Contents

Chapter 1

Building the physiological context module

Sub modules are sub parts of one single module and are thus integrated simultaneously. The main reasons we decide to divide a module into sub modules are that first, we can exchange some modules with more detailed description as the knowledge about this process or your needs are changing. Secondly, one is now able to test them independently before he puts them together in order to build a bigger model.

1.1 Build the modularity tree (fig 1.1)

The first step is to divide the physiological context module into several sub modules we can individually test easily and that correspond to different classes of reactions.

Ideally the different sub modules: *expression*, *competition*, etc. should be picked such as minimizing the number of shared variables.

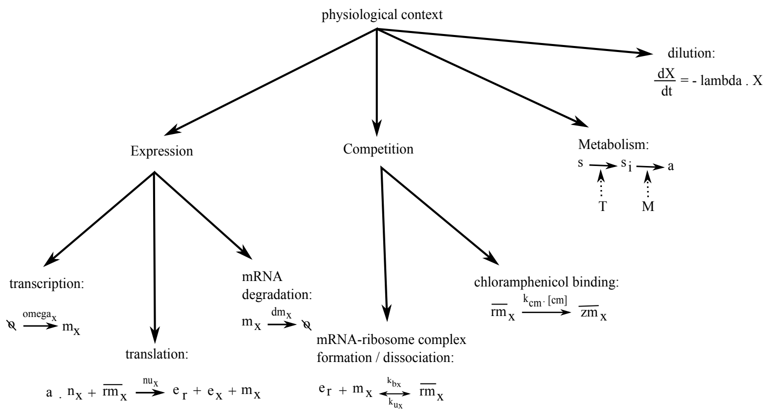
physiological context
dilution:
$\frac{dX}{dt} = - \text{lambda} . X$
Expression
Competition
Metabolism:
$s \rightarrow s_i \rightarrow a$
T
M
transcription:
$\varnothing \xrightarrow{\text{omega}_x} m_x$
translation:
$a . n_x + \overline{rm}_x \xrightarrow{\text{nu}_x} e_r + e_x + m_x$
mRNA degradation:
$m_x \xrightarrow{\text{dm}_x} \varnothing$
mRNA-ribosome complex formation / dissociation:
$e_r + m_x \underset{k_{ux}}{\overset{k_{bx}}{\rightleftarrows}} \overline{rm}_x$
chloramphenicol binding:
$\overline{rm}_x \xrightarrow{k_{cm} \cdot [cm]} \overline{zm}_x$

Figure 1.1: Schematic representation of the Weisse-Swain core model. This model is divided into 4 modules: expression, competition, metabolism and dilution. The expression and competition sub modules are themselves divided into three and two sub modules respectively. The modules are built from bottom to top

1.2 Implementation of the model

1.2.1 Defining the main object classes of the model

The different objects:

An object P in MATLAB is defined like:
$P = struct();$
Then several variables can be defined inside this object:
$P.variable1 = 12;$
$P.variable2 = 0.5;$
In order to implement the model, the parameters shall be an object called for example P and the species that are integrated another object S. Another object could be V for the dynamic parameters or auxiliary functions that are used during the integration such as $V.\gamma$ which has to be updated at each step of the integration as it depends on the amount of the species $S.a$ and $S.rm_x$ where $x \in [r, t, m, q]$. Those object can be passed as input into a function.

Dedicated functions for definition of those objects:

Each of those object can be defined in a separate function to facilitate the addition of new parameters, species or variables:
$function\ P = parameters()$
$P.nx = 300$
$P.M = 10^8$
...

1.2.2 Generic functions

Generic functions that are used many times in the model such as hill functions can have a dedicated function in order to facilitate the implementation.

1.2.3 Mapping indices to objects

An effort shall be made in order to avoid using indices to call species as it will become increasingly difficult to remember to what specie correspond a particular index of the integration vector as the number of species in the model increases. In order to avoid to do so, a function that maps the indices of the integration vector to the name of the species and reciprocally the name of species to the indices shall be built. The aim is, when writing the differential equations to use only the "human" names of the species $S.a, S.s\_i, ...$ and not $X(1), X(2), ...$

1.2.4 Initial Conditions

The initial conditions might also be set in a dedicated function so it is easy to change their values.

1.2.5 observables

At the end of the integration another dedicated function can be made in which computations of different observables are made such as the ribosomal mass fraction Φ_r.

1.2.6 Tests

A "*Tests.m*" shall be made to perform several tests that come from hand derivation on the equations of the core model to check whether it is implemented correctly or not. See the following section on the unit testing methodology.

1.3 Unit testing

The philosophy behind unit testing is to make sure that the model you are thinking about is correctly implemented. In order to do so, one have to think about the expected behaviour or equalities that should be verified if the thought model is implemented correctly. Here several examples of tests are described for each module of the core model. The final tests are the one that are used in the implemented function " *Tests.m*".

1.3.1 Expression

Transcription: In an isolated transcription module we expect the number of mRNA of type x to grow linearly with a rate ω_x:

$$\frac{dm_x}{dt} = w_x \cdot \frac{a}{\Theta_x + a} \tag{1.1}$$

We can test if this is accurate by varying the number of a and check if the growth of mRNAs x follows this equation.

Translation: In this module the number of protein grows linearly and is given by $\nu(\overline{rm_x}, a)$:

$$\nu(\overline{rm_x}, a) = \overline{rm_x} \cdot \frac{\gamma(a)}{n_x} \tag{1.2}$$

where $\gamma(a) := \dfrac{\gamma_{max} \cdot a}{K_\gamma + a}$
We can test the accuracy of the module by measuring the growth of the number of protein x in different settings of $(\overline{rm_x}, a)$

degradation: the number of a mRNA x goes to 0 at a rate $m_x \cdot dm_x$

1.3.2 Competition

mRNA-ribosome binding: at steady-state we must have:

$$\frac{kb_x}{ku_x} = \frac{e_r \cdot m_x}{\overline{rm_x}} \tag{1.3}$$

chloramphenicol binding: Starting from initial conditions with some $\overline{rm_x}$ the number of complexes $\overline{zm_x}$ should increase linearly with a slope proportional $k_{cm} \cdot [cm]$ before stabilizing when all the $\overline{rm_x}$ is consumed.

1.3.3 Metabolism

nutrient import: Starting with some transporters e_t and fixing the external source of nutrient s, the number of inside nutrient s_i should increase linearly. The slope of this increase follows a sigmoid function $\zeta_{v_t, K_t(e_t, s)}$ that takes as input the number of e_t and s. When fixing $s = 0$ the slope should be equal to $0 \Longrightarrow$ no sugar import. When giving to s a really high value, the value of the slope should reach its maximum at v_t.

nutrient metabolism: The production rate of a follows a similar sigmoidal function of the form $n_s.\zeta_{v_m,K_m}(e_m, s_i)$ where n_s is the nutrient efficiency. We thus expect a to grow linearly for any initial input of s, the slope will depends linearly on n_s. If $e_t \cdot v_t >> e_m \cdot v_m$ then a huge increase in s will not saturate the increase of the production slope of a. if $s = 0$ there will be no production of a.

1.3.4 Connecting the expression sub modules

Test 1 - Upon connecting the three expression sub modules we would expect the following behaviour. The translation sub module should behave the same as if it was isolated as the other modules don't impact its behaviour. Given $a > \sum_x \overline{rm_x} \cdot n_x$, all the $\overline{rm_x}$ will disappear with a rate:

$$\nu_x(\overline{rm_x}, a) = \overline{rm_x} \cdot \frac{\gamma(a)}{n_x} \tag{1.4}$$

Protein numbers e_x will increase linearly before saturating as $\overline{rm_x}$ is consumed.

Test 2 - if $a < \sum_x \overline{rm_x} \cdot n_x$ then the messenger numbers m_x will increase over time before starting to go down when $m_x \cdot dm_x > \omega_x(a)$ because of the translation consumption of a. On the opposite, if there is still some a after all the $\overline{rm_x}$ is consumed, then m_x will increase before stabilizing at steady state following:

$$m_x = \frac{\omega_x}{dm_x} \tag{1.5}$$

We also expect that connecting transcription with translation increases in a first place the production of m_x as some of them are released during the translation reaction.

1.3.5 Connecting the competition sub modules

Test 1 - Starting with $e_r > \sum_x m_x$ and m_x, for $k_{cm} \cdot [cm] > 0$ we expect the number of complexes $\overline{zm_x}$ to first increase linearly before stabilizing such that:

$$\sum_x \overline{zm_{x,t_{end}}} = \sum_x m_{x,t_0} \tag{1.6}$$

with:

$$e_{r,t_{end}} = e_{r,t_0} - \sum_x m_{x,t_0} \tag{1.7}$$

Where e_x with $x = [r, t, m, p, q]$ is the protein of type x

Test 2 - Starting with $e_r < \sum_x m_x$ and m_x, for $k_{cm} \cdot [cm] > 0$ we have :

$$\sum_x \overline{zm_{x,t_{end}}} = e_{r,t_0} \tag{1.8}$$

$$\sum_x m_{x,t_{end}} = \sum_x m_{x,t_0} - e_{r,t_0} \tag{1.9}$$

1.3.6 Connecting the expression and competition modules

Test 1 - for $k_{cm}.[cm] > 0$ we expect m_x and $\overline{rm_x}$ to go to 0 as a progressively disappears until there is none of it left. We thus expect at steady state to have:

$$a_{t_0} = \sum_x e_{x,t_{end}}.n_x \tag{1.10}$$

Test 2 - We expect $\overline{zm_x}$ to increase before stabilizing after all the a and $\overline{rm_x}$ are consumed. We also expect $e_{x,t_{end}}$ and $\overline{zm_{x,t_{end}}}$ to go up as we increase the initial number of a.

Test 3 - given a same a, during the dynamics leading to steady state, m_x should always be lower in the expression+competition than in the expression alone as some m_x are consumed formation of the $\overline{rm_x}$ complex.

1.3.7 Connecting with the metabolic module

Test 1 - The number of each protein should increase in an exponentially way as there are no way to degrade them and the more e_t, e_m there are, the more a are produced per time unit, the more e_r there are, the more proteins are produced per time unit.

Test 2 - for $s_{t_0} > 0$, $a_{t_0} > 0$ and $e_{r,t_0} > 0$, we have:

$$\dot{m_x} = \omega_x(a) + \nu_x(\overline{rm_x}, a) + k_{u_x} \cdot \overline{rm_x} - k_{b_x}.e_r - dm_x \cdot m_x \tag{1.11}$$

$$\dot{a} = \zeta_{v_m K_m}[e_m, \zeta_{v_t K_t}(e_t, s)] - n_x \cdot \nu_x(\overline{rm_x}, a) \tag{1.12}$$

1.3.8 Tests for the assembled WS core model

Test 1 - at Stationary state:

$$\lambda \cdot \frac{M}{n_s} = e_m \cdot \frac{v_m}{1 + \frac{K_m}{s_i}} = e_t \cdot \frac{v_t}{1 + \frac{K_t}{s}} \tag{1.13}$$

Test 2 -

for $x \in [r, t, m, p]$

$$\lambda \cdot M \cdot \beta_x = \gamma \cdot \frac{w_x}{1 + \frac{\theta_x}{a}} \times \frac{1}{\lambda + (\lambda + dm_x) \cdot \frac{(\lambda + k_{u_x} + \frac{\gamma}{n_x})}{k_{b_x} \cdot e_x}} \tag{1.14}$$

for $x = q$

$$\lambda \cdot M \cdot \beta_x = \gamma \cdot \frac{w_x}{1 + \frac{\theta_x}{a}} \times \frac{1}{1 + (\frac{e_q}{K_q})^\alpha} \times \frac{1}{\lambda + (\lambda + dm_x) \cdot \frac{(\lambda + k_{u_x} + \frac{\gamma}{n_x})}{k_{b_x} \cdot e_x}} \tag{1.15}$$

Test 3 -

$$\beta_{rfree} + \beta_q + \beta_p + \lambda \times [\frac{(K_t + s)}{s} \times (\frac{w_m}{w_t} + 1) \times \frac{n_t}{n_s \cdot v_t} + \frac{n_r}{\gamma}] = 1 \tag{1.16}$$

Test 4 - The dynamic mass at stationary state is the same that the mass $p.M$ we input as a parameter:

$$\sum_x \overline{rm_x} \cdot n_r + \sum_x e_x \cdot n_x = p.M \tag{1.17}$$