Note: requires **seesawSimulatorDM.wl** to be loaded.

Functions

DataToList function takes an imported data file and generates a data list in the format of {time, raw fluorescence}.
`data` is imported from a csv data file.
`StartRow` is the number of the first row containing time and fluorescence data points.
`TimeFormat` = 0 indicates a single number in minutes.
`TimeFormat` = 1 indicates three numbers in hours:minutes:seconds.
`TimeColumn` = 0 indicates a single column of time followed by all colunms of fluorescence.
`TimeColumn` = 1 indicates alternating columns of time and fluorescence.

```
DataToList[data_, StartRow_, TimeFormat_, TimeColumn_] := Module[{time}, Table[
   If[TimeFormat == 0,
    time = data[[i, If[TimeColumn == 0, 1, j - 1]]] / 60,
    time = Sum[ToExpression[
        StringSplit[data[[i, If[TimeColumn == 0, 1, j - 1]]], ":"][[n]]] * 60^ (1 - n),
      {n, 1, 3}]];
   {time, data[[i, j]]},
   {j, 2, Length[data[[1]]], If[TimeColumn == 0, 1, 2]},
   {i, StartRow, Length[data]}]]
```

NormalizeDataList function takes a list of raw fluorescence data and normalizes it to a list of output concentrations between 0× and 1×, using the raw fluorescence of the first data point of the `OFFtrajectory` as minimum (0×) and the average raw fluorescence of the last five data points of the `ONtrajectory` as maximum (1×).
`datalist` is a list of trajectories, each containing a list of data points in the format of {time, raw fluorescence}.
`OFFtrajectory` is a trajectory which corresponds to output being OFF.
`ONtrajectory` is a trajectory which corresponds to output being ON.

```
NormalizeDataList[datalist_, OFFtrajectory_, ONtrajectory_] := Module[{min, max},
  min = datalist[[OFFtrajectory, 1, 2]];
  max = Mean[Table[datalist[[ONtrajectory, i, 2]],
     {i, Length[datalist[[ONtrajectory]]] - 4, Length[datalist[[ONtrajectory]]]}]];
  Table[{datalist[[i, j, 1]], (datalist[[i, j, 2]] - min) / (max - min)},
   {i, 1, Length[datalist]}, {j, 1, Length[datalist[[1]]]}]]
```

PlotRowData function takes a list of raw fluorescence data and plots multiple kinetics trajectories over time.
`datalist` is a list of trajectories, each containing a list of data points in the format of {time, raw fluorescence}.
`input` is a list of input values, relative to a standard concentration 1×.
`time` defines the range of time plotted, starting from the first data point. The unit is hours. The default is all data points.
`InputLabel` is a label for the legend of inputs.
`CircuitLabel` is a label for the plot. It can be the function of the circuit. The default is none.

```
PlotRowData[datalist_, input_, time_: All, InputLabel_: "", CircuitLabel_: ""] :=
 ListPlot[datalist, PlotLabel → Style[CircuitLabel, 20],
  Frame -> True, FrameLabel -> {"Time (hours)", "Fluorescence (a.u.)"},
  PlotStyle → Table[{Thickness[0.01], ColorData["Rainbow"][i]},
    {i, If[Length[datalist] < 8, 0.12, 0], 0.96, 0.96 / Length[datalist]}],
  PlotLegends →
   SwatchLegend[Automatic, input, LegendLabel → InputLabel, LegendMarkerSize → 14],
  LabelStyle -> Directive[Gray, FontSize → 20, FontFamily -> "Helvetica"],
  GridLines → Automatic,
  PlotRange → {{0, time}, All}, AspectRatio -> 1 / 1.3, ImageSize → 400]
```

PlotNormalizedData function takes a list of normalized data and plots multiple kinetics trajectories over time.
`datalist` is a list of trajectories, each containing a list of data points in the format of {time, output concentration}.
`input` is a list of input values, relative to a standard concentration 1×.
`time` defines the range of time plotted, starting from the first data point. The unit is hours. The default is all data points.
`InputLabel` is a label for the legend of inputs.
`CircuitLabel` is a label for the plot. It can be the function of the circuit. The default is none.

```
PlotNormalizedData[datalist_, input_,
  time_: All, InputLabel_: "", CircuitLabel_: ""] :=
 ListPlot[datalist, PlotLabel → Style[CircuitLabel, 20],
  Frame -> True, FrameLabel -> {"Time (hours)", "Output"},
  PlotStyle → Table[{Thickness[0.01], ColorData["Rainbow"][i]},
    {i, If[Length[datalist] < 8, 0.12, 0], 0.96, 0.96 / Length[datalist]}],
  PlotLegends →
   SwatchLegend[Automatic, input, LegendLabel → InputLabel, LegendMarkerSize → 14],
  LabelStyle -> Directive[Gray, FontSize → 20, FontFamily -> "Helvetica"],
  GridLines → Automatic,
  PlotRange → {{0, time}, {-0.05, 1.05}}, AspectRatio -> 1 / 1.3, ImageSize → 400]
```

AverageTrajectory function takes a list of data and calculates the average value of a specific trajectory.
`StartDataPoint` defines the range of values averaged, ending with the last data point. The default is to start from the first data point. To calculate the average of the last `x` data points, set `StartDataPoint` to Length[`datalist`[[1]]-`x+1`], assuming all trajectories have the same length, or Length[`datalist`[[`trajectory`]]-`x+1`] for a specific trajectory.
`digit` defines the precision of the average value, rounding to the nearest multiple of `digit`. The default is 0.01.

```
AverageTrajectory[datalist_, trajectory_, StartDataPoint_: 1, digit_: 0.01] :=
 Round[Mean[Table[datalist[[trajectory, i, 2]],
    {i, StartDataPoint, Length[datalist[[trajectory]]]}]], digit]
```

TrajectoryValue function takes a list of data and calculates the value of a specific trajectory when a reference trajectory (`REFtrajectory`) reaches a reference value (`REFvalue`).
`digit` defines the precision of the average value, rounding to the nearest multiple of `digit`. The default is 0.01.

```
TrajectoryValue[datalist_, trajectory_, REFtrajectory_, REFvalue_, digit_: 0.01] :=
 Module[{i}, i = 1;
  While[Mean[Table[datalist[[REFtrajectory, j, 2]], {j, i, i + 4}]] < REFvalue, i++];
  Round[Mean[Table[datalist[[trajectory, j, 2]], {j, i, i + 4}]], digit]]
```

SIMSigRest function simulates a signal restoration circuit.
`input` is a list of input values, relative to a standard concentration 1×.
`threshold` is a threshold value, relative to a standard concentration 1×.
`time` defines the range of time simulated, starting from 0. The unit is hours.

```
SIMSigRest[input_, threshold_, time_] := Table[
   SignalRest = {
     seesaw[5, {53}, {6, f}],
     reporter[6, 5],

     concd[th[w[53, 5], 5], threshold * c],
     concd[g[5, w[5, 6]], 1 * c],
     concd[w[5, f], 2 * c],
     concd[w[53, 5], x * c]
    };
   sol = SimulateRxnsys[SignalRest, time * 60 * 60];
   Fluor[6][t * 60 * 60] / maxSignal[srb] / c /. sol,
{x, input}];
```

PlotSim function plots a simulation including multiple trajectories.
`SIMcircuit` is a simulation.
`input` is a list of input values, relative to a standard concentration 1×.
`time` defines the range of time plotted, starting from 0. The unit is hours.
`InputLabel` is a label for the legend of inputs. The default is none.
`CircuitLabel` is a label for the plot. It can be the function of the circuit. The default is none.

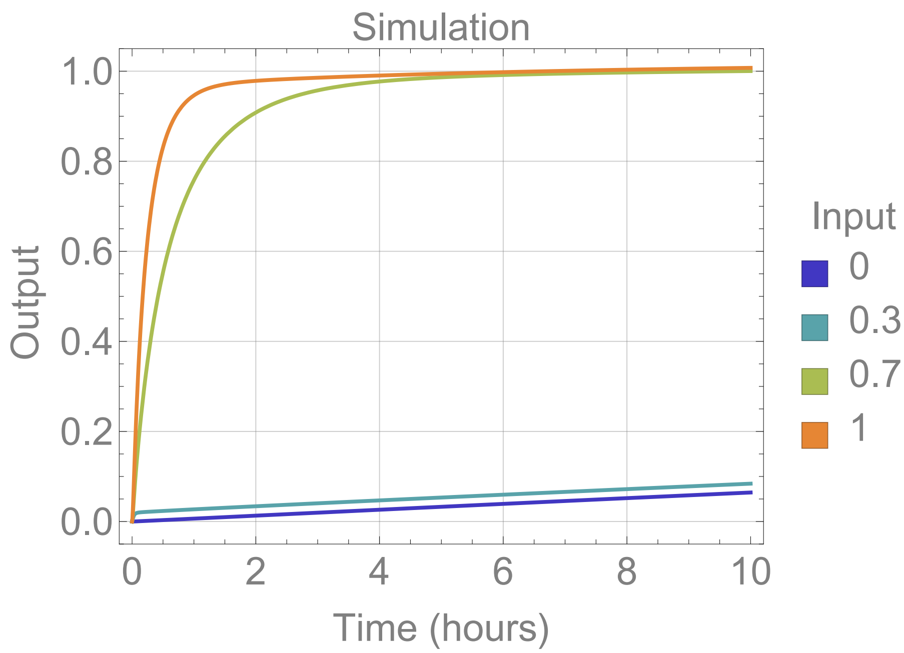
```
PlotSim[SIMcircuit_, input_, time_, InputLabel_: "", CircuitLabel_: ""] :=
 Plot[SIMcircuit, {t, 0, time},
  PlotLabel → Style[CircuitLabel, 20],
  Frame -> True, FrameLabel -> {"Time (hours)", "Output"},
  PlotStyle → Table[{Thick, ColorData["Rainbow"][i]},
    {i, If[Length[input] < 8, 0.12, 0], 0.96, 0.96 / Length[input]}],
  PlotLegends → SwatchLegend[Automatic, input,
    LegendLabel → InputLabel, LegendMarkerSize → 14],
  LabelStyle -> Directive[Gray, FontSize → 20, FontFamily -> "Helvetica"],
  GridLines → Automatic,
  PlotRange → {All, {-0.05, 1.05}}, AspectRatio -> 1 / 1.3, ImageSize → 400]
```

A signal restoration circuit / threshold calibration

Simulation

```
input = {0, 0.3, 0.7, 1};
threshold = 0.5;
time = 10; (* unit: hours *)
```

```
PlotSim[SIMSigRest[input, threshold, time], input, time, "Input", "Simulation"]
```

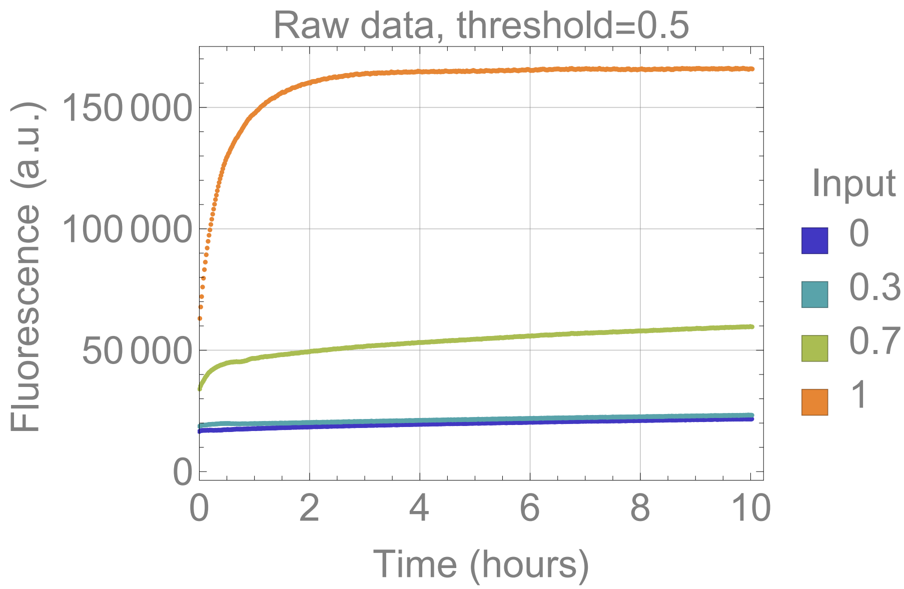


Experiment

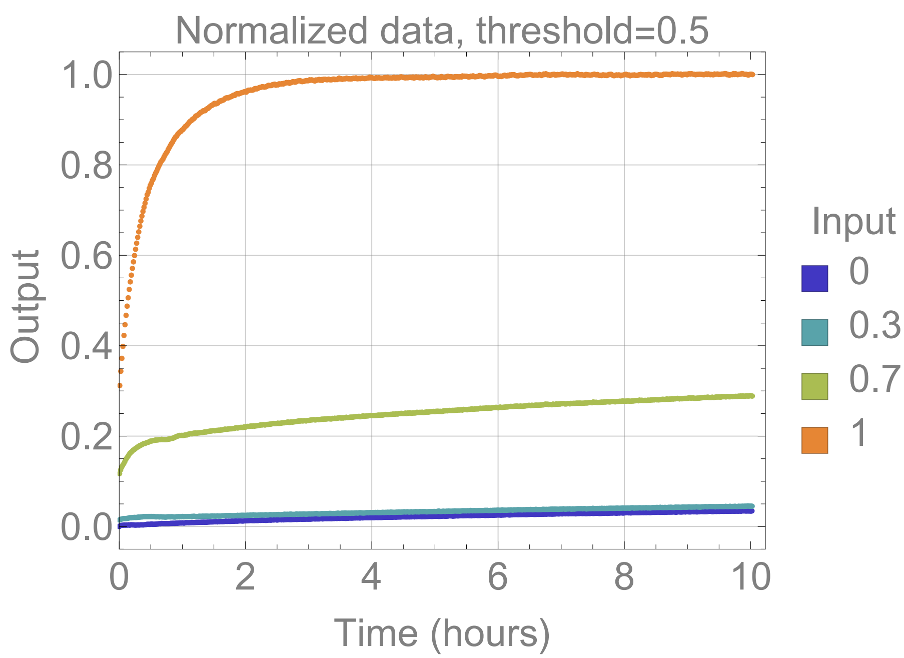
```
SetDirectory[NotebookDirectory[]];
SignalRestorationData = Import["Signal_restoration.csv"];
```

```
input = {0, 0.3, 0.7, 1};
threshold = 0.5;
```

```
PlotRowData[DataToList[SignalRestorationData, 3, 1, 1], input, All,
 "Input", Row[{"Raw data, threshold=", threshold}]]
```




```
PlotNormalizedData[
 NormalizeDataList[DataToList[SignalRestorationData, 3, 1, 1], 1, 4],
 input, All, "Input", Row[{"Normalized data, threshold=", threshold}]]
```

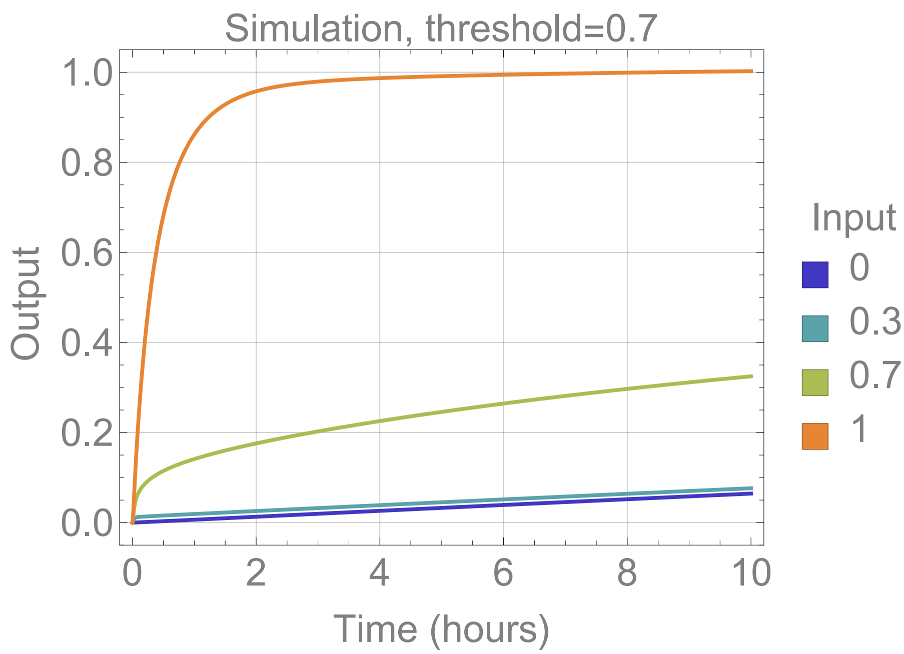


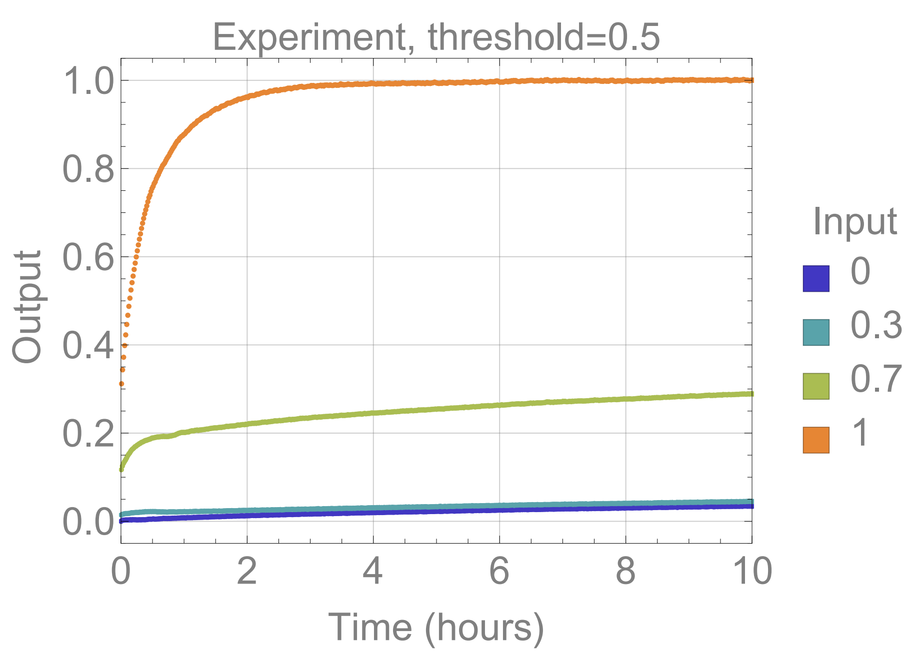
Threshold to signal ratio (β / α)

Change the value of threshold in simulation (**`thresholdSim`**) until the trajectories qualitatively agree with the experimental data.

```
input = {0, 0.3, 0.7, 1};
thresholdExp = 0.5;
thresholdSim = 0.7;
```

```
Row[{PlotSim[SIMSigRest[input, thresholdSim, time], input, time, "Input",
   Row[{"Simulation, threshold=", thresholdSim}]], PlotNormalizedData[
   NormalizeDataList[DataToList[SignalRestorationData, 3, 1, 1], 1, 4],
   input, time, "Input", Row[{"Experiment, threshold=", thresholdExp}]]}]
```





The threshold to signal ratio (β/α) can be estimated as effective threshold (**thresholdSim**) / nominal threshold (**thresholdExp**):

```
ThresholdToSignal = thresholdSim / thresholdExp
```

```
1.4
```

For threshold to signal ratio > 1.2, adjust the nominal threshold in a logic gate

The lower bound of a two-input OR gate:

```
Floor[0.4 / ThresholdToSignal, 0.01]
```

```
0.28
```

The upper bound of a two-input OR gate:

```
Floor[0.8 / ThresholdToSignal, 0.01]
```

```
0.57
```

The lower bound of a two-input AND gate:

```
Floor[1.2 / ThresholdToSignal, 0.01]
```

```
0.85
```

The upper bound of a two-input AND gate:

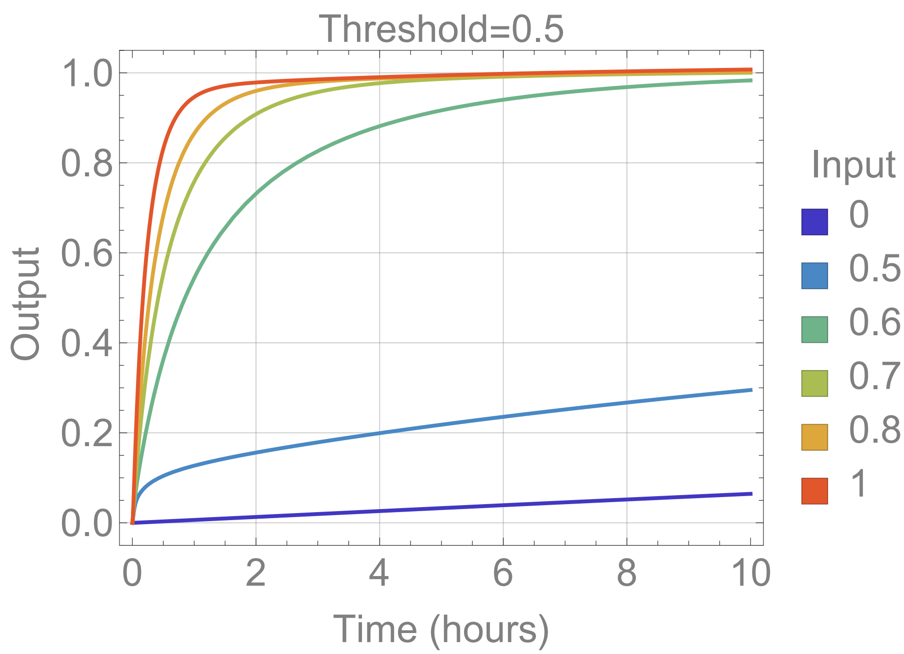
```
Floor[1.6 / ThresholdToSignal, 0.01]
```

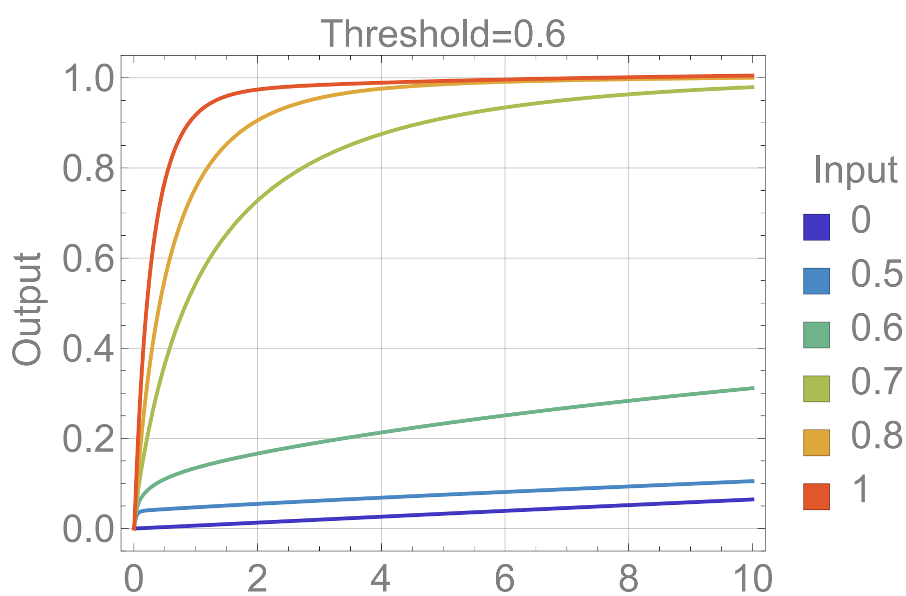
```
1.14
```

A simple rule of thumb is that the effective threshold is roughly the same as the input of a trajectory that falls between an ideal ON state and OFF state, as illustrated in the following simulations.

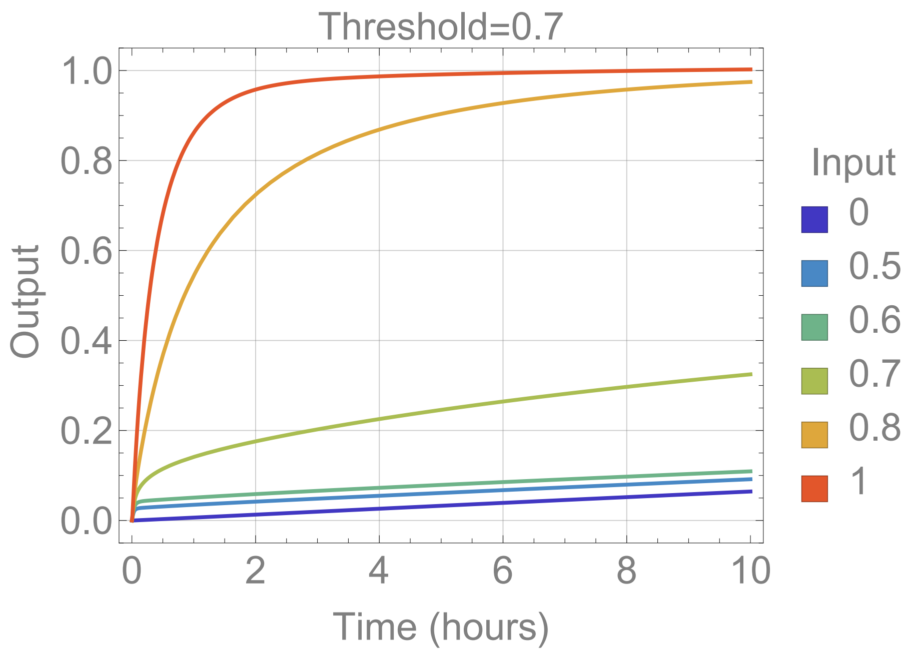
```
input = Flatten[{0, Range[0.5, 0.8, 0.1], 1}, 1];
time = 10; (* unit: hours *)
```

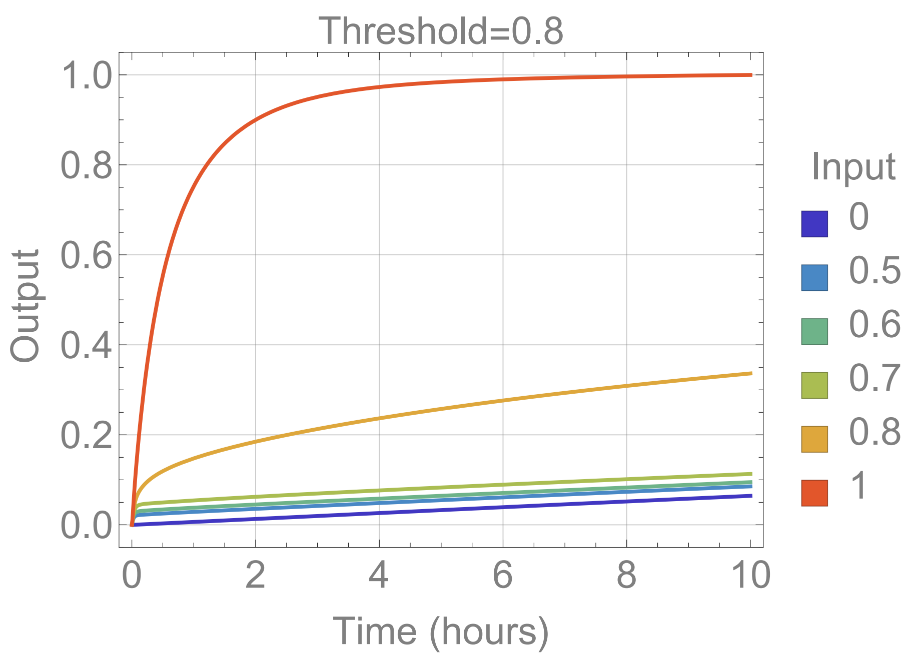
```
Row[Table[PlotSim[SIMSigRest[input, threshold, time], input, time,
   "Input", Row[{"Threshold=", threshold}]], {threshold, 0.5, 0.8, 0.1}]]
```





Time (hours)


Threshold=0.7
Output
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10
Time (hours)
Input
0
0.5
0.6
0.7
0.8
1

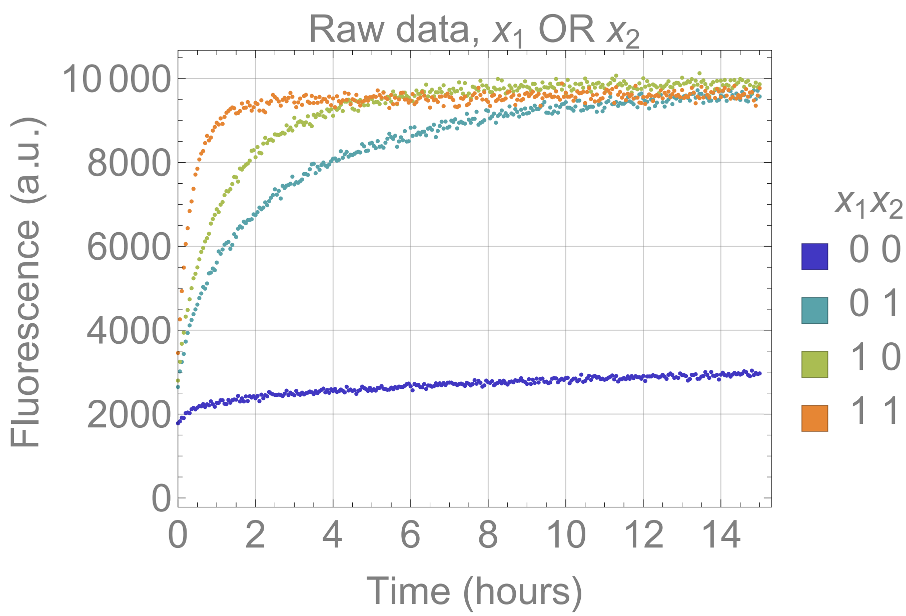

Threshold=0.8
Output
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10
Time (hours)
Input
0
0.5
0.6
0.7
0.8
1

Logic OR

Experiment

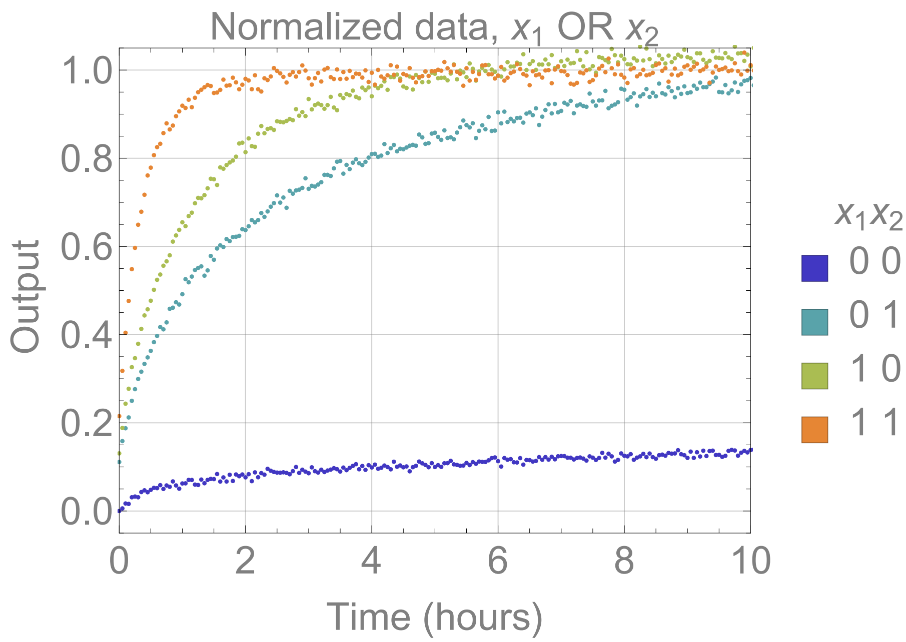
```
SetDirectory[NotebookDirectory[]];
LogicORData = Import["Logic_OR.csv"];
```

```
PlotRowData[DataToList[LogicORData, 1, 0, 1],
 {"0 0", "0 1", "1 0", "1 1"}, All, "    x₁x₂", "Raw data, x₁ OR x₂"]
```



```
NormalizedLogicORData = NormalizeDataList[DataToList[LogicORData, 1, 0, 1], 1, 4];
```

```
PlotNormalizedData[NormalizedLogicORData,
 {"0 0", "0 1", "1 0", "1 1"}, 10, "    x1x2", "Normalized data, x1 OR x2"]
```



ON/OFF seperateion

The OFF trajectory (trajectory 1) is well below 0.2 when the slowest ON trajectory (trajectory 2) reaches 0.8.
Thus it is a good OR gate.

```
TrajectoryValue[NormalizedLogicORData, 1, 2, 0.8]
```

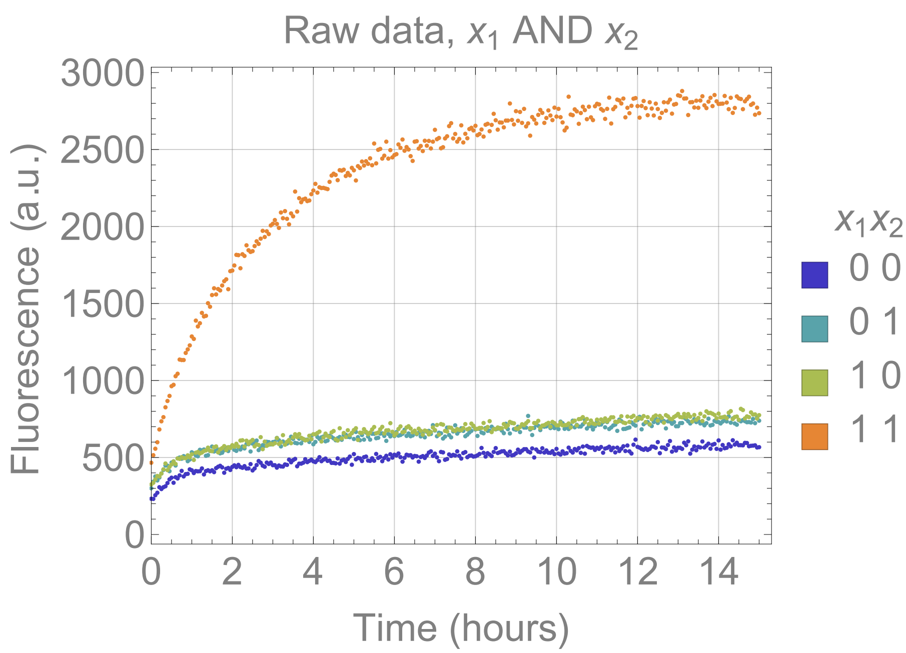
```
0.1
```

Logic AND

Experiment

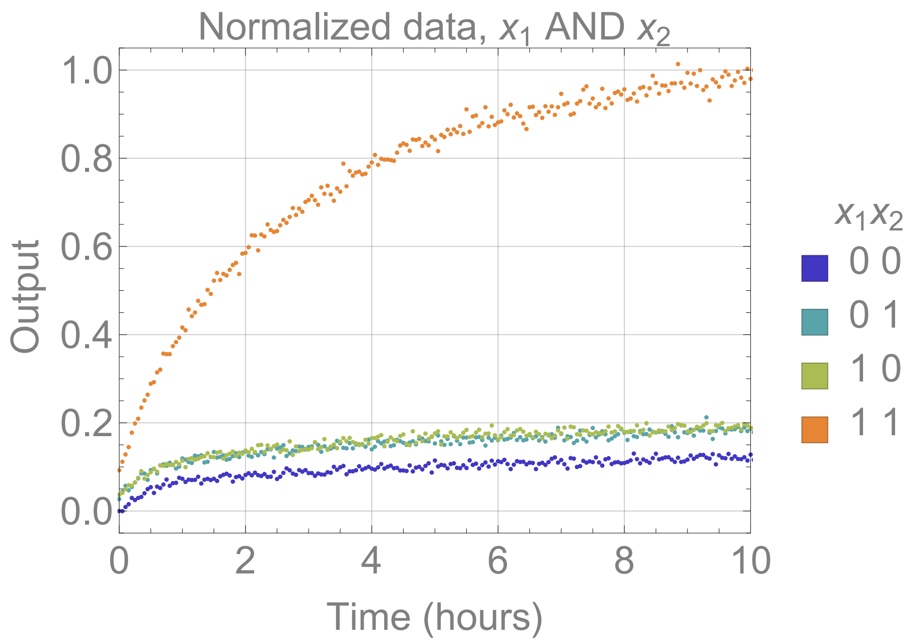
```
SetDirectory[NotebookDirectory[]];
LogicANDData = Import["Logic_AND.csv"];
```

```
PlotRowData[DataToList[LogicANDData, 1, 0, 1],
 {"0 0", "0 1", "1 0", "1 1"}, All, "   x₁x₂", "Raw data, x₁ AND x₂"]
```



```
NormalizedLogicANDData = NormalizeDataList[DataToList[LogicANDData, 1, 0, 1], 1, 4];
```

```
PlotNormalizedData[NormalizedLogicANDData,
 {"0 0", "0 1", "1 0", "1 1"}, 10, "    x1x2", "Normalized data, x1 AND x2"]
```



ON/OFF seperateion

The faster OFF trajectory (trajectory 3) is below 0.2 when the ON trajectory (trajectory 4) reaches 0.8. Thus it is a good AND gate.

```
TrajectoryValue[NormalizedLogicANDData, 3, 4, 0.8]
```

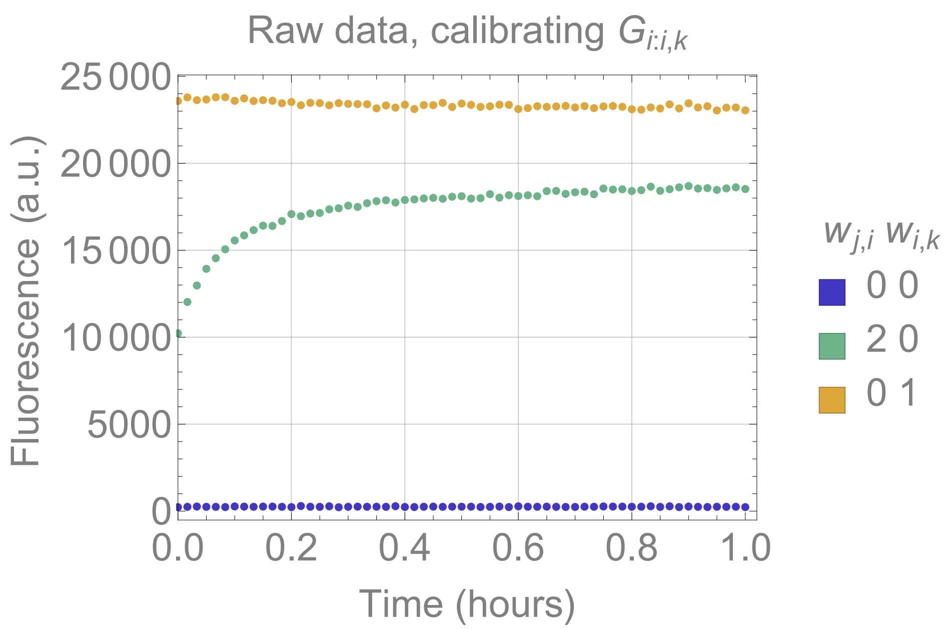
```
0.16
```

Gate calibration

Experiment

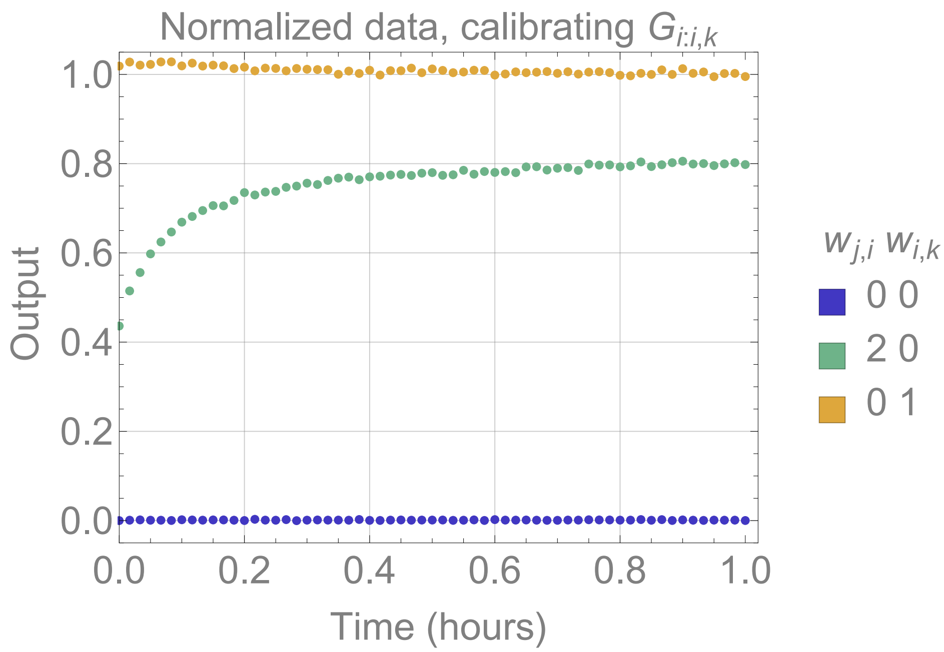
```
SetDirectory[NotebookDirectory[]];
GateCalibrationData = Import["Gate_calibration.csv"];

PlotRowData[DataToList[GateCalibrationData, 2, 1, 0],
 {"0 0", "2 0", "0 1"}, All, "  wj,i wi,k", "Raw data, calibrating Gi:i,k"]
```



```
NormalizedGateCalibrationData =
  NormalizeDataList[DataToList[GateCalibrationData, 2, 1, 0], 1, 3];
```

```
PlotNormalizedData[NormalizedGateCalibrationData, {"0 0", "2 0", "0 1"},
 All, "  wj,i wi,k", "Normalized data, calibrating Gi:i,k"]
```



Gate to signal ratio (γ / α)

The gate to signal ratio (γ/α) can be estimated as fully triggered gate (with $w_{j,i} = 2$) / signal (with $w_{i,k} = 1$):

```
GateToSignal = AverageTrajectory[NormalizedGateCalibrationData,
  2, Length[NormalizedGateCalibrationData[[1]]] - 4]
```

```
0.8
```

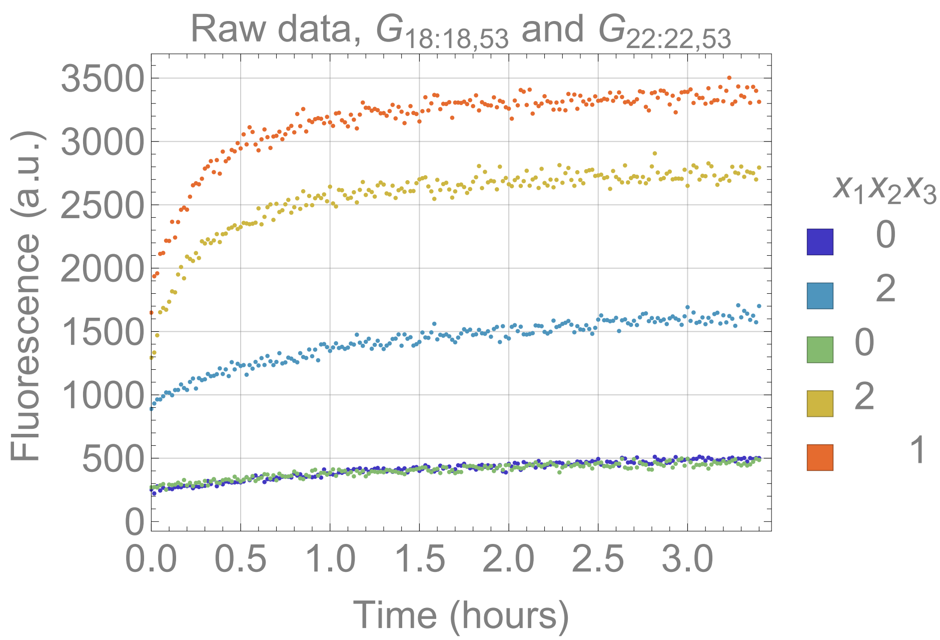
For gate to signal ratio >= 0.8, no need to adjust the nominal gate

Gate outlier

Experiment

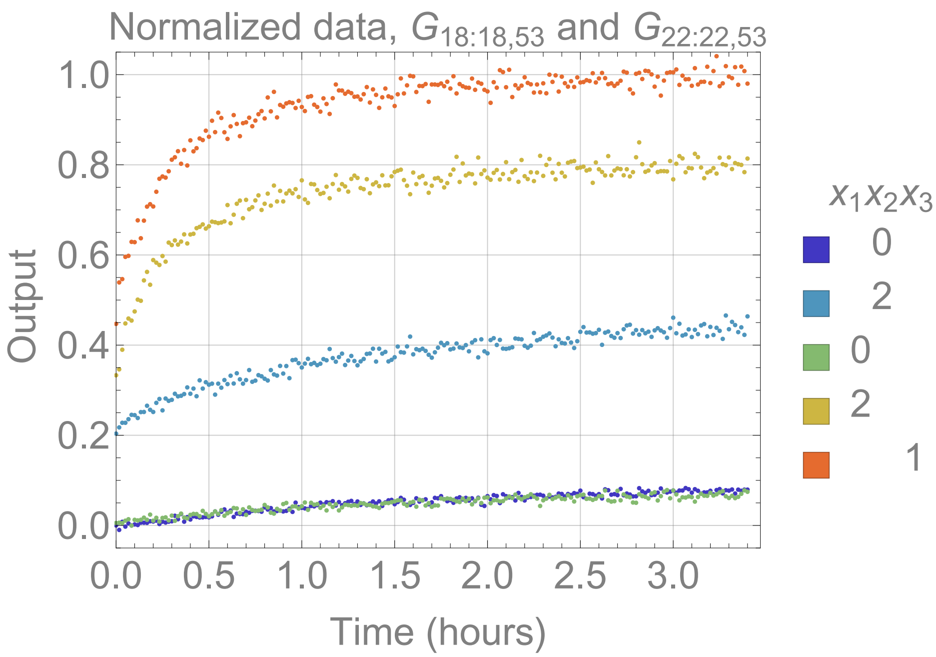
```
SetDirectory[NotebookDirectory[]];
GateOutlierData = Import["Gate_outlier.csv"];
```

```
PlotRowData[DataToList[GateOutlierData, 2, 1, 0], {"   0", "   2", "0", "2", "      1"},
 All, "    x1x2x3", "Raw data, G18:18,53 and G22:22,53"]
```



```
NormalizedGateOutlierData =
  NormalizeDataList[DataToList[GateOutlierData, 2, 1, 0], 1, 5];
```

```
PlotNormalizedData[NormalizedGateOutlierData, {"   0", "   2", "0", "2", "      1"},
 All, "   x₁x₂x₃", "Normalized data, G18:18,53 and G22:22,53"]
```



Gate to signal ratio

The gate to signal ratio (γ/α) of $G_{18:18,53}$ can be estimated as fully triggered gate (with $x_1 = 2$, trajectory 4) / signal (with $x_3 = 1$).
$G_{18:18,53}$ is not an outlier.

```
Gate18ToSignal = AverageTrajectory[NormalizedGateOutlierData,
  4, Length[NormalizedGateOutlierData[[1]]] - 10]
```

```
0.8
```

The gate to signal ratio (γ/α) of $G_{22:22,53}$ can be estimated as fully triggered gate (with $x_2 = 2$, trajectory 2) / signal (with $x_3 = 1$).
$G_{22:22,53}$ is an outlier.

```
Gate22ToSignal = AverageTrajectory[NormalizedGateOutlierData,
  2, Length[NormalizedGateOutlierData[[1]]] - 10]
```

```
0.44
```

For gate to signal ratio < 0.8, adjust the nominal gate

The nominal concentration of $G_{22:22,53}$ can be adjusted as:

```
Round[1 / Gate22ToSignal * GateToSignal, 0.1]
```

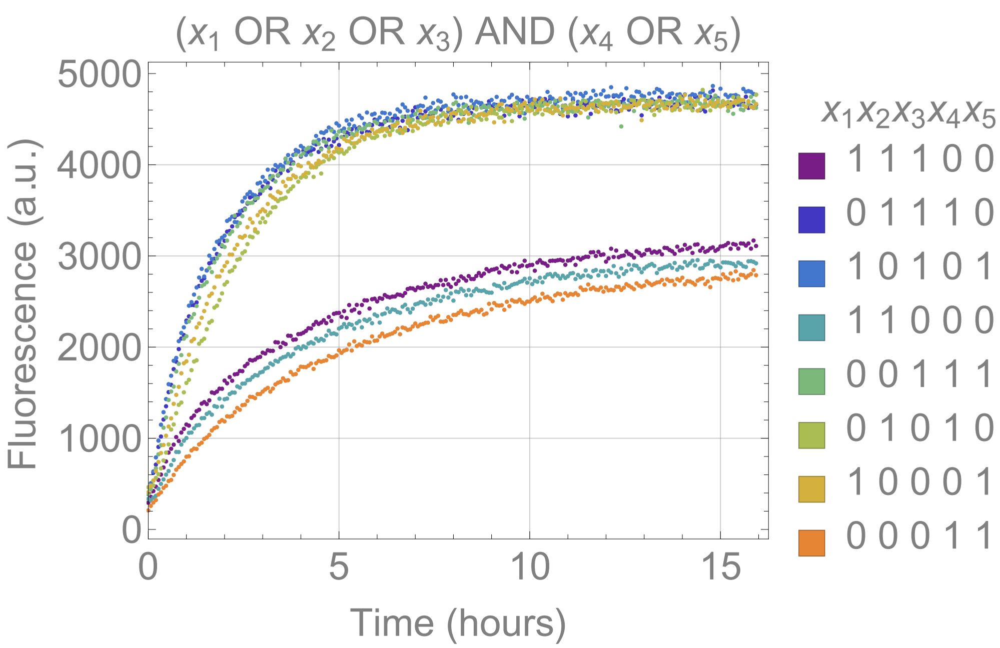
```
1.8
```

A two layer logic circuit that requires tuning

Experiment

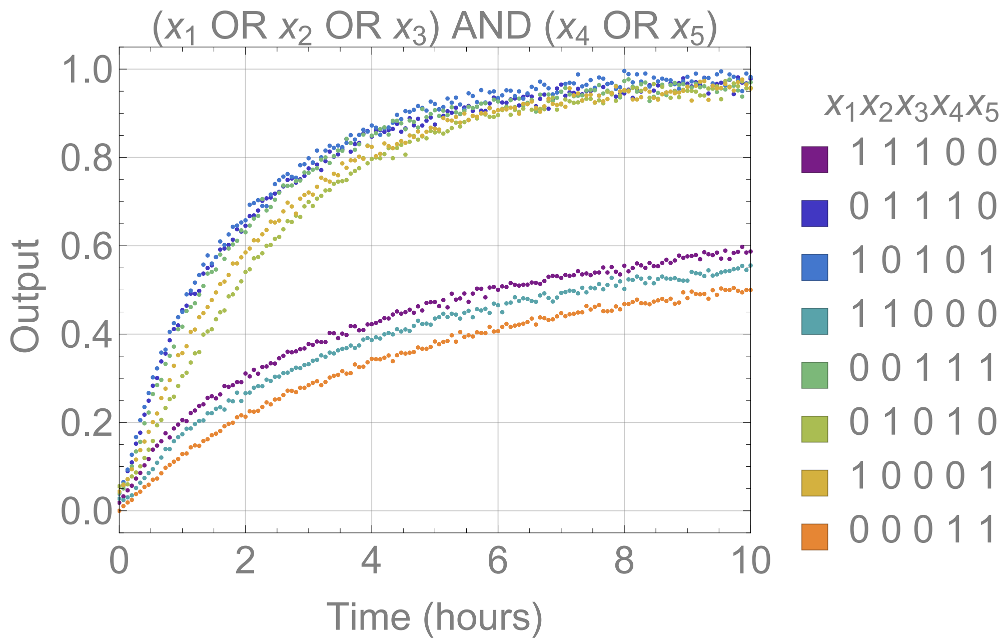
```
SetDirectory[NotebookDirectory[]];
TwoLayerCircuitData = Import["Layer2_circuit_OR_AND_R124.csv"];
```

```
PlotRowData[DataToList[TwoLayerCircuitData, 2, 1, 0],
 {"1 1 1 0 0", "0 1 1 1 0", "1 0 1 0 1", "1 1 0 0 0", "0 0 1 1 1", "0 1 0 1 0",
  "1 0 0 0 1", "0 0 0 1 1"}, All, "  x₁x₂x₃x₄x₅", "(x₁ OR x₂ OR x₃) AND (x₄ OR x₅)"]
```




```
NormalizedTwoLayerCircuitData =
  NormalizeDataList[DataToList[TwoLayerCircuitData, 2, 1, 0], 8, 3];
```

```
PlotNormalizedData[NormalizedTwoLayerCircuitData, {"1 1 1 0 0", "0 1 1 1 0",
  "1 0 1 0 1", "1 1 0 0 0", "0 0 1 1 1", "0 1 0 1 0", "1 0 0 0 1", "0 0 0 1 1"},
 10, "  x1x2x3x4x5", "(x1 OR x2 OR x3) AND (x4 OR x5)"]
```



ON/OFF seperateion

The fastest OFF trajectory (trajectory 1) is above 0.3 when the slowest ON trajectory (trajectory 6) reachs 0.7.
Thus the circuit requires tuning.

```
TrajectoryValue[NormalizedTwoLayerCircuitData, 1, 6, 0.7]
```

0.38

Increase of the nominal threshold ($\delta \times \alpha / \beta$) in the downstream AND gate

The lower bound of δ:

```
TrajectoryValue[NormalizedTwoLayerCircuitData, 1, 6, 0.7] - 0.3
```

0.08

The upper bound of δ:

```
TrajectoryValue[NormalizedTwoLayerCircuitData, 1, 6, 0.9] - 0.1
```

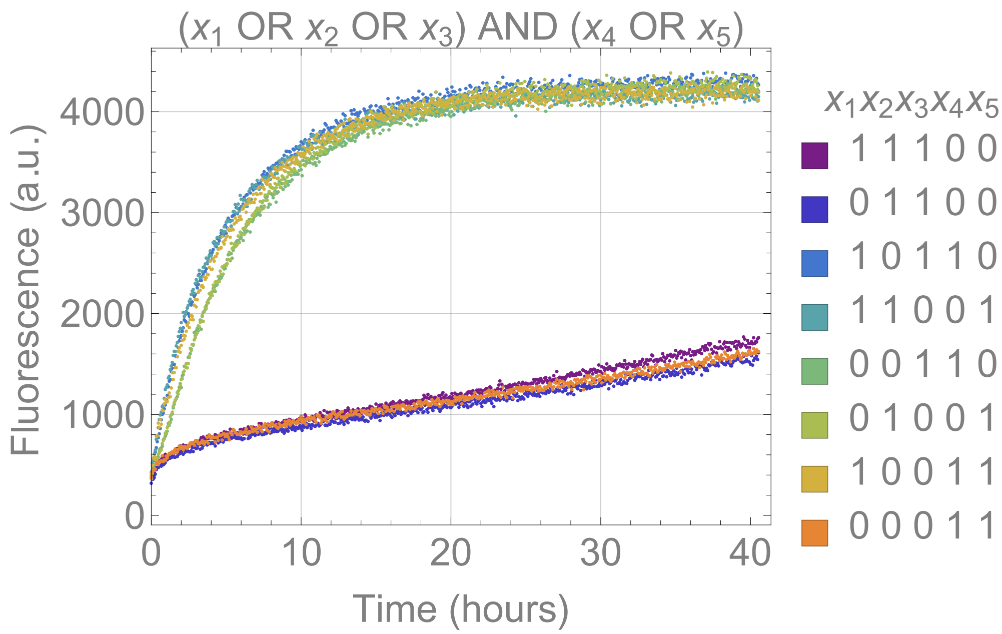
0.41

A two layer logic circuit with good ON/OFF seperation

Experiment

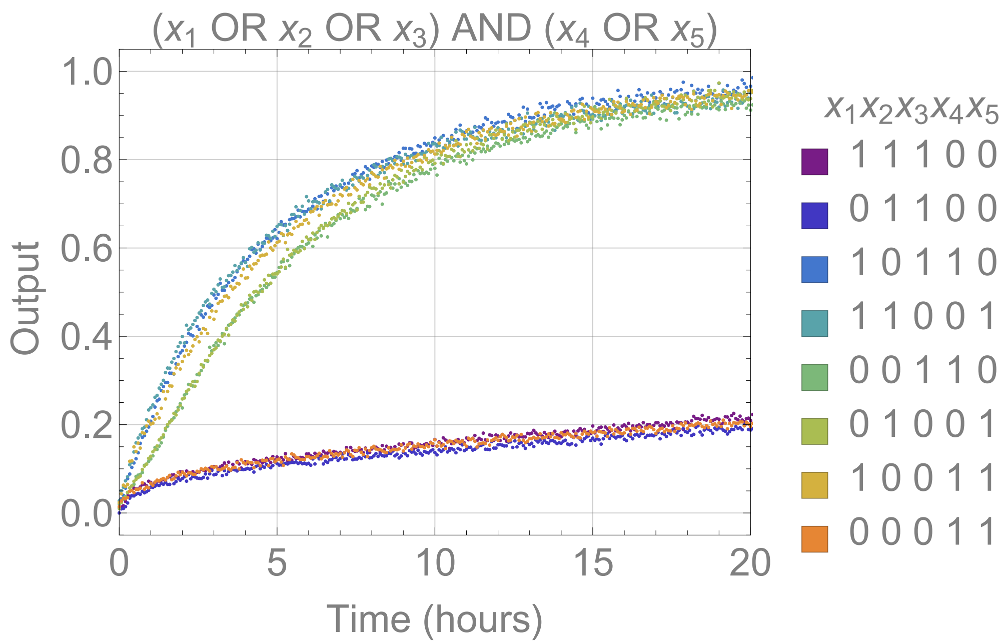
```
SetDirectory[NotebookDirectory[]];
TwoLayerCircuitData2 = Import["Layer2_circuit_OR_AND_R110.csv"];
```

```
PlotRowData[DataToList[TwoLayerCircuitData2, 1, 0, 0],
 {"1 1 1 0 0", "0 1 1 0 0", "1 0 1 1 0", "1 1 0 0 1", "0 0 1 1 0", "0 1 0 0 1",
  "1 0 0 1 1", "0 0 0 1 1"}, All, "  x1x2x3x4x5", "(x1 OR x2 OR x3) AND (x4 OR x5)"]
```



```
NormalizedTwoLayerCircuitData2 =
  NormalizeDataList[DataToList[TwoLayerCircuitData2, 1, 0, 0], 2, 3];
```

```
PlotNormalizedData[NormalizedTwoLayerCircuitData2,
 {"1 1 1 0 0", "0 1 1 0 0", "1 0 1 1 0", "1 1 0 0 1", "0 0 1 1 0", "0 1 0 0 1",
  "1 0 0 1 1", "0 0 0 1 1"}, 20, "  x1x2x3x4x5", "(x1 OR x2 OR x3) AND (x4 OR x5)"]
```



ON/OFF seperateion

The OFF trajectories (e.g. trajectory 1) are well below 0.3 when the ON trajectories (e.g. trajectory 6) reachs 0.7.
This is a good ON/OFF seperation for a two-layer circuit.

```
TrajectoryValue[NormalizedTwoLayerCircuitData2, 1, 6, 0.7]
```

```
0.14
```