Flow programming: adding non-linear behaviour to parameterised process models for prospective Material Flow Analysis and Life Cycle Assessment

Rick Lupton[a,*]

[a] *University of Bath, Institute of Sustainability and Climate Change, Claverton Down, Bath, BA2 7AY, UK,*

Abstract

Industrial ecology modelling commonly aims to understand how the system's processes, stocks and flows would respond to changes, and the consequences for changes in environmental impacts and resource requirements. While marginal changes can be assessed with linear models, discontinuities in behaviour often mean that results for marginal changes cannot be extrapolated to larger shifts. Here, we present a new framework for introducing non-linear behaviour into process-flow models (such as in Material Flow Analysis, or for the foreground inventory of a Life Cycle Assessment), based on the idea of flexibly building up the model from a set of basic building-block operations. The approach is implemented in an open source Python library based on the Sympy computer algebra system. The resulting parameterised model can be expressed as algebraic equations, or translated into computer code to evaluate results, apply uncertainty and sensitivity analysis, or embed into a broader modelling framework. We demonstrate the approach by constructing a stock-driven model of plastics demand and petrochemical production processes, considering recycling loops, co-products and capacity limits. This new framework offers a systematic way to formulate models which respond non-linearly, which is important in understanding the impacts of, and interactions between, systemic changes in the transition to a decarbonised and more circular economy.

Article type: Methods and tools

Conflict of interest statement: The author declares no conflict of interest.

Data availability statement: Data sharing not applicable to this article as no datasets were generated or analyzed during the current study. The data and code for the examples is openly available on Github at https://github.com/ricklupton/flowprog-paper (and will be archived to Zenodo before final publication).

Keywords: industrial ecology, material flow analysis, systemic change, non-linearity

1. Introduction

A common aim of modelling systems using methods like Material Flow Analysis (MFA) (Graedel, 2019) and Life Cycle Assessment (LCA) (Hauschild et al., 2018) is to understand how the system's processes, stocks and flows would respond to changes, and the consequences for changes in environmental impacts and resource requirements. While marginal changes can be reasonably assessed with linearised models, generally the systems are not actually linear: there are discontinuities and other non-linearities in behaviour so the linear marginal changes cannot be extrapolated to larger changes (Figure 1).

Non-linear behaviour is important for understanding the effect of systemic changes like the transformation to a more circular economy (Kirchherr et al., 2017), or even in more immediate cases where limited resources

*Corresponding author

exist. For example, Pauliuk et al. (2013) developed a model for future development of the steel sector, accounting for changes in the system structure (i.e. the split between primary and secondary steelmaking is not a fixed assumption, but based on the availability of scrap). The original motivation for the work being presented in this paper was to support analysis of the interaction between supply and demand levers for decarbonising the petrochemical industry's supply of plastics (Doliente et al.), which in addition brings discontinuities in supply of primary chemicals via different feedstocks, due to capacity constraints, as well as complexities between co-production and "on-purpose" production routes. In the LCA context, although it is much less common to model the foreground system explicitly as a parameterised system of processes and flows, in principle the same issues appear there. Cases where non-linearity has been discussed as an issue within LCA include Haupt et al. (2018), who studied nonlinear effects on transport requirements and purity as plastics recycling rates are varied; Lodato et al. (2021), who discuss the need for modelling different aspects of nonlinear process responses in LCA of waste management; and Pizzol et al. (2021), who discuss non-linearity due to changes in process characteristics, and cases when newly installed capacity has a different impact to the average.

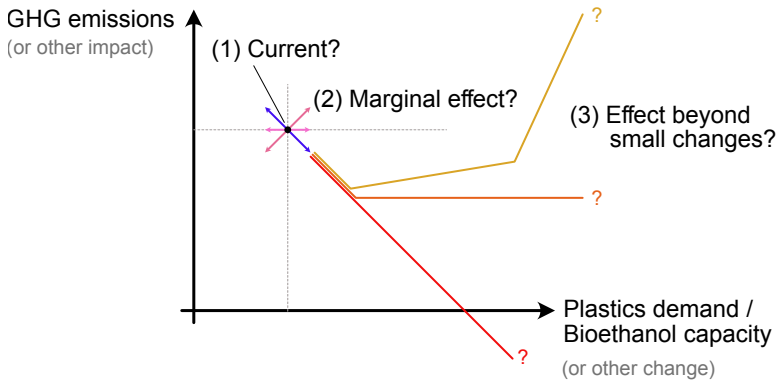


Figure 1: Illustration of the response of a system to changes. Typically, modelling is concerned with one or more of these questions: (1) what is the current state of the system and level of impact? (2) what is the effect of marginal changes? (3) what is the effect of larger (systemic) changes? Although illustrated with examples of petrochemical decarbonisation, the point applies equally to other types of impact and other types of change.

However, existing MFA and LCA process modelling frameworks are essentially linear, and so cannot be directly used for modelling systems such as these with discontinuous and non-linear behaviour. Many models are based on the solution of a well-posed system of linear equations. This includes the matrix approach to LCA inventory modelling (Heijungs and Suh, 2002), Input-Output analysis (IOA) (Leontief, 1970), and IOA-like MFA models based on transfer coefficients (Bornhöft et al., 2016). Because part of the input to these models is the market shares of the available production processes (or, equivalently, the assumed market shares or transfer coefficients in some MFA models), these do not change during the model solution process. This means that while they are certainly appropriate for assessing marginal changes in many situations, they do not tackle the problem of modelling non-linear behaviour, when the structure of the markets and system may respond to changes in inputs.

More generally, there are models based on constrained optimisation of the processes used within the system to meet some objective (typically cost or emissions minimisation), used for energy system modelling (DeCarolis et al., 2017) and broader Integrated Assessment Models (IAMs) (Stehfest et al., 2014). The approach of constrained optimisation does allow for the market shares of different technologies to change, within defined limits, which means systemic change can be modelled. However, since the models are formulated as a complicated optimisation problem, the outcomes can be highly sensitive to non-obvious interactions between parameters (Keepin and Wynne, 1984), and it does not allow for exploring non-optimal solutions, which

are likely to be where reality lies (Trutnevyte, 2016). Additional assumptions about costs are also needed, which may be inappropriate if the goal is to achieve a technical and biophysical understanding of system changes.

Amongst these, there is (to the author's knowledge) no suitable framework available for modelling in an explicit way the behaviour of process systems that respond discontinuously to changes in demand, supply, or other parameters. In this paper we therefore propose a new approach, based on the idea of flexibly building up linear or non-linear process-flow models from a set of basic building-block operations. The resulting model is fully parameterised and can be handled in the form of algebraic equations, or translated into computer code in different programming languages for use with uncertainty and sensitivity analysis, or embeded into a broader modelling framework. Specifically, we:

- Propose a "flow programming" framework and set of basic operations involved in building up a process model (Section 2), and how these are implemented in an open source Python library using a computer algebra system (Section 3).
- Demonstrate that the approach can also be used to construct a moderately complex stock-driven model of plastics demand and petrochemical production processes (Section 4).

We conclude by discussing how our approach integrates with other tools, how it compares to other related work, and by highlighting areas for further development (Section 5).

2. The "flow programming" framework

The core idea of the "flow programming" approach is to build up the mathematical structure of the final flow-process model step by step. These steps are defined by the program code, made up of the basic operations introduced below. At the start of the program, the model is empty; each step add flows and process activity to the model, and at the end, the final equations have been defined and are ready to be used. This "flow program" is run once per model [variant] to establish its structure, and then the resulting model equations may be evaluated once or (in the case of Monte Carlo uncertainty analysis, say) many thousands of times.

In the following subsections, we first introduce the conceptual structure of the system being modelled in terms of processes, flows, and objects, and the "model state" that accumulates the model equations as they are being defined step-by-step. The different basic operations used to define the state are then introduced, with simple examples to illustrate.

2.1. System structure

The system being modelled is viewed as a collection of *processes*, connected by flows of *objects* (i.e. materials or energy). This terminology follows the formal ontology developed by Germano et al. (2021), and is consistent with the structure of LCA models consisting of *activities/processes* and *products*.

Each process is assume to have a defined "recipe", which describes the quantity of input and output flows associated with a particular level of activity of the process. These recipes can be fixed ratios (as in a typical LCA model), or can be themselves parameterised functions (as in an MFA model with parameterised process yields). In a typical system, the majority of processes are assumed to contain no stock accumulation: the mass input is equal to the mass output. In some processes, however, there may be stock accumulated within the modelled time period, and therefore the input of the process must be allowed to vary independently of the output. The input and output activity of process j are denoted as:

$$X_j = \text{input activity of process } j \tag{1}$$

$$Y_j = \text{output activity of process } j \tag{2}$$

For most processes $X_j = Y_j$ (no stock accumulation), but where appropriate there is a stock accumulation $\Delta S_j = X_j - Y_j$. These input/output activities are linked to the input/output flows by the process recipe supply and use coefficients:

$$S_{ij} = \text{supply of object } i \text{ per unit output activity of process } j \tag{3}$$

$$U_{ij} = \text{use of output } i \text{ to per unit input activity process } j \tag{4}$$

The flow quantity for input/output of object i to/from process j are therefore $X_i U_{ij}$ and $Y_j S_{ij}$ respectively.

2.2. Accumulated model state

The model state is denoted by a vector $\mathbf{Z}$, which consists of the process input/output activities for the M processes in the model:

$$\mathbf{Z} = \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_M \\ Y_1 \\ Y_2 \\ \vdots \\ Y_M \end{bmatrix}$$

The model starts empty, and is added to step by step. After completing step s, the model state is denoted $\mathbf{z}^{(s)}$, where capital letters X_j, Y_j are used to refer to those quantities in general, and lower case letters refer to specific values at a given step.

To build up the model, additional activity $\hat{\mathbf{z}}^{(s)}$ is proposed to be added to the model, so that the resulting state at the end of step s is:

$$\mathbf{z}^{(s)} = \mathbf{z}^{(s-1)} + \hat{\mathbf{z}}^{(s)}$$

2.3. Basic model-building operations

The model structure above is generic; to represent a specific material flow model, the logic for that system must be described through a series of steps needed to reach the full system material flows. The basic steps used in the flow programming framework are introduced below.

2.3.1. Simplest steps: pull production and push consumption

The simplest step is a linear "demand pull" or "supply push" operation, which defines a required flow quantity at one point in the model, and propagates this either upstream or downstream to identify the additional activity from multiple processes in the model needed to achieve the specified additional supply or demand.

Figure 2 shows a very simple model for production of "Object 1" via "Process 1", which in turn requires input of two further objects.

Initially, the model is empty. Let us apply a "pull production" step for the output `Object 1`, with symbolic quantity f: this involves recursively stepping through the model to determine how much process activity is needed at each stage. The first step is to determine the process activity of Process 1, based on its supply coefficient for Object 1:

$$\hat{x}_1 = \hat{y}_1 = \frac{f}{S_{11}} \tag{5}$$

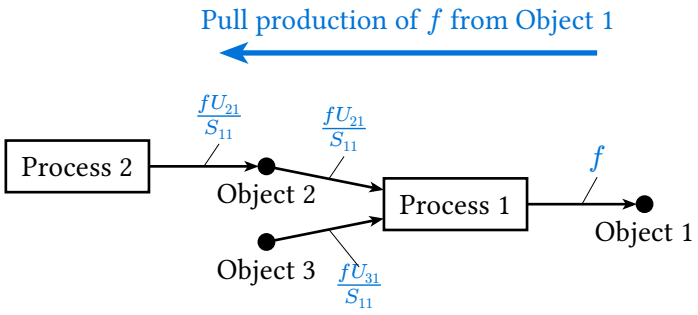


Figure 2: Symbolic flows driven by a demand of f units production of Object 1.

where S_{11} is the supply of Object 1 per unit activity of Process 1. Because we are assuming there is no stock allowed to accumulate in Process 1, the process output activity Y_1 and input activity X_1 are equal.

The framework then repeats the process based on the demand by Process 1 for Object 2 of fU_{21}/S_{11}. "Pulling production" to meet this demand results in:

$$\hat{x}_2 = \hat{y}_2 = \frac{fU_{21}}{S_{11}} \frac{1}{S_{22}} \tag{6}$$

From these process activities X_j and Y_j, the symbolic value of any flow can be calculated, as annotated in Figure 2.

The opposite of "pull production" is "push consumption", which works exactly the same way but from processes towards their outputs. Similarly, both operations can either be defined in terms of total production or consumption of an object type, or by a specific input or output flow of a specific process.

2.3.2. Market mixes

When multiple processes produce the same object, there is an ambiguity about which processes' activity should increase in response to a demand for production. Similarly, when "pushing consumption" of an object which is used by more than one process, the modeller must specify which processes' activity should increase. This behaviour is determined by "market mixes".

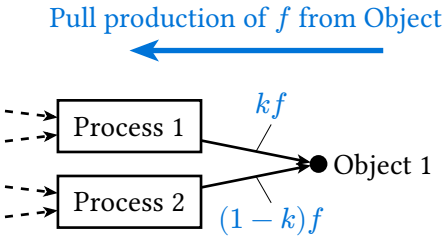


Figure 3: Market mixes: flows resulting from "pulling" production of f units of Object 1, with a supply mix of k from Process 1, $(1-k)$ from Process 2.

Figure 3 illustrates the flows resulting from "pulling" production of f units of Object 1, which can be produced by two processes. Here the market mix is defined so that Process 1 has a share of k, and Process 2 the remainder. In general these shares can be any symbolic expressions.

2.3.3. Building up models in steps

To be able to specify more complex model behaviour, the "pull production" and "push consumption" operations can be limited in scope to stop when certain objects are reached, so that the flows in different parts of the model can be built up in different ways. For example, consider that there is a limited supply of recycled material which should always be used first, with any remaining demand met from virgin material production. This might be described in three steps:

1. Calculate the flows required to produce the desired output, but only as far upstream as the recycled material object node.
2. Produce recycled material, at a quantity dictated by the available recyclate.
3. If demand has not been satisfied via recycling, now produce virgin material to make up the deficit.

Figure 4 illustrates an example system, with the flow expressions resulting from each of these three steps, with the stop boundary illustrated by a vertical line at Object 2. This allows the basic operations already introduced above to affect only part of the system, enabling this multi-step logic to be defined. Table 1 shows the state of the system after each step illustrated in Figure 4 has been completed.

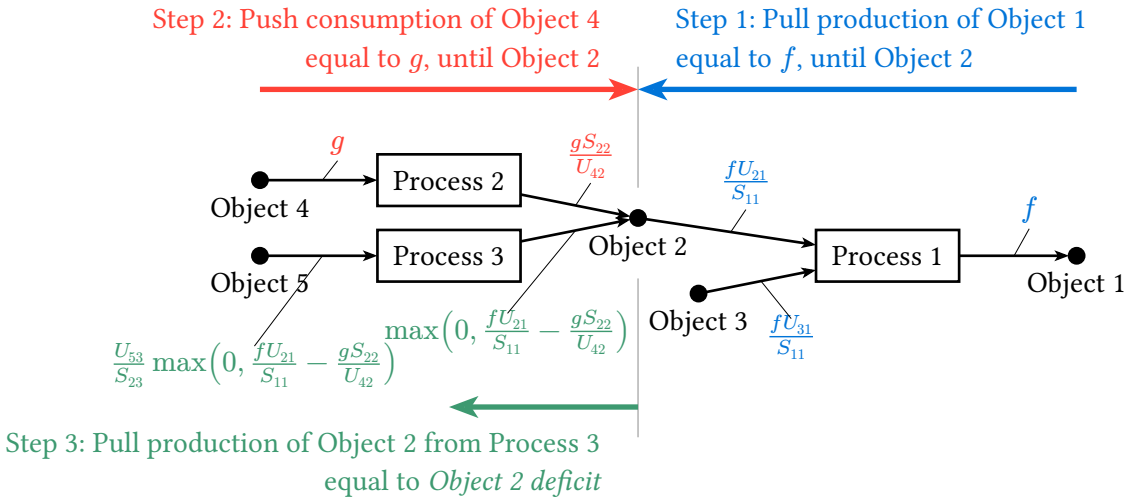


Figure 4: Symbolic flows built up in three steps. Steps 1 and 2 stop at Object 2, allowing for the discontinuous logic of limited recyclate availability followed by virgin production as required.

Table 1: System state for processes 1–3 at each step of building up the system. In this example, the processes do not have stock accumulation, and so $X_j = Y_j$ at every step (not shown). $v_2^{(2)}$ is the production deficit for Object 2, as of step 2 (see main text).

Step	Description	X_1	X_2	X_3
0	Starting state	0	0	0
1	Pull production of Object 1	f/S_{11}	0	0
2	Push consumption of Object 4	f/S_{11}	g/U_{42}	0
3	Pull production of Object 2	f/S_{11}	g/U_{42}	$v_2^{(2)}/S_{23}$

This example uses the idea of *production and consumption deficits* V_i and W_i, which can be calculated

based on the flows so far accumulated in the model:

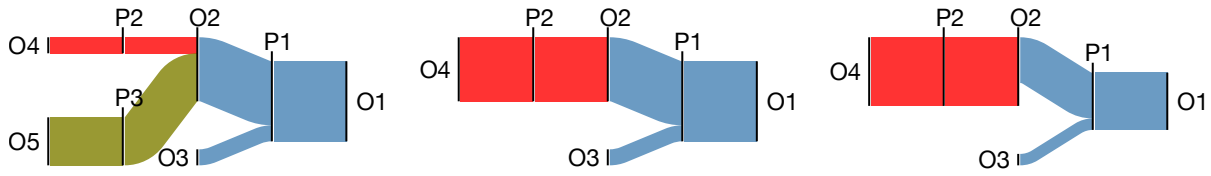
$$V_i = \max\left(0, \sum_j X_j U_{ij} - \sum_j Y_j S_{ij}\right) \tag{7}$$

$$W_i = \max\left(0, \sum_j Y_j S_{ij} - \sum_j X_j U_{ij}\right) \tag{8}$$

Specifically, the production deficit for Object 2, as of step 2 (referred to in Table 1) is

$$v_2^{(2)} = \max\left(0, \frac{fU_{21}}{S_{11}} - \frac{gS_{22}}{U_{42}}\right) \tag{9}$$

Examples of the Sankey diagrams resulting from this model for different parameter values are shown in Figure 5, illustrating how the Step 3 (green) production adjusts to meet the residual demand, after the fixed Step 2 (red) production has occurred. This also shows that there can be an excess production, if the original Step 1 (blue) demand is less than the fixed Step 2 production; this will be addressed in the following subsection.



(a) $f = 5,\ g = 1$ (b) $f = 5,\ g = 4$ (c) $f = 5,\ g = 6$

Figure 5: Sankey diagrams resulting from the system shown in Figure 4, for different parameter values.

2.3.4. Capacity (or other) limits

In some cases it is useful to modify the proposed additional flows before adding them to the model. Two examples are:

- In the example of Figure 5c, we might wish to limit the Step 2 (red) production so that it does not exceed the demand. If there is not enough demand, the recyclate (O4) should go unused, rather than producing an excess of the recycled material (O2).
- When there are multiple processes that produce the same object, rather than choosing between them based on fixed shares (as in Figure 3), we might wish to dispatch processes in order: using the first process up to some capacity limit, then using the second available process if further demand remains, and so on.

These cases can both be represented as a "limit" operation, in which some additional activity $\hat{\mathbf{z}}$ tentatively to be added to the model is modified, such that a given *expression* $\mathcal{E}$ does not exceed a given *limit* L when evaluated on the updated model. First, the original and proposed value for $\mathcal{E}$ are found as:

$$\mathcal{E}_{\text{current}} = \mathcal{E}\left(\mathbf{z}^{(s-1)}\right) \tag{10}$$

$$\mathcal{E}_{\text{proposed}} = \mathcal{E}\left(\mathbf{z}^{(s-1)} + \hat{\mathbf{z}}\right) \tag{11}$$

That is, $\mathcal{E}_{\text{current}}$ is the value that the expression takes when evaluated with the values so-far accumulated in the model, and $\mathcal{E}_{\text{proposed}}$ is the value that the expression would take after the proposed additional activity is added to the model.

The proposal is then modified as:

$$\underset{\mathcal{E},L}{limit}(\hat{z}_j) = \begin{cases} 0 & \text{if } \mathcal{E}_{\text{current}} \geq L \\ \hat{z}_j & \text{if } \mathcal{E}_{\text{proposed}} \leq L \\ \hat{z}_j \cdot \left(\frac{L - \mathcal{E}_{\text{current}}}{\mathcal{E}_{\text{proposed}} - \mathcal{E}_{\text{current}}} \right) & \text{otherwise} \end{cases} \tag{12}$$

In the first case, the limit is already exceeded, so nothing further should be added. In the second, the value for $\mathcal{E}$ after adding the additional activity is still less than the limit, so no change is needed. Otherwise, the proposed additional activity is scaled down to just reach the limiting value.

Returning to the examples which introduced this subsection, these can be represented by the following limits:

- In the example of Figure 5c, the Step 2 (red) production can be limited to avoid exceeding the demand from Step 1 with:

$$\mathcal{E} = Y_2 S_{22} \tag{13}$$
$$L = x_1^{(1)} U_{21} \tag{14}$$

  where $x_1^{(1)}$ is the value of the activity of process 1 found at the end of step 1. With this modification, the imbalance in Figure 5c no longer occurs (the results are the same as Figure 5b for all $g \geq 4$)

- To dispatch processes in order, limited by their capacity, the limiting expression for the output of object i from process j would be:

$$\mathcal{E} = Y_j S_{ij} \tag{15}$$
$$L = C_j^{\max} \tag{16}$$

  where $C_j^{\max}$ is the maximum capacity for production of object i from process j. Each available process can be proposed in turn, with its own capacity limit, and the last one added unlimited.

2.4. Working with the complete model

Once the model logic has been built up step by step, the model state represents a finished algebraic equation for each process activity (and hence each flow), in terms of the process recipe data and the parameters introduced during the logic program. The scope of the flowprog framework ends with describing a system of processes and flows, but further custom calculations can easily be layered on top of the flow equations using custom symbolic expressions (e.g. to calculate environmental impacts or circularity metrics). For example, a direct process emissions factor E_j can be defined for each process j, and the direct emissions from the system calculated as:

$$E_{\text{direct}} = \sum_j Y_j E_j$$

These equations can be automatically compiled to code in Python or another language, to efficiently evaluate the flows corresponding to any given parameter values. As well as directly using the model to evaluate flows, many other Python-based tools are also available for use, and some of these are discussed later.

3. Implementation: the `flowprog` package

The flow-programming modelling framework described above has been implemented as a Python package `flowprog`, building on the Sympy computer algebra system (Meurer et al., 2017). Using this package, the logical steps to define the system are expressed directly as Python code.

For example, the steps listed in Table 1 (Section 2.3.3) are implemented using this Python code:

```
import sympy as sy

# Define the parameters as Sympy symbolic expressions
demand = sy.Symbol("f")
supply_recyclate = sy.Symbol("g")

# Set up the model structure by defining "O1" (Object 1),
# "P3" (Process 3) etc -- details skipped here,
# see accompanying code.
model = define_model()

# Step 1
model.add(
    model.pull_production(
        "O1",
        demand,
        until_objects=["O2"]
    ),
    label="Add demand for O1"
)

# Step 2
model.add(
    model.push_consumption(
        "O4",
        supply_recyclate,
        until_objects=["O2"]
    ),
    label="Add supply of O4 recyclate"
)

# Step 3
model.add(
    model.pull_production(
        "O2",
        model.object_production_deficit("O2"),
        allocate_backwards={"O2": {"P3": 1}}
    ),
    label="Production deficit from P3"
)
```

After running this code, the final model state (last row of Table 1) is accessible as symbolic Sympy expressions, which can be converted to Python (or another supported language) code to evaluate the model numerically with specific parameter values, printed as human-readable mathematical notation, or further manipulated symbolically.

The definitions of the objects, processes, and process recipes which make up the available elements of the model can be defined directly via custom Python code. An optional integration with the PRObs ontology (Germano et al., 2021) also allows processes and objects to be defined as RDF data and imported directly.

More details and examples of the Python interface are available with the `flowprog` package documentation[1] and in the code accompanying this paper[2].

4. Case study: petrochemical production system

To illustrate a more realistic and complete use of the flowprog system, this section describes a simplified version of a non-linear model for the operation of the global petrochemical production system, which was developed for another study using `flowprog`. To keep the focus on the modelling approach, the actual system and data have been simplified to reduce the number of processes and objects, and simplify the data – but major elements of the modelling logic (which require the non-linear flowprog framework, rather than a simpler matrix-based model) are preserved.

This example demonstrates in particular the following modelling capabilities:

- *Flows driven by a dynamic stock model*, which needs mixed supply-push / demand-pull logic in different parts of the system (Section 4.1); and
- *Dispatching production* to meet demand to different processes in order of preference, based on the processes' capacity limits, and then *"using up" by-products* as far as possible (supply push), with any remaining demand supplied by "on-purpose production processes" (demand pull) (Section 4.2).

4.1. Stock-driven flows

In the first part of the model, changes in the in-use stock (represented here by two product types, packaging and transport equipment) are assumed to drive both demand for new material, and supply of end-of-life material. The dynamic stock model calculations themselves not included in this example: model parameters specifying additions to stock and end-of-life flows are assumed to be set directly. Each product type has a fixed recipe (mix of polymer types needed to produce it). Here, end-of-life products are assumed to be separated back into their constituent polymers, and then a set of recycling rate parameters determine how much of each waste polymer is mechanically recycled, with the remainder going to incineration. The symbolic parameters used as inputs to the model are listed in Table 2.

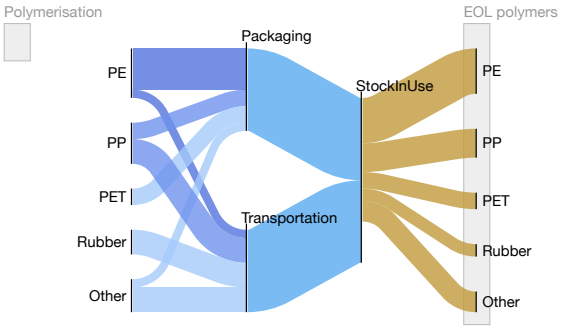
Table 2: Parameters used to control the flows in the stock model. The arbitrary values are chosen to illustrate the model in Figure 6.

Parameter	Description	Value
Z_i^{product}	Addition to stock of product i	[2, 2]
Z_i^{EOL}	End-of-life flow leaving stock of product i	[2, 1]
RR_j	Recycling rate for polymer type j	[0.6, 0.3, 0.6]

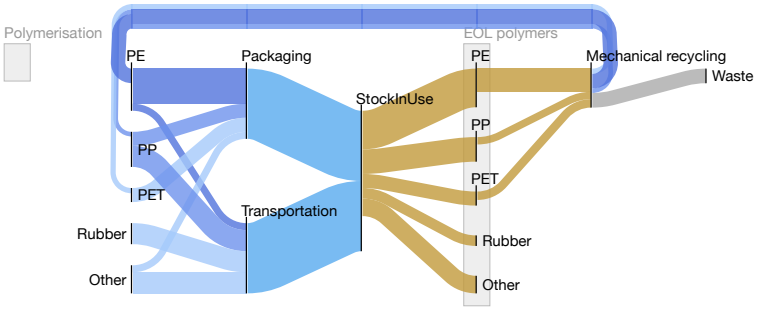
Note that this is just one approach, and the framework could be used to set up alternative models, such as where the recycling rate is dependent on the product as well as the polymer type; where recycling is determined by a fixed capacity limit rather than a fixed percentage recycling rate; or where different age cohorts have different compositions.

Figure 6 shows how the flows are built up step-by-step, starting from the assumed flows in and out of the in-use stock, through mechanical recycling, and then via polymerisation and organic chemical synthesis processes with inputs of primary chemicals. Details of the processes and their recipes in this model are available in the Supplementary Information.

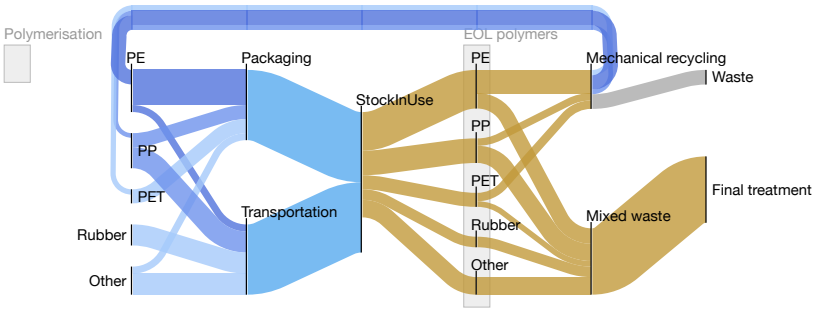
[1] https://github.com/probs-lab/flowprog
[2] https://github.com/ricklupton/flowprog-paper



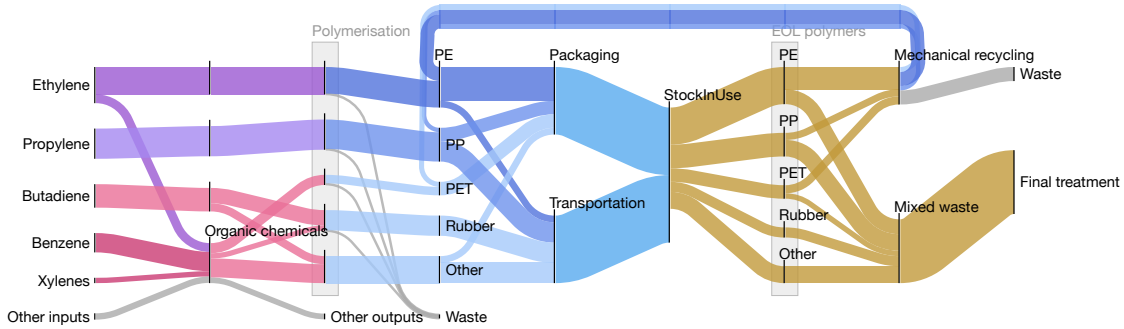
(a) Step 1: additions to stock are "pulled" through the model as far as the 5 polymer types, and end-of-life flows are "pushed" downstream as far as the 5 separated polymer types.



(b) Step 2: mechanical recycling rates are applied to the relevant polymer types (here 60% for polyethylene (PE), 30% for polypropylene (PP), and 60% for PET).



(c) Step 3: any remaining excess supply of waste polymers after Step 2 are sent to mixing and final treatment.



(d) Step 4: now, any remaining excess demand for polymers (which has not already been met by recycling) is "pulled" through the polymerisation and organic chemical synthesis processes until primary chemicals are reached.

Figure 6: Step-by-step illustration of building up the material flows driven by the example stock model.

4.2. Choice of limited-capacity production processes

Next, we define the operation of the model which supplies the primary chemicals (ethylene, propylene, etc) which appear at the left-hand side of Figure 6d. This model has two main features, illustrated in Figure 7 and Figure 8: deployment of a series of limited-capacity processes, and the presence of various co-products which do not initially match the desired mix of outputs.

Table 3: Parameters used to control the flows in the stock model. The arbitrary values are chosen to illustrate the model in Figure 7 and Figure 8.

Parameter	Description	Value
$C_i^{\text{bioethanol}}$	Capacity for bioethanol production	0.5
k^{ethane}	Fraction of steam-cracking ethylene from ethane feedstock	0.5

The aim is to create a model which behaves according to a certain logic, deploying different production processes in a sensible order based on varying demand. In the example, there is a limited capacity to produce ethylene from bioethanol: let us assume this is a lower-impact process and should be used first if available, but is unlikely to fully satisfy demand for ethylene (Figure 7a). In that case, a fall-back process (steam cracking of fossil ethane and naphtha) should then be deployed to supply the remaining demand (Figure 7b). Note that this means that the market shares of the two processes are not a fixed parameter to be specified, but will naturally alter as the scale of the demand varies relative to the process capacity; an alternative model could however be defined with fixed proportions of ethylene demand being supplied by each process.

The steam cracking process produces some co-products which are not required, but can be converted into products that are needed (Figure 8a to Figure 8c). This should be attempted first, after which any remaining demand can be supplied by specific "on-purpose" production processes for those chemicals.

By defining the model logic in this step-by-step way, the model can respond non-linearly to changes in the demand mix. For example, changes in the stock model or recycling rates might substantially reduce the demand for PP polymer relative to PE, which would cause a large shift in the market mix of PP production processes.

4.3. Final model behaviour

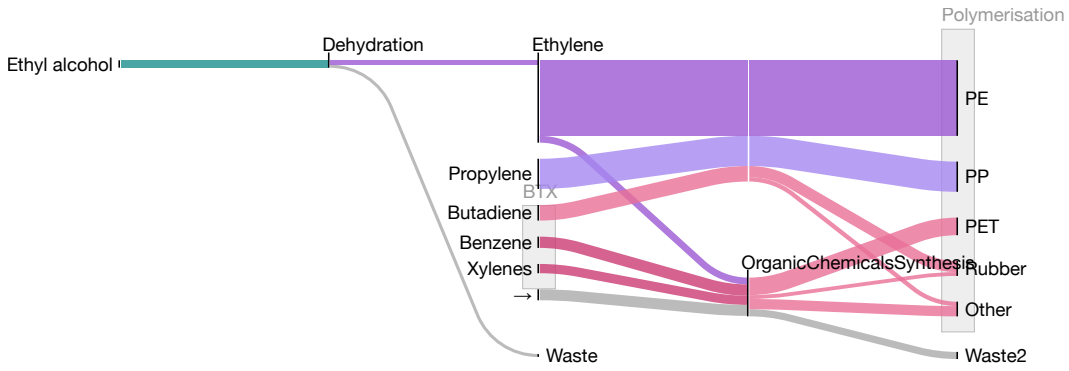
Using the logic introduced above, the model is able to maintain plausible patterns of supply and demand mass flows across a very wide range of scenarios. For example, high levels of plastics demand reduction combined with high levels of bioethanol-to-ethylene capacity can result in the need for fossil-based steam cracking for ethylene to entirely disappear, but due to the lost co-production of propylene and BTX from steam cracking, entirely new alternative processes need to be introduced into the model to meet demand for all primary chemicals. Thus, the mix of processes used to supply each 'market' responds dynamically and discontinuously to the other parts of the model, in a way that a linear model with fixed capacities or market shares would be unable to capture (Figure 9).

An expanded version of this model is used in a forthcoming analysis of the petrochemical industry. This includes a wider range of practically-important processes and realistic data estimates (not included here for simplicity). Coupled with estimates of parameter values today and in a range of future scenarios, the model can assess the effect of (and interactions between) levers on system-wide GHG emissions across a large solution space.

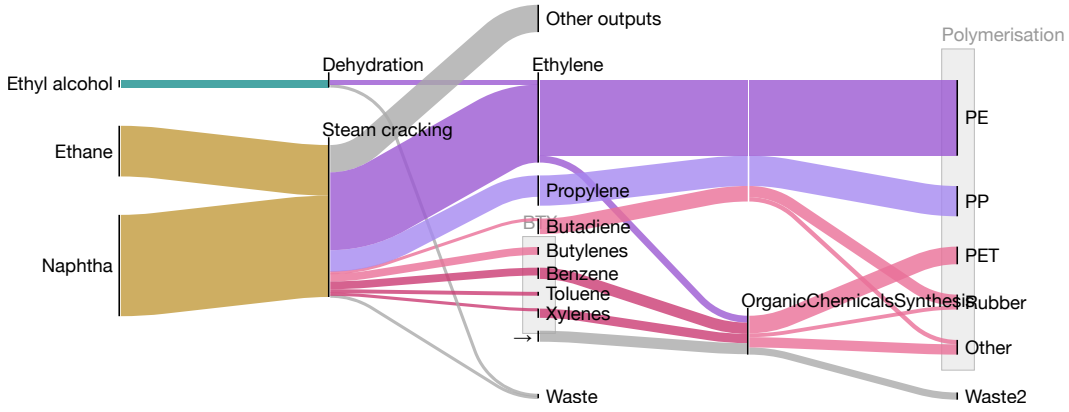
5. Discussion

5.1. Integration with other modelling tools

Since the result of using the flowprog framework to define a model is just a set of algebraic equations describing the flows and processes in the model as a function of the parameters (or, equivalently, a Python

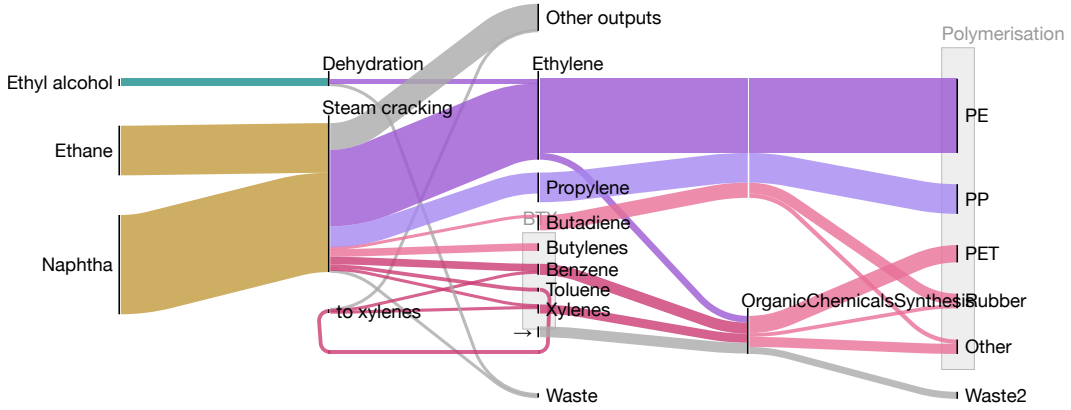


(a) Step 5: Starting from the demand for the primary chemicals (ethylene, etc) established from Step 4 of Figure 6, supply the demand for ethylene from bioethanol (ethyl alcohol), but only up to the capacity set by $C_{\text{bioethanol}}$.

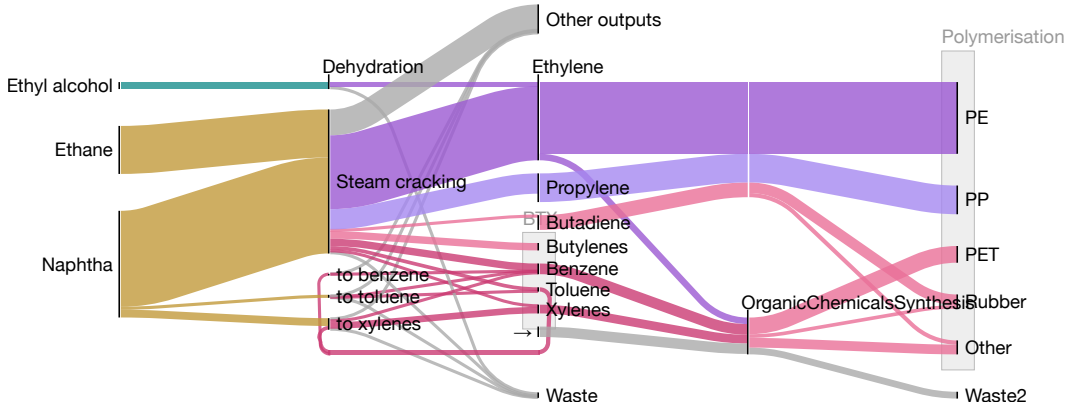


(b) Step 6: Supply any remaining demand for ethylene from steam cracking of fossil naphtha and ethane. This also creates outputs of co-products, some of which will be used in subsequent steps.

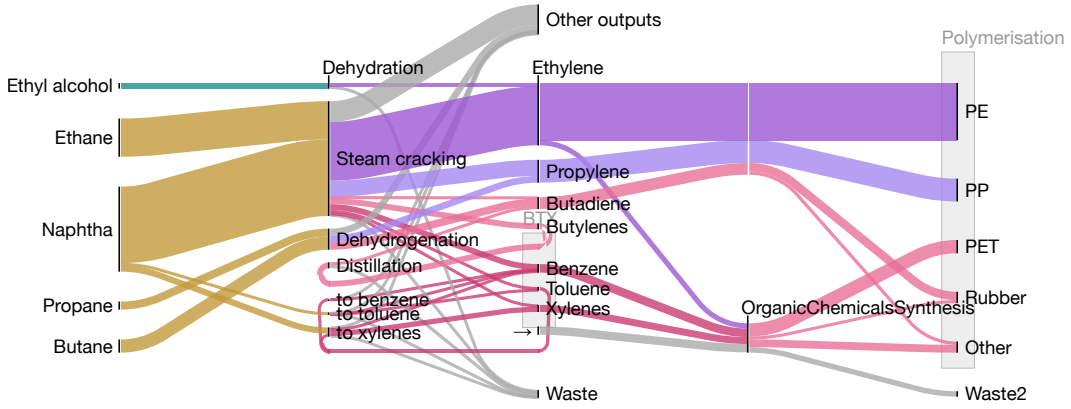
Figure 7: Step-by-step illustration of building up the material flows for production of primary chemicals (i.e. the part of the system upstream of Figure 6). Primary production model response illustrated for a demand of PE = 5, PP = 2, PET = Rubber = Other = 1.



(a) Step 7: If there is insufficient supply of xylenes from Step 5, "pull" the excess demand for xylenes from the toluene conversion process, but only up to the limit of the excess toluene available.



(b) Step 8: If there is still insufficient supply of xylenes from Step 6, "pull" the excess demand for xylenes from an "on-purpose" process converting from fossil naphtha.



(c) Step 9: If there is still insufficient supply of propylene, "pull" the excess demand from a conversion process from fossil propane. If there is still insufficient supply of butadiene, convert it from butylenes (as far as there is excess supply from Step 7) and then if still needed from fossil butane.

Figure 8: Continued from Figure 7.

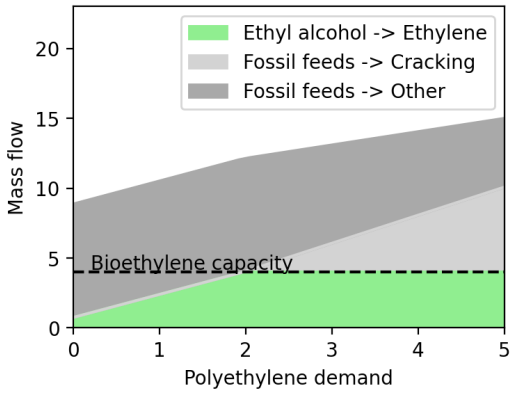


Figure 9: Change in model structure as demand for polyethylene varies, showing discontinuities due to the capacity limit on bioethylene. The additional fossil feedstocks required for the other conversion processes is reduced once steam cracking starts being used to produce ethylene, due to the co-products.

function which evaluates the equations for particular values of the parameters), it is straightforward and powerful to integrate such models with many of the modelling and visualisation tools available in the Python modelling ecosystem.

Uncertainty analysis can be carried out straightforwardly using Monte Carlo simulations by sampling parameter values and evaluating the model for each. This approach preserves the non-linear behaviour of the model, and makes sure that each individual sample is logically consistent (e.g. preserving mass balances). Contrast this with the common method of uncertainty analysis in LCA, where individual flow values are sampled independently of one another, which does not generally preserve mass balances (Marsh et al., 2024). If appropriate, the model can also be linearised at a specified operating point, and uncertainty analysis performed analytically (Gauessian error propagation). Various global sensitivity analysis algorithms are available in the SALib Python package (Herman and Usher, 2017; Iwanaga et al., 2022) which can be directly applied to a model constructed using flowprog.

Visualising the flows defined by a model, such as via Sankey diagrams, is very useful for understanding its behaviour and communicating results. All the Sankey diagrams in this paper were created using the floweaver Python package (Lupton and Allwood, 2017). Floweaver uses a two-step approach to Sankey diagram creation, where first a reusable "Sankey diagram definition" is created, which describes the desired structure and aggregation of the diagram. The Sankey diagram definition can then be repeatedly applied to visualise many different realisations of the same model structure. This works well with flowprog, since after defining the Sankey diagram definition once, many Sankey diagrams can be easily created showing the behaviour of the model with different parameter values.

The flowprog framework could also be effectively used to define non-linear foreground inventory models for LCA, which could be linked to Brightway (Mutel, 2017) to perform LCA calculations. To do this, two areas of linking would be needed:

1. *Technosphere flows across the system boundary:* any object production deficits in the flowprog model (i.e. material used but not produced in the foreground system) would become a technosphere exchange in the Brightway inventory, linking to the relevant background database product. Any object consumption deficits representing wastes requiring treatment can be linked similarly.
2. *Process biosphere flows*: If processes in the foreground system (i.e. modelled within the flowprog framework) have direct exchanges with the environment, these can be included in the Brightway inventory based on the process activities X_j or Y_j.

5.2. Related work

There are some similarities between the flowprog framework presented here, and the "lca_algebraic" approach (Jolivet et al., 2021). lca_algebraic also uses the Sympy computer algebra system to define a parameterised process-based model. It does not however share the approach presented here of building up a model in logical steps, which enables the definition of non-linear models with limited capacity processes, and flexible logic for recycling and use of co-products. Instead it focuses on linear process-based LCA systems. It provides the machinery needed to translate between the symbolic Sympy world and the world of Brightway LCA databases, which could be valuable in linking flowprog foreground models with LCA calculations as discussed above.

The ODYM framework (Pauliuk and Heeren, 2020) provides a flexible structure for modelling processes, stocks and flows over multiple regions, time periods and products. Its scope does not extend to formulating the model equations themselves, but instead focuses on organising the calculations and datasets across these different dimensions. It is therefore complementary to the flowprog framework: the generic model equations defined using flowprog could be embedded with the ODYM structure to model analogous sets of processes and flows across different regions and time periods, for example.

More generally, since flowprog reinterprets the definition of an MFA/LCA process model as a programmatic sequence of steps, many general-purpose programming approaches become relevant. For example, to compare the results of two slightly different versions of a model, rather than duplicating and editing the model definition directly, the shared parts can be extracted into Python functions which can be called repeatedly in different context. Code can be organised into Python modules, repetitive definitions expressed as Python loops, and so on.

5.3. Limitations and future work

Non-linearity, being defined only in terms of what it is not, includes many types of behaviour of which this paper focuses on a few important cases in particular. There could be other types of non-linear behaviour which would be useful to model but cannot be easily represented with the basic operations presented here. Pizzol et al. (2021) discuss several aspects of non-linearity in the context of LCA models, which can be interpreted from the perspective of the flowprog framework to a large extent. One aspect relates to changes in process characteristics themselves (i.e. the process recipes S_{ij} and U_{ij}), such as when the efficiency of a transformation process increases over time. In the cases presented in this current paper, we have assumed process recipes to be constant, but this assumption could be relaxed in future to account for evolving processes. The current implementation of the flowprog Python library already allows recipes to be general symbolic expressions. Alternatively, the more-efficient version of the process can be modelled as an alternative process and a parameter used to interpolate between the two fixed recipes, which achieves a similar result in simple cases.

Another example discussed by the same authors is about the impact of Bitcoin "mining", which depends on the location and type of current and additional mining equipment capacity. This could be modelled within the current flowprog framework by defining the existing equipment and new equipment as different processes, with demand being met first using existing capacity, and then falling back on new capacity if needed (analogous to the bioethanol in Section 4.2).

While the framework is already functional and useful, there is potential to provide more useful features to support modellers in developing their models. Firstly, due to the way the model is built up step by step in code, there is a detailed log of the model logic to explain the modeller's assumptions and intent. While this log is there, it is not currently exploited, and in future it may be possible to automatically produce model documentation "for free" based on the same code that the modeller is already writing. Secondly, the step-by-step illustrations provided in Figure 6 and Figure 7 were hopefully useful to the reader in understanding the way the model is defined and behaves. For this paper, these illustrations were produced in practice by defining not one but a series of models, each one including all the previous steps plus one more. However, given that each step is already being logged by the flowprog framework as it is defined, it should be possible

to make a "time travelling debugger" for flowprog models. This is a technique already applied in some programming languages, and would enable the modeller to travel forwards and backwards through the steps of building up the model, while simultaneously varying parameter values and seeing the effect. This may prove a useful tool for developing and testing complex models.

6. Conclusions

In this paper we have presented *flow programming*, as a novel framework for building up non-linear process-flow models step by step, based on a new set of basic "building block" model elements, which can be combined to produce complex models. The result is a set of parameterised algebraic equations which can be further manipulated (e.g. to calculate greenhouse gas emissions from the modelled processes) or compiled to code and evaluated numerically for given parameter values. The framework is implemented in an open source Python library `flowprog`, using the Sympy computer algebra system.

The basic model-building operations in the flowprog framework were presented with small illustrative examples, and then a simplified model of the petrochemical/plastics system was used to show how the framework can be used to represent flows driven by a dynamic stock model, to dispatch production processes to meet demand subject to capacity limits, and to flexibly handle mixed supply-driven and demand-driven logic for co-products.

The significance of this new flow programming framework is that it provides a systematic way to formulate models whose response to changes is non-linear or discontinuous, such as due to interactions between demand, recycling and primary production, or limited availability of production capacity as demand varies. This allows exploration of the impacts of and interactions between systemic changes in the transition to a decarbonised and more circular economy. The structured approach of the framework facilitates wider modelling best practices such as uncertainty and sensitivity analysis, and visualisation of model behaviour.

7. Acknowledgements

The author would like to acknowledge funding for the research project "C-THRU: Carbon clarity in the global petrochemical supply chain project" (www.c-thru.org), which supported this work, and the input of those involved in the project's petrochemical model development which informed the example system presented in this paper.

8. Supporting information

Supporting information is provided as a PDF document listing all the processes and objects used in the petrochemical example system (Section 4), and the recipe data assumed for the processes.

The Python code defining the model logic, corresponding to the steps presented in Figure 6 and Figure 7, is available at https://github.com/ricklupton/flowprog-paper

References

Bornhöft, N.A., Sun, T.Y., Hilty, L.M., Nowack, B., 2016. A dynamic probabilistic material flow modeling method. Environmental Modelling & Software 76, 69–80. doi:10.1016/j.envsoft.2015.11.012.

DeCarolis, J., Daly, H., Dodds, P., Keppo, I., Li, F., McDowall, W., Pye, S., Strachan, N., Trutnevyte, E., Usher, W., Winning, M., Yeh, S., Zeyringer, M., 2017. Formalizing best practice for energy system optimization modelling. Applied Energy 194, 184–198. doi:10.1016/j.apenergy.2017.03.001.

Doliente, S.S., Lupton, R., Cullen, L., Gao, Y., Jin, E., Meng, F., Masanet, E., Cabrera Serrenho, A., Cullen, J.M., . Exploring the technically-plausible solution space for greenhouse gas emissions reduction in the global petrochemical industry. under review .

Germano, S., Saunders, C., Horrocks, I., Lupton, R., 2021. Use of Semantic Technologies to Inform Progress Toward Zero-Carbon Economy. The Semantic Web – ISWC 2021 , 665–681doi:10.1007/978-3-030-88361-4_39.

Graedel, T.E., 2019. Material Flow Analysis from Origin to Evolution. Environ. Sci. Technol. 53, 12188–12196. doi:10.1021/acs.est.9b03413.
Haupt, M., Waser, E., Würmli, J., Hellweg, S., 2018. Is there an environmentally optimal separate collection rate? Waste Management 77, 220–224. doi:10.1016/j.wasman.2018.03.050.
Hauschild, M.Z., Rosenbaum, R.K., Olsen, S.I. (Eds.), 2018. Life Cycle Assessment. Springer International Publishing, Cham. doi:10.1007/978-3-319-56475-3.
Heijungs, R., Suh, S., 2002. The Computational Structure of Life Cycle Assessment. Springer Netherlands. doi:10.1007/978-94-015-9900-9.
Herman, J., Usher, W., 2017. SALib: An open-source python library for sensitivity analysis. The Journal of Open Source Software 2. URL: https://doi.org/10.21105/joss.00097, doi:10.21105/joss.00097.
Iwanaga, T., Usher, W., Herman, J., 2022. Toward SALib 2.0: Advancing the accessibility and interpretability of global sensitivity analyses. Socio-Environmental Systems Modelling 4, 18155. URL: https://sesmo.org/article/view/18155, doi:10.18174/sesmo.18155.
Jolivet, R., Clavreul, J., Brière, R., Besseau, R., Prieur Vernat, A., Sauze, M., Blanc, I., Douziech, M., Pérez-López, P., 2021. lca_algebraic: a library bringing symbolic calculus to lca for comprehensive sensitivity analysis. The International Journal of Life Cycle Assessment 26, 2457–2471. doi:10.1007/s11367-021-01993-z.
Keepin, B., Wynne, B., 1984. Technical analysis of iiasa energy scenarios. Nature 312, 691–695. doi:10.1038/312691a0.
Kirchherr, J., Reike, D., Hekkert, M., 2017. Conceptualizing the circular economy: An analysis of 114 definitions. Resources, Conservation and Recycling 127, 221–232. doi:10.1016/j.resconrec.2017.09.005.
Leontief, W., 1970. Environmental repercussions and the economic structure: An input-output approach. The Review of Economics and Statistics 52, 262. doi:10.2307/1926294.
Lodato, C., Zarrin, B., Damgaard, A., Baumeister, H., Astrup, T.F., 2021. Process-oriented life cycle assessment modelling in easetech. Waste Management 127, 168–178. doi:10.1016/j.wasman.2021.04.026.
Lupton, R., Allwood, J., 2017. Hybrid Sankey diagrams: Visual analysis of multidimensional data for understanding resource use. Resources, Conservation and Recycling 124, 141–151. doi:10.1016/j.resconrec.2017.05.002.
Marsh, E., Hattam, L., Allen, S., 2024. Stochastic error propagation with independent probability distributions in lca does not preserve mass balances and leads to unusable product compositions—a first quantification. The International Journal of Life Cycle Assessment 30, 221–234. doi:10.1007/s11367-024-02380-0.
Meurer, A., Smith, C.P., Paprocki, M., Čertík, O., Kirpichev, S.B., Rocklin, M., Kumar, A., Ivanov, S., Moore, J.K., Singh, S., Rathnayake, T., Vig, S., Granger, B.E., Muller, R.P., Bonazzi, F., Gupta, H., Vats, S., Johansson, F., Pedregosa, F., Curry, M.J., Terrel, A.R., Roučka, v., Saboo, A., Fernando, I., Kulal, S., Cimrman, R., Scopatz, A., 2017. Sympy: symbolic computing in python. PeerJ Computer Science 3, e103. URL: https://doi.org/10.7717/peerj-cs.103, doi:10.7717/peerj-cs.103.
Mutel, C., 2017. Brightway: An open source framework for life cycle assessment. Journal of Open Source Software 2, 236. URL: https://doi.org/10.21105/joss.00236, doi:10.21105/joss.00236.
Pauliuk, S., Heeren, N., 2020. ODYM — An open software framework for studying dynamic material systems: Principles, implementation, and data structures. Journal of Industrial Ecology 24, 446–458. doi:10.1111/jiec.12952.
Pauliuk, S., Milford, R.L., Müller, D.B., Allwood, J.M., 2013. The Steel Scrap Age. Environ. Sci. Technol. 47, 3448–3454. doi:10.1021/es303149z.
Pizzol, M., Sacchi, R., Köhler, S., Anderson Erjavec, A., 2021. Non-linearity in the life cycle assessment of scalable and emerging technologies. Frontiers in Sustainability 1. doi:10.3389/frsus.2020.611593.
Stehfest, E., van Vuuren, D., Kram, T., Bouwman, L., 2014. Integrated Assessment of Global Environmental Change with IMAGE 3.0: Model description and policy applications. Technical Report. PBL Netherland Environmental Assessment Agency.
Trutnevyte, E., 2016. Does cost optimization approximate the real-world energy transition? Energy 106, 182–193. doi:10.1016/j.energy.2016.03.038.