RCGAToolbox User Guide

Version 1.3

Contents

1. Introduction

1.1. What is RCGAToolbox?

RCGAToolbox is a MATLAB toolbox that contains two real-coded genetic algorithms (RCGAs): the unimodal normal distribution crossover with minimal generation gap (UNDX/MGG) and the real-coded ensemble crossover star with just generation gap (REX^{star}/JGG). The stochastic ranking method is implemented to efficiently handle constrained optimization problems. RCGAToolbox not only provides access to RCGAs but also several useful features for parameter estimation in systems biology. Video tutorials are available on YouTube. RCGAToolbox is licensed under the GNU General Public License v3.0.

This document includes contents of our previous paper [1] with permission from the publisher (Information Processing Society of Japan), including descriptions of RCGAs (Section 6 and Figure 5).

Publications obtained with the help of RCGAToolbox are asked to reference the following paper:
Maeda K, Boogerd FC, Kurata H: **RCGAToolbox: A real-coded genetic algorithm software for parameter estimation of kinetic models**. *bioRxiv* 2021:2021.2002.2015.431062.

1.2. Problem formulation

RCGAs in RCGAToolbox handle the following constrained optimization problem.

$$\text{Minimize} \quad f(\mathbf{x}), \tag{1a}$$

$$\text{Subject to} \quad \mathbf{g}(\mathbf{x}) \le \mathbf{0}, \tag{1b}$$

$$\mathbf{x}^{lb} \le \mathbf{x} \le \mathbf{x}^{ub}, \tag{1c}$$

where $\mathbf{x} = (x_1, x_2, \ldots)$ is the vector with decision variables, f is the objective function, and $\mathbf{g} = (g_1, g_2, \ldots)$ is the vector with constraint functions. A positive value of g_i indicates that the ith constraint is violated. $\mathbf{x}^{lb}$ and $\mathbf{x}^{ub}$ are the lower and upper bounds, respectively. Eq. (1c) defines the search space. Eqs. (1a-c) together represent a constrained optimization problem. In an *unconstrained* optimization problem, Eq. (1b) is omitted.

In the context of parameter estimation, the decision variables ($\mathbf{x}$) indicate kinetic parameters to be estimated, and f evaluates the model's fit to experimental data, whereas g_i's are constraint functions that the kinetic model must satisfy.

2. Installation

2.1. Requirements

- MATLAB R2016a or later.
- C compiler compatible with MATLAB. You can get C compilers at no charge. For details, see Appendix A.2.
- Parallel Computing Toolbox (optional) is required for parallel computation (opts.n_par > 1, see Section 4.1.1). It is not required for sequential computation.
- Optimization Toolbox (optional) is required for local optimization using fmincon (opts.local = 1, see Section 4.1.1). It is not required if the local optimization function is not used.

The third-party tools, IQM Tools, SundialsTB, and libSBML, are included in the distribution of RCGAToolbox. These tools are used by RCGAToolbox: IQM Tools (formerly known as SBToolbox2/SBPD [2, 3]) are used for SBML and a fast simulation (fast_flag = 2, see Section 4.2.1). SundialsTB is used for a fast simulation with CVODE (fast_flag = 1, see Section 4.2.1). libSBML is required for IQM Tools in Linux and macOS. A C compiler compatible with MATLAB is required to install IQM Tools and SundialsTB.

2.2. Installation

Installation tutorials are available on YouTube.

1. Download RCGAToolbox from https://github.com/kmaeda16/RCGAToolbox.
2. Place the directory RCGAToolbox somewhere favorable (e.g. Documents/MATLAB/).
3. Run the installation script **RCGAToolbox_Install.m** under the directory RCGAToolbox/install/. It installs not only RCGAToolbox core components but also IQM Tools, SundialsTB, and libSBML.

2.3. Uninstallation

1. Run the uninstallation script **RCGAToolbox_Uninstall.m** under the directory RCGAToolbox/install/.
2. Delete the directory RCGAToolbox.

2.4. Troubleshooting

RCGAToolbox/install/**RCGAToolbox_Diagnosis.m** is the self-diagnosis script that checks whether the RCGAToolbox is properly installed. It also tests the RCGAToolbox functions that depend on optional toolboxes. For the diagnosis, run **RCGAToolbox_Diagnosis.m** under the directory RCGAToolbox/install/. See also Appendix A for FAQ.

3. Quick start

3.1. General optimization problem

RCGAToolbox/doc/demo/general/**run_QuickStart.m** demonstrates how to run RCGAs using RCGAToolbox. Simply define your problem and assign it to either function **RCGA_UNDXMGG** or **RCGA_REXstarJGG** to solve.

```
RCGAToolbox/doc/demo/general/run_QuickStart.m
```

```matlab
% This script demonstrates how to run a real-coded genetic algorithm to
% solve an example constrained optimization problem.
%
% --------------------- Example Problem ---------------------
% Minimize:
%   f = x(1)^2 + x(2)^2 + ... + x(10)^2
%
% Subject to:
%   g(1) = x(1) * x(2) + 1 <= 0
%   g(2) = x(1) + x(2) + 1 <= 0
%   -5.12 <= x(i) <= 5.12 for all i
%
%
% Global minimum is f = 3, g(1) = 0, g(2) = 0 at x = (-1.618, 0.6180, 0, 0,
% 0, 0, 0, 0, 0, 0) or at x = (0.6180, -1.618, 0, 0, 0, 0, 0, 0, 0, 0)
% -----------------------------------------------------------


clearvars;

% ========= Problem Settings ========= %
problem.n_gene = 10; % Number of Decision Variables
problem.n_constraint = 2; % Number of Constraints
problem.fitnessfun = @Fitness_Example; % Fitness Function
problem.decodingfun = @Decoding_Example; % Decoding Function

% ========== Executing RCGA ========== %
% Results = RCGA_UNDXMGG(problem); % UNDX/MGG
Results = RCGA_REXstarJGG(problem); % REXstar/JGG
```

```
% ======== Convergence Curve ========= %
figure;
plot(Results.Transition.time,Results.Transition.f,'LineWidth',2);
set(gca,'FontSize',10,'FontName','Arial');
title('Convergence Curve');
xlabel('Time (sec)');
ylabel('Objective Function');
```

Executing **run_QuickStart.m** gives you the following messages in the MATLAB Command Window.

```
>> run_QuickStart

===============================================

          RCGA_REXstarJGG by RCGAToolbox

===============================================



------------------- Problem --------------------
         n_gene :  10
   n_constraint :  2
     fitnessfun :  Fitness_Example
    decodingfun :  Decoding_Example
------------------- Options --------------------
   n_population :  300
     n_children :  300
       n_parent :  11
      t_rexstar :  6
 selection_type :  0
             Pf :  0.45
          local :  0
         maxgen :  1000
        maxtime :  600
        maxeval :  Inf
            vtr :  -Inf
          n_par :  1
    output_intvl :  1
   out_transition :  None
         out_best :  None
   out_population :  None
       out_report :  None
interimreportfun :  RCGAdefaultinterimreportfun
   finalreportfun :  RCGAdefaultfinalreportfun
---------------------------------------------


RCGA_REXstarJGG started.
Elapsed Time = 5.748680e-02, Generation = 1, f = 4.451477e+01, phi = 0.000000e+00
Elapsed Time = 1.497601e-01, Generation = 2, f = 2.558670e+01, phi = 0.000000e+00
Elapsed Time = 2.270435e-01, Generation = 3, f = 1.315321e+01, phi = 0.000000e+00
...
Elapsed Time = 3.363103e+01, Generation = 998, f = 3.142870e+00, phi = 0.000000e+00
Elapsed Time = 3.366195e+01, Generation = 999, f = 3.142870e+00, phi = 0.000000e+00
Elapsed  Time  =  3.369350e+01,  Generation  =  1000,  f  =  3.142870e+00,  phi  =
0.000000e+00
Maximum number of generations (maxgen) reached.
```

In the messages, *f* indicates the objective function value, and *phi* indicates the penalty function value given as $\phi(\mathbf{x}) = \sum_{i=1}^{m} \left[\max\left\{0, g_i(\mathbf{x})\right\}\right]^2$ (see Section 1.2 for g_i and **x**). *m* is the number of constraint functions. After the RCGA, a convergence curve appears that shows changes in the objective function over time.

As shown in **run_QuickStart.m**, the following four fields of the structure **problem** need to be set before it can be passed to **RCGA_UNDXMGG** or **RCGA_REXstarJGG**:

- **problem.n_gene**: The number of decision variables.
- **problem.n_constraint**: The number of constraint functions. For unconstrained problems, it is zero.
- **problem.fitnessfun**: The function handle for a fitness function that takes **x** as an input and returns *f* and **g**. For unconstrained problems, it returns only *f*.
- **problem.decodingfun**: The function handle for a decoding function that defines the search space and how to decode genes to **x**.

In **run_QuickStart.m**, an RCGA solves the following constrained optimization problem:

$$\text{Minimize} \quad f(\mathbf{x}) = \sum_{i=1}^{10} x_i^2 \,, \tag{2a}$$

$$\text{Subject to} \quad g_1(\mathbf{x}) = x_1 x_2 + 1 \le 0 \,, \tag{2b}$$

$$g_2(\mathbf{x}) = x_1 + x_2 + 1 \le 0 \,, \tag{2c}$$

$$-5.12 \le x_i \le 5.12 \,, \tag{2d}$$

where f is the objective function and g_i is the i^{th} constraint function. $\mathbf{x}$ is a decision variable vector: $\mathbf{x} = (x_1, x_2, ..., x_{10})$. The fitness function and the decoding function for this constrained optimization problem are given as **Fitness_Example.m** and **Decoding_Example.m**, respectively:

```
RCGAToolbox/doc/demo/general/Fitness_Example.m
```

```
function [f, g] = Fitness_Example(x)
% Fitness_Example is an example of fitness function.
%
% [SYNTAX]
% [f, g] = Fitness_Example(x)
%
% [INPUT]
% x : Decision variables.
%
% [OUTPUT]
% f : Objective function to be minimized.
% g : Constraint functions to be less than zero.
%
% --------------------- Example Problem ---------------------
% Minimize:
%   f = x(1)^2 + x(2)^2 + ... + x(10)^2
%
% Subject to:
%   g(1) = x(1) * x(2) + 1 <= 0
%   g(2) = x(1) + x(2) + 1 <= 0
%   -5.12 <= x(i) <= 5.12 for all i
%
%
% Global minimum is f = 3, g(1) = 0, g(2) = 0 at x = (-1.618, 0.6180, 0, 0,
% 0, 0, 0, 0, 0, 0) or at x = (0.6180, -1.618, 0, 0, 0, 0, 0, 0, 0, 0)
% -----------------------------------------------------------
```

```
f    = sum( x .^ 2 );

g(1) = x(1) * x(2) + 1;
g(2) = x(1) + x(2) + 1;
```

RCGAToolbox/doc/demo/general/Decoding_Example.m

```
function x = Decoding_Example(gene)
% Decoding_Example is an example of decoding function for
% "Fitness_Example.m". "gene" takes values from 0 to 1. The purpose of
% decoding functions is to change the value range, i.e. to decode "gene"
% and return it as x.
%
% [SYNTAX]
% x = Decoding_Example(gene)
%
% [INPUT]
% gene : Encoded decision variables.
%
% [OUTPUT]
% x    : Decoded decision variables.
%
% --------------------- Example Problem ---------------------
% Minimize:
%   f = x(1)^2 + x(2)^2 + ... + x(10)^2
%
% Subject to:
%   g(1) = x(1) * x(2) + 1 <= 0
%   g(2) = x(1) + x(2) + 1 <= 0
%   -5.12 <= x(i) <= 5.12 for all i
%
%
% Global minimum is f = 3, g(1) = 0, g(2) = 0 at x = (-1.618, 0.6180, 0, 0,
% 0, 0, 0, 0, 0, 0) or at x = (0.6180, -1.618, 0, 0, 0, 0, 0, 0, 0, 0)
% -----------------------------------------------------------
```

```
lb = -5.12;
ub =  5.12;

x = gene * ( ub - lb ) + lb;
```

Note that for *unconstrained* optimization problems, fitness functions should return only *f*. In RCGAToolbox, decision variables are internally expressed as a vector with a range of 0–1. The internal decision variable is called a **gene**. Decoding functions take genes as input, decode these to decision variables **x**, and return these. Decoding functions provide flexibility in defining search space: either linear, logarithmic, continuous, or discontinuous.

The functions **RCGA_UNDXMGG** and **RCGA_REXstarJGG** return the structure **Results**, which has four fields: **Transition**, **Best**, **FinalPopulation**, and **end_crit**. The former three has the same information as the transition file, the best individual file, and the final population file, respectively (see Section 4.1.1 for these files). **end_crit** is the exit flag, whereas **end_crit** = 0 indicates that the RCGAs successfully found a solution. A solution is the decision variable vector that provides an objective function value (*f*) less than a user-defined value (called the value to be reached or **vtr**) with all the constraints satisfied ($\mathbf{g}_i \leq 0$). Non-zero **end_crit** indicates the RCGAs finished without finding a solution: **end_crit** = 1, 2, or 3 indicates "the maximal number of generations (maxgen) was reached," "the maximum CPU time (maxtime) was reached," or "the maximum number of fitnessfun evaluations (maxeval) was reached," respectively.

3.2. Parameter estimation in systems biology

RCGAToolbox/doc/demo/PE/**run_QuickStart.m** demonstrates how to use RCGAs for a parameter estimation problem in systems biology. Simply call **RCGA_UNDXMGG_PE** or **RCGA_REXstarJGG_PE** with a kinetic model (**modelfile**), target experimental data (**measurement**), and a decoding function (**decodingfun**).

Model_Example_odefun.m is an ODE file (IQM Tools format) representing a simple kinetic model with two state variables and seven model parameters. **Measurement_Example.csv** is a CSV file that represents the target experimental data. **Decoding_Example.m** is the decoding function.

```
RCGAToolbox/doc/demo/PE/run_QuickStart.m
```

```
% This script demonstrates how to run a real-coded genetic algorithm to
% estimate model parameters in an example kinetic model.
%
% ------------------------ Example Kinetic Model ------------------------
% - INITIAL CONDITION
% X1 = 0
% X2 = 0
%
% - PARAMETERS
% X0 = 0.1
% k1 = 1
% k2 = 1
% k3 = 1
% K2 = 1
% K3 = 1
% rootCompartment = 1
%
% - VARIABLES
% X12 = X1 + X2
%
% - REACTIONS
% v1 = k1 * X0
% v2 = k2 * (X1/rootCompartment) / (K2 + (X1/rootCompartment))
% v3 = k3 * (X2/rootCompartment) / (K3 + (X2/rootCompartment))
%
% - BALANCE
% X1_dot = v1 - v2;
% X2_dot = v2 - v3;
% -----------------------------------------------------------------------


clear mex;
clear all;
close all;

% ========= Problem Settings ========= %
```

```
modelfile = @Model_Example_odefun; % ODE file (IQM Tools format)
decodingfun = @Decoding_Example; % Decoding Function
% measurement = 'Measurement_Example.xls'; % Measurement File (EXCEL format)
measurement = 'Measurement_Example.csv'; % Measurement File (CSV)

% ========== Executing RCGA ========== %
% Results = RCGA_UNDXMGG_PE(modelfile,decodingfun,measurement); % UNDX/MGG
Results = RCGA_REXstarJGG_PE(modelfile,decodingfun,measurement); % REXstar/JGG

% ======== Convergence Curve ========= %
figure;
plot(Results.Transition.time,Results.Transition.f,'LineWidth',2);
set(gca,'FontSize',10,'FontName','Arial');
title('Convergence Curve');
xlabel('Time (sec)');
ylabel('Objective Function');
```

Instead of **RCGA_REXstarJGG** (which is used in Section 3.1), **RCGA_REXstarJGG_PE** is used here, where "PE" stands for parameter estimation. Executing **run_QuickStart.m** provides the following messages in the MATLAB Command Window. This calculation takes 10 min to complete. To abort the running script, press 'Control + C' in the MATLAB Command Window.

```
>> run_QuickStart
Reading Measurement_Example.csv ... Finished.

================================================


          RCGA_REXstarJGG by RCGAToolbox


================================================



------------------- Problem --------------------
         n_gene :  7
    n_constraint :  0
      fitnessfun :  @(x)fitnessfun(x,Simulation,model,mst,simopts)
```

```
     decodingfun :  Decoding_Example
------------------- Options --------------------
    n_population :  300
      n_children :  300
        n_parent :  8
       t_rexstar :  6
  selection_type :  0
              Pf :  0.45
           local :  0
          maxgen :  1000
         maxtime :  600
         maxeval :  Inf
             vtr :  -Inf
           n_par :  1
    output_intvl :  1
  out_transition :  None
        out_best :  None
  out_population :  None
      out_report :  None
interimreportfun                                                                :
@(elapsedTime,generation,problem,opts,Population,best)interimreportfun(elapsedTi
me,generation,problem,opts,Population,best,Simulation,model,mst,simopts)
  finalreportfun :  RCGAdefaultfinalreportfun
------------------------------------------------

RCGA_REXstarJGG started.
Elapsed Time = 2.339310e+00, Generation = 1, f = 6.562991e-03
Elapsed Time = 5.673242e+00, Generation = 2, f = 3.941497e-03
Elapsed Time = 7.961011e+00, Generation = 3, f = 2.229762e-03
...
Elapsed Time = 5.967471e+02, Generation = 290, f = 1.746201e-06
Elapsed Time = 5.987983e+02, Generation = 291, f = 1.746201e-06
Elapsed Time = 6.008096e+02, Generation = 292, f = 1.746201e-06
Maximum CPU time (maxtime) reached.

--- Best parameter set (f = 1.746201e-06) ---
X0 = 2.358726e-01
k1 = 4.199921e-01
```

```
k2 = 1.329753e+00
k3 = 1.318928e+00
K2 = 1.168590e+00
K3 = 1.153963e+00
rootCompartment = 1.179145e+00
```

In the command window, *f* is the objective function value, which is defined as the squared difference between model predictions and experimental data: $f(\mathbf{x}) = \sum_{i=1}^{n_{point}} \sum_{j=1}^{n_{var}} \left[y_{i,j}^{sim}(\mathbf{x}) - y_{i,j}^{exp} \right]^2$, where **x** is the model parameter vector (the decision variables in the terminology of optimization problems). $y_{i,j}^{sim}$ and $y_{i,j}^{exp}$ indicate the simulated and experimental data, respectively. n_{point} and n_{var} are the number of data points and the number of model variables, respectively. Note that by default, the objective function for parameter estimation supports only the *unconstrained* optimization formulation. That is, the objective function returns the squared sum of the differences between simulated data and experimental data as *f* and does not return g_i. The functions **RCGA_UNDXMGG_PE** and **RCGA_REXstarJGG_PE** return the structure **Results** in the same way that **RCGA_UNDXMGG** and **RCGA_REXstarJGG** does.

After starting the RCGAs, a pop-up figure appears which will update during the calculation (Figure 1). In the figure, circles and lines indicate the target experimental data and the current best model behavior, respectively. The model fitting improves as the calculations progress. After the RCGA, a convergence curve appears that shows changes in the objective function over time.

RCGAToolbox/doc/demo/PE/**Measurement_Example.csv** contains the experimental data for this demonstration. The comments in **Measurement_Example.csv** explain how to define experimental data.

```
RCGAToolbox/doc/demo/PE/Measurement_Example.csv
```

```
% Example measurement data file in the format required by IQM Tools
% ======================================================================
%
% This file is based on MeasurementExample.csv from SBPOP PACKAGE.
% A line starting with the percentage character "%" is interpreted as a
% comment line and neglected during the import of the measurement data.

[Name]
```

```
% The following line defines the name of the measurement data.
% It makes sense to keep the name relatively short.
Measurement

[Notes]
Here you can enter information about the experiment and the data.
Experimental conditions, when the measurement was performed, everything you
think will be of use to you later.

[Components]
% A comma separated list, defining the names of the measured components.
% The 'time' component should always be called 'time'!
time,X1,X2

[Componentnotes]
% For each component a note can be defined. The syntax is as follows:
X1: Metabolite 1
X2: Metabolite 2

[Values]
% Finally, here are the data defined for the components. In this example the
% time-vector is defined in the first column (which is not required but makes
% most sense). The time vector consists of all time instants at which
% measurements have been made. If a certain component has not been measured
% at a certain time instant then this can be indicated by 'NaN' or not
% entering a value for this time step.
0,0,0
1,0.064485063,0.025592125
2,0.090888764,0.058378583
3,0.102250309,0.081147069
4,0.107235215,0.094886024
5,0.109406623,0.1025948
6,0.110365043,0.106758366
7,0.110786853,0.108937286
8,0.110970383,0.110043733
9,0.111050028,0.110593817
10,0.11108463,0.1108633
```

4. Tutorial

4.1. Solving general optimization problems

4.1.1. Execution via script files

Having explained the basic usage of RCGAToolbox, we explain its advanced usage in this section. RCGAToolbox/doc/demo/general/**run_Tutorial.m** is the script to solve the same constrained optimization problem as that in Section 3.1 but with different options.

```
RCGAToolbox/doc/demo/general/run_Tutorial.m
```

```
% This script demonstrates how to run a real-coded genetic algorithm to
% solve an example constrained optimization problem.
% ...



clearvars;

% ========= Problem Settings ========= %
problem.n_gene = 10; % Number of Decision Variables
problem.n_constraint = 2; % Number of Constraints
problem.fitnessfun = @Fitness_Example; % Fitness Function
problem.decodingfun = @Decoding_Example; % Decoding Function

% ========= Option Settings ========== %
opts.n_population = 100; % Population Size
opts.n_children = 100; % Number of Children per Generation
opts.n_parent = problem.n_gene + 1; % Number of Parents Used for REXstar
opts.t_rexstar = 6.0; % Step-size Parameter for REXstar
opts.selection_type = 0; % Parameter for JGG (0: Chosen from Children, 1: Chosen
from Family)
opts.Pf = 0.45; % Pf
opts.local = 0; % Local Optimizer
% opts.localopts =
optimoptions(@fmincon,'ConstraintTolerance',0,'MaxFunctionEvaluations',opts.n_ch
ildren,'Display','off'); % Options for Local Optimizer
opts.maxgen = 1000; % Max Number of Generations
opts.maxtime = 60; % Max Time (sec)
opts.maxeval = inf; % Max Number of fitnessfun Evaluations
opts.vtr = 0; % Value To Be Reached
opts.n_par = 1; % Number of Workers for Parallel Computation
opts.output_intvl = 10; % Output Interval Generation
opts.out_transition = 'Transition.txt'; % Transition File Name
opts.out_best = 'BestIndividual.txt'; % Best Individual File Name
opts.out_population = 'FinalPopulation.txt'; % Final Population File Name
opts.out_report = 'Report.mat'; % Report File Name
opts.interimreportfun = @RCGAdefaultinterimreportfun; % Interim Report Function
opts.finalreportfun = @RCGAdefaultfinalreportfun; % Final Report Function

% ======= Setting Random Seed ======== %
rng(0); % Random Seed

% ========== Executing RCGA ========== %
% Results = RCGA_UNDXMGG(problem,opts); % UNDX/MGG
Results = RCGA_REXstarJGG(problem,opts); % REXstar/JGG

% ======== Convergence Curve ========= %
figure;
plot(Results.Transition.time,Results.Transition.f,'LineWidth',2);
set(gca,'FontSize',10,'FontName','Arial');
title('Convergence Curve');
xlabel('Time (sec)');
ylabel('Objective Function');
```

As shown in **run_Tutorial.m**, the following fields of the structure **opts** can be set:

- **opts.n_population**: Population size, that is, the number of individuals in the population. (Default: 300)
- **opts.n_children**: Number of children that are generated per generation. (Default: **opts.n_population**)
- **opts.n_parent**: Number of parents that are used to generate the children. This option is active only for REX^{star}/JGG. (default: **problem.n_gene** + 1)
- **opts.t_rexstar**: Step size parameter. This option is only active for REX^{star}/JGG. (Default: 6.0)
- **opts.selection_type**: Selection type. **opts.selection_type** must be 0 or 1. If it is 0, a certain number (**opts.n_parent**) of the best children are selected. If it is 1, a certain number (**opts.n_parent**) of individuals in the family (children and parents) are selected. This option is only active for REX^{star}/JGG. (Default: 0)

- **opts.Pf**: Probability that only the objective function *f* is used for comparisons of individuals in the stochastic ranking. This option is only active for constrained optimization problems (**problem.n_constraint** > 0). The recommended value is $0.4 < P_f < 0.5$. (Default: 0.45)
- **opts.local**: Local optimizer. **opts.local** must be 0 or 1. If it is 1, the local optimizer is used: when the best individual is updated after generation alternation by UNDX/MGG or REX^{star}/JGG, the local optimizer is called to further improve the best individual. Whether the use of the local optimizer improves the performance of RCGAs depends on the specific problems. Generally, in problems with multiple local optima, the local optimizer does not work well, and thus, its use leads to a higher computational cost. Note that Optimization Toolbox is required for the local optimizer. (Default: 0)
- **opts.localopts**: Options for the local optimizer. The options are active only when **opts.local** = 1. (Default: optimoptions(@fmincon, 'ConstraintTolerance', 0, 'MaxFunctionEvaluations', opts.n_children, 'Display', 'off'); Moreover, "UseParallel" is set to "true" if **opts.n_par** > 1)
- **opts.maxgen**: Number of maximum generations. RCGAs end once they reach **opts.maxgen**. (Default: 1000)
- **opts.maxtime**: Maximum time (sec). RCGAs end when they reach **opts.maxtime**. (Default: 600)
- **opts.maxeval**: Maximum number of fitness function evaluations. RCGAs end when they reach **opts.maxeval**. (Default: Inf)
- **opts.vtr**: Value to be reached. RCGAs end when they reach **opts.vtr**. (Default: –Inf)
- **opts.n_par**: Number of workers in parallel computation. In parallel computation, each worker evaluates **opts.n_population/opts.n_par** individuals in the first generation and **opts.n_ children/opts. n_par** individuals in the second generation to the final generation. Thus, the more workers, the fewer individuals to be evaluated per worker, leading to faster progression of RCGAs. Note that Parallel Computing Toolbox is required for parallel computation (**opts.n_par** > 1). (Default: 1)
- **opts.output_intvl**: The number of generations after which the transition file and the report file are each time updated. When this option is properly applied, users can avoid generating huge output files. (Default: 1)
- **opts.out_transition**: Name of an output file called the transition file. The transition file is a text file in which each row corresponds to the best individual in each generation. The columns show the computational time (*Time*), the number of fitness function evaluations (*NEval*), generation (*Generation*), objective function value (*f*), penalty function (*phi*), decision variables (*x*), and constraint function values (*g*). (Default: 'None')
- **opts.out_best**: Name of an output file called the best individual file. The best individual file is a text file that is generated at the end of RCGAs and contains the information on the best individual found in the computation. It also contains the same columns as the transition file. (Default: 'None')

- **opts.out_population**: Name of an output file called the final population file. The final population file is a text file that is generated at the end of RCGAs and contains information on all individuals in the final population. Each row corresponds to each individual. The columns are the objective function value (*f*), penalty function (*phi*), decision variables (*x*), and constraint function values (*g*). (Default: 'None')
- **opts.out_report**: Name of an output file called the report file. The report file is a MATLAB MAT file in which the structure **Results** is stored. The structure **Results** is the same as the structure **Results** returned by **RCGA_UNDXMGG** and **RCGA_REXstarJGG** at the end of the RCGAs. The structure contains four fields: **Transition**, **Best**, **FinalPopulation**, and **end_crit**. The field **Transition** is updated at each **opts.interval** generation, while the other fields are generated at the end of the RCGAs. (Default: 'None')
- **opts.interimreportfun**: Function handle of the interim report function called at each **opts.output_intvl** generation. The function **RCGAdefaultinterimreportfun** can be used as a template for custom interim report functions. (Default: @RCGAdefaultinterimreportfun)
- **opts.finalreportfun**: Function handle of the final report function called at the end of RCGAs. The function **RCGAdefaultfinalreportfun** can be used as a template for custom final report functions. (Default: @RCGAdefaultfinalreportfun)

In **run_Tutorial.m**, the function **rng** is used for reproducibility. If **rng** is not called before starting RCGAs, the results will change each time users run RCGAs. Executing **run_Tutorial.m** provides similar messages to **run_QuickStart.m** in the MATLAB Command Window. Four output files (Transition.txt, BestIndividual.txt, FinalPopulation.txt, and Report.mat) are generated in the directory RCGAToolbox/doc/demo/general/.

4.1.2. Execution via GUI

For those unfamiliar with codes or scripts, the RCGAToolbox has graphical user interfaces (GUIs). To start the GUI "RCGAToolbox Mission Control", execute **run_GUI.m** under the directory RCGAToolbox/doc/demo/general/, or alternatively type "RCGAToolbox_MissionControl" in the MATLAB Command Window. Using the GUI, users can specify problems and options (Figure 2). To run RCGAs, click the "Launch" button.

Most of the items in the GUI correspond to the structures **problem** and **opts** explained above. The "Save" button saves the current settings in a MAT file. The "Load" button allows loading the settings from a created MAT file. By clicking the "Reset" button, all the settings in the GUI can be reset. If you are not familiar with RCGAs, you can use recommended settings by clicking the "Use Recommended Values" button. By clicking the "Create" button, the GUI creates an executable script (a file similar to

run_Tutorial.m) with the current settings. We recommend that beginners first use the GUI. After getting comfortable with RCGAs, they can create the executable script by the "Create" button and modify it for advanced use.

4.2. Solving parameter estimation problems in systems biology

4.2.1. Execution via script files

RCGAToolbox/doc/demo/PE/**run_Tutorial.m** is another script to solve the parameter estimation problem found in Section 3.2 but with different options.

RCGAToolbox/doc/demo/PE/run_Tutorial.m

```
% This script demonstrates how to run a real-coded genetic algorithm to
% estimate model parameters in an example kinetic model.
% ...


clear mex;
clear all;
close all;

% ========= Problem Settings ========= %
% modelfile = 'Model_Example_SBML.xml'; % SBML file (IQM Tools required)
% modelfile = IQMmodel('Model_Example_SBML.xml'); % Creating an IQMmodel object
(IQM Tools required)
modelfile = @Model_Example_odefun; % ODE file (IQM Tools format)
% modelfile = 'Model_Example_odefun.m'; % ODE file (IQM Tools format)
% modelfile = 'Model_Example_mex.c'; % C source code (IQM Tools required)
% modelfile = 'Model_Example_mex.mexw64'; % MEX model for Windows
% modelfile = 'Model_Example_mex.mexmaci64'; % MEX model file for macOS
% modelfile = 'Model_Example_mex.mexa64'; % MEX model file for Linux
decodingfun = @Decoding_Example; % Decoding Function
% measurement = 'Measurement_Example.xls'; % Measurement File (EXCEL format)
measurement = 'Measurement_Example.csv'; % Measurement File (CSV)

% ========= Option Settings ========== %
opts.n_population = 50; % Population Size
opts.n_children = 25; % Number of Children per Generation
opts.n_parent = 7 + 1; % Number of Parents Used for REXstar
opts.t_rexstar = 6.0; % Step-size Parameter for REXstar
opts.selection_type = 0; % Parameter for JGG (0: Chosen from Children, 1: Chosen
from Family)
opts.local = 0; % Local Optimizer
% opts.localopts =
optimoptions(@fmincon,'ConstraintTolerance',0,'MaxFunctionEvaluations',opts.n_ch
ildren,'Display','off'); % Options for Local Optimizer
opts.maxgen = 200; % Max Number of Generations
opts.maxtime = 60; % Max Time (sec)
opts.maxeval = inf; % Max Number of fitnessfun Evaluations
opts.vtr = 0; % Value To Be Reached
opts.n_par = 1; % Number of Workers for Parallel Computation
opts.output_intvl = 1; % Output Interval Generation
opts.out_transition = 'Transition.txt'; % Transition File Name
opts.out_best = 'BestIndividual.txt'; % Best Individual File Name
opts.out_population = 'FinalPopulation.txt'; % Final Population File Name
opts.out_report = 'Report.mat'; % Report File Name
fast_flag = 0; % fast_flag (0: MATLAB ODEXX)
% fast_flag = 1; % fast_flag (1: SundialsTB CVODE) (SundialsTB required)
% fast_flag = 2; % fast_flag (2: IQM Tools CVODE MEX) (IQM Tools required)

% ======= Setting Random Seed ======== %
rng(0); % Random Seed

% ========== Executing RCGA ========== %
% Results =
RCGA_UNDXMGG_PE(modelfile,decodingfun,measurement,fast_flag,[],opts); % UNDX/MGG
Results =
RCGA_REXstarJGG_PE(modelfile,decodingfun,measurement,fast_flag,[],opts); %
REXstar/JGG

% ======== Convergence Curve ========= %
figure;
plot(Results.Transition.time,Results.Transition.f,'LineWidth',2);
set(gca,'FontSize',10,'FontName','Arial');
title('Convergence Curve');
xlabel('Time (sec)');
```

```
ylabel('Objective Function');
```

The following three are the inputs specific for parameter estimation:

- **modelfile**: Name of a model file or IQMmodel object. The model file can be an SBML file, an ODE file (IQM Tools format), a C source code, or a MEX model. The IQMmodel object is a MATLAB object used in IQM Tools. (For details, see the user guide of IQM Tools).
- **measurement**: Measurement file that provides experimental data to be fitted by the model. The measurement file can be a CSV file or an XLS file. See **Measurement_Example.csv** and **Measurement_Example.xls** for the IQM measurement format. These files can be used as templates for user-defined measurement files.
- **fast_flag**: Input specifying which ODE solver is used. For **fast_flag** = 0, a built-in MATLAB solver is used. For **fast_flag** = 1, a pre-compiled ODE solver (CVODE) provided by SundialsTB is used. For **fast_flag** = 2, a pre-compiled ODE solver (CVODE) provided by IQM Tools is used with a MEX model.

4.2.2. Execution via GUI

RCGAToolbox has a GUI for parameter estimation (Figure 3). To start the GUI "RCGAToolbox Mission Control PE", execute **run_GUI.m** in the directory RCGAToolbox/doc/demo/PE/, or type "RCGAToolbox_MissionControl_PE" in the MATLAB Command Window. Most input fields shown in the GUI are the same as those in the GUI for general use (see Section 4.1.2). Additional input fields are **modelfile**, **measurement**, and **fast_flag**.

4.3. Simulating kinetic models

RCGAToolbox/doc/demo/sim/**run_SimulateModel.m** demonstrates how to simulate a kinetic model. A kinetic model can be provided as an SBML file, an IQMmodel object, an ODE file (IQM Tools format), a C source code, or a MEX model (In **run_SimulateModel.m**, you can comment/uncomment the lines starting with "model ="). An IQMmodel object is an object that represents a kinetic model in IQM Tools (for details, see the user guide of IQM Tools). The function **RCGAsimulate** simulates the model with an ODE solver specified by **fast_flag** (you can comment/uncomment the lines starting with "fast_flag ="). When **fast_flag** = 0, MATLAB built-in ODE solvers are used. When **fast_flag** = 1, a pre-compiled ODE solver provided by SundialsTB is used. When **fast_flag** = 2, the model is compiled by IQM Tools and combined with a pre-compiled ODE solver. As **run_SimulateModel.m** is executed, a figure pops up, which shows the model behavior. The function **RCGAsimulate** is useful because it checks the model format and **fast_flag**, and then automatically chooses an appropriate ODE solver.

RCGAToolbox/doc/demo/sim/run_SimulateModel.m

```
% This script demonstrates how to run simulation using RCGAToolbox.



clear mex;
clear all;
close all;

% =============== Model =============== %
% model = 'Model_Example_SBML.xml'; % SBML file (IQM Tools required)
% model = IQMmodel('Model_Example_SBML.xml'); % Creating an IQMmodel object (IQM
Tools required)
model = @Model_Example_odefun; % ODE file (IQM Tools format)
% model = 'Model_Example_odefun.m'; % ODE file (IQM Tools format)
% model = 'Model_Example_mex.c'; % C source code (IQM Tools required)
% model = 'Model_Example_mex.mexw64'; % MEX model for Windows
% model = 'Model_Example_mex.mexmaci64'; % MEX model file for macOS
% model = 'Model_Example_mex.mexa64'; % MEX model file for Linux

% =============== Time ================ %
tspan = 0 : 0.1 : 10;

% ========= Initial Condition ========= %
y0(1) = 0; % X1
y0(2) = 0; % X2

% ========= Parameter Values ========== %
param(1) = 0.1; % X0
param(2) = 1;   % k1
param(3) = 1;   % k2
param(4) = 1;   % k3
param(5) = 1;   % K2
param(6) = 1;   % K3
param(7) = 1;   % rootCompartment

% ============ ODE Solver ============= %
fast_flag = 0; % # fast_flag (0: MATLAB ODEXX)
```

```
% fast_flag = 1; % # fast_flag (1: SundialsTB CVODE) (SundialsTB required)
% fast_flag = 2; % # fast_flag (2: IQM Tools CVODE MEX) (IQM Tools required)

% ============ Simulation ============= %
[ T, Y ] = RCGAsimulate(model, tspan, y0, param, fast_flag);

% =============== Figure ============== %
figure;
plot(T,Y,'-','LineWidth',2);
legend('X_1','X_2','Location','best');
xlabel('Time');
ylabel('Concentration');
```

5. Useful features

5.1. Model format conversion

RCGAToolbox/doc/demo/conversion/**run_Conversion.m** demonstrates how to convert an IQMmodel object, an SBML file, an ODE file, a concise ODE file, a C source code, and a MEX model into one another. RCGAToolbox, combined with IQM Tools, provides different conversion functions, as summarized in Figure 4. The script **run_Conversion.m** creates different model files from the original model file **Model_Example_conciseOdefun.m**, which is written in the concise ODE file format as shown below.

RCGAToolbox/doc/demo/conversion/Model_Example_conciseOdefun.m

```
function dydt = Model_Example_conciseOdefun(t, y)
% Model_Example_conciseOdefun is an example of the concise ODE file
% (RCGAToolbox format) ready for conversion into an IQMmodel object by
% RCGAreadConciseODEfile. In this file, the only sections sandwiched
% between "BEGIN" and "END" are used for the conversion.
%
% [SYNTAX]
% dydt = Model_Example(t, y)
%
% [INPUT]
% t    :  Time.
% y    :  X1 and X2.
%
% [OUTPUT]
% dydt :  dX1/dt and dX2/dt.



X1 = y(1);
X2 = y(2);

% === BEGIN NAME ===
% Model_Example
% === END NAME ===

% === BEGIN NOTES ===
% Simple metabolic pathway with two Michaelis-Menten rate equations.
% === END NOTES ===
% === BEGIN INITIAL CONDITION ===
X1_0 = 0;
X2_0 = 0;
% === END INITIAL CONDITION ===

% === BEGIN PARAMETERS ===
X0 = 0.1;
k1 = 1;
k2 = 1;
k3 = 1;
K2 = 1;
K3 = 1;
% === END PARAMETERS ===

% === BEGIN VARIABLES ===
X12 = X1 + X2;
% === END VARIABLES ===

% === BEGIN REACTIONS ===
v1 = k1 * X0;
v2 = k2 * X1 / ( K2 + X1 );
v3 = k3 * X2 / ( K3 + X2 );
% === END REACTIONS ===

% === BEGIN BALANCE ===
X1_dot = v1 - v2;
X2_dot = v2 - v3;
% === END BALANCE ===

dydt = zeros(2,1);
dydt(1) = X1_dot;
dydt(2) = X2_dot;
```

The concise ODE file format is simpler than the ODE file format used in IQM Tools (RCGAToolbox/doc/demo/sim/**Model_Example_odefun.m** is written in the IQM Tools ODE file format). The concise ODE file can be used for MATLAB ODE solvers and yet ready for conversion to SBML. Users can easily implement their models in the concise ODE file format, and once done, those models can be automatically converted into IQMmodel objects or SBML files.

5.2. Wrapper for the fast ODE solver CVODE

RCGAToolbox/doc/demo/stb/**run_ODESTB.m** demonstrates how to use the fast ODE solver CVODE. In general, the ODE solver CVODE provided by SundialsTB is faster than MATLAB's built-in ODE solvers such as ode15s and ode23s [4]. However, the use of CVODE is more complicated than that of MATLAB's built-in ODE solvers. The function **odestb** provided by RCGAToolbox is a wrapper for CVODE and enables the use of CVODE in the same way as the built-in ODE solvers are used. SundialsTB is required to run **run_ODESTB.m**.

```
RCGAToolbox/doc/demo/stb/run_ODESTB.m
```

```
% This script demonstrates how to use CVODE by SundialsTB. SundialsTB is
% required to use the function odestb.



clear mex;
clear all;
close all;



% =============== Time ================ %
tspan = 0 : 0.1 : 10;

% ========= Initial Condition ========= %
y0(1) = 0; % X1
y0(2) = 0; % X2

% ============ Simulation ============= %
tic;
% [ T, Y ] = ode45(@Model_Example_conciseOdefun, tspan, y0); % MATLAB Built-in
% [ T, Y ] = ode23(@Model_Example_conciseOdefun, tspan, y0); % MATLAB Built-in
% [ T, Y ] = ode113(@Model_Example_conciseOdefun, tspan, y0); % MATLAB Built-in
% [ T, Y ] = ode15s(@Model_Example_conciseOdefun, tspan, y0); % MATLAB Built-in
% [ T, Y ] = ode23s(@Model_Example_conciseOdefun, tspan, y0); % MATLAB Built-in
% [ T, Y ] = ode23t(@Model_Example_conciseOdefun, tspan, y0); % MATLAB Built-in
% [ T, Y ] = ode23tb(@Model_Example_conciseOdefun, tspan, y0); % MATLAB Built-in
[ T, Y ] = odestb(@Model_Example_conciseOdefun, tspan, y0); % CVODE provided by
SundialsTB
toc

% ============== Figure ============== %
figure;
plot(T,Y,'-','LineWidth',2);
legend('X_1','X_2','Location','best');
xlabel('Time');
ylabel('Concentration');
```

5.3. Using PEtab-style parameter and measurement files

Recently, the parameter estimation problem definition format PEtab, was proposed to improve interoperability among parameter estimation tools [5]. RCGAToolbox/doc/demo/PEtab/**run_PEtab.m** demonstrates how to use PEtab-style parameter and measurement files for RCGAToolbox.

```
RCGAToolbox/doc/demo/PEtab/run_PEtab.m
```

```
% This script shows how to use a PEtab parameter file and a PEtab
% measurement file for RCGAToolbox. This script works as follows.
%
% 1. RCGAcreateDecodingfun creates a decoding function from a
%    PEtab parameter file. RCGAToolbox uses a decoding function to define
%    parameter ranges and scaling.
%
% 2. RCGAcreateMeasurement creates an IQM measurement file from a PEtab
%    measurement file. For parameter estimation, RCGAToolbox uses an IQM
%    measurement file to specify experimental data to be fitted.
%
% 3. RCGA_UNDXMGG_PE or RCGA_REXstarJGG_PE runs with a model file (e.g. an
%    SBML file), the created decoding function, and the created IQM
%    measurement file.
%
% ------------------------ Example Kinetic Model ------------------------
% - INITIAL CONDITION
% X1 = 0
% X2 = 0
%
% - PARAMETERS
% X0 = 0.1
% k1 = 1
% k2 = 1
% k3 = 1
% K2 = 1
% K3 = 1
% rootCompartment = 1
%
% - VARIABLES
% X12 = X1 + X2
%
% - REACTIONS
% v1 = k1 * X0
% v2 = k2 * (X1/rootCompartment) / (K2 + (X1/rootCompartment))
% v3 = k3 * (X2/rootCompartment) / (K3 + (X2/rootCompartment))
%
% - BALANCE
% X1_dot = v1 - v2;
% X2_dot = v2 - v3;
% ----------------------------------------------------------------------


clear mex;
clear all;
close all;

% ============== Model =============== %
modelfile = 'Model_Example_SBML.xml'; % SBML file (IQM Tools required)
% modelfile = IQMmodel('Model_Example_SBML.xml'); % Creating an IQMmodel object
(IQM Tools required)
% modelfile = @Model_Example_odefun; % ODE file (IQM Tools format)
% modelfile = 'Model_Example_odefun.m'; % ODE file (IQM Tools format)
% modelfile = 'Model_Example_mex.mexw64'; % MEX model for Windows
% modelfile = 'Model_Example_mex.mexmaci64'; % MEX model file for macOS
% modelfile = 'Model_Example_mex.mexa64'; % MEX model file for Linux

% ==== Parameter File Conversion (PEtab --> RCGAToolbox Decoding Function)
==== %
PEtabParameterFile = 'PEtab_Parameter.tsv';    % Parameter file in PEtab format
DecodingfunFile    = 'Created_Decondingfun.m'; % RCGAToolbox decoding function
RCGAcreateDecodingfun(PEtabParameterFile,DecodingfunFile);
[~, funcname, ~] = fileparts(DecodingfunFile);
decodingfun = str2func(funcname); % Function handle

% ==== Measurement File Conversion (PEtab --> IQM Tools) ==== %
PEtabMeasurementFile = 'PEtab_Measurement.tsv';     % Measurement file in PEtab
format
IQMmeasurementFile   = 'Created_IQMmeasurement.csv'; % Measurement file in IQM
Tools format
RCGAcreateMeasurement(PEtabMeasurementFile,IQMmeasurementFile,modelfile);

% ========= Option Settings ========== %
opts.n_population = 50; % Population Size
opts.n_children = 25; % Number of Children per Generation
opts.n_parent = 7 + 1; % Number of Parents Used for REXstar
opts.t_rexstar = 6.0; % Step-size Parameter for REXstar
opts.selection_type = 0; % Parameter for JGG (0: Chosen from Children, 1: Chosen
from Family)
opts.local = 0; % Local Optimizer
% opts.localopts =
optimoptions(@fmincon,'ConstraintTolerance',0,'MaxFunctionEvaluations',opts.n_ch
ildren,'Display','off'); % Options for Local Optimizer
opts.maxgen = 200; % Max Number of Generations
opts.maxtime = 60; % Max Time (sec)
opts.maxeval = inf; % Max Number of fitnessfun Evaluations
opts.vtr = 0; % Value To Be Reached
opts.n_par = 1; % Number of Workers for Parallel Computation
opts.output_intvl = 1; % Output Interval Generation
opts.out_transition = 'Transition.txt'; % Transition File Name
opts.out_best = 'BestIndividual.txt'; % Best Individual File Name
opts.out_population = 'FinalPopulation.txt'; % Final Population File Name
opts.out_report = 'Report.mat'; % Report File Name
fast_flag = 0; % fast_flag (0: MATLAB ODEXX)
% fast_flag = 1; % fast_flag (1: SundialsTB CVODE) (SundialsTB required)
% fast_flag = 2; % fast_flag (2: IQM Tools CVODE MEX) (IQM Tools required)

% ======= Setting Random Seed ======== %
rng(0); % Random Seed

% ========== Executing RCGA ========== %
% Results =
RCGA_UNDXMGG_PE(modelfile,decodingfun,IQMmeasurementFile,fast_flag,[],opts); %
UNDX/MGG
Results =
RCGA_REXstarJGG_PE(modelfile,decodingfun,IQMmeasurementFile,fast_flag,[],opts);
% REXstar/JGG

% ======== Convergence Curve ========= %
figure;
plot(Results.Transition.time,Results.Transition.f,'LineWidth',2);
set(gca,'FontSize',10,'FontName','Arial');
title('Convergence Curve');
xlabel('Time (sec)');
ylabel('Objective Function');
```

PEtab_Parameter.tsv and **PEtab_Measurement.tsv** are a PEtab parameter file and PEtab measurement file, respectively.

```
RCGAToolbox/doc/demo/PEtab/PEtab_Parameter.tsv
```

parameterId	parameterScale	lowerBound	upperBound	nominalValue	estimate
X0	lin			0.1	0
k1	lin	0	10	1	1
k2	lin	0	10	1	1
k3	log10	0.1	10	1	1
K2	log10	0.1	10	1	1
K3	log10	0.1	10	1	1
rootCompartment	lin			1	0

```
RCGAToolbox/doc/demo/PEtab/PEtab_Measurement.tsv
```

```
 observableId  simulationConditionId  measurement      time
           X1      normal_simulation            0         0
           X1      normal_simulation     0.064485         1
           X1      normal_simulation     0.090889         2
           X1      normal_simulation      0.10225         3
           X1      normal_simulation     0.107235         4
           X1      normal_simulation     0.109407         5
           X1      normal_simulation     0.110365         6
           X1      normal_simulation     0.110787         7
           X1      normal_simulation      0.11097         8
           X1      normal_simulation      0.11105         9
           X1      normal_simulation     0.111085        10
           X2      normal_simulation            0         0
           X2      normal_simulation     0.025592         1
           X2      normal_simulation     0.058379         2
           X2      normal_simulation     0.081147         3
           X2      normal_simulation     0.094886         4
           X2      normal_simulation     0.102595         5
           X2      normal_simulation     0.106758         6
           X2      normal_simulation     0.108937         7
           X2      normal_simulation     0.110044         8
           X2      normal_simulation     0.110594         9
           X2      normal_simulation     0.110863        10
```

RCGAToolbox requires a decoding function file and an IQM measurement file for the search parameter settings and measurement data, respectively. RCGAToolbox has functions to create these necessary files from a PEtab parameter file and PEtab measurement file. The script **run_PEtab.m** works as follows:

1. The function **RCGAcreateDecodingfun** creates a decoding function (**Created_Decondingfun.m**) from a PEtab parameter file (**PEtab_Parameter.tsv**).
2. The function **RCGAcreateMeasurement** creates an IQM measurement file (**Created_IQMmeasurement.csv**) from a PEtab measurement file (**PEtab_Measurement.tsv**).
3. The function **RCGA_REXstarJGG_PE** initiates parameter estimation with a model file (**Model_Example_SBML.xml**), the created decoding function (**Created_Decondingfun.m**), and the created IQM measurement file (**Created_IQMmeasurement.csv**).

Note that RCGAToolbox currently assumes that the measurement data in a PEtab measurement file have been collected in a single experiment.

5.4. Implementing custom RCGAs

RCGAToolbox enables users to efficiently implement their custom RCGAs [or other evolution computation algorithms such as the evolution strategy [6], differential evolution [7], and scatter search [8]. Users need to provide a user-defined generation alternation function that defines how to reproduce new individuals (children) and how to update the population. The script **run_CustomRCGA.m** demonstrates how to run user-defined custom RCGAs.

```
RCGAToolbox/doc/demo/CustomRCGA/run_CustomRCGA.m
```

```matlab
% This script demonstrates how to run a user-defined custom real-coded
% genetic algorithm to solve an example constrained optimization problem.
% ...


clearvars;

% ========= Problem Settings ========= %
problem.n_gene = 10; % Number of Decision Variables
problem.n_constraint = 2; % Number of Constraints
problem.fitnessfun = @Fitness_Example; % Fitness Function
problem.decodingfun = @Decoding_Example; % Decoding Function

% ========== Executing RCGA ========== %
% GenerationAlternation_Example is a user-defined custom generation
% alternation function
Results = RCGA_CustomRCGA(problem, @GenerationAlternation_Example);

% ======== Convergence Curve ========= %
figure;
plot(Results.Transition.time,Results.Transition.f,'LineWidth',2);
set(gca,'FontSize',10,'FontName','Arial');
title('Convergence Curve');
xlabel('Time (sec)');
ylabel('Objective Function');
```

As you can see, the function **RCGA_CustomRCGA** receives the structure **problem** and the function handle for a generation alternation function. If necessary, users can provide **RCGA_CustomRCGA** with the structure **opts**. **GenerationAlternation_Example.m** is a sample generation alternation function and can be used as a template for custom generation alternation functions.

```
RCGAToolbox/doc/demo/CustomRCGA/GenerationAlternation_Example.m
```

```
function Population = GenerationAlternation_Example(problem, opts, Population)
% GenerationAlternation_Example updates population by a simple generation
% alternation algorithm. This function can be used as a template for custom
% generation alternation functions.
% ...


%% Shortening variable names
n_population = opts.n_population;
n_children = n_population - 1; % Note that opts.n_children is not used.
n_constraint = problem.n_constraint;
n_gene = problem.n_gene;
Pf = opts.Pf;


%% Generating new children
ip = randperm(n_population,2);

c(1,1:n_children) =
struct('gene',zeros(1,n_gene),'g',zeros(1,n_constraint),'f',zeros,'phi',zeros);

for i = 1 : n_children

    % Crossover
    c(i).gene = 0.5 * ( Population(ip(1)).gene + Population(ip(2)).gene );

    % Mutation
    mutation_rate = 0.1;
    for j = 1 : n_gene
        if rand < mutation_rate
            c(i).gene(j) = rand;
        end
    end

end
c = RCGArequestFitnessCalc(problem,opts,c);



%% Making a new population
Population = [ Population(1) c ];
Population = RCGAsrsort(Population,Pf);
```

The input arguments are the structure **problem**, structure **opts**, and structure array **Population**. An element of **Population** is the structure **individual** that has the vector **gene**, vector **g**, scaler **f**, and scaler **phi** as fields. The function **GenerationAlternation_Example** produces new individuals (children) based on the given population. Primitive crossover and mutation methods are used for the sake of simplicity. **GenerationAlternation_Example** keeps the best individual in the population and replaces the other individuals with the children. The predefined function **RCGArequestFitnessCalc** calculates values of the objective function (*f*), constraint functions (*g*), and penalty function (*phi*) for the children. The predefined function **RCGAsrsort** sorts individuals in the population according to the stochastic ranking sort algorithm.

6. Real-coded genetic algorithms

Genetic algorithms (GAs) are metaheuristic techniques developed with inspiration from the evolution of living organisms [9, 10]. The basic procedure is as follows:

1. Generate an initial population in which each individual is characterized by a set of different values for the decision variables.
2. Select a subset of individuals (called parents) from the population.
3. Generate children using the selected parents (outside the population).
4. Select children that show good fitness (i.e., featuring small values of the objective function).
5. Replace the parents in the population with the same number of selected children, thereby creating a partly changed population, while maintaining the total number of individuals.
6. If a termination criterion is not met by the new population, return to step 2.

The above procedure is terminated when a sufficiently good individual is obtained or the number of generations reaches the predefined maximal number. Various GAs have been proposed so far, and they differ in their implementation for each step. For parameter estimation in systems biology, real-coded GAs (RCGAs) are used, in which a set of kinetic parameters (i.e., a real number vector) is handled as an individual in steps 1-6. In parameter estimation, a good fitness in step 4 means good fitting to the experimental data for training the model.

In constrained optimization problems, it is essential to balance the objective function and the constraint functions for the appropriate ranking of individuals. To achieve a well-balanced ranking, the stochastic ranking method based on the bubble-sort-like procedure was proposed (Figure 5). For details on constraint handling, see section 2.3.3 of our previous study [1].

6.1. UNDX/MGG

UNDX/MGG employs unimodal normal distribution crossover (UNDX) [11] as the crossover method and minimal generation gap (MGG) [12] as the generation alternation method. The UNDX and MGG algorithms are described below. Here, n_v is the number of decision variables, and n_c is the number of children. In the UNDX, there are two parameters: α and β. We followed the original recommendations [11] of $\alpha = 0.5$ and $\beta = 0.35$. It should be noted that UNDX uses three parents, but MGG was originally developed for two-parent-based crossover. To combine MGG and UNDX, the subparent ($\mathbf{x}_3$) in UNDX is randomly chosen from the population but not treated as a parent in MGG.

The algorithm of MGG is described as follows:

1. Randomly select two individuals from the population.
2. Using these two individuals (hereafter "parents"), create n_c children.
3. Calculate the objective function f and the penalty function ϕ for the children.
4. Replace the two parents in the population with the best individual and a randomly-chosen individual from the "family." The family consists of parents and children.

The algorithm of UNDX is described as follows:

1. Two main parents $\mathbf{x}_1$ and $\mathbf{x}_2$ and a single sub parent $\mathbf{x}_3$ are used for the crossover.
2. Calculate the midpoint of $\mathbf{x}_1$ and $\mathbf{x}_2$: $\mathbf{x}_p = (\mathbf{x}_1 + \mathbf{x}_2)/2$.
3. Calculate the vector from $\mathbf{x}_1$ to $\mathbf{x}_2$: $\mathbf{d} = \mathbf{x}_2 - \mathbf{x}_1$.
4. Calculate the distance (D) between $\mathbf{x}_3$ and the primary search line (the line passing through $\mathbf{x}_1$ and $\mathbf{x}_2$).
5. Generate a child $\mathbf{x}_c$ using the equation below:

$$\mathbf{x}_c = \mathbf{x}_p + \xi\mathbf{d} + D\sum_{i=1}^{n_v-1}\eta_i\mathbf{e}_i \text{ ,} \quad (3)$$

where $\xi \sim N\left(0,\sigma_\xi^2\right)$, $\eta_i \sim N\left(0,\sigma_\eta^2\right)$, and $\mathbf{e}_i$ is the ith orthogonal basis vector. $\sigma_\xi = \alpha$ and $\sigma_\eta = \beta / \sqrt{n_v}$. Multiple children can be generated by applying Eq. (3) multiple times. $N(x, y)$ is the normal random distribution with the mean x and standard deviation y.

6.2. REX$^{\text{star}}$/JGG

REX$^{\text{star}}$/JGG employs real-coded ensemble crossover star (REX$^{\text{star}}$) as the crossover method and just generation gap (JGG) as the generation alternation method [13]. The algorithms of REX$^{\text{star}}$ and JGG are described below. Here, n_v is the number of decision variables, n_p is the number of parents, n_c is the number of children, and t is the step-size parameter. t determines the aggressiveness of the search. $n_p \leq n_c$ must be satisfied to maintain a constant population size. In the original study, $n_p = n_v + 1$ is recommended, and t is set between 2.5 and 15 [13].

The algorithm of JGG is described as follows:

1. Randomly select n_p individuals from the population.
2. Using these selected individuals (hereafter "parents"), create n_c children.
3. Calculate the objective function f and the penalty function ϕ for the children.
4. Replace the parents in the population with the best n_p children.

The algorithm of REX$^{\text{star}}$ is described as follows:

1. n_p parents $\mathbf{x}_1$, $\mathbf{x}_2$, ..., $\mathbf{x}_{n_p}$ are used for the crossover.
2. Generate the reflection points $\mathbf{x}'_1$, $\mathbf{x}'_2$, ..., $\mathbf{x}'_{n_p}$ of the parents using Eqs. (4) with the center of gravity $\mathbf{x}_g$:

$$\mathbf{x}_g = \frac{1}{n_p}\sum_{i=1}^{n_p}\mathbf{x}_i\ , \qquad (4a)$$

$$\mathbf{x}'_i = 2\mathbf{x}_g - \mathbf{x}_i\ . \qquad (4b)$$

3. Calculate the objective function f and the penalty function ϕ for the reflection points.
4. Select the best n_p individuals from the parents and their reflection points.
5. Compute the center of gravity ($\mathbf{x}_b$) of the selected individuals.
6. Generate a child $\mathbf{x}_c$ using the equation below:

$$\mathbf{x}_c = \mathbf{x}_g + \mathrm{diag}(\xi_1^t, \cdots, \xi_{n_v}^t)\cdot(\mathbf{x}_b - \mathbf{x}_g) + \sum_{i=1}^{n_p}\xi^i(\mathbf{x}_i - \mathbf{x}_g), \qquad (5)$$

where $\xi_i^t \sim U(0,t)$ (i = 1, ..., n_v), and $\xi^i \sim U\left(-\sqrt{3/n_p}, \sqrt{3/n_p}\right)$ (i = 1, ..., n_p). Multiple children can be generated by applying Eq. (5) multiple times. $U(x, y)$ is the uniform random distribution with the lower bound x and the upper bound y.

Appendix A. FAQ

A.1. I cannot run demo scripts.

If the RCGAToolbox does not work properly, run the diagnosis script **RCGAToolbox_Diagnosis.m** to verify that the core components and third-party tools are installed. If SBML-related functions or the fast ODE solver option (**fast_flag** = 2) are not available, it is likely that IQM Tools or libSBML are not properly installed. If the fast ODE solver option (**fast_flag** = 1) is not available, it is likely that SundialsTB is not properly installed.

A.2. I cannot install IQM Tools and SundialsTB.

The MATLAB **mex** command is used to install the third-party tools, IQM Tools and SundialsTB. In order to use the **mex** command, you need a C compiler compatible with MATLAB. To check whether **mex** is ready for use on your computer, type “mex -setup” in the MATLAB Command Window. If you get a message like “MEX configured to use 'gcc' for C language compilation.”, the **mex** is ready. If not, install one of C compilers compatible with MATLAB. Among them, MinGW, GCC, and Xcode are available at no charge for Windows, Linux, and Mac, respectively. We successfully installed IQM Tools and SundialsTB, with Microsoft Visual C++ 2015 Professional, MinGW, GCC and Xcode.

A.3. SBML-related functions are not working on my Mac.

If you successfully installed RCGAToolbox core components, IQM Tools, and libSBML without any errors, and SBML-related functions are not working, it is likely that macOS is blocking libSBML. In that case, you need to approve libSBML (OutputSBML.mexmaci64 and TranslateSBML.mexmaci64) from the Security & Privacy Panel. This video tutorial might help.

Appendix B. List of RCGAToolbox functions

To use the functions provided in the RCGAToolbox, type "help *function_name*" in the MATLAB Command Window. The RCGAToolbox has the functions shown below.

B.1. RCGAToolbox/source/RCGA

RCGAToolbox/source/RCGA/shared

- RCGA_Main

RCGAToolbox/source/RCGA/shared/sort

- RCGAsrsort
- RCGAswap

RCGAToolbox/source/RCGA/shared/IO

- RCGAprintTransition
- RCGAprintWelcomeMessage
- RCGAwriteBest
- RCGAwritePopulation
- RCGAwriteTransition

RCGAToolbox/source/RCGA/shared/misc

- RCGAcheckInputs
- RCGAfindBest
- RCGAgetFitness
- RCGAgetInitPopulation
- RCGArequestFitnessCalc
- ScriptInitTransition
- ScriptStoreBestAndFinalPopulation
- ScriptStoreTransition

RCGAToolbox/source/RCGA/shared/local

- cst_wrapper
- obj_wrapper
- RCGAlocalOptimize

RCGAToolbox/source/RCGA/UNDXMGG

- RCGA_MGG
- RCGA_UNDX
- RCGA_UNDXMGG
- RCGAgetNewChild

RCGAToolbox/source/RCGA/REXstarJGG

- RCGA_JGG
- RCGA_REXstar
- RCGA_REXstarJGG

RCGAToolbox/source/RCGA/misc

- RCGA_CustomRCGA
- RCGAdefaultfinalreportfun
- RCGAdefaultinterimreportfun

B.2. RCGAToolbox/source/PE

- RCGA_PE
- RCGA_REXstarJGG_PE
- RCGA_UNDXMGG_PE

RCGAToolbox/source/PE/sim

- odestb
- RCGAsimulate
- RCGAsimulateMEX
- RCGAsimulateODE
- RCGAsimulateSTB
- wrapper_odefun

RCGAToolbox/source/PE/misc

- RCGAcreateConciseODEfile
- RCGAcreateDecodingfun
- RCGAcreateMeasurement
- RCGAcreateODEfile
- RCGAmakeMEXmodel
- RCGAplotter
- RCGAreadConciseODEfile

- RCGAreplaceWords
- RCGAsensitivity
- RCGAssr

RCGAToolbox/source/PE/IO

- RCGAinterimreportfun_PE

B.3. RCGAToolbox/source/GUI

- RCGAToolbox_MissionControl
- RCGAToolbox_MissionControl_PE
- RCGAcreateExecutableScript
- RCGAcreateExecutableScript_PE

Figures

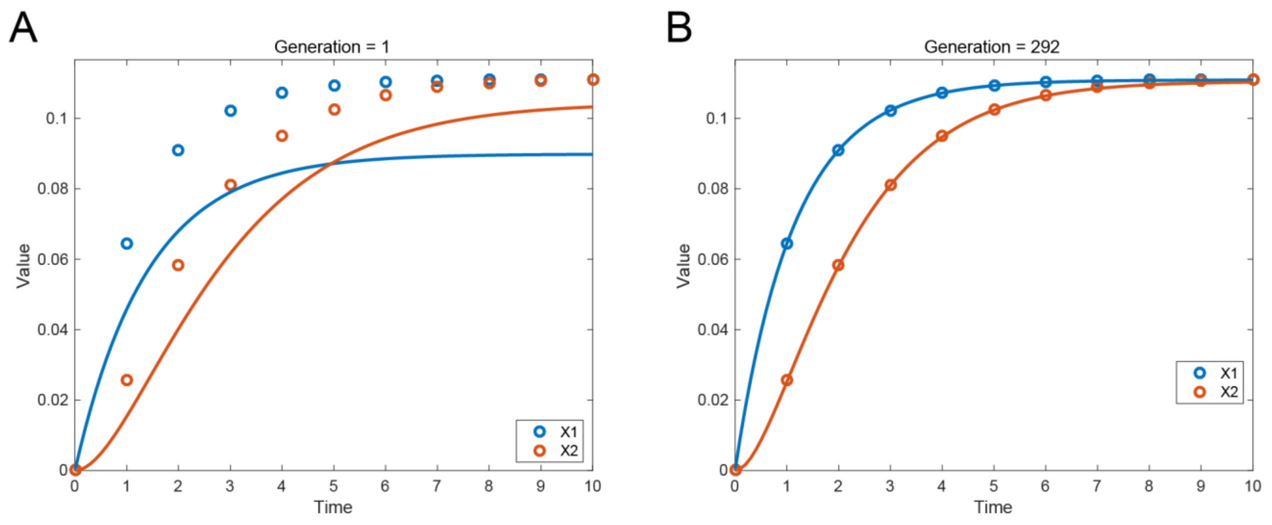
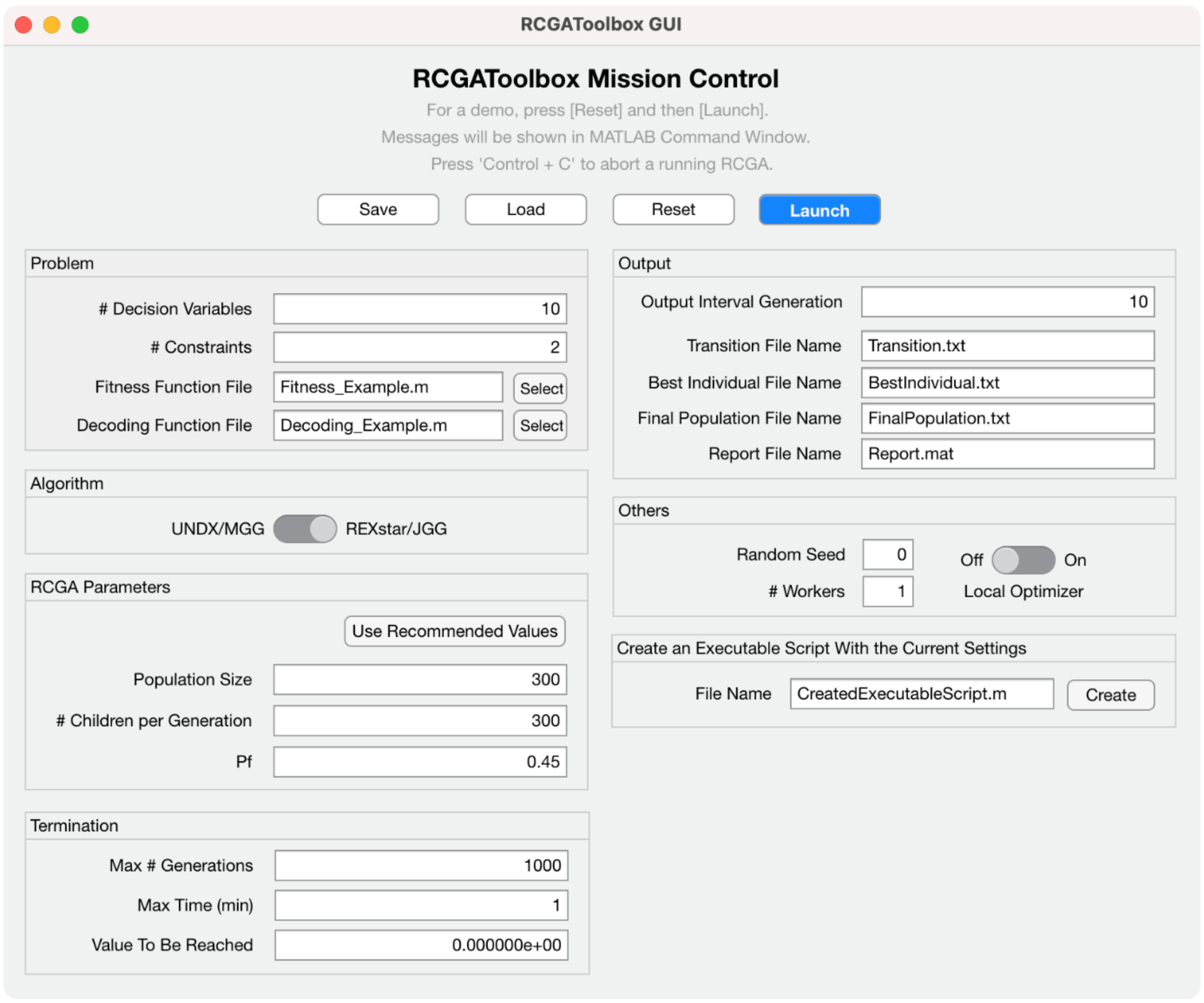


Figure 1. Examples of figures that appear during parameter estimation

A figure pops up just after starting **RCGA_UNDXMGG_PE** or **RCGA_REXstarJGG_PE**, and it will be updated during parameter estimation. The circles and the lines indicate the target experimental data and the current best model behavior, respectively. (A) The behavior of the best parameter set at the 1st generation. (B) The behavior of the best parameter set at the 292nd generation. (Back to Text)


RCGAToolbox GUI

RCGAToolbox Mission Control

For a demo, press [Reset] and then [Launch].
Messages will be shown in MATLAB Command Window.
Press 'Control + C' to abort a running RCGA.

Problem

# Decision Variables	10
# Constraints	2
Fitness Function File	Fitness_Example.m
Decoding Function File	Decoding_Example.m

Algorithm

UNDX/MGG REXstar/JGG

RCGA Parameters

Population Size	300
# Children per Generation	300
Pf	0.45

Termination

Max # Generations	1000
Max Time (min)	1
Value To Be Reached	0.000000e+00

Output

Output Interval Generation	10
Transition File Name	Transition.txt
Best Individual File Name	BestIndividual.txt
Final Population File Name	FinalPopulation.txt
Report File Name	Report.mat

Others

Random Seed	0
# Workers	1

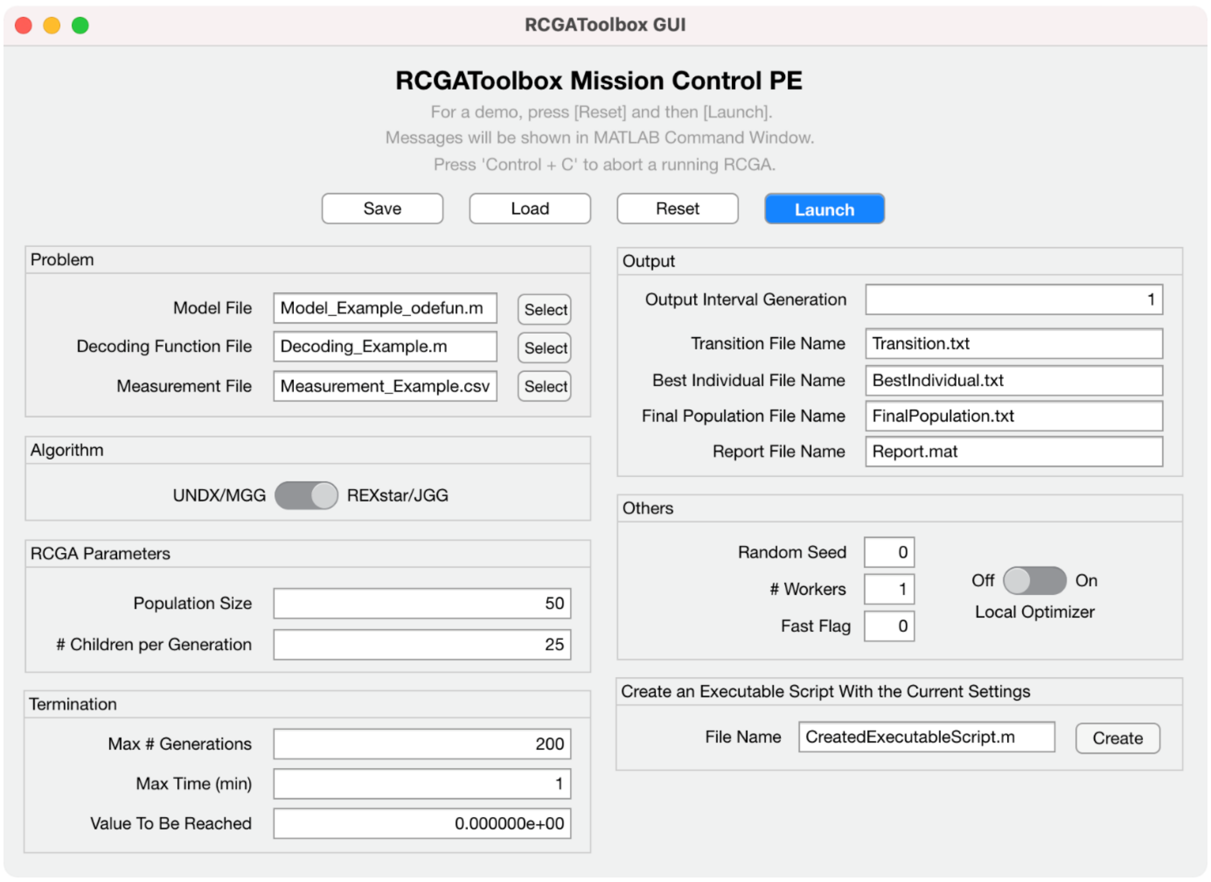
Off On
Local Optimizer

Create an Executable Script With the Current Settings

File Name: CreatedExecutableScript.m

Figure 2. GUI for general optimization problems

(Back to Text)


RCGAToolbox GUI

RCGAToolbox Mission Control PE

For a demo, press [Reset] and then [Launch].
Messages will be shown in MATLAB Command Window.
Press 'Control + C' to abort a running RCGA.

Problem

Model File	Model_Example_odefun.m	Select
Decoding Function File	Decoding_Example.m	Select
Measurement File	Measurement_Example.csv	Select

Algorithm

UNDX/MGG REXstar/JGG

RCGA Parameters

Population Size	50
# Children per Generation	25

Termination

Max # Generations	200
Max Time (min)	1
Value To Be Reached	0.000000e+00

Output

Output Interval Generation	1
Transition File Name	Transition.txt
Best Individual File Name	BestIndividual.txt
Final Population File Name	FinalPopulation.txt
Report File Name	Report.mat

Others

Random Seed	0
# Workers	1
Fast Flag	0

Off On
Local Optimizer

Create an Executable Script With the Current Settings

File Name CreatedExecutableScript.m Create

Figure 3. GUI for parameter estimation in systems biology

(Back to Text)

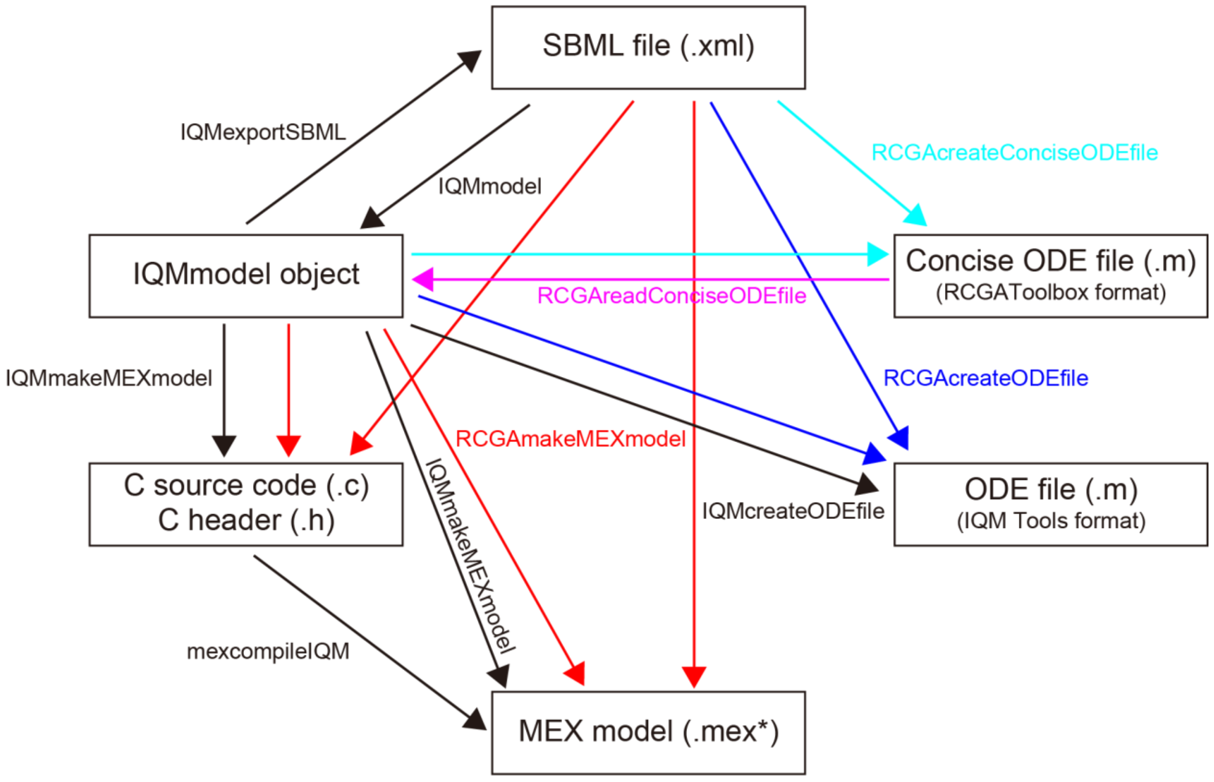


Figure 4. Model formats and conversion functions

The rectangles and arrows indicate the model formats and conversion functions, respectively. For example, the function RCGAreadConciseODEfile converts a concise ODE file into an IQM model object. The conversion functions highlighted in red, blue, cyan, and magenta are provided by RCGAToolbox. Those indicated by black arrows are provided by IQM Tools. (Back to Text)

```
for $i = 1$ to $n_{ps}$ do
    for $j = 1$ to $n_{ps} - 1$ do
        Sample $u \in U(0,1)$
        if $\left[\phi(\mathbf{x}_j) = \phi(\mathbf{x}_{j+1})\right]$ or $\left[u < P_f\right]$ then
            if $\left[f(\mathbf{x}_j) > f(\mathbf{x}_{j+1})\right]$ then
                swap$(\mathbf{x}_j, \mathbf{x}_{j+1})$
            fi
        else
            if $\left[\phi(\mathbf{x}_j) > \phi(\mathbf{x}_{j+1})\right]$ then
                swap$(\mathbf{x}_j, \mathbf{x}_{j+1})$
            fi
        fi
    od
    if no swap done
        break
    fi
od
```

Figure 5. Stochastic ranking using a bubble-sort-like procedure

x_j indicates the jth-ranked individual (i.e., decision variable vector). The initial ranking is generated randomly. $U(0, 1)$ is a uniform random number generator, and n_{ps} is the population size. P_f specifies the probability that only the objective function f is used to compare individuals when they have different values of the penalty function ϕ. It has been demonstrated that the stochastic ranking works well when $0.4 < P_f < 0.5$ [14]. (Back to Text)

References

1. Maeda K, Boogerd FC, Kurata H: **libRCGA: a C library for real-coded genetic algorithms for rapid parameter estimation of kinetic models**. *IPSJ Transactions on Bioinformatics* 2018, **11**(0):31-40.

2. Schmidt H, Jirstrand M: **Systems Biology Toolbox for MATLAB: a computational platform for research in systems biology**. *Bioinformatics* 2006, **22**(4):514-515.

3. Schmidt H: **SBaddon: high performance simulation for the Systems Biology Toolbox for MATLAB**. *Bioinformatics* 2007, **23**(5):646-647.

4. Maeda K, Boogerd FC, Kurata H: **RCGAToolbox: A real-coded genetic algorithm software for parameter estimation of kinetic models**. *bioRxiv* 2021:2021.2002.2015.431062.

5. Schmiester L, Schalte Y, Bergmann FT, Camba T, Dudkin E, Egert J, Frohlich F, Fuhrmann L, Hauber AL, Kemmer S *et al*: **PEtab-Interoperable specification of parameter estimation problems in systems biology**. *PLoS Comput Biol* 2021, **17**(1):e1008646.

6. Runarsson TP, Yao X: **Search biases in constrained evolutionary optimization**. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews* 2005, **35**(2):233-243.

7. Takahama T, Sakai S: **Constrained optimization by the ε constrained differential evolution with an archive and gradient-based mutation**. In: *IEEE Congress on Evolutionary Computation: 2010; Barcelona, Spain*. 1680-1688.

8. Egea JA, Balsa-Canto E, Gracia M-SG, Banga JR: **Dynamic Optimization of Nonlinear Processes with an Enhanced Scatter Search Method**. *Industrial & Engineering Chemistry Research* 2009, **48**(9):4388-4401.

9. Sun JY, Garibaldi JM, Hodgman C: **Parameter Estimation Using Metaheuristics in Systems Biology: A Comprehensive Review**. *IEEE/ACM Trans Comput Biol Bioinform* 2012, **9**(1):185-202.

10. Reali F, Priami C, Marchetti L: **Optimization Algorithms for Computational Systems Biology**. *Frontiers in Applied Mathematics and Statistics* 2017, **3**(6).

11. Ono I, Kobayashi S: **A real-coded genetic algorithm for function optimization using unimodal normal distribution crossover**. *Proc of 7th Int Conf on Genetic Algorithms* 1997:246-253.

12. Satoh H, Yamamura M, Kobayashi S: **Minimal generation gap model for GAs considering both exploration and exploitation**. *Proc of Int Conf on Fuzzy Logic, Neural Networks and Soft Computing* 1997:494-497.

13. Kobayashi S: **The Frontiers of Real-coded Genetic Algorithms**. *J Jpn Soc Artif Intell* 2009, **24**(1):147-162.

14. Runarsson TP, Yao X: **Stochastic ranking for constrained evolutionary optimization**. *IEEE Transactions on Evolutionary Computation* 2000, **4**(3):284-294.