**Preprint**

# Analysing and simulating energy-based models in biology using BondGraphTools

**Peter Cudmore**[1,2], **Michael Pan**[1,2], **Peter J. Gawthrop**[1], **Edmund J. Crampin**[1,2,3,a]

[1]Systems Biology Laboratory, School of Mathematics and Statistics, and Department of Biomedical Engineering, University of Melbourne, Parkville, Victoria 3010
[2]ARC Centre of Excellence in Convergent Bio-Nano Science and Technology, Faculty of Engineering and Information Technology, University of Melbourne, Parkville, Victoria 3010
[3]School of Medicine, Faculty of Medicine, Dentistry and Health Sciences, University of Melbourne, Parkville, Victoria 3010

Received: date / Accepted: date

**Abstract** Like all physical systems, biological systems are constrained by the laws of physics. However, mathematical models of biochemistry frequently neglect the conservation of energy, leading to unrealistic behaviour. Energy-based models that are consistent with conservation of mass, charge and energy have the potential to aid the understanding of complex interactions between biological components, and are becoming easier to develop with recent advances in experimental measurements and databases. In this paper, we motivate the use of bond graphs (a modelling tool from engineering) for energy-based modelling and introduce, `BondGraphTools`, a Python library for constructing and analysing bond graph models. We use examples from biochemistry to illustrate how `BondGraphTools` can be used to automate model construction in systems biology while maintaining consistency with the laws of physics.

## 1 Introduction

Biological systems are highly complex, composed of numerous interacting components. Systems biology aims to explain how complex phenomena arise from interactions between these components [1]. While mathematical models have been long been used to understand biology, efforts to integrate these models together have gathered momentum since the turn of the century, together with the availability of omics data. Examples of this approach include whole-cell models that simulate the evolution of each biomolecule through time, [2–4] through to multi-scale physiological models [5, 6].

Like any physical system, interactions between biological components are governed by the laws of physics; notably conservation of mass, charge and energy. Many existing approaches in the field of physiology take advantage of such conservation laws as a means of coupling models together. This has been used to great success in using conservation of mass and charge to couple models of electrophysiology [7, 8] and coupling boundary conditions in models of fluid flow [9].

[a]e-mail: edmund.crampin@unimelb.edu.au (corresponding author)

Energy is fundamental to biological systems, as it is to physical systems. In his book *What is Life?*, the physicist Erwin Schrodinger wrote that living systems are in a constant fight against a decay to thermodynamic equilibrium, and must feed on free energy from the environment to avoid this eventual approach to equilibrium [10]. *Energy-based models* that account for the conservation of mass, charge and energy have the potential to gain a deeper understanding of the forces driving biological networks. However, in the field of systems biology, energy conservation is commonly ignored, and many models are thermodynamically inconsistent. As discussed by Soh and Hatzimanikatis [11], difficulties in measuring the thermodynamic parameters have slowed the uptake of energy-based models. However, recent developments in thermodynamic databases such as eQuilibrator [12] have made constructing thermodynamically detailed models more tractable.

In addition to developments in data availability, there have also been recent advancements in methodologies for constructing energy-based models. The approach was first introduced to biology by Oster *et al.* [13], but more recent examples include application to systems biology by Ederer and Gilles [14], Liebermeister and Klipp [15] and Gawthrop and Crampin [16]. Whereas traditional kinetic models in biochemistry depend on complex detailed balance constraints to account for energy conservation, a key advantage of energy-based models is that individual parameters can be individually perturbed without violating thermodynamic consistency. Such developments are starting to find application in whole-cell modelling, with Mason and Covert [17] finding that parameterising models in a thermodynamically consistent fashion can in fact lead to more robust parameter identification.

The development and analysis of energy-based models can be significantly simplified if one knows the energetic properties of each component in a system and how the components are connected. To this effect, bond graphs have proved to be valuable tools for enabling the energy-based approach. Bond graphs were invented by Henry Paynter as a method of modelling multi-physics systems in engineering. They were subsequently proposed as a modelling approach in biology by Oster *et al.* [13], and more recently developed by Gawthrop and Crampin [16]. Because bond graphs are based on a graphical representation, they allow the complexities of energy-based modelling to be abstracted away so that a modeller need only deal with a map (graph) of biological components and their connections.

However, despite recent developments in bond graph theory, software tools for constructing and simulating bond graph models, in particular of biological systems, have not kept up with recent developments in software practices. Almost all of the existing software implementations of bond graph modelling are graphical in nature and within a proprietary or isolated software ecosystem. For some applications, this is beneficial; vendors can provide a standardised visual interface with integrated analysis tools. However, for systems biology, the existing software lacks the capacity for automation, is hampered by restricted access to source code and is challenging to integrate with existing algorithms. Hence, there is need for a symbolic modelling toolkit that is open-source, easy to integrate into existing workflows, and written in an accessible and widely used scripting language. In particular, there is a need for infrastructure to support automated model building and simplification.

Here we introduce `BondGraphTools`, a python library for building and manipulating symbolic models of complex physical systems, built upon the standard scientific python libraries. The `BondGraphTools` package is different from existing software in both design and implementation in that it:

1. is explicitly based on physical modelling principles;
2. provides an *application program interface* (API) as opposed to a graphical user interface (GUI);
3. is designed for *symbolic* model composition and order reduction, as opposed to being primarily a numerical toolkit;
4. is intended to be used in conjunction with the standard python libraries, as opposed to being used as a stand-alone software package;
5. allows for modification and integration as it is open source, version controlled, and readily available, instead of closed-source and proprietary;
6. is designed with modularity and extensibility in mind, as opposed to being a monolithic software suite.

The aims of this paper are threefold: firstly to use some simple examples to motivate the use of an energy-based approach for modelling cellular biochemistry (Sect. 2); secondly to illustrate how bond graphs facilitate this approach (Sect. 3); and thirdly to introduce the `BondGraphTools` package for simulating energy-based models in systems biology (Sects. 5–7). We apply this energy-based approach to a model of multisite phosphorylation to illustrate the advantages of this approach for large-scale modelling (Sect. 8).

## 2 A primer on modelling biochemical cycles

Biochemical networks contain numerous thermodynamic cycles. Here we motivate the need to consider thermodynamics when such cycles exist. As a simple example, consider the reactions

$$\mathrm{X} \rightleftharpoons \mathrm{Y} \rightleftharpoons \mathrm{Z} \rightleftharpoons \mathrm{X} \tag{1}$$

Assuming the law of mass action, the system can be described by the differential equations

$$\dot{x}_X = v_3 - v_1 \tag{2a}$$
$$\dot{x}_Y = v_1 - v_2 \tag{2b}$$
$$\dot{x}_Z = v_2 - v_3 \tag{2c}$$

where $x_s$ is the concentration of species $s \in \{\mathrm{X}, \mathrm{Y}, \mathrm{Z}\}$. The reaction fluxes are defined as

$$v_1 = k_1^+ x_X - k_1^- x_Y \tag{3a}$$
$$v_2 = k_2^+ x_Y - k_2^- x_Z \tag{3b}$$
$$v_3 = k_3^+ x_Z - k_3^- x_X \tag{3c}$$

where $k_i^\pm$ are kinetic constants.

It is clear that at steady state, $v_1 = v_2 = v_3$. However, because the system is closed, this steady state must satisfy the stricter constraint that $v_1 = v_2 = v_3 = 0$. Otherwise, the reactions would dissipate heat without any energy being input into the system, violating the second law of thermodynamics. With this in mind, from Eq. 3,

$$k_1^+ x_X = k_1^- x_Y \tag{4a}$$
$$k_2^+ x_Y = k_2^- x_Z \tag{4b}$$
$$k_3^+ x_Z = k_3^- x_X \tag{4c}$$

By taking the product of the three equations, it is easy to see that

$$k_1^+ x_X k_2^+ x_Y k_3^+ x_Z = k_1^- x_Y k_2^- x_Z k_3^- x_X \Rightarrow \frac{k_1^+ k_2^+ k_3^+}{k_1^- k_2^- k_3^-} = 1 \tag{5}$$

Since Eq. 5 is independent of the concentrations of species, it is a constraint on the kinetic parameters. In other words, the parameters $k_i^\pm$ cannot be independently chosen and must satisfy Eq. 5 to avoid describing a system analogous to a perpetual motion machine [14]. This poses a few issues:

1. Detailed balance constraints become substantially more difficult to calculate for complex biochemical networks [14]
2. It is challenging to specify kinetic parameters (or infer from data) while adhering to detailed balance constraints [18]

An alternative method of representing biochemical kinetics is the energy-based approach, which explicitly accounts for the fact that reaction fluxes arise from chemical potentials [16]. Here, each of the species has an associated chemical potential

$$\mu_X = RT \ln(K_X x_X) \tag{6a}$$
$$\mu_Y = RT \ln(K_Y x_Y) \tag{6b}$$
$$\mu_Z = RT \ln(K_Z x_Z) \tag{6c}$$

where $R = 8.314$ J/mol is the ideal gas constant, $T$ [K] is the temperature and $K_i$ are the thermodynamic constants of the species. The chemical potentials are related to reaction rates through the Marcelin-de Donder equation

$$v_1 = r_1(e^{\mu_X/RT} - e^{\mu_Y/RT}) \tag{7a}$$
$$v_2 = r_2(e^{\mu_Y/RT} - e^{\mu_Z/RT}) \tag{7b}$$
$$v_3 = r_3(e^{\mu_Z/RT} - e^{\mu_X/RT}) \tag{7c}$$

where $r_i$ [mol/s] is the rate parameter for reaction $i$. By substituting Eq. 6 into Eq. 7, the equations can be recast in the same form as Eq. 2, but with the reaction rates

$$v_1 = r_1 K_X x_X - r_1 K_Y x_Y \tag{8a}$$
$$v_2 = r_2 K_Y x_Y - r_2 K_Z x_Z \tag{8b}$$
$$v_3 = r_3 K_Z x_Z - r_3 K_X x_X \tag{8c}$$

While these equations exhibit the same mass-action behaviour as the kinetic approach, the thermodynamic approach is immune to thermodynamic inconsistencies; by comparing coefficients between Eqs. 3 and 8,

$$\frac{k_1^+ k_2^+ k_3^+}{k_1^- k_2^- k_3^-} = \frac{r_1 K_X r_2 K_Y r_3 K_Z}{r_1 K_Y r_2 K_Z r_3 K_X} = 1 \tag{9}$$

that is, the parameters in the energetic formulation automatically satisfy the detailed balance constraint in Eq. 5 and thus the parameters are free to be specified independently. This has been found to aid in estimating the parameters for a model of glycolysis [17].

While the above example considered an closed system, biological systems in general continuously dissipate energy. The equilibrium constraint is broken when external chemical species are allowed to interact with the cycle. For example, we now consider the alternative reaction network

$$\mathrm{X + A \rightleftharpoons Y} \tag{10a}$$
$$\mathrm{Y \rightleftharpoons Z} \tag{10b}$$
$$\mathrm{Z \rightleftharpoons X + B} \tag{10c}$$

and assume that A and B are kept at constant concentrations through external flows. This is a biochemical cycle frequently used by enzymes. The reaction rates are

$$v_1 = r_1 K_X x_X K_A \bar{x}_A - r_1 K_Y x_Y \tag{11a}$$
$$v_2 = r_2 K_Y x_Y - r_2 K_Z x_Z \tag{11b}$$
$$v_3 = r_3 K_Z x_Z - r_3 K_X x_X K_B \bar{x}_B \tag{11c}$$

where $K_A$ and $K_B$ are the species constants for A and B respectively and $\bar{x}_A$ and $\bar{x}_B$ are the constant concentrations of A and B. By comparing coefficients with Eq. 3,

$$\frac{k_1^+ k_2^+ k_3^+}{k_1^- k_2^- k_3^-} = \frac{r_1 K_1 K_A \bar{x}_A r_2 K_2 r_3 K_3}{r_1 K_2 r_2 K_3 r_3 K_1 K_B \bar{x}_B} = \frac{K_A \bar{x}_A}{K_B \bar{x}_B} \tag{12}$$

which is not 1 in general, and therefore admits a non-equilibrium steady state. Thus the addition of external species provides a source of energy to the system, allowing the system to dissipate energy under steady state conditions.

## 3 Introduction to bond graph modelling

Bond graphs provide an abstracted and network-based framework for understanding and developing energy-based models. The main principle of bond graph modelling is that the laws of physics provide a justifiable

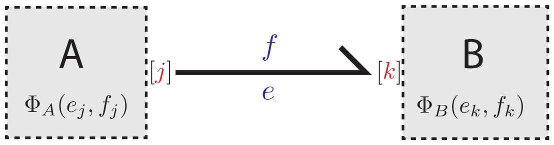

**Fig. 1** The transfer of power through a bond between port $j$ on system A and port $k$ on system B.

interface between different models of physical subsystems. In particular any *connection* between two models of physical processes must conserve energy.

Bond graphs are a port-based modelling approach that describes the flow of power through a network of energy storage and dissipation sub-systems (such as species and reactions respectively). Ports attached to a particular subsystem can be related via an energy conserving 'power bond' and are defined in terms of *force-like* potentials $e$ (also known in the bond graph literature as 'efforts'), and *flux-like* flows $f$ such that the (signed) power transfer at any instant is $P = ef$. In bond graphs, variables such as voltage, pressure and chemical potential are potentials; while current, mass flow and molar flow rate are flow variables. Figure 1 shows an example of a power bond. Here, the power entering system A is given by $P_{\mathrm{A};j} = e_j f_j$, similarly for B; $P_{\mathrm{B};k} = e_k f_k$, such that power is conserved between systems, *i.e.* $P_{\mathrm{A};j} + P_{\mathrm{B};k} = 0$. The directed power transfer through the bond is therefore

$$P_{bond} = ef, \text{ with } e = e_j = e_k \text{ and } f = -f_j = f_k. \quad (13)$$

The internal behaviour of systems A and B are captured via the constitutive relationships $\Phi_{\mathrm{A}}$ and $\Phi_{\mathrm{B}}$, which store or dissipate energy. We refer interested readers to the tutorial by Gawthrop and Bevan [19] for an overview of bond graph modelling and for an in depth treatment see [20].

In order to model complex systems, we must first have an idea of what the model subsystems are, and how they should be connected. All bond graph models have:

- a number of internal components,
- a (possibly zero) number of external power *ports* each of which has two associated variables, a potential-like $e$ (voltage, force, chemical potential, etc.) and a flow-like $f$ (current, velocity, molar flow rate, etc.) such that power $P = ef$ is positive when the process is consuming or accumulating energy through that particular port,
- a set of constitutive relations that characterise the behaviour for a particular model and are either derived from internal components or specified a-priori; for example generalised linear dissipation (friction, Ohm's law) which relates the effort and flow via the implicit relation $0 = e - Rf$. We note that the power entering this model $P = ef = Rf^2$ is positive semi-definite for $R > 0$ indicating, as one would expect, that resistance always consumes power.
- a (possibly zero) number of parameters/controls which govern the behaviour of that particular model. One would consider $R$ in the above example to be a parameter if $R$ is constant, and a control in all other cases.
- a (possibly zero) number of state variables (with associated derivatives), related to power ports via the model constitutive relations. For example, a linear potential energy storage has governing equations $Ce - x = 0, \quad f - \dot{x} = 0, \quad \implies P = \frac{1}{C} x\dot{x}$ so that the energy $E(t)$ stored in the state variable $x(t)$ at a particular time $t$ in that component (up to a constant) is $E(t) = \int_0^t P\,\mathrm{d}t = x(t)^2/(2C)$.

Thus building a bond graph model consists of instantiating the models one wishes to use, defining parent-child relationships, then specifying energy sharing between ports. The resulting equations can then be manipulated and simplified to give rise to a set of differential equations. `BondGraphTools` can automatically simplify these equations; this is discussed in Sect. 7. A list of basic components can be found in Table 1.

In bond graphs, network conservation laws, such as Kirchhoff's Laws, are themselves considered components, and hence must be added just as any other model. For example, the **0** junction describes the conservation law where all efforts are equal. Suppose there are $n$ ports associated with this junction, then the constitutive relation is

$$0 = e_k - e_0 \quad \forall\, k \in 1, \ldots, n-1, \qquad \sum_{k=0}^{n-1} f_k = 0. \quad (14)$$

It is clear by inspection that this is indeed power conserving. Similarly, the $n$-port **1** junction, or 'common flow', has relations given by

$$0 = \sigma_j f_j - \sigma_0 f_0 \quad \forall k \in 1, \ldots n-1, \qquad \sum_{k=0}^{n-1} \sigma_k e_k = 0 \quad (15)$$

where $\sigma_k = 1$ if the associated port $k$ is oriented inwards, or $\sigma_k = -1$ if oriented outwards.

## 4 Bond graph models of biochemical systems

The components relevant to biochemical systems are as follows. Note that below, the effort variables $e$ represent chemical potentials $\mu$ and the flow variables represent the fluxes $v$.

**Table 1** Basic bond graph variables and components, and their analogues in different physical domains.

| **Bond graph** | **Chemical** | **Electrical** | **Mechanical** | **Hydraulic** |
|---|---|---|---|---|
| Potential ($e$) | Chemical potential $\mu$ [J/mol] | Voltage $V$ [V] | Force $F$ [N] | Pressure $P$ [Pa] |
| Flow ($f$) | Molar flow rate $v$ [mol/s] | Current $I$ [A] | Velocity $v$ [m/s] | Volume flow $Q$ [m$^3$/s] |
| **C** | Chemical species | Capacitor | Spring | Tank |
| **R** | Reaction | Resistor | Damper | Pipe |
| **Se** | Chemostat | Voltage source | Applied force | Pressure source |
| **TF** | Stoichiometry | Transformer | Lever | Pressure transformer |
| **0** | Common species | Parallel connection | Series connection | Parallel connection |
| **1** | Reactant/product complexes | Series connection | Parallel connection | Series connection |

– Species are represented using **Ce** components – a nonlinear version of the capacitor (**C** component). They have the constitutive relationships

$$\dot{x} - f = 0 \tag{16a}$$
$$e - RT\ln(Kx) = 0 \tag{16b}$$

As mentioned previously, this component contains the species thermodynamic parameter $K$.

– In cases where a species has a constant concentration, it is represented by the alternative **Se** ("potential source") component with a constant chemical potential (which is related to concentration). Such a 'chemostat' component has the constitutive relationship

$$e = RT\ln(K\bar{x}) = \text{constant} \tag{17}$$

*i.e.* this is the same as the **Ce** component, but with the dynamics (Eq. 16a) omitted. Since the concentration is constant, we use the notation $\bar{x}$ to indicate that $d\bar{x}/dt = 0$.

– Reactions are represented using **Re** components, qhich are equivalent to two-port nonlinear resistors. These have the constitutive relationships

$$f_1 + f_2 = 0 \tag{18a}$$
$$f_1 - r(e^{e_1/RT} - e^{e_2/RT}) = 0 \tag{18b}$$

where $e_i$ and $f_i$ are the chemical potentials and fluxes associated with the $i$th bond. These equations are parameterised by the parameter $r$, the reaction rate parameter. The power dissipated is $P = f_1(e_1 - e_2) \geq 0$ because $f_1 > 0 \Leftrightarrow e_1 > e_2$.

– In order to incorporate stoichiometry, **0** and **1** junctions are used. In particular, **0** junctions describe the involvement of a species in multiple reactions and the **1** junction describes the involvement of multiple reactants or products in a single reaction. This is discussed in more detail below.

– When multiple stoichiometries are involved, **TF** (transformer) components can be used to scale the chemical potentials and fluxes accordingly. These follow the constitutive relations

$$f_1 - nf_2 = 0 \tag{19}$$
$$e_2 - ne_1 = 0 \tag{20}$$

where $n$ is a parameter for the stoichiometry.

To illustrate, we will represent the reaction networks discussed in Sect. 2 as bond graphs. The reaction network X $\rightleftharpoons$ Y $\rightleftharpoons$ Z $\rightleftharpoons$ X (Fig. 2A) is represented by the bond graph in Fig. 2B, where each bond has been labelled with its potential and flow variable. As illustrated by the labels, each **Ce** component (blue) has constitutive relations defined by Eq. 16 and each **Re** component (green) has constitutive relations defined by Eq. 18. Since each of the species is involved in two reactions, a common potential (**0**; purple) junction is used to equate the chemical potential contributions of the species to each reaction it is involved in, while simultaneously accounting for conservation of mass. When these conservation equations are accounted for, the efforts and flows can be computed in terms of chemical potentials and fluxes; these are shown in Fig. 2C, where the chemical potentials and fluxes are defined as per Eqs. 6–7. Thus, by using Eq. 16a, it is easily seen that we can recover the dynamics defined by the differential equations

$$\dot{x}_X = r_3K_Zx_Z - r_3K_Xx_X - r_1K_Xx_X + r_1K_Yx_Y \tag{21a}$$
$$\dot{x}_Y = r_1K_Xx_X - r_1K_Yx_Y - r_2K_Yx_Y + r_2K_Zx_Z \tag{21b}$$
$$\dot{x}_Z = r_2K_Yx_Y - r_2K_Zx_Z - r_3K_Zx_Z + r_3K_Xx_X \tag{21c}$$

We can apply similar principles in defining the dynamics of the reaction network X + A $\rightleftharpoons$ Y $\rightleftharpoons$ Z $\rightleftharpoons$ X + B (Fig. 3A), shown in the bond graph in Fig. 3B. In addition to the components used in the previous example, we also use **Se** components and **1** junctions. The **Se**

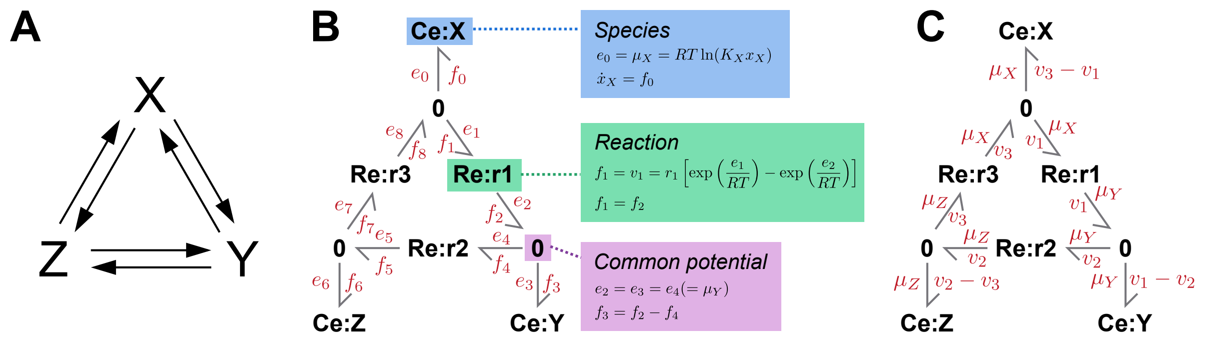
A
X
Z
Y
B
Ce:X
Species
$e_0 = \mu_X = RT\ln(K_X x_X)$
$\dot{x}_X = f_0$
Re:r3
Re:r1
Reaction
$f_1 = v_1 = r_1\left[\exp\left(\frac{e_1}{RT}\right) - \exp\left(\frac{e_2}{RT}\right)\right]$
$f_1 = f_2$
Re:r2
Common potential
$e_2 = e_3 = e_4 (= \mu_Y)$
$f_3 = f_2 - f_4$
Ce:Z
Ce:Y
C
Ce:X
Re:r3
Re:r1
Re:r2
Ce:Z
Ce:Y

**Fig. 2** Representations of the reaction network $X \rightleftharpoons Y \rightleftharpoons Z \rightleftharpoons X$. (A) Biochemical network; (B) Bond graph with efforts and flows labelled; (C) The same bond graph as in (B), but with chemical potentials and flux variables computed from conservation laws.

components (blue) are used for the species A and B due to the fact that we assume that they are chemostats, *i.e.* have constant concentration. In this second example, some reactions have multiple reactants or products. In these cases common flow junctions (**1**; purple) are required to equate their flows while also summing their chemical potential contributions to each reaction; for example, that the forward affinity of the reaction r1 is $\mu_X + \mu_A$. Fig. 3B shows the chemical potential and flux variables associated with each bond, and it can be shown through substuting the relevant chemical potentials into the Marcelin-de Donder equation (Eq. 18b) that the dynamics are governed by the differential equations

$$\dot{x}_X = r_3 K_Z x_Z - r_3 K_X x_X K_B \bar{x}_B - r_1 K_X x_X K_A \bar{x}_A + r_1 K_Y x_Y \tag{22a}$$

$$\dot{x}_Y = r_1 K_X x_X K_A \bar{x}_A - r_1 K_Y x_Y - r_2 K_Y x_Y + r_2 K_Z x_Z \tag{22b}$$

$$\dot{x}_Z = r_2 K_Y x_Y - r_2 K_Z x_Z - r_3 K_Z x_Z + r_3 K_X x_X K_B \bar{x}_B \tag{22c}$$

which as expected, is identical to the equations for the energy-based approach (Eqs. 2,11).

The approach outlined above can be used to build dynamic energy-based models of arbitrary scale, which account for mass, charge and energy conservation. The approach extends naturally to incorporate electrochemical [21] and biomechanical systems.

## 5 BondGraphTools

Broadly speaking, there are two main computational approaches to bond graph modelling; computer aided design tools and mathematical software. Computer Aided Design (CAD) tools follow in the tradition of electrical design automation and include Dymola[1] and 20-sim[2] in which users draft technical schematics of the system from a library of components. Similarly, mathematical software such as MATLAB (Simulink), Maple (MapleSim) or Mathematica (SystemModeller) also allows users to construct graphical representations of the model. While graphical interfaces can be useful in intuitively understanding bond graph models, they lack the capacity to automate the construction of large systems and do not interface very well with the existing ecosystem of numerical analysis software. Furthermore, the vast majority of bond graph software is embedded within proprietary software, limiting the scope for expansion.

`BondGraphTools` aims to resolve the above issues by providing a scripting interface rather than a graphical interface, and is developed within the open-source python language. As a programming interface, `BondGraphTools` gives modellers a means to integrate the tools and techniques of software development into their modelling workflow. This includes being able to script tasks like model re-parametrisation and batch replacement of model subcomponents both of which can be tedious and time-consuming in graphical environments.

Building upon and integrating with the existing python ecosystem means that `BondGraphTools` can specialise in providing an interface for model building without concerning itself with other tasks. This results in a smaller codebase, and hence more sustainable software. Using modern open source practices allows other developers to easily modify, contribute and build upon the `BondGraphTools` codebase. Unlike proprietary software, users are free to implement new features and extend `BondGraphTools` as they see fit, for example by using `BondGraphTools` as a foundation for graphical modelling environments.

For a large class of systems, particularly in systems biology, only the network topology of a system may be known at the time of modelling. As `BondGraphTools` represents model parameters symbolically, values are

[1] `www.3ds.com/products-services/catia/products/dymola/`
[2] `www.20sim.com`

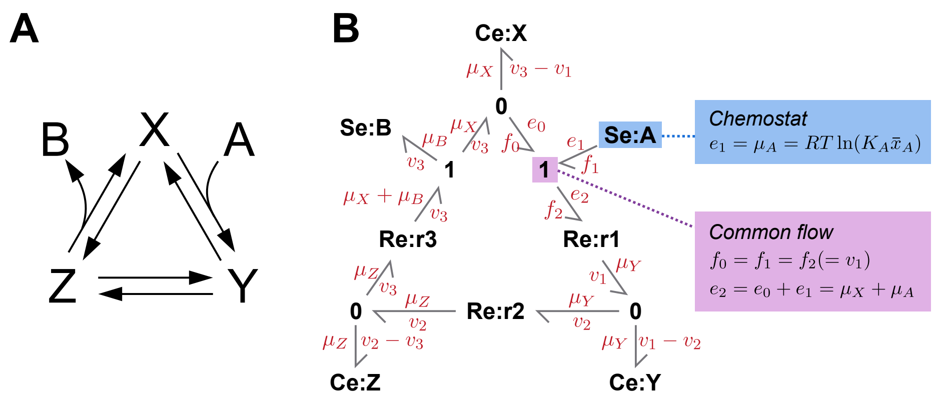

**Fig. 3** Representations of the reaction network $X + A \rightleftharpoons Y \rightleftharpoons Z \rightleftharpoons X + B$. (A) Biochemical network; (B) Bond graph.

free to be determined later in the modelling process via existing parameter estimation techniques.

`BondGraphTools` provides a programming language for building bond graph models which are automatically converted into differential equations to be analysed or simulated. Python is well established as a robust and easy-to-use general purpose programming language with a wide variety of standardised and well supported libraries for standard scientific tasks, and `BondGraphTools` adheres to python language idioms by emphasising self-explanatory, self-documenting and self-contained code.

## 6 Design motivation and basic use

As discussed in Sect. 3, bond graphs are defined in terms of their subsystems are and how they are related. Here we outline the relevant classes of models and how they are related in `BondGraphTools`. Throughout the code examples, we assume `BondGraphTools` has been imported as follows:

```
import BondGraphTools as bgt
```

We demonstrate code as it appears in a python script, which could be equally executed by entering the code into an IPython session or a Jupyter notebook [22].

### 6.1 Creating models

Models in `BondGraphTools` are broadly split into two classes: *atomics*, which represent processes that are considered indecomposable and often fundamental; and *composites*, which are assembled using other models in a has-a relationship. Both of these are constructed using the `new` function.

New composites can be constructed using

```
model = bgt.new(name="New␣Model")
```

which results in the variable `model` containing a new instance of the `BondGraph` class, the composite base class, with no components and with the name 'New Model'. This is identical to creating new instances of the `BondGraph` class directly via

```
model = bgt.BondGraph(name="New␣Model")
```

which is available for the purposes of providing an object oriented interface.

New atomics can be created in a similar manner by specifying the model class and value. For example, a **Ce** component (available in the `BioChem` library) can be initialised by

```
species = bgt.new("Ce", library="BioChem",
    value={'k':1,'R':8.314,'T':310})
```

so that each variable now contains new instances of the respective atomics. This particular atomic has three parameters; the thermodynamic constant $K = 1$, the ideal gas constant $R = 8.314$ and the temperature $T = 310$.

### 6.2 Defining relationships

There are two categories of relationships between models in `BondGraphTools`; structural and energetic, corresponding to the components and bonds respectively. Structural relationships describe composition, how a given model can contain many simpler models. This organises systems into a tree-like hierarchy of parent-children relations between models. Energetic relationships describe how the ports of a given set of child models are connected within a given parent model.

#### *6.2.1 Structural relationship (components)*

In `BondGraphTools`, components can be added (removed) from composite models with the `add` and `remove` functions. For example, adding a species component to the composite model can be achieved with

```
bgt.add(model, species)
```

where the first argument is the parent model, and the remaining arguments are the intended children or components.

A file system interface navigates the model hierarchy and is implemented within the `ModelBase` class from which all models inherit. In particular,

– `model.uri` is a uniform resource identifier (URI) locating that particular model.
– `model.parent` refers to the parent model if it exists.
– `model.root` refers to the top of the model tree.
– For composite models, `Composite.components` will contain a list of sub-models.

Atomics are thus analogous to files, and composites to directories, with the root model analogous to the unix hostname.

#### *6.2.2 Energetic relationships (bonds)*

The energetic relationship defines how power is transferred between components inside a particular model via ports which belong to components, and depends on first establishing a model hierarchy. Once a set of component-wise relationships is established within a model, components are connected to each other (and hence energy bonds defined, in the bond graph terminology). When it is not ambiguous, for example when connecting a one-port component to a junction with arbitrarily many identical ports, it is sufficient to connect the component directly via

```
bgt.connect(component_1, component_2)
```

Otherwise one must specify the port by index

```
target_port = (component_2, 0)
```

and connect the port directly:

```
bgt.connect(component_1, target_port)
```

Composite models keep track of the energy relationships (bonds) in the member attribute `Composite.bonds`. Similarly, two components can be disconnected via the `disconnect` method which has an identical interface to `connect`.

### 6.3 Model attributes

Composite and Atomic models often have associated parameters which can be accessed by the member attribute `ModelBase.params`. In keeping with the emphasis on symbolic equations, parameters can be either numeric or symbolic values. Once the structure of a composite model has been established, and the internal connections defined, one can generate governing equations for the entire model via `ModelBase.constitutive_relations` in terms of the derived state variables `ModelBase.state_vars` to be used for further analysis.

## 7 Symbolic composition and reduction

Once a model has been constructed, simplified equations are automatically derived using symbolic algebraic tools. Having a simplified symbolic representation of the system is valuable as it provides modellers a way to export the equations into whatever format they desire. In particular, the set of implicit equations can easily be fed into standard parameter estimation routines, discretised for implementation in other architectures, or passed to solver routines.

Core to `BondGraphTools` is symbolic model reduction. This relies on the fact that each model or component has a set of constitutive relations $\Phi(X) = 0$ which are implicit equations defining the model behaviour in terms of how power is manipulated.

### 7.1 Model structure

The local co-ordinate space of a model $\alpha$ is taken to be $\mathcal{X}_\alpha = \{\dot{x}_\alpha, e_\alpha, f_\alpha, x_\alpha, u_\alpha\}$ where $x, \dot{x}$ are the vectors of storage co-ordinates and time derivatives respectively, $e, f$ are the power interconnection variables effort and flow, and $u$ are controls or inputs. Let us assume that $\dim x_\alpha = n_\alpha$, $\dim e_\alpha = \dim f_\alpha = m_\alpha$, $\dim u_\alpha = k_\alpha$ for finite $n_\alpha, m_\alpha, k_\alpha$ and $n_\alpha + m_\alpha \geq 1$ so that there is non-trivial behaviour. We also define $N_\alpha = 2(n_\alpha + m_\alpha) + k_\alpha$. One can define a matrix $L_\alpha \in \mathbb{R}^{N_\alpha \times N_\alpha}$, a vector field $V_\alpha : \mathcal{X}_\alpha \to \mathbb{R}^{N_\alpha}$ such that the constitutive relations are equivalent to

$$\mathbf{0} = L_\alpha X + V_\alpha(X) \qquad \forall X \in \mathcal{X}_\alpha. \tag{23}$$

We expect $L_\alpha$ to be sparse and $\operatorname{rank} L_\alpha \leq n_\alpha + m_\alpha + k_\alpha < N_\alpha$ so that there is at least one eigenspace of $L_\alpha$ per state variable pair, unconnected port or control input.

### 7.2 Composition

Any number of constitutive relations $\Phi = [\Phi_\alpha, \Phi_\beta, \ldots] = 0$ can thus be combined via

$$\begin{aligned} 0 &= LX + V(X) \\ &= \begin{pmatrix} L_\alpha & 0 & \ldots \\ 0 & L_\beta & 0 \\ \vdots & 0 & \ddots \end{pmatrix} X + \begin{pmatrix} V_\alpha \circ \pi_\alpha \\ V_\beta \circ \pi_\beta \\ \vdots \end{pmatrix} (X) \end{aligned} \tag{24}$$

where $X \in \mathcal{X} = \mathcal{X}_\alpha \oplus \mathcal{X}_\beta \oplus \dots$ is the direct sum of local co-ordinate spaces and $\pi_\alpha : \mathcal{X} \to \mathcal{X}_\alpha$ is a projection back into local co-ordinates. One can easily incorporate interconnecting power bonds by noting that a bond connecting port $i$ on component $\alpha$ to port $j$ on component $\beta$ can be represented by the rows

$$0 = e_{\alpha;i} - e_{\beta;j} = [\theta_{e;\alpha;i}, -\theta_{e;\beta;j}] \begin{pmatrix} X_\alpha \\ X_\beta \end{pmatrix} \tag{25}$$

and

$$0 = f_{\alpha;i} + f_{\beta;j} = [\theta_{f;\alpha;i}, \theta_{f;\beta;j}] \begin{pmatrix} X_\alpha \\ X_\beta \end{pmatrix} \tag{26}$$

where $\theta_{e;\alpha;i}$ is the co-basis vector such that $\theta_{e;\alpha;i} X_\alpha = e_{\alpha;i}$ i.e,.; $\theta_{e;\alpha;i}$ is a row vector with one in the column corresponding to $e_{\alpha;i}$ (similarly for $\theta_{e;\beta;j}$, and $\theta_{f;\alpha;i}$ and $\theta_{f;\beta;j}$ for the flow variables). It follows that the set of all bonds form a junction structure on the larger space $\mathcal{X}$ represented by a full rank matrix $J$ such that $JX = 0$ with row rank identical to the number of bonds. It is convenient to simply consider this as an additional constitutive relation, and append it to the linearised matrix $L$ in Eq. 24.

Given the space $X$, there exists an orthonormal permutation matrix $P$ such that $X' = P^{-1}X$, and

$$X' = (\dot{x}_\alpha, \dot{x}_\beta, \dots e_\alpha, f_\alpha, e_\beta, f_\beta, \dots, x_\alpha, x_\beta, \dots, u_\alpha, u_\beta, \dots). \tag{27}$$

It also follows from elementary linear algebra that there is exists an invertible matrix $\Lambda$ such that $L' = \Lambda L P$ is in a reduced upper triangular form (reduced row echelon form with leading terms always on the diagonal), resulting in an exact simplification of Eq. 24

$$\Lambda L X + \Lambda V(X) = L'X' + V'(X') = 0 \tag{28}$$

where $V'(X') = \Lambda V(PX')$. The coordinate ordering, and hence the permutation matrix $P$ is chosen so that (in the linear case) triangularisation produces the correct order of dependence when substitution is performed; rates of change, efforts and flows are expressed in terms of state and control variables. This can be trivially extended to simple nonlinear cases as non-zero diagonal entries of $L'$ determine substitution rules, though more complicated systems can produce irreducible algebraic constraints.

### 7.3 Output

Constitutive relations, retrieved via member attribute `ModelBase.constitutive_relations` by evaluating Eq. 28, can be generated for any model at any level of the structural tree, and is performed in a recursive manner to reduce the number of calculations. Reduced symbolic models can be passed into a simulation service, which renders Eq. 28 as a function, then initialises and solves the associated initial value problem, for example by

```
u = "sin(t)"
t,x = simulate(model,
               x0=x0,
               timespan=[0,100],
               control_vars=[u]),
               dt=0.1
```

Numerical integration is provided by the SUNDIALS [23] suite of differential algebraic equation solvers.

## 8 Examples

### 8.1 Closed biochemical cycle

Here we demonstrate the basic features of `BondGraphTools` by constructing the the bond graph of the biochemical cycle in Fig. 2. This is shown in the snippet below.

```
def biochemical_cycle(name="Cycle"):
    R = 8.314
    T = 310.0

    model = bgt.new(name=name)
    X = bgt.new("Ce",name="X",library="BioChem",
                 value={'k':1,'R':R,'T':T})
    Y = bgt.new("Ce",name="Y",library="BioChem",
                 value={'k':2,'R':R,'T':T})
    Z = bgt.new("Ce",name="Z",library="BioChem",
                 value={'k':3,'R':R,'T':T})
    common_X = bgt.new("0",name="X")
    common_Y = bgt.new("0",name="Y")
    common_Z = bgt.new("0",name="Z")
    r1 = bgt.new("Re",name="r1",library="BioChem",
                 value={'r':1,'R':R,'T':T})
    r2 = bgt.new("Re",name="r2",library="BioChem",
                 value={'r':2,'R':R,'T':T})
    r3 = bgt.new("Re",name="r3",library="BioChem",
                 value={'r':3,'R':R,'T':T})
    bgt.add(model,X,Y,Z,common_X,common_Y,common_Z,
            r1,r2,r3)

    bgt.connect(common_X,X)   # bond 0
    bgt.connect(common_X,r1)  # bond 1
    bgt.connect(r1,common_Y)  # bond 2
    bgt.connect(common_Y,Y)   # bond 3
    bgt.connect(common_Y,r2)  # bond 4
    bgt.connect(r2,common_Z)  # bond 5
    bgt.connect(common_Z,Z)   # bond 6
    bgt.connect(common_Z,r3)  # bond 7
    bgt.connect(r3,common_X)  # bond 8

    return model

model1 = biochemical_cycle()
```

Line 5 instantiates a new model. Lines 6–22 create the components in the bond graph model and add them to the model. Finally, the components are connected in lines 24–32, with the bonds labelled according to their indices in Fig. 2B

Note that for this particular model, we define a function `biochemical_cycle` to construct the model. One of the advantages offered by implementing `BondGraphTools` in a scripting language is that functions can be used to

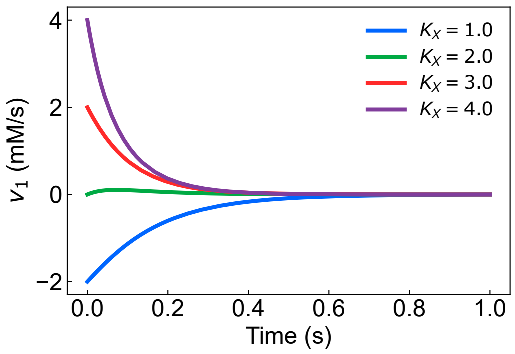

**Fig. 4** Time-dependence of $v_1$ for the reaction network $X \rightleftharpoons Y \rightleftharpoons Z \rightleftharpoons X$.

make multiple copies of the same template. Later, we use this function as a basis for constructing the similar model of an open biochemical cycle (Fig. 3).

For this model, we use the parameters $(K_X, K_Y, K_Z) = (1, 2, 3)$ mM$^{-1}$ and $(r_1, r_2, r_3) = (1, 2, 3)$ s$^{-1}$. These are added to the model by passing in value arguments; 'k' refers to the species constant $K$ and 'r' refers to the reaction rate $r$. For **Ce** and **Re** components, the ideal gas constant $R$ and temperature $T$ also need to be specified.

Once the components, connections and parameters have been defined, BondGraphTools will automatically derive the constitutive relations through the command `model1.constitutive_relations`. The output is shown on the snippet below

```
[dx_0 + 4*x_0 - 2*x_1 - 9*x_2,
 dx_1 - x_0 + 6*x_1 - 6*x_2,
 dx_2 - 3*x_0 - 4*x_1 + 15*x_2]
```

As can be seen, these correspond to Eq. 21 once the parameters have been substituted, where the state variables x_0, x_1 and x_2 have been mapped to $x_X$, $x_Y$ and $x_Z$ respectively.

We run simulations of the model using the initial conditions $(x_X, x_Y, x_Z) = (2, 2, 2)$ mM over the time span 0 s $< t <$ 1 s. In the code below, we run separate simulations of the model with different values of $K_X$.

```
x0 = [2.0,2.0,2.0]
tspan = (0,1.0)
K_X_vals = [1.0, 2.0, 3.0, 4.0]
sol_t = []
sol_x = []
for K_X in K_X_vals:
    (model1/"C:X").set_param('k',K_X)
    t,x = bgt.simulate(model1,tspan,x0,dt=0.01)
    sol_t.append(t)
    sol_x.append(x)
```

We use line 7 to change the value of $K_X$ once the model has been created. Here, the uri is used to retrieve the **Ce**:A component (`model1/"C:A"`), following which the `.set_params` method is used to alter its $K$ value.

The flux through reaction 1 (Eq. 8a) is plotted against time in Fig 4. As can be seen, regardless of the value of $K_X$, the flux settles to zero at steady state, as expected of a thermodynamically consistent model. It can easily be verified by modifying the above code that the same is true when the reaction parameters are altered. This illustrates that parameters in an energy-based model can be independently perturbed without violating thermodynamic consistency, which is not the case in general for kinetic models.

## 8.2 Open biochemical cycle

We next create a bond graph model of the open biochemical cycle in Fig. 3 using BondGraphTools. While we could use the same approach as in above section, here we demonstrate the flexibility of BondGraphTools by instead making an incremental change to the above model. This is demonstrated in the code snippet below.

```
model2 = biochemical_cycle(name="Open␣cycle")
r1 = model2/("R:r1")
r3 = model2/("R:r3")
common_X = model2/("0:X")
bgt.disconnect(common_X,r1)
bgt.disconnect(r3,common_X)

R = 8.314
T = 310.0
K_A = 1
x_A = 1
K_B = 2
x_B = 1

A = bgt.new("Se",name="A",
            value={'e':R*T*np.log(K_A*x_A)})
B = bgt.new("Se",name="B",
            value={'e':R*T*np.log(K_B*x_B)})
XA = bgt.new("1",name="XA")
XB = bgt.new("1",name="XB")
bgt.add(model2,A,B,XA,XB)

bgt.connect(common_X,XA)
bgt.connect(A,XA)
bgt.connect(XA,r1)
bgt.connect(r3,XB)
bgt.connect(XB,common_X)
bgt.connect(XB,B)
```

We first create a new copy of the closed biochemical cycle and assign it to the model2 variable (line 1). Following this, in lines 2–6 we use the uri interface to retrieve and disconnect the connections of the **0** junction connected to **Ce**:X from the reactions **Re**:r1 and **Re**:r2. Finally, we add two **Se** components and two **1** junctions to the model and then connect them according to Fig. 3. By default, the parameters have been set to $K_A = 1$ mM$^{-1}$, $K_B = 2$ mM$^{-1}$, $\bar{x}_A = \bar{x}_B = 1$ mM.

We next run simulations of the model for different values of $\bar{x}_A$, varying from 0.02 to 200 mM (Fig. 5). The simulations are run from 0 to 10 seconds to ensure the model reaches a steady state, and for each simulation, the flux $v_1$ is recorded at $t = 10$ s and plotted

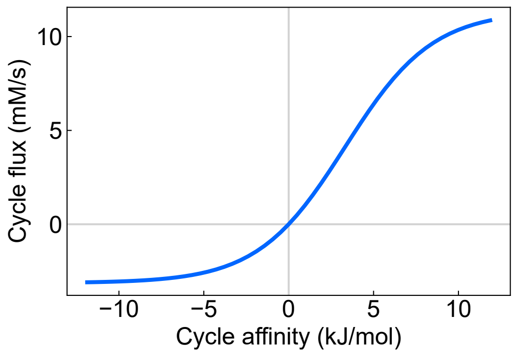

**Fig. 5** Simulations of the reaction network $X + A \rightleftharpoons Y \rightleftharpoons Z \rightleftharpoons X + B$. The affinities have been plotted against the steady-state cycle fluxes.

against the cycle affinity $A = \mu_A - \mu_B$. As expected, the steady-state flux has the same sign as the affinity. The cycle has the overall reaction A $\rightleftharpoons$ B, and the laws of thermodynamics dictate that the conversion of A to B will only proceed if $\mu_A > \mu_B$, regardless of the complexity of the transition. This is reflected in the plot being constrained to the bottom left and upper right quadrants. Similarly, the cycle stalls at $\bar{x}_A = 2$ mM, where $\mu_A = \mu_B = RT\ln(2)$, which also corresponds to the equilibrium condition $x_B/x_A = K_A/K_B = 1/2$. These results are an illustration of the fact that energy-based models ensure open cycles are consistent with the laws of thermodynamics, just as they are with closed cycles.

## 8.3 Multisite phosphorylation

To further demonstrate the the capabilities that `BondGraphTools` has for automation, we make a model of a more complex multisite phosphorylation system. As discussed in Mallavarapu *et al.* [24], the phosphorylation of proteins is an interesting case study for modular modelling approaches as these networks can be highly complex and are capable of reaching multiple steady states. Protein phosphorylation can have three types of properties:

1. The number of sites available for phosphorylation
2. They can be *sequential* or *non-sequential*. Sequential phosphorylation is when sites must be phosphorylated in a specific order, whereas sites in non-sequential phosphorylation may be phosphorylated in any order.
3. They can be *distributive* or *processive*. Distributive phosphorylation only allows a single site to be phosphorylated at a time, whereas processive phosphorylation allows multiple sites (up to an integer $p$) to be phosphorylated or dephosphorylated at once. We call $p$ the processivity number, and $p = 1$ for distributive phosphorylation.

Here we construct a bond graph model of a distributive, sequential four-site phosphorylation network, based on the system described by Thomson and Gunawardena [25]. The reactions for this network are shown in Fig. 6. The substrate is denoted as S, with subscripts indicating the number of phosphorylated sites. The kinase is denoted by E and the phosphatase is denoted by F. We have made the following adaptations to the Thomson and Gunawardena model [25] to make it compatible with the energy-based approach:

1. Since thermodynamic consistency requires reversible reactions, all reactions are assumed to operate in both directions. In cases where reactions were originally defined to be irreversible, the parameters were chosen to give small reverse rate constants.
2. Because energy from the hydrolysis of ATP is required to drive kinases and phosphatases, ATP, ADP and Pi have been added as reactants and products.
3. The energetic parameters $r$ and $K$ were used rather than kinetic parameters $k_i^{\pm}$

Details of how the parameters were determined are given in Appendix A.

In many cases, there are proteins with as many as 40 phosphorylation sites [24], resulting in complex phosphorylation mechanisms. Automation by computer software can significantly reduce the time to develop models of these systems and reduce the occurence of errors. Here we introduce an an alternative method of constructing biochemical models available in `BondGraphTools`: the reaction builder. In the code snippet below, we show how a series of reactions can be defined as a series of strings, with a bond graph model algorithmically constructed from the reaction network in Fig. 6.

```python
from BondGraphTools.reaction_builder \
    import Reaction_Network
rn = Reaction_Network(
    name="4-site sequential phosphorylation")
rn.add_reaction("E + S0 + ATP = ES0",name="e1a")
rn.add_reaction("ES0 = E + S1 + ADP",name="e1b")
rn.add_reaction("F + S1 = FS1",name="f1a")
rn.add_reaction("FS1 = F + S0 + Pi",name="f1b")
rn.add_reaction("E + S1 + ATP = ES1",name="e2a")
rn.add_reaction("ES1 = E + S2 + ADP",name="e2b")
rn.add_reaction("F + S2 = FS2",name="f2a")
rn.add_reaction("FS2 = F + S1 + Pi",name="f2b")
rn.add_reaction("E + S2 + ATP = ES2",name="e3a")
rn.add_reaction("ES2 = E + S3 + ADP",name="e3b")
rn.add_reaction("F + S3 = FS3",name="f3a")
rn.add_reaction("FS3 = F + S2 + Pi",name="f3b")
rn.add_reaction("E + S3 + ATP = ES3",name="e4a")
rn.add_reaction("ES3 = E + S4 + ADP",name="e4b")
rn.add_reaction("F + S4 = FS4",name="f4a")
rn.add_reaction("FS4 = F + S3 + Pi",name="f4b")

rn.add_chemostat("ATP")
rn.add_chemostat("ADP")
rn.add_chemostat("Pi")

model = rn.as_network_model(normalised=True)
```

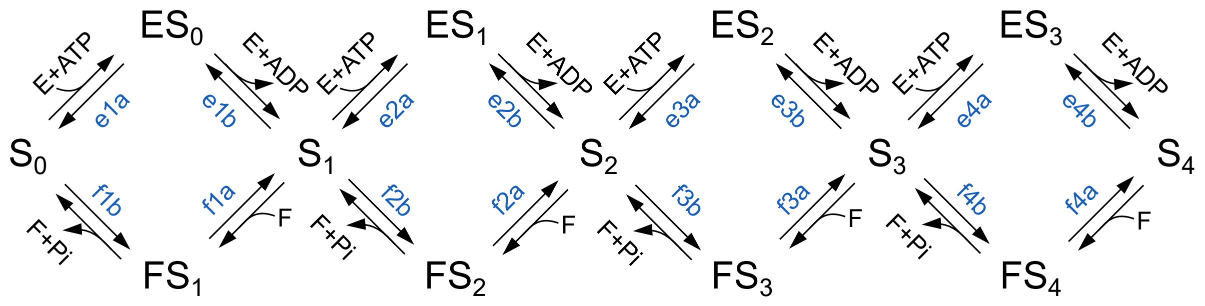

**Fig. 6** A reaction scheme of distributive sequential phosphorylation of four sites. The names of each reaction are labelled in blue.

As illustrated by the code, the `reaction_builder` module has the following functionalities:

1. A new reaction network can be initialised using the `Reaction_Network` class (lines 3–4)
2. New reactions can be added using the `.add_reaction` method (lines 5–20)
3. Chemostats can be labelled using the `.add_chemostat` method. (lines 22–24). In this example, we assume that the concentrations of ATP, ADP and Pi are maintained at constant levels, and thus define them as chemostats.
4. A bond graph model can be algorithmically built from the reaction network using the `.as_network_model` method (line 26).

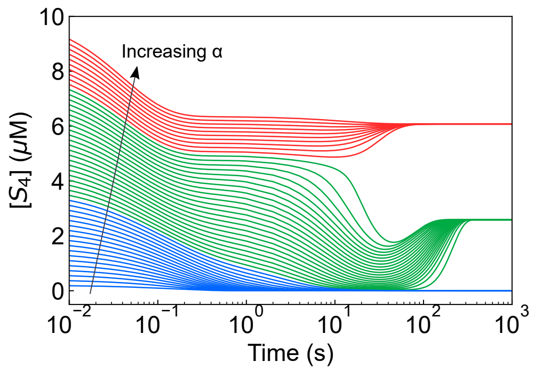

**Fig. 7** Simulations of the distributive sequential four-site phosphorylation model. The simulations were run with the initial conditions $[E] = [F] = 2.8$ $\mu$M, $[S_0] = \alpha S_{\text{tot}}$ and $[S_4] = (1-\alpha)S_{\text{tot}}$, where $\alpha$ is varied between 0 and 0.98 and $S_{\text{tot}} = 10$ $\mu$M. All other concentrations had an initial concentration of zero. Depending on the initial conditions, the system can achieve one of three steady states; these are grouped into blue, green and red lines.

Once the bond graph model has been constructed, the usual functions and methods in `BondGraphTools` can be used to define parameters and run simulations (see Sects. 8.1–8.2).

We simulate the model with the initial conditions $[E] = [F] = 2.8$ $\mu$M, $[S_0] = \alpha S_{\text{tot}}$ and $[S_4] = (1-\alpha)S_{\text{tot}}$, where $\alpha$ is a parameter varied between 0 and 0.98 and $S_{\text{tot}} = 10$ $\mu$M. The simulation results in Fig. 7 show that the system exhibits three distinct steady states (lines in blue, green and red), which is consistent with previous results on these types of models [24, 25].

We note that since `BondGraphTools` is embedded in python, it is very flexible in generating models of with similar but non-identical structures. For example, the code for this example could easily be generalised to construct energy-based models of distributive phosphorylation with an arbitrary number of sites $n$ and processivity $p$. This is shown in the code snippet below, where the function `multisite(n,p)` will create a bond graph model of a phosphorylation system with $n$ sites and processivity $p$.

```python
def multisite(n_sites,processivity=1,
              name="Multisite␣Phosphorylation"):
    substrates = [f"S{i}" for i in range(n_sites+1)]
    (E,F,ATP,ADP,Pi) = ("E","F","ATP","ADP","Pi")

    rn = Reaction_Network(name=name)
    for p in range(1,processivity+1):
        for i in range(n_sites+1-p):
            Sa = substrates[i]
            Sb = substrates[i+p]
            if p == 1:
                p_str = ""
            else:
                p_str = f"{p}*"

            # Kinase reactions
            rn.add_reaction(
                f"{Sa}+{E}+{p_str}{ATP}="+
                f"{E}{Sa}_{p_str[:-1]}{ATP}",
                name=f"e({i},{i+p})a")
            rn.add_reaction(
                f"{E}{Sa}_{p_str[:-1]}{ATP}="
                +f"{Sb}+{E}+{p_str}{ADP}",
                name=f"e({i},{i+p})b")

            # Phosphatase reactions
            rn.add_reaction(
                f"{Sb}+{F}={F}{Sb}",
                name=f"f({i+p},{i})a")
            rn.add_reaction(
                f"{F}{Sb}={Sa}+{F}+{p_str}{Pi}",
                name=f"f({i+p},{i})b")

    rn.add_chemostat(ATP)
    rn.add_chemostat(ADP)
    rn.add_chemostat(Pi)

    model = rn.as_network_model(normalised=True)
    return model
```

## 9 Discussion

Biological systems are constrained by the laws of physics in general and the laws of thermodynamics in particular. Here we have argued for the use of an energy-based approach for developing models consistent with the laws of physics. Bond graphs are an established modelling tool for enabling this approach, and we have introduced `BondGraphTools`, a software package that adds a layer of automation to bond graph modelling.

### 9.1 Model abstraction

The complexity of large projects, modelling or otherwise, is often made tractable by introducing abstractions which allow individuals to 'divide-and-conquer' the problem. For this to work, one requires a way for sub-components to interface with each other. Bond graphs bring this to modelling by framing the interconnection in terms of power, which in turn allows `BondGraphTools` to define a consistent way of joining sub-models together (via ports). Hence, one avoids some connectivity issues such an inability for models to interface and failing to account for loading, which are issues that commonly arise in systems and synthetic biology [3, 26]. Furthermore, the energy-based parameterisation of biochemical systems that an energy-based approach provides has the potential to resolve potential inconsistencies between models [17].

Over recent years, systems biologists have favoured a "white-box" approach to modularity. Here, each module is itself a simulatable model that can be later merged or composed with other models, possibly through semantic annotation of components [27, 28]. White-box modularity requires connections to be exposed on the fly and implementing this approach requires a degree of flexibility that is unavailable through traditional graphical interfaces for bond graph software. This white-box approach has recently been applied in the context of bond graphs [29, 30], and `BondGraphTools` is particularly suited to implementing white-box modularity by being embedded in a scripting environment. Furthermore, python contains an object-oriented interface that could potentially allow an annotation scheme to be built on top of components in `BondGraphTools`.

`BondGraphTools` has a similar philosophy to the programmatic modelling approach in systems biology, as implemented by *little b* [24] and more recently PySB [31]. Key features that these packages share are that: (a) each model is an object in a computer program that can be modified, and (b) that they are embedded within a scripting environment, which allows models to be created through highly abstracted means such as functions, rules and macros. However, `BondGraphTools` differs from the above two packages in that it creates energy-based models with energetic parameters, and that it is a general purpose representation, making it more suitable for multi-physical systems such as electrophysiology and mechanochemistry. Further work will aim to provide a richer set of interfaces to `BondGraphTools` for systems biology, including incorporating abstracted approaches such as rule-based modelling [32] and building a library of macros for common motifs such as enzyme-catalysed reactions.

### 9.2 Towards multi-physics, multiscale models

A characteristic of biology is that it spans across multiple physical domains: electrophysiology and redox reactions are electrochemical systems, whereas muscle contraction and mechanosensation are mechanochemical processes. Bond graphs have the potential to drive theoretical advances in these fields due to their generalised nature, which naturally allows models to span across different physical domains. Connections between ports representing different physical domains are made using the energy-transmitting **TF** (transformer) component [19, 20]. Thus, for example, the chemical and electrical domains are connected using a transformer with modulus $F$; the Faraday constant. Indeed, bond graphs have already been shown to be useful in these areas [21, 33–36]. Due to the inherent modularity of bond graphs, such models could potentially be linked models of circulation to create multi-scale models of organs [37].

An area of active research is the construction of fully integrated models of heart cells, incorporating electrophysiology, calcium signalling, contraction and metabolism [38, 39]. In some cases, heart disease has been hypothesised to arise from the dysregulation of matching energy production to demand. Thus, bond graph models are ideally suited to answering these questions with their ability to model across multiple physical domains and their thermodynamic consistency.

### 9.3 Version controlled models

Because new biological datasets are actively being generated, models in systems biology are frequently updated in light of new measurements, and later models will often inherit parameters and equations from earlier models. It is therefore of critical importance the provenance of models to be recorded, particularly with respect to the relationship between parameters and equations to experimental protocols and data [40].

In `BondGraphTools`, models are constructed via an API, and hence the entire process is textual. This means that models can be version controlled using standard concurrent versioning systems (CVS) such as git, mercurial or svn. In Sect. 8.2, we discussed one way in which `BondGraphTools` could be relevant in this space by reusing a template to make an incremental change to a model. We envision that libraries of models created by `BondGraphTools` could be assembled, shared and updated in the same way one would package and distribute a python script. While the application of `BondGraphTools` here is a subject of future work, it is clear that the utility of integrating efficient and well established 'off-the-shelf' version tracking along with the distribution of model libraries cannot be understated.

### 9.4 Sustainable software practices

The development of `BondGraphTools` employs many sustainable software techniques which benefit both developers and users. `BondGraphTools` is available on the python package index (PyPI) and requires python 3.7. The latest version can be installed using the console command `pip3 install BondGraphTools` and requires only the python and SUNDIALS binaries to be pre-installed. Further installation guides, tutorial, and code documentation is hosted on readthedocs[3] which is updated automatically as new versions are released. Source code is accessible online[4] and is distributed under the permissive, open-source Apache 2.0 license. Semantic versioning via git tags is used to keep track of releases in perpetuity. The code required to run the examples in this paper are available both on GitHub[5] and Zenodo[6] for posterity to capture both the library and dependencies' state at the time of publication to ensure reproducibility [41].

Development of `BondGraphTools` proceeds within an agile iterative and adaptative methodology, as opposed to using a sequential plan-build-test process, which ensures that usable software is prioritised and time is not wasted on planning features that users are less interested in. Instead, requirements and design emerge out of the development process and user experiences, with punctuated breaks for refactoring to simplify the codebase. Test-driven development (TDD) is employed to ensure quality, continuity and stability with automated integration/testing and code metrics provided via Travis.ci[7] and CodeClimate[8] respectively. With TDD, unit tests for new software features are specified and implemented before development begins. A feature is then complete once it passes the corresponding unit tests (in addition to the existing tests) prompting integration of the new code into the trunk of code base. TDD is eminently suited to API or library development as developing a battery of tests for common use cases is an integral part of the development process ensuring that existing functionality does not break as new features are added and encouraging developers to think hard about the library interface. At the time of writing, test coverage is around 80%, which is respectable in the context of an active, iterative development process.

The feature development and bug status can be tracked using the GitHub issue interface. Users are encourage to log bug reports about any issues they might have.

## 10 Conclusion

Specialist tools are required to build and analyse complex biophysical systems and these often take the form of monolithic computer aided design applications, or extensions to mathematical software. Here we have presented `BondGraphTools`, an open source python library for systems modelling which enables the model building process to be scripted, allowing for automation previously unavailable. Further, `BondGraphTools` uses best-practice software engineering techniques to increase the sustainability and longevity of the software. We have demonstrated how `BondGraphTools` can contribute to the modeller's workflow by considering a number of biological models in the existing literature. `BondGraphTools` continues to be actively developed and used across a variety of problem domains in systems biology, bio-electrics and electrical engineering, and also has potential applications in optomechanics [42]. Although basic symbolic model order reduction exists within `BondGraphTools`, an important future research direction involves developing and implementing scalable symbolic order reduction algorithms for nonlinear systems. Reduced order equations are a desired output as they are a simpler representation of the system dynamics, giving greater insight into how parameters effect emergent states and speeding up numerical simulations. This is particularly important for systems with multiple time scales, occurring commonly in biology and engineering, which can pose significant challenges to numerical solvers. `BondGraphTools` offers an promising platform

[3] `http://bondgraphtools.readthedocs.io`
[4] `https://github.com/BondGraphTools`
[5] `https://github.com/uomsystemsbiology/BGT-Biology`
[6] `https://doi.org/10.5281/zenodo.4626922`
[7] `https://www.travis-ci.com`
[8] `www.codeclimate.com`

on which to develop both exact and approximate symbolic reduction algorithms which would be a useful in systems biology, but also more broadly in engineering, scientific computing and computational mathematics.

**Acknowledgements** This research was in part conducted and funded by the Australian Research Council Centre of Excellence in Convergent Bio-Nano Science and Technology (project number CE140100036). PJG would like to thank the Faculty of Engineering and Information Technology, University of Melbourne, for its support via a Professorial Fellowship.

## Author contribution statement

PC designed and wrote the code for `BondGraphTools`, including developing the methodology for symbolic reduction. MP applied the methodology to biochemical systems. PC and MP wrote the manuscript with support from PJG and EJC. All authors revised and edited the manuscript.

## Appendix A: Parameters for multisite phosphorylation

When only kinetic data are available, the energetic parameters cannot be determined uniquely due to the relative nature of chemical potential and the absence of reference values. This parameter undeterminacy has been discussed in previous papers, where sets in parameter space with equivalent kinetic behaviours were defined [17, 21, 43]. However, for this model, we will avoid the issue of parameter uncertainty for simplicity and refer the reader to the above articles for further information.

We make the following assumptions on the energetic parameters:

- We set $\mu_{\mathrm{ATP}} = 50$ kJ/mol, $\mu_{\mathrm{ADP}} = 0$ kJ/mol and $\mu_{\mathrm{Pi}} = 0$ kJ/mol since the free energy of ATP hydrolysis $\Delta G = \mu_{\mathrm{ADP}} + \mu_{\mathrm{Pi}} - \mu_{\mathrm{ATP}}$ typically varies between $-50$ kJ/mol and $-60$ kJ/mol in physiological contexts.
- We set $K_E = K_F = K_{S_0} = 1\ \mathrm{nM}^{-1}$ (corresponding to a free energy of formation of zero at a standard concentration of 1 nM).
- We assume that the energetics of phosphorylation are identical for each site, *i.e.* $K_{S_{i+1}} = \gamma K_{S_i}$. Hence, in conjunction with the above assumption, $K_{S_i} = \gamma^i$. For reasons that we justify later, we have chosen $\gamma = 3.47 \times 10^4$.

Note that implicit in the above assumptions are that ADP, Pi, $\mathrm{S}_0$, E and F are the “reference species” whose chemical potentials are used to define the potentials of the rest of the species. It is relatively straightforward to adapt the assumptions when the more standard chemical free energies of formation are used, in which the chemical potentials of each species are referenced to their constituent elements in their standard states.

The rest of the parameters can be determined from the kinetic parameters in Thomson and Gunawardena [25]. Omitting chemostats, each of the enzyme-catalysed reactions follows the generic structure

$$\mathrm{S}_u + \mathrm{X} \underset{b}{\overset{a}{\rightleftharpoons}} \mathrm{XS}_u \underset{d}{\overset{c}{\rightleftharpoons}} \mathrm{S}_v + \mathrm{X} \tag{A.1}$$

where $\mathrm{S}_u$ and $\mathrm{S}_v$ are the input substrates, $\mathrm{X} = \mathrm{E}$ or F and $\mathrm{XS}_u$ is the complex. The kinetic constants of Thomson and Gunawardena are given in Table 2. Since the reaction scheme does not explicitly account for chemostats, they have been absorbed into the kinetic parameters.

We need to determine the rate constants of both reactions and the species constant of $\mathrm{XS}_u$. These can be determined using the following equations, which are derived by writing kinetic parameters in terms of the energetic parameters:

$$r_a = a/(K_{S_u} K_E e^{A^f_{\mathrm{cs}}/RT}) \tag{A.2}$$

$$K_{\mathrm{XS}_u} = b/r_a \tag{A.3}$$

$$r_b = c/K_{\mathrm{XS}_u} \tag{A.4}$$

where $A^f_{\mathrm{cs}}$ and is the potential of the reactant chemostat (if present). Similarly, we can define $A^r_{\mathrm{cs}}$ as the potential of the product chemostat. Thus, for kinases (E), $A^f_{\mathrm{cs}} = \mu_{\mathrm{ATP}}$ and $A^r_{\mathrm{cs}} = \mu_{\mathrm{ADP}}$. For phosphatases (F), $A^f_{\mathrm{cs}} = 0$ and $A^r_{\mathrm{cs}} = \mu_{\mathrm{Pi}}$.

The remaining kinetic constant $d = r_b S_v A^r_{\mathrm{cs}}$ is assumed to be zero in Thomson and Gunawardena, but this is impossible in a real system, which requires all reactions to be reversible. Thus, we choose the final parameter $\gamma$ to minimise the magnitude of these rate constants, or more precisely, we minimise their squared sum

$$J = \sum_{i=1}^{4} (d_i^E)^2 + \sum_{i=1}^{4} (d_i^F)^2 \tag{A.5}$$

where $d_i^X$ is the parameter $d$ for the $i$th kinase (X = E) or phosphatase (X = F) reaction. A value of $\gamma = 3.47 \times 10^4$ will minimise $J$. The full list of energetic parameters is given in Tables 3–4.

## References

1. P. Kohl, E.J. Crampin, T.A. Quinn, D. Noble, Clinical Pharmacology & Therapeutics **88**(1), 25 (2010). DOI 10.1038/clpt.2010.92

**Table 2** Kinetic parameters used in the Thomson and Gunawardena model [25].

| | $i=1$ | $i=2$ | $i=3$ | $i=4$ |
|---|---|---|---|---|
| $a_i^E$ (nM/s) | 0.00812 | 0.102 | 0.00812 | 0.102 |
| $b_i^E$ (nM/s) | 0.016 | 0.204 | 0.016 | 0.204 |
| $c_i^E$ ($s^{-1}$) | 0.100 | 10.00 | 0.100 | 10.000 |
| $a_i^F$ (nM/s) | 0.112 | 0.00264 | 0.651 | 0.00285 |
| $b_i^F$ (nM/s) | 0.224 | 0.005 | 1.30 | 0.006 |
| $c_i^F$ ($s^{-1}$) | 11.0 | 0.017 | 63.9 | 0.136 |

**Table 3** Species parameters $K$ for the four-site phosphorylation example. All quantities are in $nM^{-1}$

| Species | Value |
|---|---|
| E | $1.000\times10^{0}$ |
| F | $1.000\times10^{0}$ |
| $S_0$ | $1.000\times10^{0}$ |
| $S_1$ | $3.470\times10^{4}$ |
| $S_2$ | $1.204\times10^{09}$ |
| $S_3$ | $4.178\times10^{13}$ |
| $S_4$ | $1.450\times10^{18}$ |
| $ES_0$ | $5.246\times10^{8}$ |
| $ES_1$ | $1.848\times10^{13}$ |
| $ES_2$ | $6.316\times10^{17}$ |
| $ES_3$ | $2.225\times10^{22}$ |
| $FS_1$ | $6.940\times10^{4}$ |
| $FS_2$ | $2.280\times10^{9}$ |
| $FS_3$ | $8.344\times10^{13}$ |
| $FS_4$ | $3.052\times10^{18}$ |

**Table 4** Reaction parameters $r$ for the four-site phosphorylation example. All quantities are in nM/s.

| Reaction | Value |
|---|---|
| e1a | $3.050\times10^{-11}$ |
| e1b | $1.906\times10^{-10}$ |
| f1a | $3.228\times10^{-6}$ |
| f1b | $1.585\times10^{-4}$ |
| e2a | $1.104\times10^{-14}$ |
| e2b | $5.412\times10^{-13}$ |
| f2a | $2.193\times10^{-12}$ |
| f2b | $7.455\times10^{-12}$ |
| e3a | $2.533\times10^{-20}$ |
| e3b | $1.583\times10^{-19}$ |
| f3a | $1.558\times10^{-14}$ |
| f3b | $7.659\times10^{-13}$ |
| e4a | $9.170\times10^{-24}$ |
| e4b | $4.495\times10^{-22}$ |
| f4a | $1.966\times10^{-21}$ |
| f4b | $4.456\times10^{-20}$ |

2. J.R. Karr, J.C. Sanghvi, D.N. Macklin, M.V. Gutschow, J.M. Jacobs, B. Bolival, N. Assad-Garcia, J.I. Glass, M.W. Covert, Cell **150**(2), 389 (2012). DOI 10.1016/j.cell.2012.05.044
3. B. Szigeti, Y.D. Roth, J.A.P. Sekar, A.P. Goldberg, S.C. Pochiraju, J.R. Karr, Current Opinion in Systems Biology **7**, 8 (2018). DOI 10.1016/j.coisb.2017.10.005
4. D.N. Macklin, T.A. Ahn-Horst, H. Choi, N.A. Ruggero, J. Carrera, J.C. Mason, G. Sun, E. Agmon, M.M. DeFelice, I. Maayan, K. Lane, R.K. Spangler, T.E. Gillies, M.L. Paull, S. Akhter, S.R. Bray, D.S. Weaver, I.M. Keseler, P.D. Karp, J.H. Morrison, M.W. Covert, Science **369**(6502), eaav3751 (2020). DOI 10.1126/science.aav3751
5. E.J. Crampin, N.P. Smith, P.J. Hunter, Journal of Molecular Histology **35**(7), 707 (2004)
6. P. Hunter, IEEE Pulse **7**(4), 36 (2016). DOI 10.1109/MPUL.2016.2563841
7. D. Noble, Y. Rudy, Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences **359**(1783), 1127 (2001). DOI 10.1098/rsta.2001.0820
8. T.J. Hund, J.P. Kucera, N.F. Otani, Y. Rudy, Biophysical Journal **81**(6), 3324 (2001). DOI 10.1016/S0006-3495(01)75965-6
9. S. Safaei, C.P. Bradley, V. Suresh, K. Mithraratne, A. Muller, H. Ho, D. Ladd, L.R. Hellevik, S.W. Omholt, J.G. Chase, L.O. Müller, S.M. Watanabe, P.J. Blanco, B. de Bono, P.J. Hunter, Journal of Physiology **594**(23), 6909 (2016). DOI 10.1113/JP272660
10. E. Schrödinger, *What is life? The physical aspect of the living cell* (The University Press; The Macmillan Company, New York, 1962)
11. K.C. Soh, V. Hatzimanikatis, Current Opinion in Microbiology **13**(3), 350 (2010). DOI 10.1016/j.mib.2010.03.001
12. A. Flamholz, E. Noor, A. Bar-Even, R. Milo, Nucleic Acids Research **40**(D1), D770 (2012). DOI 10.1093/nar/gkr874
13. G.F. Oster, A.S. Perelson, A. Katchalsky, Quarterly Reviews of Biophysics **6**(1), 1 (1973). DOI 10.1017/S0033583500000081
14. M. Ederer, E.D. Gilles, Biophysical Journal **92**(6), 1846 (2007). DOI 10.1529/biophysj.106.094094
15. W. Liebermeister, E. Klipp, Theoretical Biology and Medical Modelling **3**, 41 (2006). DOI 10.1186/1742-4682-3-41
16. P.J. Gawthrop, E.J. Crampin, Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences **470**(2171), 20140459 (2014). DOI 10.1098/rspa.2014.0459
17. J.C. Mason, M.W. Covert, Journal of Theoretical Biology **461**, 145 (2019). DOI 10.1016/j.jtbi.2018.10.041
18. P. Saa, L.K. Nielsen, PLOS Computational Biology **11**(4), e1004195 (2015). DOI 10.1371/journal.pcbi.1004195
19. P. Gawthrop, G. Bevan, IEEE Control Systems Magazine **27**(2), 24 (2007). DOI 10.1109/MCS.2007.338279
20. D.C. Karnopp, D.L. Margolis, R.C. Rosenberg, *System Dynamics: Modeling, Simulation, and Control of Mechatronic Systems*, 5th edn. (John Wiley & Sons, Hoboken, New Jersey, 2012). DOI 10.1002/9781118152812
21. M. Pan, P.J. Gawthrop, K. Tran, J. Cursons, E.J. Crampin, Journal of Theoretical Biology **481**, 10 (2019). DOI 10.1016/j.jtbi.2018.09.034
22. F. Perez, B.E. Granger, Computing in Science & Engineering **9**(3), 21 (2007). DOI 10.1109/MCSE.2007.53
23. A.C. Hindmarsh, P.N. Brown, K.E. Grant, S.L. Lee, R. Serban, D.E. Shumaker, C.S. Woodward, ACM Transactions on Mathematical Software **31**(3), 363 (2005). DOI 10.1145/1089014.1089020
24. A. Mallavarapu, M. Thomson, B. Ullian, J. Gunawardena, Journal of The Royal Society Interface **6**(32), 257 (2008). DOI 10.1098/rsif.2008.0205
25. M. Thomson, J. Gunawardena, Nature **460**, 274 (2009). DOI 10.1038/nature08102
26. D. Del Vecchio, Annual Reviews in Control **37**(2), 333 (2013). DOI 10.1016/j.arcontrol.2013.09.011

27. M.L. Neal, M.T. Cooling, L.P. Smith, C.T. Thompson, H.M. Sauro, B.E. Carlson, D.L. Cook, J.H. Gennari, PLOS Computational Biology **10**(10), e1003849 (2014). DOI 10.1371/journal.pcbi.1003849
28. M.L. Neal, B.E. Carlson, C.T. Thompson, R.C. James, K.G. Kim, K. Tran, E.J. Crampin, D.L. Cook, J.H. Gennari, PLOS ONE **10**(12), e0145621 (2015). DOI 10.1371/journal.pone.0145621
29. P.J. Gawthrop, IEEE Transactions on NanoBioscience **16**(3), 177 (2017). DOI 10.1109/TNB.2017.2674683
30. N. Shahidi, M. Pan, S. Safaei, K. Tran, E.J. Crampin, D.P. Nickerson, bioRxiv p. 2021.03.09.434672 (2021). DOI 10.1101/2021.03.09.434672
31. C.F. Lopez, J.L. Muhlich, J.A. Bachman, P.K. Sorger, Molecular Systems Biology **9**(1), 646 (2013). DOI 10.1038/msb.2013.1
32. L.A. Chylek, L.A. Harris, C.S. Tung, J.R. Faeder, C.F. Lopez, W.S. Hlavacek, WIREs Syst. Biol. Med. **6**(1), 13 (2014). DOI 10.1002/wsbm.1245
33. P.J. Gawthrop, P. Cudmore, E.J. Crampin, Journal of Theoretical Biology **493**, 110223 (2020). DOI 10.1016/j.jtbi.2020.110223
34. M. Pan, P.J. Gawthrop, K. Tran, J. Cursons, E.J. Crampin, Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences **474**(2214), 20180106 (2018). DOI 10.1098/rspa.2018.0106
35. P.J. Gawthrop, M. Pan, Bioelectricity (2020). DOI 10.1089/bioe.2020.0042
36. V. Díaz-Zuccarini, C. Pichardo-Almarza, Interface Focus **1**(3), 426 (2011). DOI 10.1098/rsfs.2010.0038
37. B. de Bono, S. Safaei, P. Grenon, P. Hunter, Interface Focus **8**(1), 20170026 (2018). DOI 10.1098/rsfs.2017.0026
38. E.J. Crampin, M. Pan, K. Tran, S. Ghosh, V. Rajagopal, in *Modeling and Simulating Cardiac Electrical Activity*, ed. by T. Krogh-Madsen, D.J. Christini, 2053-2563 (IOP Publishing, 2020), pp. 6–1 to 6–45. DOI 10.1088/978-0-7503-2064-1ch6
39. K. Tran, D.S. Loiselle, E.J. Crampin, Physiological Reports **3**(7), e12464 (2015). DOI 10.14814/phy2.12464
40. A.C. Daly, M. Clerx, K.A. Beattie, J. Cooper, D.J. Gavaghan, G.R. Mirams, Progress in Biophysics and Molecular Biology **139**, 3 (2018). DOI 10.1016/j.pbiomolbio.2018.05.011
41. D.G. Hurley, D.M. Budden, E.J. Crampin, Briefings in Bioinformatics **16**(5), 901 (2015)
42. P. Cudmore, C.A. Holmes, Chaos: An Interdisciplinary Journal of Nonlinear Science **25**(2), 023110 (2015). DOI 10.1063/1.4908604
43. P.J. Gawthrop, J. Cursons, E.J. Crampin, Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences **471**(2184), 20150642 (2015). DOI 10.1098/rspa.2015.0642