Numerical model of the forced expiratory manoeuvre for detecting respiratory pathologies

R.Grebert[a]

advisor: M.Filoche[b]

a. Mines ParisTech, raphael.grebert@mines-paristech.fr
b. Ecole Polytechnique, marcel.filoche@polytechnique.edu

Abstract

Keywords: *respiratory system, tracheobronchial tree, forced expiratory manoeuvre, COPD*

Background Respiratory diseases are a topical health problem particularly difficult to treat because of the lack of information easily available about a patient's pulmonary system. Before resorting to the most costly and complex modern imagery techniques (such as functional MRI or high-resolution CT-scan), it is essential to have at one's disposal investigation techniques able to assist diagnosis in the clinical practice. We hypothesized we could develop a less complex model, based on the geometrical property of the tracheobronchial tree, that could allow us to reproduce forced expiration of a human respiratory system. We aim at using it to better understand the functioning of the lungs and to simulate and even help recognize widespread respiratory pathologies.

Method Based on morphological measurements, we first build a physical model of the tracheobronchial tree. Then, after integrating the Navier-Stokes equations under simplifying asumptions, we obtain a set of equations and variables describing the state of the respiratory system at any given time. After implementing an algorithm based on these physical constraint, we use a Newton-Raphson method to solve the steady state problem. We reproduce the forced expiratory manoeuvre with a temporally discrete semi-dynamic method, each initialisation depending on the state of the system after the precedent iteration to obtain a realistic spirometry. This model opens the possibility to simulate model pathologies by altering the physical and physiological parameters that describe the mechanical behaviour of the pulmonary airway system.

Results The developed algorithm managed to perform the forced expiratory manoeuvre for a 12-generation tree in about 2 days (1h30 for a 10-generation one). The temporal variations of the expired air flow is realistic and can be compared to experimental data. Thus, the model successfully reproduces spirometry for healthy individuals and for widespread respiratory pathologies. We could visualize how the pressure and air flow vary in the lungs, and how the airway system behaves during the expiration. The effect of the initial geometrical and physiological parameters also makes this method patient specific, and can help bring some insight about the behaviour of an individual's particular respiratory system. The main limit during the study was the computational capacity. The python algorithm could not model more than 12 generations because of computation time matters. The effect was a too brief expiration and the impossibility to accurately simulate some pathologies which affect the lower airway (generations above 12th). We believe these problems of computation time can be solved by transferring the code from Python to C++. Still, the method and algorithm show great results relative to their simplicity. As it is very promising, it definitely deserves further enhancement and optimisation.

Conclusion Flow-volume loop of the forced expiratory manoeuvre can be reproduced with simple analytical model and non-linear equation solving algorithm. The obtained data can be used to model the behaviour of individual's respiratory system. The flow-volume loops obtained for different pathology simulations could help us provide training data for a neural network in order to automatically diagnose pulmonary diseases on real data, with high performances.

Introduction

Respiratory diseases are a one of the main causes of mortality among adult populations worldwide [1], counting for about 8% of all annual deaths in Europe [2]. If causes such as harmful particle absorption or, more recently, infectious diseases are sources of respiratory issues, chronic obstructive pulmonary disease (COPD) are among the most life-threatening conditions, being the 4th leading cause of deaths worldwide with an estimated prevalence of 11% for individuals above the age of 30 [3]. Worsening the consequences of lung cancer [4], COPD are a major health issue which requires new treatment approaches.

The most commonly used technique to study and diagnose the mechanical property of the pulmonary airway system is spirometry, an experimental process measuring the pulmonary flow via the mouth. Doing so, it allows to characterize the functioning of the patient's respiratory system but still does not provide direct information about the mechanics of the tracheobronchial tree. It is thus necessary to develop new methods, thanks to the progress of bio-imaging, computer science and bio engineering, to allow better drug delivery procedures, damaged tissue identification and pathology diagnosis.

In particular, numerical simulations are a comprehensive way to model the respiratory system based on precise anatomy and to simulate its behaviour using physical and mathematical models. While the latest computational heavy simulation use lung mesh modelling and applied computational fluid dynamics (CFD) [5], the goal of this study is to elaborate a simple and efficient numerical model of the human lung expiration to reproduce realistic and patient-specific spirometry, with a computation time compatible with the clinic practice. This algorithm should first help us study the lung system behaviour, and then help us simulate pathology's effect on the forced expiration, which we will use to train supervised learning algorithm for pathology detection.

In order to do so, we will first present the adequate physical and mathematical models describing

the tracheobronchial tree behaviour, before building and optimising an algorithm. Then, we will run pathology simulation and detection, and investigate how our model can be used to simulate the effect of widespread pathologies on the respiratory system, and to provide training data for a learning algorithm.

Model

1 Physical Model

Building our algorithm to simulate a forced expiration requires a physical model for the human lungs and the expiration process. In this part, we will introduce the main hypotheses and models from the literature to build ours.

1.1 Anatomy

The respiratory airway system of the human body is responsible for gas exchange with the environment, that is providing the blood with dioxygen and eliminating carbon dioxide. This complex structure can be classified in **three main areas** (Fig 2) [6]. The first one is the **extrathoracic region**, including the airways between the oral cavity and the upper trachea. Then, the **tracheobronchial region** covers the trachea down to the transitional bronchioles (generation 15). Then, the **acinar region** (generation 16-23), or deep lung region is the one responsible for the gas exchange, in the alveolar sacs located in generations 20-23.

The tracheobronchial tree is a conducting zone, linking the entrance of the lungs to the acinar region. This is the zone we will study and on which we will work with our model. This is justified by the fact the acinar region amounts to less than 20% of the total flow resistance of the respiratory system. It thus has a smaller contribution to the expiratory manoeuver behaviour and will be neglected to drastically reduce the amount of variables.

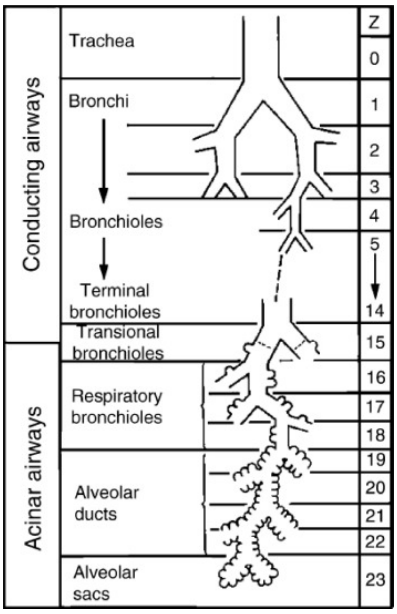


Fig 1, Schematic of the branching airways in Weibel's model (Weibel et al., 2005).

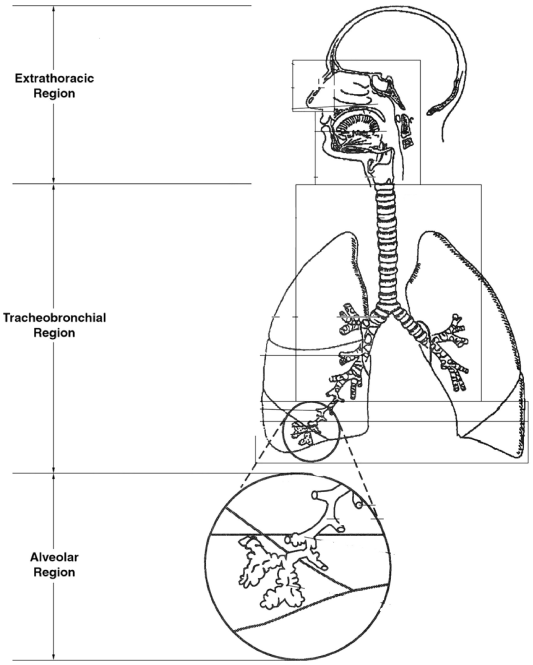


Fig 2, Schematic of the respiratory system from Hofmann (2011) [16].

The tracheobronchial tree, below the trachea, is in charge of transporting the air from the extra-thoracic airways to the gas exchange region. It is similar to a tree (Fig 1) as it is composed of airways which ramify at each generation: a main branch divides in two others, ect... creating a fractal binary tree of air conducts. These **vessels or bronchi** are connected by bifurcations that we will call "**nodes**".

1.2 Geometry

To build our model, we use Weibel's description of the geometry [6], which provides a relationship between the diameters of air conducts at each intersection, and relates the length and the diameter of a conduct. The fractal nature of this structure makes it natural to describe the geometry using the ratio between the lengths of successive generations instead of using their absolute value. The tracheobronchial tree is asymmetrical, meaning that the diameters of both subsequent conducts are different. Thus, we define at each intersection a quantity h_{min} and a quantity h_{max}, respectively equal to the smallest and largest ratios (Fig 3).

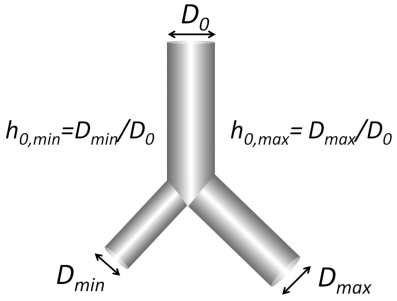


Fig 3, geometrical model of Florens et al., 2011 [7].

These ratios verify the **Murray-Hess law** (Eq. 1). In a symmetric tree, this would give us $h_{min} = h_{max} = 2^{-1/3}$.

$$D_0^3 = D_{min}^3 + D_{max}^3 \tag{1}$$

In our case, we will take the parameters of Florens et al. [7], that give us both these coefficients against the generation, and also the coefficient L/D which links the length of each conduct to its diameter (Table 1).

Generation	Coefficient h_{min}	Coefficient h_{max}	Coefficient L/D
n=1	0.87	0.69	3.07
n=2	0.80	0.67	1.75
n=3	0.83	0.67	1.43
n=4	0.86	0.74	1.85
n>=5	0.87	0.67	3.0

Table 1, Geometrical parameters of the model.

With this data, we can now build a numerical **lung model, consisting of successive generations of conducts which branch in a binary way at each of their ends.**

1.3 Expiration behaviour

Our model is **pressure-driven**, as the conditions imposed on the system are the alveolar pressure, i.e. the pressure inside the airways at the distal end of the tree, and the pleural pressure, i.e. the pressure in the lung, outside the airways. We **neglect here the influence of gravity** and assume the pleural

pressure to be **homogeneous** in the lungs, outside of the air conducts. A more detailed model in the future could account for a non-homogeneous pleural pressure whose gradients are created by the spatial variation of hydrostatic pressure.
The behaviour of the model is thus dictated by those pressures, the **pleural pressure**, P_{pl} (Eq. 3 and Eq. 4) and the **alveolar pressure**, P_{alv} (Eq 2). Indeed, the airway's compliance and the fluid's equation are driven by the **transmural pressure** (Fig 4), $P_{tm} = P_{in} - P_{pl}$, P_{in} being the pressure inside the airways (equal to P_{alv} at the end of the tree).

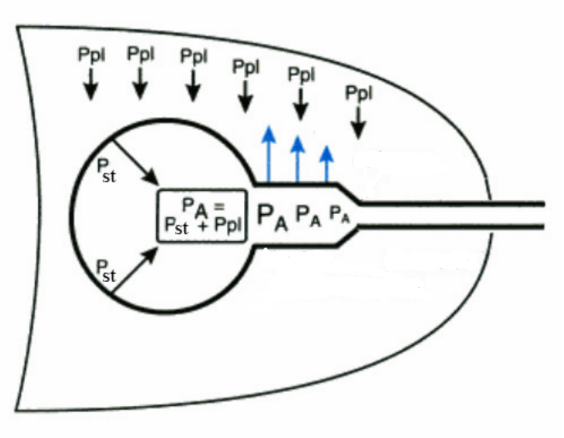


Fig 4, Schema of the lung's pressure, H.Guénard [10].

The temporal behaviour of these variables can be described thanks to the work of Polak [8,9] in which we can find:

$$P_{alv} = P_m(1 - e^{-t/\tau})(\frac{V_L(t) - RV}{VC}) - R_T.\phi_0(t) \tag{2}$$

$$P_{pl} = P_{alv} - P_{st} \tag{3}$$

where P_{st} is the static recoil pressure,

$$P_{st} = \frac{V_{max} - V_{min}}{C_{L0}} ln(\frac{V_{max} - V_{min}}{V_{ma}x - V_L(t)}) \tag{4}$$

P_m the maximal expiratory pressure, produced by the intercostal muscles and elasticity
VC the vital capacity
R_T non linear resistance caused by respiratory tissue resistance
τ is the time constant
$\phi_0(t)$ the total air flow at time t
V_{max} and V_{min} the maximum and minimum lung volume
$V_L(t)$ the lung volume at time t
C_{L0} the lung compliance during expiration at zero recoil pressure

1.4 Airway compliance

As the bronchus are not rigid, we use a constitutive law to model the behaviour of the tube with respect to the transmural pressure (Fig 5). Indeed, the compliance of the pulmonary airways has an important effect on their flow resistance.

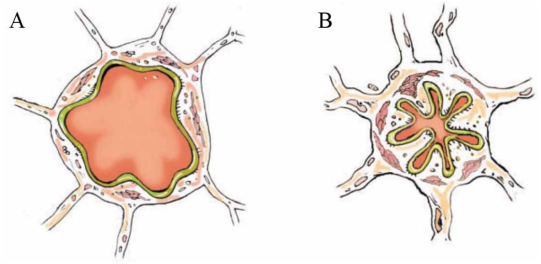


Fig 5, A bronchus under positive (A) and negative (B) transmural pressure [14].

To model the **relationship between the airway's diameter *D* and the transmural pressure** P_{tm}, we use the work of Lambert et al. [11]. It provides us with two analytical functions (Eq. 5 and Eq. 6) to describe the state of the airways.

$$D(P_{tm}) = D_{max}\sqrt{\alpha_0(1-\frac{P_{tm}}{P_1})^{-n_1}}, \qquad P_{tm} < 0 \tag{5}$$

$$D(P_{tm}) = D_{max}\sqrt{1-(1-\alpha_0)(1-\frac{P_{tm}}{P_2})^{-n_2}}, \qquad P_{tm} >= 0 \tag{6}$$

Where $\alpha_0, P_1, P_2, n_1, n_2$ are adjustable parameters tabulated that depend of the airway's generation. These analytical functions can be thus represented generation-wise (Fig 6).

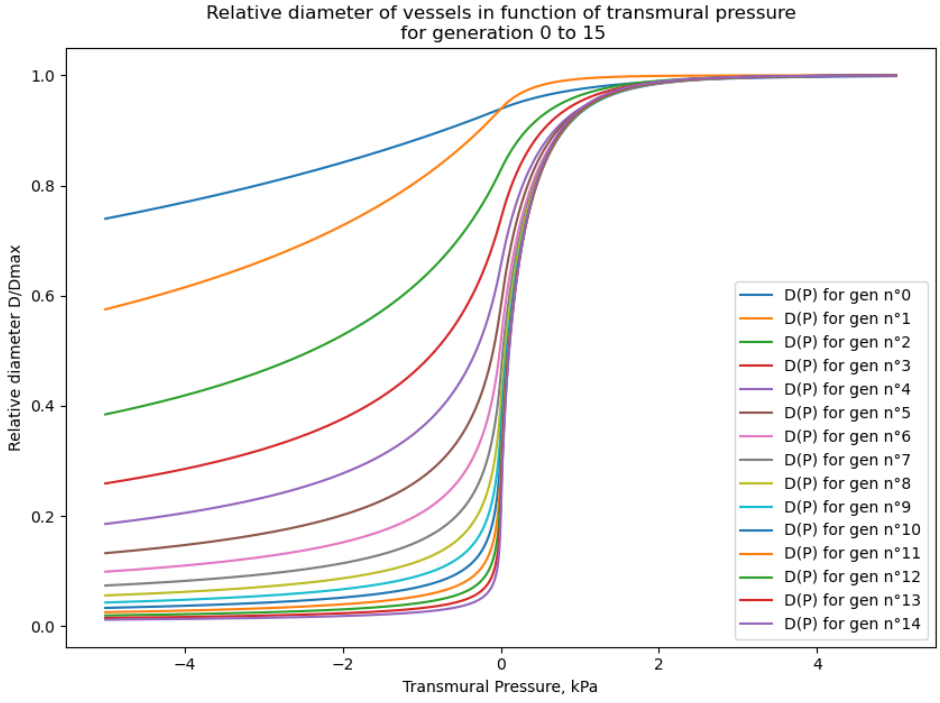


Fig 6, Bronchus compliance functions. [2]

2 Mathematical Model

Lets now take a look into the equations that our algorithm has to solve in order to find the variables defining our system for every time step.
As previously said, our system is pressure driven. To know the mechanical state of the airways at a

certain time, we therefore need to know the pressures at each end of the conducts, and the air flow in each conduct. These will be our **variables**, numerically implemented as two vectors **P (pressures at each node) and Q (flow rates in each airway)**. For any given airway, P_{in} **and** P_{out} designate the entrance and exit **transmural** pressures, respectively. As the air flow is oriented towards the oral cavity during the expiration, leaving the body, please note that the **entrance is actually the distal end of the bronchus, while the exit is the proximal end**.
The boundary conditions at each step are the pleural pressure P_{pl}, the alveolar pressure P_{al} (and thus the transmural pressure at the end of the tree) and the outside pressure at the oral cavity, which will be set at the reference of 0 Pa.
We will now present the equations we need to solve and the solving method.

2.1 Flow equation

To find all the flow rates and pressures, one pressure equation per airway and one flow equation per node are needed.

Air flow equation The flow conservation equations are the **kirchhoff** laws. For each node, where are connected the conduct n_i of generation k and its subsequent conducts n_j and n_{j+1} of generation k+1, the flow rate solve the Kirchhoff equation (Eq 7):

$$Q_{n_i} - Q_{n_j} - Q_{n_{j+1}} = 0 \tag{7}$$

Extra-thoracic pressure equation The extra-thoracic region is modelled by a single pressure drop (Eq 8) according to an empiric non-linear resistance [12]. The formula is non linear to account for the inertial effects due to the high Reynolds's number of respiratory system's region:

$$\Delta(P) - \Phi_0^r R_{ext} = 0 \tag{8}$$

with R_{ext} the estimated flow resistance of the extra-thoracic region, r an exponent set at 1.68, and Φ_0 the total flow rate. The values are tabulated from [12].

Lung pressure equation The model we use for the airway's pressure equation was developed by Florens and Filoche [13], which we will summarize briefly.
Our hypotheses are a **steady flow in the collapsible tube** and a **mostly one-dimensional flow along the longitudinal coordinate x of the conduct** (Fig 7).

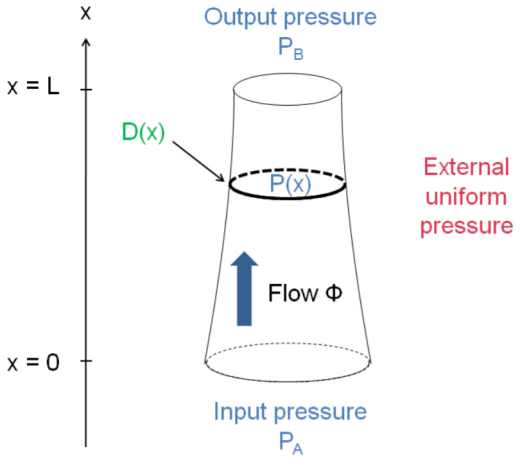


Fig 7, Schematic representation of the flow in one elementary flexible pipe of the network [13].

Thus, we can use the **Bernoulli equation for a steady and incompressible one-dimensional flow** (Eq 9), bewteen the entrance *"in"* and the abscissa x inside the conduct :

$$P(x) = P_{in} + \frac{1}{2}.\rho.v_{in}^2 - \frac{1}{2}.\rho.v(x)^2 - \int_{in}^{x} f(y)\,dy \tag{9}$$

where $f(x)$ accounts for the **energy loss** per distance unit in the conduct.

After computing the gradient of $P(x)$, some manipulation and integration that we will not detail here (see [13] for details), the equation eventually becomes:

$$g(P_{out}) - g(P_{in}) - \frac{32\rho\Phi^2}{\pi^2} ln(\frac{D(P_{out})}{D(P_{in})}) + \frac{128\eta\Phi L}{\pi}(1.5 + 0.0035 < Re >) = 0 \tag{10}$$

where:
- Φ is the air flow through the airway.
- g(P) is a primitive of $D(P)^4$, meaning $g(P_{out}) - g(P_{in}) = \int_{in}^{out} D^4(x)dx$.
- $<Re>$ is the average of the Reynolds number in the conduct, estimated at $\frac{4\rho\Phi}{\eta\pi\frac{D_{in}+D_{out}}{2}}$ [13].
- L is the length of the conduct, and η, ρ the air viscosity and density respectively.

In order to minimize the upcoming numerical computation time, we compute the **exact expression of** $g(P)$ (Eq 11&12). With the airway compliance law (1.4), we obtain:

$$g(P) = D_{max}^4 \alpha_0^2 \left(\frac{P_1}{2n_1 - 1}\right)\left(1 - \frac{P_{tm}}{P_1}\right)^{-2n_1+1} \qquad P_{tm} < 0 \tag{11}$$

$$g(P) = D_{max}^4 \left(P_{tm} - \frac{2(1-\alpha_0)P_2}{n_2 - 1}\right)\left(1 - \frac{P_{tm}}{P_2}^{-n_2+1}\right) + (1-\alpha_0)^2 \frac{P_2}{2n_2 - 1}\left(1 - \frac{P_{tm}}{P_2}\right)^{-2n_2+1}) \;\; P_{tm} \geq 0 \tag{12}$$

with the parameters $\alpha_0, P_1, P_2, n_1, n_2$ from Lambert et al [12].

2.2 Problem formalisation

Now that we have identified our variables, being the vectors P and Q, respectively containing the values of the pressures and flow rates at each node and in each conduct of the system, and that we have the relationship between them and the initial conditions, it is time to formalize the problem and consider the system we will be solving.
For a given time, let **X be the variable** of our system, X is a vector containing the values of P and Q that we have to determine. Namely all of them except the first value of P (which is set to 0 Pa) and the terminal values of P, as we know the end transmural pressures to be equal to $P_{alv}(t) - P_{pl}(t)$.
Let **F be the vector of the equations (7), (8) and (10)**. With N_{nd} and N_{vs} the number of nodes and airways in the intra-thoracic model, F is the concatenation of N_{nd} (7) equations on the flow rates, one (8) equation for the extra-thoracic part, and N_{vs} (10) equations for the pressure. F contains only the left part of these equations, and we will now use a method to find the correct value of X (i.e. the values of P and Q) to **minimize the norm of F.**

2.3 Newton-Raphson method

To solve $F(x) = 0$, we use **recursively a Newton-Raphson method**.
As the equations composing F are all differentiable with respect to to the flow rates and pressures, we

can compute the **Jacobian matrix of F, which we will note dF**. Generally speaking, the Newton-Raphson method works by approximating the solution of the equation by the intersection of the tangent of F at X with the origin axis. With a typical scalar function, this process can be represented as follows:

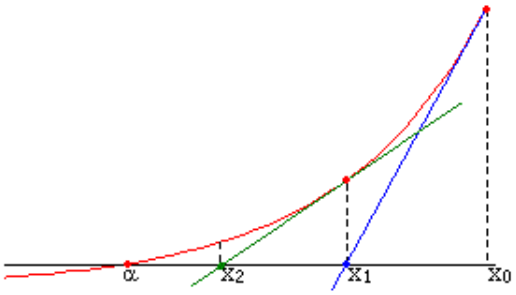


Fig 8, Iterations of the Newton Raphson method for a scalar function.

We implement an algorithm that computes X_{k+1}, the new value of our variable X, knowing its previous value X_k. Step by step (Fig 8), solving the problem with a Newton-Raphson method, we thus reach a point where the equations are all solved satisfyingly enough.
In our case, this means we consider, with X_k the current value of our variable, F_k the values of F on vector X_k, dF_k the values of dF on vector X_k, and $dX_k = X_{k+1} - X_k$:

$$F_{k+1} = 0 = F_k + dF_k(X_{k+1} - X_k)$$
$$0 = F_k + dF_k(X_{k+1} - X_k)$$
$$0 = F_k + dF_k dX_k$$
thus:

$$dX_k = -dF_k^{-1} F_k \tag{13}$$

We can now update X: $X_{k+1} = X_k + dX_k$, with (Eq 13) and repeat the process until the norm (discussion about the norm will be made further) of F_{k+1} goes below a prescribed value ε.

Numerical implementation

3 Algorithm

Now that we have explained the physical and mathematical bases of our model, we can proceed to introduce the algorithm. We will discuss here the main choices that were made for it to function, and **you can find the whole algorithm in the *additional content* section**.

3.1 Airway's Structure

Firstly, we need to **numerically translate the geometry of the lungs**. For this, we set the different physical constants, the compliance parameters and Weibel's parameters for the airway dimensions.
The airway's **structure is represented using 4 matrices**, called *mat-link, mat-node, gen-count and D-mat-gen.* Browsing through them allow us to know **which nodes connect which airways and reciprocally**, and to know the maximum diameter and length of each airway.

These baseline matrices are built using the aforementioned parameters taken from the literature, and depend of the number of generation we want.
As we want the tree to be **non symmetrical**, we use a random distribution to determine for which of the airways issued from a node we use h_{min} or h_{max} to set the diameter according to the the Murray-Hess law and the Weibel's parameter values.

3.2 Physical equations and given time solution

3.2.1 Equations and initialisation

To carry out the given time computation, we built the *refresh* function (cf. algorithm). This function takes as inputs the **geometrical parameters** (the 4 tree structure matrices described in 3.1), the **initial pressure and flow rate values**, **stopping criteria**, the **given time t** and the **duration dt** of its state, among other parameters.
To know the **state of the system, i.e. the transmural pressure and air flow values in each node and airway** respectively, we use **functions returning the left parts of equations (7), (8) and (10)**. For instance, please refer to the function f of the algorithm, which is equal to the left part of equation (10), giving the relationship between the pressures and flow rate inside of an airway.
Our objective is to **find the pressure and air flow values which set the values of these functions to 0**. The functions are **stored in a vectorial function F**. We also compute its **Jacobian**, the matrix **dF** to perform the Newton-Raphson method. The **variables we aim to find** (transmural pressure and air flows) are **stored in a vector X**.
This *refresh* function first **computes the alveolar and pleural pressure**, which drive the resolution. The initialisation is then completed by building the initial vectors F, X and the matrix dF.

3.2.2 Stopping criteria

The goal of the refresh function is to give the pressures and flow values for each point of the tree structure at time t. Thus, we want to have each of the state functions (the coefficients of vector F, which correspond to the left part of equations (7), (8) and (10)) to be as close to 0. The **stopping criteria are**:

- The **norm of F is close enough to 0** (Eq 14), below a prescribed value ε

- The **maximum amount of step** has been reached.

Norm: As we are working on vectors of a finite dimension, all the mathematical norms are equivalent. The norm we chose to use to monitor when to stop the iterative process **is the Euclidean one**. But to ensure that each of the equation has the same effect on the value of the norm, we have **introduced corrective coefficients α to weight the effect of each equation**.
Thus, we have :

$$norm(F) = \sqrt{\sum_i \alpha.F_i^2} \tag{14}$$

where:
-F_i is the i-th coefficient of F
-α is a constant, equal to 10^{-2} if F_i is an air flow equation (a kirchhoff law, equation I), and 10^{-6} if F_i is a pressure equation (a f function, Eq. (10)).

These coefficients were chosen based on the **average magnitude** of the debit and pressure, to ensure that each term $\alpha.F_i^2$ of the sum has approximately the same weight.

Steps: The maximum amount of steps was chosen to be 1000. The reasonable amount depends of how we compute the next step dX in each iteration, but experience has proven that a resolution requires between 10 and 600 steps usually. So the amount 1000 was chosen to be sure to fit every normal situation, even if the next-step computation is particularly slow.

3.2.3 Next step computation

While the number of steps did not reach the maximum allowed and the norm of the function-vector F is above the prescribed value ε, **the following algorithm is followed (Newton-Raphson scheme)** :
$-$dX = - dF^{-1}F (Eq 13) is computed thanks to the values of F and dF at the end of the previous step.
- the X value is updated to X+dX
- the pressure and flow vectors are updated accordingly to the values of the variables in X
- using the equations (7),(8),(10), the new value of X and the structure matrices, the new values of function F and dF are computed.
In this process, we have made some empirical choices that were necessary to limit the computation time as much as possible, and to ensure the model would converge.

dX computation First, we made some tests on how to **compute dX the fastest way possible**. Using the "numpy" extension, optimised for computation, we had two options: compute dF^{-1} and then compute its product with the matrix F, or use the module *np.linalg.solve* that is build in to give us the result of the equation B=AX.
To determine the best solution, we carried out some tests on matrices of different sizes:

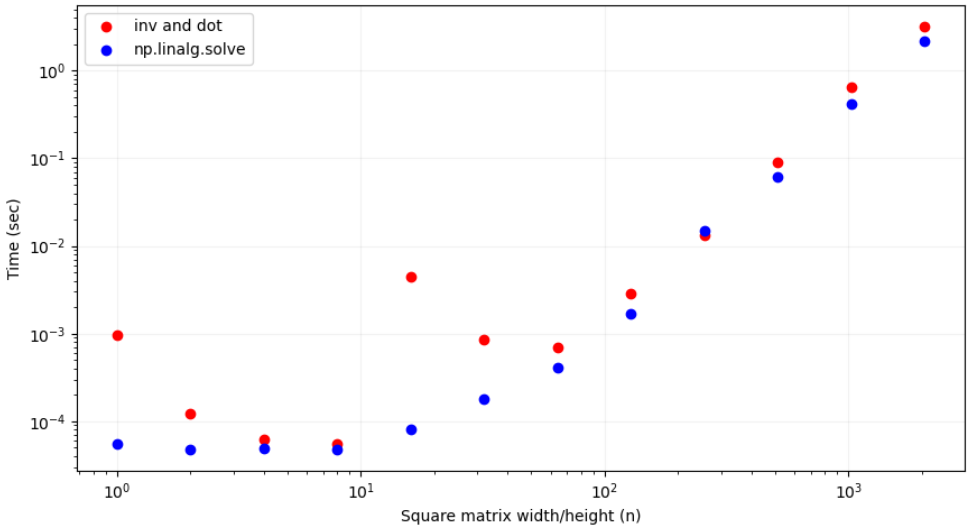


Fig 9, Time to compute dX, in function of F and dF's dimensions.

According to the test results (Fig 9), that show the second method's better performances, **we used the *np.linalg.solve* module.**

F and dF computation Secondly, computation time measurements showed that approximately 75% of the computation time is spent on computing the values of the function f (coefficients of F) and its derivative (corresponding to the equation (10)). This was the first reason for us to **compute the exact expression** of the primitive of the function $g(P) = D(P)^4$ (see function g, Eq (11)), as the python built-in modules displayed slow performances.
This issue also motivated us to compute the values of F and dF less frequently. After some tests, we empirically found out we could avoid compute dF for **1 out of 3 iterations** with the same results. Raising further the amount of iteration where we did not compute dF caused massive divergence.

Step computation Finally, we decided to **adapt the norm of the step dX** (Eq 15). Two effects can be observed:
- if the norm of dX is too high, the model diverges, which was our main problem early on in the development,
- if the norm is too small, the model takes massive amounts of time to eventually converge.
Thus, we chose an adaptative step size. Our method is inspired by the work of Duchi et al. [15] on gradient descent algorithms. **The expression we chose and empirically adjusted is**:

$$step_k = \frac{dX_k}{\ln(1 + S_k^{\beta})} \tag{15}$$

where $S_k = (\sum_{i=0}^{k} dX_i)$, and β is an empirical parameter, set between 1 and 3 depending on the size and geometrical parameters of the tree.

3.3 Quasi-static solution

With the given-time solver (*refresh* function of the algorith) correctly working, we can now simulate a full expiration. The process is much simpler, as we only have to repeatedly implement the given-time solver. As you can see in the additionnal content section on the full algorithm, we thus have created an *expiration* function. **Given the geometrical parameters** (the tree structure matrices), a **starting time** value, an **initial lung volume** and a **time step dt**, this function solves the given-time problem for each time value. Each new iteration uses as initial conditions the pressure and flow rate values found by the previous iteration, and the driving pressures P_{alv} and P_{pl} evolve with the time value according to the equations (2), (3) and (4). The pressures and flow rates are stored for data visualisation, and the air volume remaining in the lungs is updated in function of the volume dV of air expired during a single iteration.

The **stopping criteria** are:
- the lung is **empty** (lung volumes reached minimum volume),
- the **expired volume** dV during the last given-time iteration is **critically low**, below a prescribed value (meaning the lung is almost empty, and emptying too slowly).

The main **difficulty is the choice of the time step**. A **large time step** allows for a **faster total computation time**, but depending on the number of generations and the geometric parameter of the tree, it can also **lead to divergence** and thus make the whole expiration simulation fail.
Loads of test and empirical trial and error led us to believe a time step of 0.01 to 0.05 seconds is the best compromise. This time step is adaptive, and decreases if the model struggles to converge (when the flow rate peaks for instance). This **process allows the simulation to overcome singular time values where it is harder to converge**.

Another issue was faced after few trials. Due to computation time, our model could not handle the whole 15 generations of the tracheobronchial region. We therefore had to limit our simulation to tracheobronchial tree containing at most 10 generations. However, such models do not account properly for the overall airway resistance of the lung, leading to a too large flow rate at the trachea.
Hence, we decided to **compute the equivalent resistance for a Weibel's tree** [3], and artificially introduced a linear equation after the last generation, equivalent to the missing generations we could not simulate. This addition to the given-time simulation slightly improved our results, which we will discuss in the next section.

4 Simulating the forced expiratory manoeuvre

Now that the model gives us the state of the lungs at any time during the expiration, we can visualise the different variables and compare our results to previous studies and experimental measures.

4.1 Driving pressures

4.1.1 Results

As explained in part 1, the pleural and alveolar pressure drive our model. Here are these pressure (Fig 10), for a 8-generation simulation.

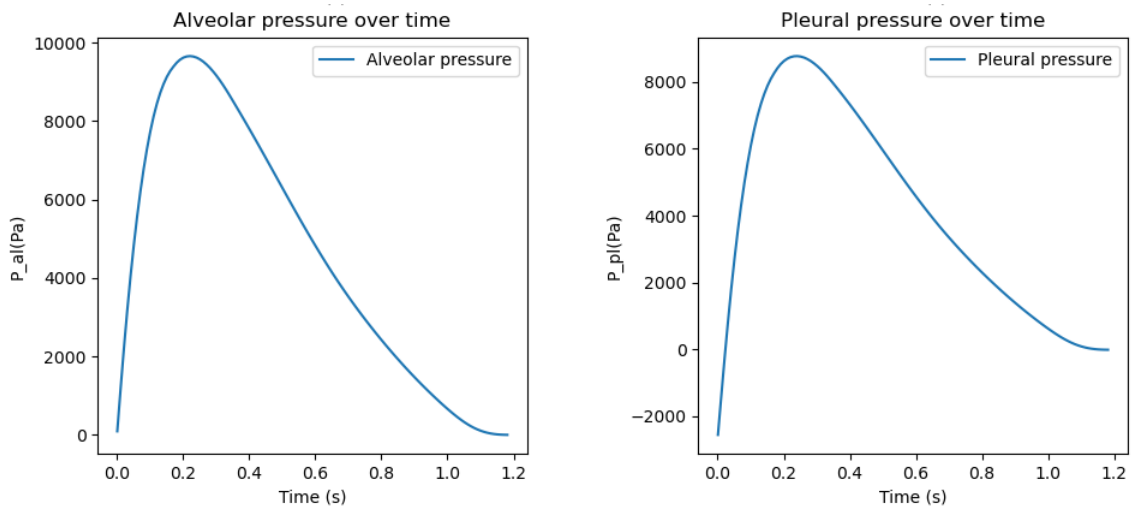


Fig 10, Pleural and alveolar pressures over time for a 8-generation simulation.

4.1.2 Discussion

The shape of the pleural and alveolar pressures are exactly what we expected from Polak's model. The system was driven as intended, which is comforting for the rest of the study. The main issue, that we will observe in all the temporal plots, is that the expiration ends to rapidly in trees with less than 15 generations. Here, for 8 generations, the lungs are emptied out in 1.2 seconds. For a 12-generation simulation, the shape is exactly the same but the expiration lasts 2 seconds, which is better but still falls a bit short of the experimentally expected values. As the difference between the 8 and 12 generation simulations show, we believe this issue of the lungs emptying too quickly will be solved by a full 15-generation simulation. As the whole tree is not properly modelled, the total air flow resistance is not as high as it should be, which allows the flow to be higher on average and thus the lungs to empty sooner. If

this problem is partially corrected by the additional resistance mentioned earlier, this resistance is linear and does not account for the compliance of the missing airways.

4.2 Flow-volume loops

4.2.1 Results

The simulations gave us realistic profiles (Fig 11&13) for the flow vs time and lung volume vs time curves. Here are these curves, that have the shape we would expect from experimental measures (Fig 12&14)(except for the expiration time of 1.2 sec, that we discussed in 4.1.2).
Here is a comparison of our result (left) with some experimental data from the literature (right).

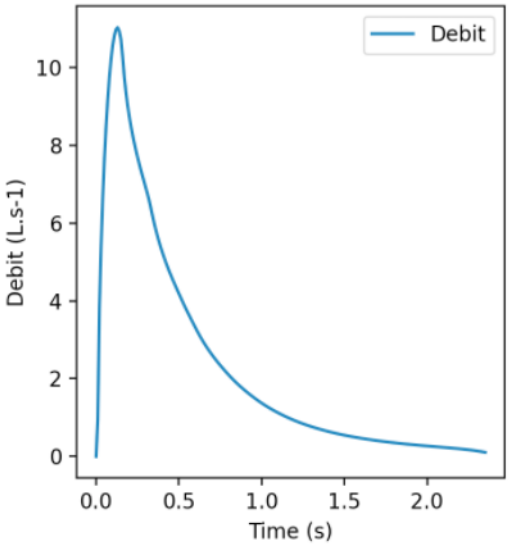


Fig 11, Flow rate vs time for 8 generations.

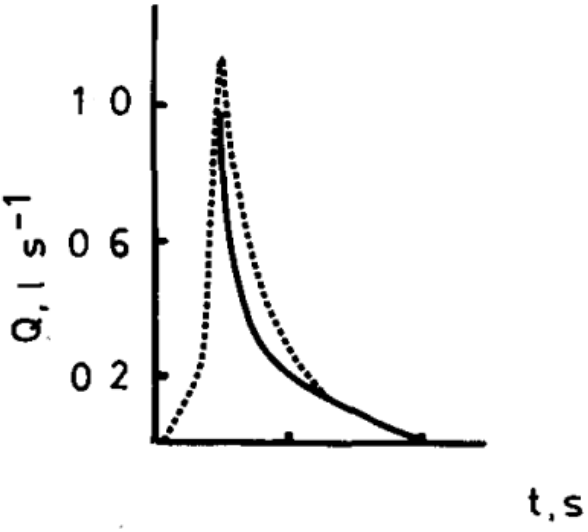


Fig 12, Experimental flow rate profile from Thiriet et al. [17]

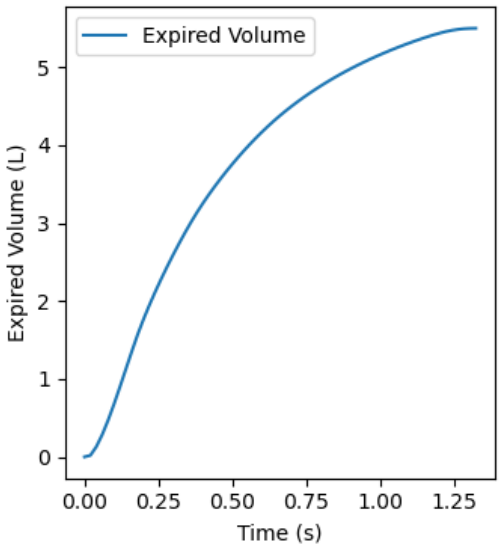


Fig 13, Expired volume vs time for 8 generation.

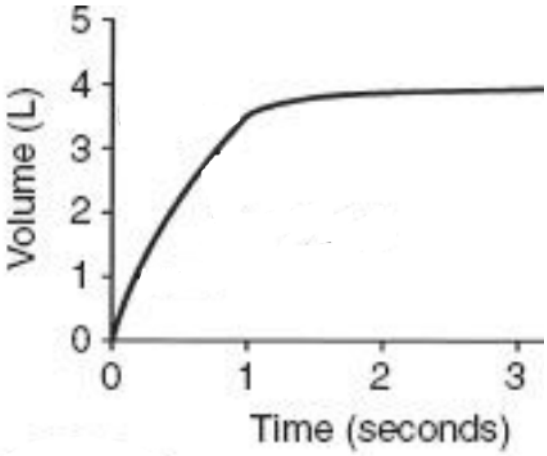


Fig 14, Expired volume vs time, experimental data from thoracickey.com.

The obtained curves for the flow and expired volume temporal values have the expected shape and

present the characteristics of the experimental ones. The air flow curve possesses the same initial linear increase, maximum value (10L/s reached at about 0.5 sec), and homographic decrease. The expired volume also displays the initial tangent and saturation effect that we find on the experimental curve. Only the total lung volume (and thus the asymptotic value of the expired volume) differs slightly.

We will now examine the flow-volume loops, i.e. the flow rate plotted against the lung volume. These curves are of high relevance since they are the one to be used by practitioners to diagnose patients with respiratory diseases. Here are the results (Fig 15&16) we obtained for different amounts of tree depth. We ran the algorithm for up to 12 generations, which took over 2 days to fully converge.

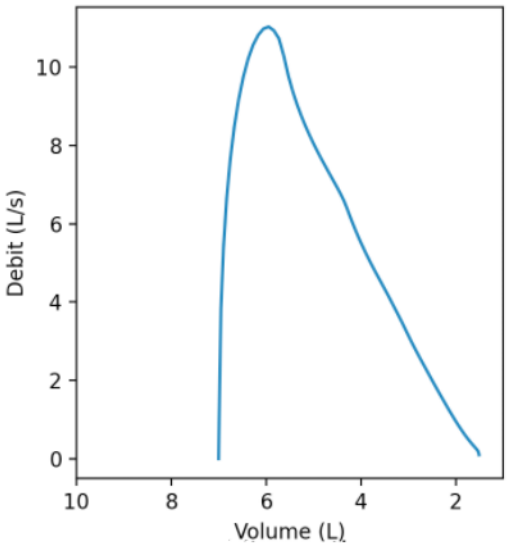


Fig 15, Flow-Volume loop for 12 generations.

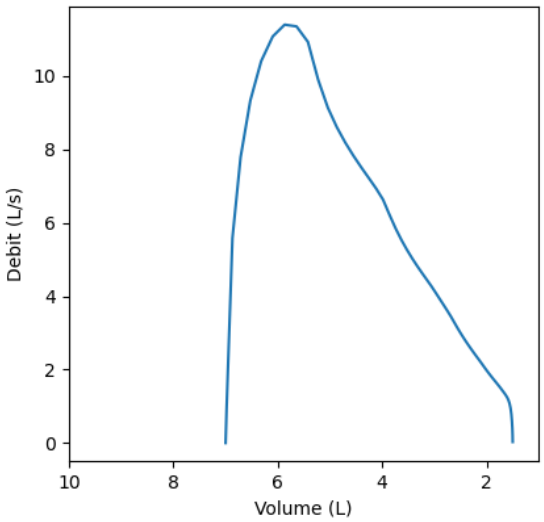


Fig 16, Flow-Volume loop for 8 generations.

4.2.2 Discussion

The flow-volume loops are shaped as expected (Fig 17). In comparison, here are the results from Polak [9], who also used the Weibel's parameters [6] (Fig. 9). We indeed find the same fast increase and slower linear decrease. The main difference is that the maximum simulated air-flow is a bit higher.

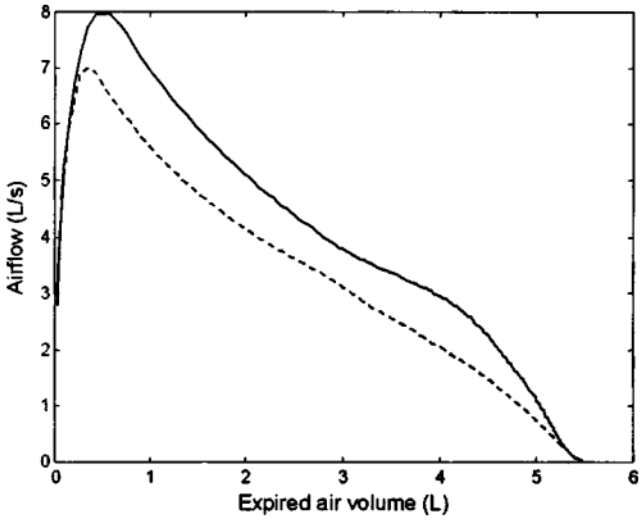


Fig 17, Flow volume spirometry curve from Polak, 2003 [9].

If this curve is more dilated on the abscissa than ours, we observe the same shapes and behaviour. It is worth noticing that if **Polak's abscissa is the "expired air volume", ours was the total remainng lung air volume**, which decreases during the expiratory manoeuvre.
The **shape of these curves can be explained by the airways' behaviour**. The **first part** of the expiration process is the rapid rise of the flow rate: at the beginning of the expiration, the air flowing inside the airways produces a rise of the pressure, which **keeps the airways open**. The relative diameters of the airways are close to 1. Hence, the **airway resistance is essentially constant and the flow rate defined by the pressure difference between both ends of the airways quickly rises**. But as the flow rises in the proximal bronchus, the Reynolds number grows. Thus, the more turbulent flow regime induces a **transmural pressure drop**, which can reach negative values (Fig 18&19). This corresponds to the **air flow peak**. This pressure drop, when it reaches the null transmural pressure, induces a reduction of the airways lumen, and almost closes the airway. And as this perturbation progresses towards the distal bronchus, the **null transmural pressure point travels inside the system towards the higher generations** (Fig 20). During this process, the **flow rate is limited in the constricted airways**, which explains the **air flow's decrease** and the **linear decrease** of the flow relatively to the volume.
This reasoning will be easier to understand with the next section, about the airway's behaviour.

4.3 Airways behaviour

4.3.1 Results

As explained in the previous section, the airway behaviour is what explains the mechanics of the respiratory system. To better visualize this behaviour, we plot the pressure in the main airways for each generation, and the relative diameter of these airways. To clarify the graph, we only plot this data for the 8 first generations (Fig 18).

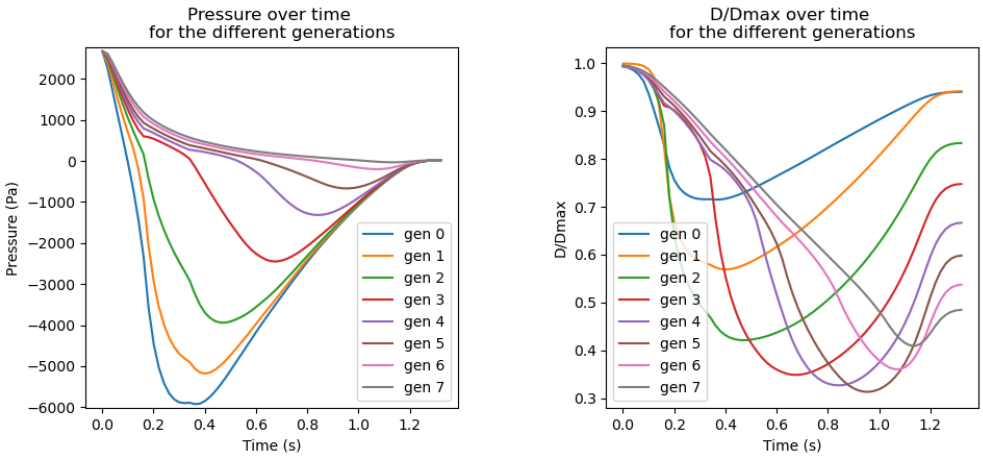
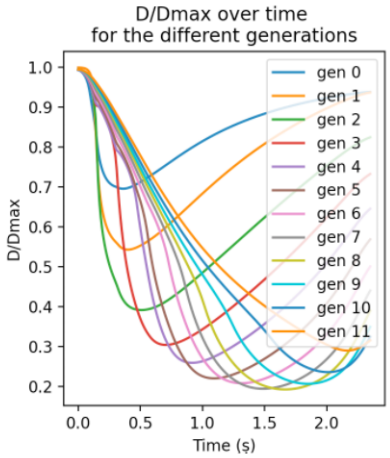


Fig 18, Pressure and relative diameters of the airways for a 8 generation simulation.

The behaviour of the more distal airway (generation > 8) follows the same pattern, as we can see with one example of a 12-generation simulation (Fig 19).

Fig 19, Pressure and relative diameters of the airways for a 12-generation simulation.



As evoked in 4.2.2, the **overall air flow is limited by the constriction of the airways**, which is critical when an airway reaches a null transmural pressure. Thus, it is interesting to know where exactly is this critial null transmural pressure point (Eq 16). We tried to find it in for the main airways. For this, we use the equation (10) of part 2.1. But instead of integrating this equation between the proximal and the distal end of the airway, we **integrate it between the proximal part and the relative abscissa x where is supposedly located the null pressure point**. This gives us the following equation:

$$g(0) - g(P_{in}) - \frac{32\rho\Phi^2}{\pi^2} \ln\left(\frac{D(0)}{D(P_{in})}\right) + \frac{128\eta\Phi x}{\pi}(1.5 + 0.0035 < Re >) = 0 \tag{16}$$

with x the abscissa of the null transmural pressure point inside the airway. We can now find the relative abscissa of this point:

$$x/L = \pi \frac{-g(0) + g(P_{in}) + \frac{32\rho\Phi^2}{\pi^2} \ln(\frac{D(0)}{D(P_{in})})}{128L\eta\Phi(1.5 + 0.0035 < Re >)} \tag{17}$$

When the relative abscissa x/L (Eq 17) reaches 1, it means the null transmural pressure point reached the next generation. Knowing the generation of the airway and the relative abscissa inside the airway, we can plot the **global relative abscissa** (n_{gen}+x/L) of this critical point inside the system (Fig 20).

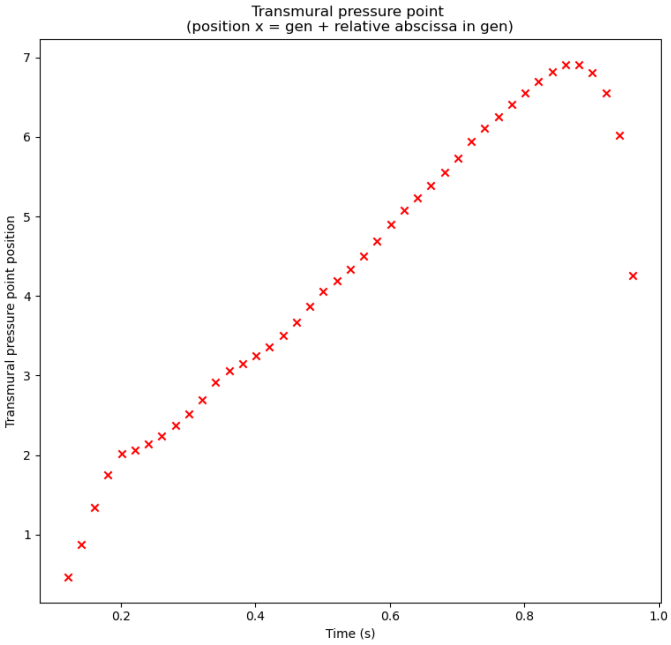


Fig 20, Global relative abscissa of the null transmural pressure point for a 8-generation simulation.

In the previous graph, we can observe the global relative abscissa versus the time. For instance, when the abscissa is equal to 3.5, it means the point is in the 4th generation, at 50% of the 4th generation airway's length.

4.3.2 Discussion

We can know clearly understand the shape of the flow-volume loop and flow rate temporal curve (c.f. Fig 11,13,15,16). The flow limitations can be observed as soon as the pressure drop, around 0.2 second for the 8-generation simulation. At this point, as the pressure drops in the first generation, the diameter difference between the proximal and the distal ends of the trachea is maximum, and the flow rate is therefore limited by this very bronchus. As the diameters of the following generations successively close, **this point of limitation moves towards the more distal airways**, as we can witness with the position of the null transmural pressure point.
This simulation gives us a **prime insight on the behaviour of the airways**, and can help us better understand the relationship between the airway's state and the evolution of the flows rate and lung volume of an individual. This visualisation technique could be specially useful when considering cases of respiratory diseases, which we will discuss in the next part.

Pathology simulation and detection

Pulmonary diseases are a leading cause of death. Modelling them is a way to **understand the way they work and put in place efficient treatments**. They may affect the airways, lung tissue or blood circulation in the lungs.
We will now model **two widespread life-threatening conditions**, **COPD** (for chronic obstructive pulmonary disease), an obstructive disease (which reduces the air flow), and **lung fibrosis** which is a restrictive one (which reduces the overall volume).

5 Pathology characterisation and modelling

5.1 COPD

COPD affects the tracheobronchial tree by **permanently obstructing the smaller airways**, which causes an overall **higher flow resistance** and thus difficulties to breath in and out. The airways' diameters reduction can have three main origins:
- partial destruction of the lung parenchyma (the functional tissue where the gas exchange take place), which reduces the airway internal diameter and makes them more compliant.
- thickened bronchial airway tissue due to a respiratory inflammation.
- excessive pulmonary secretions, which partially fills some of the airways.

This has consequences on the respiratory system's behaviour. We will try to model theses changes in our model, by tuning our model's parameters. We will change the compliance and lung volume parameters according to West [18], the lung tissue resistance according to Verbeken et al. [19] and the null pulmonary compliance C_0 according to Greaves et al. [20].

5.2 Lung Fibrosis

Lung fibrosis is characterised by an excess of fibrous deposit around the smaller airways, and a thickened bronchial wall due to inflammation. These symptoms **reduce the smaller airways' compliance and their diameter**. But contrary to the COPD, the effect on the compliant behaviour is a **more rigid** (but smaller nonetheless) airway instead of a more easily closed one. Thanks to Tiddens *et al* .[21], we will tune our compliance parameters. As shown in Sansores *et al.* [22], all the lung's volumes (total, minimum...) are drastically reduced (as much as 50%), and the overall lung tissue resistance is also strongly higher according to Bachofen *et al.* [23].

6 Modelling results and discussion

To model the two diseases, we took from the literature **new lung parameters, for both a mild and a severe case**. To model various disease severities, we assumed a linear variation of these parameters between their values in the mild and severe states.
We then ran an expiration simulation for a 8-generation tree, for 3 different states of the diseases. The most relevant graph to plot is the **flow-volume spirometry curve, as it is the one observed by practitioners**. But it is also interesting to visualise the global resistance of the system, and the temporal evolutions of the air flows and volume, to **compare our modelling of the disease to its real symptoms**. Here are the simulated curves (Fig. 21).

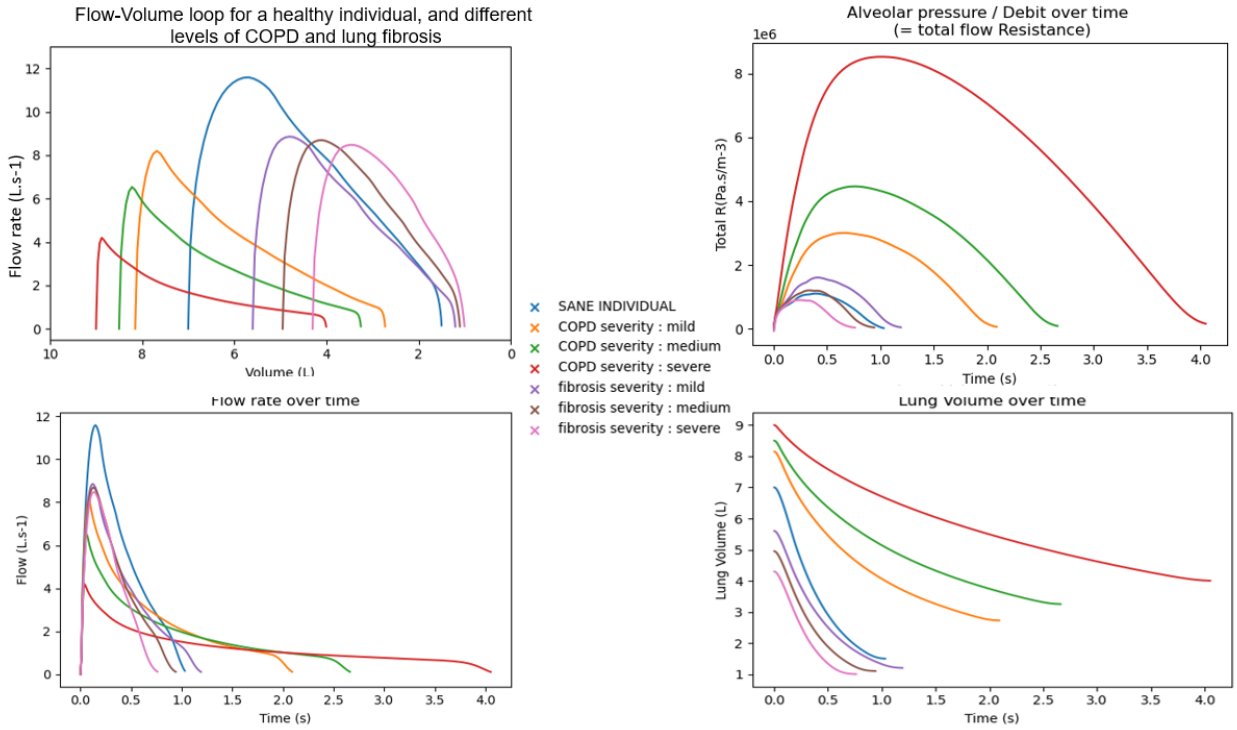


Fig 21, Simulation of the forced expiration manoeuvre for a 8-generation tree.

First, the shape of the flow-volume loop tells us that the algorithm correctly models the diseases. Indeed, the simulated curves **present the characteristics of the experimental measures**: higher volumes and drastically lower flow rates which decreases faster for COPD, and lower volumes and less linear flow rate decrease for lung fibrosis. These results are comforting for the use that can be made of the algorithm.
Moreover, the effects on the flow rate vs. time and lung volume vs. time curves correctly depicts what is expected for the simulated diseases. Both show a reduction of the flow rate peak, while the lung volumes are higher for COPD and lower for lung fibrosis, which is indeed characteristic of these diseases.
Finally, the total flow resistance also show consequences of COPD and lung fibrosis. COPD does make the airways more flexible, meaning their diameters shrink more easily which translates to lower flow rate values and thus a much higher overall flow resistance of the system. On the other hand, the lung fibrosis does shrink the maximum diameter of the airways, but also makes them more rigid: the air flow value is lower on average, but more stable. Hence the overall resistances being close to the healthy one.

To complete our study of the diseases' effects on the lungs, we track the position of the null transmural pressure point inside the lung's major airways (Fig. 22).

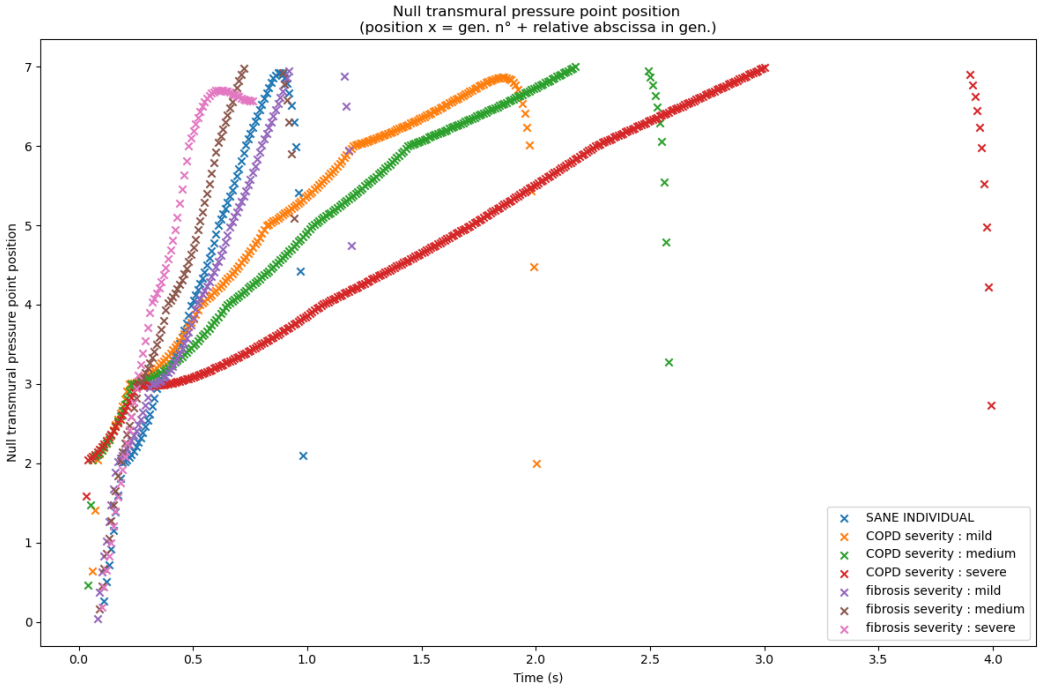


Fig 22, Position of the null transmural pressure point in the 8-generation tree.

This last graph tells us how the pathologies affect the behaviour of the air flow limitation. As expected, the more rigid airways of the lung fibrosis make the limitation point travel faster to the distal airways as the lung are emptying faster. On the opposite, due to more constriced airways, the limitation point travels slower in the COPD case.

Conclusion and opening

The developed model **meets the goals of simulating a patient-specific flow-volume loop**, and **providing information about the state of the lung's airway system at every moment of the forced expiratory manoeuvre**. Tweaking the geometrical and physiological parameters also allows to **correctly simulate different pathologies**.
If the relative simplicity of the model translates to reasonable computing capacity requirements and computing time, the python implementation is a limit. **Translating the algorithm in C++ is the next step to enhance the overall performances**, and make a real tool out of this prototype.
Later on, an interesting lead to follow would be **the use of the flow-volume loops for machine learning training data set**. Simulating these curves using different geometrical and physiological parameters, for different types of conditions, would quickly provide a lot of standardized and labelled data. As data labelling and standardizing represents about 80% of the work when it comes to supervised machine learning, **using our algorithm to develop such a method could be an efficient way to obtain a respiratory disease detecting AI.**

References

[1] WHO “The top 10 causes of death” Google Scholar, 2015.
[2] Irena Maniecka-Bryła, Paulina Paciej-Gołębiowska, Elżbieta Dziankowska-Zaborszczyk, Marek Bryła. "Advances in Clinical and Experimental Medicine" *Lost life years due to premature mortality caused by diseases of the respiratory system*, 2018.
[3] Adeloye D, Chua S, Lee C, et al. Global and regional estimates of COPD prevalence: systematic review and meta-analysis. J Glob Health, 2015.
[4] Mouronte-Roibas C, Leiro-Fernandez V, Ruano-Ravina A, et al. Chronic obstructive pulmonary disease in lung Cancer patients: prevalence, Underdiagnosis, and clinical characterization. Respiration, 2018.
[5] Agujetas, R., Barrio-Perotti, R., Ferrera, C., Pandal-Blanco, A., Walters, D.K., Fernández-Tena, A.d. Construction of a hybrid lung model by combining a real geometry of the upper airways and an idealized geometry of the lower airways, 2020.
[6] Weibel and al. Morphometry of the Human Lung, 1963.
[7] Florens, Sapoval, Filoche. An anatomical and functional model of the human tracheobronchial tree, 2011.
[8] Adam. G. Polak. A forward model for maximum expiration, "Computers in Biology and Medicine", 1998.
[9] Adam. G. Polak, Kenneth. R. Lutchen Computational Model for Forced Expiration from Asymmetric Normal Lungs, "Annals of biomedical engineering", 2003.
[10] H.Guénard, Physiologie humaine, 2001.
[11] Lambert R. K., Wilson T. A., Hyatt R. E. and Rodarte J. R.*Journal of Applied Physiology*, 1982.
[12] J. R. Wheatley, T. C. Amis, and L. A. Engel. Nasal and oral airway pressure-flow relationships. *Journal of Applied Physiology*, 1991.
[13] M.Filoche, M.Florens. The stationary flow in a heterogeneous compliant vessel network, *J. Phys.: Conf. Ser. 319*, 2011
[14] H. A. W. M. Tiddens, S. H. Donaldson, M. Rosenfeld, and P. D. Pare. Cystic fibrosis lung disease starts in the small airways : can we treat it more effectively ? *Pediatric Pulmonology*, 2010.
[15] J.Duchi, E.Hazan, Y.Singer. Adaptive Subgradient Methods for Online Learning and Stochastic optimisation, 2011.
[16] W.Hofmann, Modelling inhaled particle deposition in the human lung-a review, 2011.
[17] M. Thiriet, M. Bonis, A.S. Adedjouma, C. Hatzfeld, J.P. Yvon. Experimental and theoretical models of flow during forced expiration: pressure and pressure history dependence of flow rate, 1987.
[18] J. B. West, Pulmonary Pathophysiology, The Essentials, Seventh Edition, 2008.
[19]E. K. Verbeken, M. Cauberghs, I. Mertens, J. M. Lauweryns, and K. P. Van de Woestijne. Tissue and airway impedance of excised normal, senile, and emphysematous lungs, 1992.
[20] I. A. Greaves and H. J. H. Colebatch, Elastic behaviour and structure of normal and emphysematous lungs post mortem, 1980.
[21] H. A. W. M. Tiddens, S. H. Donaldson, M. Rosenfeld, and P. D. Pare. Cystic fibrosis lung disease starts in the small airways : can we treat it more effectively? 2010.
[22] R. H. Sansores, A. Ramirez-Venegas ans R. Perez-Padilla, M. Montano, C. Ramos, C. Becceril, M. Gaxiola, P. Paré, and M. Selman. Correlation between pulmonary fibrosis and the lung pressure-volume curve 1996.
[23] H. Bachofen and M. Scherrer. Lung tissue resistance in difuse interstitial pulmonary fibrosis. Journal of Clinical Inverstigation, 1967.

Additional content

Algorithm :

```
import math as m
import numpy as np
from numba import vectorize

###########UNITES###########
#pression : Pa
#volume : m^3
#debit : m^3/s
#longueur : m
#temps : s
############################

###########VARIABLES########

####SANE INDIVIDUAL DATA####

#R sistance des tissus respiratoires (Pa.m^-3.s)
R=0.028*1e+06
#R sistance    l' coulement  de l'air dans les voies sup rieures
R_ext = 1.2e7   #Pa.(m^3.s^-1)^-1.68
r = 1.68
#Choix de la pression motrice initiale
Pm=24000
#Choix de la constante de temps de relaxation des muscles expiratoires
tau=0.2
#Volume r siduel (volume d'air min dans les poumons)
RV=1.5e-3
#Capacit  vitale (Volume d'air max dans les poumons)
CV=5.5e-3
# Diam tre de la trach e et dimension fractale
Dmax_0 = 2e-2
h_list = [[1,0.87,0.80,0.83,0.86]+[0.85 for k in range(15)],[1,0.69,0.67,0.67,0.74]+[0.67
                                        for k in range(15)]]
L_list = [3,3.07,1.75,1.43,1.85]+[3 for k in range(15)]
# Compliance pulmonaire    P de r traction  lastique  nulle:
C_0 = 5.7e-6 #m^3.Pa-1
# Volumes pulmonaires min et max, =    106 et 20% de la CPT (Capacit  Pulmonaire Totale
                                        = CV+RV)
V_max = 7.5e-3 #m^3
V_min = 1.4e-3 #m^3

Compliance_param  = [ [0,0.882,0.0108e-2,1,10],
                      [1,0.882,0.0294e-2,1,10],
                      [2,0.686,0.050e-2,1,10],
                      [3,0.546,0.078e-2,1,10],
                      [4,0.428,0.098e-2,1,10],
                      [5,0.337,0.123e-2,1,10],
                      [6,0.265,0.139e-2,1,10],
                      [7,0.208,0.156e-2,1,10],
                      [8,0.164,0.171e-2,1,10],
                      [9,0.129,0.180e-2,1,10],
                      [10,0.102,0.190e-2,1,10],
                      [11,0.080,0.202e-2,1,9],
                      [12,0.063,0.214e-2,1,8],
                      [13,0.049,0.221e-2,1,8],
                      [14,0.039,0.228e-2,1,8],
                      [15,0.031,0.234e-2,1,7] ]

####SEVERE COPD INDIVIDUAL DATA####

# Diam tre de la trach e et dimension fractale
Dmax_0 = 2e-2
h_list = [1]+[0.68,0.7,0.75,0.8]+[0.85 for k in range(20)] #+[2**(-1/3) for k in range(20
```

```
                                                )] +
h_th = 2**(-1/3)

#R sistance des tissus respiratoires (Pa.m^-3.s)
R=0.050*1e+06
#R sistance    l' coulement   de l'air dans les voies sup rieures
R_ext = 1.2e7   #Pa.(m^3.s^-1)^-1.68
r = 1.68
#Choix de la pression motrice initiale
Pm=24000
#Choix de la constante de temps de relaxation des muscles expiratoires
tau=0.2
#Volume r siduel (volume d'air min dans les poumons)
RV=4e-3
#Capacit  vitale (Volume d'air max dans les poumons)
CV=5e-3

# Compliance pulmonaire    P de r traction  lastique  nulle:
C_0 = 11.4e-6 #m^3.Pa-1
# Volumes pulmonaires min et max. CPT(Capacit  Pulmonaire Totale  = CV+RV)
V_max = 9.5e-3 #m^3
V_min = 3.5e-3 #m^3

Compliance_param  = [ [0,0.56,0.01e-2,1,10],
                      [1,0.44,0.02e-2,1,10],
                      [2,0.32,0.03e-2,1,10],
                      [3,0.20,0.036e-2,1,10],
                      [4,0.15,0.036e-2,1,10],
                      [5,0.13,0.034e-2,1,10],
                      [6,0.10,0.030e-2,1,10],
                      [7,0.07,0.031e-2,1,10],
                      [8,0.042,0.029e-2,1,10],
                      [9,0.038,0.027e-2,1,10],
                      [10,0.032,0.024e-2,1,10],
                      [11,0.028,0.023e-2,1,9],
                      [12,0.023,0.023e-2,1,8],
                      [13,0.018,0.024e-2,1,8],
                      [14,0.015,0.025e-2,1,8],
                      [15,0.010,0.026e-2,1,7] ]



##############################################################################
eta=1.8e-05          #Viscosit  de l'air (Pa.s)
rho=1.14             #Masse volumique de l'air (kg/m^3)
P_atm = 0.0# 1013e2       #Atmospheric pressure
##############################################################################

import time  #Lets create a class to mesure our computation time
class Timer(object):
    def start(self):
        if hasattr(self, 'interval'):
            del self.interval
        self.start_time = time.time()

    def stop(self):
        if hasattr(self, 'start_time'):
            self.interval = time.time() - self.start_time
            del self.start_time # Force timer reinit


def mat_links(n):
    '''A matrix that gives the ending node (lower gen) and starting node (higher gen) of
                                                a dichotomous and
    symmetric tree of n generations.

    General form of the Matrix : (for any, non dichotomous or non symmetric system)
```

```
    Each line's index is the index of the link, and each line gives the indexes of the
                                                    two related nodes
    '''

    M = [['atm',0],[0,1]]
    for i in range(2,2**n):
        if i%2 == 0 :
            M.append([i//2,i])
        else :
            M.append([(i-1)//2,i])
    return M

def mat_nodes(n):
    '''A matrix that gives the links connected to a node in a dichotomous and symmetric
                                                    tree for n gen

    General form of the Matrix : (for any non dich. or non sym. system)
    Each line's index is the index of the node. Gives the list of the index of the
                                                    connected links.
    '''

    M = [[0,1]]
    for i in range(1,2**(n-1)):
        M.append([i,2*i,2*i+1])
    return M

################################################################################


def gen_count(mat_link, mat_node):
    '''builds a vector where the index is the same as the links and the value is the
                                                    generation of the link
    i.e. : the i-th value is the generation of the link n  i
    '''
    N = len(mat_link)
    gen = [-1]+[0 for k in range(N-1)]
    for i in range(2,N): #the first two links are the one depicting the trachea and the
                                                    first, only branch, we set them at gen =
                                                     -1 and gen = 0
        j = i
        c = 0
        while j>1: #we stop if we reach the gen 0, i.e. the link n   1
            a = mat_link[j][0] #a = index of node of lower gen than j
            j = min(mat_node[a]) #we refresh j by taking the index of the link that leads
                                                            to j (of lower gen)
            c+=1 #we count how many gen we go up
            if c >= 40:
                break
        gen[i] = c

    return(gen)


def D_mat_gen(mat_link,mat_node,random='stable',obst=[0,1]):
    '''builds a vector where the index is the same as the links and the value is the max
                                                    diameter of the link
    i.e. :  the i-th value is the max diameter of the link n  i
    obstruction : multiplies all the diameters after the obstruction[0]e gen by a ration
                                                    = to obstruction[1]

    '''

    obstruction = obst

    gen = gen_count(mat_link,mat_node)
    N = len(mat_link)
    if random == 'stable':
```

```
        R=randomvect #copy past a fixed random vect to keep the same parameters
    if random == 'random':
        R=[np.random.random_sample()for k in range(1000)]
    D_mat = [Dmax_0,Dmax_0] + [0 for k in range(N-2)]
    i=0

    for node in mat_node[1:]:
        g=gen[node[-1]]
        h_m,h_M = h_list[0][g],h_list[1][g]
        r=R[i%1000]
        if r<0.5:
            D_mat[node[1]]=D_mat[node[0]]*h_m
            D_mat[node[2]]=D_mat[node[0]]*h_M
        if r>=0.5:
            D_mat[node[1]]=D_mat[node[0]]*h_M
            D_mat[node[2]]=D_mat[node[0]]*h_m
        i+=1

    for idx,D in enumerate(D_mat): #Simple obstruction modelling
        if gen[idx]>=obstruction[0]:
            D_mat[idx] = D*obstruction[1]

    return D_mat



def P_alv(t,VL_t,Phi):
    """Computes the initial pleural pression for a last-gen node (to determine Initial
                                                    Conditions)
    """
    P_al = Pm*(1-np.exp(-t/tau))*((VL_t-RV)/CV)-(R*Phi)
    return P_al

def P_Pl(t,VL_t,Phi):
    '''Computes the pleural pressure, considered uniform for the lung system at time t:
    '''
    Pst = (V_max - V_min)*(1/C_0)*np.log((V_max-V_min)/(V_max-VL_t))
    return P_alv(t,VL_t,Phi) - Pst

def DMAX(g):
    '''Gives the max diameter for gen g
    '''
    h=1
    for i in range(g+1):
        h = h*h_list[0][i]
    return(Dmax_0*h)

def D(P,Dmax,g):
    '''Compliance of a branch (link)
    returns the diameters D of the branch in function of the local pressure and
                                                    generation
    '''
    a_0,n_1,n_2 = Compliance_param[g][1],Compliance_param[g][3],Compliance_param[g][4]
    P_1 = (Compliance_param[g][1]*Compliance_param[g][3])/Compliance_param[g][2]
    P_2 = -((1-Compliance_param[g][1])*Compliance_param[g][4])/Compliance_param[g][2]

    if P<0:
        return Dmax*np.sqrt(a_0*((1-P/P_1)**-n_1))
    else :
        return Dmax*np.sqrt(1-((1-a_0)*((1-P/P_2)**-n_2)))

def D4(P,Dmax,g):
    return D(P,Dmax,g)**4

def int_D4(Po,Pi,Dmax,g):
    '''
    Computes the integral of D**4 from Pi to Po at gen g
```

```
    '''
    #Loading compliance parameters
    a_0,n_1,n_2 = Compliance_param[g][1],Compliance_param[g][3],Compliance_param[g][4]
    P_1 = (Compliance_param[g][1]*Compliance_param[g][3])/Compliance_param[g][2]
    P_2 = -((1-Compliance_param[g][1])*Compliance_param[g][4])/Compliance_param[g][2]

    #Computation
    def neg(P):
        return (Dmax**4)*(a_0**2)*(P_1/(2*n_1-1))*((1-P/P_1)**(-2*n_1+1))

    def pos(P):
        return (Dmax**4)*( P - 2*(1-a_0)*P_2*(1/(n_2-1))*((1-P/P_2)**(-n_2+1)) + ((1-a_0)
                                                **2)*P_2*(1/(2*n_2-1))*((1-P/P_2)**(
                                                -2*n_2+1)) )

    if Po<0:
        if Pi<0:
            return neg(Po) - neg(Pi)
        if Pi>=0:
            return neg(Po) - neg(0) + pos(0) - pos(Pi)
    if Po>=0:
        if Pi>=0:
            return pos(Po) - pos(Pi)
        if Pi<0:
            return neg(0) - neg(Pi) + pos(Po) - pos(0)

def dDdP(P,Dmax,g):
    '''Returns the derivative of D on P
    '''
    a_0,n_1,n_2 = Compliance_param[g][1],Compliance_param[g][3],Compliance_param[g][4]
    P_1 = (Compliance_param[g][1]*Compliance_param[g][3])/Compliance_param[g][2]
    P_2 = -((1-Compliance_param[g][1])*Compliance_param[g][4])/Compliance_param[g][2]

    if P<0:
        return Dmax*np.sqrt(a_0)*(n_1/P_1)*((1-(P/P_1))**(-n_1-1))*(1/(2*np.sqrt((1-(P/
                                                P_1))**-n_1)))
    else :
        return -Dmax*(1-a_0)*(n_2/P_2)*((1-(P/P_2))**(-n_2-1))*(1/(2*np.sqrt(1-(1-a_0)*((
                                                1-(P/P_2))**-n_2))))

def f(Po,Pi,Dmax,g,Q,P_plr):
    """Function to solve = 0 for each node to find the local Pi, Po and Q of the link
                                                between
    """
    L = Dmax*L_list[g]
    Re = 4*rho*Q/(eta*np.pi*(D(Po-P_plr,Dmax,g)+D(Pi-P_plr,Dmax,g))/2)
    return np.array( int_D4(Po-P_plr,Pi-P_plr,Dmax,g)-32*((rho*Q*(1/np.pi))**2)*np.log(D(
                                                Po-P_plr,Dmax,g)/D(Pi-P_plr,Dmax,g))+1*(
                                                128*eta*L*Q*(1/np.pi))*(1.5+0.0035*Re) )

def debug_f(Po,Pi,Dmax,g,Q,P_plr):
    """Function to solve = 0 for each node to find the local Pi, Po and Q of the link
                                                between
    """
    L = Dmax*L_list[g]
    Re = 4*rho*Q/(eta*np.pi*(D(Po-P_plr,Dmax,g)+D(Pi-P_plr,Dmax,g))/2)
    a=int_D4(Po-P_plr,Pi-P_plr,Dmax,g)
    b=-32*((rho*Q*(1/np.pi))**2)*np.log(D(Po-P_plr,Dmax,g)/D(Pi-P_plr,Dmax,g))
    c=(128*eta*L*Q*(1/np.pi))*(1.5+0.0035*Re)
    print('int D4 :', a,'\n log funtion : ',b,'\n loss term: ',c)
    print('assert that residue is negligeable in linear case : ',(abs(a)-abs(c))/(abs(a)+
                                                abs(c)))
    return

def dfdQ(Po,Pi,Dmax,g,Q,P_plr):
    '''returns df/dQ
    '''
```

```
    L = Dmax*L_list[g]
    Re = 4*rho*Q/(eta*np.pi*(D(Po-P_plr,Dmax,g)+D(Pi-P_plr,Dmax,g))/2)
    return np.array(  -64*Q*((rho*(1/np.pi))**2)*np.log(D(Po-P_plr,Dmax,g)/D(Pi-P_plr,
                                                Dmax,g))+1*(128*eta*L*(1/np.pi))*(1.5+2*
                                                0.0035*Re) )

def dfdPo(Po,Pi,Dmax,g,Q,P_plr):
    '''returns df/dPo
    '''
    return np.array( D4(Po-P_plr,Dmax,g)-32*((rho*Q*(1/np.pi))**2)*dDdP(Po-P_plr,Dmax,g)/
                                                D(Po-P_plr,Dmax,g) )

def dfdPi(Po,Pi,Dmax,g,Q,P_plr):
    '''returns df/dPi
    '''
    return np.array( -D4(Pi-P_plr,Dmax,g)+32*((rho*Q*(1/np.pi))**2)*dDdP(Pi-P_plr,Dmax,g)
                                                /D(Pi-P_plr,Dmax,g) )


def sat_fun(x,sat):
    x=np.array(x)
    if np.linalg.norm(x)!=0:
        return (x/np.linalg.norm(x))*np.linalg.norm(sat)*np.log(1+ (np.linalg.norm(x)/np.
                                                    linalg.norm(sat)))
    else :
        return 0

def R_lin_eq(g_in,g_max=15):
    'Compute the equivalent resistance of gmax-g_in generations of linear'
    assert g_in <= g_max, 'GENERATION NB ERROR'
    R0 = (128*eta*3*Dmax_0) / (np.pi * Dmax_0**4) #Poiseuille hydraulic resistance
    return (g_max - g_in)*(2**(g_in))*R0




def refresh_system(mat_link,mat_node,P_ini,Q_ini,t=0,DeltaP=100000,stop_crit=1e-4,epsilon
                                            =[1e-2,1e-6],it_lim=1e3,V=CV,DeltaT=0.01,
                                            grad='naive',obst=[0,1]):
    """Computes the state of the system for 1 temporal iteration

    P_ini, Qini = set of value for P and Q

    epsilon : the precision we want on F=0, first digit for the kirshoff equation and
                                                second for the flow one

    it_lim : max amount of iterations

    DeltaP : Linearises the D(P) :  if DeltaP if very high, the behaviour of D(P) is
                                                asymptotical and D(P) = Dmax, dDdP(P) =
                                                0.
    If it is set to 0, we have full non linear behaviour. Lim of divergence aroud P_Pl

    V = Initial lung volume at the beginning of the iteration

    Delta_t = duration of the iteration

    """

    total_time = Timer()
    total_time.start()

    print('---\nSTART ITERATION\n---')

    mat_D = D_mat_gen(mat_link,mat_node,obst=obst)
    Phi = Q_ini[0]
```

```
n = len(mat_link) #N   of nodes and intersections
P = P_ini.copy() #will contain the pressures at time t
Q = Q_ini.copy() #will contain the debit at time t
gen = gen_count(mat_link,mat_node)
N_gen = max(gen) #N   of gen


N_f = gen.count(max(gen)) # N   of end node/link
N_Q = len(Q)-1-N_f #Nb of debit equations
N_Qv = len(Q)-1 #Nb of debit variables
N_P = len(P)-1-N_f #Nb of pressure equation NOT COUNTING THE EXTRA
N_Pv = len(P)-1-N_f #Nb of pressure variables


assert len(P) == n+1+N_f , "wrong amount of Pressures eq"
assert len(Q) == n , "wrong amount of Debit eq"


F=[0 for k in range(N_Qv+N_Pv)]
dF = np.zeros((N_Qv+N_Pv,N_Qv+N_Pv))
X = np.array( Q[1:]+P[1:len(P)-N_f] )


P_pl = P_Pl(t, V, Phi)  - DeltaP



assert len(X) == N_Pv+N_Qv, "Var X len issue"


steps = 0
start = 1


########################################INITIALIZE F AND DF
                                        ##############################################

P = P[1:] #BC the index 0 is the pressure BEFORE the tracheat (above gen 0), and in
                                          the P list, index 0 = P_atm, after the
                                          trachea
F = F[:-N_f] #We cut of the end DeltaP to append them afterward

for i in range(N_Q):  #N of nodes equation

    node = mat_node[i+1] #No node equation to solve for the node 0 between link 0 and
                                              trachea
    for xn in node :
        if xn == min(node):
            F[i]+=Q[xn]
        else :
            F[i]-=Q[xn]

    dF[i][min(node)-1] = 1

    for j in node:
        if j != min(node):
            dF[i][j-1] = -1

for i in range(N_Q,N_Q+N_P): # N of link equation

    if i == N_Q : #Different equation for the link of the upper trachea

        F[i] = P[0] - P_atm - R_ext*abs(Q[1])**r
        dF[i][0] = -r*R_ext*abs(Q[1])**(r-1)
        dF[i][N_P-1] = 1
        dF[i+1][N_P-1] = dfdPo(P[0], P[1], mat_D[0], gen[1], Q[0], P_pl )
```

```
        else :

            link = mat_link[i-N_Q]

            F[i] = f( P[link[0]], P[link[1]], mat_D[link[1]], gen[link[1]], Q[link[1]],
                                                    P_pl )

            dF[i][i-N_Q-1] = dfdQ( P[link[0]], P[link[1]], mat_D[link[1]], gen[link[1]],
                                                    Q[link[1]], P_pl )

            for xl in link :

                if gen[xl] != -1 and gen[xl] != N_gen:

                    if xl == link[0]:

                        dF[i][N_Qv+xl]=dfdPo(P[link[0]], P[link[1]], mat_D[link[1]], gen[
                                                                    link[1]], Q[link[1]]
                                                                    , P_pl )


                    else :

                        dF[i][N_Qv+xl]=dfdPi(P[link[0]], P[link[1]], mat_D[link[1]], gen[
                                                                    link[1]], Q[link[1]]
                                                                    , P_pl )

                if gen[xl] == N_gen :

                    dF[i][N_P+xl-1]=dfdPi(P[link[0]], P[link[1]], mat_D[link[1]], gen[
                                                                link[1]],Q[link[1]],
                                                                P_pl )

                    dF[N_Qv+xl][N_Qv+xl] = -1

                    dF[N_Qv+xl][xl-1] = -R_lin_eq(N_gen)

                    F.append(P[xl+N_f]-P[xl]- R_lin_eq(N_gen)*Q[xl])


#############################################################################

def norm_corr(F,treshold,a=epsilon[0],b=epsilon[1]): #Personnalised norm
    R=0
    for x in F[:treshold[0]]:
        R+=a*x*x
    for x in F[treshold[0]:treshold[1]]:
        R+=b*x*x
    for x in F[treshold[1]:]:
        R+=1*x*x
    return np.sqrt(R)

N=0

total_calc_time = 0
total_matrix_update_time = 0

#############################################################################
#############################################################################
while norm_corr(F,[N_Q,N_Qv+N_Pv-N_f])>=stop_crit or start == 1:

    start = 0
```

```
        #update X with Newton Raphson scheme-------------------

       # dX = np.dot(np.linalg.inv(dF),F)
        timer_calc=Timer()
        timer_calc.start()

        dX = np.linalg.solve(dF,F)

        timer_calc.stop()
        total_calc_time += timer_calc.interval

        N+=np.linalg.norm(dX)
        if grad == 'adapt' :
            X -= dX/np.log(1+N**(1.2))
        if grad == 'naive' :
            X -= dX
        if grad == 'sat':
            dX_sat = [sat_fun(dx,0.0001)for dx in dX[:N_Qv]]+[sat_fun(dx,5)for dx in dX[
                                                              N_Qv:]]
            X-= dX_sat
        steps += 1

        print('Step  N   : \n',steps,'
                                                    ----------------------------------------------------
                                                    ')

        #Update P and Q for next iteration-----------------------
        P = [P_atm + P_pl]+list(X[len(Q[1:]):])+[P_alv(t,V,Phi)for k in range(N_f)]
        Q = [X[0]]+list(X[:len(Q[1:])])
        Q = [q for q in Q]

        P=P[1:]
        F = F[:-N_f]
        #Update F and dF

        matrix_update_time = Timer()
        matrix_update_time.start()

        for i in range(N_Q):  #N of nodes equation

            node = mat_node[i+1] #No node equation to solve for the node 0 between link 0
                                                              and trachea
            for xn in node :
                if xn == min(node):
                    F[i]+=Q[xn]
                else :
                    F[i]-=Q[xn]

            dF[i][min(node)-1] = 1

            for j in node:
                if j != min(node):
                    dF[i][j-1] = -1


        for i in range(N_Q,N_Q+N_P): # N of link equation

            if i == N_Q : #Different equation for the link of the upper trachea

                F[i] = P[0] - P_atm - R_ext*(abs(Q[1])**r)
                dF[i][0] = -r*R_ext*abs(Q[1])**(r-1)
                dF[i][N_P-1] = 1
                dF[i+1][N_P-1] = dfdPo(P[0], P[1], mat_D[0], gen[1], Q[0], P_pl )

            else :
```

```
                link = mat_link[i-N_Q]


                F[i] = f( P[link[0]], P[link[1]], mat_D[link[1]], gen[link[1]], Q[link[1]
                                                        ], P_pl )

                if steps%3!=0:
                    dF[i][i-N_Q-1] = dfdQ( P[link[0]], P[link[1]], mat_D[link[1]], gen[
                                                            link[1]], Q[link[1]],
                                                            P_pl )

                for xl in link :

                    if gen[xl] != -1 and gen[xl] != N_gen:

                        if xl == link[0]:

                            if steps%3!=0:
                                dF[i][N_Qv+xl]=dfdPo(P[link[0]], P[link[1]], mat_D[link[1
                                                                        ]], gen[link
                                                                        [1]], Q[link
                                                                        [1]], P_pl )


                        else :

                            if steps%3!=0:
                                dF[i][N_Qv+xl]=dfdPi(P[link[0]], P[link[1]], mat_D[link[1
                                                                        ]], gen[link
                                                                        [1]], Q[link
                                                                        [1]], P_pl )

                    if gen[xl] == N_gen :

                        dF[i][N_P+xl-1]=dfdPi(P[link[0]], P[link[1]], mat_D[link[1]], gen
                                                                [link[1]], Q[link[1]
                                                                ], P_pl )

                        dF[N_Qv+xl][N_Qv+xl] = -1

                        dF[N_Qv+xl][xl-1] = -R_lin_eq(N_gen)

                        F.append(P[xl+N_f]-P[xl]- R_lin_eq(N_gen)*Q[xl])

        matrix_update_time.stop()
        total_matrix_update_time+=matrix_update_time.interval

        print('\n----------------------------------------------------\n')


        if steps >= it_lim:
            return X,[P_atm]+P,Q,'it_lim_reached',P_pl,Q[0]*DeltaT


    print('Number of total steps : \n',steps)
    print('Pleural Pressure : \n',P_pl)
    print('---\nEND ITERATION\n---')

    total_time.stop()
    print('TIMINGS :')
    print('total exec time:',total_time.interval)
    print('time spent on resolution:',total_calc_time)
    print('total time spent updating matrix:',total_matrix_update_time)
    print('---')

    return X,[P_atm]+P,Q,P_pl,Q[0]*DeltaT
```

```
def expiration(links,nodes,t0=0.001,DeltaT=0.1,DeltaP=100000,V_ini=CV,grad='naive',
                                                stop_crit=1e-4,obst=[0,1]):

    t=t0
    V=V_ini
    c=0
    dV = 1
    T_list = [t]
    D_mat = D_mat_gen(links,nodes)

    gen = gen_count(links,nodes)
    gen_unique = [k for k in range(max(gen)+1)]
    id_gen = [gen.index(k) for k in gen_unique]
    P_over_time = [[] for k in gen_unique]
    D_over_time = [[] for k in gen_unique]
    Q_over_time = [[] for k in gen_unique]


    NL=len(links)
    N_f = gen.count(max(gen))
    P=[P_atm for k in range(NL+1)]+[P_alv(t,V_ini,0)for k in range(N_f)] #At t = 0, we
                                                    consider the lungs full, with no flow
                                                    thus phi = 0
    Q=[0 for k in range(NL)]

    Debits=[]
    Volumes=[]
    lung_vol=[]

    print('EXPIRATION MODELISATION START','\nInitial Lung Volume:',V_ini,'\nMin Lung
                                                    Volume:',V_min,'\nDelta P:',DeltaP)

    while V >= V_min and dV > 0.000001:

        dt = DeltaT
        c+=1
        dt_r_count =0
        iter = refresh_system(links,nodes,P,Q,t,it_lim=1000,DeltaP=DeltaP,stop_crit=
                                                        stop_crit,V=V,DeltaT=dt,grad=grad,
                                                        obst=obst)

        while iter[-2] == 'it_lim_reached' and dt_r_count <2:

            dt = dt/10
            iter = refresh_system(links,nodes,P,Q,t,it_lim=1000,DeltaP=DeltaP,stop_crit=
                                                            stop_crit,V=V,DeltaT=dt,grad=
                                                            grad,obst=obst)
            dt_r_count+=1


        t = t + dt
        T_list.append(t)


        dV = iter[-1] #volume expired during DeltaT

        if dV <= 0 :
            print('ERROR : dV negative :', dV)
            break

        for i,idx in enumerate(id_gen):
            P_over_time[i].append(iter[1][idx]-iter[3])
```

```
            D_over_time[i].append(D(iter[1][idx]-iter[3],D_mat[idx],i)/D_mat[idx])
            Q_over_time[i].append(iter[2][idx])

        V-=dV #Volume remaining in lungs
        print(' Volume after step :',V)
        Expired_Volume = V_ini-V #Volume expired (total)

        lung_vol.append(V) #Lung volume at time t
        Debits.append(iter[2][0]) #Debit during the last iteration
        Volumes.append(Expired_Volume) #Total expired volume at time t

        P_end=P_alv(t,V,iter[2][0])
        alpha=P_end/iter[1][-1]
        P=[alpha*p for p in iter[1]]
        Q=iter[2]

        if c>=10000:
            break

    print('time to empty lungs :',t)

    return Debits, Volumes, lung_vol, P_over_time, D_over_time, T_list, Q_over_time
```