A Theoretical appendix

A.1 Assumptions of the Gaussian linear model

Most deconvolution methods handle the issue using *linear regression techniques*: for a given sample, they proceed by minimising the Euclidean distance between observed values, noted $\boldsymbol{y_i}$, and the predicted values, $\hat{\boldsymbol{y}}_{\boldsymbol{i}}$, enforcing the linear constraint defined in Eq. 1 of the set of covariables. The method, called OLS *ordinary least squares*, is explicitly stated in Eq. 12, with its matricial solution in Eq. 13:

$$\hat{\boldsymbol{p}}_i^{\text{OLS}} \equiv \underset{\boldsymbol{p}_i}{\arg\min} \, ||\hat{\boldsymbol{y}}_i - \boldsymbol{y}_i||^2 = \underset{\boldsymbol{p}_i}{\arg\min} \, ||\boldsymbol{X}\boldsymbol{p}_i - \boldsymbol{y}_i||^2 = \sum_{g=1}^{G} \left(y_{gi} - \sum_{j=1}^{J} x_{gj} p_{ji} \right) \tag{12}$$

$$\hat{\boldsymbol{p}}_i^{\text{OLS}} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y}_i \tag{13}$$

An alternative method consists of deriving a *generative model* that models the variability of the measured observations by drawing them from parametric probability density functions. The set of parameters, $\boldsymbol{p}$, that described the best the observations, is termed the *maximum likelihood estimate* (MLE). Formally, the MLE maximises the likelihood or the log-likelihood of observed data: $\ell(\boldsymbol{p}|\boldsymbol{y}, \boldsymbol{X}) \equiv \mathbb{P}_{\boldsymbol{p}}(\boldsymbol{y}|\boldsymbol{X})$ for our conditional distribution and with $\boldsymbol{p}$ the ratios to maximise.

Under some strong assumptions, enumerated in Theorem A.1, we can corroborate the OLS and the MLE estimate:

THEOREM A.1 (**GAUSS-MARKOV THEOREM**). *If the following assumptions hold,*

1. ***Strong exogeneity**: The cell type-specific expression profiles are not random variables but rather fixed and constant observations, underlying implicitly that cell populations do no interact: $\forall i \in \widetilde{J}, \forall j \in \widetilde{J}, \, i \neq j, \quad \mathbb{C}ov[\boldsymbol{x_{.i}}, \boldsymbol{x_{.j}}] = 0$.*
2. ***Gaussian-Markov noise:** This hypothesis assumes an independent white Gaussian noise, of null mean and variance not depending on the gene (**homoscedasticity**), for the distribution of the error term. Formally, it is likewise to adding a Gaussian distributed error term, $y_{gi} = \sum_{j=1}^{J} x_{gj} p_{ji} + \epsilon_{gi}$, $\epsilon_{gi} \sim \mathcal{N}\left(0, \sigma_{gi}^2\right)$, from which, directly injecting the exogeneity and homoscedasticity properties, we deduce the univariate Gaussian nature of the distribution of each transcript:*
   $y_{gi} \sim \mathcal{N}\left(\sum_{j=1}^{J} x_{gj} p_{ji}, \sigma_i^2\right)$
3. ***Independence:** From the aforementioned Gaussian-Markov and exogeneity assumptions, we readily deduce that the gene expressions of the bulk measures are independent: $\forall j \in \widetilde{G}, \forall k \in \widetilde{G}, j \neq k, \quad \mathbb{C}ov[y_{ji}, y_{ki}] = 0$.*
4. ***Completeness:** We assume no additional latent variable, such as a non-surveyed cell population, that would contribute to the variability of the bulk mixture.*

*then, the corresponding MLE estimate is equal to the OLS estimate, solution of Eq. 12 that can be computed using the **Normal equations**. Additionally, the MLE is unique (only one global maximum of the log-likelihood function) and BLUE (best linear unbiased estimator), i.e. the unbiased estimator with the lowest variance.*

A.2 Matrix calculus

However, before deriving the first and second-order derivatives of the log-likelihood function Eq. 4, let's review basic requirements about linear algebra properties in Property A.4 and matrix calculus at first order in Property A.3 that will considerably ease the calculations.

Definition A.2 (Positive-definite matrix). *A symmetric real matrix $\boldsymbol{A}$ of rank G is said to be positive-definite if $\boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{x} > 0$ for all non-zero vectors $\boldsymbol{x}$ in $\mathbb{R}^G$.*

Property A.3 (Linear algebra properties). *First, we introduce below some properties associated to the determinant and trace operators. For a squared matrix A of rank G with defined inverse variance A^{-1} and a constant p, the following properties hold:*

a) $\mathrm{Det}(p\boldsymbol{A}) = p^G \mathrm{Det}(\boldsymbol{A})$ b) $\mathrm{Tr}\,(p\boldsymbol{A}) = p\,\mathrm{Tr}(\boldsymbol{A})$ c) $\mathrm{Det}(A^{-1}) = \frac{1}{\mathrm{Det}(A)}$

Then, we introduce some properties practical with the transpose operator, which switch the row and column indexes. Given two matrices $\boldsymbol{A}$ and $\boldsymbol{B}$, the following properties hold when computing their transpose:

1. $(\boldsymbol{A}^\top)^\top = \boldsymbol{A}$
2. $(\boldsymbol{A}\boldsymbol{B})^\top = \boldsymbol{B}^\top \boldsymbol{A}^\top$
3. ${\boldsymbol{A}^{-1}}^\top = \boldsymbol{A}^{-1}$ *with the additional constraint that $\boldsymbol{A}$ is a symmetric matrix.*

Another useful equality, given two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ in $\mathbb{R}^G$ and $\boldsymbol{A}$ a symmetric matrix [5] *of rank G:*

$$\boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{y} = \boldsymbol{y}^\top \boldsymbol{A}\boldsymbol{x}$$

Property A.4 (**matrix calculus:** first order). *Given two invertible, positive-definite matrices (see Definition in A.2) $\boldsymbol{A}$ and $\boldsymbol{B}$, with respective inverses $\boldsymbol{A}^{-1}$ and $\boldsymbol{B}^{-1}$, $A = \boldsymbol{A}(p)$ and $B = \boldsymbol{B}(p)$ being functions of a scalar variable p, the following properties hold:*

(a) $\frac{\partial \mathrm{Det}(\boldsymbol{A})}{\partial p} = \mathrm{Det}(\boldsymbol{A})\,\mathrm{Tr}\left(\boldsymbol{A}^{-1}\frac{\partial \boldsymbol{A}}{\partial p}\right)$ $\frac{\partial \boldsymbol{U}\boldsymbol{A}\boldsymbol{V}}{\partial p} = \boldsymbol{U}\frac{\partial \boldsymbol{A}}{\partial p}\boldsymbol{V}$ (c) $\frac{\partial \boldsymbol{A}^{-1}}{\partial p} = -\boldsymbol{A}^{-1}\frac{\partial \boldsymbol{A}}{\partial p}\boldsymbol{A}^{-1}$

From equation a), we can readily compute $\frac{\partial \log(\mathrm{Det}(\boldsymbol{A}))}{\partial p} = \mathrm{Tr}\left(\boldsymbol{A}^{-1}\frac{\partial \boldsymbol{A}}{\partial p}\right)$ using the chain rule applied to a logarithmic function. Additionally, the derivative is well-defined and continuous, since $\boldsymbol{A}$ is positive-definite, henceforth its determinant is strictly positive. Finally, using the linear algebras formulas stated in Property A.3, we deduce directly that: $\frac{\partial \log\left(\mathrm{Det}(\boldsymbol{A}^{-1})\right)}{\partial p} = -\,\mathrm{Tr}\left(\boldsymbol{A}^{-1}\frac{\partial \boldsymbol{A}}{\partial p}\right)$.

Additionally, both the additive property: $\frac{\partial c_1\boldsymbol{A} + c_2\boldsymbol{B}}{\partial p} = c_1\frac{\partial \boldsymbol{A}}{\partial p} + c_2\frac{\partial \boldsymbol{B}}{\partial p}$, c_1 and c_2 constants, and the multiplicative property: $\frac{\partial \boldsymbol{A}\boldsymbol{B}}{\partial p} = \boldsymbol{A}\frac{\partial \boldsymbol{B}}{\partial p} + \boldsymbol{B}\frac{\partial \boldsymbol{A}}{\partial p}$ of the derivative operator hold similarly for matrices as for real valued functions.

Given three matrices $\boldsymbol{A}$, $\boldsymbol{C}$, $\boldsymbol{D}$ and two vectors of same size $\boldsymbol{b}$ and $\boldsymbol{e}$, none of them depending on variable $\boldsymbol{p}$, the partial derivative with respect to $\boldsymbol{p}$ of the quadratic form $(\boldsymbol{A}\boldsymbol{p}+\boldsymbol{b})^\top \boldsymbol{C}(\boldsymbol{D}\boldsymbol{p}+\boldsymbol{e})$ is:

$$\frac{\partial (\boldsymbol{A}\boldsymbol{p}+\boldsymbol{b})^\top \boldsymbol{C}(\boldsymbol{D}\boldsymbol{p}+\boldsymbol{e})}{\partial \boldsymbol{p}} = (\boldsymbol{D}\boldsymbol{p}+\boldsymbol{e})^\top \boldsymbol{C}^\top \boldsymbol{A} + (\boldsymbol{A}\boldsymbol{p}+\boldsymbol{b})^\top \boldsymbol{C}\boldsymbol{D}$$

[5] if a matrix is symmetric, then by definition, $\boldsymbol{A}^\top = \boldsymbol{A}$

Notably, using the properties associated to the transpose operator, enumerated in Property A.3, setting $\boldsymbol{A} = \boldsymbol{D} = -\boldsymbol{x}$ *as vectors in* $\mathbb{R}^G$, $\boldsymbol{b} = \boldsymbol{e} = \boldsymbol{y}$ *and* $\boldsymbol{C} = \boldsymbol{Theta}$ *a symmetric matrix, the derivative simplifies to :*

$$\frac{\partial(\boldsymbol{y} - \boldsymbol{x}p)^\top \Theta(\boldsymbol{y} - \boldsymbol{x}p)}{\partial p} = -2(\boldsymbol{y} - \boldsymbol{x}p)^\top \Theta \boldsymbol{x} = -2\boldsymbol{x}^\top \Theta(\boldsymbol{y} - \boldsymbol{x}p)$$

6

To compute the Hessian matrix, we complete the matrix calculus properties detailed in Property A.4 with the following second-order matrix calculus derivations (Property A.5):

PROPERTY A.5 (**MATRIX CALCULUS:** SECOND ORDER). *Given an invertible matrix* $\boldsymbol{A}$ *depending on a variable* p*, the following calculus formulas hold:*

(a)

$$\frac{\partial^2 \boldsymbol{A}^{-1}}{\partial p_i \partial p_j} = \boldsymbol{A}^{-1}\left(\frac{\partial \boldsymbol{A}}{\partial p_i}\boldsymbol{A}^{-1}\frac{\partial \boldsymbol{A}}{\partial p_j} - \frac{\partial^2 \boldsymbol{A}}{\partial p_i \partial p_j} + \frac{\partial \boldsymbol{A}}{\partial p_j}\boldsymbol{A}^{-1}\frac{\partial \boldsymbol{A}}{\partial p_i}\right)\boldsymbol{A}^{-1}$$

(b)

$$\frac{\partial \operatorname{Tr}(\boldsymbol{A})}{\partial p_i} = \operatorname{Tr}\left(\frac{\partial \boldsymbol{A}}{\partial p_i}\right)$$

If the first-order derivatives of our matrix $\boldsymbol{A}$ *are symmetric, equation (a) further simplifies to:*

$$\frac{\partial^2 \boldsymbol{A}^{-1}}{\partial p_i \partial p_j} = \boldsymbol{A}^{-1}\left(2\frac{\partial \boldsymbol{A}}{\partial p_i}\boldsymbol{A}^{-1}\frac{\partial \boldsymbol{A}}{\partial p_j} - \frac{\partial^2 \boldsymbol{A}}{\partial p_i \partial p_j}\right)\boldsymbol{A}^{-1}$$

Combining the derivative formula for the inverse of a matrix, given in Property A.4 with the linearity of the trace operator and the chain rule formula yields in particular:

$$\frac{\partial^2 \log(\operatorname{Det}(\boldsymbol{A}))}{\partial^2 p} = -\operatorname{Tr}\left[\boldsymbol{A}^{-1}\frac{\partial^2 \boldsymbol{A}}{\partial^2 p_i}\right] + \operatorname{Tr}\left[\left(\boldsymbol{A}^{-1}\frac{\partial \boldsymbol{A}}{\partial p_i}\right)^2\right]$$

Finally, the generalised Leibniz product rule, given below , reproduced from [12], is also valid for the derivation of any product of m *matrices depending on a scalar* p *(we restrain the formula to the first order only):*

$$\frac{\partial(\boldsymbol{A}_1 \dots \boldsymbol{A}_m)}{\partial p} = \sum_{\boldsymbol{\pi}_m} \prod_{1 \le t \le m} \boldsymbol{A}_t^{k_t}$$

in which $\boldsymbol{\pi}_m = \{(k_1, \dots k_m) \in \mathbb{N}_0^m\} : \sum_{i=1}^m k_t = 1$ *(* $\boldsymbol{\pi}_m$ *is the set of integer tuples of size* m*, whose total sum equals the order of derivation, here 1). At the first order, deriving a product of* m *matrices is thus equivalent to sum all tuples of matricial products products, each deriving a different matrix.*

A.3 Tensor product

Before displaying the Hessian, we need to introduce the **tensors** concept. We consider their definition as multidimensional arrays, $\boldsymbol{X}[i_1, \dots, i_n]$, with the number of distinct indices used to index the elements of the array, n, being the *level* of the tensor (a vector is hence of level 1, while standard matrices are level 2), while the value $i_j, j \in \{1, \dots, n\}$ denotes the number of rows or columns at a given position.

[6] Some proofs about matrix calculus can be found in [11].

But first, let's consider the definition of a generalised tensor product Definition A.6 and the Einstein summation Definition A.7, which would enable us to write these derivatives more elegantly and to implement them more efficiently, using vectorised, highly-parallel calculations:

DEFINITION A.6 (**GENERALISED TENSOR PRODUCT**:). *Consider two tensors,* $\boldsymbol{X}_{i_1 \dots i_n h_1 \dots h_l}$, *of level* $n+l$ *and* $\boldsymbol{Y}_{j_1 \dots j_n k_1 \dots k_m}$, *of level* $n+m$, *with the additional constraint that the dimensions over the* n *extents match:*$(i_1, \dots i_n) = (j_1, \dots, i_n)$ *then the resulting tensor multiplication outcome is given by:*

$$\boldsymbol{E}_{h_1 \dots h_l k_1 \dots k_m} = \sum_{(i_1, \dots i_n)=(j_1, \dots j_n)} \boldsymbol{X}_{i_1 \dots i_n h_1 \dots h_l} \boldsymbol{Y}_{i_1 \dots i_n k_1 \dots k_m}$$

whose level is $l+m$ *(the operation collapses the matching* n *dimensions) and which is indexed by* $[\boldsymbol{h}, \boldsymbol{k}]$.

We can better interpret this operation as the following two-stage process:

1. ***Outer product:*** *We paired all elements,* $\boldsymbol{x}[\boldsymbol{i}] \bigotimes \boldsymbol{y}[\boldsymbol{i}]$ *, for which the multiindices* $\boldsymbol{i}$ *and* $\boldsymbol{j}$ *match*
2. ***Inner product:*** *We reduce by summation over the shared indexes the resulting cross product result.*

Note that the tensor product is a generalisation of the standard matrix product. The matrix product between $\boldsymbol{X} \in \mathcal{M}_{h_1, i_1}$ *and* $\boldsymbol{Y} \in \mathcal{M}_{j_1, k_1}$, *with* $i_1 = k_1$ *is given by, below:*

$$\begin{aligned} \boldsymbol{E} &= \boldsymbol{X} \times \boldsymbol{Y} \\ &= \langle \boldsymbol{x}_{h.}, \boldsymbol{y}_{.k} \rangle_{h \in \widetilde{h_1}, k \in \widetilde{k_1}} \quad (1) \\ &= \sum_{i=1}^{i_1} \underbrace{\boldsymbol{x}_{.i} \otimes \boldsymbol{y}_{i.}}_{\in \mathcal{M}_{h_1 k_1}} \quad (2) \end{aligned} \tag{14}$$

[7]

DEFINITION A.7 (**EINSTEIN SUMMATION:**). *The Einstein summation convention, is a practical and compact way, implemented in a variety of R packages, to represent the generalised tensor product. Briefly, we have the following equivalence:*

$$y = \sum_i c_i x_i \iff y = c_i x^i$$

where repeated indexes, also termed ***dummy indexes****, are summed over (here,* i*), while indexes not summed other (see for instance the set* $\{\boldsymbol{h}, \boldsymbol{k}\}$ *in Eq. 11) are called* ***free index****, and should appear only once.*

Hence, all dimensions of a tensor sharing the same index are summed together. Additionally, it is common use to separate covariant vectors (they are indexed with lower indices, subscripts and refer to row vectors, the index then indicating the position of the column) and contravariant vectors (column vectors indexed with subscripts, whom return the position of the row). A covariant vector can only be contracted with a contravariant vector (summation of the products of coefficients)[8].

[7] Note how the inner product can be interpreted as the reciprocal function of the outer product, comparing line (1) with line (2).

[8] However, in a Banach space, including the Euclidean one, it is possible to work only with subscripts

With the **R** packages **tensorA** and **tensor**, we compute as a toy example the numerical computations required to compute respectively D $\ell_{\boldsymbol{y}|\boldsymbol{\zeta}}(\boldsymbol{\psi})\cdot D^2\boldsymbol{\psi}$ in Algo. 1 and $D\boldsymbol{\psi}\cdot D^2\ell_{\boldsymbol{y}|\boldsymbol{\zeta}}(\boldsymbol{\psi})\cdot D\boldsymbol{\psi}$ in Algo. 2:

Listing 1. Computation of the First part of the Hessian

```
library(tensorA) # load package
J_log <- to.tensor(1:3,c("i"=3)) # Jacobian of the original log-likelihood
H_psi <- to.tensor(1:12,c(j=2,k=2,"i"=3)) # Hessian of the mapping function

# Einstein summation
hess_einstein <- J_log %e% H_psi # %e% refers to the Einstein operator,
# repeated indexes (here i)
# in both tensors are coupled and summed over,
# of note, any number of tensors can be passed
# %e% operator, provided you respect the shape configuration

# multivariate tensor product
hess_tensor_product <- mul.tensor(J_log,"i",H_psi,"i")
# the standard tensor product,
# in which you have to indicate explicitly the indexes to aggregate

#  with the tensor package
library(tensor)
# in opposition to the mult product, you instead indicate the distinct indexes
hess_tensor <- tensor(A = J_log, B = H_psi, alongA = 1, alongB = 3)

# with base R operators
hess_base <- matrix(0, nrow = 2, ncol = 2)
for (i in 1:3) {
  hess_base <- hess_base +
    (J_log[i] %>% as.numeric()) * (H_psi[,,i] %>% to.matrix.tensor(j="I2"))
}

# check that all these elements are equal
testthat::expect_equal(hess_rieman, hess_tensor %>% to.tensor()) # pass
testthat::expect_equal(hess_rieman, hess_tensor_product) # pass
# pass but requires cumbersome operation
testthat::expect_equal(hess_rieman, hess_base %>%
                         to.tensor() %>% extract2(I1 =~"j", I2 =~"k"))
```

Listing 2. Computation of the Second part of the Hessian

```
J_psi <- to.tensor(1:6,c("i"=3, "j"=2))
H_log <- to.tensor(1:9, c("i"=3, "l"=3))

# with base R
hess_base_R <- t(J_psi) %*% H_log %*% J_psi
# %*% designs here the dot product

# with the Einstein summation (needs an index change)
hess_einstein_second <- J_psi %e% H_log %e% J_psi[[i =~"l", j =~"k"]]
testthat::expect_equal(hess_base_R,
                       hess_einstein_second %>% matrix(nrow=2, ncol=2)) # pass
```

A.4 Iterative descent algorithms

General algorithmic steps are reviewed in Definition A.8.

Definition A.8 (**Iterative optimisation algorithms**). *Given $\boldsymbol{p}_0 \in]0,1[^J$, an initial starting point (without prior information, consider equally balanced cellular ratios, $p_j = \frac{1}{J}$ seems a rather good compromise for our problem), f any real-valued function we want to optimise and the iteration counter set to $i = 0$, we performed the following iterated series of actions at each step i, while the stopping criterion has not been met (likely a user defined number of iterations combined with a relative or absolute differential between the computation of two consecutive solutions):*

- *Define a direction descent d_i. The general idea is to find a relevant direction which is the perfect trade-off between stability and speed to the converging point. Notably, we generally expect that $f(\boldsymbol{p}_{i+1}) > f(\boldsymbol{p}_i)$.*
- *Define a positive step size, or learning rate $\alpha_i > 0$. While the simplest way is to keep it constant throughout the descent process, if f met some conditions, better updating convergence rates can be designed. Update the solution: $\boldsymbol{p}_{i+1} = \boldsymbol{p}_i + \alpha_i d_i$ (if the objective is minimisation, we instead have to progress in the opposite direction).*
- *Update the counter: $i \leftarrow i + 1$*

It is worth noted that the roots of the gradient only indicate *critical points*. To discriminate a maximum from any other topological configurations, such as a saddle point, a plateau or a minimum, we must enforce that the Hessian matrix is at that point negative semi-definite [13, 14]. If the Hessian matrix is globally negative semi-definite, it is even better, since this demonstrates the strict convexity of the function to optimise, hence insuring that it only displays a global maximum. However, when it not the case, we have no stability and theoretical guaranty of converging to the global maximum. The choice of the starting point gets then paramount to converge to a global optimum, and alternative methods, such as simulated annealing that smooth the distribution, may display better performance.

B Perspectives

The new deconvolution algorithm we introduced here, DeCovart, is the first one based on a multivariate generative statistical model that additionally implements explicilty the simplex constraint. Hence, it provides a strong basis to further derive theoretical confidence bands and generate statistical tests to assert whether a cell population is absent or not, or if the proportion of a cell population differs between two distinct biological conditions. We are also able to leverage a new set of genes with close mean expression profiles but distinct co-expression patterns, which were traditionally discarded by other deconvolution methods. We hence believe that the higher flexibility of our deconvolution algorithm will make it relevant to increase the currently poor cell resolution of deconvolution methods, with an extended ability to discriminate highly correlated cell population.

However, we still need to assert in a real-world experience the performance of our algorithm, by benchmarking it comprehensively against the standard deconvolution aglorithms DeconRNASeq [15], CIBERSORT [16], MuSiC [17] and xCell [18]. A good estimation on the monocytes subset, till now poorly estimated, would make our method unavoidable.

Intensive work has still to be done to refine the plug-in parameters provided to our deconvolution optimisation algorithm. The sparse estimate of the precision matrix returned by the gLasso [19] algorithm is generally shrunk, entailing in practice that the non-null partial correlations are generally underestimated. *Parameter shrinkage* is a common and well-documented issue of regularisation methods that penalise the complexity of the model. A way to circumvent this problem is to use the *support* returned by the penalisation method to refine the estimation by a canonical maximum likelihood strategy that would integrate the topological constraints induced

by the null inputs of the precision matrix (an item set to zero means no direct edge connecting the two transcripts).

Working on the whole set of the transcriptome is likely to result in intensive, irrelevant computational issues, hence, a first step of most deconvolution algorithms consist of retrieving a minimal subset of genes to discriminate any circulating cell population. While none of the previous approaches account for differential network structures, the INDEED algorithm [20], tailored to select biomarkers between two biological conditions using both differential mean expression and specific cellular network connections, may be highly relevant for that purpose. However, it has not been designed to compare many different groups. While we worked around the problem by performing an one-vs-all strategy, such an approach is quite controversial, since this heuristic strategy does not account for the specificity of each cell profile, the remaining expression profiles being averaged, (discrimination of minor cell populations with lower depth sequencing are likely to be confused with such strategy), nor enable to identify collectively the minimal subset of genes able to discriminate any cell population. Additionally, the way the Indeed algorithm performs to combine the close transcriptomic neighbourhood and the mean expression within a single metric function is derogatory, since the units are not on the same scale.

Rather, a multi-objective and global optimisation approach seems an excellent alternative to manage the trade-off of neighbourhood and mean gene expression discrimination at the scale of a cell population. Finally, while exploring globally the space of gene subsets is generally infeasible due to the combinatorial explosion of possibilities (2^G with G the number of genes retained after background filtering), a genetic and evolutionary approach, likewise to the AutoGeneS algorithm [21]), in which a population of candidate solutions are randomly modified (each solution is the bite wise or indicator function of a set of genes, a zero input representing a gene discarded) and iteratively optimised to finally return a collection of "Pareto-solutions" that represent on a two-dimensional plot the best trade off between the two metrics compared.

Finally, we could refine our generative model to integrate heterotypic interactions, namely accounting for modifications induced by a change of the environmental medium, like a the release of a signalling molecule or a dysregulation of the metabolic pathways induced by a genetic mutation, or a technical batch. A mixed linear-model could be used to account for known environmental and technical confusing factors. However, if no reference profile is available or the nature of the confusing variable is unknown, a better alternative would be to encompass any latent driver within a fully Bayesian framework. Additionally, A Bayesian approach would also alleviate the difficulty of asserting the statistical relevance of the estimates, since it would directly return from the simulation *likelihood intervals*.