Optimal allocation of bacterial resources in a bioreactor

Introduction to optimal control problems

MAM4 : Project report

Lélio Astruc & Nathan Edery

Optimal allocation of bacterial resources in a bioreactor

Introduction to optimal control problems

by

Lélio Astruc & Nathan Edery

Instructor :	Jean-Baptiste Caillau
With the help of :	Agustín Gabriel Yabo
Project Duration :	October, 2023 - December, 2023
Department :	Applied Mathematics and Modelling

Cover: Foocus (see this GitHub page)

Preface

Studying tiny life forms using resource sharing models has been a game-changer in batch bioprocessing. It helps one understand how bacteria behave naturally by figuring out how they divide their resources through smart control methods.

This report dives into batch bioprocessing but from a different angle—looking at how resources are used since we are the one poked around in this model to see how things change over time and if they stay stablecontrolling it through a mathematical theory. We've used a basic model for bacteria growth called the self-replicator model that considers what's happening inside the bioreactor. We made numerical resolution of PDEs using Julia and helped Mr. Yabo in the completion of his thesis (surement un peu ampbitieux).

Solving complex mathematical problems were interesting as we faced several problems that taught us how real-worlds mathematical research is.

Lélio Astruc & Nathan Edery
Biot, December 2023

Contents

1 Introduction

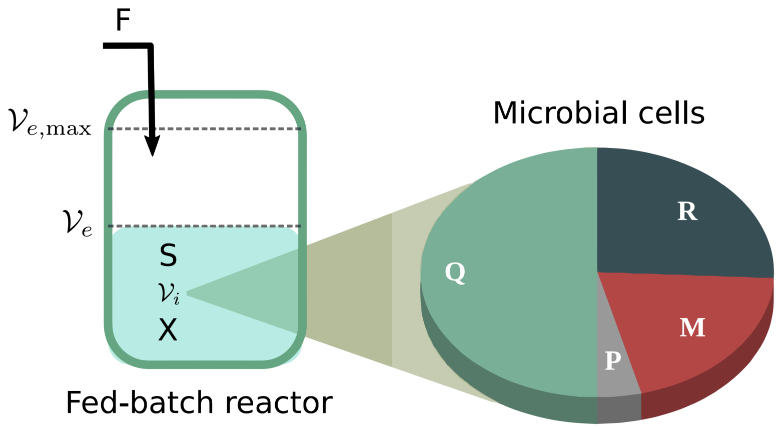
The exploration of microorganisms using resource allocation models has gained significance in understanding their behaviors via simplified dynamic models. This project illustrates how cellular resources are managed via optimal control theory. It's crucial for tackling various challenges, like optimizing metabolite production, or final volume, while controlling bacterial growth, which has practical applications in industries like food preservation and biofuel production.

This report focuses on batch bioprocessing from a resource allocation perspective, exploring different scenarios and focusing on the biomass maximization case.

2

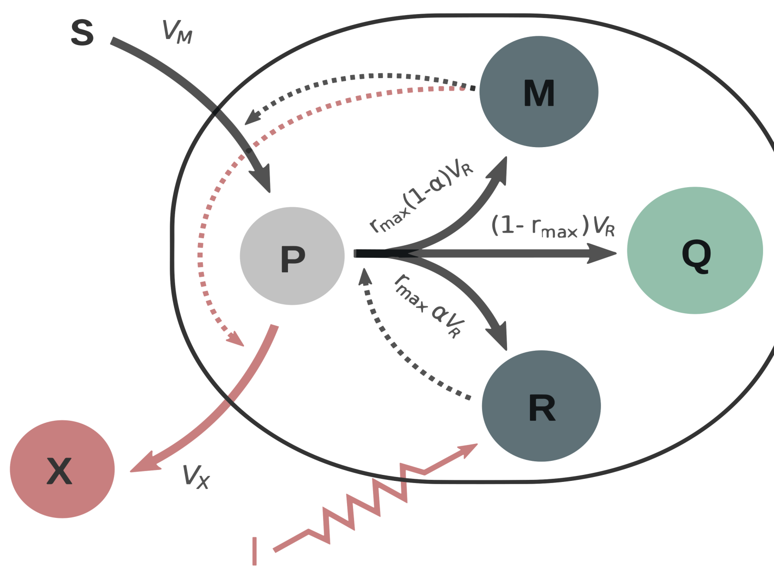
Definition of the model

Expliquer ce schéma



The self-replicator model

The self-replicator model illustrates the variation over time of a microbial population's caracteristics inside a bioreactor. We start with a constant volume $\mathcal{V}_e$ of microbial population and a certain mass S of substrat that will be "eaten" by the microbial population to be transformed in precursor metabolite called P. Those precurors produce proteins that are of 3 different types : M, R and Q.
The class M protein is responsible for the absorption of substrat S and production of precurors P and metabolites of interest called X.
The class R protein is representing ribosomes, handling the production of proteins of class M, R and Q.
The class Q protein is independant of the growth-rate and are the proteins maintning the cell and the ribosomes.



The self-replicator model

Solid arrows on the scheme represent a flow of resources while dashed arrows are a catalyzing effect, for example the production of proteins M, R and Q are all three catalyzed by the proteins of class R.
V_R represents the synthesis rate of intercellular proteins measured in grams per hour. r_{max} is an empirical constant that imposes a maximum to the rate of production of proteins M and R. The growth-rate of Q is in some way fixed whereas the balance between the one of R and M is decided by the control α (u ou α ?). We define $\alpha(t) \in [0, 1]$ where $\alpha = 0$ means there will be no production of proteins R and $\alpha = 1$ means there will be no production of proteins M.

The dynamics of the self-replicator model

The dynamics of the self-replicator system are described by

$$\begin{cases} \dot{S} &= -V_M \\ \dot{P} &= V_M - V_X - V_R \\ \dot{R} &= r_{max} u V_r \\ \dot{M} &= r_{max}(1-u)V_R \\ \dot{Q} &= (1-r_{max})V_R \\ \dot{X} &= V_X \end{cases} \tag{SRM-D}$$

$u(t)$ is the allocation control previously defined and we define the volume in liters of the microbial population $\mathcal{V}(t)$ as

$$\mathcal{V} \doteq \beta(M + R + Q)$$

where β is some real constant.
Following this principle, we can define the concentration varying over time as

$$p \doteq \frac{P}{\mathcal{V}} \qquad r \doteq \frac{R}{\mathcal{V}} \qquad m \doteq \frac{M}{\mathcal{V}} \qquad q \doteq \frac{Q}{\mathcal{V}} \qquad s \doteq \frac{S}{\mathcal{V}_e} \qquad x \doteq \frac{X}{\mathcal{V}_e}$$

We can also define

$$v_M(s, m) \doteq \frac{V_M}{\mathcal{V}} \qquad v_R(p, r) \doteq \frac{V_R}{\mathcal{V}}$$

Supposing that these synthesis rate of ribosomes and precurors are linear in m, we can define

$$v_M(s, m) = w_M(s)m \qquad v_R(p, r) = w_R(p)r$$

where $w_M(s)$ and $w_R(p)$ are *Michaelis-Menten* kinetics functions defined as

$$w_R(p) \doteq \frac{k_R p}{K_R + p} \quad \text{and} \quad w_M(s) \doteq \frac{k_M s}{K_M + s}$$

3

A glance at optimal control theory

Optimal control theory is a branch of mathematics and engineering that deals with finding the best control inputs to maximize or minimize a certain objective function, subject to a set of constraints. It's used in various fields, including engineering, economics, and biology, to determine the most efficient way to control a system.

At its core, optimal control involves a few key components :

* **System dynamics** : this deals with how the system evolves over time. We represent it using $PDEs$ in general.
* **Control inputs** : these are the actions one can make to influence the modifications of the system. Optimizating a criterion means to find the best control input possible.
* **Objective** : this is what we want to achieve (*e.g. maximizing a money profit, minimizing energy consumption*)

The main idea behind optimal control is to find the control inputs that minimize or maximize the objective while considering system dynamics and any constraints imposed on the system. The solution typically involves calculus of variations, dynamic programming, Pontryagin's maximum principle, or numerical optimization techniques.

Optimal control theory has applications in various fields, including robotics, aerospace engineering (like spacecraft trajectory optimization), economics (such as optimal resource allocation), and even in designing optimal medical treatments.

Here we use optimal control theory to maximize the volume $\mathcal{V}$ of the substrat (cf le modele).

Formulation of an optimal control problem

Let's consider the following problem,

$$\begin{cases} \dot{x} = f(t,x,u) \\ x(t_0) = x_0 \qquad x_0, t \in \mathcal{I} \end{cases}$$

where $\mathcal{I}$ is a compact interval of $\mathbb{R}$, f is a continuous function from $\mathcal{I} \times \Omega \times U$ to X where X is a *Banach* space. Ω is an open set of X and U a topological space. U is generally $\mathbb{R}^n$.
The variable x is the state and u is the control. We assume that $x \longmapsto f(t,x,u)$ is differentiable for all t and u.

$$J(u) = K(t_f, x_f) + \int_{t_0}^{t_f} \mathcal{L}(t, x(t), u(t))\, dt$$

where the lagrangian $\mathcal{L}$ satisfies the same conditions as f and K is differtiable on $\mathcal{V}_f$.

A simple example of an optimal control problem

Let's take the example of a plant growth. The factors that influence the growth of a plant are numerous and we obviously can't take them all in consideration. Yet, we can take some of the most important. Let's consider the height of the plant (h), the thickness of its stem (s), the concentration of chlorophyll (C) and lastly the average length of its leaves ($\bar{l}$).
Let's say we want to maximize its height using a control u being for example the light intensity over time. We model this as follows :

Let $u \in \mathbb{R}$, $x \in \mathbb{R}^4$, $t \in [t_0, t_f]$ and $x(t_0) = {}^t[h_0, s_0, C_0, \bar{l}_0]$

$$\begin{cases} \dot{x}(t) = F_0(x(t)) + u \cdot F_1(x(t)) \\ h(t_f) \longrightarrow max \end{cases} \tag{$*$}$$

Where $F_0(x)$ and $F_1(x)$ are vector fields.

We can solve this using numerical simulations that will give us the control u that satisfies the optimization problem.

4

The biomass maximization in a finite-time horizon

As we saw in chapter 3, optimal control problem are really useful for optimizing a criterion. In our model, the states that could be maximized or minimized would be the concentration of substrat (s), concentration of precursor metabolites (p), concentration of ribosomes (r) and the volume ($\mathcal{V}$).
The values of these quantities are expressed in

$$\Gamma = \left\{(s,\, p,\, r,\, \mathcal{V},\, x) \in \mathbb{R}^5 \mid s \geq 0,\, p \geq 0,\, 0 \leq r \leq 1\, \mathcal{V} \geq 0,\, x \geq 0\right\}$$

which allows us to fix the initial conditions

$$s(0) = s_0 > 0,\, p(0) = p_0 > 0,\, x(0) = 0, r(0) = r_0 \in (0,1),\, \mathcal{V}(0) = \mathcal{V}_0 > 0 \tag{IC}$$

The biomass maximization case

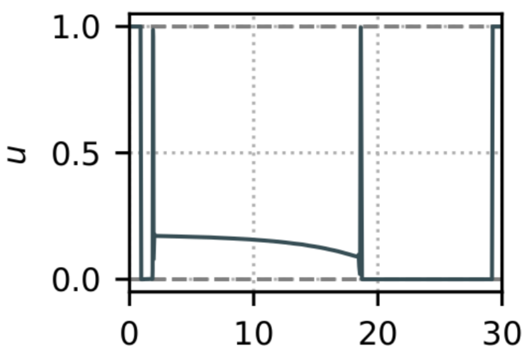
The following model fits both for an infinite time and a finite time horizon. We focused on making the numerical simulations for the finite-time horizon. In the Wild-Type Bacteria Model (WTB-M) it's assumed that no metabolite is produced since it's not naturally produced and only artificially added.

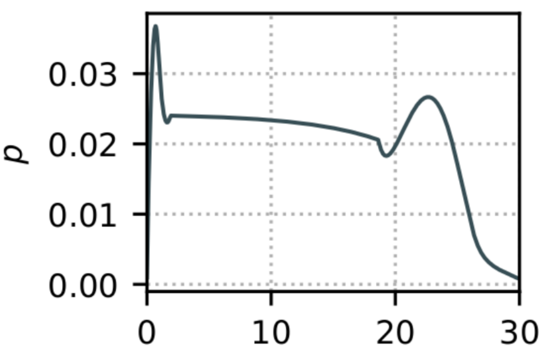
$$\begin{cases} \dot{s} & = -w_M(s)(1-r)\mathcal{V} \\ \dot{p} & = w_M(s)(1-r) - w_R(p)r(p+1) \\ \dot{r} & = (u-r)w_R(p)r \\ \dot{\mathcal{V}} & = w_R(p)r\mathcal{V} \end{cases} \tag{WTB-M}$$

Now that the model is defined, we can define the optimal control problem to be solved.

$$\begin{cases} \mathcal{V}(t_f) \longrightarrow max \\ \text{using the dynamics of (WTB-M)} \\ \text{using initial conditions (IC)} \\ u(\cdot) \in \mathcal{U} \end{cases} \tag{BM-OCP}$$

To numerically solve this *OCP* we used the language **`Julia`** that is really handful for mathematical computations as we will see in chapter 5. On top of that, our tutor's team at INRIA developped an interesting tool called **`controll-toolbox`**. Basically this **`Julia`** package solves *OCP* of our type as long as the problem is well defined. You can find the code we used in section A. The simulations for an infinite time horizon being already done, we did the one for a finite-time horizon and made sure the results were converging to the same limits.





Numerical simulations of (BM-OCP) with (IC) $s_0 = 0.1$, $p_0 = 0.001$, $r_0 = 0.1$, $\mathcal{V}_0 = 0.003$ and $t_f = 30$.

5

Computing the Lie brackets

What is a Lie bracket

On a vector space, *Lie* brackets is an internal composition law on V (meaning a *Lie* bracket of two vectors is still a vector) that satisfies the following properties :

* **Bilinearity** : $\forall x, x', y \in V$, $\forall \lambda,\ \mu \in K$

$$[\lambda x + \mu x', y] = \lambda[x, y] + \mu[x', y]$$

* **Alternation** : $\forall x \in V$

$$[x, x] = 0$$

* **Jacobi's relation** : $\forall x, y, z \in V$

$$[x, [y, z]] + [y, [z, x]] + [z, [x, y]] = 0$$

Why did we use Lie brackets

In the preprint produced by sir Yabo and sir Caillau, a propostion was left to prove. Here it is

Proposition 1 (***Order of singular extremals***)

If the Lie bracket F_{101} belongs to the span of F_1 and F_{01} , then singular extremals must be of (local) order at least two.

Since our problem is modeled as

$$\dot{x}(t) = F_0(x(t)) + u \cdot F_1(x(t))$$

we proved this proposition using *Lie* brackets with $F_{01} = [F_0, F_1]$ and $F_{101} = [F_1, F_{01}]$ and in our case $(V = \mathbb{R}^n (n = 4))$ **Lie** brackets are defined as follows :

$$[X, Y] = Y'X - X'Y$$

where $X, Y \in \mathbb{R}^n$ and X' is the Jacobian of X (same for Y).

Computing Lie brackets by hand

In (BM-OCP), we have vectors in $\mathbb{R}^4$, so the calculations of *Lie* brackets imply hand-computing 4×4 matrices determinants which can be quite tough (especially with non trivial functions). Even though this seemed hard, we gave it a try and quickly faced a wall : the matrices were not fitting in landscape layout.

Using symbolic mathematics

Having in mind our problem was probably coming from the fact that some rational functions were not simplifying with each other, we thought it would probably be better if we used symbolic computation to do it for us.
That's why we used the `Symbolics` in `Julia`. To prove the proposition, we adopted the following method : we calculated the *Lie* brackets F_{01} and F_{101}, then made sure that $rank(F_1, F_{01}) = rank(F_1, F_{01}, F_{101}) = 2$
We remember that

$$F_0 = \begin{pmatrix} -w_m(s)(1-r)V \\ w_m(s)(1-r) - w_r(p)r(1+p) \\ -w_r(p)r^2 \\ w_r(p)rV \end{pmatrix} \quad \text{and} \quad F_1 = \begin{pmatrix} 0 \\ 0 \\ w_r(p) \\ 0 \end{pmatrix}$$

We then have

$$F_{01} = \begin{pmatrix} -\frac{k_M k_R p r s v}{(K_M+s)(K_R+p)} \\ \frac{k_R p r \left(\frac{k_M s}{K_M+s} + \frac{k_R p(1+p)}{K_R+p} \right)}{K_R+p} \\ -\frac{r^2 k_R p \frac{k_R p}{K_R+p}}{K_R+p} + \frac{2r^2 p^2 k_R^2}{(K_R+p)^2} + \left(\frac{k_R r}{K_R+p} + \frac{-k_R p r}{(K_R+p)^2} \right) \left(\frac{k_M(1-r)s}{K_M+s} + \frac{-k_R p(1+p)r}{K_R+p} \right) \\ -\frac{p^2 k_R^2 r v}{(K_R+p)^2} \end{pmatrix}$$

and

$$F_{101} = \begin{pmatrix} -\frac{p^2 k_R^2 k_M r s v}{(K_R+p)^2(K_M+s)} \\ \frac{p^2 k_R^2 r \left(\frac{k_M s}{K_M+s} + \frac{k_R p(1+p)}{K_R+p} \right)}{(K_R+p)^2} \\ -\frac{k_R p \left(\frac{-r^2 k_R p \frac{k_R p}{K_R+p}}{K_R+p} + \frac{2r^2 p^2 k_R^2}{(K_R+p)^2} + \left(\frac{k_R p}{K_R+p} - \frac{k_R p r}{(K_R+p)^2} \right) \left(\frac{k_M(1-r)s}{K_M+s} - \frac{k_R p(1+p)r}{K_R+p} \right) \right)}{K_R+p} + \frac{k_R p r}{K_R+p}\Bigg(\text{simplifiable à la main} \\ -\frac{p^3 k_R^3 r v}{(K_R+p)^3} \end{pmatrix}$$

In a first time, to prove that $rank(F_1, F_{01}) = 2$ we just had to make sure that at least one determinants of the minor matrices was not null.
Then, to still have a rank of 2 when adding F_{101}, we need to make sure that all the determinants of the minors of $(F_1 \ F_{01} \ F_{101})$ null (else rank would be 3).

6 Conclusion

This three month project helped us understanding x main things.

A
Appendix

Here is the code we used to do the calculations on the *Lie* brackets and to solve the *OCP* in a finite-time horizon. The code does not exactly use the same syntax since symbols encoded in `utf-8` that are permitted in `Julia` are not allowed in the environment `lstlisting`. Hence, code like `t in [ t0, tf ], time` actually uses $\in$ instead of plain-text "in".

```
"""This is the code
for the ocp problem in
a finite time horizon"""
using OptimalControl

t0 = 0      # initial time
tf = 30     # final time
s0 = 0.1    # initial substrat
p0 = 0.001  # initial pdkd
r0 = 0.1    # initial
V0 = 0.003  # initial volume

@def ocp begin # definition of the optimal control problem

   t in [t0, tf ], time
   x in R^4, state
   u in R, control

   s = x_1
   p = x_2
   r = x_3
   v = x_4

   x(t0) == [s0, p0, r0, V0 ]

   s(t) ⩾ 0
   p(t) ⩾ 0
   0 ⩽ r(t) ⩽ 1
   v(t) ⩾ 0
   0 ⩽ u(t) ⩽ 1

   dot_x(t) == F0(x(t)) + u(t) * F1(x(t))

   v(tf) -> max

end;

# Dynamics
```

```julia
const k_r = 1.1
const k_m = 1.2
const K_r = 1.3
const K_m = 1.4


w_r(p) = k_r * p / (K_r + p)
w_m(s) = k_m * s / (K_m + s)

F0 = VectorField( phi -> begin
   s, p, r, V = phi
   return [-w_m(s) * (1 - r) * V
             w_m(s) * (1 - r) - w_r(p) * r * (p + 1)
            -w_r(p) * r^2
             w_r(p) * r * V ]
end )

F1 = VectorField( phi -> begin
   s, p, r, V = phi
   return [0, 0, w_r(p) * r, 0 ]
end )

direct_sol1 = solve(ocp, grid_size=100)

direct_sol2 = solve(ocp, grid_size=1000)

plt1 = plot(direct_sol1, size=(600, 600))
plt2 = plot(direct_sol2, size=(600, 600))
```

```julia
"""Symbolic computation of Lie brackets
for 4x4 matrices determinants"""
using Symbolics

@variables s, p, r, v, k_r, k_m, K_r, K_m

w_r = k_r * p / (K_r + p)
w_m = k_m * s / (K_m + s)

F0 = [-(k_m * s / (K_m + s)) * (1-r)*v,
 (k_m * s / (K_m + s))*(1-r) - (k_r * p / (K_r + p)) *r * (p+1),
 -(k_r * p / (K_r + p))*r^2,
 (k_r * p / (K_r + p))*r*v]
F1 = [0, 0, (k_r * p / (K_r + p))*r, 0]

# Calcul du crochet de Lie de F0 et F1

F0_prime = Symbolics.jacobian(F0, [s, p, r, v])
F1_prime = Symbolics.jacobian(F1, [s, p, r, v])

F01 = F1_prime * F0 - F0_prime * F1

# Calcul du crochet de Lie de F1 et F01

F01_prime = Symbolics.jacobian(F01, [s, p, r, v])

F101 = F01_prime * F1 - F1_prime * F01

# Calcul du rang de la matrice F1 et F01 :
mat_F1_F01 = [F1 F01]
```

```
# Sous matrice de mat_F1_F01
sousDet1_F1_F01 = [mat_F1_F01[1] mat_F1_F01[5] ; mat_F1_F01[2] mat_F1_F01[6]]
sousDet2_F1_F01 = [mat_F1_F01[1] mat_F1_F01[5] ; mat_F1_F01[3] mat_F1_F01[7]]
sousDet3_F1_F01 = [mat_F1_F01[1] mat_F1_F01[5] ; mat_F1_F01[4] mat_F1_F01[8]]
sousDet4_F1_F01 = [mat_F1_F01[2] mat_F1_F01[6] ; mat_F1_F01[3] mat_F1_F01[7]]
sousDet5_F1_F01 = [mat_F1_F01[2] mat_F1_F01[6] ; mat_F1_F01[4] mat_F1_F01[8]]
sousDet6_F1_F01 = [mat_F1_F01[3] mat_F1_F01[7] ; mat_F1_F01[4] mat_F1_F01[8]]

mineur1_F1_F01 = Symbolics.det(sousDet1_F1_F01)
mineur2_F1_F01 = Symbolics.det(sousDet2_F1_F01)
mineur3_F1_F01 = Symbolics.det(sousDet3_F1_F01)
mineur4_F1_F01 = Symbolics.det(sousDet4_F1_F01)
mineur5_F1_F01 = Symbolics.det(sousDet5_F1_F01)
mineur6_F1_F01 = Symbolics.det(sousDet6_F1_F01)

# Calcul du rang de la matrice F1 F01 et F101
mat_F1_F01_F101 = [F1 F01 F101]

# Sous matrice de mat2
sousDet1_F1_F01_F101 = [mat_F1_F01_F101[1] mat_F1_F01_F101[5] mat_F1_F01_F101[9] ;
mat_F1_F01_F101[2] mat_F1_F01_F101[6] mat_F1_F01_F101[10] ; mat_F1_F01_F101[3]
mat_F1_F01_F101[7] mat_F1_F01_F101[11]]

sousDet2_F1_F01_F101 = [mat_F1_F01_F101[1] mat_F1_F01_F101[5] mat_F1_F01_F101[9] ;
mat_F1_F01_F101[2] mat_F1_F01_F101[6] mat_F1_F01_F101[10] ; mat_F1_F01_F101[4]
mat_F1_F01_F101[8] mat_F1_F01_F101[12]]

sousDet3_F1_F01_F101 = [mat_F1_F01_F101[1] mat_F1_F01_F101[5] mat_F1_F01_F101[9] ;
mat_F1_F01_F101[3] mat_F1_F01_F101[7] mat_F1_F01_F101[11] ; mat_F1_F01_F101[4]
mat_F1_F01_F101[8] mat_F1_F01_F101[12]]

sousDet4_F1_F01_F101 = [mat_F1_F01_F101[2] mat_F1_F01_F101[6] mat_F1_F01_F101[10] ;
mat_F1_F01_F101[3] mat_F1_F01_F101[7] mat_F1_F01_F101[11] ; mat_F1_F01_F101[4]
mat_F1_F01_F101[8] mat_F1_F01_F101[12]]

mineur1_F1_F01_F101 = Symbolics.det(sousDet1_F1_F01_F101)
mineur2_F1_F01_F101 = Symbolics.det(sousDet2_F1_F01_F101)
mineur3_F1_F01_F101 = Symbolics.det(sousDet3_F1_F01_F101)
mineur4_F1_F01_F101 = Symbolics.det(sousDet4_F1_F01_F101)
```

Bibliography

[1] AGUSTÍN GABRIEL YABO, JEAN-BAPTISTE CAILLAU, JEAN-LUC GOUZÉ. **OPTIMAL BACTERIAL RESOURCE ALLOCATION STRATEGIES IN BATCH PROCESSING**, 2022.

[2] Agustin Gabriel Yabo, Jean-Baptiste Caillau, Jean-Luc Gouzé. Optimal allocation of bacterial resources in fed-batch reactors. 2022 European Control Conference (ECC), Jul 2022, London, United Kingdom. pp.1466-1471, 10.23919/ECC55457.2022.9838346 . hal-03421358v2

[3] Crochet de *Lie*

[4] Control-toolbox

[5] Optimal Control Theory

[6] The Julia Programming Language

[7] The Symbolics package