

Introduction to the Data Analysis of the Roche xCELLigence™ System with RTCA Package

Jitao David Zhang

January 24, 2026

1 Introduction to the xCELLigence™ System

The xCELLigence™ System, also known as RT-CES™ system, which contains a series of Real-Time Cell Analyzer (RTCA), is a labeling-free cell-based assay system integrating micro electronics and cell biology, suitable for uninterrupted monitoring of biological processes of living cells.

The system relies on a micro-electronic biosensor built into each each of the 6 standard 96-well *E-plate*™ microtitre plates. The sensor measures the electrical impedance of the cell population in the well. Cells that have contact with the sensor change the electrical impedance between the microelectrodes. The impedance measurement provides quantitative read-time information about the status of the cells, including cell number, viability and morphology.

The Xcelligence system was invented by ACEA Biosciences and was co-developed by Roche and ACEA. Further information about the device and application notes are provided at <http://www.xcelligence.roche.com>.

1.1 Technology

In principle, assay with the xCELLigence™ System begins similar as other cell-based assays. However, instead of usual 96-well plates, cells are seeded in special plates (*E-plate*™ with micro electrodes covering well bottoms. The more cells attached on the electrodes, the larger the increases in electrode impedance. However, the cell number is not the factor determining the impedance. Other factors, like cell interaction strength with the electrodes and cell morphology, will also affect the impedance measurement. Therefore, electrode impedance, which is displayed and recorded as Cell Index (CI) values, reflect the biological statue of monitored cells, including the cell number, cell viability, morphology and adhesion degree.

The main read-out of the xCELLigence™ System, *Cell Index* (CI), is a dimensionless parameter. It is derived as a relative change in measured electrical impedance to represent cell status. It follows following rules:

- When cells are not present or not adhered on the electrodes, the CI is zero

- Under the same physiological conditions, the more cells are attached on the electrodes, the CI values are larger. That is, CI is marginally a monotonical function of cell number
- Additionally, change in cell status, such as cell morphology, cell adhesion or cell viability, will lead to a change in CI.

2 Basic features of the *RTCA* package

The open-source *RTCA* package is developed in the framework of Bioconductor with programming language R. As an alternative to the commercial software *RTCA* (current version 1.2, stand 07/2009), the *RTCA* package in R mainly provides following functionalities:

- Command-line operation of *RTCA* data in the environment of R.
- Alternatives of transformation methods besides the default strategy *ratioTransform* used by the commercial software, which is also included in the package.
- Various visualization possibilities of the data from xCELLigence™ System. Users could also develop other visualization tools with the data structure provided by the package based on their need.
- The analysis procedure can be further combined with other tools in the framework of Bioconductor, like *KEGGgraph* package for pathway graph analysis or *RpsiXML* for interaction analysis. Annotation packages could be also very useful to annotate the experiment results.

We draw the outline of basic use of the *RTCA* by a probe experiment performed by Dr. Ulrich Tschulena *et al.* from the Division of Molecular Genome Analysis, DKFZ. They used the xCELLigence™ System to examine the effect of gene knockdown in human HeLa cells with siRNA transfection. The experiments were replicated with three biological replicates. For the sake of simplicity we use the data from one *E-plate* to illustrate the pipeline of analysis with *RTCA*.

First of all we load the *RTCA* package

```
> library(RTCA)
```

2.1 Data import

The first step is to import the data from the *RTCA* device. In the software provided with the device, user can find the option to export tab-delimited files. Please check the help page of `parseRTCA` for a brief description of the file format. Once the data is exported, the import can be as simple as following two commands:

```
> ofile <- system.file("extdata/testOutput.csv", package="RTCA")
> x <- parseRTCA(ofile)
```

The variable `ofile` refers to an exported tab-delimited file. Here we use the one provided with the package by calling `system.file` function. If the file is saved in other directory, one could modify the reference to the file by using the `file.path` command, like

```
> ofile <- file.path("/directory/of/the/file/myFile.txt")
```

`parseRTCA` parses the file into a `S4`-class object `RTCA`. We can print the object to get an impression of how is the object constructed.

```
> x
```

```
Experiment ID: 0811171136P1
RTCAtimeline
=====
time action
  0 start
=====
Time unit: hour
RTCA-run start time: 2026-01-24 21:21:52.004071

ExpressionSet (storageMode: lockedEnvironment)
assayData: 242 features, 96 samples
  element names: exprs
protocolData: none
phenoData
  rowNames: A01 A02 ... H12 (96 total)
  varLabels: Well GeneSymbol
  varMetadata: labelDescription
featureData
  featureNames: 0 0.93 ... 90.33 (242 total)
  fvarLabels: timepoints
  fvarMetadata: labelDescription
experimentData: use 'experimentData(object)'
Annotation:
```

The `x`, as an `RTCA` object, contains following information

1. Experiment ID, which can be later associated with meta-information of the experiment,
2. RTCA time line, containing the time track of actions performed during the assay,
3. ExpressionSet, the data structure containing core data of the RTCA experiment.

2.1.1 Annotation

Quite often we would like to annotate the wells of the *E-plate*, for example the siRNA number and targeting gene in our case. We have provided the annotation information of the experiment in a *csv* file. We build the annotation (which we call *phenoData* as an convention from the Bioconductor community) of the plate in the following steps.

```
> pfile <- system.file("extdata/testOutputPhenoData.csv", package="RTCA")
> pData <- read.csv(pfile, sep="\t", row.names="Well")
> metaData <- data.frame(labelDescription=c(
+           "Rack number",
+           "siRNA catalogue number",
+           "siRNA gene symbol",
+           "siRNA EntrezGene ID",
+           "siRNA targeting accession"
+         ))
> phData <- new("AnnotatedDataFrame", data=pData, varMetadata=metaData)
```

The *phData*, our *phenoData*, contains the annotation information of the plate. We can assign it to the *phenoData* slot of the *RTCA* object by

```
> phenoData(x) <- phData
```

which annotates the object *x*

```
> head(pData(x))
```

	Rack	CatalogNumber	GeneSymbol	EntrezGeneID	Accession
A01	Plate 2	<NA>	GFP	NA	<NA>
A02	Plate 2	<NA>	mock	NA	<NA>
A03	Plate 2	M-003104-02	AXL	558	NM_001699
A04	Plate 2	M-008914-00	AZU1	566	NM_001700
A05	Plate 2	M-004932-00	BCKDK	10295	NM_005881
A06	Plate 2	M-003875-04	BCR	613	NM_004327

The procedure of annotation, alternatively, can be also accomplished together with the parsing, namely by the following codes (note that the *phData* object has been created)

```
> x <- parseRTCA(ofile, phenoData=phData)
```

See the help page of *parseRTCA* for more information on the use of the function.

It may be useful to examine the imported data visually, with *plot* function. Figure 1 visualizes the cell index readouts of the wells 1–4.

```
> plot(x[,1:4], type="l", col="black", lty=1:4, xlab="time point", ylab="Cell
```

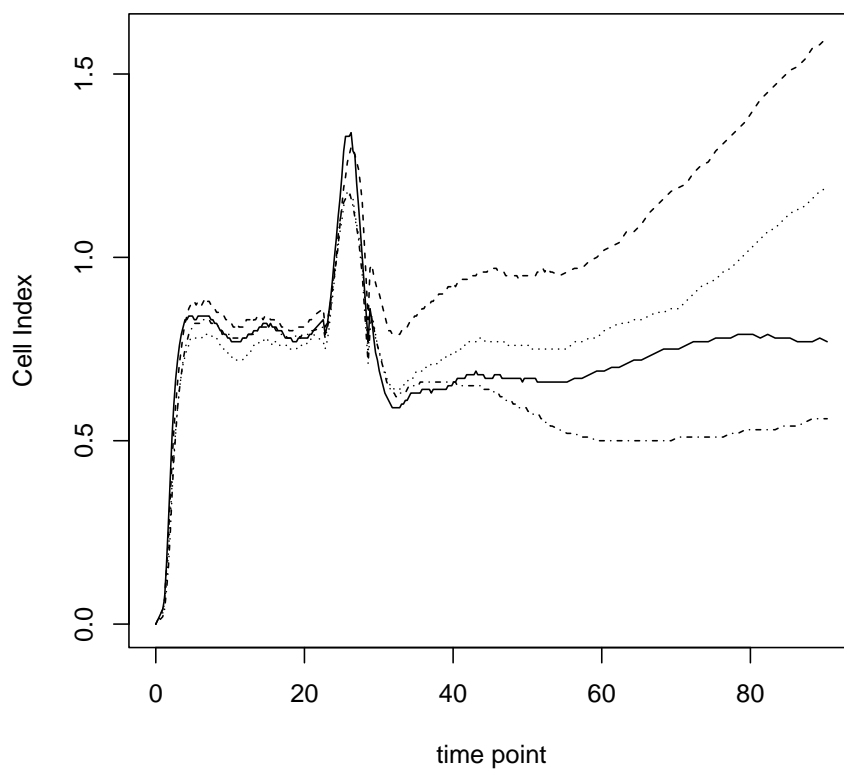


Figure 1: Plot RTCA data of the well 1 to 4. Note that the subset of the `RTCA` object is supported natively like that of `data.frame`.

2.2 Edit time line

One can edit the time line of the RTCA assay according to the experiment procedure. The feature allows to append the protocol to the data and simplifies the following manipulations. Suppose that in our case two main steps have been performed during the assay:

1. siRNA transfection at the 22th hour, and
2. cell culture medium change at the 30th hour.

We could edit the time line in the following way

```
> x <- addAction(x, 22, "trasfection")
> x <- addAction(x, 30, "medium change")
```

We could use `show(x)` to print the whole RTCA object to see the effect of editing time line, or we could extract the time line in the following way:

```
> timeline(x)
```

```
RTCAtimeline
```

```
=====
time      action
  0        start
 22  trasfection
 30 medium change
=====
```

```
Time unit: hour
```

```
RTCA-run start time: 2026-01-24 21:21:52.004071
```

Note that corresponding changes have been made. The time line slot of RTCA supports CRUD (create, read, update and delete) manipulations. The following toy examples shows the basic use of these functions. For more information we refer the readers to help pages.

```
> tl <- timeline(x)
> show(tl)
```

```
RTCAtimeline
```

```
=====
time      action
  0        start
 22  trasfection
 30 medium change
=====
```

```
Time unit: hour
```

```
RTCA-run start time: 2026-01-24 21:21:52.004071
```

```

> tlAdd <- addAction(tl, 35, "normalization"); tlAdd

RTCAtimeline
=====
time          action
    0           start
   22  trasfection
   30 medium change
   35 normalization
=====
Time unit: hour
RTCA-run start time: 2026-01-24 21:21:52.004071

> getAction(tl, 22)

[1] "trasfection"

> tlComp <- updateAction(tl, 22, "compound transfection")
> tlComp

RTCAtimeline
=====
time          action
    0           start
   22 compound transfection
   30          medium change
=====
Time unit: hour
RTCA-run start time: 2026-01-24 21:21:52.004071

> tlRm <- rmAction(tlAdd, 35 ); tlRm

RTCAtimeline
=====
time          action
    0           start
   22  trasfection
   30 medium change
=====
Time unit: hour
RTCA-run start time: 2026-01-24 21:21:52.004071

```

2.3 Transformation

The software provided with the device takes the strategy which we call *ratio transformation*, which divides cell index (CI) readouts of all wells at all time points over the CI value of individual wells at a same time point, which is known as the *base-time*. The transformation scales the CI values in different wells at the base-time uniformly as 1, making the transformed (normalized) cell index more comparable between the wells. This function is implemented in the *RTCA* package as the function `ratioTransform`, which takes a *RTCA* object and the *base-time* as parameter. The resulted *normalized cell index* is actually the relative cell impedance presented in the percentage of the value at the base-time.

Figure 2 shows the effect of such a transformation.

The ratio transformation scales the cell index in different wells and makes them comparable, which is desired in many situations. However, we believe that this transformation also bring in problems:

- Since the *base-time* is often arbitrarily set (in our case it is set as 5 hours later than the medium change), its selection may affect the curve of normalized cell index dramatically. In our special case, if we assume that the cell populations are similar before siRNA transfection and set the base-time right before the step, the normalized values of the time points after the transfection will incorporate the effects caused by siRNA transfection and medium change, which do not interest us very much. When we set the base-time short after the medium change, the shift of the base-time would lead to very different shapes of the curve.
- Another concern is the variance distortion caused by the ratio transformation. It is observed easily that the variance of transformed values are very small around the base-time and get significantly larger as the time increases. This raises concerns when similar variance is expected, for instance in the situation of many canonical statistical tests.

Therefore we also search for other transformation possibilities, including *derivative transformation* and *relative growth rate transformation*, which are discussed in the other vignette of the package.

2.4 Visualization

Besides the `plot` function we have seen before, *RTCA* provides also other possibilities to visualize the data. For example, the `plateView` depicts the *RTCA* result of one *E-plate* in one figure, so that user could get an impression of the assay with a single glance, like in Figure 3

In addition the package also provide function `controlView` for the visualization of a few samples. Please check the help page for the use of the function.


```

> xRatio <- ratioTransform(x, 35)
> plot(xRatio[,1:4], lty=1:4, type="l", col="black",xlab="time point", ylab="Normalized Cell Index")
> abline(v=35, col="red", lwd=2)

```

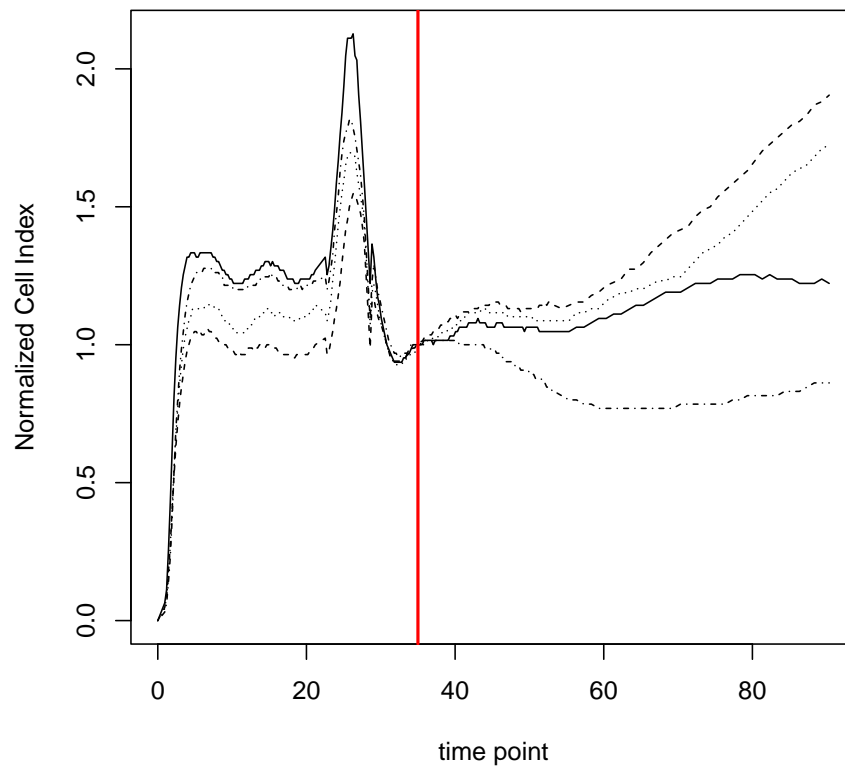


Figure 2: Plot ratio transformed RTCA data of the well 1 to 4, with the base-time indicated by the red vertical solid line. Note the normalized cell index at the base-time is set to 1 in all wells

```
> plateView(x, col="orange")
```



Figure 3: Plate view of a 96-well *E-plate* from a RTCA assay run.

3 What to do next?

Users familiar with the basic data structures and operations are advised to explore more functionalities documented in the vignette *RTCAtransformation*.

4 Session Info

The script runs within the following session:

- R version 4.5.2 (2025-10-31), x86_64-pc-linux-gnu
- Locale: LC_CTYPE=en_US.UTF-8, LC_NUMERIC=C, LC_TIME=en_US.UTF-8, LC_COLLATE=C, LC_MONETARY=en_US.UTF-8, LC_MESSAGES=en_US.UTF-8, LC_PAPER=en_US.UTF-8, LC_NAME=C, LC_ADDRESS=C, LC_TELEPHONE=C, LC_MEASUREMENT=en_US.UTF-8, LC_IDENTIFICATION=C
- Time zone: Etc/UTC
- TZcode source: system (glibc)
- Running under: Ubuntu 24.04.3 LTS
- Matrix products: default
- BLAS: /usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3
- LAPACK:
/usr/lib/x86_64-linux-gnu/openblas-pthread/libopenblas-p0.3.26.so
; LAPACK version 3.12.0
- Base packages: base, datasets, grDevices, graphics, methods, stats, utils
- Other packages: Biobase 2.70.0, BiocGenerics 0.56.0, RColorBrewer 1.1-3, RTCA 1.62.0, generics 0.1.4, gtools 3.9.5, xtable 1.8-4
- Loaded via a namespace (and not attached): buildtools 1.0.0, cli 3.6.5, compiler 4.5.2, evaluate 1.0.5, knitr 1.51, maketools 1.3.2, rlang 1.1.7, sys 3.4.3, tools 4.5.2, xfun 0.56