Data

Generate simulated data

Let’s start with simulated problem. Here you can find a binary network of 78 genes and 9 TFs

```
binaryCS = Import[NotebookDirectory[] <> "Data files/binaryCS.csv"];
{numGenes, numTFs} = Dimensions[binaryCS]
```

```
{78, 9}
```

To see this network in matrix form (Transposed for easier viewing):

```
binaryCS // Transpose // MatrixForm
```

$$\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}$$

To make a tractable problem, let’s consider a data set of 12 samples, one deletion sample per TF, and 3 additional samples from different conditions.
The binary knowledge of activity for such a data set can be found here:

```
binaryTFALabeled =
  Import[NotebookDirectory[] <> "Data files/binaryTFALabeled.csv"];
binaryTFA = binaryTFALabeled[[2 ;;, 2 ;;]];
binaryTFALabeled // MatrixForm
numSamples = Dimensions[binaryTFA][[2]];
```

	YCR065W	YDR123C	YDR174W	YDR463W	YGL131C	YHL020C	YJR060W	YLR403W	YO
YCR065W-HCM1	0	1	1	1	1	1	1	1	
YDR123C-INO2	1	0	1	1	1	1	1	1	
YDR174W-HMO1	1	1	0	1	1	1	1	1	
YDR463W-STP1	1	1	1	0	1	1	1	1	
YGL131C-SNT2	1	1	1	1	0	1	1	1	
YHL020C-OPI1	1	1	1	1	1	0	1	1	
YJR060W-CBF1	1	1	1	1	1	1	0	1	
YLR403W-SFP1	1	1	1	1	1	1	1	0	
YOL108C-INO4	1	1	1	1	1	1	1	1	

The activity file is labeled with a TF (systematic name and common name) along the rows. A TF name for a column label refers to the TF whose gene has been deleted. WT is short for Wild Type and refers to a sample that does not have a tampered genome.

Let’s populate the control strength and activity matrices with actual values:

```
SeedRandom[1];
simulatedControlStrength = Table[RandomReal[{0.1, 5}] * RandomChoice[{-1, 1}],
    {i, numGenes}, {j, numTFs}] * binaryCS;
simulatedControlStrength // Transpose // MatrixForm
(*simulatedActivity =
 Table[RandomReal[{0.5, 3}], {i, numTFs}, {j,  numSamples}]*binaryTFA;
simulatedActivity//MatrixForm*)
meanCS =
  Map[(Mean[DeleteCases[#, 0.]]) &, Transpose[Abs[simulatedControlStrength]]];
simulatedActivity =
  Map[Abs[RandomVariate[NormalDistribution[#, 1], numSamples]] &, meanCS] *
   binaryTFA;
simulatedActivity // MatrixForm
```

0.	0.	0.	0.	0.	0.	0.	0.	0.	
1.19191	0.	0.	-0.543188	0.	0.	0.	0.	0.	
0.	0.	0.	0.	0.	0.	0.	0.	0.	
2.56188	4.98167	0.	0.	0.	0.	4.52124	0.	-2.37323	
0.	0.	0.	0.	-2.04668	0.	0.	0.	0.	
0.	0.	0.	0.	0.	0.	0.	0.	0.	
0.	0.	0.	0.	0.	0.	0.	4.51807	0.	
0.	0.	4.45579	0.	0.	-2.61987	0.	0.	0.	4.
0.	0.	0.	0.	0.	0.	0.	0.	0.	

0.	1.24117	0.12947	1.5599	2.51042	1.30268	0.347586	0.940853	2.17573	1.97
0.912078	0.	1.78076	1.0669	3.97335	1.32734	1.96982	3.20312	2.54618	1.45
0.603013	3.42316	0.	4.26511	3.37181	2.60816	1.81614	3.35422	2.65859	0.78
2.43026	1.0888	1.05238	0.	1.97413	2.36927	2.96996	2.23792	3.01849	1.54
3.52297	2.09912	1.18948	3.83606	0.	3.71079	4.2806	3.34341	1.37869	3.37
3.73918	1.71179	3.62245	1.68938	3.42949	0.	3.01256	2.05134	1.74369	3.0
3.57928	3.98276	2.21499	3.29969	4.15633	3.42688	0.	1.75989	3.36976	3.99
4.13876	2.78589	1.82991	2.21474	2.40967	1.99697	3.51937	0.	4.14716	0.35
1.5146	1.9428	0.86989	2.15099	1.49213	2.32178	1.85101	1.61412	0.	2.76

Given these values, and assuming our non-linear model of TFA with saturation is correct, what would we expect the gene expression values to be?

This is the model:

Expression of gene 1 in sample 1 =

baseline of gene 1

+ scaling factor of gene 1 *

[Sum over activators (Activity of TF / (Activity of TF + control strength of TF on gene 1))

+ Sum over repressors (1 / (Activity of TF + control strength of TF on gene 1))]

We still need baseline and scaling factors, so let’s simulate those too:

```
SeedRandom[2];
simulatedBaselines = RandomReal[{0.1, 5}, numGenes];
simulatedScaling = RandomReal[{0.1, 5}, numGenes];
```

And now, the expression of gene 1 is:

```
gene1Sample1Expression = 0;
For[i = 1, i ≤ numTFs, i++,
  sample1TFA = simulatedActivity[[i, 1]];
  (*the first index is the TF, the second index is the sample*)
  tfCS = simulatedControlStrength[[1, i]];
  (*the first index is the gene, the second index is the TF*)
  If[tfCS > 0, (*if the TF is an activator*)
   gene1Sample1Expression += sample1TFA / (sample1TFA + tfCS)];
  If[tfCS < 0, (*if the TF is a repressor*)
   gene1Sample1Expression += 1 / (sample1TFA - tfCS)]
 ];
gene1Sample1Expression *= simulatedScaling[[1]];
gene1Sample1Expression += simulatedBaselines[[1]]
```

```
4.4438
```

Go ahead and write a function to return an entire expression matrix of 54 genes x 9 samples using the simulated data:

```
simulatedExpressionMatrix = calculateExpression[simulatedControlStrength,
   simulatedActivity, simulatedBaselines, simulatedScaling];
simulatedExpressionMatrix // Dimensions
```

```
{78, 12}
```

Please check that the first 10 rows are the same as what I got:

```
simulatedExpressionMatrix[[1 ;; 10]] // MatrixForm
```

4.4438	3.89979	4.41748	4.05203	4.69229	4.51991	4.65332	4.68407	4.70768	4.44
2.00889	1.38713	1.3664	0.636298	1.82438	1.98555	2.19986	1.93393	2.21577	1.62
3.64251	3.3939	3.1537	3.25868	3.30736	3.20081	3.53916	2.40644	3.64381	2.59
4.14168	6.52154	3.61189	4.00539	3.18103	3.82703	3.54511	3.27492	3.39199	3.75
3.35703	3.49422	3.64506	3.33577	4.04461	3.34399	3.3092	3.37033	3.60708	3.36
2.08383	2.21985	2.36583	2.30012	2.27068	2.33596	2.13866	2.94244	2.08315	2.77
1.03185	0.977313	0.975466	0.909257	1.01583	1.02983	1.04828	1.02536	1.04964	0.998
4.18099	4.24898	3.89073	4.12976	4.27628	4.15351	3.04616	3.76585	4.14294	4.25
4.24869	4.42423	4.43092	4.71261	4.29623	4.25452	4.20293	4.26759	4.19927	4.35
4.46929	4.34825	4.23237	4.28288	4.30637	4.25501	4.41883	3.8785	4.46993	3.96

what I got :

4.4438	3.89979	4.41748	4.05203	4.69229	4.51991	4.65332	4.68407
2.00889	1.38713	1.3664	0.636298	1.82438	1.98555	2.19986	1.93393
3.64251	3.3939	3.1537	3.25868	3.30736	3.20081	3.53916	2.40644
4.14168	6.52154	3.61189	4.00539	3.18103	3.82703	3.54511	3.27492
3.35703	3.49422	3.64506	3.33577	4.04461	3.34399	3.3092	3.37033
2.08383	2.21985	2.36583	2.30012	2.27068	2.33596	2.13866	2.94244
1.03185	0.977313	0.975466	0.909257	1.01583	1.02983	1.04828	1.02536
4.18099	4.24898	3.89073	4.12976	4.27628	4.15351	3.04616	3.76585
4.24869	4.42423	4.43092	4.71261	4.29623	4.25452	4.20293	4.26759
4.46929	4.34825	4.23237	4.28288	4.30637	4.25501	4.41883	3.8785

Infer the simulated data

Setting up the optimization problem

Ok, now let's work backwards. Assume that the only information we have are:

- the expression values
- the binaryCS matrix **With Signs
- the binaryTFA matrix
- the model

```
simulatedExpressionMatrix // Dimensions
signedBinaryCS = Sign[simulatedControlStrength];
signedBinaryCS // Dimensions
binaryTFA // Dimensions
```

```
{78, 12}
```

```
{78, 9}
```

```
{9, 12}
```

Can you find the simulated data values?
First thing you should do is read over the documentation for NMinimize, then take a look at this example:

```
(*TFs regulating gene 73*)
signedBinaryCS[[73]]
```

```
{0, 0, 0, 0, 0, 0, 1, -1, 0}
```

```
(*activity of TFs 7 and 8 across samples*)
binaryTFA[[{7, 8}]] // MatrixForm
```

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

gene 73 expression across the first 8 samples is modeled:

b[73] + s[73] * [tfa[7, 1] / (tfa[7, 1] + cs[73, 7]) + 1 / (tfa[8, 1] + cs[73, 8])]
b[73] + s[73] * [tfa[7, 2] / (tfa[7, 2] + cs[73, 7]) + 1 / (tfa[8, 2] + cs[73, 8])]
b[73] + s[73] * [tfa[7, 3] / (tfa[7, 3] + cs[73, 7]) + 1 / (tfa[8, 3] + cs[73, 8])]
b[73] + s[73] * [tfa[7, 4] / (tfa[7, 4] + cs[73, 7]) + 1 / (tfa[8, 4] + cs[73, 8])]
b[73] + s[73] * [tfa[7, 5] / (tfa[7, 5] + cs[73, 7]) + 1 / (tfa[8, 5] + cs[73, 8])]
b[73] + s[73] * [tfa[7, 6] / (tfa[7, 6] + cs[73, 7]) + 1 / (tfa[8, 6] + cs[73, 8])]
b[73] + s[73] * [0 / (0 + cs[73, 7]) + 1 / (tfa[8, 7] + cs[73, 8])]
b[73] + s[73] * [tfa[7, 8] / (tfa[7, 8] + cs[73, 7]) + 1 / (0 + cs[73, 8])]

where :
b[i] is baseline of gene i

s[i] is the scaling factor of gene i
tfa[j, k] is the activity of TF j in sample k
cs[i, j] is the control strength on gene i by TF j

We can create an expression to minimize the error for predicting gene 73 expression across the first 8 samples:

```
toMinimize =
 (b[73] + s[73] * (tfa[7, 1] / (tfa[7, 1] + cs[73, 7]) + 1 / (tfa[8, 1] + cs[73, 8])) -
     simulatedExpressionMatrix[[73, 1]]) ^2 +
  (b[73] + s[73] * (tfa[7, 2] / (tfa[7, 2] + cs[73, 7]) + 1 / (tfa[8, 2] + cs[73, 8])) -
     simulatedExpressionMatrix[[73, 2]]) ^2 +
  (b[73] + s[73] * (tfa[7, 3] / (tfa[7, 3] + cs[73, 7]) + 1 / (tfa[8, 3] + cs[73, 8])) -
     simulatedExpressionMatrix[[73, 3]]) ^2 +
  (b[73] + s[73] * (tfa[7, 4] / (tfa[7, 4] + cs[73, 7]) + 1 / (tfa[8, 4] + cs[73, 8])) -
     simulatedExpressionMatrix[[73, 4]]) ^2 +
  (b[73] + s[73] * (tfa[7, 5] / (tfa[7, 5] + cs[73, 7]) + 1 / (tfa[8, 5] + cs[73, 8])) -
     simulatedExpressionMatrix[[73, 5]]) ^2 +
  (b[73] + s[73] * (tfa[7, 6] / (tfa[7, 6] + cs[73, 7]) + 1 / (tfa[8, 6] + cs[73, 8])) -
     simulatedExpressionMatrix[[73, 6]]) ^2 +
  (b[73] + s[73] * ((*tfa[7,7]*)0 / ((*tfa[7,7]*)0 + cs[73, 7]) +
        1 / (tfa[8, 7] + cs[73, 8])) - simulatedExpressionMatrix[[73, 7]]) ^2 +
  (b[73] + s[73] * (tfa[7, 8] / (tfa[7, 8] + cs[73, 7]) +
        1 / ((*tfa[8,8]*)0 + cs[73, 8])) - simulatedExpressionMatrix[[73, 8]]) ^2
```

$$\left(-5.94564 + b[73] + s[73]\left(\frac{1}{cs[73,8]} + \frac{tfa[7,8]}{cs[73,7] + tfa[7,8]}\right)\right)^2 +$$
$$\left(-5.72998 + b[73] + s[73]\left(\frac{tfa[7,1]}{cs[73,7] + tfa[7,1]} + \frac{1}{cs[73,8] + tfa[8,1]}\right)\right)^2 +$$
$$\left(-5.8526 + b[73] + s[73]\left(\frac{tfa[7,2]}{cs[73,7] + tfa[7,2]} + \frac{1}{cs[73,8] + tfa[8,2]}\right)\right)^2 +$$
$$\left(-5.72198 + b[73] + s[73]\left(\frac{tfa[7,3]}{cs[73,7] + tfa[7,3]} + \frac{1}{cs[73,8] + tfa[8,3]}\right)\right)^2 +$$
$$\left(-5.84158 + b[73] + s[73]\left(\frac{tfa[7,4]}{cs[73,7] + tfa[7,4]} + \frac{1}{cs[73,8] + tfa[8,4]}\right)\right)^2 +$$
$$\left(-5.89786 + b[73] + s[73]\left(\frac{tfa[7,5]}{cs[73,7] + tfa[7,5]} + \frac{1}{cs[73,8] + tfa[8,5]}\right)\right)^2 +$$
$$\left(-5.87716 + b[73] + s[73]\left(\frac{tfa[7,6]}{cs[73,7] + tfa[7,6]} + \frac{1}{cs[73,8] + tfa[8,6]}\right)\right)^2 +$$
$$\left(-3.38528 + b[73] + \frac{s[73]}{cs[73,8] + tfa[8,7]}\right)^2$$

We can test that the error expression is correct by defining replacement rules for the params:

```
activityParams = Table[tfa[k, j], {k, numTFs}, {j, numSamples}];
(*table/matrix of activity params*)
controlStrengthParams = Table[cs[i, k], {i, numGenes}, {k, numTFs}];
activityParams // MatrixForm
baselineParams = Table[b[i], {i, numGenes}];
(*table/vector of baseline params*)
scalingParams = Table[s[i], {i, numGenes}];
```

tfa[1, 1]	tfa[1, 2]	tfa[1, 3]	tfa[1, 4]	tfa[1, 5]	tfa[1, 6]	tfa[1, 7]	tfa[1, 8]	tfa[
tfa[2, 1]	tfa[2, 2]	tfa[2, 3]	tfa[2, 4]	tfa[2, 5]	tfa[2, 6]	tfa[2, 7]	tfa[2, 8]	tfa[
tfa[3, 1]	tfa[3, 2]	tfa[3, 3]	tfa[3, 4]	tfa[3, 5]	tfa[3, 6]	tfa[3, 7]	tfa[3, 8]	tfa[
tfa[4, 1]	tfa[4, 2]	tfa[4, 3]	tfa[4, 4]	tfa[4, 5]	tfa[4, 6]	tfa[4, 7]	tfa[4, 8]	tfa[
tfa[5, 1]	tfa[5, 2]	tfa[5, 3]	tfa[5, 4]	tfa[5, 5]	tfa[5, 6]	tfa[5, 7]	tfa[5, 8]	tfa[
tfa[6, 1]	tfa[6, 2]	tfa[6, 3]	tfa[6, 4]	tfa[6, 5]	tfa[6, 6]	tfa[6, 7]	tfa[6, 8]	tfa[
tfa[7, 1]	tfa[7, 2]	tfa[7, 3]	tfa[7, 4]	tfa[7, 5]	tfa[7, 6]	tfa[7, 7]	tfa[7, 8]	tfa[
tfa[8, 1]	tfa[8, 2]	tfa[8, 3]	tfa[8, 4]	tfa[8, 5]	tfa[8, 6]	tfa[8, 7]	tfa[8, 8]	tfa[
tfa[9, 1]	tfa[9, 2]	tfa[9, 3]	tfa[9, 4]	tfa[9, 5]	tfa[9, 6]	tfa[9, 7]	tfa[9, 8]	tfa[

```
(*define replacement rules between params and simulated values*)
activityParamRules =
  MapThread[#1 → #2 &, {activityParams, simulatedActivity}, 2];
controlStrengthParamRules = MapThread[#1 → #2 &,
   {controlStrengthParams, Abs[simulatedControlStrength]}, 2];
activityParamRules // MatrixForm
controlStrengthParamRules[[ ;; , 1 ;; 2]] // Transpose // MatrixForm
baselineParamRules = MapThread[#1 → #2 &, {baselineParams, simulatedBaselines}];
scalingParamRules = MapThread[#1 → #2 &, {scalingParams, simulatedScaling}];
```

tfa[1, 1] → 0.	tfa[1, 2] → 1.24117	tfa[1, 3] → 0.12947	tfa[1, 4] → 1.5599	tfa[1
tfa[2, 1] → 0.912078	tfa[2, 2] → 0.	tfa[2, 3] → 1.78076	tfa[2, 4] → 1.0669	tfa[2
tfa[3, 1] → 0.603013	tfa[3, 2] → 3.42316	tfa[3, 3] → 0.	tfa[3, 4] → 4.26511	tfa[3
tfa[4, 1] → 2.43026	tfa[4, 2] → 1.0888	tfa[4, 3] → 1.05238	tfa[4, 4] → 0.	tfa[4
tfa[5, 1] → 3.52297	tfa[5, 2] → 2.09912	tfa[5, 3] → 1.18948	tfa[5, 4] → 3.83606	tf
tfa[6, 1] → 3.73918	tfa[6, 2] → 1.71179	tfa[6, 3] → 3.62245	tfa[6, 4] → 1.68938	tfa[6
tfa[7, 1] → 3.57928	tfa[7, 2] → 3.98276	tfa[7, 3] → 2.21499	tfa[7, 4] → 3.29969	tfa[7
tfa[8, 1] → 4.13876	tfa[8, 2] → 2.78589	tfa[8, 3] → 1.82991	tfa[8, 4] → 2.21474	tfa[8
tfa[9, 1] → 1.5146	tfa[9, 2] → 1.9428	tfa[9, 3] → 0.86989	tfa[9, 4] → 2.15099	tfa[9

cs[1, 1] → 0.	cs[2, 1] → 0.	cs[3, 1] → 0.	cs[4, 1] → 0.	cs[5, 1] → 0.	cs[6,
cs[1, 2] → 1.19191	cs[2, 2] → 0.	cs[3, 2] → 0.	cs[4, 2] → 0.543188	cs[5, 2] → 0.	cs[6,

If we use those rules on the error expression to minimize, we get 0 (or very close, due to precision error)

```
toMinimize /.
 Flatten[{activityParamRules,
   controlStrengthParamRules, baselineParamRules, scalingParamRules}]
```

7.91942×10^{-31}

To use the NMinimize function, we need a list of unknown parameters:

```
params = {b[73], s[73],
  tfa[7, 1], tfa[7, 2], tfa[7, 3], tfa[7, 4], tfa[7, 5], tfa[7, 6], tfa[7, 8],
  tfa[8, 1], tfa[8, 2], tfa[8, 3], tfa[8, 4], tfa[8, 5], tfa[8, 6], tfa[8, 7],
  cs[73, 7], cs[73, 8]}
```

```
{b[73], s[73], tfa[7, 1], tfa[7, 2], tfa[7, 3], tfa[7, 4],
 tfa[7, 5], tfa[7, 6], tfa[7, 8], tfa[8, 1], tfa[8, 2], tfa[8, 3],
 tfa[8, 4], tfa[8, 5], tfa[8, 6], tfa[8, 7], cs[73, 7], cs[73, 8]}
```

And we should constrain all the values to be positive

```
constraints = Map[# > 0 &, params]
```

```
{b[73] > 0, s[73] > 0, tfa[7, 1] > 0, tfa[7, 2] > 0, tfa[7, 3] > 0, tfa[7, 4] > 0,
 tfa[7, 5] > 0, tfa[7, 6] > 0, tfa[7, 8] > 0, tfa[8, 1] > 0, tfa[8, 2] > 0, tfa[8, 3] > 0,
 tfa[8, 4] > 0, tfa[8, 5] > 0, tfa[8, 6] > 0, tfa[8, 7] > 0, cs[73, 7] > 0, cs[73, 8] > 0}
```

Can we minimize?

```
{error, paramRules} = NMinimize[{toMinimize, constraints}, params]
```

```
{1.62086 × 10^-14, {b[73] → 2.41537, s[73] → 3.13434, tfa[7, 1] → 1.35124,
  tfa[7, 2] → 1.09767, tfa[7, 3] → 0.608921, tfa[7, 4] → 0.268794, tfa[7, 5] → 2.27149,
  tfa[7, 6] → 1.59422, tfa[7, 8] → 0.233867, tfa[8, 1] → 0.70335, tfa[8, 2] → 0.47384,
  tfa[8, 3] → 0.295889, tfa[8, 4] → 0.0538201, tfa[8, 5] → 0.954789,
  tfa[8, 6] → 0.686959, tfa[8, 7] → 2.22326, cs[73, 7] → 1.50372, cs[73, 8] → 1.00834}}
```

Is this correct?

```
{Prepend[params, "params"],
  Prepend[paramRules[[ ;; , 1]] /. Flatten[{activityParamRules,
      controlStrengthParamRules, baselineParamRules, scalingParamRules}],
   "simulated"], Prepend[paramRules[[ ;; , 2]], "inferred"]} // MatrixForm
```

params	b[73]	s[73]	tfa[7, 1]	tfa[7, 2]	tfa[7, 3]	tfa[7, 4]	tfa[7, 5]	tfa[7,
simulated	2.96582	2.76681	3.57928	3.98276	2.21499	3.29969	4.15633	3.4268
inferred	2.41537	3.13434	1.35124	1.09767	0.608921	0.268794	2.27149	1.5942

```
(*square difference b/w params*)Total[
 ((paramRules[[ ;; , 1]] /. Flatten[{activityParamRules, controlStrengthParamRules,
       baselineParamRules, scalingParamRules}]) - paramRules[[ ;; , 2]]) ^2]
```

```
69.5455
```

Some are pretty close, but most are pretty off. How does the error compare between the true simulation values, and the inferred values?

```
toMinimize /.
 Flatten[{activityParamRules,
   controlStrengthParamRules, baselineParamRules, scalingParamRules}]
```

```
7.91942 × 10^-31
```

```
error
```

```
1.62086 × 10^-14
```

So the real values would've have given us a better solution, but the optimizer didn't find it, probably because they found a solution that would generally be considered good enough, or it converged on a local minimum, or it hit some artificial threshold like the max number of iterations.
Let's first try increasing the max iterations (default is 100)

```
{error, paramRules} =
 NMinimize[{toMinimize, constraints}, params, MaxIterations → 1000]
```

```
{1.69095 × 10^-14, {b[73] → 2.30105, s[73] → 2.91542, tfa[7, 1] → 2.96017,
  tfa[7, 2] → 0.638007, tfa[7, 3] → 0.946642, tfa[7, 4] → 0.72465, tfa[7, 5] → 2.91179,
  tfa[7, 6] → 2.75161, tfa[7, 8] → 0.662296, tfa[8, 1] → 1.63221, tfa[8, 2] → 0.0498405,
  tfa[8, 3] → 0.449165, tfa[8, 4] → 0.133679, tfa[8, 5] → 1.15139, tfa[8, 6] → 1.14973,
  tfa[8, 7] → 1.23193, cs[73, 7] → 0.512467, cs[73, 8] → 1.45701}}
```

```
{Prepend[params, "params"],
  Prepend[paramRules[[ ;; , 1]] /. Flatten[{activityParamRules,
      controlStrengthParamRules, baselineParamRules, scalingParamRules}],
   "simulated"], Prepend[paramRules[[ ;; , 2]], "inferred"]} // MatrixForm
```

params	b[73]	s[73]	tfa[7, 1]	tfa[7, 2]	tfa[7, 3]	tfa[7, 4]	tfa[7, 5]	tfa[7,
simulated	2.96582	2.76681	3.57928	3.98276	2.21499	3.29969	4.15633	3.426
inferred	2.30105	2.91542	2.96017	0.638007	0.946642	0.72465	2.91179	2.751

```
(*square difference b/w params*)Total[
 ((paramRules[[ ;; , 1]] /. Flatten[{activityParamRules, controlStrengthParamRules,
        baselineParamRules, scalingParamRules}]) - paramRules[[ ;; , 2]]) ^2]
```

```
53.6514
```

There's an improvement, but not by much. Let's instead try using more data.

NMinimize

Multiple solutions check tfa[1,2]

Multiple solutions check TF 1

Sensitivity check tfa[1,2]

Sensitivity check tfa[2,1]

Sensitivity check cs[1,2]