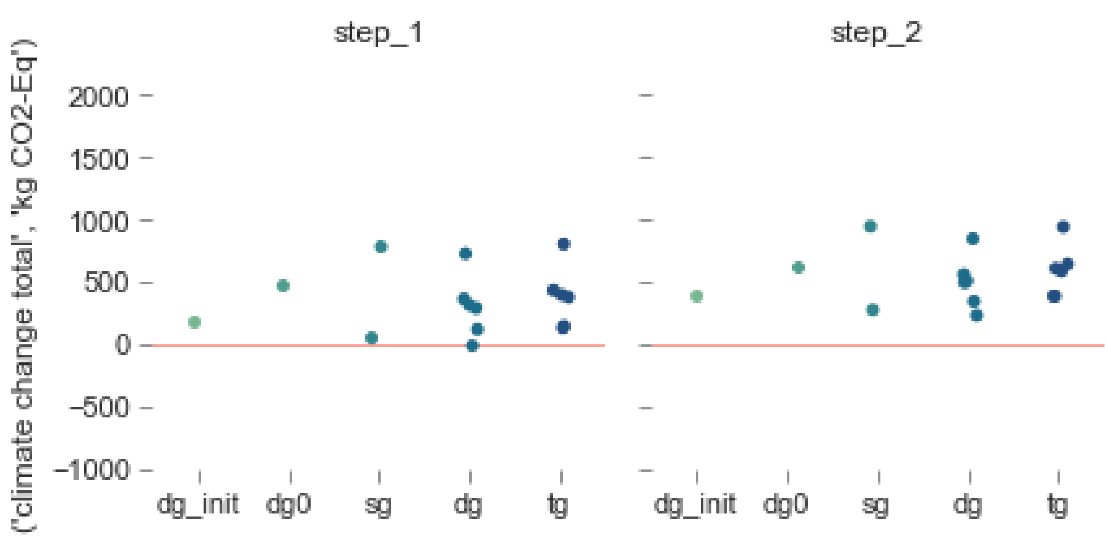
	impact at year = 41, points	year at net-zero
g_a_2126_dg_init_vrf_int	408.282551	na
g_b_2126_dg0_vrf_int	440.833649	na
g_c_2126_sg1_vrf_int	627.505451	na
g_d_2126_sg2_vrf_int	363.878169	na
g_e_2126_dg1_vrf_int	537.772915	na
g_f_2126_dg2_vrf_int	400.943849	na
g_g_2126_dg3_vrf_int	433.504646	na
g_h_2126_dg4_vrf_int	318.440977	na
g_i_2126_dg5_vrf_int	345.725315	na
g_j_2126_dg6_vrf_int	401.485513	na
g_k_2126_tg1_vrf_int	631.061152	na
g_l_2126_tg2_vrf_int	504.329131	na
g_m_2126_tg3_vrf_int	476.588924	na
g_n_2126_tg4_vrf_int	425.632444	na
g_o_2126_tg5_vrf_int	412.945663	na
g_p_2126_tg6_vrf_int	491.694877	na

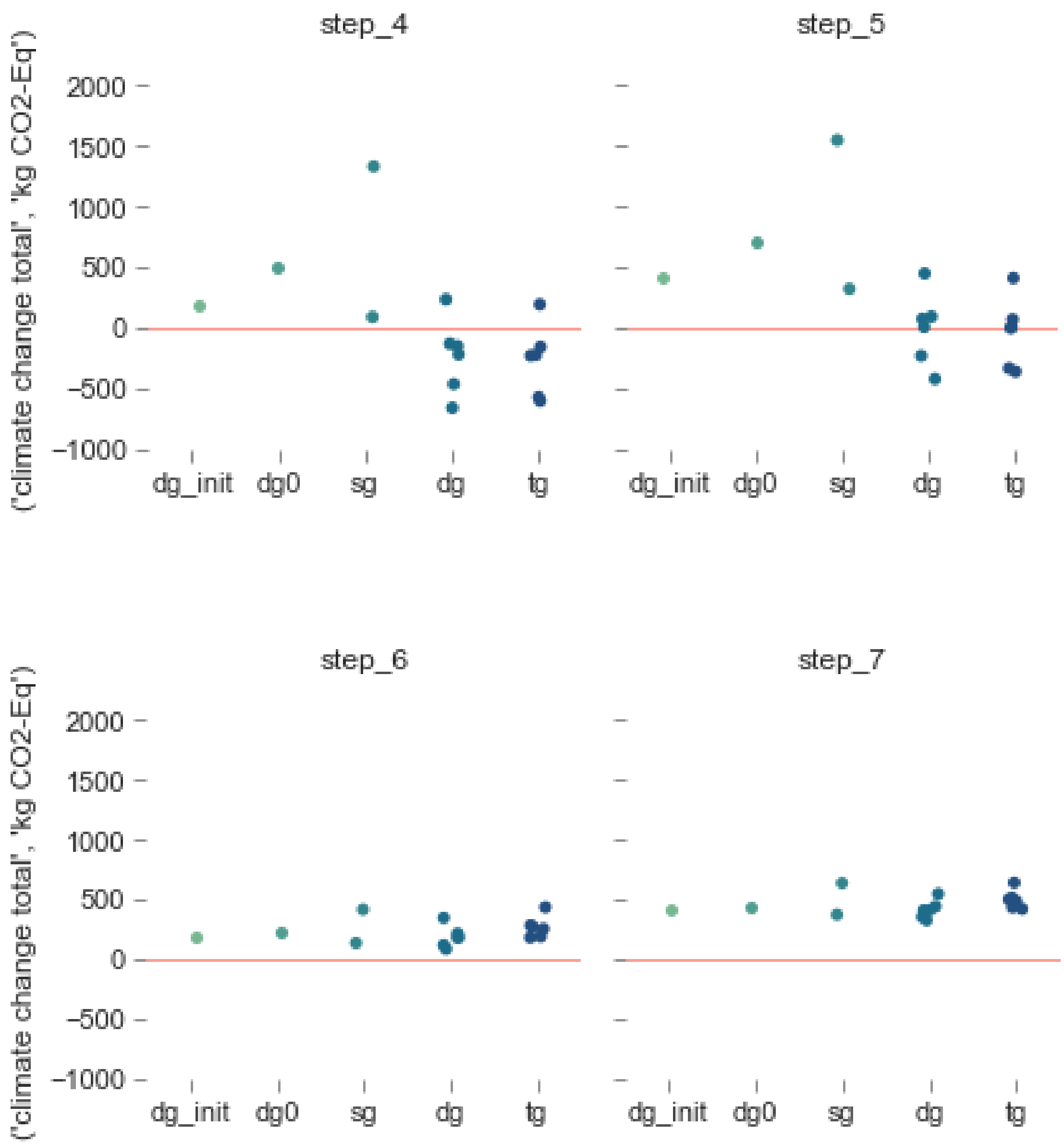


12.2.5 Overall Impact

[200]:
```
# Define the impact category:
n = 1

ic = df_ilcd_methods.xs(n, level=1)["Subcategory"][0]
i = df_ilcd_methods.index[df_ilcd_methods['Subcategory'] == ic][0][0]
ic_unit = df_ilcd_methods.xs(n, level=1)["Unit"][0]

for i, j in [('step_1', 'step_2'),
             ('step_4', 'step_5'),
             ('step_6', 'step_7')]:
```

```python
# Keep the lca impact results at the end of life:
df_plot = df_lca_lifespan[[i, j]].xs(
    ic, axis=1, level=4, drop_level=False).loc[[41]]

# Transpose:
df_plot = df_plot.T.unstack(level=(4, 5))[41].reset_index()

# Category plot:
g = sns.catplot(data=df_plot, x="IGU",
                y=(ic, ic_unit),
                hue="IGU", col="Step",
                palette="crest", height=3, aspect=1
                )

for ax in g.axes.flat:
    # ax.yaxis.set_major_formatter(FuncFormatter(thousand_divide))
    style_ax(ax)
    ax.axhline(y=0, c='salmon', linestyle='-', linewidth=0.75)

(g.set_titles("{col_name}", fontsize=17, y=1.1)
 .set(ylim=(-1000, 2000))
 .despine(left=True, bottom=True, offset=5)
 )

plt.show()
```



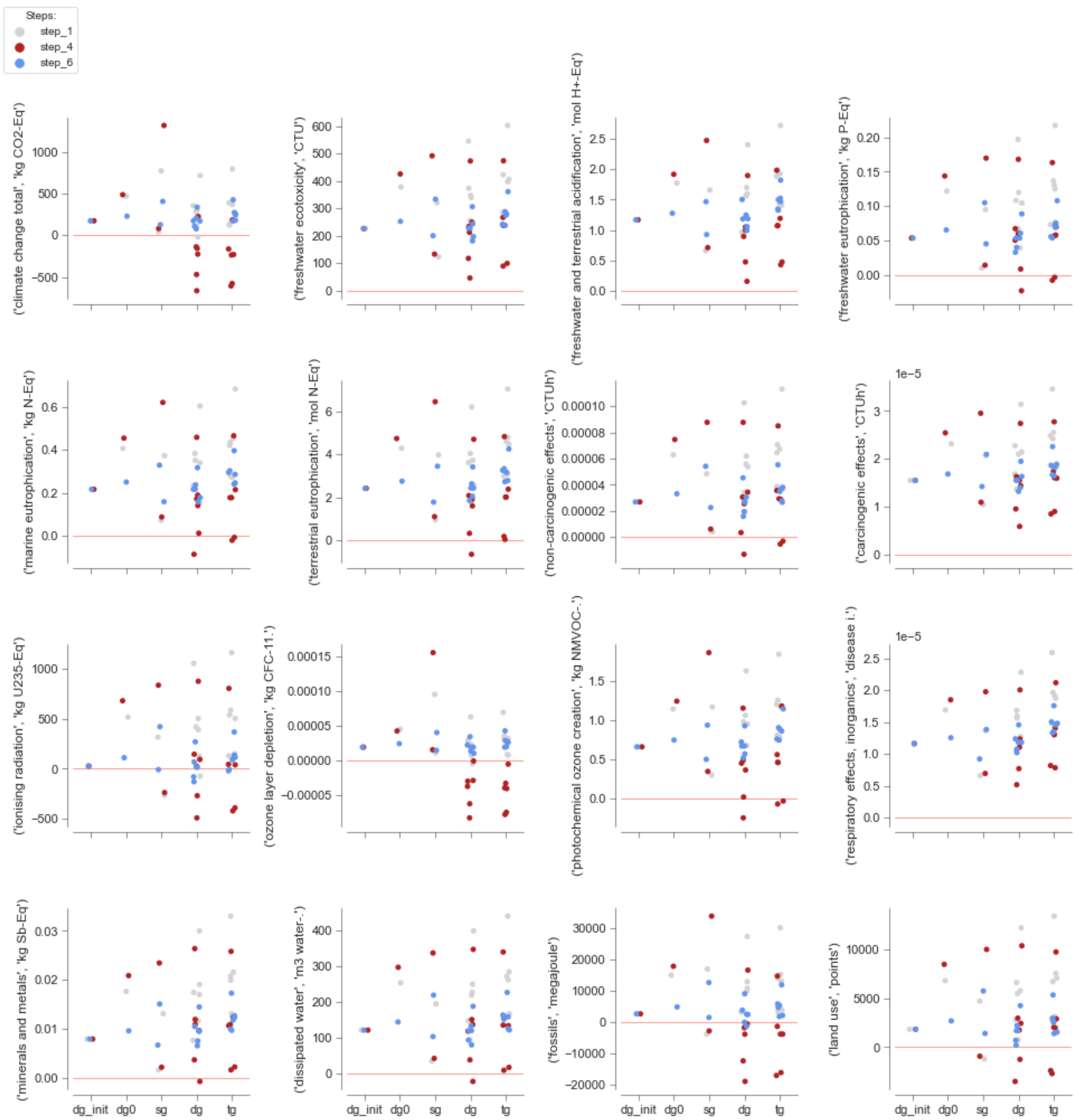


Overview of the full LCIA:

[201]:
```
fig, axes = plt.subplots(nrows=4, ncols=4,
                         sharex=True, sharey=False,
                         figsize=(16, 16))

n = 1

for row in range(4):
    for col in range(4):
```

```
        ax = axes[row][col]

        ic = df_ilcd_methods.xs(n, level=1)["Subcategory"][0]
        i = df_ilcd_methods.index[df_ilcd_methods['Subcategory'] == ic][0][0]
        ic_unit = df_ilcd_methods.xs(n, level=1)["Unit"][0]

        # Keep the lca impact results at the end of life:
        df_plot = df_lca_lifespan[['step_1', 'step_4', 'step_6']].xs(
            ic, axis=1, level=4, drop_level=False).loc[[41]]

        # Transpose:
        df_plot = df_plot.T.unstack(level=(4, 5))[41].reset_index()

        mycolors = sns.color_palette(
            ['lightgrey', 'firebrick', 'cornflowerblue'])

        # Category plot:
        sns.stripplot(data=df_plot, x="IGU",
                      y=(ic, ic_unit),
                      hue="Step",
                      palette=mycolors, ax=ax
                      )

        ax.axhline(y=0, c='salmon', linestyle='-', linewidth=0.75)

        ax.get_legend().remove()
        style_ax(ax)

        n += 1

fig.subplots_adjust(wspace=0.55, hspace=0.45)

# Add legend:
handles, labels = ax.get_legend_handles_labels()
fig.legend(handles, labels, loc='center', ncol=1,
           title='Steps:',
           bbox_to_anchor=(0.1, 0.94))

fig.suptitle('', y=0.95)

sns.despine(offset=5)

if export:
    # Save image:
    fig.savefig(os.path.join(path_img, 'FullLCIA_Step4-7.png'),
                dpi=600, bbox_inches='tight')
    fig.savefig(os.path.join(path_img, 'FullLCIA_Step4-7.pdf'),
```

```
                 bbox_inches='tight')

plt.show()
```

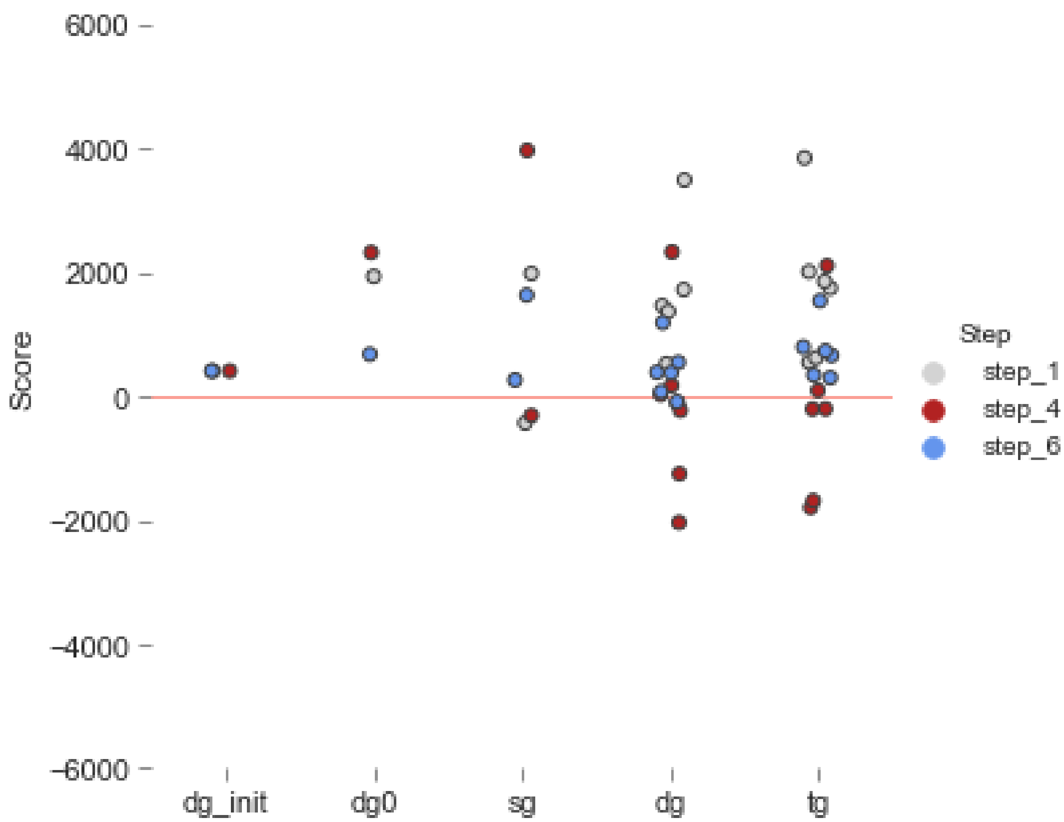


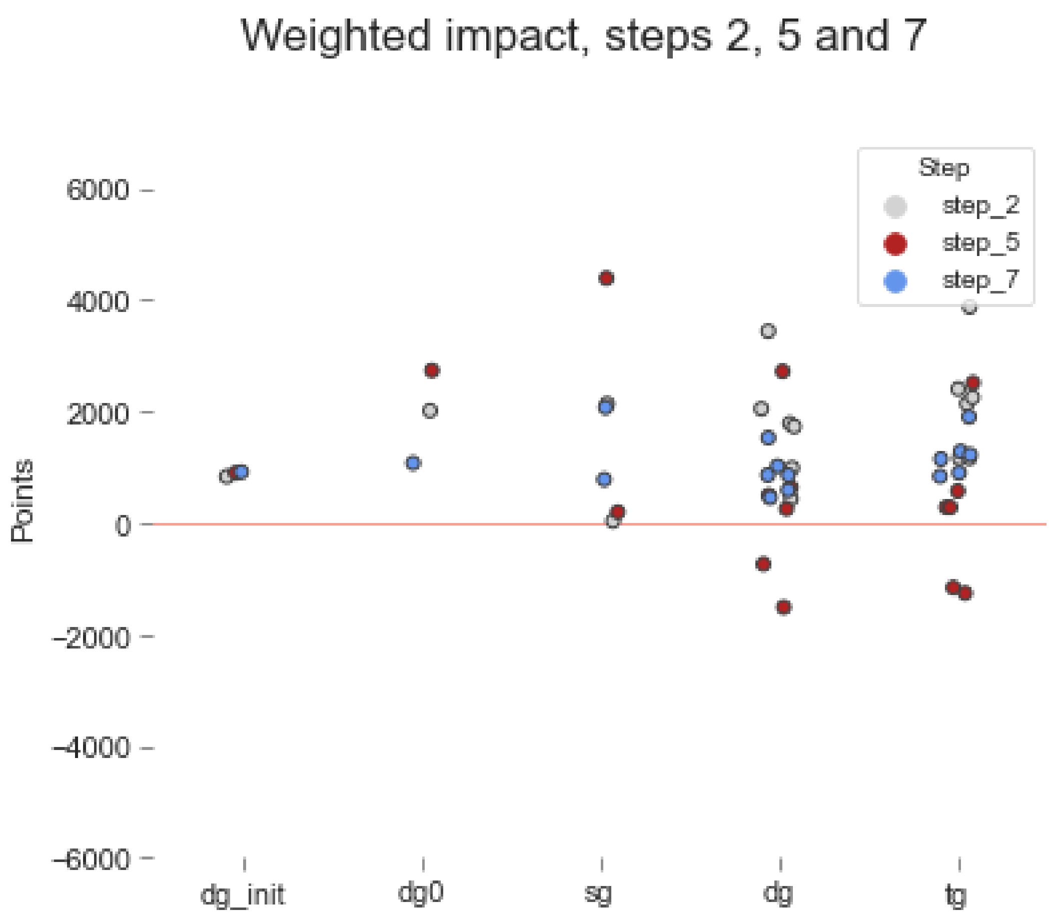
Analysis of the weighted impact:

[202]:
```
# Keep the lca impact results at the end of life:
df_plot = df_weighted[['step_1', 'step_4', 'step_6']]

# Transpose:
df_plot = df_plot.T.reset_index()
```

```python
mycolors = sns.color_palette(['lightgrey', 'firebrick', 'cornflowerblue'])

# Category plot:
g = sns.catplot(data=df_plot, x="IGU",
                y="Score",
                hue="Step", jitter=0.1, linewidth=1,
                palette=mycolors, height=5, aspect=1
                )

for ax in g.axes.flat:
    # ax.yaxis.set_major_formatter(FuncFormatter(thousand_divide))
    style_ax(ax)
    ax.axhline(y=0, c='salmon', linestyle='-', linewidth=0.75)

(g.set_titles("{col_name}", fontsize=17, y=1.1)
 .set(ylim=(-6000, 7000))
 .despine(left=True, bottom=True, offset=5)
 )

if export:
    # Save image:
    g.savefig(os.path.join(path_img, 'WeightedLCIA_Step4-7.png'),
              dpi=600, bbox_inches='tight')
    g.savefig(os.path.join(path_img, 'WeightedLCIA_Step4-7.pdf'),
              bbox_inches='tight')

plt.show()

print("\n\n")

fig, ax = plt.subplots(figsize=(6, 5))

# A second graph:
df_plot = df_weighted[['step_2', 'step_5', 'step_7']]

# Transpose:
df_plot = df_plot.T.reset_index()

mycolors = sns.color_palette(['lightgrey', 'firebrick', 'cornflowerblue'])

# Category plot:
sns.stripplot(data=df_plot, x="IGU",
              y="Score",
              hue="Step", jitter=0.1, linewidth=1,
              palette=mycolors, dodge=False, ax=ax
              )
```

```
style_ax(ax)
ax.axhline(y=0, c='salmon', linestyle='-', linewidth=0.75)

fig.suptitle("Weighted impact, steps 2, 5 and 7",
             fontsize=17, y=1)

sns.despine(left=True, bottom=True, offset=5)

ax.set(xlabel="", ylabel="Points")
ax.set_ylim(ymin=-6000, ymax=7000)

plt.show()
```

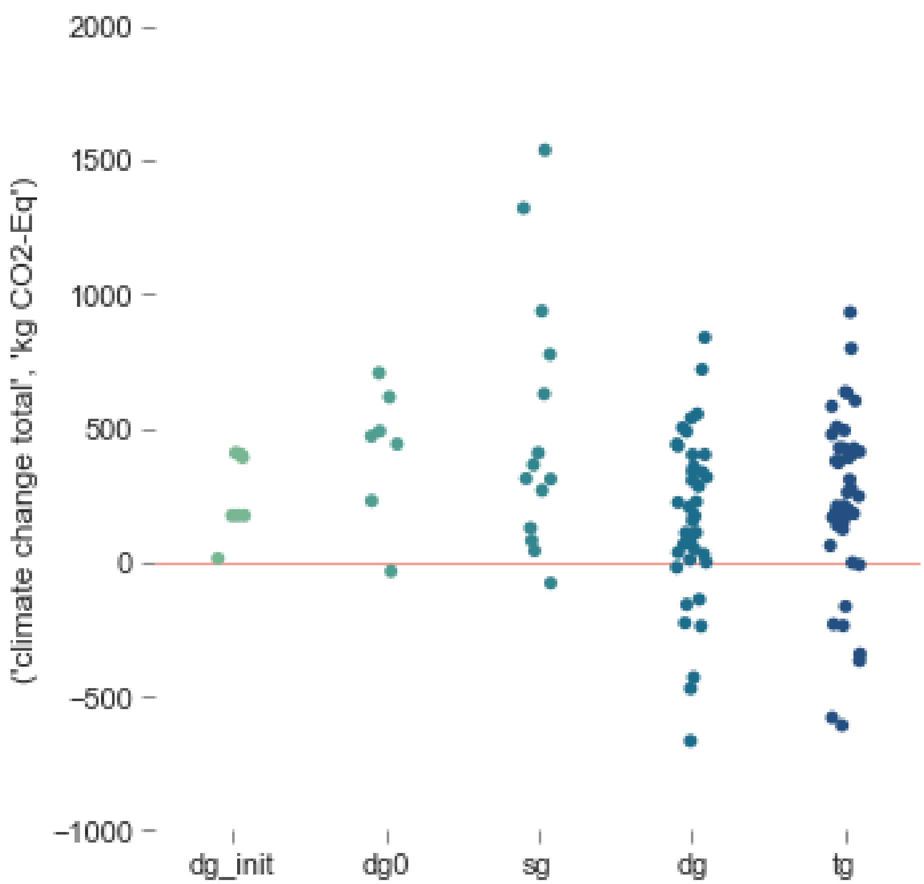




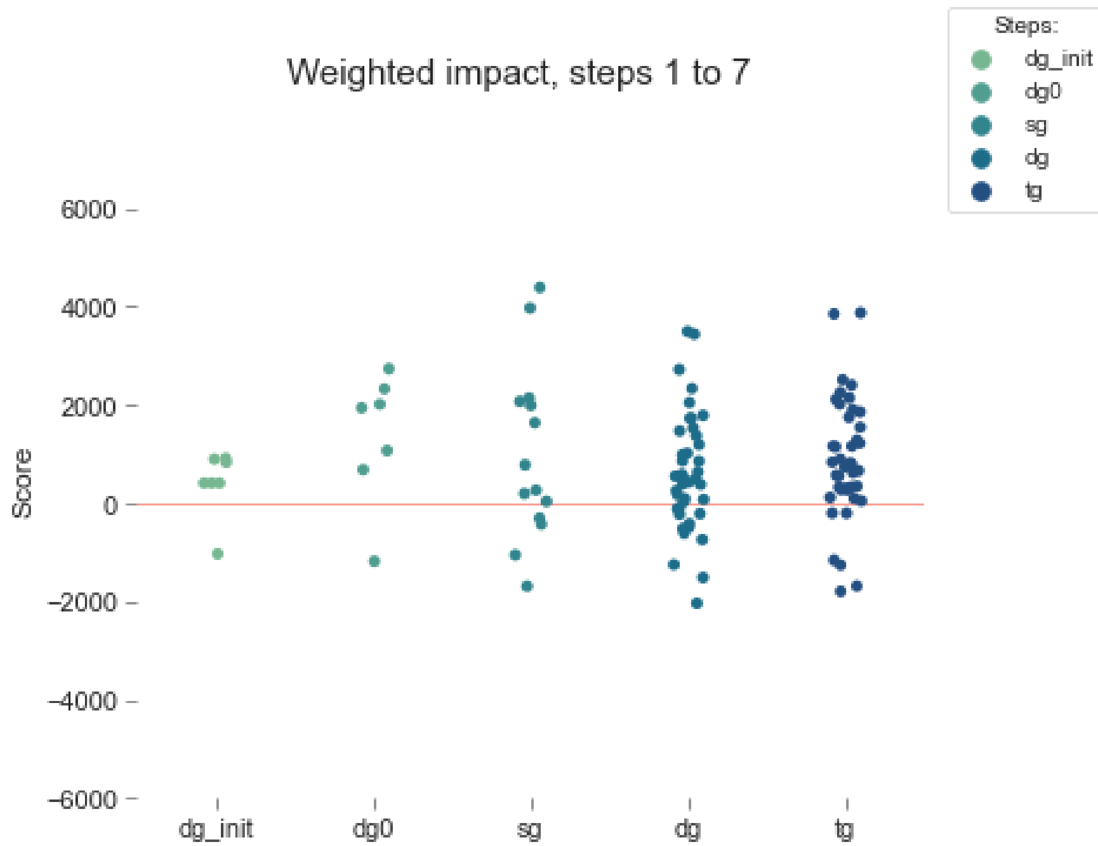
12.3 Steps 1 to 7: A Comparative Analysis

[203]:
```
# Define the impact category:
n = 1

ic = df_ilcd_methods.xs(n, level=1)["Subcategory"][0]
i = df_ilcd_methods.index[df_ilcd_methods['Subcategory'] == ic][0][0]
ic_unit = df_ilcd_methods.xs(n, level=1)["Unit"][0]

ls_steps = ['step_1', 'step_2', 'step_3',
            'step_4', 'step_5',
            'step_6', 'step_7']

# Keep the lca impact results at the end of life:
df_plot = df_lca_lifespan[ls_steps].xs(
    ic, axis=1, level=4, drop_level=False).loc[[41]]

# Transpose:
```

```python
df_plot = df_plot.T.unstack(level=(4, 5))[41].reset_index()

# Category plot:
g = sns.catplot(data=df_plot, x="IGU",
                y=(ic, ic_unit),
                hue="IGU",
                # kind="box",
                palette="crest", height=5, aspect=1
                )

for ax in g.axes.flat:
    # ax.yaxis.set_major_formatter(FuncFormatter(thousand_divide))
    style_ax(ax)
    ax.axhline(y=0, c='salmon', linestyle='-', linewidth=0.75)

(g.set_titles("{col_name}", fontsize=17, y=1.1)
 .set(ylim=(-1000, 2000))
 .despine(left=True, bottom=True, offset=5)
 )

plt.show()
```



[204]:
```python
# Keep the lca impact results at the end of life:
df_plot = df_weighted[['step_1', 'step_2', 'step_3',
                       'step_4', 'step_5',
                       'step_6', 'step_7']]

# Transpose:
df_plot = df_plot.T.reset_index()

fig, ax = plt.subplots(figsize=(6, 5))

# Category plot:
sns.stripplot(data=df_plot, x="IGU",
              y="Score",
                hue="IGU",
              palette="crest", ax=ax
              )

style_ax(ax)
ax.get_legend().remove()
ax.axhline(y=0, c='salmon', linestyle='-', linewidth=0.75)

fig.suptitle("Weighted impact, steps 1 to 7",
             fontsize=15, y=1)

sns.despine(left=True, bottom=True, offset=5)

ax.set(xlabel="", ylabel="Score")
ax.set_ylim(ymin=-6000, ymax=7000)

# Add legend:
handles, labels = ax.get_legend_handles_labels()
fig.legend(handles, labels, loc='center', ncol=1,
           title='Steps:',
           bbox_to_anchor=(1, 0.94))

plt.show()

print("\n\n")
```



12.4 Steps 8: Reduction of the Window-to-Wall Ratio

12.4.1 75% of the Initial WtW Ratio

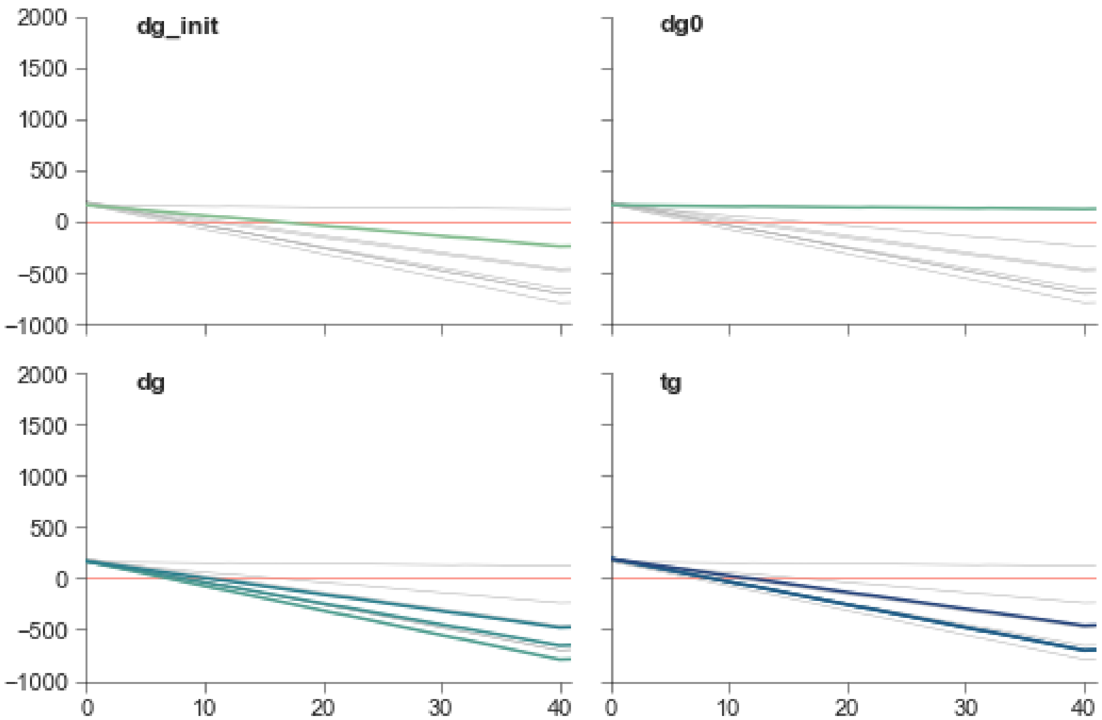
[205]:
```
# Define the rank of the impact category (#):
impact_cat = 1
var = "IGU"
step = "step_8"

plot_multilca_40(step, impact_cat, var, -1000, 2000, 500)
```

```
step_8
climate change , climate change total
Unit is: kg CO2-Eq
```

[205]:

	impact at year = 41, points	year at net-zero
h_a_2126_dg_init_wtw	-239.647558	16

```
h_b_2126_dg0_wtw                          123.089244              na
h_c_2126_dg4_wtw                         -790.169779               7
h_d_2126_dg5_wtw                         -653.213298               8
h_e_2126_dg6_wtw                         -477.488079              11
h_f_2126_tg4_wtw                         -700.703160               9
h_g_2126_tg5_wtw                         -697.435846               9
h_h_2126_tg6_wtw                         -458.042443              12
```

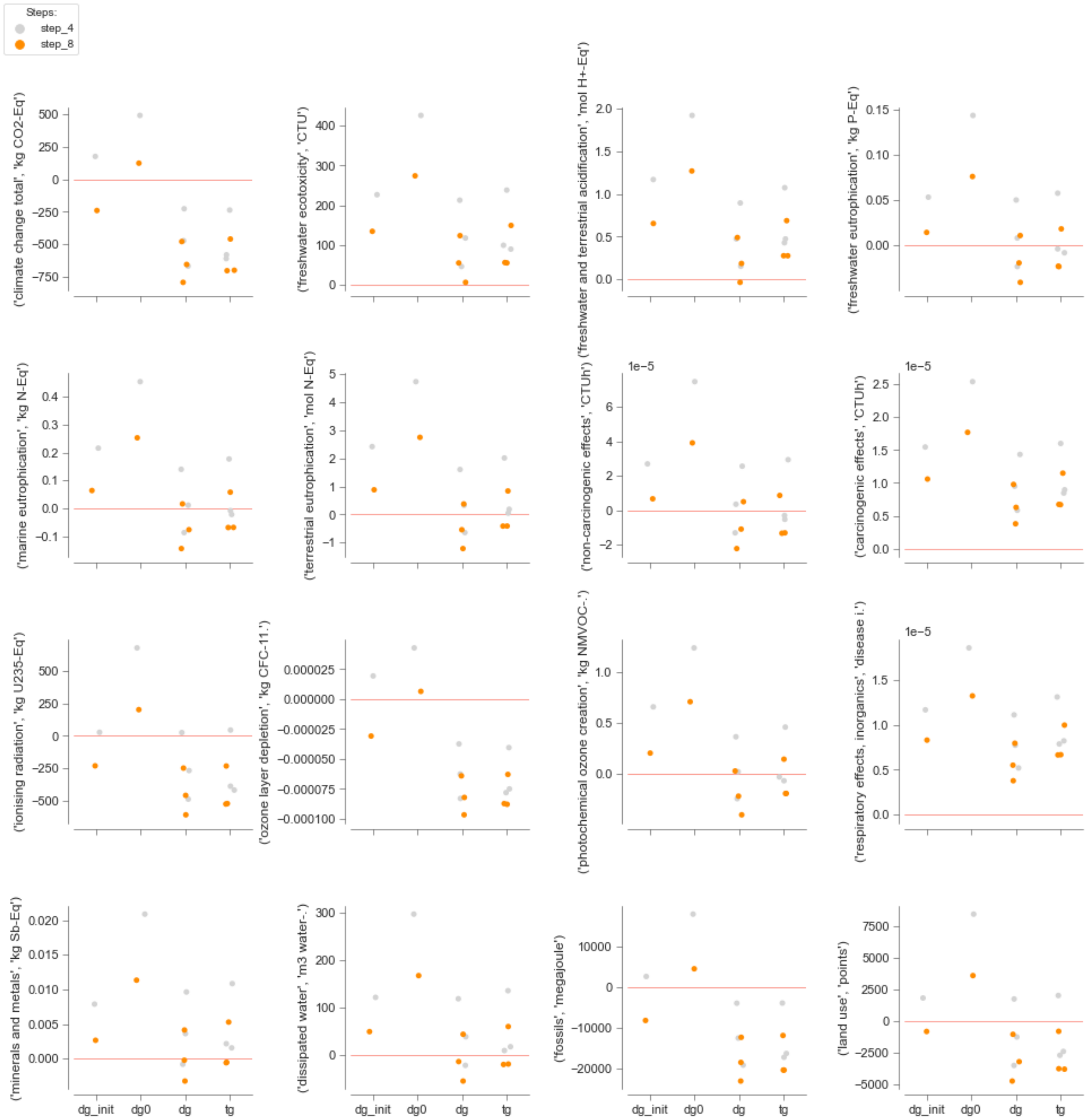


12.4.2 Overall Impact

Compare step 4 and 8. Difference comes from the window-to-wall ratio, 100% and 75% respectively.

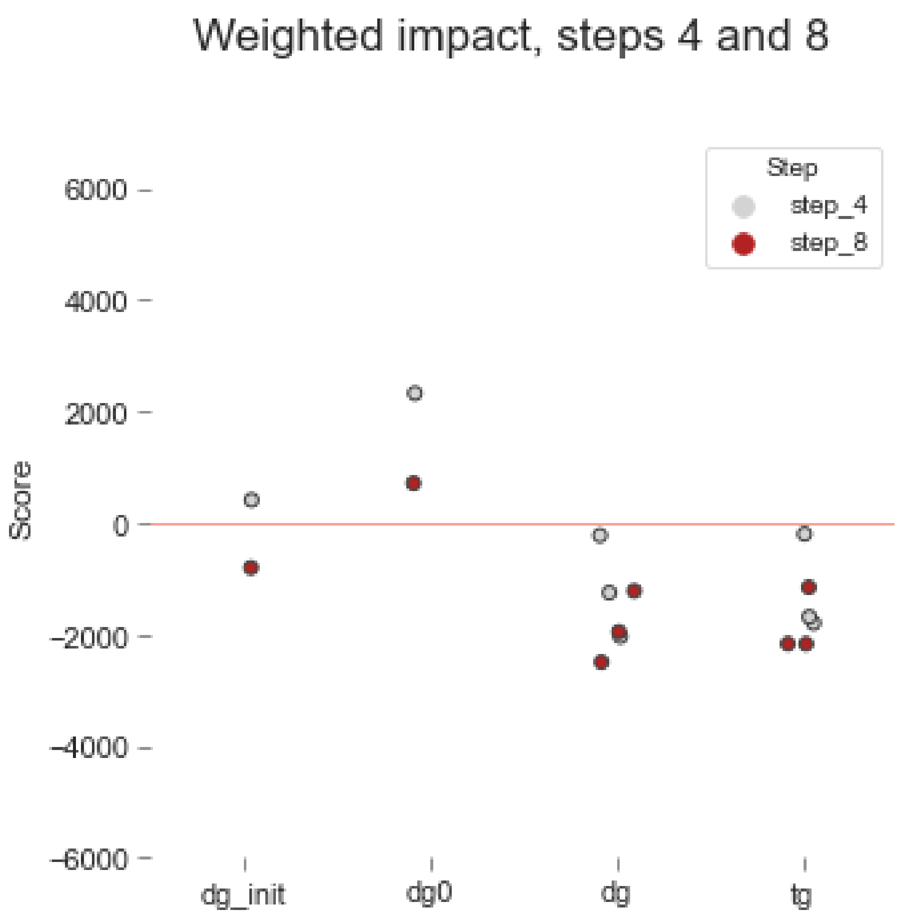
[206]:
```
fig, axes = plt.subplots(nrows=4, ncols=4,
                         sharex=True, sharey=False,
                         figsize=(16, 16))

n = 1

for row in range(4):
    for col in range(4):

        ax = axes[row][col]

        ic = df_ilcd_methods.xs(n, level=1)["Subcategory"][0]
```

```
        i = df_ilcd_methods.index[df_ilcd_methods['Subcategory'] == ic][0][0]
        ic_unit = df_ilcd_methods.xs(n, level=1)["Unit"][0]

        # Keep the lca impact results at the end of life:
        df_plot = df_lca_lifespan[['step_4', 'step_8']].xs(
            ic, axis=1, level=4, drop_level=False).loc[[41]]

        # Transpose:
        df_plot = df_plot.T.unstack(level=(4, 5))[41].reset_index()
        df_plot = df_plot[(df_plot.IGU != "sg")]
        df_plot = df_plot[(df_plot.Scenario != "d_e_2126_dg1_vav") &
                          (df_plot.Scenario != "d_f_2126_dg2_vav") &
                          (df_plot.Scenario != "d_g_2126_dg3_vav") &
                          (df_plot.Scenario != "d_k_2126_tg1_vav") &
                          (df_plot.Scenario != "d_l_2126_tg2_vav") &
                          (df_plot.Scenario != "d_m_2126_tg3_vav")
                          ]

        mycolors = sns.color_palette(['lightgrey', 'darkorange'])

        # Category plot:
        sns.stripplot(data=df_plot, x="IGU",
                      y=(ic, ic_unit),
                      hue="Step", jitter=0.1,
                      palette=mycolors, ax=ax
                      )

        ax.axhline(y=0, c='salmon', linestyle='-', linewidth=0.75)

        ax.get_legend().remove()
        style_ax(ax)

        n += 1

fig.subplots_adjust(wspace=0.55, hspace=0.45)

# Add legend:
handles, labels = ax.get_legend_handles_labels()
fig.legend(handles, labels, loc='center', ncol=1,
           title='Steps:',
           bbox_to_anchor=(0.1, 0.94))

fig.suptitle('', y=0.95)

sns.despine(offset=5)
plt.show()
```




```
[207]: fig, ax = plt.subplots(figsize=(5, 5))

# A second graph:
df_plot = df_weighted[['step_4', 'step_8']]

# Transpose and clean:
df_plot = df_plot.T.reset_index()
df_plot = df_plot[(df_plot.IGU != "sg")]
df_plot = df_plot[(df_plot.Run != "d_e_2126_dg1_vav") &
                  (df_plot.Run != "d_f_2126_dg2_vav") &
                  (df_plot.Run != "d_g_2126_dg3_vav") &
                  (df_plot.Run != "d_k_2126_tg1_vav") &
```

```
                  (df_plot.Run != "d_l_2126_tg2_vav") &
                  (df_plot.Run != "d_m_2126_tg3_vav")
                  ]

mycolors = sns.color_palette(['lightgrey', 'firebrick', 'cornflowerblue'])

# Category plot:
sns.stripplot(data=df_plot, x="IGU",
              y="Score",
              hue="Step", jitter=0.1, linewidth=1,
              palette=mycolors, dodge=False, ax=ax
              )

style_ax(ax)
ax.axhline(y=0, c='salmon', linestyle='-', linewidth=0.75)

fig.suptitle("Weighted impact, steps 4 and 8",
             fontsize=17, y=1)

sns.despine(left=True, bottom=True, offset=5)

ax.set(xlabel="", ylabel="Score")
ax.set_ylim(ymin=-6000, ymax=7000)

plt.show()
```



12.5 Steps 9: High-Tech Glazing Units

12.5.1 Ecological Payback Period

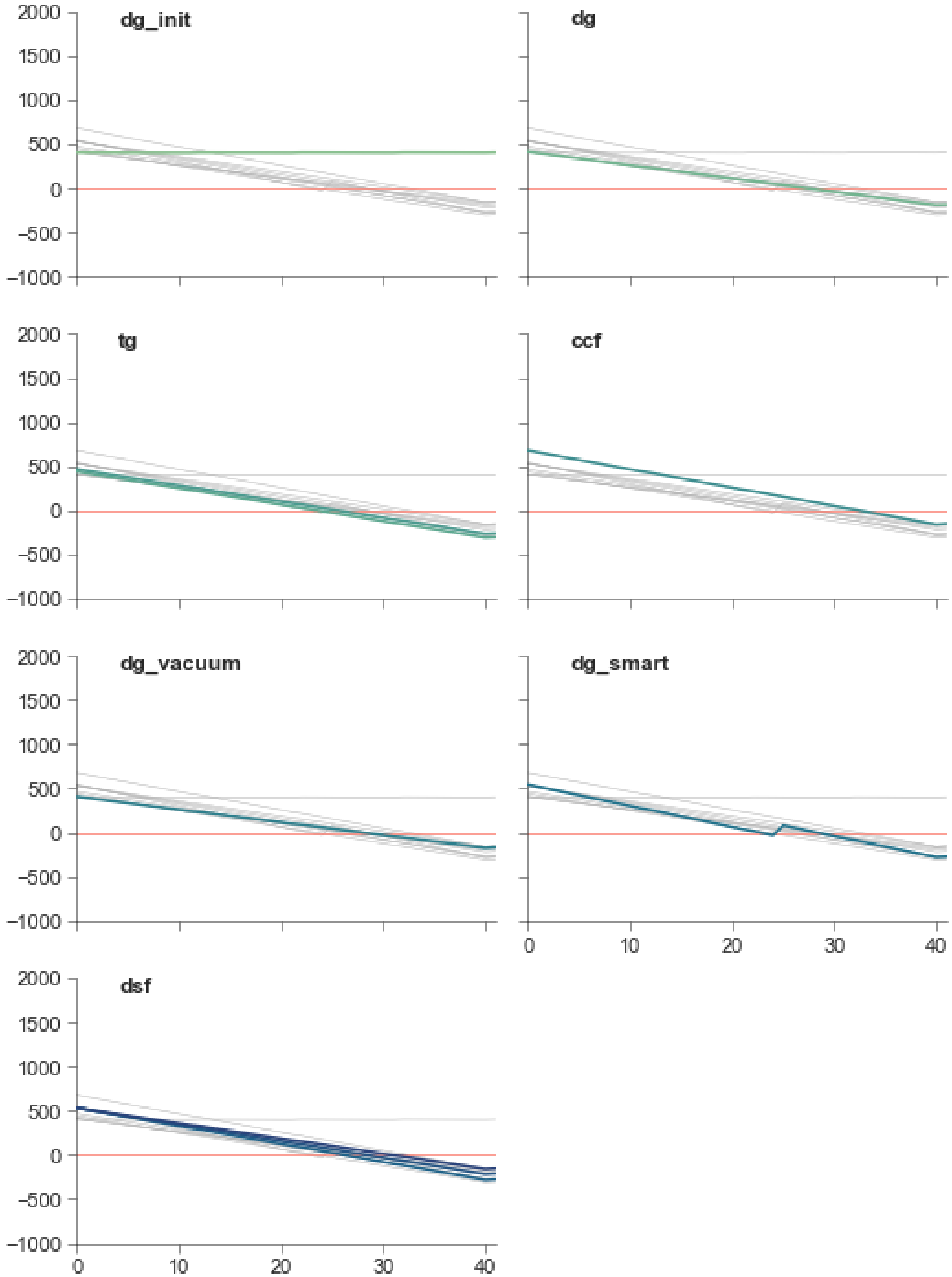
[208]:
```
# Define the rank of the impact category (#):
impact_cat = 1
var = "IGU"
step = "step_9"

plot_multilca_40(step, impact_cat, var, -1000, 2000, 500)
```

```
step_9
climate change , climate change total
Unit is: kg CO2-Eq
```

[208]:

	impact at year = 41, points	year at net-zero
i_a_2126_dg_init_vav_int	402.515952	na
i_b_2126_dg5k	-188.585778	28
i_c_2126_tg5k	-303.938377	24

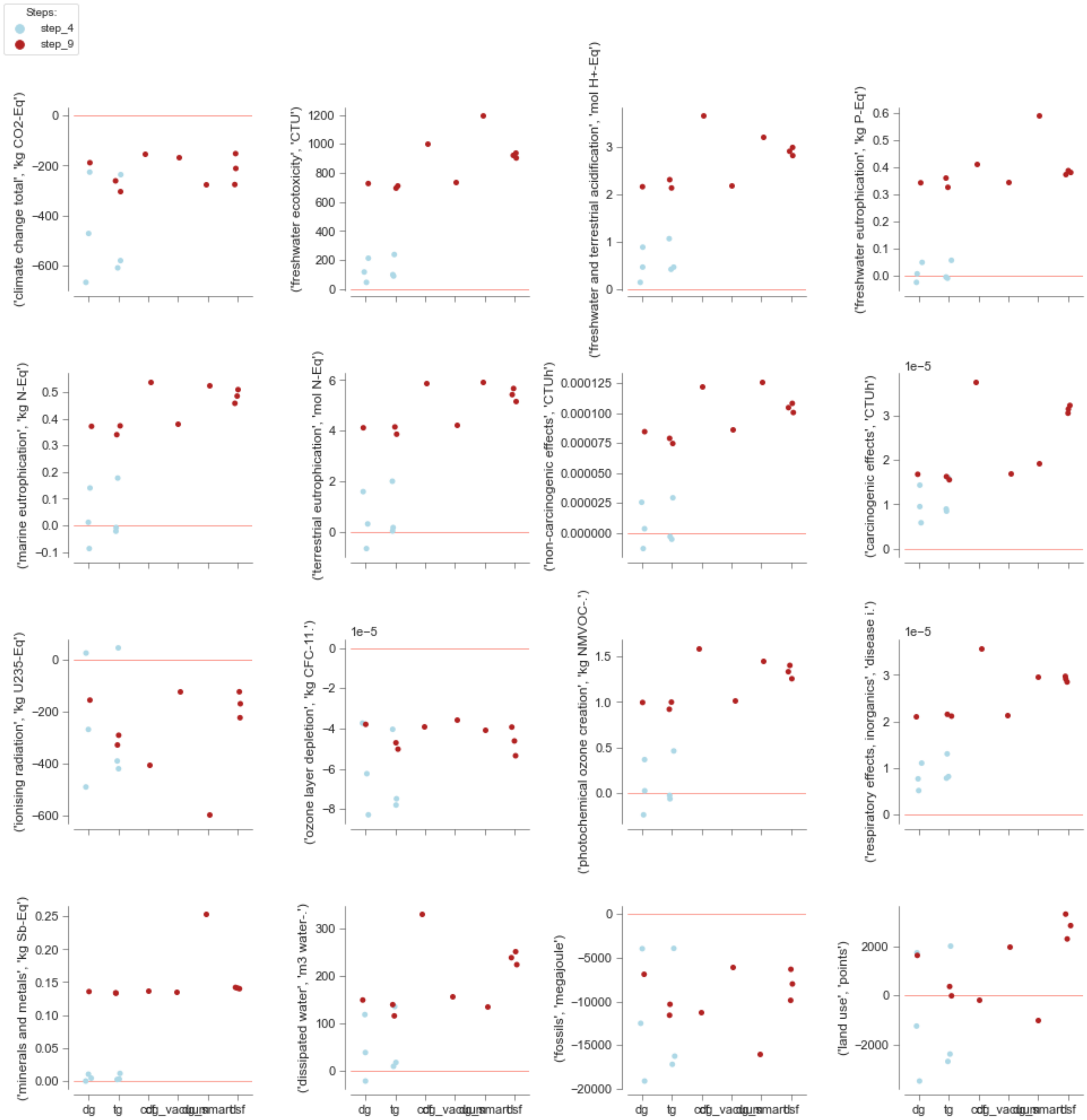
i_d_2126_tg5x	-261.014460	26
i_e_2126_ccf	-150.611092	33
i_f_2126_dg_vacuum	-164.353327	29
i_g_2126_dg_smart	-273.174458	28
i_h_2126_dsf_min	-275.702593	27
i_i_2126_dsf_mean	-211.516432	29
i_j_2126_dsf_max	-152.313493	31


dg_init
dg
tg
ccf
dg_vacuum
dg_smart
dsf
2000
1500
1000
500
0
−500
−1000
0
10
20
30
40

12.5.2 Overall Impact

[209]:
```
fig, axes = plt.subplots(nrows=4, ncols=4,
                         sharex=True, sharey=False,
                         figsize=(16, 16))

n = 1

for row in range(4):
    for col in range(4):

        ax = axes[row][col]

        ic = df_ilcd_methods.xs(n, level=1)["Subcategory"][0]
        i = df_ilcd_methods.index[df_ilcd_methods['Subcategory'] == ic][0][0]
        ic_unit = df_ilcd_methods.xs(n, level=1)["Unit"][0]

        # Keep the lca impact results at the end of life:
        df_plot = df_lca_lifespan[['step_4', 'step_9']].xs(
            ic, axis=1, level=4, drop_level=False).loc[[41]]

        # Transpose and clean:
        df_plot = df_plot.T.unstack(level=(4, 5))[41].reset_index()
        df_plot = df_plot[(df_plot.IGU != "sg")]
        df_plot = df_plot[(df_plot.IGU != "dg0")]
        df_plot = df_plot[(df_plot.IGU != "dg_init")]
        df_plot = df_plot[(df_plot.Scenario != "d_e_2126_dg1_vav") &
                          (df_plot.Scenario != "d_f_2126_dg2_vav") &
                          (df_plot.Scenario != "d_g_2126_dg3_vav") &
                          (df_plot.Scenario != "d_k_2126_tg1_vav") &
                          (df_plot.Scenario != "d_l_2126_tg2_vav") &
                          (df_plot.Scenario != "d_m_2126_tg3_vav")
                          ]

        mycolors = sns.color_palette(['lightblue', 'firebrick'])

        # Category plot:
        sns.stripplot(data=df_plot, x="IGU",
                      y=(ic, ic_unit),
                      hue="Step",
                      palette=mycolors, ax=ax
                      )

        ax.axhline(y=0, c='salmon', linestyle='-', linewidth=0.75)

        ax.get_legend().remove()
        style_ax(ax)
```

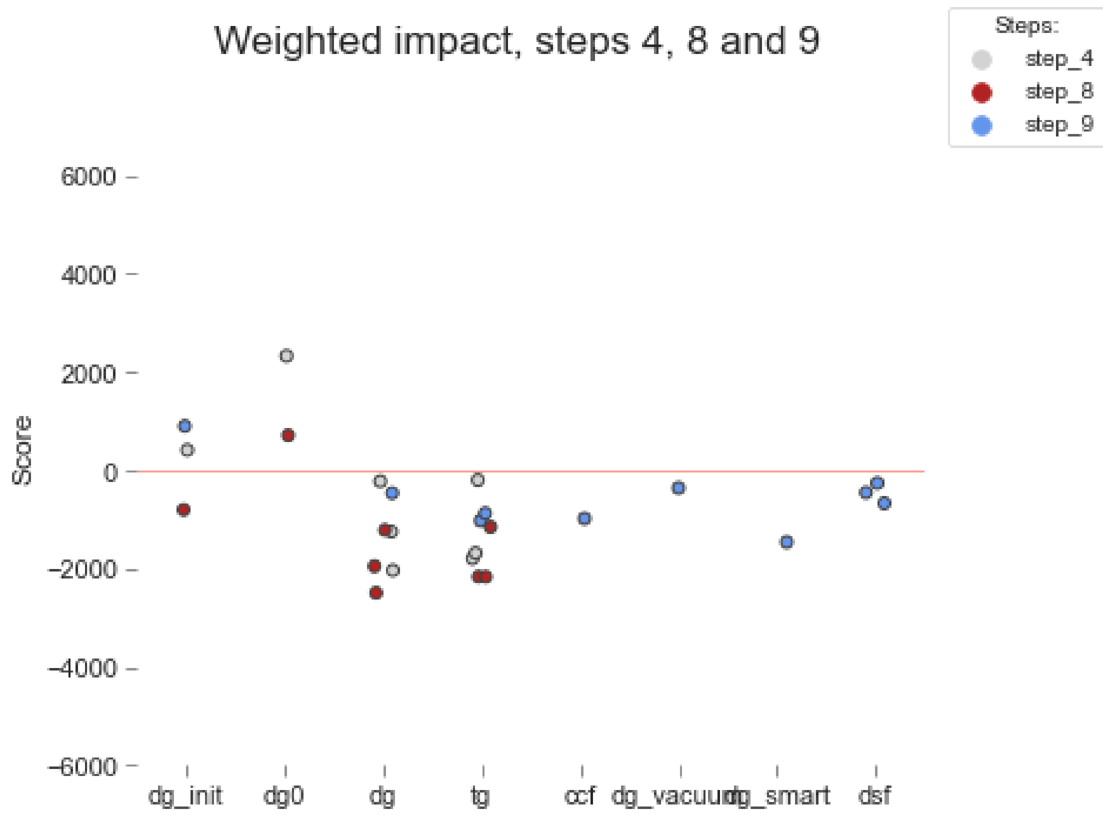
```
        n += 1

fig.subplots_adjust(wspace=0.55, hspace=0.45)

# Add legend:
handles, labels = ax.get_legend_handles_labels()
fig.legend(handles, labels, loc='center', ncol=1,
           title='Steps:',
           bbox_to_anchor=(0.1, 0.94))

fig.suptitle('', y=0.95)

sns.despine(offset=5)

if export:
    # Save image:
    fig.savefig(os.path.join(path_img, 'FullLCIA_Step9.png'),
                dpi=600, bbox_inches='tight')
    fig.savefig(os.path.join(path_img, 'FullLCIA_Step9.pdf'),
                bbox_inches='tight')

plt.show()
```



Analysis of the weighted impact:

[210]:
```
fig, ax = plt.subplots(figsize=(6, 5))

# A second graph:
df_plot = df_weighted[['step_4', 'step_8', 'step_9']]

# Transpose:
df_plot = df_plot.T.reset_index()
df_plot = df_plot[(df_plot.IGU != "sg")]
df_plot = df_plot[(df_plot.Run != "d_e_2126_dg1_vav") &
                  (df_plot.Run != "d_f_2126_dg2_vav") &
```

```
                  (df_plot.Run != "d_g_2126_dg3_vav") &
                  (df_plot.Run != "d_k_2126_tg1_vav") &
                  (df_plot.Run != "d_l_2126_tg2_vav") &
                  (df_plot.Run != "d_m_2126_tg3_vav")
                  ]

mycolors = sns.color_palette(['lightgrey', 'firebrick', 'cornflowerblue'])

# Category plot:
sns.stripplot(data=df_plot, x="IGU",
              y="Score",
              hue="Step", jitter=0.1, linewidth=1,
              palette=mycolors, dodge=False, ax=ax
              )

style_ax(ax)
ax.axhline(y=0, c='salmon', linestyle='-', linewidth=0.75)

fig.suptitle("Weighted impact, steps 4, 8 and 9",
             fontsize=17, y=1)

sns.despine(left=True, bottom=True, offset=5)

ax.set(xlabel="", ylabel="Score")
ax.set_ylim(ymin=-6000, ymax=7000)

# Add legend:
ax.get_legend().remove()
handles, labels = ax.get_legend_handles_labels()
fig.legend(handles, labels, loc='center', ncol=1,
           title='Steps:',
           bbox_to_anchor=(1, 0.94))

if export:
    # Save image:
    fig.savefig(os.path.join(path_img, 'WeightedLCIA_Step8-9.png'),
                dpi=600, bbox_inches='tight')
    fig.savefig(os.path.join(path_img, 'WeightedLCIA_Step8-9.pdf'),
                bbox_inches='tight')

plt.show()
```



12.6 Steps 10-11: Indoor Climate, a Sensitivity Analysis

12.6.1 Ecological Payback Period

"Americanisation"

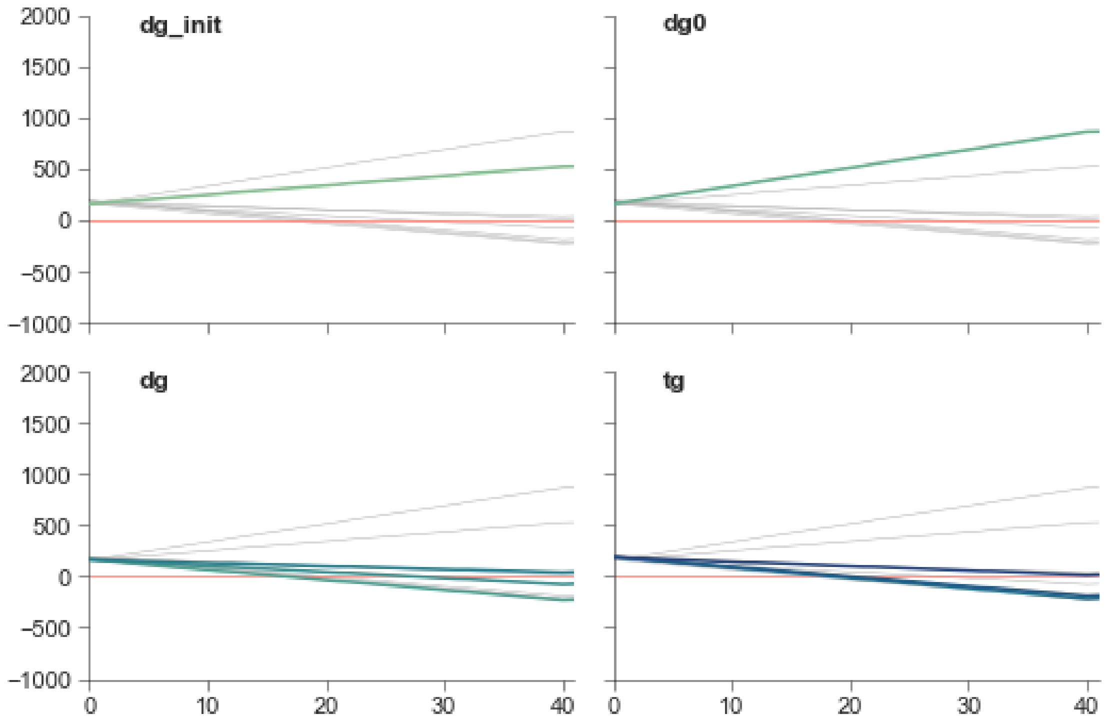
[211]:
```python
# Define the rank of the impact category (#):
impact_cat = 1
var = "IGU"
step = "step_10"

plot_multilca_40(step, impact_cat, var, -1000, 2000, 500)
```

```
step_10
climate change , climate change total
Unit is: kg CO2-Eq
```

[211]:

	impact at year = 41, points	year at net-zero
j_a_2124_dg_init	525.323906	na
j_b_2124_dg0	866.011405	na
j_c_2124_dg4	-228.436596	17
j_d_2124_dg5	-73.100941	28

j_e_2124_dg6	41.682853	na
j_f_2124_tg4	-214.669566	19
j_g_2124_tg5	-185.791912	20
j_h_2124_tg6	17.331677	na



"Sufficiency"

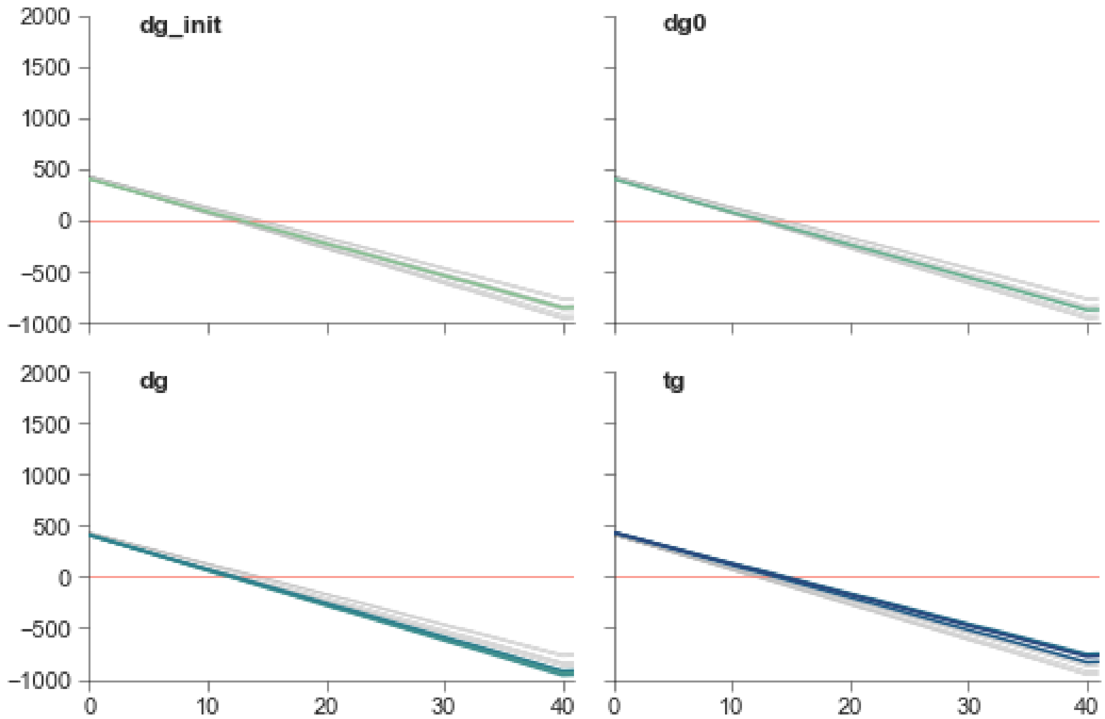
[212]:
```
# Define the rank of the impact category (#):
impact_cat = 1
var = "IGU"
step = "step_11"

plot_multilca_40(step, impact_cat, var, -1000, 2000, 500)
```

```
step_11
climate change , climate change total
Unit is: kg CO2-Eq
```

[212]:

	impact at year = 41, points	year at net-zero
k_a_1927_dg_init_ext	-849.379895	13
k_b_1927_dg0_ext	-872.628187	13
k_c_1927_dg4_ext	-938.165568	12
k_d_1927_dg5_ext	-958.469711	12
k_e_1927_dg6_ext	-913.809837	13

```
k_f_1927_tg4_ext                        -751.593721          15
k_g_1927_tg5_ext                        -829.027837          14
k_h_1927_tg6_ext                        -773.955321          15
```

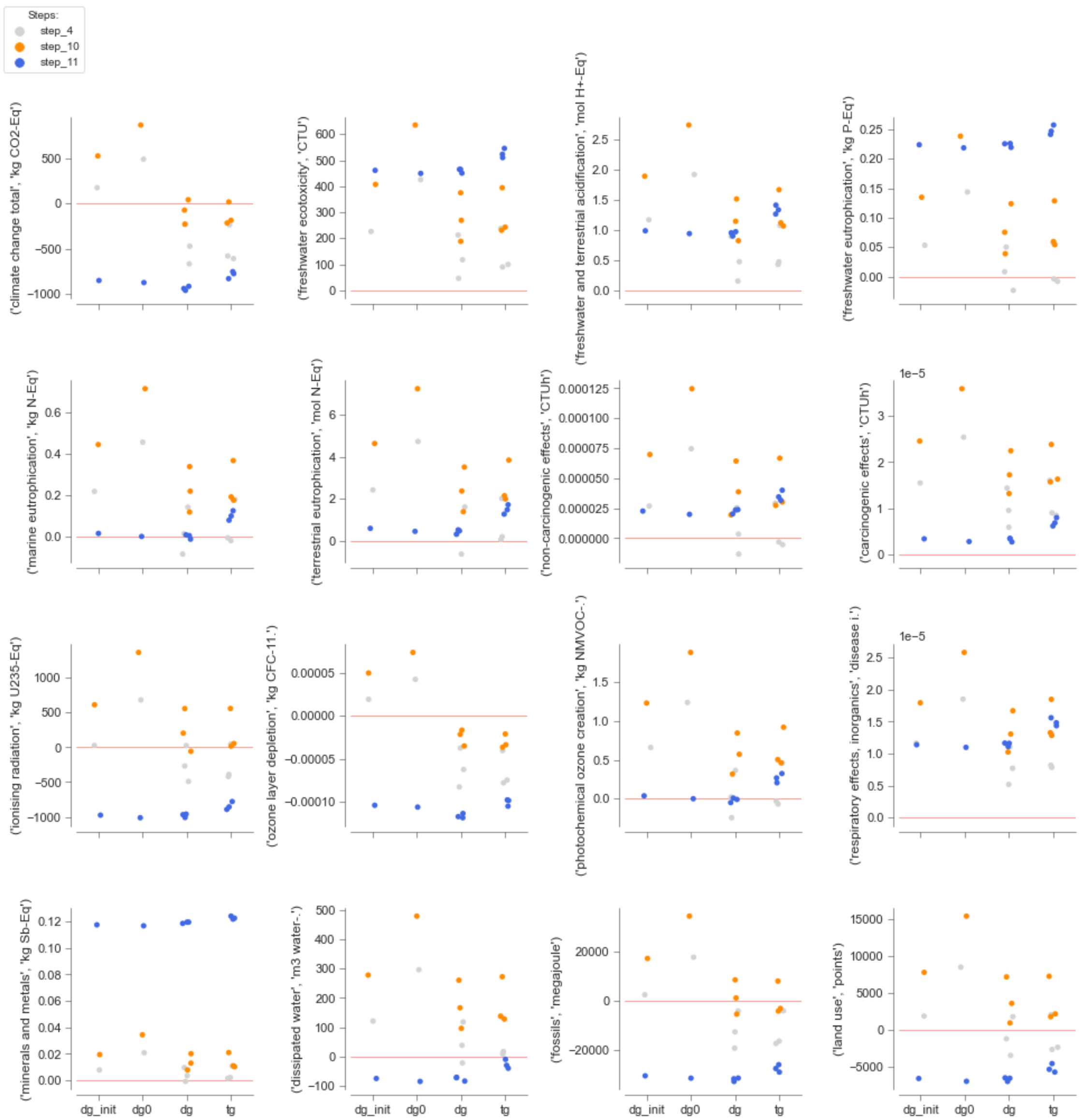


12.6.2 Façade Design and Indoor Comfort: Overall Impact

[213]:
```python
fig, axes = plt.subplots(nrows=4, ncols=4,
                         sharex=True, sharey=False,
                         figsize=(16, 16))

n = 1

for row in range(4):
    for col in range(4):

        ax = axes[row][col]

        ic = df_ilcd_methods.xs(n, level=1)["Subcategory"][0]
        i = df_ilcd_methods.index[df_ilcd_methods['Subcategory'] == ic][0][0]
        ic_unit = df_ilcd_methods.xs(n, level=1)["Unit"][0]

        # Keep the lca impact results at the end of life:
        df_plot = df_lca_lifespan[['step_4', 'step_10', 'step_11']].xs(
```

```
            ic, axis=1, level=4, drop_level=False).loc[[41]]

        # Transpose:
        df_plot = df_plot.T.unstack(level=(4, 5))[41].reset_index()
        df_plot = df_plot[(df_plot.IGU != "sg")]
        df_plot = df_plot[(df_plot.Scenario != "d_e_2126_dg1_vav") &
                          (df_plot.Scenario != "d_f_2126_dg2_vav") &
                          (df_plot.Scenario != "d_g_2126_dg3_vav") &
                          (df_plot.Scenario != "d_k_2126_tg1_vav") &
                          (df_plot.Scenario != "d_l_2126_tg2_vav") &
                          (df_plot.Scenario != "d_m_2126_tg3_vav")
                          ]

        mycolors=sns.color_palette(['lightgrey',
                                     'darkorange', 'royalblue'])

        # Category plot:
        sns.stripplot(data=df_plot, x="IGU",
                      y=(ic, ic_unit),
                      hue="Step",
                      palette=mycolors, ax=ax
                      )

        ax.axhline(y=0, c='salmon', linestyle='-', linewidth=0.75)

        ax.get_legend().remove()
        style_ax(ax)

        n += 1

fig.subplots_adjust(wspace=0.55, hspace=0.45)

# Add legend:
handles, labels = ax.get_legend_handles_labels()
fig.legend(handles, labels, loc='center', ncol=1,
           title='Steps:',
           bbox_to_anchor=(0.1, 0.94))

fig.suptitle('', y=0.95)
sns.despine(offset=5)

if export:
    # Save image:
    fig.savefig(os.path.join(path_img, 'FullLCIA_Step10-11.png'),
                dpi=600, bbox_inches='tight')
    fig.savefig(os.path.join(path_img, 'FullLCIA_Step10-11.pdf'),
                bbox_inches='tight')
```

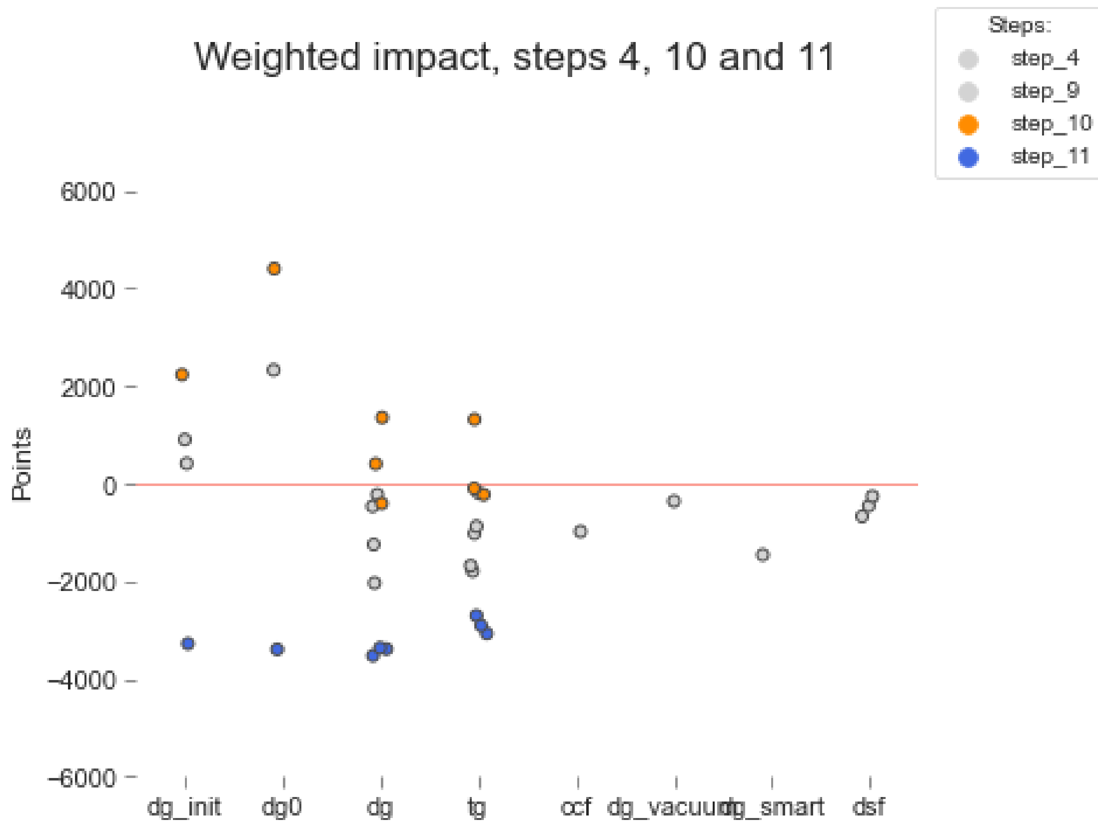
```python
plt.show()
```



Analysis of the weighted impact:

[214]:
```python
fig, ax = plt.subplots(figsize=(6, 5))

# A second graph:
df_plot = df_weighted[['step_4', 'step_9',
                       'step_10', 'step_11']]

# Transpose:
```

```
df_plot = df_plot.T.reset_index()
df_plot = df_plot[(df_plot.IGU != "sg")]
df_plot = df_plot[(df_plot.Run != "d_e_2126_dg1_vav") &
                  (df_plot.Run != "d_f_2126_dg2_vav") &
                  (df_plot.Run != "d_g_2126_dg3_vav") &
                  (df_plot.Run != "d_k_2126_tg1_vav") &
                  (df_plot.Run != "d_l_2126_tg2_vav") &
                  (df_plot.Run != "d_m_2126_tg3_vav")
                  ]

mycolors=sns.color_palette(['lightgrey', 'lightgrey',
                            'darkorange', 'royalblue'])

# Category plot:
sns.stripplot(data=df_plot, x="IGU",
              y="Score",
              hue="Step", jitter=0.1, linewidth=1,
              palette=mycolors, dodge=False, ax=ax
              )

style_ax(ax)
ax.axhline(y=0, c='salmon', linestyle='-', linewidth=0.75)

fig.suptitle("Weighted impact, steps 4, 10 and 11",
             fontsize=17, y=1)

sns.despine(left=True, bottom=True, offset=5)

ax.set(xlabel="", ylabel="Points")
ax.set_ylim(ymin=-6000, ymax=7000)

ax.get_legend().remove()

# Add legend:
handles, labels = ax.get_legend_handles_labels()
fig.legend(handles, labels, loc='center', ncol=1,
           title='Steps:',
           bbox_to_anchor=(1, 0.94))

if export:
    # Save image:
    fig.savefig(os.path.join(path_img, 'WeightedLCIA_Step10-11.png'),
                dpi=600, bbox_inches='tight')
    fig.savefig(os.path.join(path_img, 'WeightedLCIA_Step10-11.pdf'),
                bbox_inches='tight')

plt.show()
```



12.7 Steps 12-13: Internal Heat Gains, a Sensitivity Analysis

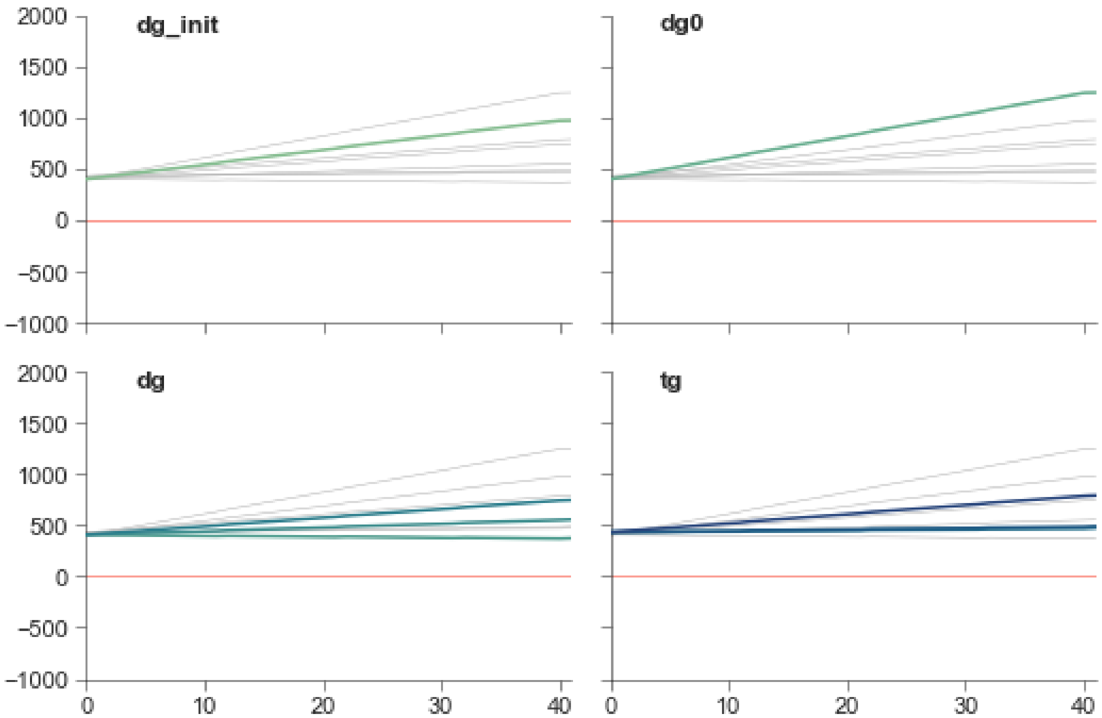
[215]:
```
# Define the rank of the impact category (#):
impact_cat = 1
var = "IGU"
step = "step_12"

plot_multilca_40(step, impact_cat, var, -1000, 2000, 500)
```

```
step_12
climate change , climate change total
Unit is: kg CO2-Eq
```

[215]:

	impact at year = 41, points	year at net-zero
l_a_2126_dg_init_intgain	974.729154	na
l_b_2126_dg0_intgain	1245.691670	na
l_c_2126_dg4_intgain	371.831044	na
l_d_2126_dg5_intgain	553.317264	na
l_e_2126_dg6_intgain	744.032463	na
l_f_2126_tg4_intgain	472.384611	na
l_g_2126_tg5_intgain	488.264803	na

l_h_2126_tg6_intgain	789.483646	na

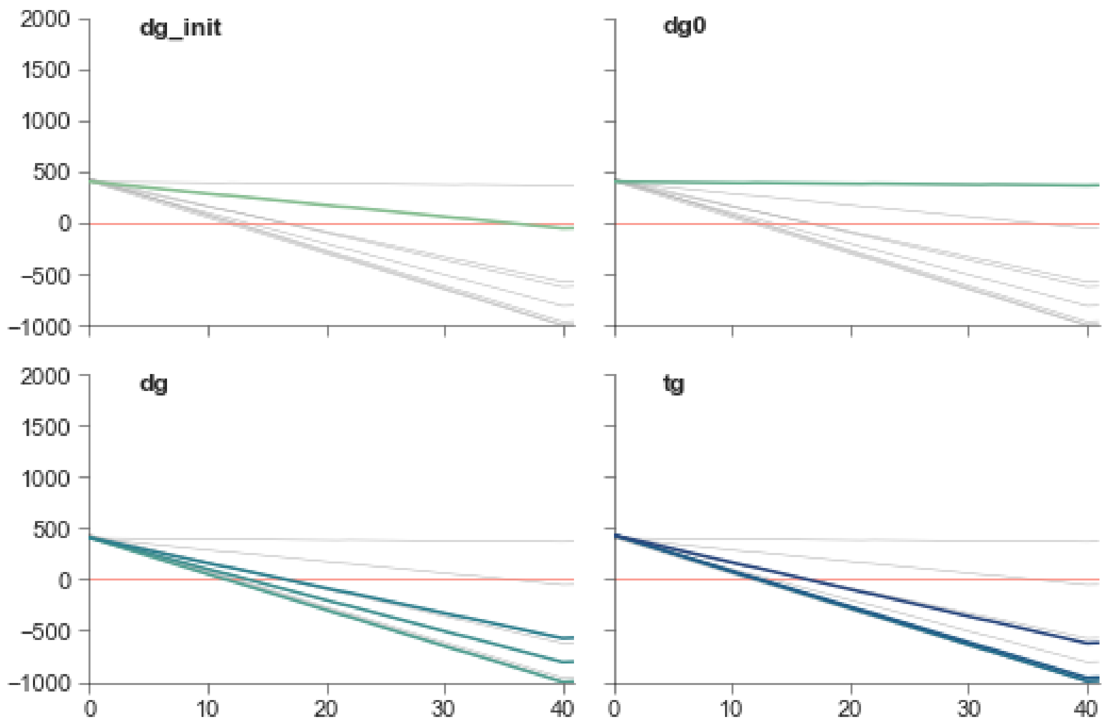


[216]:
```
# Define the rank of the impact category (#):
impact_cat = 1
var = "IGU"
step = "step_13"

plot_multilca_40(step, impact_cat, var, -1000, 2000, 500)
```

```
step_13
climate change , climate change total
Unit is: kg CO2-Eq
```

[216]:

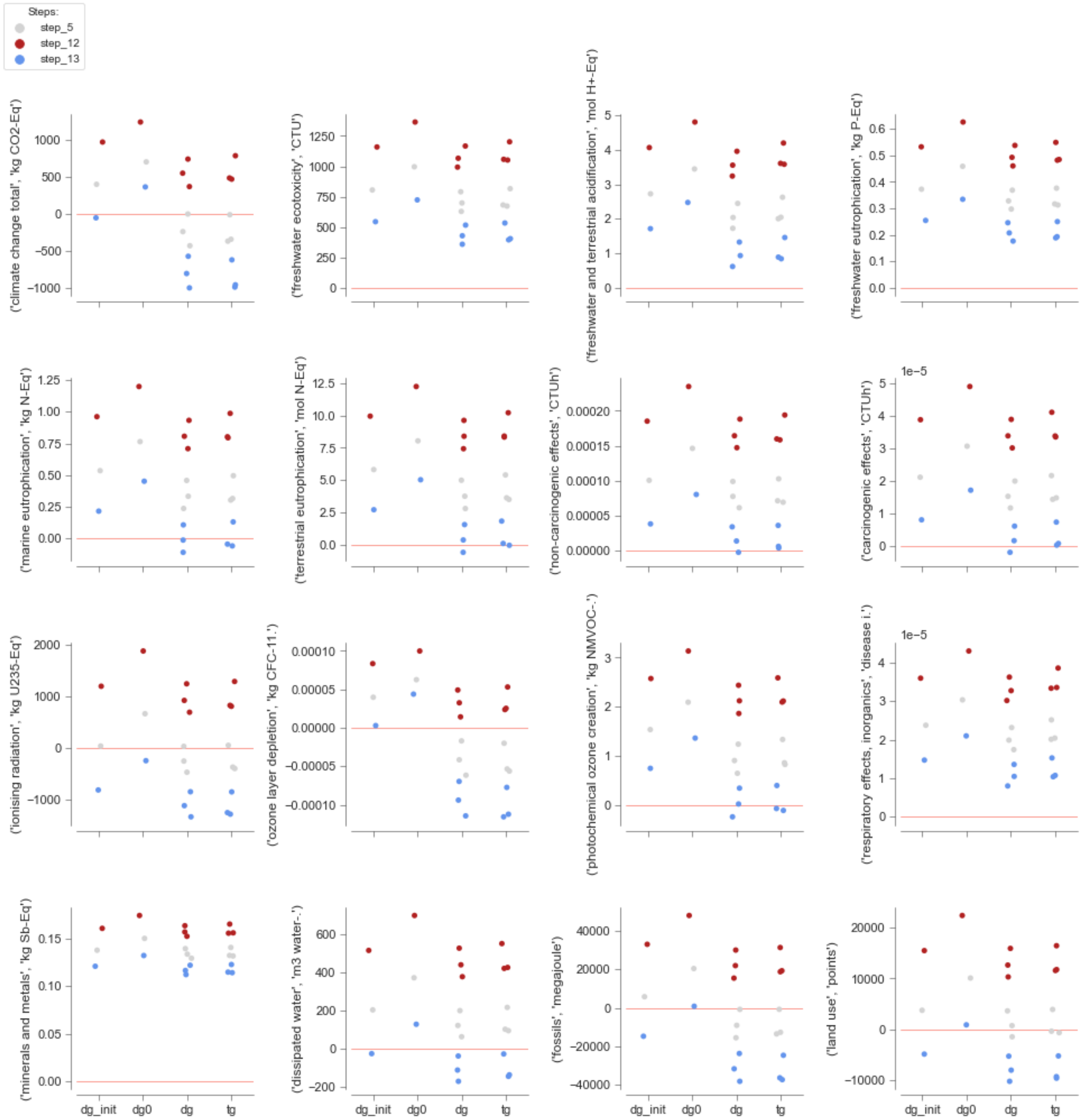
	impact at year = 41, points	year at net-zero
m_a_2126_dg_init_intgain	-51.174822	36
m_b_2126_dg0_intgain	367.226549	na
m_c_2126_dg4_intgain	-998.997317	12
m_d_2126_dg5_intgain	-804.990347	14
m_e_2126_dg6_intgain	-571.499972	17
m_f_2126_tg4_intgain	-988.643503	13
m_g_2126_tg5_intgain	-957.844398	13
m_h_2126_tg6_intgain	-619.884462	17



12.7.1 Overall Impact

[217]:
```
fig, axes = plt.subplots(nrows=4, ncols=4,
                         sharex=True, sharey=False,
                         figsize=(16, 16))

n = 1

for row in range(4):
    for col in range(4):

        ax = axes[row][col]

        ic = df_ilcd_methods.xs(n, level=1)["Subcategory"][0]
        i = df_ilcd_methods.index[df_ilcd_methods['Subcategory'] == ic][0][0]
        ic_unit = df_ilcd_methods.xs(n, level=1)["Unit"][0]

        # Keep the lca impact results at the end of life:
        df_plot = df_lca_lifespan[['step_5', 'step_12', 'step_13']].xs(
            ic, axis=1, level=4, drop_level=False).loc[[41]]

        # Transpose:
        df_plot = df_plot.T.unstack(level=(4, 5))[41].reset_index()
```

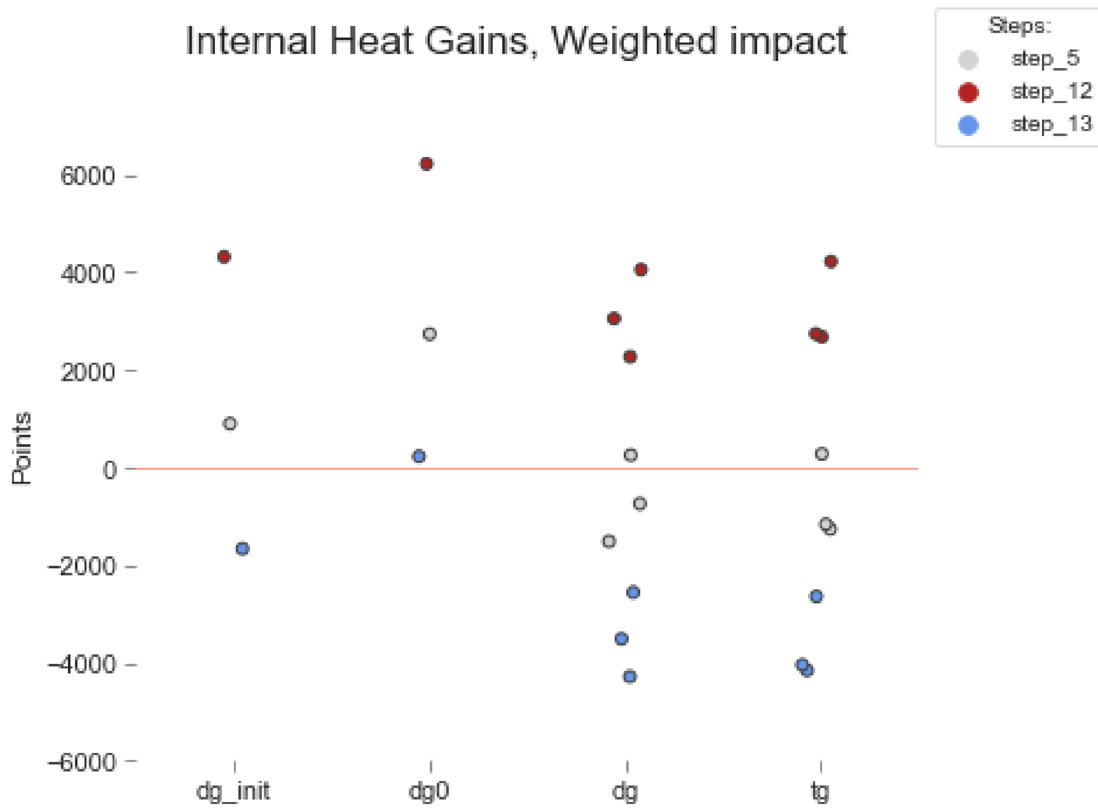
```
        df_plot = df_plot[(df_plot.IGU != "sg")]
        df_plot = df_plot[(df_plot.Scenario != "e_e_2126_dg1_vav_int") &
                          (df_plot.Scenario != "e_f_2126_dg2_vav_int") &
                          (df_plot.Scenario != "e_g_2126_dg3_vav_int") &
                          (df_plot.Scenario != "e_k_2126_tg1_vav_int") &
                          (df_plot.Scenario != "e_l_2126_tg2_vav_int") &
                          (df_plot.Scenario != "e_m_2126_tg3_vav_int")
                          ]

        mycolors = sns.color_palette(['lightgrey', 'firebrick',
                                      'cornflowerblue'])

        # Category plot:
        sns.stripplot(data=df_plot, x="IGU",
                      y=(ic, ic_unit),
                      hue="Step",
                      palette=mycolors, ax=ax
                      )

        ax.axhline(y=0, c='salmon', linestyle='-', linewidth=0.75)

        ax.get_legend().remove()
        style_ax(ax)

        n += 1

fig.subplots_adjust(wspace=0.55, hspace=0.45)

# Add legend:
handles, labels = ax.get_legend_handles_labels()
fig.legend(handles, labels, loc='center', ncol=1,
           title='Steps:',
           bbox_to_anchor=(0.1, 0.94))

fig.suptitle('', y=0.95)

sns.despine(offset=5)

if export:
    # Save image:
    fig.savefig(os.path.join(path_img, 'FullLCIA_Step12-13.png'),
                dpi=600, bbox_inches='tight')
    fig.savefig(os.path.join(path_img, 'FullLCIA_Step12-13.pdf'),
                bbox_inches='tight')
plt.show()
```



Analysis of the weighted impact:

[218]:
```
fig, ax = plt.subplots(figsize=(6, 5))

# A second graph:
df_plot = df_weighted[['step_5', 'step_12', 'step_13']]

# Transpose:
df_plot = df_plot.T.reset_index()
df_plot = df_plot[(df_plot.IGU != "sg")]
df_plot = df_plot[(df_plot.Run != "e_e_2126_dg1_vav_int") &
                  (df_plot.Run != "e_f_2126_dg2_vav_int") &
```

```
                  (df_plot.Run != "e_g_2126_dg3_vav_int") &
                  (df_plot.Run != "e_k_2126_tg1_vav_int") &
                  (df_plot.Run != "e_l_2126_tg2_vav_int") &
                  (df_plot.Run != "e_m_2126_tg3_vav_int")
                  ]

mycolors = sns.color_palette(['lightgrey', 'firebrick', 'cornflowerblue'])

# Category plot:
sns.stripplot(data=df_plot, x="IGU",
              y="Score",
              hue="Step", jitter=0.1, linewidth=1,
              palette=mycolors, dodge=False, ax=ax
              )

style_ax(ax)
ax.axhline(y=0, c='salmon', linestyle='-', linewidth=0.75)

fig.suptitle("Internal Heat Gains, Weighted impact",
             fontsize=17, y=1)

sns.despine(left=True, bottom=True, offset=5)

ax.set(xlabel="", ylabel="Points")
ax.set_ylim(ymin=-6000, ymax=7000)

ax.get_legend().remove()

# Add legend:
handles, labels = ax.get_legend_handles_labels()
fig.legend(handles, labels, loc='center', ncol=1,
           title='Steps:',
           bbox_to_anchor=(1, 0.94))

if export:
    # Save image:
    fig.savefig(os.path.join(path_img, 'WeightedLCIA_IntGain.png'),
                dpi=600, bbox_inches='tight')
    fig.savefig(os.path.join(path_img, 'WeightedLCIA_IntGain.pdf'),
                bbox_inches='tight')

plt.show()
```



12.8 Steps 14-16: Climate Change (2069-2098 - RCP 8.5)

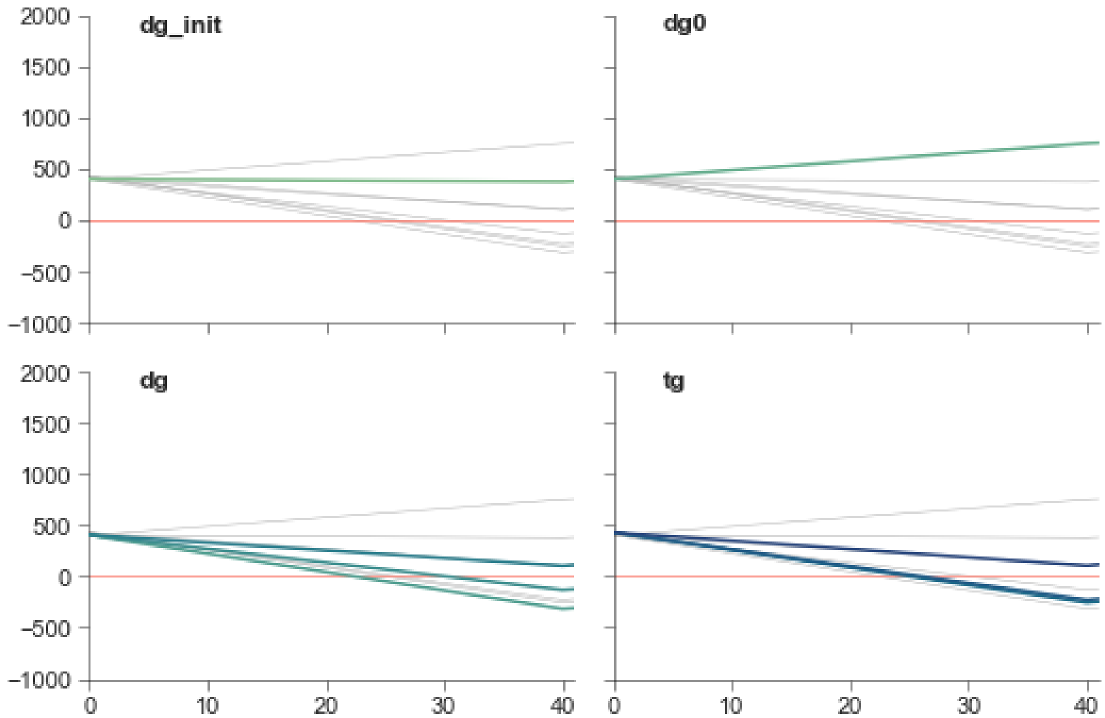
[219]:
```
# Define the rank of the impact category (#):
impact_cat = 1
var = "IGU"
step = "step_14"

plot_multilca_40(step, impact_cat, var, -1000, 2000, 500)
```

```
step_14
climate change , climate change total
Unit is: kg CO2-Eq
```

[219]:

	impact at year = 41, points	year at net-zero
n_a_2126_dg_init_cc	380.989276	na
n_b_2126_dg0_cc	754.894501	na
n_c_2126_dg4_cc	-309.786535	23
n_d_2126_dg5_cc	-124.417211	31
n_e_2126_dg6_cc	111.430375	na
n_f_2126_tg4_cc	-246.594137	26
n_g_2126_tg5_cc	-219.947079	27

n_h_2126_tg6_cc	115.292413	na

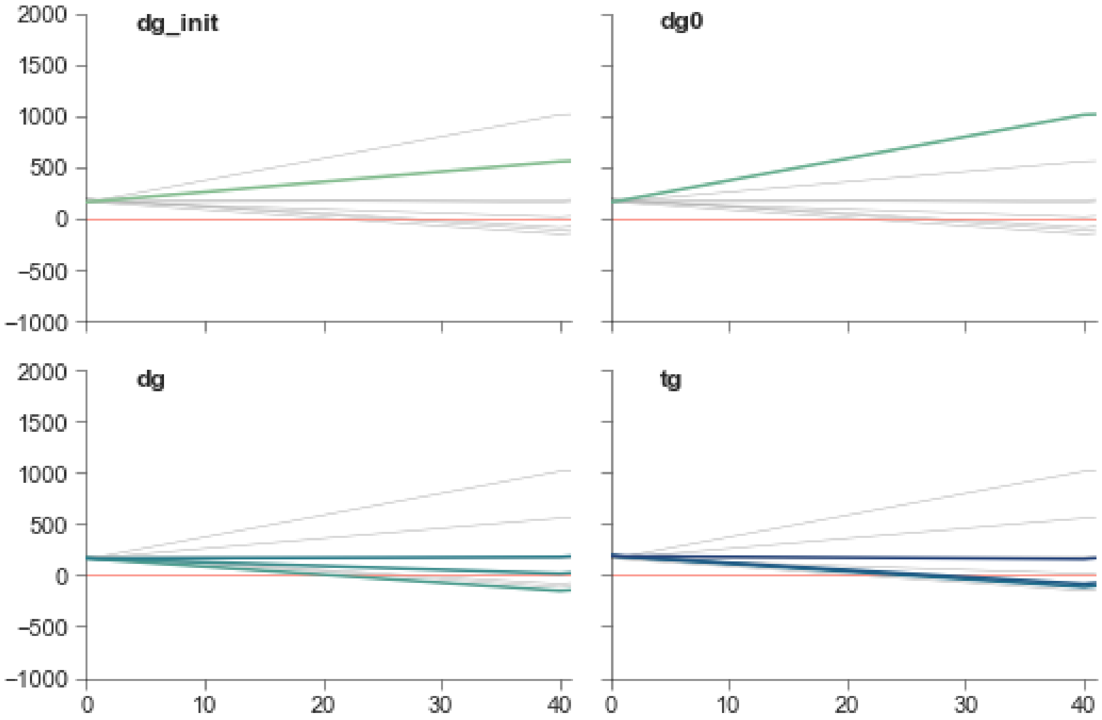


[220]:
```
# Define the rank of the impact category (#):
impact_cat = 1
var = "IGU"
step = "step_15"

plot_multilca_40(step, impact_cat, var, -1000, 2000, 500)
```

```
step_15
climate change , climate change total
Unit is: kg CO2-Eq
```

[220]:

	impact at year = 41, points	year at net-zero
o_a_2124_dg_init_cc	559.855181	na
o_b_2124_dg0_cc	1015.596892	na
o_c_2124_dg4_cc	-145.315737	21
o_d_2124_dg5_cc	23.247869	na
o_e_2124_dg6_cc	185.516369	na
o_f_2124_tg4_cc	-105.743312	25
o_g_2124_tg5_cc	-76.429466	28
o_h_2124_tg6_cc	166.890057	na

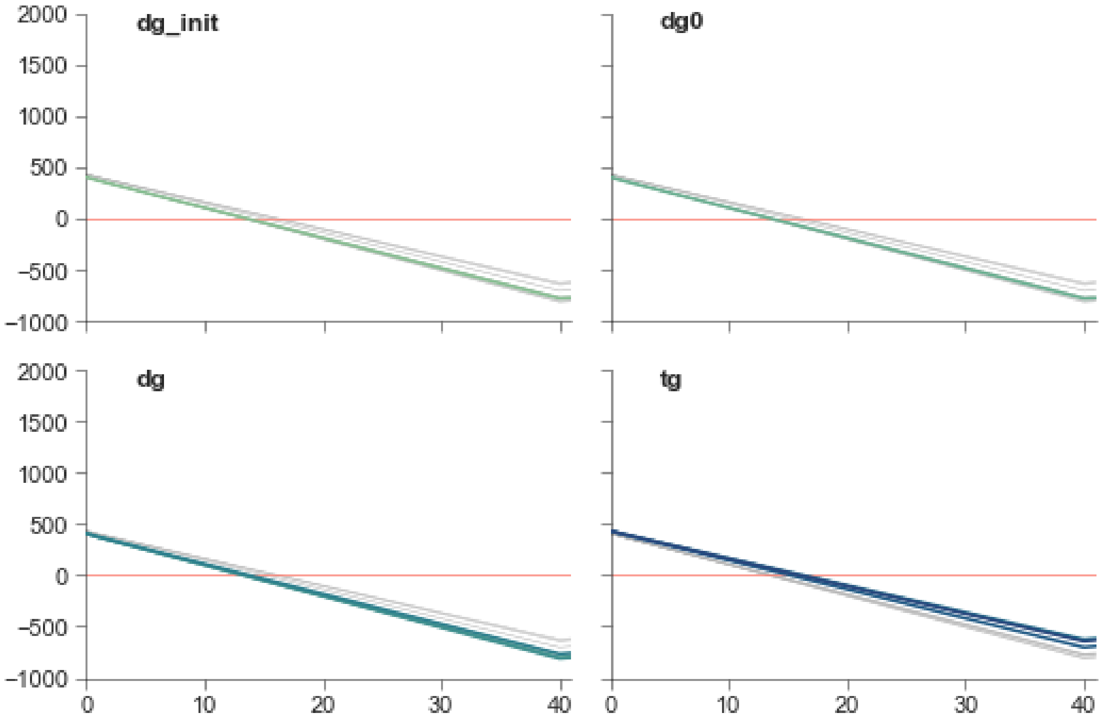


[221]:
```
# Define the rank of the impact category (#):
impact_cat = 1
var = "IGU"
step = "step_16"

plot_multilca_40(step, impact_cat, var, -1000, 2000, 500)
```

```
step_16
climate change , climate change total
Unit is: kg CO2-Eq
```

[221]:

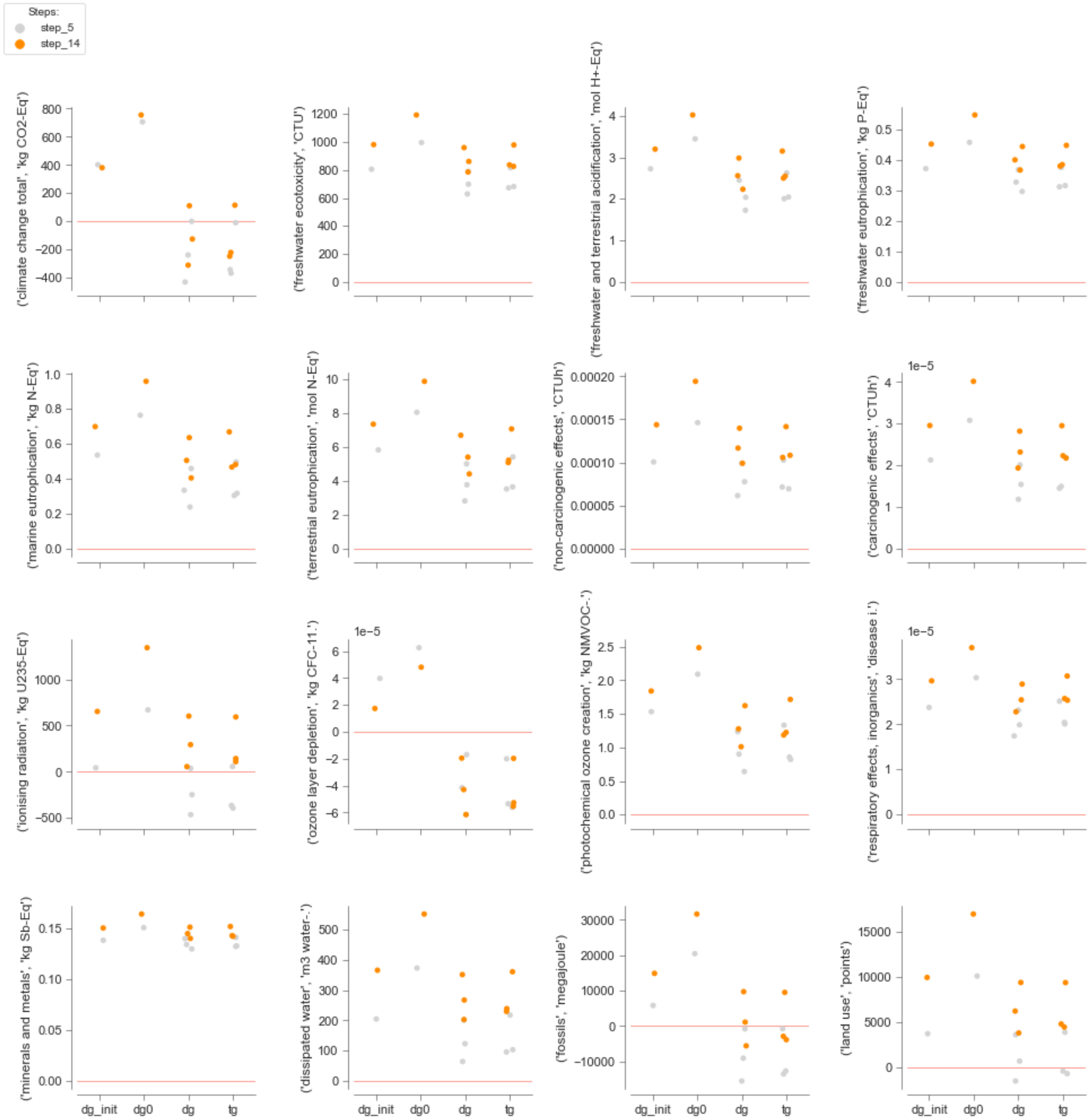
	impact at year = 41, points	year at net-zero
p_a_1927_dg_init_cc	-773.767676	14
p_b_1927_dg0_cc	-774.793824	14
p_c_1927_dg4_cc	-794.917618	14
p_d_1927_dg5_cc	-805.817334	14
p_e_1927_dg6_cc	-760.092198	14
p_f_1927_tg4_cc	-617.289157	17
p_g_1927_tg5_cc	-693.803474	16
p_h_1927_tg6_cc	-633.972912	16



12.8.1 Overall Impact

[222]:
```
fig, axes = plt.subplots(nrows=4, ncols=4,
                         sharex=True, sharey=False,
                         figsize=(16, 16))

n = 1

for row in range(4):
    for col in range(4):

        ax = axes[row][col]

        ic = df_ilcd_methods.xs(n, level=1)["Subcategory"][0]
        i = df_ilcd_methods.index[df_ilcd_methods['Subcategory'] == ic][0][0]
        ic_unit = df_ilcd_methods.xs(n, level=1)["Unit"][0]

        # Keep the lca impact results at the end of life:
        df_plot = df_lca_lifespan[['step_5', 'step_14']
                                ].xs(
            ic, axis=1, level=4, drop_level=False).loc[[41]]

        # Transpose:
```

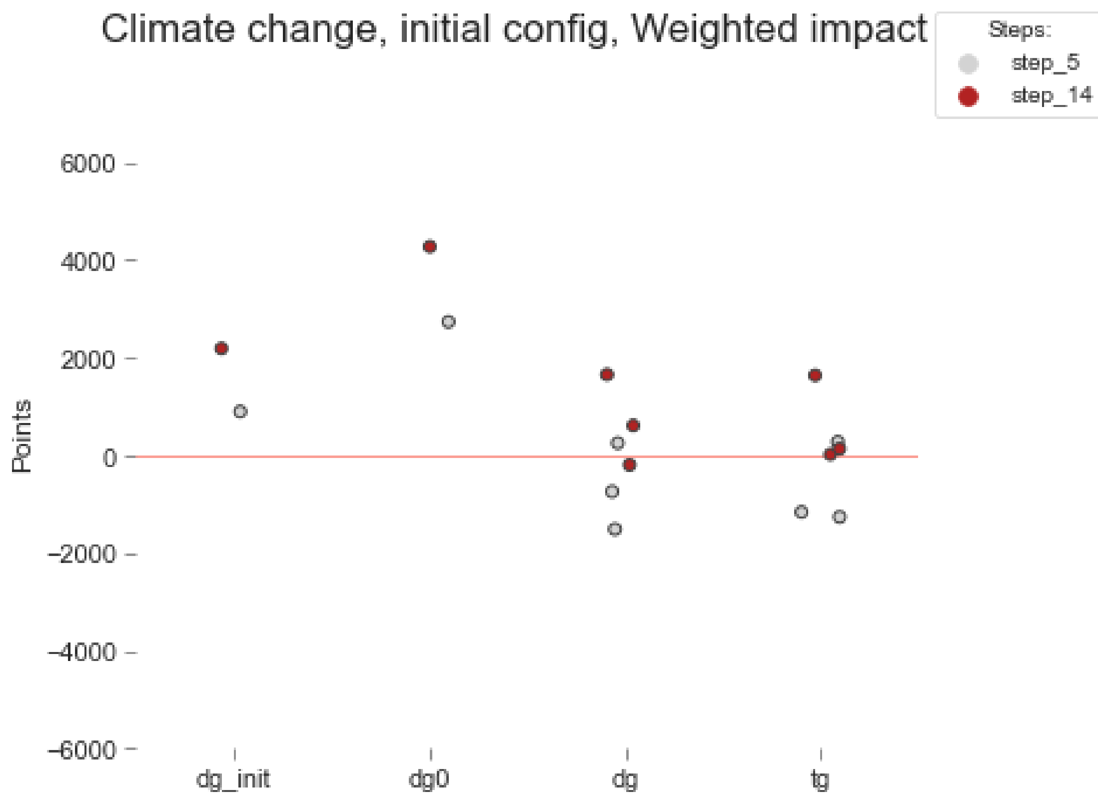
```
        df_plot = df_plot.T.unstack(level=(4, 5))[41].reset_index()
        df_plot = df_plot[(df_plot.IGU != "sg")]
        df_plot = df_plot[(df_plot.Scenario != "e_e_2126_dg1_vav_int") &
                          (df_plot.Scenario != "e_f_2126_dg2_vav_int") &
                          (df_plot.Scenario != "e_g_2126_dg3_vav_int") &
                          (df_plot.Scenario != "e_k_2126_tg1_vav_int") &
                          (df_plot.Scenario != "e_l_2126_tg2_vav_int") &
                          (df_plot.Scenario != "e_m_2126_tg3_vav_int")
                          ]

        mycolors = sns.color_palette(['lightgrey', 'darkorange',
                                      'firebrick', 'royalblue']
                                     )

        # Category plot:
        sns.stripplot(data=df_plot, x="IGU",
                      y=(ic, ic_unit),
                      hue="Step",
                      palette=mycolors, ax=ax
                      )

        ax.axhline(y=0, c='salmon', linestyle='-', linewidth=0.75)

        ax.get_legend().remove()
        style_ax(ax)

        n += 1

fig.subplots_adjust(wspace=0.55, hspace=0.45)

# Add legend:
handles, labels = ax.get_legend_handles_labels()
fig.legend(handles, labels, loc='center', ncol=1,
           title='Steps:',
           bbox_to_anchor=(0.1, 0.94))

fig.suptitle('', y=0.95)

sns.despine(offset=5)
plt.show()
```



Analysis of the weighted impact:

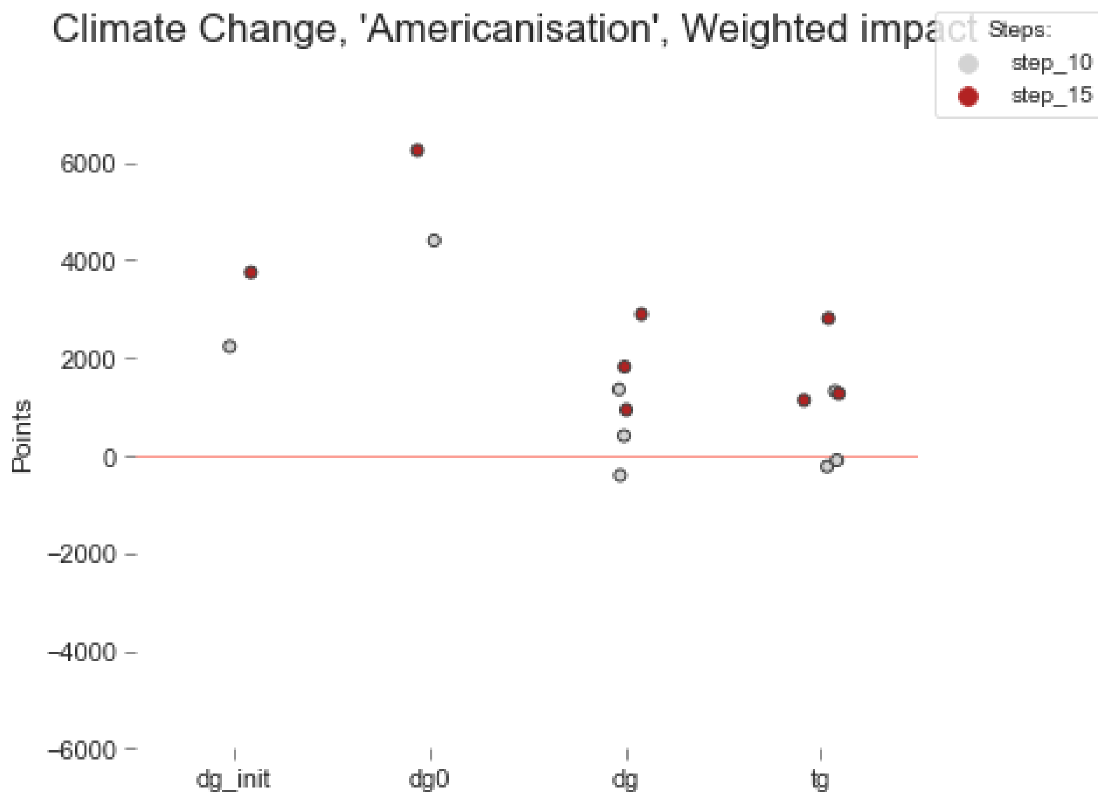
[223]:
```
fig, ax = plt.subplots(figsize=(6, 5))

# A second graph:
df_plot = df_weighted[['step_5', 'step_14']]

# Transpose and clean:
df_plot = df_plot.T.reset_index()
df_plot = df_plot[(df_plot.IGU != "sg")]
df_plot = df_plot[(df_plot.Run != "e_e_2126_dg1_vav_int") &
                  (df_plot.Run != "e_f_2126_dg2_vav_int") &
```

```python
                  (df_plot.Run != "e_g_2126_dg3_vav_int") &
                  (df_plot.Run != "e_k_2126_tg1_vav_int") &
                  (df_plot.Run != "e_l_2126_tg2_vav_int") &
                  (df_plot.Run != "e_m_2126_tg3_vav_int")
                  ]

mycolors = sns.color_palette(['lightgrey', 'firebrick'])

# Plot:
sns.stripplot(data=df_plot, x="IGU",
              y="Score",
              hue="Step", jitter=0.1, linewidth=1,
              palette=mycolors, dodge=False, ax=ax
              )

style_ax(ax)
ax.axhline(y=0, c='salmon', linestyle='-', linewidth=0.75)

fig.suptitle("Climate change, initial config, Weighted impact",
             fontsize=17, y=1)

sns.despine(left=True, bottom=True, offset=5)

ax.set(xlabel="", ylabel="Points")
ax.set_ylim(ymin=-6000, ymax=7000)

ax.get_legend().remove()

# Add legend:
handles, labels = ax.get_legend_handles_labels()
fig.legend(handles, labels, loc='center', ncol=1,
           title='Steps:',
           bbox_to_anchor=(1, 0.94))

if export:
    # Save image:
    fig.savefig(os.path.join(path_img, 'WeightedLCIA_CC_mid.png'),
              dpi=600, bbox_inches='tight')
    fig.savefig(os.path.join(path_img, 'WeightedLCIA_CC_mid.pdf'),
              bbox_inches='tight')

plt.show()
```



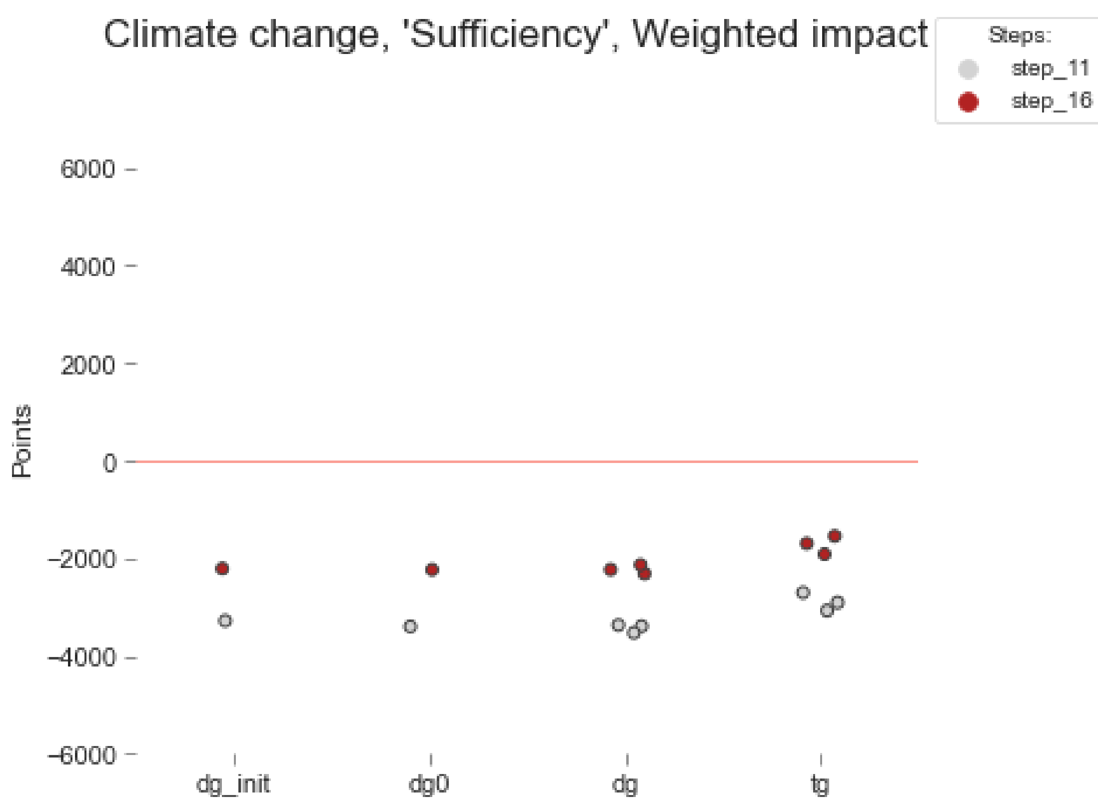
[224]:
```
fig, ax = plt.subplots(figsize=(6, 5))

# A second graph:
df_plot = df_weighted[['step_10', 'step_15']]

for step, run, igu in df_plot.columns:
    if igu == "sg":
        df_plot = df_plot.drop((step, run, igu), axis=1)

# Transpose:
df_plot = df_plot.T.reset_index()
df_plot

mycolors = sns.color_palette(['lightgrey', 'firebrick'])

# Plot:
sns.stripplot(data=df_plot, x="IGU",
              y="Score",
              hue="Step", jitter=0.1, linewidth=1,
              palette=mycolors, dodge=False, ax=ax
              )
```

```python
style_ax(ax)
ax.axhline(y=0, c='salmon', linestyle='-', linewidth=0.75)

fig.suptitle("Climate Change, 'Americanisation', Weighted impact",
             fontsize=17, y=1)

sns.despine(left=True, bottom=True, offset=5)

ax.set(xlabel="", ylabel="Points")
ax.set_ylim(ymin=-6000, ymax=7000)

ax.get_legend().remove()

# Add legend:
handles, labels = ax.get_legend_handles_labels()
fig.legend(handles, labels, loc='center', ncol=1,
           title='Steps:',
           bbox_to_anchor=(1, 0.94))

if export:
    # Save image:
    fig.savefig(os.path.join(path_img, 'WeightedLCIA_CC_high.png'),
              dpi=600, bbox_inches='tight')
    fig.savefig(os.path.join(path_img, 'WeightedLCIA_CC_high.pdf'),
              bbox_inches='tight')

plt.show()
```



[225]:
```
fig, ax = plt.subplots(figsize=(6, 5))

# A second graph:
df_plot = df_weighted[['step_11', 'step_16']]

# Transpose and clean:
df_plot = df_plot.T.reset_index()
df_plot = df_plot[df_plot.IGU != "sg"]

mycolors = sns.color_palette(['lightgrey', 'firebrick'])

# Plot:
sns.stripplot(data=df_plot, x="IGU",
              y="Score",
              hue="Step", jitter=0.1, linewidth=1,
              palette=mycolors, dodge=False, ax=ax
              )

style_ax(ax)
ax.axhline(y=0, c='salmon', linestyle='-', linewidth=0.75)
```

```python
fig.suptitle("Climate change, 'Sufficiency', Weighted impact",
             fontsize=17, y=1)

sns.despine(left=True, bottom=True, offset=5)

ax.set(xlabel="", ylabel="Points")
ax.set_ylim(ymin=-6000, ymax=7000)

ax.get_legend().remove()

# Add legend:
handles, labels = ax.get_legend_handles_labels()
fig.legend(handles, labels, loc='center', ncol=1,
           title='Steps:',
           bbox_to_anchor=(1, 0.94))

if export:
    # Save image:
    fig.savefig(os.path.join(path_img, 'WeightedLCIA_CC_low.png'),
                dpi=600, bbox_inches='tight')
    fig.savefig(os.path.join(path_img, 'WeightedLCIA_CC_low.pdf'),
                bbox_inches='tight')

plt.show()
```



13 Electricity Mix, Sensitivity Analysis

13.1 Setup: Locations and LCI

[226]:
```
# List of activities to change, in this case electricity markets:
locations = ["FR", "BE", "DE", "PL", "NL", "CH", "DK"]

act_name = "market for electricity, low voltage"

elec_market = [('ecoinvent 3.7 cut-off', act['code'])
               for act in eicutdb.search(act_name, limit=200)
               for location in locations
               if act_name in act['name'] and location in act['location']
               and "US-FRCC" not in act['location']
               and "US-SERC" not in act['location']
               ]

# Remove "market for electricity, low voltage, label-certified" for CH:
elec_market.pop(5)

elec_market
```

[226]:
```
[('ecoinvent 3.7 cut-off', '6e1189e153866a1560758372211ec84c'),
 ('ecoinvent 3.7 cut-off', '305ad0ec795ec36bf78943c707a31268'),
 ('ecoinvent 3.7 cut-off', 'a7fa45115215a361e25f837326598b29'),
 ('ecoinvent 3.7 cut-off', '7bc4b453729a015dd7e893756faac612'),
 ('ecoinvent 3.7 cut-off', 'ee8156af3a095a1ca3851ebc3aa5e8e2'),
 ('ecoinvent 3.7 cut-off', 'e9afdf474c494ac44701e8bea53a1f28'),
 ('ecoinvent 3.7 cut-off', '25d7b9f0c2e006f0a6564c9732e1e276')]
```

[227]:
```
# Displaying the exchanges
print('My activity is:\n', prod_and_use_cw,
      '\n-------\nAnd its exchanges:\n-------')

for i in list(prod_and_use_cw.exchanges()):
    print(i['type'])
    print(i)
    print(i['input'])
    print('-------')
```

```
My activity is:
 'use of curtain wall' (square meter, BE, ('building components', 'windows'))
-------
And its exchanges:
-------
```

```
technosphere
Exchange: 1 square meter 'curtain wall, production' (square meter, BE,
('building components', 'windows')) to 'use of curtain wall' (square meter, BE,
('building components', 'windows'))>
('exldb_cw', 'production_cw')
-------
technosphere
Exchange: 0.0 kilowatt hour 'market for electricity, low voltage' (kilowatt
hour, BE, None) to 'use of curtain wall' (square meter, BE, ('building
components', 'windows'))>
('ecoinvent 3.7 cut-off', 'e9afdf474c494ac44701e8bea53a1f28')
-------
technosphere
Exchange: 0.0 megajoule 'heat production, natural gas, at boiler condensing
modulating >100kW' (megajoule, Europe without Switzerland, None) to 'use of
curtain wall' (square meter, BE, ('building components', 'windows'))>
('ecoinvent 3.7 cut-off', 'deecfcb7f97e73711df8990176bfcbb9')
-------
production
Exchange: 1 square meter 'use of curtain wall' (square meter, BE, ('building
components', 'windows')) to 'use of curtain wall' (square meter, BE, ('building
components', 'windows'))>
('exldb_cw', 'use_cw')
-------
technosphere
Exchange: 1 square meter 'curtain wall, end of life' (square meter, BE,
('building components', 'windows')) to 'use of curtain wall' (square meter, BE,
('building components', 'windows'))>
('exldb_cw', 'eol_cw')
-------
```

[228]:
```
exc_elec = list(prod_and_use_cw.exchanges())[1]
exc_elec
```

[228]:
```
Exchange: 0.0 kilowatt hour 'market for electricity, low voltage' (kilowatt
hour, BE, None) to 'use of curtain wall' (square meter, BE, ('building
components', 'windows'))>
```

The following boolean defines whether the LCIA if conducted (True) or if the csv file, where previous results are stored, is directly imported (False).

[229]:
```
# Conducting the LCIA?
calc_lcia_energymix = True
```

13.2 Analysis with Fan Coil Chiller and Natural Gas Boiler

This section conducts a LCIA according to the ILCD midpoint method, including the 16 impact indicators.

If the calculation is undertaken, be patient, it takes time!

[230]:
```
step_code = "step_1"

if calc_lcia_energymix:

    # Dropping scenarios w/ single glazing & double/triple w/ low light transm
    # Defining a new dataframe only with parameters useful for the LCIA:
    df_param = df_step1.drop(
        ["a_a_2126_dg_init", "a_c_2126_sg1", "a_d_2126_sg2",
         "a_e_2126_dg1", "a_f_2126_dg2", "a_g_2126_dg3",
         "a_k_2126_tg1", "a_l_2126_tg2", "a_m_2126_tg3"]
    ).drop(['glazing', 'heating_setpoint',
            'cooling_setpoint'], axis=1)

    df_energymix_results_step1 = pd.DataFrame()

    # Converting dataframe in a numpy array:
    val_np = df_param.to_numpy()

    # A list to save the results:
    ls_mlca_full_results = []

    for run_n, v in enumerate(val_np):

        for param_name in df_param.columns:
            loc_param = df_param.columns.get_loc(param_name)

            (ActivityParameter.update(amount=v[loc_param])
             .where(ActivityParameter.name == f'param_{param_name}'
                    ).execute()
             )

        ActivityParameter.recalculate_exchanges("cw_use_param_group")
        ActivityParameter.recalculate_exchanges("cw_eol_param_group")

        for n, m in enumerate(elec_market):
            my_act_elec = (
                Database('ecoinvent 3.7 cut-off').get(elec_market[n][1])
            )

            loc = my_act_elec['location']

            name_scenario = str(df_param.index[run_n])+"_"+loc

            # Make a copy of the activity, substitute the background process
            prod_and_use_cw_copy = prod_and_use_cw.copy()
```

```
        exc_elec = list(prod_and_use_cw_copy.exchanges())[1]
        exc_elec['input'] = m
        exc_elec.save()

        lca = LCA({prod_and_use_cw_copy: 1})
        lca.lci()

        step = step_code+"_"+loc

        # Conducting the LCIA:
        for method in ls_method_full:
            lca.switch_method(method)
            lca.lcia()
            ls_mlca_full_results.append((step, name_scenario,
                                         method[1], method[2],
                                         lca.score,
                                         bw.methods.get(method).get('unit'))
                                         )

# New DataFrame from list of results:
df_energymix_results_step1 = pd.DataFrame(ls_mlca_full_results,
                                          columns=["Step",
                                                   "Name",
                                                   "Category",
                                                   "Subcategory",
                                                   "Score",
                                                   "Unit"
                                                   ]
                                          )

# Pivot the DataFrame:
df_energymix_results_step1 = pd.pivot_table(df_energymix_results_step1,
                                            index=["Step",
                                                   "Name"
                                                   ],
                                            columns=["Category",
                                                     "Subcategory",
                                                     "Unit"
                                                     ],
                                            values="Score"
                                            )

# Save df_mlca_full_raw_results to csv:
df_energymix_results_step1.unstack([0, 1]).to_csv(
    'outputs\lca\df_energymix_results_'+str(step_code)+'.csv',
    index=True)
```

```
else:
    # Open the csv file, to avoid recalculating the impacts:
    if os.path.isfile(
            'outputs\lca\df_energymix_results_'+str(step_code)+'.csv'):
        with pd.option_context('display.precision', 10):
            df_energymix_results_step1 = (
                pd.read_csv(
                    'outputs\lca\df_energymix_results_'+str(step_code)+'.csv',
                    float_precision=None)
            )

        df_energymix_results_step1 = df_energymix_results_step1.pivot_table(
            values='0',
            index=['Step', 'Name'],
            columns=['Category', 'Subcategory', 'Unit']
        )

    else:
        print("df_mlca_full_raw_results does not exist!")
```

Reorganising the DataFrame, integrating the type of IGU for each simulation run:

[231]:
```
# Add a row to sort by IGU type (column indexing):
ls_igu = []
for code in df_energymix_results_step1.index.get_level_values('Name'):
    if "dg_init" in code:
        ls_igu.append("dg_init")
    if "dg0" in code:
        ls_igu.append("dg0")
    if "sg" in code:
        ls_igu.append("sg")
    if (("dg1" in code) or ("dg2" in code) or ("dg3" in code)
            or ("dg4" in code) or ("dg5" in code) or ("dg6" in code)):
        ls_igu.append("dg")
    if (("tg1" in code) or ("tg2" in code) or ("tg3" in code)
            or ("tg4" in code) or ("tg5" in code) or ("tg6" in code)):
        ls_igu.append("tg")
    if "dg_vacuum" in code:
        ls_igu.append("dg_vacuum")
    if "dg_smart" in code:
        ls_igu.append("dg_smart")
    if "dsf" in code:
        ls_igu.append("dsf")
    if "ccf" in code:
        ls_igu.append("ccf")

df_energymix_results_step1.loc[:, ('IGU')] = ls_igu
```

```
df_energymix_results_step1 = (
    df_energymix_results_step1.reset_index().set_index(
        ["Step", "Name", "IGU"])
)
```

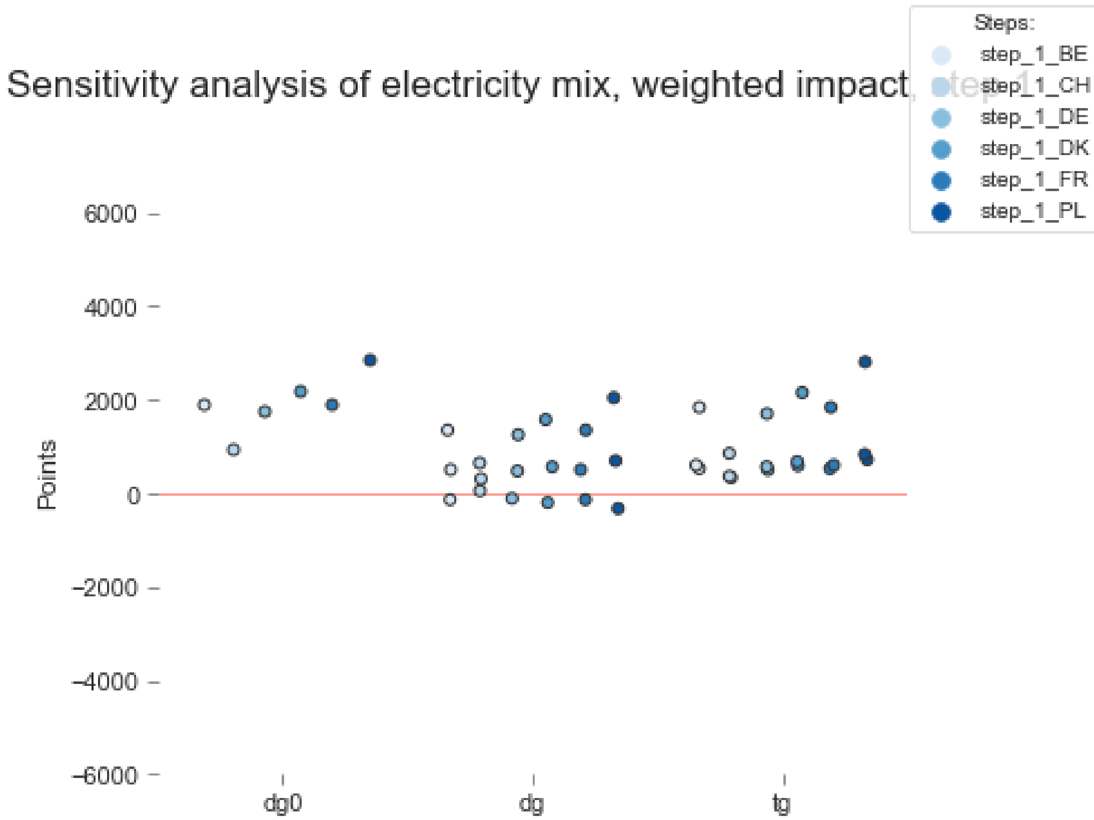
Weighting:

[232]:
```
df_weighted_energymix_step1 = pd.DataFrame(
    (df_energymix_results_step1.multiply(df_weighting["Weighting factor"].T,
                                         axis=1) / float(100)).sum(axis=1),
    columns=["Weighted impact"]
).T
```

Analysis of the weighted impact:

[233]:
```
fig, ax = plt.subplots(figsize=(6, 5))

df_plot = df_weighted_energymix_step1.T.reset_index()

# Category plot:
sns.stripplot(data=df_plot, x="IGU",
              y="Weighted impact",
              hue="Step", jitter=0.1, linewidth=1,
              palette="Blues", dodge=True, ax=ax
              )

style_ax(ax)
ax.axhline(y=0, c='salmon', linestyle='-', linewidth=0.75)

fig.suptitle("Sensitivity analysis of electricity mix, weighted impact, step 1",
             fontsize=17, y=1)

sns.despine(left=True, bottom=True, offset=5)

ax.set(xlabel="", ylabel="Points")
ax.set_ylim(ymin=-6000, ymax=7000)

ax.get_legend().remove()

# Add legend:
handles, labels = ax.get_legend_handles_labels()
fig.legend(handles, labels, loc='center', ncol=1,
           title='Steps:',
           bbox_to_anchor=(1, 0.94))

if export:
    # Save image:
```

```
    fig.savefig(os.path.join(path_img, 'WeightedLCIA_Elec_HVAC_1.png'),
                dpi=600, bbox_inches='tight')
    fig.savefig(os.path.join(path_img, 'WeightedLCIA_Elec_HVAC_1.pdf'),
                bbox_inches='tight')

plt.show()
```



13.3 Analysis with Optimised VAV System

This section conducts a LCIA according to the ILCD midpoint method, including the 16 impact indicators.

If the calculation is undertaken, be patient, it takes time!

[234]:
```
step_code = "step_4"

if calc_lcia_energymix:

    # Dropping scenarios w/ single glazing & double/triple w/ low light transm
    # Defining a new dataframe only with parameters useful for the LCIA:
    df_param = df_step4.drop(
```

```
    ["d_c_2126_sg1_vav", "d_d_2126_sg2_vav", "d_e_2126_dg1_vav",
     "d_f_2126_dg2_vav", "d_g_2126_dg3_vav", "d_k_2126_tg1_vav",
     "d_l_2126_tg2_vav", "d_m_2126_tg3_vav", "d_a_2126_dg_init_vav"]
).drop(['glazing', 'heating_setpoint',
        'cooling_setpoint'], axis=1)

df_energymix_results_step4 = pd.DataFrame()

# Converting dataframe in a numpy array:
val_np = df_param.to_numpy()

# A list to save the results:
ls_mlca_full_results = []

for run_n, v in enumerate(val_np):

    for param_name in df_param.columns:
        loc_param = df_param.columns.get_loc(param_name)

        (ActivityParameter.update(amount=v[loc_param])
         .where(ActivityParameter.name == f'param_{param_name}'
                ).execute()
         )

    ActivityParameter.recalculate_exchanges("cw_use_param_group")
    ActivityParameter.recalculate_exchanges("cw_eol_param_group")

    for n, m in enumerate(elec_market):
        my_act_elec = (
            Database('ecoinvent 3.7 cut-off').get(elec_market[n][1])
        )

        loc = my_act_elec['location']

        name_scenario = str(df_param.index[run_n])+"_"+loc

        # Make a copy of the activity, substitute the background process
        prod_and_use_cw_copy = prod_and_use_cw.copy()
        exc_elec = list(prod_and_use_cw_copy.exchanges())[1]
        exc_elec['input'] = m
        exc_elec.save()

        lca = LCA({prod_and_use_cw_copy: 1})
        lca.lci()

        step = step_code+"_"+loc
```

```
            # Conducting the LCIA:
            for method in ls_method_full:
                lca.switch_method(method)
                lca.lcia()
                ls_mlca_full_results.append((step, name_scenario,
                                            method[1], method[2],
                                            lca.score,
                                            bw.methods.get(method).get('unit'))
                                            )

    # New DataFrame from list of results:
    df_energymix_results_step4 = pd.DataFrame(ls_mlca_full_results,
                                              columns=["Step",
                                                       "Name",
                                                       "Category",
                                                       "Subcategory",
                                                       "Score",
                                                       "Unit"
                                                       ]
                                              )

    # Pivot the DataFrame:
    df_energymix_results_step4 = pd.pivot_table(df_energymix_results_step4,
                                                index=["Step",
                                                       "Name"
                                                       ],
                                                columns=["Category",
                                                         "Subcategory",
                                                         "Unit"
                                                         ],
                                                values="Score"
                                                )

    # Save df_mlca_full_raw_results to csv:
    df_energymix_results_step4.unstack([0, 1]).to_csv(
        'outputs\lca\df_energymix_results_'+str(step_code)+'.csv',
        index=True)

else:
    # Open the csv file, to avoid recalculating the impacts:
    if os.path.isfile(
            'outputs\lca\df_energymix_results_'+str(step_code)+'.csv'):
        with pd.option_context('display.precision', 10):
            df_energymix_results_step4 = (
                pd.read_csv(
                    'outputs\lca\df_energymix_results_'+str(step_code)+'.csv',
                    float_precision=None)
```

```
            )

        df_energymix_results_step4 = df_energymix_results_step4.pivot_table(
            values='0',
            index=['Step', 'Name'],
            columns=['Category', 'Subcategory', 'Unit']
        )

    else:
        print("df_mlca_full_raw_results does not exist!")
```

Reorganising the DataFrame, integrating the type of IGU for each simulation run:

[235]:
```
# Add a row to sort by IGU type (column indexing):
ls_igu = []
for code in df_energymix_results_step4.index.get_level_values('Name'):
    if "dg_init" in code:
        ls_igu.append("dg_init")
    if "dg0" in code:
        ls_igu.append("dg0")
    if "sg" in code:
        ls_igu.append("sg")
    if (("dg1" in code) or ("dg2" in code) or ("dg3" in code)
            or ("dg4" in code) or ("dg5" in code) or ("dg6" in code)):
        ls_igu.append("dg")
    if (("tg1" in code) or ("tg2" in code) or ("tg3" in code)
            or ("tg4" in code) or ("tg5" in code) or ("tg6" in code)):
        ls_igu.append("tg")
    if "dg_vacuum" in code:
        ls_igu.append("dg_vacuum")
    if "dg_smart" in code:
        ls_igu.append("dg_smart")
    if "dsf" in code:
        ls_igu.append("dsf")
    if "ccf" in code:
        ls_igu.append("ccf")

df_energymix_results_step4.loc[:, ('IGU')] = ls_igu

df_energymix_results_step4 = (
    df_energymix_results_step4.reset_index().set_index(
        ["Step", "Name", "IGU"])
)
```

Weighting:

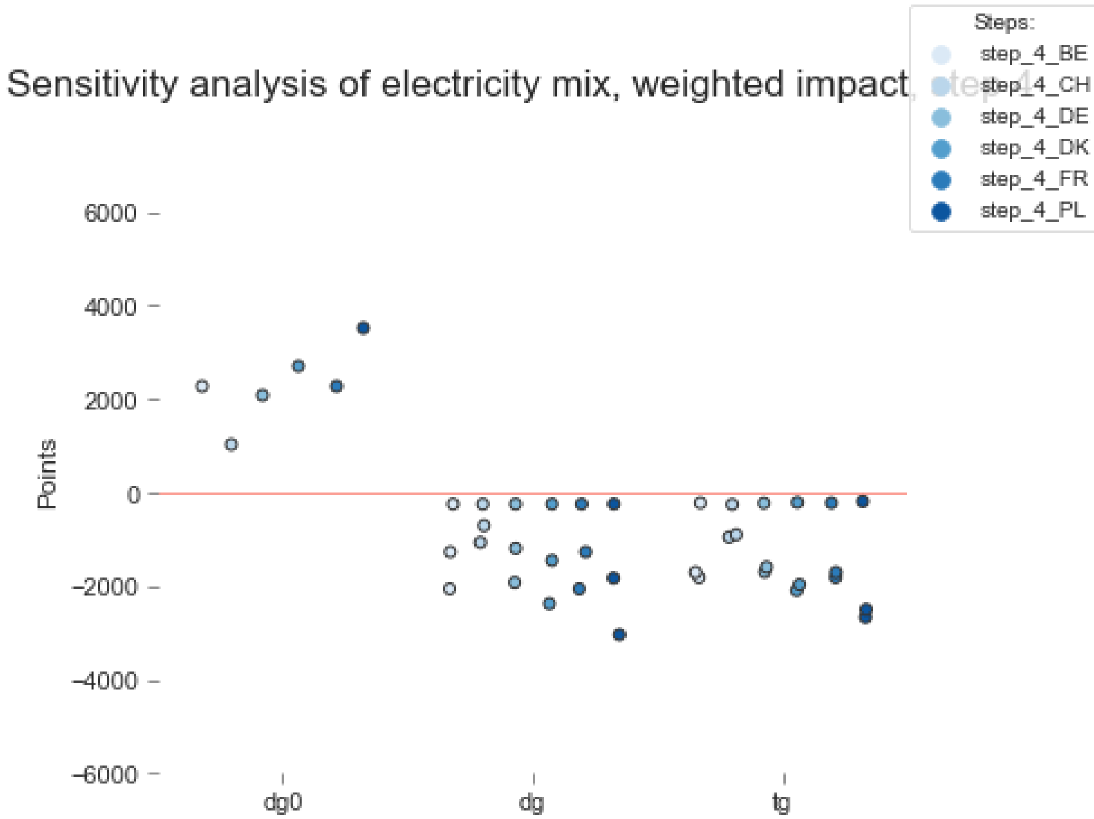
[236]:
```
df_weighted_energymix_step4 = pd.DataFrame(
    (df_energymix_results_step4.multiply(df_weighting["Weighting factor"].T,
```

```
                                                axis=1) / float(100)).sum(axis=1),
    columns=["Weighted impact"]
).T
```

Analysis of the weighted impact:

[237]:

```
fig, ax = plt.subplots(figsize=(6, 5))

df_plot = df_weighted_energymix_step4.T.reset_index()

# Category plot:
sns.stripplot(data=df_plot, x="IGU",
              y="Weighted impact",
              hue="Step", jitter=0.1, linewidth=1,
              palette="Blues", dodge=True, ax=ax
              )

style_ax(ax)
ax.axhline(y=0, c='salmon', linestyle='-', linewidth=0.75)

fig.suptitle("Sensitivity analysis of electricity mix, weighted impact, step 4",
             fontsize=17, y=1)

sns.despine(left=True, bottom=True, offset=5)

ax.set(xlabel="", ylabel="Points")
ax.set_ylim(ymin=-6000, ymax=7000)

ax.get_legend().remove()

# Add legend:
handles, labels = ax.get_legend_handles_labels()
fig.legend(handles, labels, loc='center', ncol=1,
           title='Steps:',
           bbox_to_anchor=(1, 0.94))

if export:
    # Save image:
    fig.savefig(os.path.join(path_img, 'WeightedLCIA_Elec_HVAC_VAV.png'),
              dpi=600, bbox_inches='tight')
    fig.savefig(os.path.join(path_img, 'WeightedLCIA_Elec_HVAC_VAV.pdf'),
              bbox_inches='tight')

plt.show()
```




```
[238]: df_weighted_energymix_step4["step_4_PL"]
```

```
[238]: Name             d_b_2126_dg0_vav_PL d_h_2126_dg4_vav_PL d_i_2126_dg5_vav_PL  \
       IGU                              dg0                  dg                  dg
       Weighted impact           3550.85843        -3035.278042        -1826.902472

       Name             d_j_2126_dg6_vav_PL d_n_2126_tg4_vav_PL d_o_2126_tg5_vav_PL  \
       IGU                               dg                  tg                  tg
       Weighted impact          -244.688049        -2666.331741        -2499.713167

       Name             d_p_2126_tg6_vav_PL
       IGU                               tg
       Weighted impact          -187.517298
```

13.4 Analysis with Optimised VRF System, Fully Electrified

This section conducts a LCIA according to the ILCD midpoint method, including the 16 impact indicators.

If the calculation is undertaken, be patient, it takes time!

[239]:
```
step_code = "step_6"

if calc_lcia_energymix:

    # Dropping scenarios w/ single glazing & double/triple w/ low light transm
    # Defining a new dataframe only with parameters useful for the LCIA:
    df_param = df_step6.drop(
        ["f_a_2126_dg_init_vrf", "f_c_2126_sg1_vrf", "f_d_2126_sg2_vrf",
         "f_e_2126_dg1_vrf", "f_f_2126_dg2_vrf", "f_g_2126_dg3_vrf",
         "f_k_2126_tg1_vrf", "f_l_2126_tg2_vrf", "f_m_2126_tg3_vrf"]
    ).drop(['glazing', 'heating_setpoint',
            'cooling_setpoint'], axis=1)

    df_energymix_results_step6 = pd.DataFrame()

    # Converting dataframe in a numpy array:
    val_np = df_param.to_numpy()

    # A list to save the results:
    ls_mlca_full_results = []

    for run_n, v in enumerate(val_np):

        for param_name in df_param.columns:
            loc_param = df_param.columns.get_loc(param_name)

            (ActivityParameter.update(amount=v[loc_param])
             .where(ActivityParameter.name == f'param_{param_name}'
                    ).execute()
             )

        ActivityParameter.recalculate_exchanges("cw_use_param_group")
        ActivityParameter.recalculate_exchanges("cw_eol_param_group")

        for n, m in enumerate(elec_market):
            my_act_elec = (
                Database('ecoinvent 3.7 cut-off').get(elec_market[n][1])
            )

            loc = my_act_elec['location']

            name_scenario = str(df_param.index[run_n])+"_"+loc

            # Make a copy of the activity, substitute the background process
            prod_and_use_cw_copy = prod_and_use_cw.copy()
            exc_elec = list(prod_and_use_cw_copy.exchanges())[1]
            exc_elec['input'] = m
```

```python
            exc_elec.save()

            lca = LCA({prod_and_use_cw_copy: 1})
            lca.lci()

            step = step_code+"_"+loc

            # Conducting the LCIA:
            for method in ls_method_full:
                lca.switch_method(method)
                lca.lcia()
                ls_mlca_full_results.append((step, name_scenario,
                                            method[1], method[2],
                                            lca.score,
                                            bw.methods.get(method).get('unit'))
                                            )

    # New DataFrame from list of results:
    df_energymix_results_step6 = pd.DataFrame(ls_mlca_full_results,
                                              columns=["Step",
                                                       "Name",
                                                       "Category",
                                                       "Subcategory",
                                                       "Score",
                                                       "Unit"
                                                       ]
                                              )

    # Pivot the DataFrame:
    df_energymix_results_step6 = pd.pivot_table(df_energymix_results_step6,
                                                index=["Step",
                                                       "Name"
                                                       ],
                                                columns=["Category",
                                                         "Subcategory",
                                                         "Unit"
                                                         ],
                                                values="Score"
                                                )

    # Save df_mlca_full_raw_results to csv:
    df_energymix_results_step6.unstack([0, 1]).to_csv(
        'outputs\lca\df_energymix_results_'+str(step_code)+'.csv',
        index=True)

else:
    # Open the csv file, to avoid recalculating the impacts:
```

```python
    if os.path.isfile(
            'outputs\lca\df_energymix_results_'+str(step_code)+'.csv'):
        with pd.option_context('display.precision', 10):
            df_energymix_results_step6 = (
                pd.read_csv(
                    'outputs\lca\df_energymix_results_'+str(step_code)+'.csv',
                    float_precision=None)
            )

        df_energymix_results_step6 = df_energymix_results_step6.pivot_table(
            values='0',
            index=['Step', 'Name'],
            columns=['Category', 'Subcategory', 'Unit']
        )

    else:
        print("df_mlca_full_raw_results does not exist!")
```

Reorganising the DataFrame, integrating the type of IGU for each simulation run:

[240]:
```python
# Add a row to sort by IGU type (column indexing):
ls_igu = []
for code in df_energymix_results_step6.index.get_level_values('Name'):
    if "dg_init" in code:
        ls_igu.append("dg_init")
    if "dg0" in code:
        ls_igu.append("dg0")
    if "sg" in code:
        ls_igu.append("sg")
    if (("dg1" in code) or ("dg2" in code) or ("dg3" in code)
            or ("dg4" in code) or ("dg5" in code) or ("dg6" in code)):
        ls_igu.append("dg")
    if (("tg1" in code) or ("tg2" in code) or ("tg3" in code)
            or ("tg4" in code) or ("tg5" in code) or ("tg6" in code)):
        ls_igu.append("tg")
    if "dg_vacuum" in code:
        ls_igu.append("dg_vacuum")
    if "dg_smart" in code:
        ls_igu.append("dg_smart")
    if "dsf" in code:
        ls_igu.append("dsf")
    if "ccf" in code:
        ls_igu.append("ccf")

df_energymix_results_step6.loc[:, ('IGU')] = ls_igu

df_energymix_results_step6 = (
```

```
    df_energymix_results_step6.reset_index().set_index(
        ["Step", "Name", "IGU"])
)
```

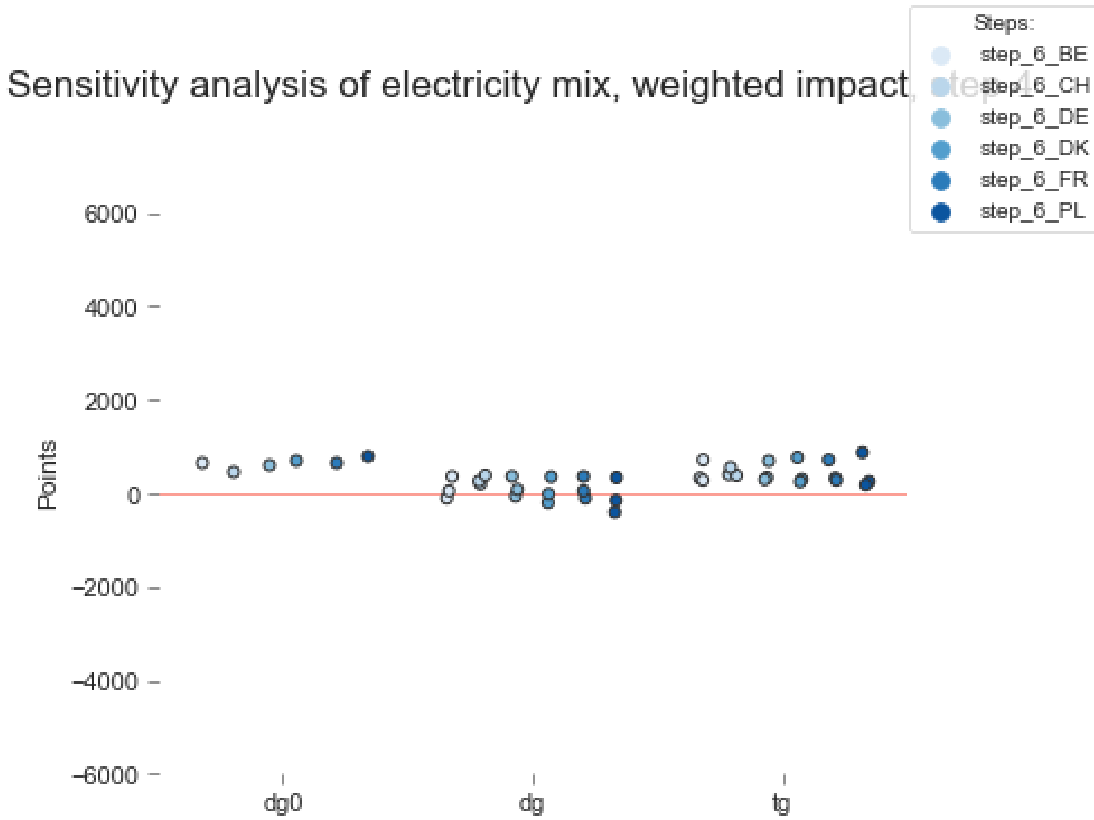
Weighting:

[241]:
```
df_weighted_energymix_step6 = pd.DataFrame(
    (df_energymix_results_step6.multiply(df_weighting["Weighting factor"].T,
                                         axis=1) / 100).sum(axis=1),
    columns=["Weighted impact"]
).T
```

Analysis of the weighted impact:

[242]:
```
fig, ax = plt.subplots(figsize=(6, 5))

df_plot = df_weighted_energymix_step6.T.reset_index()

mycolors = sns.color_palette(['lightgrey', 'firebrick', 'cornflowerblue'])

# Category plot:
sns.stripplot(data=df_plot, x="IGU",
              y="Weighted impact",
              hue="Step", jitter=0.1, linewidth=1,
              palette="Blues", dodge=True, ax=ax
              )

style_ax(ax)
ax.axhline(y=0, c='salmon', linestyle='-', linewidth=0.75)

fig.suptitle("Sensitivity analysis of electricity mix, weighted impact, step 4",
             fontsize=17, y=1)

sns.despine(left=True, bottom=True, offset=5)

ax.set(xlabel="", ylabel="Points")
ax.set_ylim(ymin=-6000, ymax=7000)

ax.get_legend().remove()

# Add legend:
handles, labels = ax.get_legend_handles_labels()
fig.legend(handles, labels, loc='center', ncol=1,
           title='Steps:',
           bbox_to_anchor=(1, 0.94))

if export:
    # Save image:
```

```
    fig.savefig(os.path.join(path_img, 'WeightedLCIA_Elec_HVAC_VRF.png'),
                dpi=600, bbox_inches='tight')
    fig.savefig(os.path.join(path_img, 'WeightedLCIA_Elec_HVAC_VRF.pdf'),
                bbox_inches='tight')

plt.show()
```



13.5 Analysis with High-Tech Façade, VAV HVAC System

This section conducts a LCIA according to the ILCD midpoint method, including the 16 impact indicators.

If the calculation is undertaken, be patient, it takes time!

[243]:
```
step_code = "step_9"

if calc_lcia_energymix:

    # Dropping scenarios w/ single glazing & double/triple w/ low light transm
    # Defining a new dataframe only with parameters useful for the LCIA:
    df_param = df_step9.drop("i_a_2126_dg_init_vav_int"
```

```
                        ).drop(['glazing',
                                'heating_setpoint',
                                'cooling_setpoint'], axis=1)

df_energymix_results_step9 = pd.DataFrame()

# Converting dataframe in a numpy array:
val_np = df_param.to_numpy()

# A list to save the results:
ls_mlca_full_results = []

for run_n, v in enumerate(val_np):

    for param_name in df_param.columns:
        loc_param = df_param.columns.get_loc(param_name)

        (ActivityParameter.update(amount=v[loc_param])
         .where(ActivityParameter.name == f'param_{param_name}'
                ).execute()
         )

    ActivityParameter.recalculate_exchanges("cw_use_param_group")
    ActivityParameter.recalculate_exchanges("cw_eol_param_group")

    for n, m in enumerate(elec_market):
        my_act_elec = (
            Database('ecoinvent 3.7 cut-off').get(elec_market[n][1])
        )

        loc = my_act_elec['location']

        name_scenario = str(df_param.index[run_n])+"_"+loc

        # Make a copy of the activity, substitute the background process
        prod_and_use_cw_copy = prod_and_use_cw.copy()
        exc_elec = list(prod_and_use_cw_copy.exchanges())[1]
        exc_elec['input'] = m
        exc_elec.save()

        lca = LCA({prod_and_use_cw_copy: 1})
        lca.lci()

        step = step_code+"_"+loc

        # Conducting the LCIA:
        for method in ls_method_full:
```

```
                lca.switch_method(method)
                lca.lcia()
                ls_mlca_full_results.append((step, name_scenario,
                                            method[1], method[2],
                                            lca.score,
                                            bw.methods.get(method).get('unit'))
                                            )

    # New DataFrame from list of results:
    df_energymix_results_step9 = pd.DataFrame(ls_mlca_full_results,
                                              columns=["Step",
                                                       "Name",
                                                       "Category",
                                                       "Subcategory",
                                                       "Score",
                                                       "Unit"
                                                       ]
                                              )

    # Pivot the DataFrame:
    df_energymix_results_step9 = pd.pivot_table(df_energymix_results_step9,
                                                index=["Step",
                                                       "Name"
                                                       ],
                                                columns=["Category",
                                                         "Subcategory",
                                                         "Unit"
                                                         ],
                                                values="Score"
                                                )

    # Save df_mlca_full_raw_results to csv:
    df_energymix_results_step9.unstack([0, 1]).to_csv(
        'outputs\lca\df_energymix_results_'+str(step_code)+'.csv',
        index=True)

else:
    # Open the csv file, to avoid recalculating the impacts:
    if os.path.isfile(
            'outputs\lca\df_energymix_results_'+str(step_code)+'.csv'):
        with pd.option_context('display.precision', 10):
            df_energymix_results_step9 = (
                pd.read_csv(
                    'outputs\lca\df_energymix_results_'+str(step_code)+'.csv',
                    float_precision=None)
            )
```

```
        df_energymix_results_step9 = df_energymix_results_step9.pivot_table(
            values='0',
            index=['Step', 'Name'],
            columns=['Category', 'Subcategory', 'Unit']
        )

    else:
        print("df_mlca_full_raw_results does not exist!")
```

Reorganising the DataFrame, integrating the type of IGU for each simulation run:

[244]:
```
# Add a row to sort by IGU type (column indexing):
ls_igu = []
for code in df_energymix_results_step9.index.get_level_values('Name'):
    if "dg_init" in code:
        ls_igu.append("dg_init")
    if "dg0" in code:
        ls_igu.append("dg0")
    if "sg" in code:
        ls_igu.append("sg")
    if (("dg1" in code) or ("dg2" in code) or ("dg3" in code)
            or ("dg4" in code) or ("dg5" in code) or ("dg6" in code)):
        ls_igu.append("dg")
    if (("tg1" in code) or ("tg2" in code) or ("tg3" in code)
            or ("tg4" in code) or ("tg5" in code) or ("tg6" in code)):
        ls_igu.append("tg")
    if "dg_vacuum" in code:
        ls_igu.append("dg_vacuum")
    if "dg_smart" in code:
        ls_igu.append("dg_smart")
    if "dsf" in code:
        ls_igu.append("dsf")
    if "ccf" in code:
        ls_igu.append("ccf")

df_energymix_results_step9.loc[:, ('IGU')] = ls_igu

df_energymix_results_step9 = (
    df_energymix_results_step9.reset_index().set_index(
        ["Step", "Name", "IGU"])
)
```

Weighting:

[245]:
```
df_energymix_results_step9 = pd.DataFrame(
    (df_energymix_results_step9.multiply(df_weighting["Weighting factor"].T,
                                         axis=1) / 100).sum(axis=1),
    columns=["Weighted impact"]
```

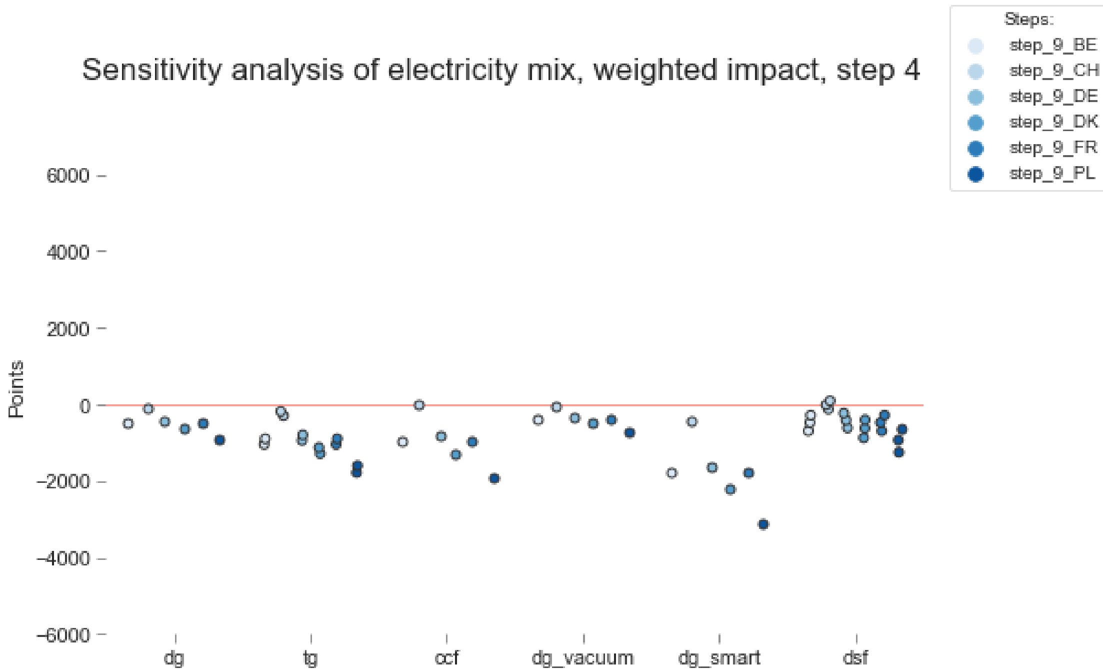
```
).T
```

Analysis of the weighted impact:

[246]:
```
fig, ax = plt.subplots(figsize=(8, 5))

df_plot = df_energymix_results_step9.T.reset_index()

mycolors = sns.color_palette(['lightgrey', 'firebrick', 'cornflowerblue'])

# Category plot:
sns.stripplot(data=df_plot, x="IGU",
              y="Weighted impact",
              hue="Step", jitter=0.1, linewidth=1,
              palette="Blues", dodge=True, ax=ax
              )

style_ax(ax)
ax.axhline(y=0, c='salmon', linestyle='-', linewidth=0.75)

fig.suptitle("Sensitivity analysis of electricity mix, weighted impact, step 4",
             fontsize=17, y=1)

sns.despine(left=True, bottom=True, offset=5)

ax.set(xlabel="", ylabel="Points")
ax.set_ylim(ymin=-6000, ymax=7000)

ax.get_legend().remove()

# Add legend:
handles, labels = ax.get_legend_handles_labels()
fig.legend(handles, labels, loc='center', ncol=1,
           title='Steps:',
           bbox_to_anchor=(1, 0.94))

if export:
    # Save image:
    fig.savefig(os.path.join(path_img, 'WeightedLCIA_Elec_hightech.png'),
              dpi=600, bbox_inches='tight')
    fig.savefig(os.path.join(path_img, 'WeightedLCIA_Elec_hightech.pdf'),
              bbox_inches='tight')

fig.show()
```

```
C:\Users\souvi\AppData\Local\Temp/ipykernel_3832/2361662638.py:40: UserWarning:
Matplotlib is currently using module://matplotlib_inline.backend_inline, which
is a non-GUI backend, so cannot show the figure.
```

```
fig.show()
```


Sensitivity analysis of electricity mix, weighted impact, step 4
Steps:
step_9_BE
step_9_CH
step_9_DE
step_9_DK
step_9_FR
step_9_PL
Points
6000
4000
2000
0
−2000
−4000
−6000
dg
tg
ccf
dg_vacuum
dg_smart
dsf