Data Processing with KETCindy

Yasuyuki Kubo(✉)

National Institute of Technology, Yuge College, Ehime, Japan
kubo@gen.yuge.ac.jp

Abstract. KETCindy is helpful not only for making TEX documents needing mathematically precise artwork but also for performing data processing efficiently. Furthermore, graphs obtained using KETCindy can be used in TEX documents without conversion.

1 Introduction

Spreadsheet software, such as Excel, is convenient for checking simple statistics and calculations. Entries can be easily typed into corresponding cells, and by selecting a range, such things as data number, average, and total can be displayed on the status bar and confirmed immediately. Scatter charts and line graphs can also be displayed on the screen if selected from the menu. However, as this report will show, KETCindy [1] can be used to handle complicated data processing with greater efficiency than spreadsheets such as Excel.

The advantage of KETCindy for data processing can be illustrated with an example from a project the author participated in. The project analyzed pollutants found in a certain area since 2013 onwards. Figure 1 shows part of the data (April 1st 2015 to February 29th 2016) as displayed in an Excel spreadsheet. The names of pollutants are displayed in the first row, e.g. PM2.5, PM10, etc.

To obtain the average level of each pollutant measured in ppm over the period, the following formula

```
=AVERAGE(B2..B336)
```

was entered in cell B337 of the Excel file and applied to all cells to the right.

Also, in order to obtain the moving average [2] of PM2.5, the following formula

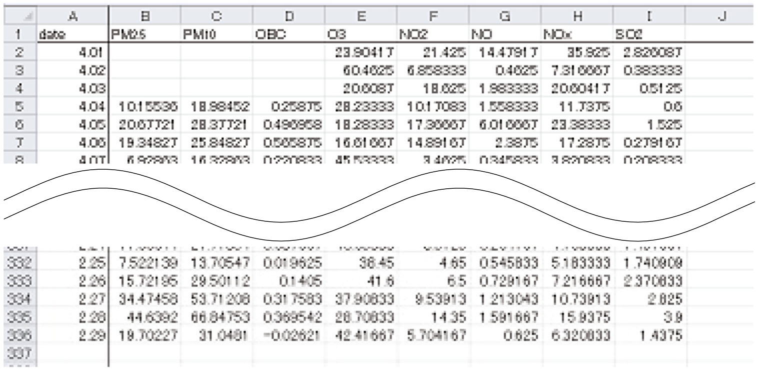
```
=AVERAGE(B2,B8)
```

was entered in cell J8 and applied to all the other cells in the column. By applying

```
=IF(MOD(ROW(B7),7)=0,AVERAGE(B2:B8),"")
```

to the column, the average of every week could be displayed in every 7th cell.

O. Gervasi et al. (Eds.): ICCSA 2017, Part IV, LNCS 10407, pp. 240–250, 2017.
DOI: 10.1007/978-3-319-62401-3_18



	A	B	C	D	E	F	G	H	I	J
1	date	PM2.5	PM10	OBC	O3	NO2	NO	NOx	SO2	
2	4.01				23.90417	21.425	14.47917	35.925	2.820087	
3	4.02				60.4025	6.858333	0.4025	7.316667	0.383333	
4	4.03				20.0087	18.025	1.983333	20.00417	0.5125	
5	4.04	10.15536	18.98452	0.25875	28.23333	10.17083	1.558333	11.7375	0.6	
6	4.05	20.67721	28.37721	0.490958	18.28333	17.36667	6.016667	23.38333	1.525	
7	4.06	19.34827	25.84827	0.565875	16.61667	14.89167	2.3875	17.2875	0.279167	
8	4.07	6.92963	16.32963	0.220833	45.53333	3.4025	0.345833	3.820833	0.208333	
331	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	
332	2.25	7.522139	13.70547	0.019625	36.45	4.65	0.545833	5.183333	1.740909	
333	2.26	15.72195	29.50112	0.1405	41.6	6.5	0.729167	7.216667	2.370833	
334	2.27	34.47458	53.71208	0.317583	37.90833	9.53913	1.213043	10.73913	2.825	
335	2.28	44.6392	66.84753	0.369542	28.70833	14.35	1.591667	15.9375	3.9	
336	2.29	19.70227	31.0481	-0.02621	42.41667	5.704167	0.625	6.320833	1.4375	
337										

Fig. 1. Table of pollutants

However, it was not easy to make an average list every week. For example, by entering either

```
=AVERAGE(INDIRECT(ADDRESS(2+7*(ROW(C1)-1),2)&":"&ADDRESS(1+7*
(ROW(C1)),2)))
```

or

```
=AVERAGE(OFFSET($B$2,(ROW(B2)-2)*7,0,7,1))
```

in cell J2 and copying down, the formula tended to be more error prone and difficult to correct. Furthermore, calculation of the monthly average was made more difficult because the number of days of each month varied. In the case of Excel, a macro can be used but with difficulty.

To avoid the above difficulties encountered while attempting to use Excel, the author turned to Cinderella in order to complete the task more efficiently. Cinderella is a type of DGS (dynamic geometry software) developed by J. R-Gebert and U. Kortenkamp [3]. Unlike other DGS, it incorporates CindyScript, an easy-to-use programming language. CindyScript handles data in addition to numerical values and character strings called lists. A vector (sequence data), for example, can be expressed as `vec=[1,3,5]`. In addition, a matrix (table data) can be expressed as a double nested list. For example, `mat=[[1,3,5],[2,4,6]]` is a matrix of 2 rows and 3 columns. Also we can use functions of CindyScript to handle lists as explained in [4]. The following is some samples of functions in CindyScript which are subsequently used to define new functions in this paper.

`int1..int2`	lists integers from `int1` to `int2`
	Ex : `3..7=[3,4,5,6,7]`
`list_int`	takes the `int`-th element of `list`
	Ex : `[2,5,7,1]_3=7`

`append(list,expr)`	adds `expr` to `list` such as `list++[expr]` Ex1 : `append([1,2],5)=[1,2,5]` Ex2 : `append([1,2],[5])=[1,2,[5]]`
`apply(list,oper)`	applies the operation `oper` to all elements of `list` Ex : `apply(3..5,f(#)])=[f(3),f(4),f(5)]`
`select(list,bool)`	selects the elements of `list` that are true Ex : `select(3..7,isodd(#))=[3,5,7]`
`sum(list)`	sums all the elements of `list` if they are all numbers Ex : `sum(-1..3)=5`
`row(mat,int)`	gives the `int`-th row of `mat`, if `mat` is a matrix Ex : `row([[1,3,5],[2,4,6]],2)=[2,4,6]`
`column(mat,int)`	gives the `int`-th column of `mat`, if `mat` is a matrix Ex : `row([[1,3,5],[2,4,6]],2)=[3,4]`
`transpose(mat)`	transposes a matrix `mat` Ex : `transpose([[1,3,5],[2,4,6]])` `=[[1,2],[3,4],[5,6]]`

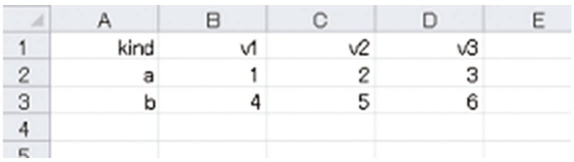
Here # represents the running variable, successively taking the value of all elements in the list.

While KETCindy was developed as a Cinderella plug-in to generate LaTeX source code for high quality mathematical artwork [5], it can perform several data processing functions as well. Data processing with KETCindy is in fact quite easy and very flexible as the next sections show.

2 Data Handling with KETCindy

2.1 Data Input, Output and Display

The following is an example of how KETCindy can be used to handle data. First, KETCindy converts the csv file (`file.csv` in this case) as shown in Fig. 2



	A	B	C	D	E
1	kind	v1	v2	v3	
2	a	1	2	3	
3	b	4	5	6	
4					
5					

Fig. 2. View in excel

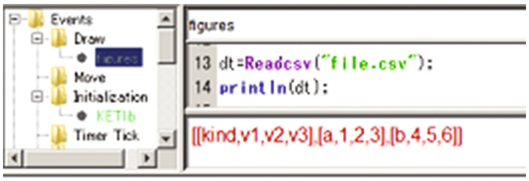
into the data named `dt` by the command

```
dt=Readcsv("file.csv");
```

if `file.csv` exists in a working directory. Also this command takes `dt` from the directory and arranges it into a matrix. Inputting and executing the command

```
println(dt);
```

on the edit area (the area on the top right of Fig. 3), the resulting matrix is displayed on the console (lower right of Fig. 3).

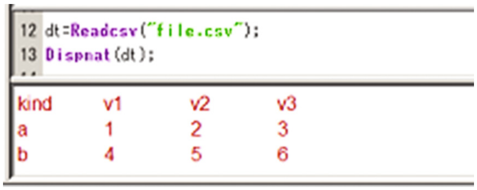


```
dt=Readcsv("file.csv");
println(dt);
```

```
[[kind,v1,v2,v3],[a,1,2,3],[b,4,5,6]]
```

Fig. 3. Readcsv and println

Substituting `Dispmat(dt)` for `println(dt)`, the matrix is displayed as tab-separated strings on the console, as shown in Fig. 4.



```
dt=Readcsv("file.csv");
Dispmat(dt);
```

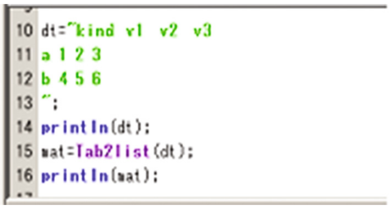
```
kind    v1    v2    v3
a       1     2     3
b       4     5     6
```

Fig. 4. Readcsv and Dispmat

Conversely, tab-separated strings can be translated into a matrix using the command, `Tab2list`. For example, Fig. 5 shows the data copied from the Excel window in Fig. 2 onto a CindyScript screen.

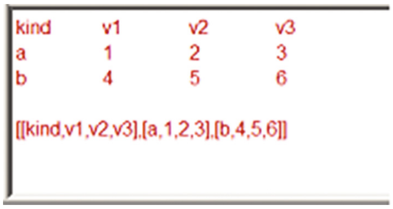
Here the data, in double quotes following `dt`, is converted into a list using the command `Tab2list`. Figure 6 shows the matrix which is made from `dt` and displayed on the console.

When the amount of data is large, it is necessary to scroll down the screen and look for the start and end of data to select and copy the range, which might cause mistakes. On the other hand, the command `Readcsv` is useful because it automatically selects the data and copies it into the edit area.



```
dt="kind v1  v2  v3
a 1 2 3
b 4 5 6
";
println(dt);
mat=Tab2list(dt);
println(mat);
```

Fig. 5. Tab2list (edit area)



```
kind    v1    v2    v3
a       1     2     3
b       4     5     6

[[kind,v1,v2,v3],[a,1,2,3],[b,4,5,6]]
```

Fig. 6. Tab2list (console)

2.2 Data Processing with KETCindy Commands

While it is necessary to scroll down the window to select a wide data range in Excel, the commands `Nrow` and `Ncol` added by the author enable us to select it automatically in KETCindy. Also, the translation of row data into column data is accomplished simply by using the CindyScript command `transpose`. Moreover, we can easily keep or delete some part of a large table simply by using KETCindy commands prepared to store and substitute variables. Time consuming cell-by-cell work is needed to accomplish these tasks in Excel. Thus, much labor can be saved by adding commands to KETCindy.

The author will introduce the commands added to KETCindy using `file.csv` as an example. Commands beginning with lower case letters were of Cindyscript origin. Those newly defined by the author in KETCindy begin with capital letters like other commands in KETCindy. The commands added to KETCindy to handle matrices are as follows

`Nrow(mat)`	returns the number of rows of matrix `mat`
`Ncol(mat)`	returns the number of columns of matrix `mat`
`RemoveR(mat,list)`	removes the n-th row from `mat` if n is in `list`
`RemoveC(mat,list)`	removes the n-th column from `mat` if n is in `list`
`AddR(mat,int,list)`	adds `list` to `int`-th row of `mat`
`AddC(mat,int,list)`	adds `list` to `int`-th column of `mat`

For example, let `dt` be the data of `file.csv` obtained using the command

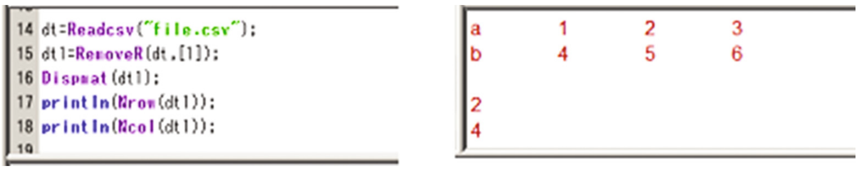
```
dt=Readcsv("file.csv");
```

We can easily delete the first row of `dt` with the command

```
dt1=RemoveR(dt,[1]);
```

Also we can easily find the number of rows and columns in `dt1` using the following commands (left side of Fig. 7) whose result is on the right of Fig. 7.

```
dt=Readcsv("file.csv");
dt1=RemoveR(dt,[1]);
println(Nrow(dt1));
println(Ncol(dt1));
```



```
dt=Readcsv("file.csv");
dt1=RemoveR(dt,[1]);
Dispmat(dt1);
println(Nrow(dt1));
println(Ncol(dt1));
```

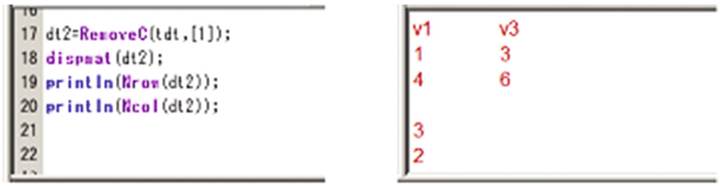
```
a        1        2        3
b        4        5        6

2
4
```

Fig. 7. Remove 1st Row and Return Nrow, Ncol

Similarly, the results obtained by the following commands

```
dt2=RemoveC(dt,[1,3]);    //remove 1st and 3rd columns
println(Nrow(dt2));
println(Ncol(dt2));
```

can be seen on the right of Fig. 8.



```
dt2=RemoveC(tdt,[1]);
dispmat(dt2);
println(Nrow(dt2));
println(Ncol(dt2));
```

```
v1        v3
1         3
4         6

3
2
```

Fig. 8. Remove 1st and 3rd Columns, and Return Nrow, Ncol

Also the command, `Removemat` was added by the author. The combination of `RemoveR` and `RemoveC` returns the same results as `Removemat`. The following commands

```
dt3=RemoveR(dt,[1]);
dt4=RemoveC(dt3,[1,3]);
```

create the matrix `dt4`, `[[1,3],[4,6]]`, and the next command

```
dt5=Removemat(dt,[1],[1,3]);
```

creates a matrix `dt5` identical to `dt4`. Furthermore, the command `Submat` was added which leaves the specified part of the matrix.

Important commands that draw and label graphs are defined as follows

`Linedata(list)`	makes a list of lists that have no empty element, even if `list` has some empty elements
`Linedata2(mat)`	makes a list of lists that has no empty element, even if the 2nd column of `mat` has some empty elements
`Linegraph(...list...)`	makes a line graph from the data specified
`Bargraph(...list...)`	makes a bar graph from the data specified
`Putlabel(...list...)`	puts labels on a graph from the data specified

Here the commands `Linegraph`, `Bargraph`, and `Putlabel` have many arguments as explained below.

For example, let `dt` be the list `[21,"",22,23,"","",24,25,26,"",27, 28,""]`.

Then, the following command

```
dt1=Linedata(dt);
```

produces `dt1` which consists of four lists

```
[[21,1]],                   [[22,3],[23,4]],
[[24,7],[25,8],[26,9]],     [[27,11],[28,12]].
```

Furthermore, let `ma` be the matrix made by the following command

```
ma=transpose([101..113,dt]);
```

which consists of [101,1], [102,""], [103,3], [104,4], [105,""], [106,""], [107,7], [108,8], [109,9], [110,""], [111,11], [112,12], [113,""]. Then, `ma1` made by

```
ma1=Linedata2(ma);
```

consists of four lists

```
[[1,101]],                      [[3,103],[4,104]],
[[7,107],[8,108],[9,109]],      [[11,111],[12,112]].
```

Based on these commands, the commands `Linegraph`, `Bargraph` and `Putlabel` were defined to generate line graphs and bar graphs from data like `dt1` and create graph labels from data like `mat1`. Both `Linegraph` and `Bargraph` take the following arguments: "`name`", "`hscale`", "`vscale`" and, "`valuelist`" which has the structure of `dt1`. `Linegraph` connects points that are continuously measured, while points with gaps are not connected. The argument "`hscale`" gives the horizontal scale of the graph, while "`vscale`" gives the vertical scale. `Putlabel` takes the following arguments: "`gap`", "`scale`", "`depth`", and "`labellist`" which has the structure of `mat1` whose element is a list consisting of pairs of natural numbers and labels (strings or real numbers). If there are too many labels, labels can be skipped by specifying gaps between the labels. Finally, "`hscale`" gives the horizontal scale of labels for `gap=1`, and "`depth`" adjusts the height of the labels with respect to the horizontal axis.

2.3 Other Functions

In addition to the before-mentioned commands, other functions are defined as follows

`Average(list)`	gives the average of values in `list`
`GroupAvg(list,opt)`	gives the average of values in `list` according to `opt`
`MovingAvg(list,int)`	gives the moving average of values in `list` by pairing the number `int` (same as `MovingAverage`)

Although CindyScript does not have a function to obtain averages, they can be calculated using the following formula

```
sum(list)/length(list);
```

However, by this calculation method, the expected value cannot be obtained in the case of a list that includes blanks and strings. As in Excel, the author's function `Average` gives an average value by ignoring blanks and strings.

In this way, labor intensive work in Excel can be replaced by the addition of efficient and easy-to-use commands.

3 Practical Example

For the data on air pollution introduced earlier, examples of data processing are shown. The data "PM20160229.csv" is given in csv format and read by the following command

```
dtorg=Readcsv("PM20160229.csv");
```

to give dtorg. The only data needed in that section is the date and part of the measured value for PM2.5. By the following command

```
dtall=Submat(dtorg,2..Nrow(dtorg),1..2);
```

the author made a submatrix dtall from the second to the last row of the data, the first and second columns of dtorg, using the command Submat.

3.1 Transition of Daily Values and of Moving Average per Week

The graph in Fig. 9 shows the daily change in levels of PM2.5 as measured in ppm and the change in the moving average over the week.

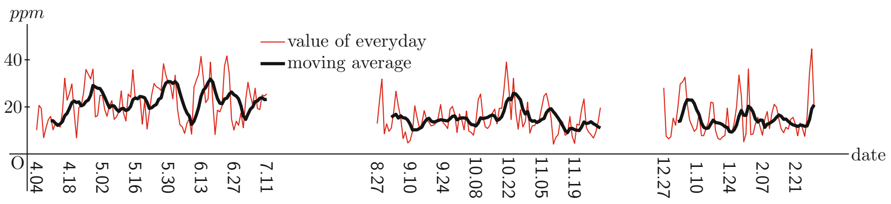


Fig. 9. Average and moving average

The graph was made using the following commands

```
dtvalue=RemoveC(dtall,1);
graphdt1=Linedata(dtvalue);
graphdt2=Movingaverage(graphdt1,7);

Setcolor("red");
Linegraph("day",graphdt1,0.05,0.05,["color->[1,0,0]"]);
Setcolor("black");
Linegraph("week",graphdt2,0.05,0.05,["dr,3"]);

ApplymatC(dtall,"Sprintf(#,2)",1);
ptnames=Linedata2(dtlabel);
Putlabel(ptnames,14,0.05,-0.5,["rot"]);
```

Here the commands ApplymatR, ApplymatC were added to modify the matrix to behave like apply. Without ApplymatC, the command

```
ApplymatC(dtall,"Sprintf(#,2)",1);
```

would be more complicated.

For the date expressed as a decimal number `Sprintf` was used, so that April 10 is displayed as "4.10" not "4.1". `Putlabel` was used to add labels. And to prevent overlap, gaps between labels were specified by skipping 14 and rotating the labels by 90° with the option `"rot"`.

3.2 Monthly Averages

To see the change in average levels of PM2.5 month by month, there was the difficulty of identifying the months consisting of different numbers of days.

To draw the graph (Fig. 10), the prepared data, `dtall`, was processed in the following way

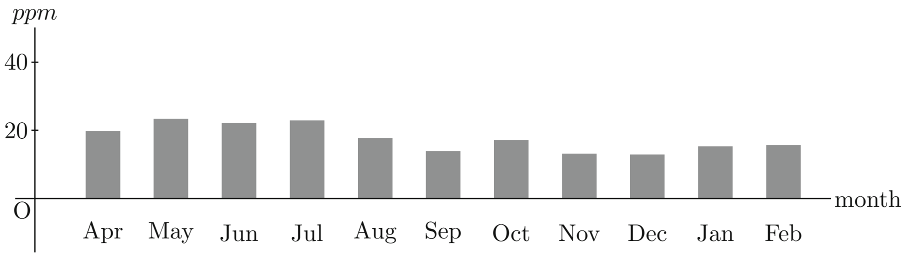


Fig. 10. Averages of each month

```
monlist=concat(4..12,1..2);
mavg=[];
forall(monlist,mon,
  mlist=select(dtall,floor(#_1)==mon);
  mavgtmp=Average(column(mlist,2));
  mavg=append(mavg,[mon,mavgtmp]);
);

graphdt=Linedata(column(mavg,2));
Setcolor([0.5,0.5,0.5]);
Bargraph("bar",graphdt,1,0.05,["dr,25"]);
Setcolor("black");

monlistE=["Apr","May","Jun","Jul","Aug","Sep","Oct","Nov",
"Dec","Jan","Feb"];
labelmon=Linedata(monlistE,["st"]);
Putlabel(labelmon,1,1,-0.5);
```

Though the list, `monlist`, is not in the natural order, it caused no problem since `forall` in CindyScript runs a temporary variable in the order listed.

4 Conclusions and Future Work

The author found that using KETCindy to process data by commands according to complicated specifications was superior to using spreadsheet software such as Excel. Also, TEX documents, including accurate graphs, could be created more efficiently using KETCindy.

Furthermore, there is no need to buy expensive software to improve the functionality of spreadsheet software, which is already expensive enough, because all related software is freely accessible, making KETCindy especially valuable for education.

Finally, there is room to improve the ease of use, processing speed and modifiability of commands and functions introduced. In addition, the author would like to integrate similar commands that are still confusing.

Acknowledgments. I am deeply grateful to Professor Setsuo Takato of Toho University for supporting and advising me in this research. I am also grateful to Mr. Akira Iritani, representative of Cinderella Japan, for advice on Cinderella and CindyScript input manners.

References

1. Takato, S.: What is and how to use KeTCindy – linkage between dynamic geometry software and KeTCindy graphics capabilities. In: Greuel, G.-M., Koch, T., Paule, P., Sommese, A. (eds.) ICMS 2016. LNCS, vol. 9725, pp. 371–379. Springer, Cham (2016). doi:10.1007/978-3-319-42432-3_46
2. Moving Average. https://en.wikipedia.org/wiki/Moving_average
3. CinderellaJapan. https://sites.google.com/site/cinderellajapan/
4. Cinderella.2 Documentation. https://doc.cinderella.de/tiki-index.php
5. KETpic.com. http://ketpic.com/