diffGEK: Differential Gene Expression Kinetics

Melania Barile and Shirom Chabra

Vignette

Introduction

diffGEK is a method designed to uncover differences in transcriptional kinetics in single cell differentiation trajectories, provided that there are at least two different biological conditions. By transcriptional kinetics we mean the rates of transcription, splicing and degradation, as classically defined by established RNA velocity tools such as (La Manno et al., 2018; Bergen et al., 2020). As in the cited papers, we aim at inferring the kinetics based on the following ODE system:

$$\begin{aligned} \dot{U}_g(t) &= \alpha_g(t) - \beta_g(t)\, U_g(t) \\ \dot{S}_g(t) &= \beta_g(t)\, U_g(t) - \gamma_g(t)\, S_g(t) \end{aligned} \tag{1}$$

where $U_g(t)$ and $S_g(t)$ are the number of unspliced and spliced counts for gene g at pseudotime t, respectively; $\alpha_g(t)$, $\beta_g(t)$ and $\gamma_g(t)$ are the kinetic parameters (transcription, splicing and degradation rate respectively) for gene g at pseudotime t. Compared to the other papers. however, we modelled the kinetic parameters in a novel way. First, the parameters are gene specific. Second, they are a function of the cellular state, i.e., of the pseudotime. Since this would imply a big number of unknown variables to estimate, we further assumed that the parameters can be modelled with natural cubic splines.

Once the kinetic rates are computed, we need a procedure to decide whether they are differential or not among the two biological conditions over the whole trajectory of interest. To this aim, we consider 8 possible models, M1-M8, as in the corresponding Table 1.

	α	β	γ	k
M1	same	same	same	32
M2	any	same	same	40
M3	same	any	same	40
M4	same	same	any	40
M5	any	any	same	48
M6	any	same	any	48
M7	same	any	any	48
M8	any	any	any	56

Table 1: Schematic of the eight possible models used to fit the data. "same" means that the corresponding rate is forced to be the same across conditions. "any" means that it can be different. α: transcription rate; β: splicing rate; γ: degradation rate; k: number of model parameters (eight spline nodes per rate, four initial conditions and four estimated errors).

For each gene, we need to fit all 8 models and then rank the models to decide which one fits the data better, relying at the same time on the smallest possible number of parameters (k in the table). This is achieved upon comparing the Aic information criterion index. Once the model is picked, we can conclude that the corresponding genes has differential kinetics or not, and which of the three kinetics are different.

Requirements and Installation

To obtain unspliced counts, users need to run specific tools such as velocyto (La Manno et al., 2018).

For our paper, we used cellRank (Lange et al., 2022) to select trajectories, but the trajectory can be selected with any tool.

To prepare the input for the MATLAB pipeline, diffGEK requires a pre-analysis performed via scVelo (Bergen et al., 2020), that imputes the spliced and unspliced counts.

After the input has been prepared, the MATLAB pipeline for fitting the model starts. Throughout the pipeline, we used the same scheme and similar codes as in Pseudodynamics (Fischer et al., 2019), and thus diffGEK requires the same software as these tools: AMICI and PESTO. AMICI solves the models in the background with c++, and thus requires a c++ compiler.

Note that diffGEK, like Pseudodynamics, is a collection of scripts and not a tool to install. The scripts have to be adapted to the specific experimental design, and the model executable needs to be recompiled on the machine used to perform the analysis.

Prepare input

For showcasing purpose, we now present the pipeline for the myelopoisis dataset.

As mentioned above, diffGEK works on a single cell differentiation trajectory; this could be, for example, stem cells producing a certain lineage. It also relies on spliced and uspliced count information, which the user has to compute prior to the analysis, usually with the velocyto tool (La Manno et al., 2018). For this dataset, particularly, we ran:

```
velocyto run10x ./bam_folder/sample_name.bam
   ./references/cellranger-hg19-3.0.0/genes.gtf
```

The above command produces a .loom file for each sample, containing the spliced and unspliced counts.

The first step is to create a single cell landscape and then subset the trajectory of interest. In our paper, we either took an erythropoiesis trajectory as computed by the authors of the respective paper (Isobe et al., 2023), or computed the myelopoiesis trajectory of a published dataset (Liu et al., 2021) by means of the CellRank tool (Lange et al., 2022). We showed how we did it in the notebook `myelopoiesis/run_models/prepare_landscape.ipynb`. Note, however, that the landscape creation/trajectory selection can be achieved with any tool of preference. Overall, we need a trajectory where, for each cell, we have spliced and unspliced counts, and a pseudotime ordering.

Next, we need to create the input for diffGEK, which will be achieved in two steps. The first consist in creating a count matrix for each condition and for the spliced/unspliced counts separately. The tables, which follow the scanpy convention cells x genes, also have a column for pseudotime. In the first step, we rely on the scVelo framework to create imputed counts to feed into the model. This step is necessary because the unspliced counts are very noisy. The foundamental steps are:

- Load an anndata object containing raw (or lognormalised) counts, a UMAP landscape, and the layers "spliced" and "unspliced".

```
adata = sc.read('./data_input/adata_myelopoiesis.h5ad')
adata_store = adata.copy()
```

- Filter low quality genes, lognormalize the layers and select HVGs. We recommend to keep the number of genes low, max 2000, because noisy genes (particularly noisy unspliced counts) affect the results at both the imputation and the later fitting step. Note that, if your counts are already lognormalised, this line of code will only filter and select HVGs. It is worth stressing once more that this filtering step is not optional. Noisy genes have to be removed, otherwise they will affect the results, reducing statistical power.

```
scv.pp.filter_and_normalize(adata,min_shared_counts=20,
   n_top_genes=1000)
```

- Impute spliced and unspliced counts. This step is necessary because of unspliced counts being noisy. It creates two extra layers in the adata object, 'Ms' and 'Mu', that will be used downstream.

```
scv.pp.moments(adata,n_neighbors=30,n_pcs=50)
```

- Save a dataframe per each count type and condition (spliced/unspliced, WT/mutant) and the list of genes used.

```
# spliced
dfs = pd.DataFrame(adata.layers['Ms'])
dfs.columns = adata.var_names
dfs.index = adata.obs.dpt_pseudotime

dfs_m = dfs.iloc[np.where(adata.obs.wt_ko == 'ko')[0],:].copy()
dfs_w = dfs.iloc[np.where(adata.obs.wt_ko == 'wt')[0],:].copy()

# unspliced
dfu = pd.DataFrame(adata.layers['Mu'])
dfu.columns = adata.var_names
dfu.index = adata.obs.dpt_pseudotime

dfu_m = dfu.iloc[np.where(adata.obs.wt_ko == 'ko')[0],:].copy()
dfu_w = dfu.iloc[np.where(adata.obs.wt_ko == 'wt')[0],:].copy()

# write
dfs_m.to_csv('./data_input/myelopoiesis_ko_spl.csv')
dfu_m.to_csv('./data_input/myelopoiesis_ko_unspl.csv')

dfs_w.to_csv('./data_input/myelopoiesis_wt_spl.csv')
dfu_w.to_csv('./data_input/myelopoiesis_wt_unspl.csv')

pd.DataFrame(adata.var_names).to_csv('./data_input/list_genes.csv')
```

All steps above are contained in the notebook: `run_models/prepare_input.ipynb`. Note, however, that in the paper analysis we added an extra set of genes taken from tradSeq for the purpose of comparison with the differentially expressed genes. The user doesn't need to add other genes, unless they want to.

For the second step, we switch to MATLAB, because the fitting procedure will require MAT-

LAB codes. The code to run is: `run_models/create_input.m`. Note that running a MALAB code means just to enter create_input on the MATLAB terminal or in a script. To run it remotely, however, the following code can be run in bash:

```
matlab -nodisplay -nojvm -r "create_input"
```

This code creates a MATLAB object (`gene_name.mat`) per each gene, containing the input for the fit (pooled spliced and unspliced counts at specific pseudotime intervals).

The code can in principle just be run. However, there are two options that can be modified if the users has a dataset where the trajectory is very "long". By this we mean that the maturation distance from the immature progenitors to the end of the trajectory is high, that is, there are several intermediate progenitors in between. More precisely, the number of pseudotime bins is set to 20 for our analysis, but can be increased or reduced. In this case however, we recommend also to change the model building codes (see section Optional: modifying the model files), because more bins may require more nodes for the kinetic parameters' estimation. Similarly, the model building codes were calibrated for a trajectory where the pseudotime goes from 0 to 0.5. If, however, the user has computed a pseudotime that spans a very different range, say, from 0 to 20, they need to either change the model building files, or simply scale their pseudotime to range between 0 and 0.5 (in this case they would need to change the parameter `scale_time` to be equal to 0.25).

After running this code, the input is ready.

Fitting procedure

We now need to fit each gene with each of the 8 models explained above. The first thing to do is to create executable files that will generate the model predictions. To do so, users can just compile the files of the type `model_building/kinetics_modeln_syms.m`, where $n = 1 - 8$ (go to the section below: Compiling the models). However, if the user has some specific requirements in their dataset, like three instead of two conditions, or they want to modify the number of bins at which they evaluate the models, or the number of splines' nodes, they can modify the files accordingly. To be able to do so, we now explain the content of these files.

Optional: modifying the model files

The `model_building/kinetics_modeln_syms.m` files contain all the information on how the ODE model is structured. Particularly:

- p is the vector of parameters: since we decided to have 8 spline nodes, there are 8 parameters for the transcription rate for the WT condition (corresponding to the splicing nodes); 8 for the splicing rate for the WT condition; 8 for the degradation rate for the WT condition; 8 for the transcription rate for the mutant condition if different from WT in the corresponding model; 8 for the splicing rate for the mutant condition if different from WT in the corresponding model; 8 for the degradation rate for for the mutant condition if different from WT in the corresponding model; we then have 4 parameters for the initial conditions (WT/KO, spliced/unspliced); 4 for the estimated errors on the time courses (WT/KO, spliced/unspliced). The number of spline nodes can be any, but, if the users wants to modify it, it has to be manually modified in all these files. Similarly, if there are three conditions all the parameters have to be updated accordingly (however, to avoid dimensionality curse, we recommend to compare two conditions at the time, instead of three). This is the respective code:

```
syms aw1 aw2 aw3 aw4 aw5 aw6 aw7 aw8...
bw1 bw2 bw3 bw4 bw5 bw6 bw7 bw8...
gw1 gw2 gw3 gw4 gw5 gw6 gw7 gw8...
ak1 ak2 ak3 ak4 ak5 ak6 ak7 ak8...
bk1 bk2 bk3 bk4 bk5 bk6 bk7 bk8...
gk1 gk2 gk3 gk4 gk5 gk6 gk7 gk8...
Iuw Isw Iuk Isk e1 e2 e3 e4

p = [aw1,aw2,aw3,aw4,aw5,aw6,aw7,aw8,...
   bw1,bw2,bw3,bw4,bw5,bw6,bw7,bw8,...
   gw1,gw2,gw3,gw4,gw5,gw6,gw7,gw8,...
   ak1,ak2,ak3,ak4,ak5,ak6,ak7,ak8,...
   bk1,bk2,bk3,bk4,bk5,bk6,bk7,bk8,...
   gk1,gk2,gk3,gk4,gk5,gk6,gk7,gk8,...
   Iuw,Isw,Iuk,Isk,e1,e2,e3,e4];
```

- t_all is the set of time intervals at which to compute the mean spliced/unspliced values. It can be changed by the user (both the number of intervals, and the length, which

can be different per each interval), but it has to be consistent with the above mentioned `create_input.m` code. This is the code:

```
   t_all = [
      0,0.0682,0.1363,0.2045,0.2727,0.3409,0.4090,0.4772];
```

- splines calculation; only change it if you modify the number of spline nodes:

```
alpha_w =
   am_spline(t,lt,t_all(1),aw1,t_all(2),aw2,t_all(3),aw3,...
t_all(4),aw4,t_all(5),aw5,t_all(6),aw6,t_all(7),aw7,...
t_all(8),aw8,0,0.0);

beta_w =
   am_spline(t,lt,t_all(1),bw1,t_all(2),bw2,t_all(3),bw3,...
t_all(4),bw4,t_all(5),bw5,t_all(6),bw6,t_all(7),bw7,...
t_all(8),bw8,0,0.0);

gamma_w =
   am_spline(t,lt,t_all(1),gw1,t_all(2),gw2,t_all(3),gw3,...
t_all(4),gw4,t_all(5),gw5,t_all(6),gw6,t_all(7),gw7,...
t_all(8),gw8,0,0.0);

%%
alpha_k =
   am_spline(t,lt,t_all(1),ak1,t_all(2),ak2,t_all(3),ak3,...
t_all(4),ak4,t_all(5),ak5,t_all(6),ak6,t_all(7),ak7,...
t_all(8),ak8,0,0.0);

beta_k =
   am_spline(t,lt,t_all(1),bk1,t_all(2),bk2,t_all(3),bk3,...
t_all(4),bk4,t_all(5),bk5,t_all(6),bk6,t_all(7),bk7,...
t_all(8),bk8,0,0.0);

gamma_k =
   am_spline(t,lt,t_all(1),gk1,t_all(2),gk2,t_all(3),gk3,...
t_all(4),gk4,t_all(5),gk5,t_all(6),gk6,t_all(7),gk7,...
```

```
t_all(8),gk8,0,0.0);
```

- ODE system; only change it if you have more than two conditions (not recommended):

```
xdot(1) = alpha_w - beta_w * x(1);
xdot(2) = beta_w * x(1) - gamma_w * x(2);
xdot(3) = alpha_k - beta_k * x(3);
xdot(4) = beta_k * x(3) - gamma_k * x(4);

x0 = [Iuw,Isw,Iuk,Isk];
```

Compiling the models

To compile such files, use the provided `kinetics_wrap.m` function. This will create 8 more auxiliary files (no need to change anything) and 8 `.mex` executables.

```
matlab -nodisplay -nojvm -r "kinetics_wrap"
```

Running the models

We now need to run the fitting procedure. This is by far the most computationally intense step. Each gene has to be run independently, and requires little memory, but many genes and many models have to be run; for this reason we recommend to parallelize the analysis as much as possible. For example, to run model 1 for a certain gene, users can just run this code:

```
matlab -nodisplay -nojvm -r "main_model1($gene_number,10,1,2)"
```

where `gene_number` ranges from 1 to the number of genes to analyze. The second parameter (10) represents the regularization parameters for the splines, as explained in our Method session. No need to change this parameter unless the user wants more/less wobbly splines. The remaining parameters (1,2) are standard values that do not need to be changed, unless the users has changed the model structure in the model building session.

For each gene, we recommend to start from 200-300 initial conditions, which translates in a few minutes to finish one gene per each model. Note that the hardcoded version uses 3000 initial conditions because we wanted to be sure to found the global minimum, but we realized

it was not necessary a posteriori. To tune this parameter, modify the part:

```
optionsMultistart.n_starts = 3000
```

Each of the `main_modeln` codes performs a fit that maximizes a likelihood function, which is provided in the auxiliary files: `run_models/llKin_model1.m`. Users don't need to change these files unless they want to change the likelihood function, or they have changed any structural parameter in the `model_building/kinetics_modeln_syms.m` files. Note that if you change the likelihood you will also need to change the gradient (the derivative of the likelihood function with respect to the parameters).

After the fitting procedure is done, we will have 8 * number of genes files in the folder data_output. Each file contains the parameters and the loglikelihood values corresponding to each trial, but sorted in such a way that the first values correspond to the best possible model (maximum likelihood).

'Samples' mode

In case of matched replicates, diffGEK can also be run in 'samples' mode.

In this case, each replicate is treated as a different dataset, but the parameters have to be shared among the replicates. This analysis showed, in our testing analysis, to spot more differential genes, but it also requires more parameters (more initial conditions) and the convergence is slower.

The scheme to follow to run the analysis in sample mode can be found in the folder: `samples_mode`. Note, however, that the provided code are hardcoded for the number of samples, in our case three. For any other number of samples, the codes have to be changed and recompiled. In principle, also the two regularization parameters (see the section Simulations) have to be re-optimized for each number of replicates (we have only tested n = 3, as in our dataset).

Analyze results

We can now perform the analysis of the results, which is separated in 4 steps for convenience.

- The file `stats/c1_analyse_genes.m` collects the best fit parameters and computes the likelihood as weighted by the regularization term (as previously estimated by the simulations procedure; see section below).
- The file `stats/c2_write_aic.m` computes the corrected Akaike information criterion index (Aic) for each model.
- The file `erythropoiesis/stats/c3_write_statistic_aic.m` picks the best model (smaller Aic), and writes a recap file for all genes together on the file `statistic_aic.txt`. Furthermore, if one wants to create plots to show the goodness of the fits and the genes' trend along pseudotime, the file `stats/c4_write_parameters.m` writes a file where all the estimated parameters are written in the same file: `fit_parameters.txt`.

The above codes can be run in a cascade:

```
matlab -nodisplay -nojvm -r "c1_analyse_gene"
matlab -nodisplay -nojvm -r "c2_write_aic"
matlab -nodisplay -nojvm -r "c3_write_statistics_aic"
matlab -nodisplay -nojvm -r "c4_write_parameters"
```

The analysis could also finish here, as the classification of the genes has been performed. The output file provides a list with two columns: the first columns is a sequence 1 - number of genes, and the second column shown a number from 1 to 8, representing the best corresponding model. If the user wants to use our codes to create some summary statistic of the selected models, they can use the codes present in the folder `create_figures`. The notebook `compare_list_genes` was used to compare our differential kinetics against some other list genes, for example the tradeSeq genes, upon performing an hypergeometric test. Plotting the model results can be useful also to visually evaluate the goodness of the fit. We discuss this in the session Plotting fit results.

Plotting fit results

To evaluate model performance and to inspect the spline functions, we provide extra codes.

The code `plot_all_parameters_best.m` plots, for all genes, the differences for the two conditions for the three kinetic parameters, corresponding to the best fit models. Let's have a

look at some examples in Figure 1. In the top row, we see a gene for which no parameter have significative differences for the two conditions. Thus, the difference among the conditions is zero over pseudotime for all parameters. In the second row, however, we have a gene for which the transcription was evaluated as different among the conditions. Thus the line is nonzero for the transcriptional parameter. Note that a line below zero means that the corresponding parameter is higher for the KO than for the WT.

We move now to the fit quality. The code `plot_all_models_best.m` plots, for all genes and all models, the best fit for both conditions, against the data points. The code also gives the accepted model (labeled as best). Let's consider the example from Figure 2: Each line is a different model, spanning from 1 to 8. Model 4 was picked as the best because it fits the data well AND with the minimum number of parameters. Visual inspection shows that indeed the fit worked and the model choice was sensible.

Simulations

In our fitting procedure we leverage two regularization parameters: ρ and μ. The function of ρ is to keep the splines less wobbly; μ regularizes the estimated error. We estimated these parameters via the codes present in the folder `in_silico`. The scheme is similar to that used for the real datasets, but with a few extra features.

- Since these are simulations, we need to create the ground truth. To this aim, we run the codes in the folder `generate_data`. Each code simulates 100 genes, for a total of 800 simulations, for each of which a `.mat` object is stored, that keeps track of the ground truth parameters.
- Then each gene is refitted with each model, for a total of 64000 fits for each regularization parameter.
- Similarly, the folder `stats` contains the files to produce the stats for each combination of the regularization parameters. From the comparison of these results to the ground truth, the best regularization parameters are picked.
- An extra file, `plot_parameters_compare_ground_truth.m`, is provided to plot the simulations and compare the estimated parameters compared to the ground truth.

Note that the user doesn't need to repeat any of this calibrations, but they may if they change the models significative or have good reason to think that the calibration parameters don't work for their particular system.

References

Bergen, V., Lange, M., Peidli, S., Wolf, F. and Theis, F. (2020). Generalizing RNA velocity to transient cell states through dynamical modeling. Nature Biotechnology *38*, 1–7.

Fischer, D. S., Fiedler, A. K., Kernfeld, E. M., Genga, R. M. J., Bastidas-Ponce, A., Bakhti, M., Lickert, H., Hasenauer, J., Maehr, R. and Theis, F. J. (2019). Inferring population dynamics from single-cell RNA-sequencing time series data. Nature Biotechnology *37*, 461–468.

Isobe, T., Kucinski, I., Barile, M., Wang, X., Hannah, R., Bastos, H. P., Chabra, S., Vijayabaskar, M., Sturgess, K. H., Williams, M. J., Giotopoulos, G., Marando, L., Li, J., Rak, J., Gozdecka, M., Prins, D., Shepherd, M. S., Watcham, S., Green, A. R., Kent, D. G., Vassiliou, G. S., Huntly, B. J., Wilson, N. K. and Göttgens, B. (2023). Preleukemic single-cell landscapes reveal mutation-specific mechanisms and gene programs predictive of AML patient outcomes. Cell Genomics , 100426.

La Manno, G., Soldatov, R., Zeisel, A., Braun, E., Hochgerner, H., Petukhov, V., Lidschreiber, K., Kastriti, M. E., Lönnerberg, P., Furlan, A., Fan, J., Borm, L. E., Liu, Z., van Bruggen, D., Guo, J., He, X., Barker, R., Sundström, E., Castelo-Branco, G., Cramer, P., Adameyko, I., Linnarsson, S. and Kharchenko, P. V. (2018). RNA velocity of single cells. Nature *560*, 494–498.

Lange, M., Bergen, V., Klein, M., Setty, M., Reuter, B., Bakhti, M., Lickert, H., Ansari, M., Schniering, J., Schiller, H. B., Pe'er, D. and Theis, F. J. (2022). CellRank for directed single-cell fate mapping. Nature Methods *19*, 159–170.

Liu, Y., Gu, Z., Cao, H., Kaphle, P., Lyu, J., Zhang, Y., Hu, W., Chung, S. S., Dickerson, K. E. and Xu, J. (2021). Convergence of oncogenic cooperation at single-cell and single-gene levels drives leukemic transformation. Nature Communications *12*.

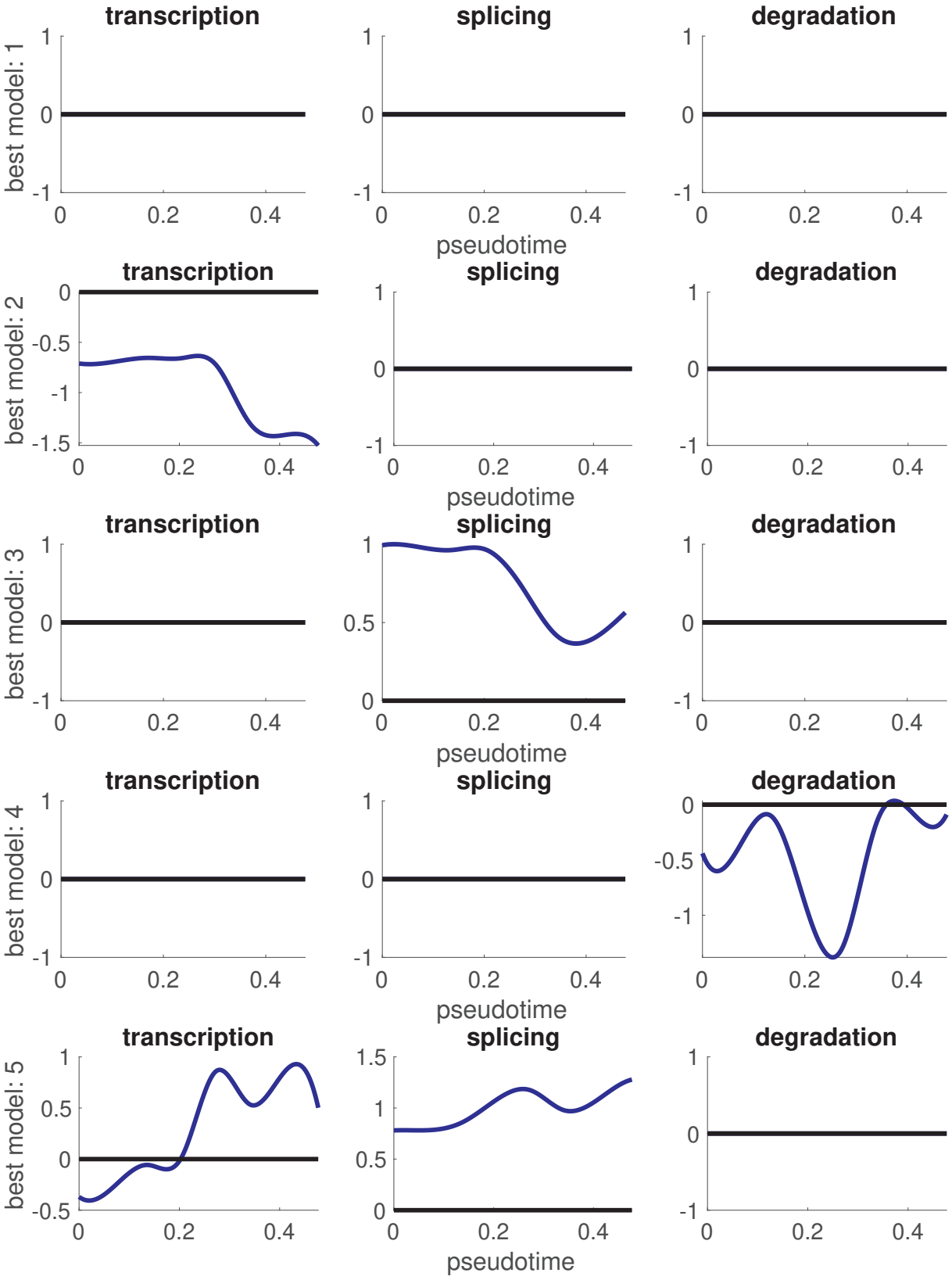

Figure 1. Examples of genes for which the best fit model ranges from 1 to 5. Model 1 means no differences for the two conditions, thus the difference for each parameter is 0. Model 2 means differences are only for transcription, and so on. Note that a line below zero means that the corresponding parameter is higher for the KO than for the WT.

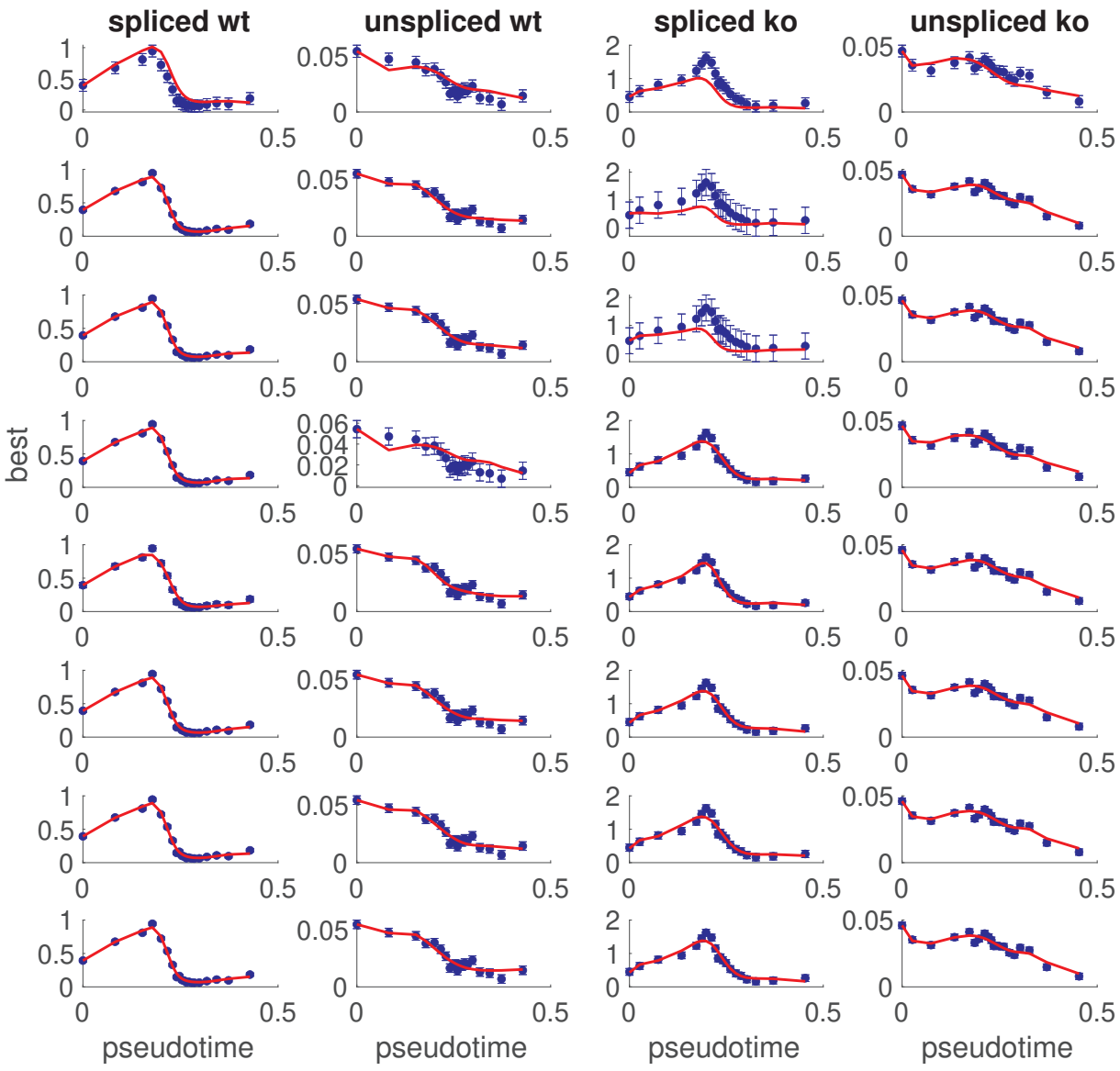

Figure 2. Examples of a gene for which the best fit model is 4. Each line is a different model, 1 to 8. Red: best fit model. Blue: data.