Possibilistic Flux Analysis Toolbox v1

User's manual

May, 2016

Contents

Introduction

Metabolic flux analysis (MFA) is a widely used procedure to estimate the metabolic fluxes within living cells. All MFA-wise methods combine a model with a set of measured fluxes to estimate those that are unknown.

However, traditional MFA methods have limitations. Particularly in scenarios of uncertainty —which are indeed common—, where measurements are scarce and imprecise. The methods implemented in this toolbox, Interval MFA and Possibilistic MFA, have been developed to face those scenarios.

Interval MFA represents the measurements as intervals and provides interval estimates. This way we can deal with imprecise measurements and provide reliable estimates even if our knowledge is incomplete (Llaneras & Picó, 2007; 2008). The method is simple and reliable. Interval MFA has been used by several groups in the last years (D'Huys,2010; Iyer, 2010; Tortajada, 2010; Zamorano, 2010; Hope, 2011; Lorh,2014).

Possibilistic MFA is an extension of Interval MFA. Instead of using intervals, the known fluxes can be represented with a possibilistic distribution. And it provides interval and distributions as estimates. These possibilistic estimates are also reliable in uncertain scenarios, as interval ones, but richer and more informative (Llaneras & Pico, 2009; Tortajada, 2012, Morales, 2014).

Herein, we present the **PFA Toolbox for MATLAB**, which provide a set of functions to easily apply Interval MFA and Possibilistic MFA. The main features of PFA Toolbox are the following:

» Provides reliable MFA estimations in uncertain (or underdetermined) scenarios, where only a few fluxes can be measured.

» Provides MFA estimations accounting for measurements imprecision.

» Makes easy to plot the results as interval estimates or flux distributions.

» Is composed of simple functions that MATLAB users can use in flexible ways and modify if needed.

» Includes a user graphic interface that helps the user to represent its measurements and their uncertainty.

The toolbox is compatible with COBRA, a well-known Toolbox to work with constraint-based metabolic models (Schellenberger, 2011; Agren,2013). Any COBRA-compatible model can be used in PFA.

List of functions

The **PFA Toolbox for MATLAB** provides a set of functions to easily apply Interval MFA and Possibilistic MFA. Here we list the functions and their purpose. Each function will be explained below.

Initialization	
`initPFAtoolbox`	It starts the PFA Toolbox
1. MFA problem formulation	
`define_MOC`	It defines the model-based constraints.
`define_PossMeasurements`	It represents the measured fluxes.
`define_MEC`	It defines the measurements-based constraints.
2. Computing estimations	
`solve_maxPoss`	It calculates the most possible set of flux values.
`solve_maxPossIntervals`	It calculates the interval of most possible flux values.
`solve_PossInterval`	It calculates the interval of flux values with the desired possibility
3. Plot	
`plot_PossMeasurements`	It plots measurements in possibilistic terms.
`plot_distribution`	It plots the distribution of a given flux.
`plot_intervals`	It plots interval estimates of a given flux.
Other	
`Solve_possintervalYMP`	(advanced function)
`solve_Interval`	It solves an Interval MFA problem.

Setting up the PFA Toolbox

The core methods implemented in PFA toolbox require solving linear programming problems (LP). To solve these problems, we use a flexible and efficient external optimizer: YALMIP (Löfberg, 2004). With the toolbox we provide a copy of YALMIP, but further information about this excellent piece of code can be found in http://users.isy.liu.se/johanl/yalmip/.

YALMIP, and therefore the PFA Toolbox, is able to use different LP solvers. Here is a list of those that we have tested:

- IBM ILOG CPLEX Optimizer. An efficient and reliable solver from IBM. A 90-day evaluation version can be downloading for free.
- GLPK solver. An efficient, reliable and widely used LP solver. Free.
- Linprog. The standard LP solver in MATLAB is not particularly efficient nor reliable, but it can works fine with small MFA problems.

If you want to use a different solver, check the YALMIP documentation.

Setting up the toolbox

These are the steps to install the PFA Toolbox: (1) copy the PFA Toolbox folder where you want it to be installed, (2) start MATLAB, (3) navigate to the PFA toolbox path, and (4) run the following script:

```
>> initPFAtoolbox
```

This action will add the PFA Toolbox directories to your MATLAB path and run the initialization script of YALMIP.

To check which LP solver will be used, run **yalmiptest**:

```
>> yalmiptest
```

The LP solver can be changed. Please read YALMIP help for further details.

Tested systems

The PFA Toolbox has been tested both on Mac OS (v. 10.01, Yosemite) and Windows 7 (64bit). The MATLAB version that have been tested are: R2012a (Windows), R2013b (Mac OS), and R2014a (Windows).

Backgrounds on MFA

Constraint-based model is an extensively used approach in metabolic modeling of living cells (Palsson, 2006; Stephanoupulos, 1998; Llaneras & Picó, 2008). In this context, MFA is a popular approach to, exploiting these models and some available measurements, estimate all the metabolic fluxes within cells.

Traditional MFA typically combines a linear stoichiometric model with a set of measurements of uptake and production rates or fluxes (which are those easy to obtain). See (Stephanopoulos, 1998) or (Heijden, 1994) for details. However, traditional MFA has some limitations that can be tackled with simple extensions, such as Interval MFA or Possibilistic MFA (Llaneras, 2011). These the two methodologies implemented in PFA Toolbox.

Firstly, let as summarize how traditional MFA is performed.

Consider a metabolic network represented by a stoichiometric matrix **N**. If we add a set of irreversibility constraints, we define a space of feasible steady-state flux distribution. These Model-based Constraints are denoted as MOC (Matrices and vector are denoted in bold):

$$MOC = \begin{cases} \mathbf{N} \cdot \mathbf{v} = 0 \\ \mathbf{D} \cdot \mathbf{v} \geq 0 \end{cases} \qquad (1)$$

Where **D** is a diagonal matrix with $\boldsymbol{D}_{ii} = 1$ if the flux is reversible (0 otherwise).

The second step of traditional MFA is to introduce a set of measurements fluxes in **v** as constraints. If we acknowledge that measurements are imprecise, these measurements could be represented as follows:

$$\mathbf{w_m} = \mathbf{v_m} + \mathbf{e_m} \qquad (2)$$

Where $\mathbf{e_m}$ is a vector that represent the intrinsic uncertainty of each of the experimental measured fluxes in the vector $\mathbf{w_m}$.

Thus, Traditional MFA can be defined as the exercise of determining the complete flux vector **v** that satisfies (1-2), for a ''reasonably small'' imprecision of the measurements in $\mathbf{w_m}$. Traditional MFA is often formulated as two steps procedure: 1) analyze the consistency of the measurements to detect gross errors —something which is only possible in overdetermined problems—, and 2) solve a Least Squares Problem to estimate **v**. A nice explanation of this approach can be found in (Klamt, 2002).

These traditional formulations have several problems, particularly in scenarios where data is scarce. 1) They do not account for inequality constraints, which are a useful way of improve the estimation with an information that is often available —

reactions irreversibilities—. 2) They only provide pointwise estimates, which are uninformative and unreliable when uncertainty is significant. 3) For the previous reason, Traditional MFA cannot be used in scenarios of uncertainty, for example, where only a few measurements are available or those we have are imprecise. Interval MFA and Possibilistic MFA, the methods implemented here, tackle those limitations.

Interval MFA

The interval MFA is a simple yet powerful extension of Traditional MFA. Basically, it is based on representing each measured flux as an interval, $[v_{m,i}^{min}, v_{m,i}^{Max}]$, which can be described as a set of inequality constraints.

$$\mathbf{v}_{\boldsymbol{m}}^{\boldsymbol{m}} \leq \mathbf{v}_{\boldsymbol{m}} \leq \mathbf{v}_{\boldsymbol{m}}^{\boldsymbol{M}} \tag{3}$$

where $\mathbf{v}_{\boldsymbol{m}}^{\boldsymbol{m}}$ and $\mathbf{v}_{\boldsymbol{m}}^{\boldsymbol{M}}$ are vectors with the minimum and maximum possible values that the measured fluxes in $\mathbf{v}_{\boldsymbol{m}}$ due to measurement's uncertainty. This way, the equations (1-3) define a constraint base*,* **CB**. These set of constraints define the space of feasible fluxes, according to our model and the constraints imposed by the measurements.

From this CB, interval estimates can be achieved for each measured and non-measured flux. The interval of feasible (possible) values for a flux distribution $\mathbf{v_i}\forall, [\mathbf{v_i^m}, \mathbf{v_i^M}]$ can be obtained solving two Linear Programing (LP) problems, as follows:

$$\begin{aligned} \mathrm{v_i^m} &= \min \mathrm{v_i} \;\; s.t. \, v \; \in \begin{cases} \mathrm{MOC} \\ \mathbf{v}_{\boldsymbol{m}}^{\boldsymbol{m}} \leq \mathbf{v_m} \leq \mathbf{v}_{\boldsymbol{m}}^{\boldsymbol{M}} \end{cases} \\ \mathrm{v_i^M} &= \max \mathrm{v_i} \;\; s.t. \, v \; \in \begin{cases} \mathrm{MOC} \\ \mathbf{v}_{\boldsymbol{m}}^{\boldsymbol{m}} \leq \mathbf{v_m} \leq \mathbf{v}_{\boldsymbol{m}}^{\boldsymbol{M}} \end{cases} \end{aligned} \tag{4}$$

This provides an interval estimate for each flux of interest.

These estimates are particularly useful in two common situations: when measurements are highly imprecise, and when only a few fluxes are measured. The main benefit of interval estimates is that we can perform MFA even if data is scarce, because the estimates will be only as precise as allowed by the actual uncertainty. Details about Interval MFA can be found in (Llaneras, 2007a; 2007b; 2011).

Example 1: Interval MFA

Now we will illustrate the use of PFA Toolbox with a simple example of Interval MFA. Consider the toy metabolic model shown in figure 1. The network has six fluxes and three metabolites, which impose three independent stoichiometric relationships. Consider also that two fluxes, v_4 and v_6, have been measured. The MFA problem will consist on estimating all other fluxes, and improve our estimates of v_4 and v_6, if possible.

Now we will use the PFA Toolbox to estimate all fluxes in the network, based on the model and the measurements.

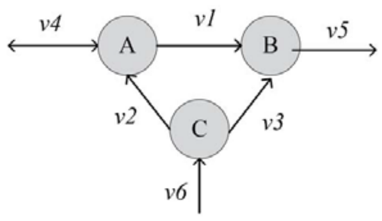

Figure 1. A toy metabolic model.

First, to define the constraint-based model we create the structure *model.* This structure contains the stoichiometric matrix and the information regarding reactions reversibility, as follows:

```
model.S= [-1 1 0 -1 0 0;
    1 0 1 0 -1 0;
    0 -1 -1 0 0 1];
model.rev= [0 0 0 1 0 0];
```

Then, we define flux variables using YALMIP syntax.

```
v = sdpvar(6,1);
```

Now we can generate the constraint base, CB, with all constraints defining the MFA problem so far.

```
CB = [model.S*v==0];
CB = [CB, diag(not(model.rev))*v>=0];
```

We can also bound any flux (if desired), as follows:

```
CB = [CB, v<=1000];
CB = [CB, v>=0];
```

The model has been defined.

Now we add the measurements, represented as intervals to capture their uncertainty. Let us assume, for example, that this uncertainty is ± 0.5 (flux units) for v_6 and ±1 for v_4. In this case, the measured fluxes can be established as follows:

```
CB = [CB,  9 <= v(6) <= 12];
CB = [CB, 9.25 <= v(4) <= 9.75];
```

This is all. The Interval MFA problem has been formulated.

Now, to compute interval estimates we use *solve_Interval*. For example, we can get interval estimate for flux v, as follows:

```
[vmin, vmax] = solve_Interval (CB, v(3));
```

In addition, we can get interval estimates for all six fluxes, as follows:

```
for i=1:6
 [vmin(i,:),vmax(i,:)]=solve_Interval(CB,v(i));
end
```

Finally, The PFA Toolbox also provides functions to plot the estimates. For example, we can use *plot_intervals* to depict interval estimates. The results are preseted in Figure 2.

```
plot_intervals([1 2 3 4 5 6],vmin,vmax)
axis square
xlabel('flux'),ylabel('Values')
```

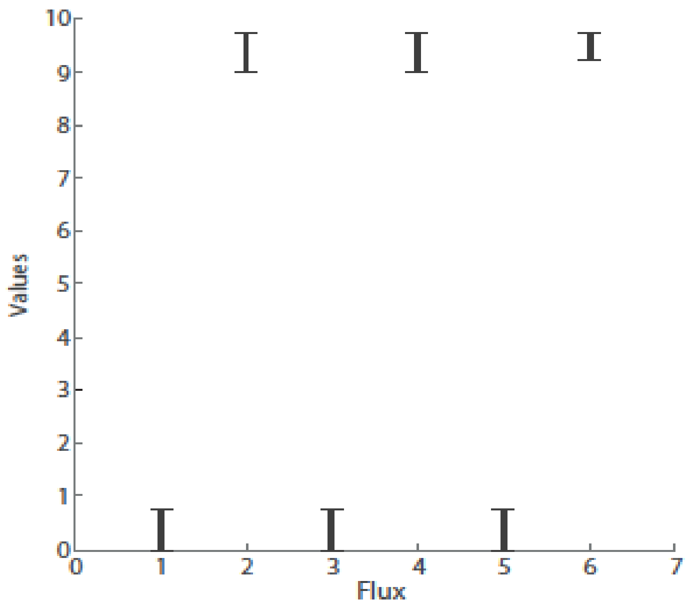

Figure 2. Fluxes estimation with our toolbox using Interval MFA approach.

Possibilistic MFA

In this section we describe Possibilistic MFA. This method can be seen as an extension of Interval MFA, more flexible and powerful able to provide richer estimates. Possibilistic MFA can represent the measured fluxes in a more flexible way —as distributions— and provides interval flux estimates of any desired degree of possibility and also possibility distributions. And in addition to perform flux estimations, Possibilistic MFA can also be used evaluate the consistency between a model and a set of measurements.

The possibilistic framework exploits the notion of "possibilistic constraint satisfaction problems", which was introduced in (Dubois, 1996). Similar optimization approaches to logic reasoning were previously explored in (Sala, 2001; Sala, 2008). The framework is based on two ideas: (1) represent knowledge with constraints satisfied to a certain degree, thus transforming the feasibility of a potential solution into a gradual notion of "possibility" that accounts for uncertainty, and (2) use computationally efficient optimization-based methods to query for the "most possible" solutions. This framework provides a simple and powerful way to deal with uncertainty both in the measurements and the model (e.g., imprecision and lack of knowledge), which is a typical difficulty in flux estimation problems.

Herein we summarize the formulation of Possibilistic MFA, but further details can be found in (Llaneras, 2009; Llaneras, 2011). First, let as summarize how Possibilistic MFA is performed.

Constraints: model and measurements. Let us start considering the constraints forming the model (MOC) that were given in equation (1). Then, we incorporate measurements of (some) extracellular fluxes as additional linear constraints, the measurement-based constraints (MEC).

Consider a constraint-based model, defined as *MOC*, as in (1). Now we will add the measurements, in a similar as in (2), but we will represent them in possibilistic terms by means of linear constraints and two non-negative slack variables that will represent measurements errors or uncertainty. These constraints are called measurement constraints, or *MEC*, as follows:

$$MEC = \begin{cases} \boldsymbol{w_m} = \mathbf{v_m} + \boldsymbol{\varepsilon_1} - \boldsymbol{\mu}_1 + \boldsymbol{\varepsilon}_2 - \boldsymbol{\mu}_2 \\ \boldsymbol{\varepsilon}_1, \boldsymbol{\mu}_1 \geq 0 \\ 0 \leq \boldsymbol{\varepsilon}_2 \leq \boldsymbol{\varepsilon}_2^{max} \\ 0 \leq \boldsymbol{\mu_2} \leq \boldsymbol{\mu}_2^{max} \end{cases} \qquad (5)$$

Where $\mathbf{v_m}$ is the vector of actual fluxes for each measured flux, and $\mathbf{w}$ is the vector of measured values. Both differ due to errors and imprecision. This uncertainty will be represented by the slack variables $\boldsymbol{\varepsilon}$, $\boldsymbol{\mu}$, $\boldsymbol{\varepsilon_2}$ and $\boldsymbol{\mu_2}$. The slack variables $\boldsymbol{\varepsilon}$ and $\boldsymbol{\varepsilon_2}$

represent errors in the measurement $\mathbf{w_m}$ in one direction (measure smaller than actual value), whereas $\boldsymbol{\mu}$ and $\boldsymbol{\mu_2}$ represent errors in the opposite direction. As explained below, error components $\boldsymbol{\varepsilon_1}$ and $\boldsymbol{\mu_1}$ will be penalized in a cost index (6) to assign a decreasing possibility to increasing errors, while $\boldsymbol{\varepsilon_2}$ and $\boldsymbol{\mu_2}$ will be remain non-penalized, thus defining a band of fully possible $\mathbf{v}_\mathrm{m}$ values around the measured $\mathbf{w_m}$, in particular $[\mathbf{w} + \boldsymbol{\varepsilon_2}^M, \mathbf{w} - \boldsymbol{\mu_2}^M]$. Once $\mathbf{v}_\mathrm{m}$ surpass these bounds, $\boldsymbol{\varepsilon_1}$ and $\boldsymbol{\mu_1}$ must be nonzero to fulfill (5) and the associated "cost" (6) will indicate some disagreement with the model (i.e., lower possibility, as later defined).

Each candidate solution of (1) and (5) can be denoted as $\delta=\{\mathbf{v}, \boldsymbol{\varepsilon_1}, \boldsymbol{\mu_1}, \boldsymbol{\varepsilon_2}, \boldsymbol{\mu_2}\}$. Now, we define a function, $\pi(\delta):\Delta\rightarrow[0,1]$, that assigns possibility in [0, 1] to each solution, ranging between impossible and fully possible. A simple yet sensible way to build this function is using a linear cost index J to penalize large deviations between actual fluxes and their measured values:

$$J = \boldsymbol{\alpha} \cdot \boldsymbol{\varepsilon_1} + \boldsymbol{\beta} \cdot \boldsymbol{\mu_1} \qquad (6)$$

And the possibility of each solution is defined as follows:

$$\pi(\delta) = \exp\bigl(-J(\delta)\bigr) \quad \delta \epsilon\, MEC\ \cap MOC \qquad (7)$$

Where $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are row vectors of accuracy coefficients or weights that define each measurement's *a priori* accuracy. These weights need to be defined by the user, e.g., if sensor error is «symmetric», $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ should be defined to be equal.

This way equations (5-7) can be interpreted as representing the statement: «given a measured value $\boldsymbol{w_m}$, the assertion $\boldsymbol{v}_m=\boldsymbol{w_m}$ is fully possible, and the more $\boldsymbol{v}_m$ and $\boldsymbol{w_m}$ differ, the less possible such situation is».

The actual possibility for each value of $\boldsymbol{v_m}$ depends on how the user defines the "sensor" accuracy coefficients in (5) and (6): the maximum bounds for $\boldsymbol{\varepsilon_2}^M$ and $\boldsymbol{\mu_2}^M$ define an interval of fully possible values (π=1), and the possibility of $\boldsymbol{v}_m$ being out of this interval depends on the user-selected weights $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ in (5). Notice that measurements uncertainty can be non-symmetric, and that very complex descriptions can be achieved by adding slack variables.

Poss-MFA: estimating fluxes

The simplest flux estimate $\mathbf{v_{mp}}$ in $\boldsymbol{\delta}_{mp}=\{\mathbf{v}_\mathrm{mp}, \boldsymbol{\varepsilon}_{\boldsymbol{1},mp}, \boldsymbol{\mu}_{\boldsymbol{1},mp}, \boldsymbol{\varepsilon}_{2,mp}, \boldsymbol{\mu}_{2,mp}\}$ is given by the maximum possibility (minimum-cost) solution of the constraint satisfaction problem (1)-(5), which can be obtained solving a linear programming problem (LP):

$$J^{min} = \min_{\boldsymbol{\varepsilon},\boldsymbol{\mu},\boldsymbol{v}} J\ s.t\ \{MOC \cap MEC \qquad (8)$$

Notice that pointwise estimates would be unreliable if multiple solutions were reasonably possible, so it is advisable to get interval estimates and distributions. Interval estimates for desired marginal (π) and conditional or *a posteriori* (γ) possibilities can be found, again, solving efficient LP optimizations (Llaneras, 2011).

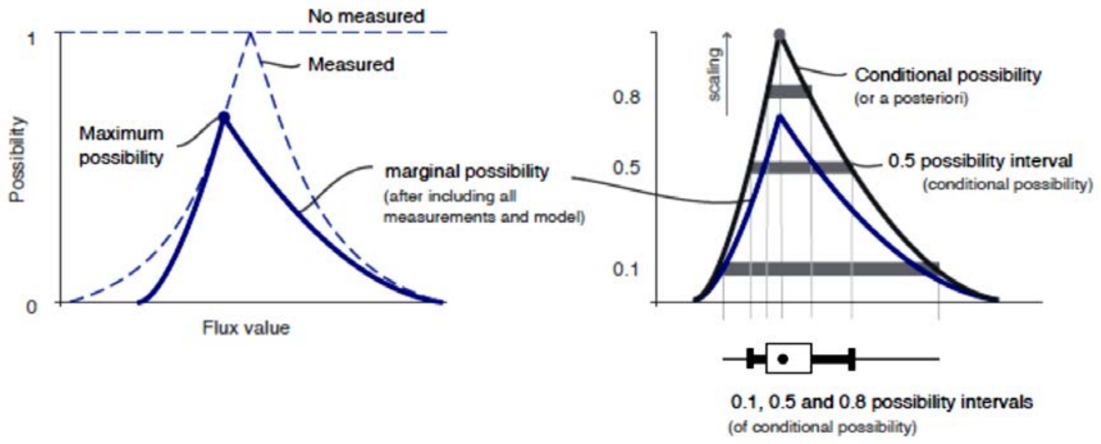

Figure 3. Possibilistic flux estimations. (Left) the figure shows the possibilistic distribution representing the original measurement, the marginal distribution and the maximum possibility flux estimation. (Right) The figure shows the marginal and conditional (a posteriori) possibility of π=0.8, 0.5 and 0.1, and the maximum possibility estimation, are depicted in a box-plot chart.

Interval estimates. The interval of values with conditional possibility higher than γ for a given flux $[v_{i,\gamma}^{m}, v_{i,\gamma}^{M}]$ can be computed solving two extra LPs:

$$\boldsymbol{v}_{i,\gamma}^{m} = \min_{\boldsymbol{\varepsilon},\boldsymbol{\mu},\boldsymbol{v}} \boldsymbol{V}_i \;\; s.t \begin{cases} MOC \cap M\varepsilon C \\ J - J_{min} < -\log\gamma \end{cases} \qquad (9)$$

The upper bound is conformed replacing minimum by maximum.

These possibilistic intervals have a similar interpretation to confidence intervals (*credible intervals*) in Bayesian statistics. In practice, getting estimates with conditional possibilities γ is equivalent to normalize the marginal possibility distributions to a maximum equal to one. We typically compute the interval of most possible values (γ=1) and other less possible intervals (γ=0.8, 0.5 and 0.1) to capture uncertainty.

Marginal and conditional possibility distribution. Additionally to interval estimates, PFA toolbox allow estimate conditional possibility distributions. Each interval is estimate as was previously explained for different possibilities to build a distribution. Figure 3 shown a conditional possibility distribution, and marginal distribution is als presented

The marginal distribution can be interpreted as the ''distribution of the possible values for each flux in the network given the measurements'' and the conditional possibility is a normalization of marginal possibility, where we assume that the model and measurements are correct (Llaneras, 2009).

Poss-MFA: evaluating data consistency

Poss-MFA can also be applied to evaluate the degree of consistency between given model (1) and a set of experimental measurements (5). Notice that the most possible solution of the constraint satisfaction problem (1) and (2) computed with (6) has an associated degree of possibility that grades consistency:

$$\pi^{mp} = \exp(-j^{min}) \qquad (10)$$

This value, π^{mp} in [0,1], grades the consistency between model and measurements. Possibility equal to one must be interpreted as complete agreement between the model and the measurements, whereas lower values imply that there is certain degree of error in the measurements, the model or both. The evaluation of consistency can be used to (a) conciliate a set of experimental measurements, (b) serve as basis for process monitoring and fault detection systems, or (c) validate a constraint-based model (Llaneras, 2009; 2011; Tortajada, 2010).

Example 2: introduction to Possibilistic MFA

We will consider the same network of example 1, showed in Figure 4. The network has six fluxes and three metabolites, which impose three independent stoichiometric relationships. Consider also that two fluxes v_4 and v_6, have been measured. The MFA problem will consist on estimating all other fluxes, and improve our estimates of v_4 and v_6, if possible.

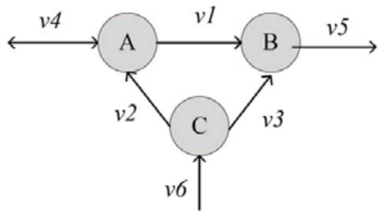

Figure 4. A toy metabolic model.

Again, we start by defining the constraint-based model. The structure *model* contains the stoichiometric matrix and the information regarding reactions reversibility, as follows:

```
model.S= [-1 1 0 -1 0 0;
    1 0 1 0 -1 0;
    0 -1 -1 0 0 1];
model.rev= [0 0 0 1 0 0];
```

The model defines the model constraints (MOC), that we use to create the Constraint Base for our Possibilistic MFA problem:

```
[PossProblem] = define_MOC(model);
```

Now we add the measurements. Let us assume, for example, that we have measurements for fluxes 4 and 6, with $w_4 = 9.5$ and $w_6 = 10.5$, and that the first measurement is very accurate and last one is unreliable. We can choose $\boldsymbol{\varepsilon}_2^{max}, \boldsymbol{\mu}_2^{max}, \boldsymbol{\alpha}$ and $\boldsymbol{\beta}$, accordingly. (In subsequent examples, we will give more details about how to choose these variables.)

```
PossMeasurements.wm    = [9.5  10.5]';
PossMeasurements.e2max = [0.25  1.5]';
PossMeasurements.m2max = [0.25  1.5]';
PossMeasurements.alpha  = [2   0.15]';
PossMeasurements.beta   = [2   0.15]';
```

At this point, we can check how our measurements look.

```
% To plot one single measured flux, for flux 4.
plot_PossMeasurements(possmeas,1);

% To plot all measured flux
index = [4 6]; % Indexes in model of the measured fluxes
for f=[1:6]
  subplot(2,3,f), xlim([0 50]), hold on
  m=find(index==f);
  if(not(isempty(m)))
    [x,y] = plot_PossMeasurements(PossMeasurements, m);
     plot(x,y,'b--')
  end
end
```

Results are presented in figure 5

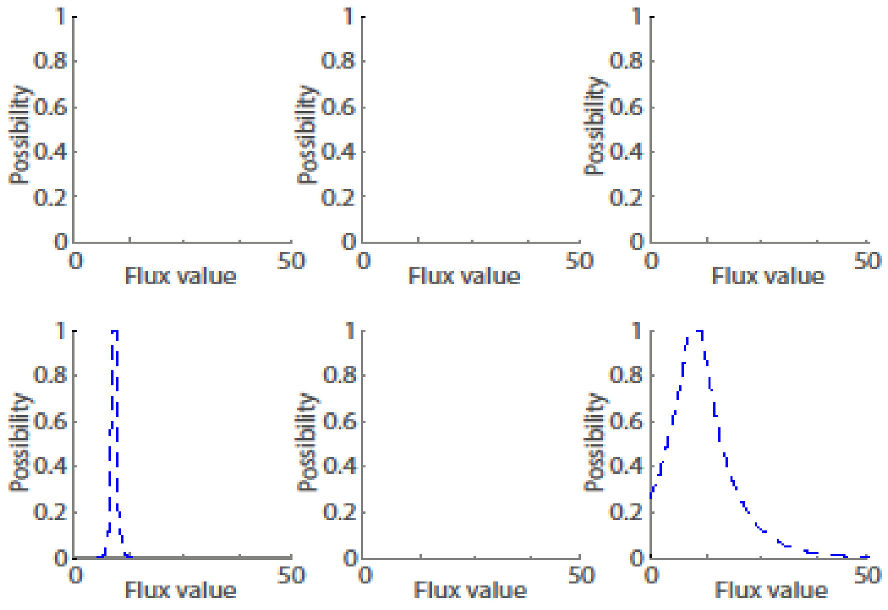

Figure 5. Measured fluxes and their uncertainty, represented as a possibilistic distribution.

Once the measurements have been defined, we can add the as measurement constraints (MEC) to our Possibilistic MFA problem:

```
index = [4 6]; % Indexes in model of the measured fluxes
[PossProblem] = define_MEC(PossProblem, PossMeasurements, index);
```

At this point, once we have added the model and the measurements, the Possibilistic MFA problem is completely defined. Now we can get the flux estimations. Let us present the three different options.

You can perform a **pointwise** estimation:

```
[v,poss]=solve_maxPoss(PossProblem);
```

It returns two outputs, v, the estimate for each flux in the network, and *poss*, the possibility of this most possible solution.

However, you can also get a more reliable estimate, the **interval** of values with maximum possibility:

```
[vmin,vmax]=solve_maxPossIntervals(PossProblem);
```

The outputs *vmin* and *vmax* are two vectors that contain the minimum and maximum values for each flux with maximum possibility. This way, if there are multiple flux values with maximum possibility, you will get all of them.

Similarly, you can also calculate any **interval estimation with specific possibility**. For example, you can get the interval of values for v_6 with a conditional possibility of 0.8 (i.e., that are "quite" possible).

```
[min6, max6]=solve_PossInterval(PossProblem,0.8,6);
```

Alternatively, you can get three intervals estimates for every flux, with a loop:

```
fluxes = [1 2 3 4 5 6];
for f=fluxes
 [mn2(f), mx2(f)]=solve_PossInterval(PossProblem,[1],f);
 [mn3(f), mx3(f)]=solve_PossInterval(PossProblem,[0.8],f);
 [mn4(f), mx4(f)]=solve_PossInterval(PossProblem,[0.5],f);
end
```

We provide a function to plot these rich and compact estimates, results are presented in Figure 6:

```
plot_intervals(fluxes,min2,max2,min3,max3,min4,max4);
```

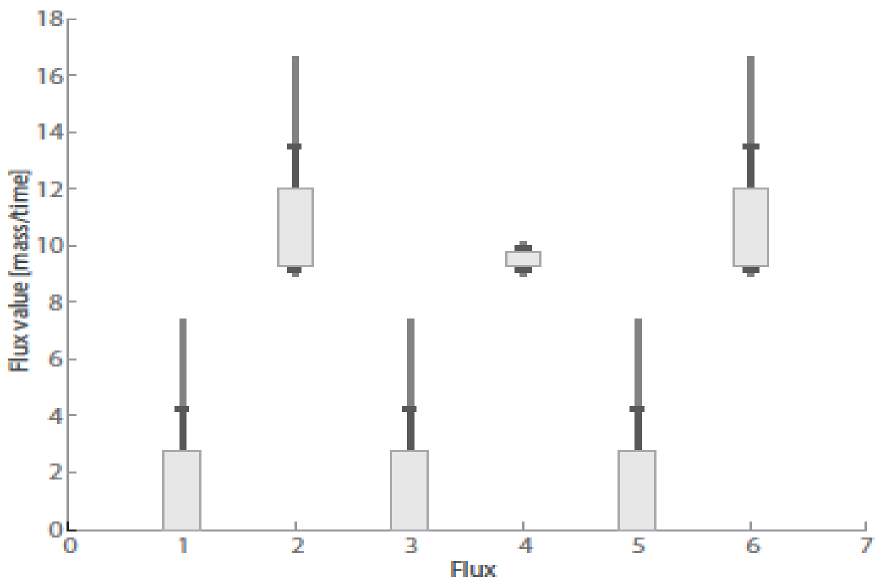

Figure 6. Estimates for every flux. Three interval estimates are given, for maximum conditional possibility (box), possibility of 0.8 (black line), and 0.5 (gray line).

Finally, you can compute a complete possibility distribution for any flux. To plot the results, a plot functions is provided. The following example computes and plots the possibility distribution for every flux in our example. Graphic is presented in figure 7.

```
possibilities = [0.05:0.05:1];        % granularity
fluxes = [1 2 3 4 5 6];               % fluxes to plot
for f=fluxes
 [min_p(:,f), max_p(:,f)]= solve_PossInterval(PossProblem,possibilities,f);
end
```

```
figure
for f=fluxes
    subplot(2,3,f), hold on
    [x,y]= plot_distribution(min_p(:,f),max_p(:,f),possibilities);
    plot(x,y,'k','lineWidth',2)
  xlim([0 50])
    if(find(index==f))
    [x,y]=plot_PossMeasurements(meas,find(f==index));
    plot(x,y,'b--')
  end
end
```

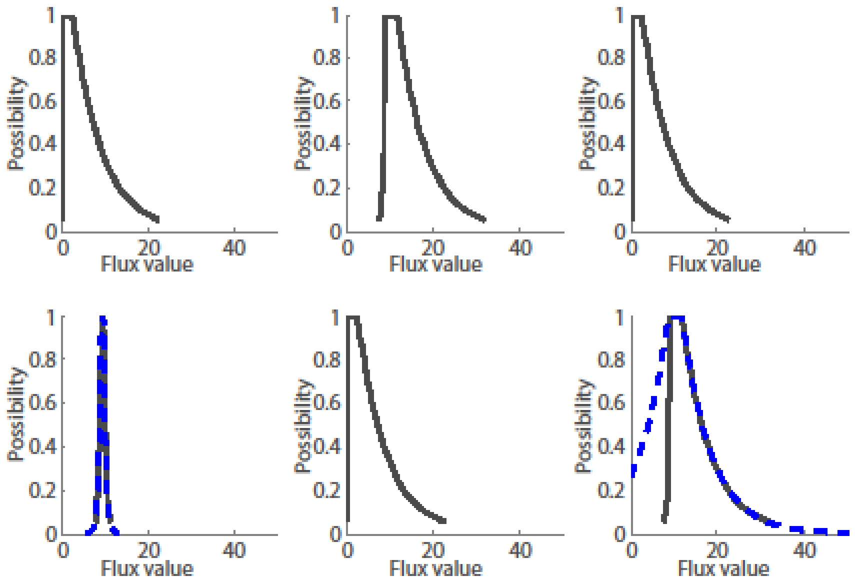

Figure 7. Distribution of possible values for every flux in the network. The original measured values are depicted in dashes lines. Solid lines represent the estimates provided by Possibilistic MFA. Notice that measured fluxes can also be estimated.

Note: all the computations for specific degrees of possibility have been computed for conditional possibilities. This is the default. If you want to compute marginal distribution, you should add a parameter when calling the functions (check their help for details). An explanation about marginal vs. conditional possibility can be found in (Llaneras, 2009; 2011).

Example 3: how to represent the measurements

In any MFA problem, there is one keystone step: decide how uncertainty your measurements are. Let us discuss this issue with a new example. Consider, for instance, that we have measurements of three uptake and production rates, for oxygen, ethanol and glucose. The measurements are: $w_{glu} = 40.6$, $w_{EtOH} = 15.96$, $w_{O_2} = 61.9$. The glucose measurement is very accurate, the ethanol one is moderately accurate, and the measurement of oxygen is quite unreliable. Now, we, as user, have to translate this information about the measurements and their uncertainty into a possibilistic representation, by choosing the weights **α** and **β** and the limits $\boldsymbol{\varepsilon_2^{max}}, \boldsymbol{\mu_2^{max}}$.

The PFA Toolbox provides two routes to achieve this.

The straightforward approach: let the user define these variables. First, we define the limits $\boldsymbol{\varepsilon_2^{max}}, \boldsymbol{\mu_2^{max}}$, which define an interval of values around the measured value with full possibility. Then, we define the weights **α** and **β**, which are associated with the slack variables $\boldsymbol{\varepsilon}_1$ and $\boldsymbol{\mu}_1$— to penalize the values out of full possibility interval and make them "less possible". If uncertainty is assumed to symmetric, **α** = **β**, and $\boldsymbol{\varepsilon_2^{max}} = \boldsymbol{\mu}_2^{max}$. These parameters need to be defined to every flux.

For example, this is a representation of our measurements:

```
PossMeasurements.wm=[40.6 15.96 61.9];
PossMeasurements.e2max=[0.5 0 4];
PossMeasurements.m2max=[0.5 0 4];
PossMeasurements.alpha=[2 0.5 0.11];
PossMeasurements.beta=[2 0.5 0.11];
```

As before, you can plot your measurements to check if they look, as you —the user–, want them to look (see plot in figure 8).

```
figure
for f=[1:3]
  subplot1,3,f), hold on
  [x,y] = plot_PossMeasurements(possmeas,f);
   plot(x,y,'b--')
    xlim([0 120]);
end
```

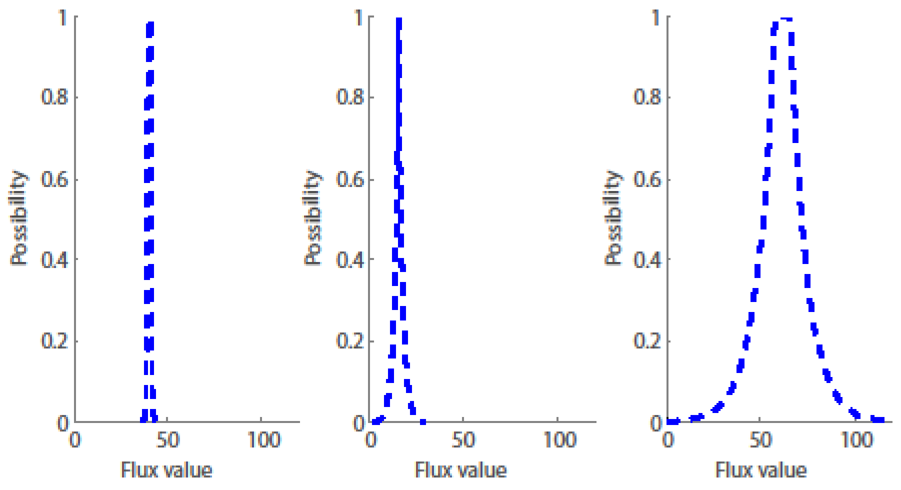

Figure 8. Our measurements, and its uncertainty, represented in possibilistic terms.

A simplest approach: use a function to define the measurements. The PFA Toolbox provides a function to assign the uncertainty of the measurements in an easier way that is suitable in many occasions. Basically, user only defines two intervals of high possibility (*intFP*) and low possibility (*intLP*), and parameters **α, β**, $\boldsymbol{\varepsilon_2^{max}}$ & $\boldsymbol{\mu_2^{max}}$ are defined accordingly:

- Full possibility (π=1) is assigned to de interval $\boldsymbol{w_m}$*±intFP*.
- Larger deviations are penalized so that values equal to $\boldsymbol{w_m}$*±intLP* have a possibility of π=0.1.

Coming back to our example, we could do the following:

```
wm=[40.6 15.96 61.9]; %define measures and its full and lower possibility
intFP=[0.5 0 4];
intLP=[1.65 4.6 24.9];

[PossMeasures]=define_PossMeasurements(wm,intFP,intLP);
```

And plot(see figure 9):

```
figure
for f=[1:3]
 subplot(1,3,f), hold on
 [x,y] = plot_PossMeasurements(PossMeasures,f);
  plot(x,y,'r--')
  xlim([0 120]);
end
```

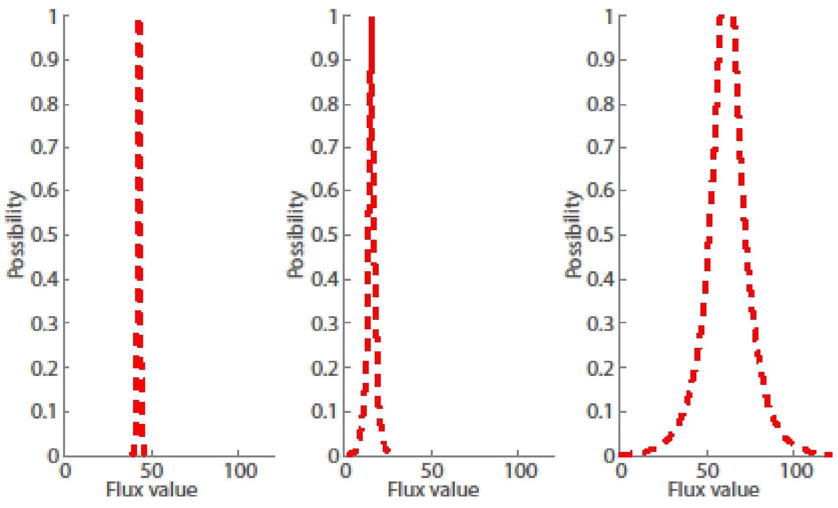

Figure 9. Our measurements, and its uncertainty, represented in possibilistic terms.

When using the PFA toolbox, we have noticed that there are often advantages in defining the interval of low and high possibility in relative terms. For example, one clear and simple way of define the uncertainty of O2 measurements is to state somethings like: «a 5% deviation from the measured values is fully possible, but a deviation larger than 20% is an event of low possibility».

If we reason like this for every measurement, they can be represented as follows:

```
wm=[40.6 15.96 61.9];
intFP = max(0.01,abs(wm.*0.05));
intLP = max(0.02,abs(wm.*0.20));

[PossMeasures]=define_PossMeasurements(wm,intFP,intLP);
```

Notice that the intervals are defined with a minum size of 0.01 and 0.02 flux units. We do this to avoid problems in case a measurement in $\mathbf{w_m}$ is zero or near zero. In general, when defining measurements uncertainty in absolute terms, is useful to consider also a minimal, absolute band of uncertainty. Otherwise, the uncertainty of near zero measurements tends to be underrepresented.

Example 4. Evaluating consistency

As mentioned above, Possibilistic MFA and the PFA Toolbox can be also useful to evaluate the degree of consistency between a given model and a set of experimental measurement. You can find real applications in (Tortajada, 2010; Morales, 2014), herein we present a simple example with the toy model of figure 1.

Let us consider the following situation: we have six different datasets, corresponding —for example— to six chemostat experiments, with four measured fluxes in each case (w_2, w_3, w_4 & w_6) and we want to check if there are larger error than expected in the measurements, by evaluating the consistency between the model and each dataset. (As we mention in the backgrounds, consistency analysis can be used both to validate a model or a set of data. In this example, we assume that the model has been shown to be precise and we use it to find error in the measurements.)

The first step was defining the measurements uncertainty; in this case, we considered the same uncertainty for all the measures. We consider that the fluxes are fully possible with deviations ±0.5 units of flux, and deviations larger than ±1.2 units will have a lower possibility.

The second one is computing the possibility of the most possible solution, with *solve_maxPoss*, which provides a simplest indication of how consistent model and measurements are. The results are provided in Table 1.

Dataset	Flux 2	Flux 3	Flux 4	Flux 6	π in [0,1]
1	1.88	1.09	0.0	2.16	1
2	2.07	0.95	0.63	2.70	1
3	1.72	1	1.48	2.90	1
4	2.02	0.57	1.33	3.55	1
5	2.32	1.13	2.62	3.52	1
6	0.73	2.04	1.98	2.55	**0.439**

Table 1. Experimental data and consistency analysis.

The results are that five in six datasets are fully possible, whereas the dataset 6 has a lower possibility (0.44). This implies that one of measured fluxes in dataset six is highly deviated, or more than one is sensibly deviated.

The procedure to perform all these calculations with the PFA Toolbox is presented below.

We start by defining the model.

```
model.S= [-1 1 0 -1 0 0;
    1  0 1 0 -1 0;
    0 -1 -1 0 0 1];
model.rev=[0  0 0 1 0 0];

[PossProblem] = define_MOC(model);
```

Then we are going to define the measurements and its uncertainties, which were defined previously as 0.5 for full possibility, and 1.2 for lower possibility.

```
index    =[ 2   3   4    6  ];
exp{1}.wm=[ 1.88 1.09 0.0  2.16];
exp{2}.wm=[ 2.07 0.95 0.63 2.70];
exp{3}.wm=[ 1.72   1   1.48 2.90];
exp{4}.wm=[ 2.02 0.57 1.33 3.55];
exp{5}.wm=[ 2.32 1.13 2.62  3.52];
exp{6}.wm=[ 0.73 2.04 1.98 2.55];

intFP=[0.5];
intLP=[1.2];
```

Then we compute the possibility of the most possible solution for each dataset.

```
for i=1:length(exp)
 [measures]=define_PossMeasurements(exp{i}.wm,intFP, intLP);
 [PossProblem]=define_MEC(PossProblem, measures, index);
 [v, poss]=solve_maxPoss(PossProblem);
 poss_all(i)=[poss];
  i
end
```

These results are those provided in Table 1.

Investigating the inconsistencies. To better understand where the inconsistencies come from, we can compute the marginal distributions for every flux in the dataset 6 (the inconsistent one).

First, we define the problem:

```
[PossProblem] = define_MOC(model);
[PossMeasures]=define_PossMeasurements(exp{6}.wm,intFP, intLP);
[PossProblem]= define_MEC(PossProblem, PossMeasures, index);
```

Then we compute the complete marginal possibility distribution for every flux and and plot the results.

```
poss = [0.01:0.01:0.439];
flux = [1 2 3 4 5 6];

for f=flux
    [minp(:,f),maxp(:,f)]=solve_PossInterval(PossProblem,poss,f,'none');
end

figure
for f=fluxes
 subplot(2,3,f), hold on
 [x,y] = plot_distribution(minp(:,f),maxp(:,f),poss);
 plot(x,y,'k','lineWidth',2)
 xlim([0 5])
 if(find(index==f))
  [x,y] = plot_PossMeasurements(PossMeasures, find(f==index));
   plot(x,y,'b--')
   xlim([0 5])
 end
end
```

These results (figure 10) shown that, taken the model into account, there are larger than expected errors in the measurements. Further investigations can be performed to find which measurements may be provoking the inconsistency. See (Llaneras, 2009; Llaneras, 2011) for more elaborate examples of these procedures.

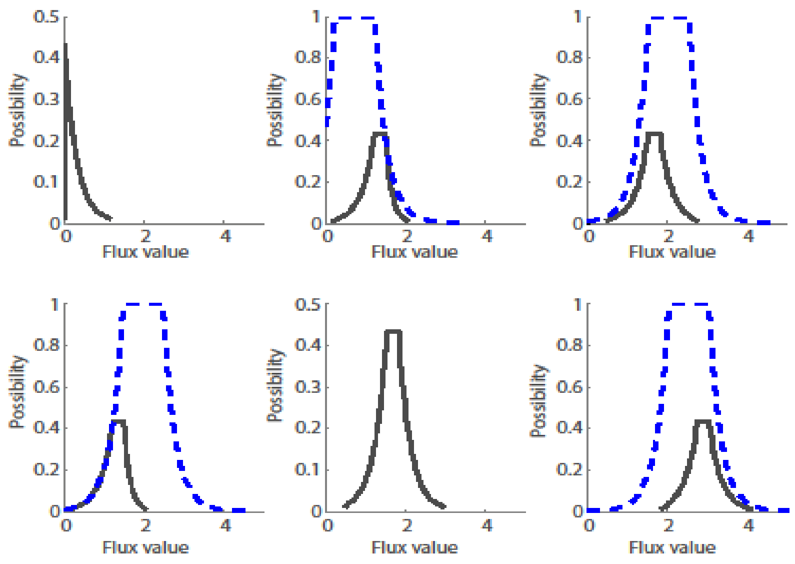

Figure 10. Possibilistic MFA performed to detect error in a set of measurements. Solid lines show the marginal possibility distribution for each flux (in the dataset 6). Dashed lines represent the original measurements, or a priori, that are of course fully possible. The results shown that, taken the model into account, it seems to be larger than expected errors in the measurements.

Functions description

The following section describes each function of the PFA Toolbox. We show its syntax, a brief description, and the lists of inputs and outputs. This information can also be consulted within MATLAB.

Initialization

`initPFAtoolbox`	Start the PFA Toolbox

1. MFA problem formulation

`define_MOC`	Define the constraint-based model and PFA problem.
`define_PossMeasurements`	Define a set of measurements in possibilistic terms.
`define_MEC`	Add the measurements as constraints.

2. Computing estimations

`solve_maxPoss`	One most possible set of flux values.
`solve_maxPossIntervals`	The interval of most possible flux values.
`solve_PossInterval`	The interval of flux values with the desired poss.

3. Plot

`plot_PossMeasurements`	Plot measurements in possibilistic terms.
`plot_distribution`	Plot the distribution of a given flux.
`plot_intervals`	Plot interval estimates of a given flux.

Other

`Solve_possintervalYMP`	(advanced function)
`solve_Interval`	To solve an Interval MFA problem.

# initPossToolbox

The function initiates the PFA Toolbox.

Syntax

```
>> initPFAtoolbox
```

Description

The function initiates the PFA Toolbox. It adds the toolbox folder to the MATLAB path. If YALMIP is not already installed, a copy is also added to the path. The optimization solver GLPK is selected, if it is installed and is detected. Otherwise, YALMIP is initialized with the available solver.

# defineMOC

Generate the constraint-based model structure.

Syntax

```
[PossProblem] = define_MOC(model)
```

Description

[PossProblem] = define_MOC(model) returns a struct that defines the Possibilistic MFA problem. Initially, it contains some symbolic decision variables (the fluxes v) and the first constraints into the CB (the stoichiometry and the irrerversibilities). The function receives as input another struct, model. This struct can be a COBRA model —it has the same fields— or be created by the user. The struct is a simple one, containing the following:

model.S	The stoichiometric matrix.
model.rev	A vector indicating which reactions are reversible with '1' for those reversible and '0' otherwise.
model.lb	(optional) Lower bound for each flux.
model.ub	(optional) Upper bound for each flux.

# define_PossMeasurements

Generates a set of possibilistic measurements.

Syntax

```
[PossMeasurements]=define_PossMeasurements(wm, intFP, intLP);
[PossMeasurements]=define_PossMeasurements(meas);
```

Description

[PossMeasurements]=define_PossMeasurements(wm, intFP, intLP) returns a *PossMeasurements* struct that contains the measured fluxes *wm* the limits *e2max* and *m2max*, and the weights *alpha* and *beta*, required to represent a set of measurements in possibilistic terms. Vector *wm* is a vector with the measured values for each flux. *intFP* represent the interval around *wm* in which flux values have maximum possibility. It can be a single value, equal for all the measurements, or a vector with a specific value for each measurement. *intLP* represent the interval around *wm* in which measurements have a low possibility. It can be a single value or a vector.

[PossMeasurements]=define_PossMeasurements(meas) does exactly the same, but receives a struct as input, with fields: *wm*, *intFP* and *intLP*.

wm	Measures values vector
intFP	Interval with maximum possibility
intLP	Interval with low possibility

Recall that *define_PossMeasurements* defines the measurements and their uncertainty based on two intervals. Basically, the user only defines two intervals of high possibility (*intFP*) and low possibility ($intLP$), and parameters **α, β**, $\boldsymbol{\varepsilon}_2^{max}$ & $\boldsymbol{\mu}_2^{max}$ are defined accordingly:

- Full possibility (π=1) is assigned to the interval *w±intFP.*
- Larger deviations are penalized so that values equal to *w ±intLP* have a possibility of π=0.1.

As an alternative, advanced user can define directly the variables in the *PossMeasurements* struct.

# defineMEC

Add the measurement-constraints to a *PossProblem* structure.

Syntax

```
[PossProblem]=define_MEC(PossProblem, PossMeasurements, index)
```

Description

[PossProblem]=define_MEC(PossProblem, PossMeasurements, index) returns the *PossProblem* received as input adding the measurements as constraints. The output struct has the following fields:

.v	The vector of fluxes (as a YALMIP decision variable)
.CB	The constraints-base, i.e., the set of all constraints (in YALMIP syntax).
.e1	The vector of slack variables (as a YALMIP decision variable)
.m1	The vector of slack variables (as a YALMIP decision variable)
.e2	The vector of slack variables (as a YALMIP decision variable)
.m2	The vector of slack variables (as a YALMIP decision variable)
.J	The objective function penalizing the deviations between fluxes and measured values, accordingly to alpha and beta.

The input *PossProblem* is a struct generated by *defineMOC* function, and defines the model constraints of the MFA problem. The input *PossMeasurements* is a structure generated by *define_PossMeasurements* or manually, and defines the measurements in possibilistic terms. Finally, the vector *index* indicates the indexes of the measured fluxes in the model.

# solve_maxPoss

Returns one flux vector estimate of maximum possibility.

Syntax

```
[v,poss] = solve_maxPoss(PossProblem);
[v, poss, diagnostic] = solve_maxPoss(PossProblem, options);
```

Description

[v, poss]=solve_maxPoss(PossProblem) returns the column vector **v** with a set of fluxes with maximum possibility. Notice, however, that it could more than one flux vector with maximum possibility, and the solver returns only one of them. To know if there are multiple candidates, use *solve_maxPossInterval*. The only mandatory input is *PossProblem*, a structure defining the Possibilistic MFA problem (see *define_MEC* and *define_MOC*).

The optional input "*options*" specifies the YALMIP options (see '*help yalmip*').

The optional output "*diagnostic*" returns information about the solver status (see '*help yalmiperror*'). "*diagnostic.error*" indicates if the problem was successfully solved (it return '0' if the problem is successfully solved, '1' if the problem is infeasible, etc. See '*yalmiperror*'). "*diagnostic.details*" provides all the info returned by the optimization solver.

# solve_maxPossIntervals

Returns the interval estimate of fluxes with maximum possibility.

Syntax

```
[vmin,vmax]=solve_maxPossIntervals(PossProblem);
[vmin, vmax, diagnostic] = solve_maxPossIntervals(PossProblem, options);
```

Description

[vmin, vmax]=solve_maxPossIntervals(PossProblem) returns the interval estimate with maximum possibility in vectors v*min*, and *vmax*, which contain the lower and upper limits for each flux. The only mandatory input is *PossProblem*, a structure defining the Possibilistic MFA problem (see *define_MEC* and *define_MOC*).

The optional input "*options*" specifies the YALMIP solver options (see *'help yalmip*').

The optional output "*diagnostic*" returns information about the solver status (see '*help yalmiperror*'). "*diagnostic.error*" indicates if the problem was successfully solved (it return '0' if the problem is successfully solved, '1' if the problem is infeasible, etc. See '*yalmiperror*'). "*diagnostic.details*" provides all the info returned by the optimization solver.

# solve_PossInterval

Returns an interval estimate for a flux for the desired degree of possibility.

Syntax

```
[vmin, vmax]=solve_PossInterval(PossProblem, poss, flux);
[vmin, vmax]=solve_PossInterval(PossProblem, poss, flux, mode);
[vmin, vmax, diagnostic]=solve_PossInterval(PossProblem, poss, flux, mode, options);
```

Description

[vmin, vmax, diagnostic]=solve_PossInterval(PossProblem, poss, flux) returns an interval estimate for a flux, for the desired degree of conditional possibility. The vectors v*min* and v*max* contain the upper and lower limits of the flux of interest for the degrees of possibility specified as input.

The input *PossProblem* is a structure defining the Possibilistic MFA problem (see *define_MEC* and *define_MOC*). The vector *poss* indicates the degree of possibility for the intervals that you want to compute (e.g., 0.8, or [0.99 0.8 0.1], etc.). Flux indicates the index of the flux to be estimated.

> Example. [vmin, vmax]= solve_PossInterval (ProblemA, [0.99 0.5 0.1], 7) computes three interval estimates for the flux 7 in the ProblemA, for conditional of possibilities 0.99, 0.5 and 0.1.

The optional input "*mode*" can be used to get estimates of marginal possibility, instead of conditional possibility. If mode is not provided, the function provides conditional possibilities as a default.

The optional input "*options*" specifies the YALMIP solver options (see '*help yalmip*').

The optional output "*diagnostic*" returns information about the solver status (see '*help yalmiperror*'). "*diagnostic.error*" indicates if the problem was successfully solved (it return '0' if the problem is successfully solved, '1' if the problem is infeasible, etc. See '*yalmiperror*'). "*diagnostic.details*" provides all the info returned by the optimization solver.

# plot_intervals

Plot interval estimates for a set of fluxes.

Syntax

```
plot_intervals(flux, vmin_c1, vmax_c1);
plot_intervals(flux, vmin_c1, vmax_c1 ,vmin_c2, vmax_c2);
plot_intervals(…, …, …, …, vmin_c3, vmax_c3);
[h_out] = plot_intervals(…);
```

Description

plot_intervals(flux, vmin_c1, vmax_c1) creates a 2D plot to show a set of interval estimates. The function receives as input a vector of x coordinates, *flux* (e.g., [1:5] or [1 7 8]). Vectors *vmin_c1*, v*max_c1* define the lower and upper limits of the intervals to be plotted, which typically would have been computed with *solve_PossInterval*.

plot_intervals(flux, vmin_c1, vmax_c1, vmin_c2, vmax_c2) and *plot_intervals(flux, vmin_c1, max_c1, vmin_c2, vmax_c2, vmin_c3, vmax_c3)* allow to plot two and three pairs of interval estimates in a single, compact graph.

If an output variable is indicated, "*h_out*", the functions will return a structure with the handles of every object in the figure. This way, it can be customized.

# plot_distribution

Plot a complete the possibilistic distribution of a flux.

Syntax

```
plot_distribution(vmin,vmax,poss)
[meas_out,poss_out] = plot_distribution(vmin,vmax,poss)
```

Description

plot_distribution(vmin,vmax,poss) plots the possibilistic distribution for a specific flux. The function receives three inputs, vectors *vmin*, and v*max*, with the lower and higher limits of a set of interval estimates, each interval corresponding to the degrees of possibility indicated in *poss*. v*min* and *vmax* are the output of the *solve_PossInterval(…, poss, …).*

If output variables are given, as in *[meas_out, poss_out]=…*, instead of drawing a graph, the data is returned as two output variables for x and y coordinates. This way, the user can use plot function to plot the data with a custom style.

Example: *[x, y] = plot_distribution(vmin, vmax, poss); plot(x, y, ‘r’).*

# plot_PossMeasurements

Plot measurements defined in possibilistic terms.

Syntax

```
plot_PossMeasurements(PossMeasurements,flux);
plot_PossMeasurements(possmeas,flux, resolution, pos_min);
[meas_out,poss_out]= plot_PossMeasurements(…);
```

Description

plot_PossMeasurements(PossMeasurements, flux) creates a 2D plot with the possibilistic distribution of one measured flux. This way the user can check if the measurements and its uncertainty is well-defined. The function receives as input a set of measurements in a structure, *PossMeasurements*. The measured flux to be plotted is indicated with *flux*.

The input *PossMeasurements* is a struct with the measured fluxes *wm* the limits *e2max* and *m2max*, and the weights *alpha* and *beta*. It can be defined manually by the user or via function *define_PossMeasurements*.

The optional inputs "*resolution*" and "*pos_min*" are used to specify the number of points used to create the plots (default, 20) and the minimum possibility to be plotted (default, 0.001).

If output variables are given, as in *[meas_out, poss_out]=…*, instead of drawing a graph, the data is returned as two output variables for x and y coordinates. This way, the user can use plot function to plot the data with a custom style.

Example: *[x, y] = plot_PossMeasurements(A, 3); plot(x, y, '*k').*

# Solve_possintervalYMP (for advanced users only)

Returns an interval estimate for a flux for the desired degree of possibility. This function uses a different and rudimentary syntax. It is of use only for advanced users wanting to do non-standard computations.

Syntax

```
[vmin,vmax,diagnostic]=solve_PossIntervalYMP(F,J,poss,var);
[__,__,__] = solve_PossIntervalYMP (__,__,__,__,mode);
[__,__,__] = solve_PossIntervalYMP (__,__,__,__,options);
```

Description

This function uses a different and rudimentary syntax. It is of use only for advanced users wanting to do non-standard computations.

[vmin, vmax]=solve_PossIntervalYMP(F, J, poss, var) returns an interval estimate for a flux, for the desired degree of conditional possibility. The vectors v*min* and v*max* contain the upper and lower limits of the flux of interest for the degrees of possibility specified as input.

The input *F* is a YALMIP structure defining a set of constraints. *J* is a YALMIP object function defining the possibility of each candidate solution of *F*. The vector *poss* indicates the degree of possibility of the intervals that you want to compute (e.g., 0.8, or [0.99 0.8 0.1], etc.). Finally, *var* is the variable —typically a flux— that you want to estimate.

The optional input "*mode*" can be used to get estimates of marginal possibility, instead of conditional possibility. If mode is not provided, the function provides conditional possibilities as a default.

The optional input "*options*" specifies the YALMIP solver options (see 'help yalmip').

# solve_interval

Solve an interval MFA problem.

Syntax

```
[vmin, vmax] = solve_interval(constraints,flux);
[__, __ , diagnostic]=solve_interval(__, __, options);
```

Description

[vmin, vmax]=solve_interval(constraints, flux, solver_options) returns the column vectors *vmin* and *vmax*, which define the interval estimates for the fluxes that have been asked. Regarding the inputs, the *constraints* are a YALMIP struct with a set of constraints for interval MFA problem. *Flux* is a vector with the indexes of the fluxes to be estimated. *options* is an optional input that allows to specify the YALMIP solver options (use 'help yalmip' for details).

Graphic user interface

New user of Possibilistic MFA find it difficult to represent the measurements in possibilistic terms. In order to facilitate this task, the PFA toolbox includes a Graphic User Interface (GUI) that allows to easily represent the measurements.

Initialize

To initiate the user interface, write in the *command window* of MATLAB:

```
>>guiPossMeasurements
```

The GUI contains four panels: one to create new or upload a set of flux measurements, a second one to add and remore measurements to the set, a third one to edit each single flux, and the last one to save the work.

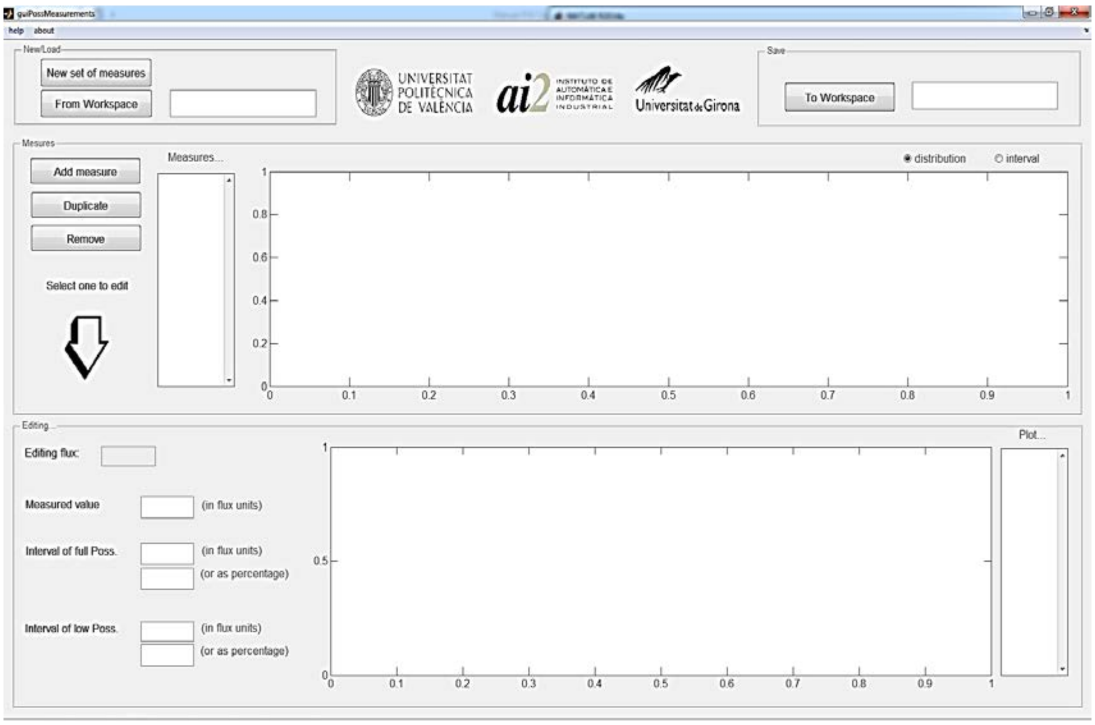

Figure 11. User interface of PFA toolbox for represent possibilistic measures.

(1) New/load: create or upload a set of measures

The initial step to use the GUI is to generate or upload a set of measurements. To create a new set of measures just click in the button *«new set of measures»*. Alternatively, you can import a set of measurements from the workspace to continue a previous work. In this case, write the name of the variable of the previously saved data and click *«from workspace»*.

(2) Measures: Addition, duplicate or remove measures

Once a set of measurements has been create or uploaded, each measure will be listed and ploted in the box below. Here you can click button to add, remove or duplicate the selected flux. In the axes on the right, all measurements will be plotted simultaneously. You can chose if measurements are plotted as possibilistic distributions (more detailed) or intervals (more compact).

(3) Editing flux measurements

To edit a flux measurement, first you must select it in the list of measurements. Then you can change the measured value and its ucertainty (by means of the intervals of flux and half possibility). To do this, simply modify the values in the corresponding boxes.

The measurements uncertainty can be defined both in absolute (flux units) and relative terms (as a percentage of the measured value).

The axes on the right allow you to visualize the measurments being edited and any other of your interest (selecting them in the box on the right).

(4) Save the measurements

After adding all the measurements and defining their uncertainty, the GUI can generate a matlab struct with the results. This struct can then be used with PFA Toolbox.

To save this structure to the MATLAB workspace, just write a name for the variable and click «*to workspace*». A struct will be saved to the workspace with the given name and three fields wm (with the measured values), intFP and intLP (with the intervals defining the uncertainty of each measurement). This structure is the input for *define_PossMeasurements*, one of the functions to perform Possibilistic MFA.

Note: If you want to save the structure into a file, use the standard MATLAB command, save.

References

The following are the references cited in the text, within others useful readings.

About constraint-based models

Llaneras, F., & Picó, J. (2008). Stoichiometric modelling of cell metabolism.*Journal of Bioscience and Bioengineering*, *105*(1), 1-11.

Bernhard, P. (2006). Systems biology: properties of reconstructed networks.

About COBRA Toolbox

Becker, S. A., Feist, A. M., Mo, M. L., Hannum, G., Palsson, B. Ø., & Herrgard, M. J. (2007). Quantitative prediction of cellular metabolism with constraint-based models: the COBRA Toolbox. *Nature protocols*, 2(3), 727-738.

Schellenberger, J., Que, R., Fleming, R. M., Thiele, I., Orth, J. D., Feist, A. M., ... & Palsson, B. Ø. (2011). Quantitative prediction of cellular metabolism with constraint-based models: the COBRA Toolbox v2. 0. *Nature protocols*, 6(9), 1290-1307.

Agren, R., Liu, L., Shoaie, S., Vongsangnak, W., Nookaew, I., & Nielsen, J. (2013). The RAVEN toolbox and its use for generating a genome-scale metabolic model for Penicillium chrysogenum. PLoS Comput Biol, 9(3), e1002980.

Interval MFA methodology

Llaneras, F., & Picó, J. (2007). An interval approach for dealing with flux distributions and elementary modes activity patterns. Journal of theoretical biology, 246(2), 290-308.

Llaneras F, Bastin G, Picó J (2007a). On metabolic flux analysis when measurements are insufficient and/or uncertain. IAP Dysco workshop.

Llaneras, F. (2011). Interval and possibilistic methods for constraint-based metabolic models. PhD Thesis. Universidad Politécnica de Valencia.

Interval MFA Applications

Llaneras, F., & Picó, J. (2007b). A procedure for the estimation over time of metabolic fluxes in scenarios where measurements are uncertain and/or insufficient. *BMC bioinformatics*, 8(1), 421.

Iyer, V. V., Yang, H., Ierapetritou, M. G., & Roth, C. M. (2010). Effects of glucose and insulin on HepG2-C3A cell metabolism. *Biotechnology and bioengineering*, *107*(2), 347-356.

D'Huys, P. J., Lule, I., Van Hove, S., Vercammen, D., Anné, J., Bernaerts, K., & Van Impe, J. (2010). Flux balance analysis and flux spectrum analysis for Streptomyces lividans batch fermentations. *Book of Abstracts-Study Day Interuniversity Attraction Pole IAP VI/4.*

Zamorano, F., Wouwer, A. V., & Bastin, G. (2010). A detailed metabolic flux analysis of an underdetermined network of CHO cells. *Journal of biotechnology*, *150*(4), 497-508.

Hoppe, A., Hoffmann, S., Gerasch, A., Gille, C., & Holzhütter, H. G. (2011). FASIMU: flexible software for flux-balance computation series in large metabolic networks. *BMC bioinformatics*, 12(1), 28.

Lohr, V., Hädicke, O., Genzel, Y., Jordan, I., Büntemeyer, H., Klamt, S., & Reichl, U. (2014). The avian cell line AGE1. CR. pIX characterized by metabolic flux analysis. *BMC biotechnology*, 14(1), 72.

Possibilistic MFA methodology

Llaneras, F., Sala, A., & Picó, J. (2009). A possibilistic framework for constraint-based metabolic flux analysis. *BMC systems biology*, 3(1), 79

Llaneras, F. (2011). Interval and possibilistic methods for constraint-based metabolic models. PhD Thesis. Universidad Politécnica de Valencia.

Sala, A., & Albertos, P. (2001). Inference error minimisation: fuzzy modelling of ambiguous functions. *Fuzzy Sets and Systems*, *121*(1), 95-111.

Sala, A. (2008). Encoding fuzzy possibilistic diagnostics as a constrained optimization problem. *Information Sciences*, 178(22), 4246-4263.

Dubois, D., Fargier, H., & Prade, H. (1996). Possibility theory in constraint satisfaction problems: Handling priority, preference and uncertainty. *Applied Intelligence*, 6(4), 287-309.

Possibilistic MFA applications

Llaneras, F., Sala, A., & Picó, J. (2009). Applications of possibilistic reasoning to intelligent system monitoring: a case study. In Control Applications,(CCA) & Intelligent Control,(ISIC), 2009 IEEE. 171-176

Tortajada, M., Llaneras, F., & Picó, J. (2010). Possibilistic Validation of a Constraint-Based Model under Data Scarcity: Application to Pichia Pastoris Cultures. In *Computer Applications in Biotechnology* (Vol. 11, No. 1, pp. 19-23).

Folch-Fortuny A, Tortajada M, Prats-Montalbán JM, Llaneras F, Picó J, Ferrer A (2014). MCR-ALS on metabolic networks: Obtaining more meaningful pathways. *Chemometrics and Intelligent Laboratory Systems*.

Morales, Y., Tortajada, M., Picó, J., Vehí, J., & Llaneras, F. (2014). Validation of an FBA model for Pichia pastoris in chemostat cultures. BMC systems biology, 8(1), 142.

About Traditional MFA methods

Klapa, M. I., & Stephanopoulos, G. (2000). Metabolic flux analysis. InBioreaction Engineering (pp. 106-124). Springer Berlin Heidelberg.

Klamt, S., Schuster, S., & Gilles, E. D. (2002). Calculability analysis in underdetermined metabolic networks illustrated by a model of the central metabolism in purple nonsulfur bacteria. *Biotechnology and Bioengineering*, 77(7), 734-751.

(Chapter II) Llaneras, F.(2011). Interval and possibilistic methods for constraint-based metabolic models. PhD Thesis. Universidad Politécnica de Valencia.

Stephanopoulos GN, Aristidou AA (1998). Metabolic Engineering: Principles and Methodologies. San Diego, USA: Academic Press.

Heijden RT, Romein B, Heijnen JJ, Hellinga C, Luyben KC (1994). Linear Constraint Relations in Biochemical Reaction Systems: I & II. *Biotechnology & Bioengineering*, 43(1):3–10.

About YALMIP and solvers

Löfberg, J. (2004, September). YALMIP: A toolbox for modeling and optimization in MATLAB. In Computer Aided Control Systems Design, 2004 IEEE International Symposium on (pp. 284-289). IEEE.

Löfberg, J. (2008, July). Modeling and solving uncertain optimization problems in YALMIP. In Proceedings of the 17th IFAC World Congress (pp. 1337-1341).

IBM ILOG CPLEX- High-performance mathematical programming engine. [http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/]

GLPK (GNU Linear programing kit) [http://www.gnu.org/software/glpk/]