Chapter 22

Genetic Network Design Automation with LOICA

Gonzalo Vidal, Carolus Vitalis, Tamara Matúte, Isaac Núñez, Fernán Federici, and Timothy J. Rudge

Abstract

Genetic design automation (GDA) is the use of computer-aided design (CAD) in designing genetic networks. GDA tools are necessary to create more complex synthetic genetic networks in a high-throughput fashion. At the core of these tools is the abstraction of a hierarchy of standardized components. The components' input, output, and interactions must be captured and parametrized from relevant experimental data. Simulations of genetic networks should use those parameters and include the experimental context to be compared with the experimental results.

This chapter introduces Logical Operators for Integrated Cell Algorithms (LOICA), a Python package used for designing, modeling, and characterizing genetic networks using a simple object-oriented design abstraction. LOICA represents different biological and experimental components as classes that interact to generate models. These models can be parametrized by direct connection to the Flapjack experimental data management platform to characterize abstracted components with experimental data. The models can be simulated using stochastic simulation algorithms or ordinary differential equations with varying noise levels. The simulated data can be managed and published using Flapjack alongside experimental data for comparison. LOICA genetic network designs can be represented as graphs and plotted as networks for visual inspection and serialized as Python objects or in the Synthetic Biology Open Language (SBOL) format for sharing and use in other designs.

Key words Computer-Aided Design, Genetic Design Automation, Genetic Network, Modeling, SBOL, Synthetic Biology.

1 Introduction

Synthetic Biology or Engineering Biology as an engineering discipline has the Design Build Test Learn (DBTL) cycle at its core. Modeling is key to the DBTL cycle and is essential to the design and learn stages, given that a model states a well-defined hypothesis about the system operation. Therefore, synthetic biologists have used mathematical and computational models to represent their hypotheses about how biological systems behave at the Design stage. Constructing physical implementations of that biological

Jeffrey Carl Braman (ed.), *Synthetic Biology: Methods and Protocols*, Methods in Molecular Biology, vol. 2760,
https://doi.org/10.1007/978-1-0716-3658-9_22,

system at the Build stage and obtaining measurements from samples of it at the Test stage. Then, experimental data should be analyzed, characterized, and compared with our design to extract information at the Learn stage. The information generated at the Test stage should feed into the Learn and Design stages to increase the understanding of the biological system improving future models and designs. Abstraction enables the construction and analysis of models based on components, devices, and systems that can be used to compose genetic networks and derive their DNA sequences. It is the basis for genetic design automation (GDA), as a computational aided design (CAD) tool for genetic design, which can accelerate and automate the synthetic genetic network design process by compiling models into DNA sequences. In order for GDA to proceed in a rational way, the abstract elements of genetic networks must be accessible to characterization, allowing parametrization of models of their operation and interactions. GDA tools enable the functional decoupling between researchers working at a sequence level and researchers working at the systems level [1, 2]. Standardization is at the foundations of synthetic biology, enabling the collaborative work of a community. The Synthetic Biology Open Language (SBOL) is a free and open-source standard for the representation of biological designs [3]. The SBOL standard was developed by the synthetic biology community to create a standardized format for the electronic exchange of information on the structural and functional aspects of biological designs. This community developed an ecosystem of software tools to, for example, design, model, visualize, store, and share SBOL files [4–10]. Flapjack is a tool from that ecosystem for Synthetic Biology experimental data management [9]. Experimental data in Flapjack can be linked to SBOL metadata in SynBioHub [5].

Logical Operators for Integrated Cell Algorithms (LOICA) is a Python package allowing programmatic design, simulation, parametrization, and analysis of genetic networks [11]. While perhaps not as accessible as a graphical user interface, this approach is more flexible, extensible, and amenable to automation. It can be easily combined with the large ecosystem of biological Python projects [9, 12–14], and uses simple programming concepts that are commonly understood by researchers from a range of disciplines. LOICA genetic network designs can be represented as graphs and plotted as networks for visual inspection. Furthermore, Operators and Metabolism have models of gene expression and growth which parameters can be defined as input or obtained through the analysis of relevant of experimental data [15]. A two-way communication with Flapjack allows LOICA to get experimental data for model parametrization and to upload simulated data to the platform in a straightforward way. GeneticNetworks can generate a SBOL representation that can be shared and used as a component in another design. Here, we describe some use cases and how LOICA can be used to design, visualize, and model genetic networks.

2 Materials

2.1 Dependencies

1. Installing LOICA via pip will automatically install some required packages. These prerequistes include Python 3.8 or a later version. We recommend the use of an environment manager, such as Anaconda (https://www.anaconda.com/).
2. Additionally, users must install pyFlapjack (*see* **Note 1**) to be able to interact with Flapjack (*see* next chapter).

2.2 Installation

1. The LOICA Python package is distributed using the Python Package Index (PyPI), which utilizes the Pip Installs Package (PIP) for installation and update management. The latest stable release of LOICA can be installed using the following commands (*see* **Note 2**):

```
pip install loica
```

2. To verify that the installation was successful, users should be able to run the following command with no errors:

```
import loica as lc
```

3 Methods

In the following sections, we detail how LOICA can be used to generate genetic network designs, graph visualization and simulations. We exemplify its use by designing a NOR gate [16], a repressilator [17], and characterizing an inverter, covering the main use cases. To learn more about the advanced features, we recommend consulting our previous publication [11] and exploring the source repository https://github.com/RudgeLab/LOICA with more examples and tutorials, as well as the documentation https://loica.readthedocs.io.

3.1 Designing a NOR Gate

To create a NOR genetic network design in LOICA, we instantiate two representing Acyl Homoserine Lactone (AHL) Supplements, each of which induce a Receiver Operator that expresses a regulator. These regulators, in turn, repress a Hill2 Operator that expresses a GFP Reporter. This process can be described step by step, as follows:

1. Create a GeneticNetwork (Table 1).

```
nor = lc.GeneticNetwork(vector=0)
```

Table 1
Supplement

Parameter	Description	Format/values	Default
name	Name of the supplement	Str	n/a
concentration	Concentration of the supplement in Molar	int – float	n/a
pubchemid	PubChemID URI of the supplement	Str	n/a
supplier_id	Supplier ID of the supplement. An URL of the product that you acquire. Accepts list of the form [product URL, catalog number, batch].	str – List	n/a
sbol_comp	SBOL component of the supplement.	Str	n/a

Table 2
Genetic network

Parameter	Description	Format/values	Default
operators	List of Operators that are part of the genetic network	List	[]
regulators	List of Regulators that are part of the genetic network	List	[]
reporters	List of Reporters that are part of the genetic network	List	[]
vector	Flapjack ID of the vector that is associated with the genetic network. If none use 0.	int – float	n/a

2. Create Supplements (Table 2) (*see* **Note 3**).

```
ahl1 = lc.Supplement(name=AHL1)

ahl2 = lc.Supplement(name=AHL2)
```

3. Create Regulators (Table 3) and add them to the nor GeneticNetwork. In this example, we use TetR and LacI.

```
tetr_reg = lc.Regulator(name=TetR, degradation_
rate=1)

laci_reg = lc.Regulator(name=LacI, degradation_
rate=1)

nor.add_regulator([tetr_reg, laci_reg])
```

4. Create a Reporter (Table 4) and add it to the nor GeneticNetwork. The example shows the creation of a GFP Reporter.

Table 3
Regulator

Parameter	Description	Format/values	Default
name	Name of the gene product	Str	n/a
init_concentration	Initial concentration of the gene product in Molar	Float	0
degradation_rate	Degradation rate of the gene product	float	0
type_	Molecular type of the gene product, options are 'PRO' (protein) or 'RNA' (RNA)	str, optional	PRO
uri	SynBioHub URI	str, optional	None
sbol_comp	SBOL Component	SBOL Component, optional	None
color	Color displayed on the network representation	str	lightgreen

Table 4
Reporter

Parameter	Description	Format/values	Default
name	Name of the gene product	str	n/a
init_concentration	Initial concentration of the gene product in Molar	int – float	0
degradation_rate	Degradation rate of the gene product	int – float	0
type_	Molecular type of the gene product, can be 'PRO' or 'RNA'	str, optional	PRO
uri	SynBioHub URI	str, optional	None
sbol_comp	SBOL Component	SBOL Component, optional	None
signal_id	Flapjack ID of the Signal that the reporter is associated with.	str, optional	None
color	Color displayed on the network representation	str, optional	white

```
gfp_rep = lc.Reporter(name=GFP, degradation_
rate=1, signal_id=0, color=green)

rep.add_reporter(gfp_rep)
```

5. Create Operators and add them to the nor GeneticNetwork. The example shows the creation of two Receiver Operators (Table 5) and one Hill2 Operator (Table 6), modeling a transcriptional NOR gate.

Table 5
Operator receiver

Parameter	Description	Format/values	Default
input	The input of the operator that regulates the expression of the output	Regulator – Supplement	n/a
output	The output of the operator that is regulated by the input	Regulator – Reporter	n/a
alpha	[Basal expression rate, Regulated expression rate in MEFL/second]	List	n/a
K	Half expression input concentration in Molar	int – float	n/a
n	Hill coefficient, cooperative degree (unitless)	int – float	n/a
uri	SynBioHub URI	str, optional	None
sbol_comp	SBOL Component	SBOL Component, optional	None
name	Name of the operator displayed on the network representation	str, optional	None
color	Color displayed on the network representation	str, optional	skyblue

Table 6
Operator Hill2

Parameter	Description	Format/values	Default
input	The inputs of the operator that regulates the expression of the output	List [Regulator – Supplement]	n/a
output	The output of the operator that is regulated by the input	Regulator – Reporter – List	n/a
alpha	Regulated and unregulated expression rates (*see* LOICA [11])	List	n/a
K	Half expression input concentration in Molar	List	n/a
n	Hill coefficient, cooperative degree (unitless)	int – float	n/a
uri	SynBioHub URI	str, optional	None
sbol_comp	SBOL Component	SBOL Component, optional	None
name	Name of the operator displayed on the network representation	str, optional	None
color	Color displayed on the network representation	str, optional	orange

This design represents a set of three transcriptional units. One transcriptional unit induced by AHL1 that expresses TetR, another transcriptional unit induced by AHL2 that expresses LacI, and finally, a transcriptional unit repressed by LacI and TetR that expresses GFP. All the Operators have parameters to model a Hill equation.

```
ahl1_rec_tetr = lc.Receiver(name=pahl1, input=ahl1,
output=[tetr_reg], alpha=[10000,0.1], K=10, n=2)

ahl2_rec_laci = lc.Receiver(name=pahl2, input=ahl2,
output=laci_reg, alpha=[10000,0.1], K=10, n=2)

tetr_laci_nor_gfp = lc.Hill2(name=NOR, input=[tetr_-
reg, laci_reg], output=gfp_rep, alpha=[10000,
10000,0.1, 0.1], K=[10, 10], n=[2,2])

  nor.add_operator([ahl1_rec_tetr, ahl2_rec_laci,
  tetr_laci_nor_gfp])
```

6. Plot the graph representation for visual inspection of the design (Fig. 1a).

   ```
   plt.figure(figsize=(3.3,3.3),dpi=300)rep.draw()
   ```

7. Alternatively, plot the contracted graph representation for a simplified visual inspection of the design (Fig. 1b).

   ```
   plt.figure(figsize=(3.3,3.3), dpi=100) rep.draw
   (contracted=True)
   ```

8. Create a Metabolism. The example shows the creation of a SimulatedMetabolism (Table 7) based on the Gompertz growth model [18].

   ```
   def growth_rate(t): return lc.metabolism.gom-
   pertz_growth_rate(t, 0.01, 1, 1, 1)

  def biomass(t): return lc.metabolism.gompertz(t,
  0.01, 1, 1, 1)

  metabolism = lc.SimulatedMetabolism(LOICA metab,
  biomass, growth_rate)
   ```

9. Create Samples (Table 8) encapsulating the GeneticNetwork and the Metabolism with different concentration of Supplement. The example shows the creation of multiple Samples containing the nor Genetic Network, the simulated metabolism and AHLs at different concentrations.

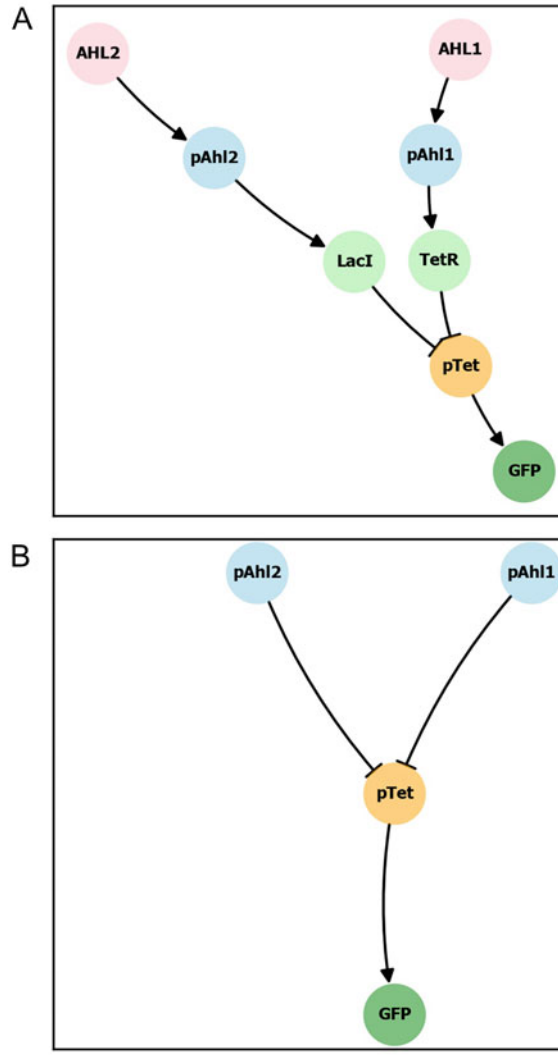

Fig. 1 NOR gate genetic network diagram. (**a**) GeneticNetwork diagram generated by its draw method, which creates a graph representation and plots it as a network. (**b**) Contracted version of the GeneticNetwork graph representation. In these networks, light blue nodes are Hill1 Operators, orange nodes are Hill2 Operators, light green nodes are Regulators, pink nodes are Supplements and Reporters are represented in their assigned color, in this case, green. Regulators are not shown in the contracted version. Pointy arrows represent production or induction and blunt arrows represent repression

```
samples = []

for conc1 in np.logspace(-3, 3, 12):
for conc2 in np.logspace(-3, 3, 12):
sample = lc.Sample(genetic_network=nor,
metabolism=metab)
```

Table 7
SimulatedMetabolism

Parameter	Description	Format/values	Default
name	Name of the metabolism	Str	None
biomass	A function of time that describes biomass f(t) = biomass	Function	n/a
growth_rate	A function of time that describes the growth rate f(t) = growth rate	Function	n/a

Table 8
Sample

Parameter	Description	Format/values	Default
genetic_network	Genetic network that is part of the sample	GeneticNetwork	None
metabolism	Metabolism that drives the genetic network in the sample	Metabolism	None
assay	Assay to which this sample belongs	Assay	None
media	Name of the media in the sample	str, optional	None
strain	Name of the strain in the sample	str, optional	None

```
sample.set_supplement(ahl1, conc1)
sample.set_supplement(ahl2, conc2)
samples.append(sample)
```

10. Create an Assay (Table 9). The example shows the creation of an Assay that takes 100 measurements every 15 min with a name, description, and using OD as the biomass signal.

    ```
    assay = lc.Assay([sample], n_measurements=100,
    interval=0.25, name=LOICA NOR gate, descrip-
    tion=Simulated NOR gate generated by LOICA,
    biomass_signal_id=0
    ```

11. Run the Assay. The example shows how to run an Assay with default parameters; this uses ODEs for simulations (Fig. 2).

    ```
    assay.run()
    ```

3.2 Genetic Ring Oscillator

To create a genetic ring oscillator design in LOICA, similar to the repressilator (*see* **Note 4**), we instantiate three repressor Regulators, each of which represses a Hill1 Operator that expresses another Regulator in a ring fashion, $A \dashv B \dashv C \dashv D$. One of these operators expresses a GFP Reporter in a bicistronic way. This process has some variation for simulations and can be described step by step as follows:

Table 9
Assay

Parameter	Description	Format/values	Default
samples	List of Samples that belongs to the Assay	List[Sample]	n/a
n_measurements	Number of measurements to take	Int	n/a
interval	Time in hours between each measurements	Float	n/a
name	Name of the Assay	Str	LOICA assay
description	Description of the Assay	Str	
biomass_signal_id	Flapjack ID of the Signal measuring culture growth (e.g. OD600)	Int	None

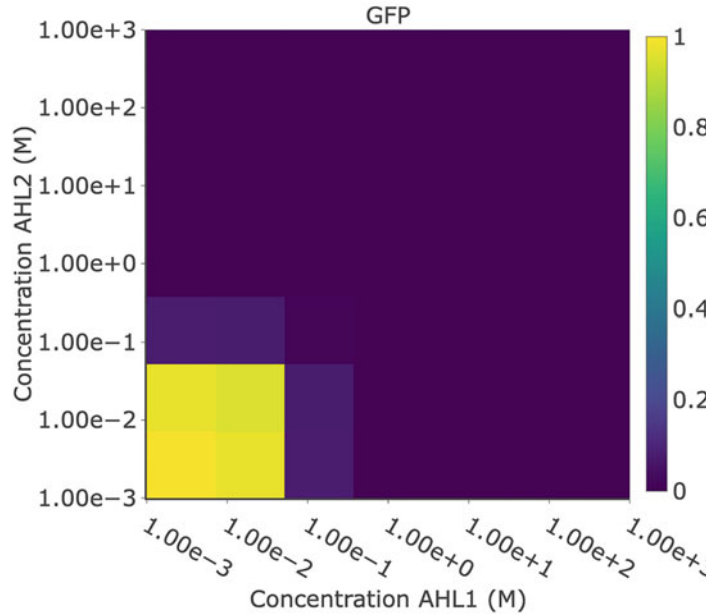

Fig. 2 NOR gate simulation heatmap. NOR gate simulated with a Gompertz metabolism and adding Supplements to Samples in 12 concentrations between −3 and 3 in log space. The Assay runs for 17.5 h measuring every 15 min. High mean expression is shown in yellow and low expression rate is shown in blue

1. Create a GeneticNetwork.

```
repressilator = lc.GeneticNetwork(vector=0)
```

2. Create Regulators and add them to the repressilator GeneticNetwork. In this example, we use TetR, LacI, and CI.

```
tetr_reg = lc.Regulator(name=TetR, degradation_
rate=1)
```

```
laci_reg = lc.Regulator(name=LacI, degradation_
rate=1)

ci_reg = lc.Regulator(name=cI, degradation_ rate=1)

repressilator.add_regulator([tetr_reg, laci_ reg,
ci_reg])
```

3. Create a Reporter and add it to the repressilator GeneticNetwork. The example shows the creation of a GFP Reporter.

```
gfp_rep = lc.Reporter(name=GFP, degradation_rate=1,
signal_id=0, color=green)

rep.add_reporter(gfp_rep)
```

4. Create Operators and add them to the repressilator Genetic Network. The example shows the creation of three Hill1 Operators (Table 10), modeling repressible transcriptional units. A transcriptional unit repressed by LacI that expresses TetR and GFP, a transcriptional unit repressed by TetR expressing CI and a transcriptional unit repressed by CI expressing LacI. All the Operators have parameters to model a Hill equation.

```
laci_not_tetr_gfp = lc.Hill1(name=pLac, input=la-
ci_reg, output=[tetr_reg, gfp_rep], alpha=
[10000,0.1], K=10, n=2)
```

Table 10
Operator Hill1

Parameter	Description	Format/values	Default
input	The input of the operator that regulates the expression of the output	Regulator – Supplement	n/a
output	The output of the operator that is regulated by the input	Regulator – Reporter	n/a
alpha	[Basal expression rate, Regulated expression rate in MEFL/second]	List	n/a
K	Half expression input concentration in Molar	int | float	n/a
n	Hill coefficient, cooperative degree (unitless)	int | float	n/a
uri	SynBioHub URI	str, optional	None
sbol_comp	SBOL Component	SBOL Component, optional	None
name	Name of the operator displayed on the network representation	str, optional	None
color	Color displayed on the network representation	str, optional	skyblue

```
tetr_not_ci = lc.Hill1(name=pTet, input=tetr_reg,
output=ci_reg, alpha=[10000,0.1], K=10, n=2)

ci_not_laci = lc.Hill1(name=pcI, input=ci_reg, out-
put=laci_reg, alpha=[10000,0.1], K=10, n=2)

rep.add_operator([laci_not_tetr_gfp, ci_not_laci,
tetr_not_ci])
```

5. Plot the graph representation for visual inspection of the design (Fig. 3a).

```
plt.figure(figsize=(3.3,3.3), dpi=100) rep.draw
()
```

6. Alternatively, plot the contracted graph representation for a simplified visual inspection of the design (Fig. 3b).

```
plt.figure(figsize=(3.3,3.3), dpi=100)

rep.draw(contracted=True)
```

7. Create a Metabolism. The example shows the creation of a SimulatedMetabolism based on the Gompertz growth model [18].

```
def growth_rate(t): return lc.metabolism.gom-
pertz_growth_rate(t, 0.01, 1, 1, 1)

def biomass(t): return lc.metabolism.gompertz(t,
0.01, 1, 1, 1)

metabolism = lc.SimulatedMetabolism(LOICA metab,
biomass, growth_rate)
```

8. Create a Sample encapsulating the GenticNetwork and the Metabolism. The example shows the creation of a Sample containing the repressilator and metabolism.

```
sample = lc.Sample(genetic_network=repressila-
tor, metabolism=metabolism)
```

9. Create an Assay. The example shows the creation of an Assay that takes 100 measurements every 15 min with a name, description and using OD as the biomass signal.

```
assay = lc.Assay([sample], n_measurements=100,
interval=0.25, name=LOICA repressilator,
description=Simulated repressilator generated
by LOICA, biomass_signal_id=0
```

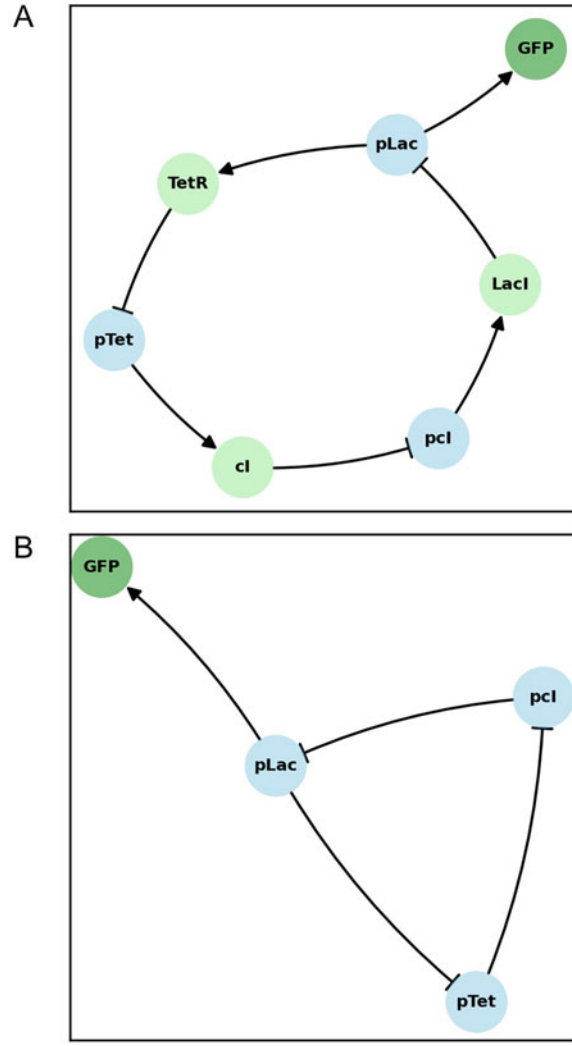

Fig. 3 Simple repressilator genetic network diagram. (**a**) GeneticNetwork diagram generated by its draw method, which creates a graph representation and plots it as a network. (**b**) Contracted version of the GeneticNetwork graph representation. In these networks light blue nodes are Operators, light green nodes are Regulators and Reporters are represented in their assigned color, in this case green. Regulators are not shown in the contracted version. Pointy arrows represent production or induction and blunt arrows represent repression

10. Run the Assay. The example shows how to run an Assay with default parameters, this uses ODEs for simulations (Fig. 4a).

```
assay.run()
```

11. Alternatively, run the Assay with noise. The example shows how to run an Assay with a noise to signal ratio of 10−3; this uses ODEs with noise for simulations (Fig. 4b).

```
assay.run(nsr=1e-3)
```

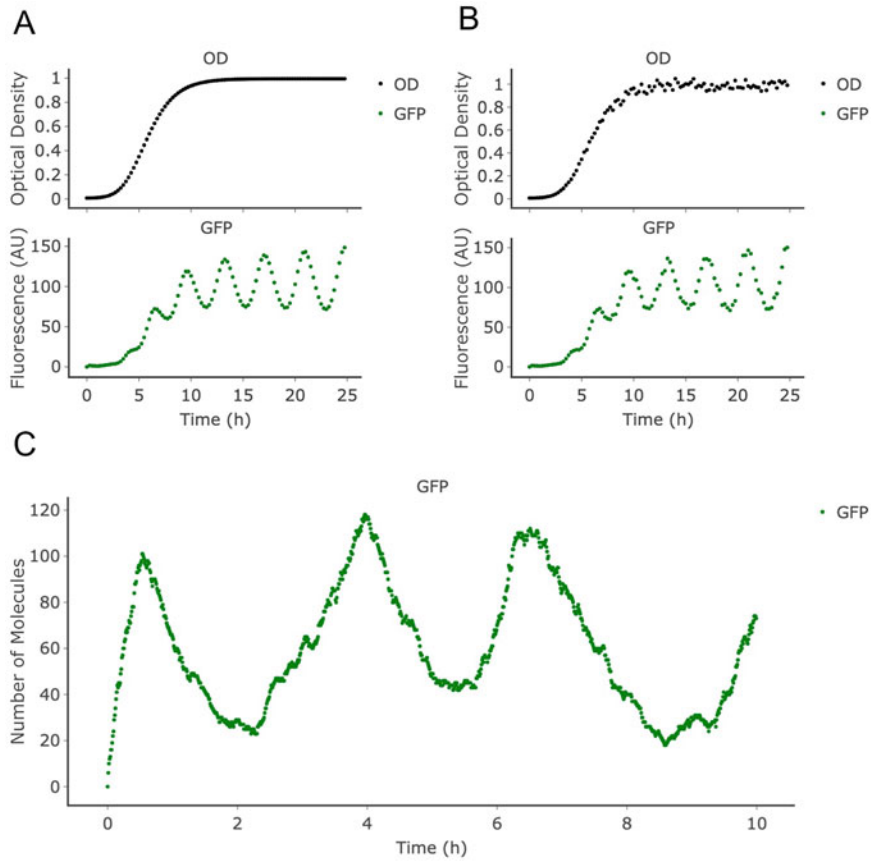

Fig. 4 Genetic ring oscillator simulations. (**a**) Genetic ring oscillator simulation using ordinary differential equations, lines correspond to the signals of OD and green fluorescence over 25 h. (**b**) Genetic ring oscillator simulation using ordinary differential equations with a noise to signal ratio (NSR) of 103, lines correspond to the signals of OD and green fluorescence over 25 h. (**c**) Genetic ring oscillator simulation using a stochastic algorithm, lines correspond to the signals of OD and green fluorescence over 10 h

12. Let us model this genetic network with another metabolism. Create a Metabolism. The example shows the creation of a SimulatedMetabolism with a biomass of 1 and no growth rate.

```
    def growth_rate(t):
return 0

def biomass(t):
return 1

metabolism = lc.SimulatedMetabolism(LOICA metab,
biomass, growth_rate)
```

13. Create a Sample encapsulating the GeneticNetwork and the Metabolism. The example shows the creation of a Sample containing the repressilator and metabolism.

```
sample = lc.Sample(genetic_network=repressila-
tor, metabolism=metabolism)
```

14. Create an Assay. The example shows the creation of an Assay that takes 100 measurements every 15 min with a name, description and using OD as the biomass signal.

```
assay = lc.Assay([sample], n_measure-
ments=1000, interval=1e-2, name=LOICA repres-
silator, description=Simulated repressilator
generated by LOICA, biomass_signal_id=0
```

15. Run the Assay. The example shows how to run an Assay with default parameters, this uses ODEs for simulations (Fig. 4c).

```
assay.run(stochastic=True)
```

3.3 Receiver and Inverter Characterization

To characterize a receiver in LOICA, we connect to experimental data in Flapjack, then instantiate a Receiver Operator (*See* **Note 5**) and use the characterize method. To characterize an inverter in LOICA, we connect to experimental data in Flapjack, then instantiate a Hill1 Operator to characterize and pass a previously characterized Receiver as argument Operator and then a Hill1 Operator. This process can be described step by step, as follows:

1. Log in to your Flapjack account. For more information on Flapjack, *see* the next Chapter.

```
fj = Flapjack(url_base=flapjack.rudge-lab.
org:8000)
```

```
fj.log_in(username=input(Flapjack username: ),
password=getpass.getpass(Password: ))
```

2. Get Flapjack data objects.

```
vector = fj.get(vector, name=pAN1818_cyan)
```

```
yfp = fj.get(signal, name=YFP)
```

```
vector = fj.get(vector, name=pAN1818_cyan)
```

```
media = fj.get(media, name=M9 Glicerol)
```

```
strain = fj.get(strain, name=Top10)
```

```
biomass_signal = fj.get(signal, name=OD))
```

3. Create a receiver GeneticNetwork.

```
receiver = lc.GeneticNetwork(vector=vector.id[0])
```

4. Create a inducer Supplement. In this example, we use IPTG.

```
iptg = lc.Supplement(name=IPTG)
```

5. Create a Reporter and add it to the receiver GeneticNetwork. The example shows the creation of a YFP Reporter.

```
yfp_rep = lc.Reporter(name=YFP, degradation_rate=0, signal_id=yfp.id[0], color=green)

receiver.add_reporter(yfp_rep)
```

6. Create an Operator and add it to the receiver Genetic Network. The example shows the creation of a Receiver Operator.

   This design represents a transcriptional units that gets induced with IPTG and expresses GFP. The Operators have parameters to model a Hill equation.

```
iptg_rec_yfp = lc.Receiver(input=iptg, output=yfp_rep, alpha=[1e-3,1e4], K=1e-5, n=2)

receiver.add_operator(iptg_rec_yfp)
```

7. Plot the graph representation for visual inspection of the receiver design (Fig. 5a left panel).

```
plt.figure(figsize=(3,3), dpi=300)

receiver.draw()
```

8. Use the characterize method. This will fit the experimental data to the Receiver Operator model parametrizing alpha, K and n.

```
iptg_rec_yfp.characterize( fj, vector=vector.id, media=media.id, strain=strain.id, signal=yfp.id, biomass_signal=biomass_signal.id)
```

9. Connect to inverter experimental data.

```
vector2 = fj.get(vector, name=pSrpR-S3_cyan)
```

10. Create an inverter GeneticNetwork.

```
inverter = lc.GeneticNetwork(vector=vector2.id[0])
```

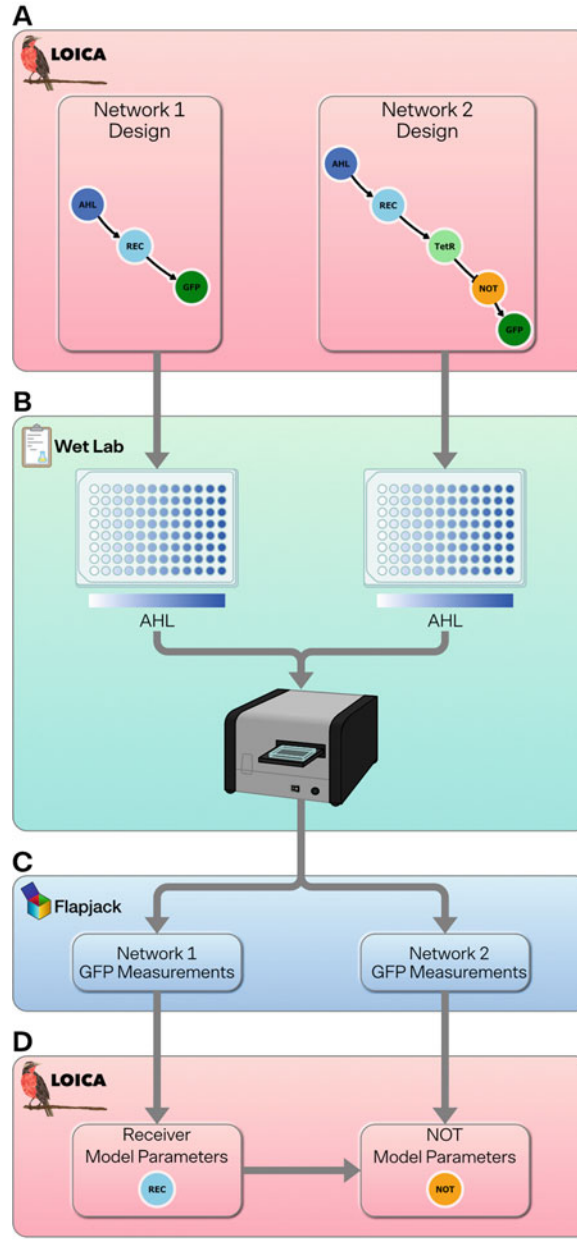

Fig. 5 LOICA inverter characterization workflow. (**a**) To characterize an inverter you need to build two genetic networks, one for the receiver and another connecting the receiver with an NOT gate to for the inverter. (**b**) These genetic networks needs to be measured under different concentration of AHL to drive the expression of the receiver. (**c**) The measurements can be uploaded to Flapjack by it built in parser. (**d**) LOICA can access experimental data in flapjack to characterize the Receiver and then the Hill1 Operator, thus parametrizing an inverter

11. Create a Regulator and add it to the inverter GeneticNetwork. In this example, we use SrpR.

```
srpr_reg = lc.Regulator(SrpR)

inverter.add_regulator(srpr_reg)
```

12. Modify the Receiver from previous steps, create an Operator and add them to the inverter GeneticNetwork. This design represents a set of two transcriptional units, one induced by IPTG that expresses SrpR and another repressed by SrpR that expresses GFP.

```
iptg_rec_srpr = iptg_rec_yfp

iptg_rec_srpr.output = srpr_reg

srpr_not_yfp = lc.Hill1(input=srpr_reg, output=yfp_rep, alpha=[10,1e-3], K=1e3, n=2)

inverter.add_operator([srpr_not_yfp, iptg_rec_srpr])
```

13. Plot the graph representation for visual inspection of the design (Fig. 5a right panel).

```
plt.figure(figsize=(3.3,3.3), dpi=300) inverter.draw()
```

14. Assemble the genetic networks from the designs, prepare samples, and measure them in a serial dilution of inducer (Fig. 5b). The example shows the preparation of two 96-well plates with a gradient of AHL inducer, one with each genetic network. Fluorescence and OD are measured in a plate reader.
15. Collect the experimental data, usually an Excel output, and upload it to Flapjack (Fig. 5d)
16. Use the characterize method. This will get experimental data from Flapjack and fit the Hill1 Operator model, parametrizing alpha, K and n (Fig. 5d). Operators characterized from experimental data can be used to build new genetic networks and simulate their behaviour (*see* **Note 6**).

```
srpr_not_yfp.characterize( fj, receiver=iptg_ rec_yfp, inverter=vector2.id, media=media.id, strain=strain.id, signal=yfp.id, biomass_signal=biomass_signal.id) gamma=0
```

4 Notes

1. A Flapjack account is required to run certain examples; if you do not already have an account, you may register following the instructions in the Flapjack chapter.
2. The version of LOICA used during in this example is 1.0.5.
3. Supplements are added to Sample.
4. The original repressilator constructed by Elowitz [17] has the functional repressilator in one plasmid and the reporter in another. For simplicity, in this design, we get the reporter expressed with one of the repressors in a bicistronic fashion.
5. This code is made using: from flapjack import*.
6. More details can be found in https://github.com/RudgeLab/LOICA/tree/master/notebooks.

References

1. Endy D (2005) Foundations for engineering biology. Nature 438(7067):449
2. Aldulijan I, Beal J, Billerbeck S, Bouffard J, Chambonnier G, Ntelkis N, Guerreiro I, Holub M, Ross P, Selvarajah V et al (2023) Functional synthetic biology.. Synth Biol 8(1):ysad006
3. McLaughlin JA, Beal J, Mısırlı G, Grünberg R, Bartley BA, Scott-Brown J, Vaidyanathan P, Fontanarrosa P, Oberortner E, Wipat A et al (2020) The synthetic biology open language (SBOL) version 3: simplified data exchange for bioengineering. Front Bioeng Biotechnol 8:1009
4. Mitchell T, Beal J, Bartley B (2022) pySBOL3: SBOL3 for python programmers. ACS Synth Biol 11(7):2523
5. McLaughlin JA, Myers CJ, Zundel Z, Mısırlı G, Zhang M, Ofiteru ID, Goni-Moreno A, Wipat A (2018) SynBioHub: a standards-enabled design repository for synthetic biology. ACS Synth Biol 7(2):682
6. Sents Z, Stoughton TE, Buecherl L, Thomas PJ, Fontanarrosa P, Myers CJ (2023) SynBioSuite: a tool for improving the workflow for genetic design and modeling. ACS Synth Biol 12(3):892
7. Crowther M, Wipat A, Goñi-Moreno Á (2022) A network approach to genetic circuit designs. ACS Synth Biol 11(9):3058
8. Jones TS, Oliveira SM, Myers CJ, Voigt CA, Densmore D (2022) Genetic circuit design automation with Cello 2.0. Nat Protoc 17(4):1097
9. Yáñez Feliú G, Earle Gómez B, Codoceo Berrocal V, Muñoz Silva M, Nuñez IN, Matute TF, Arce Medina A, Vidal G, Vitalis C, Dahlin J et al (2020) Flapjack: Data management and analysis for genetic circuit characterization. ACS Synth Biol 10(1):183
10. Samineni SP, Vidal G, Vitalis C, Feliú GY, Rudge TJ, Myers CJ, Mante J (2023) Experimental data connector (XDC): integrating the capture of experimental data and metadata using standard formats and digital repositories. ACS Synth Biol 12(4):1364
11. Vidal G, Vitalis C, Rudge TJ (2022) LOICA: Integrating models with data for genetic network design automation. ACS Synth Biol 11(5):1984
12. Bartley BA, Choi K, Samineni M, Zundel Z, Nguyen T, Myers CJ, Sauro HM (2018) pySBOL: a python package for genetic design automation and standardization. ACS Synth Biol 8(7):1515
13. Yeoh JW, Swainston N, Vegh P, Zulkower V, Carbonell P, Holowko MB, Peddinti G, Poh CL (2021) Synbiopython: an open-source software library for synthetic biology. Synth Biol 6: Article ysab001
14. Chapman B, Chang J (2000) Biopython: Python tools for computational biology. ACM Sigbio Newslett 20(2):15

15. Vidal G, Vitalis C, Muñoz Silva M, Castillo-Passi C, Yáñez Feliú G, Federici F, Rudge TJ (2022) Accurate characterization of dynamic microbial gene expression and growth rate profiles. Synth Biol 7(1):ysac020
16. Tamsir A, Tabor JJ, Voigt CA (2011) Robust multicellular computing using genetically encoded NOR gates and chemical 'wires'. Nature 469(7329):212
17. Elowitz MB, Leibler S (2000) A synthetic oscillatory network of transcriptional regulators. Nature 403(6767):335
18. Zwietering MH, Jongenburger I, Rombouts FM, Van't Riet K (1990) Modeling of the bacterial growth curve. Appl Environ Microbiol 56(6):1875