Usage of MATLAB Report Generator

Eniz Museljic

July 10, 2018

Contents

Chapter 1

Introduction

1.1 About this toolset

This toolset is to be used while programming in MATLAB to create a report or paper of any kind with the data from MATLAB, so it bridges the way and saves you the time to coppy values and make pretty tables and plots through an additional LaTeX editor. Please follow the upcoming instructions to get it up and running.

1.2 Setup

To set the LaTeX Generator up, you will need the following installed on your pc :

- LaTeX distribution for your system
- Latexmk package
- Ruby framework (on windows)

After installing these components, copy the reportGenerator main folder to a secure location on your machine and be sure to add it to your MATLABs search path so you can work from a different directory and use all the functions without restriction. To be sure that everything is set up correctly, we have provided a very simple "hello world" programme that you can run and check if it generates a pdf document with "hello world" on one page, in your current working directory.

1.3 How it works

In this section we will talk about all the different files that you can use and manipulate to get the best out of this toolset. The main files that you are gonna be changing are :

`setting.dat` Specifies a variaty of settings for the program which can be adjusted to your specific needs.

`myTextBlocks.tex` A file which contains blocks of Latex code separated by lines with `%% TAG` where `TAG` is a unique identifier.

`myCodeBlocks.m` A file which contains blocks of MATLAB code separated by lines with `%% TAG` where `TAG` is a unique identifier. It can be used to evaluate code specified in a section of this file.

`myMatlabInit.m` This is a init-File for MATLAB which is executed at the beginning. One can specify all necessarey settings there.

You can also specify files with different names for organisation purposes, if that's the case you will have to change the file names in the `setting.dat` file. Additionally you will need to edit the provided MATLAB `startup.m` file and set the LaTeX install path to the path on your computer.

Chapter 2

LaTeXGenerator

Now we will go through the functions available and show you what you can do with them, you can use this documentation and the MATLAB file `reportingDocu.m` with which it was created, to help you write your own reports. All of the functions that will be mentioned also have descriptions which can be accessed through MATLAB by using the *help* command fot the function in question.

2.1 New report

To use the following functions you will need to create a *latexGenerator* object which will be used to access the methods provided and explained in the following sections. This can be easily done with:

```
report = latexGenerator;
```

Note that the constructor function also takes in parameters with which we could like in the following example change the name of the settings file we are using, this can be very useful if we have a settings file with settings that fit our current needs but we might use a different settings file usually.

```
report = latexGenerator('settingFile', 'personalSettings.dat');
```

2.2 Set and Get

Using the *set* and *get* methods from the reportGenerator super class, we can access and change the settings provided by the `setting.dat` file from withing our MATLAB code. For example, if we wanted to change the name the program gives to the graphics it saves, we could use the following code :

```
report.set('mediaPrefix', 'plot_')
```

With this piece of code we set the name prefix of the plots that are being saved to 'plot_'. It

can be utilyzed to set and get, with the get function, all of the parameters in `setting.dat`.

2.3 Title page and TOC

Like in the most cases we would want to add a title page to our document, we will do this in the following lines of code. The title, author and date will be displayed with the corresponding settings from the setting file. We can also add a table of contents.

```
report.addTitlePage
report.addTableOfContents
```

2.4 Paragraphs

All of these paragraphs in this documentation were added using the paragraph function, you can choose if you want to add the paragraph by writing it whole as a string in MATLAB and passing it to the function or using the `myTextBlocks.tex` file from which we can import any latex code or text and use it in our documents just by passing the `TAG` to the paragraph function. An example for adding a short sentence would be :

```
report.addParagraph('This␣is␣an␣example␣sentence')
```

This is an example sentence

2.5 Headings

There are several types of headings which can be added :

- part
- chapter
- section
- subsection
- subsubsection

Using one of these heading types we can structure our document much better and provide a better overview, since without them, the table of contents would be useless. The following code was used to create the heading for this current section in this document :

```
report.addHeading('section', 'Headings')
```

2.6 Lists

There are two types of lists you can generate through MATLAB :

- itemize
- enumerate
- description

This list was created with the following code :

```
report.addList('itemize', {'itemize','enumerate','description'});
```

We can see that it is fairly easy and quick to add a MATLAB cell array of strings and make a pretty list in your document. The following examples show that nested lists are also possible, the code that was used to make these can be found in the `reportDocu.m` file under LISTS.

- Item 1
  1. Enum 1
  2. Enum2
  3. Enum3
- Item 2
  - Item 2.1
  - Item 2.2
- Item 3

Definition 1 : Text 1
Definition 2 : Text2 Some more text

2.7 Tabular and table

2.7.1 Tabular

Tables can be created directly from MATLAB arrays, following example shows a simple table containing the values from an array with random values. The following code was executed.

```
arr = randi(10,[2,3]);
report.addTabular(arr);
```

7.00	10.00	9.00
9.00	6.00	5.00

All of the settings for the tables can be adjusted in the `setting.dat` file or through the `set` function as we will see in this section. Now we will demonstrate how to lable the rows and columns. We will do that by specifying the row and/or column labels as strings with collon separated labels as we can se in the following code.

```
row_labels = 'row1,row2';
col_labels = 'col1,col2,col3';

report.set('dataFormat','%i');
report.set('tableRowLabels', row_labels);
report.set('tableColLabels', col_labels);
report.set('tableColumnAlignment','lccc');

report.addTabular(arr);
```

	col1	col2	col3
row1	7	10	9
row2	9	6	5

The following example shows that we can also pass data with Not a Number fields, the corresponding parameter in the `setting.dat` file has to be set appropriate, by default it is set to `nan`. We will also demonstrate what `booktabs` and `formating` do for a new table.

```
report.set('dataFormat','%0.1f,%0.3f,%0.1f,%0.2f');
report.set('tableColLabels','col1,col2,col3,col4');
report.set('tableRowLabels','row1,row2,row3,row4');
report.set('booktabs',1);

arr = randn(4,4)*5;
arr(3) = nan;
report.addTabular(arr);
```

	col1	col2	col3	col4
row1	-7.7	3.968	0.0	2.31
row2	4.2	7.839	-3.7	0.26
row3	-	4.262	2.8	-0.01
row4	-1.7	11.620	9.1	-9.25

In the following examples we will see some other possible adjustments we can make to our tables through MATLAB.

```
report.set('tableRowLabels','row1,␣,row3,row4');
report.set('tableColumnAlignment','c|cccc');
report.addTabular(arr);
```

row1	-7.686	3.968	0.028	2.311
	4.237	7.839	-3.731	0.256
row3	-	4.262	2.791	-0.010
row4	-1.690	11.620	9.135	-9.250

As we can see we can also leave specific rows or columns unlabeled. If we want to get rid of the table borders we just need to set the `tableBorders` to 0.

```
report.setDefault('table');
report.set('tableBorders', 0);
report.addTabular(arr);
```

-7.69	3.97	0.03	2.31
4.24	7.84	-3.73	0.26
-	4.26	2.79	-0.01
-1.69	11.62	9.13	-9.25

To reset all the modifications we did to the settings of the table, use the function `setDefault` as shown in the last example.

2.7.2 Table

Now we will demonstrate how to use the MATLAB table to create tables. We can use either the `addTabular` or `addTable` function to add the MATLAB table to our document. In the following examples we can see the usage :

```
LastName = {'Smith';'Johnson';'Williams';'Jones';'Brown'};
Age = [38;43;38;40;49];
Height = [71;69;64;67;64];
Weight = [176;163;131;133;119];
T = table(Age,Height,Weight,'RowNames',LastName);

report.addTabular(T);
```

	Age	Height	Weight
Smith	38	71.00	176.00
Johnson	43	69.00	163.00
Williams	38	64.00	131.00
Jones	40	67.00	133.00
Brown	49	64.00	119.00

Please note that all of the adjustments to the table properties we have done in the previous section are also applicable here, without any restriction. Next we will see the same MATLAB table added to our document throught the `addTable` function which allows us also to add a

caption for the table. We can also see the usage of the `transposeTable` setting which can also be applied to the former way of adding tables.

```
report.set('transposeTable',1);
report.addTable(T, 'My caption');
```

	Smith	Johnson	Williams	Jones	Brown
Age	38	43	38	40	49
Height	71.00	69.00	64.00	67.00	64.00
Weight	176.00	163.00	131.00	133.00	119.00

Table 2.1: My caption

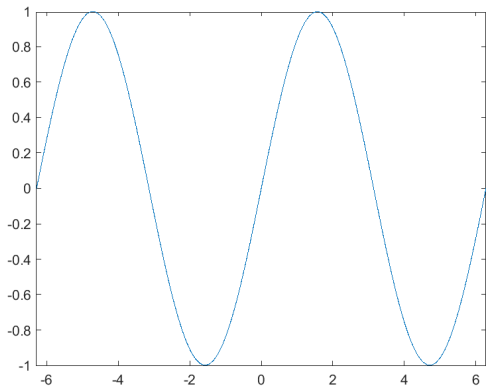
2.8 Images and Figures

To add figures to our document we have two functions at our disposal, `addImage` and `addFigure`. These functions work with figure handles. The following code was used to generate the figures which handles we are going to use to add them into our document.

```
figh = gobjects(1,3);
t = linspace(-2*pi,2*pi,100);

figh(1) = figure('visible','off');
plot(t,sin(t));
xlim([-2*pi,2*pi])

figh(2) = figure('visible','off');
plot(t,log(t), 'mo');
xlim([-2*pi,2*pi])

figh(3) = figure('visible','off');
plot(t,sinc(t), 'r-*');
xlim([-2*pi,2*pi])
```

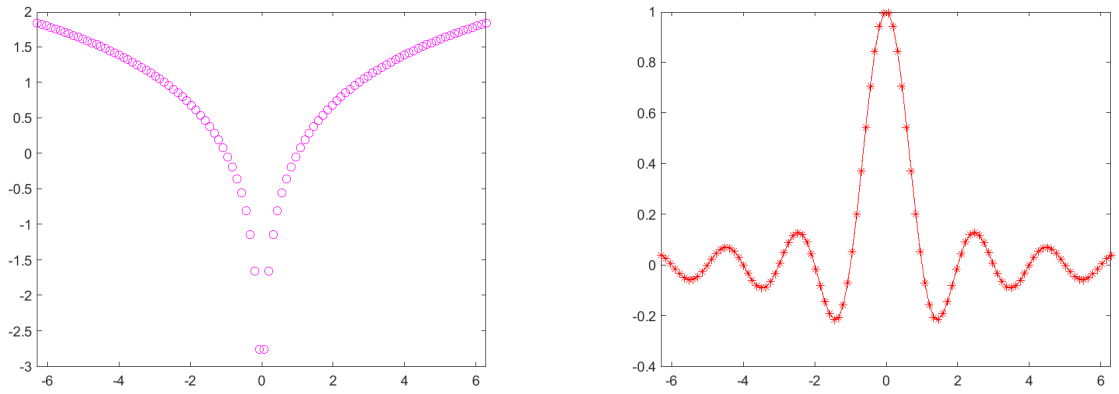
You can set the visibility of the figures off so it doesn't show them in MATLAB, like we did in the example above. Using the next line of code we added the first figure to our document.

```
report.addImage(figh(1));
```



As we have seen in the previous sections, it is also possible to edit settings for the figures and images we are adding with the `set` function. Next we will demonstrate how to add multiple figures in one line, because it is possible to pass an figure array to the `addImage` function.

```
report.set('imagePerLine',2);
report.addImage(figh(2:3));
```



```
report.set('imagePerLine',3);
report.set('imageOption', 'width=0.30\textwidth');
report.addFigure(figh, 'sin(t), log(t), sinc(t)');
```

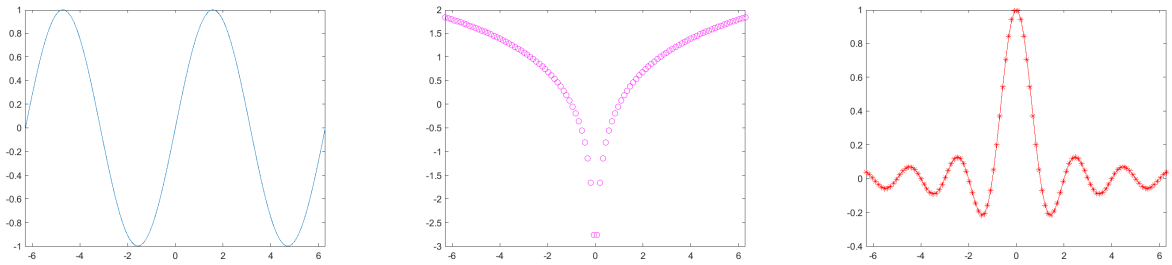


Figure 2.1: sin(t), log(t), sinc(t)

So we can see that by playing with the settings we can place the figures the way we want them. If we want to have bigger figures we can also add them in a vertical positioning.

```
report.set('imagePerLine',1);
report.set('imageOption', 'width=0.8\textwidth');
report.addFigure(figh, {'sin(t)', ' ', 'sinc(t)'});
```

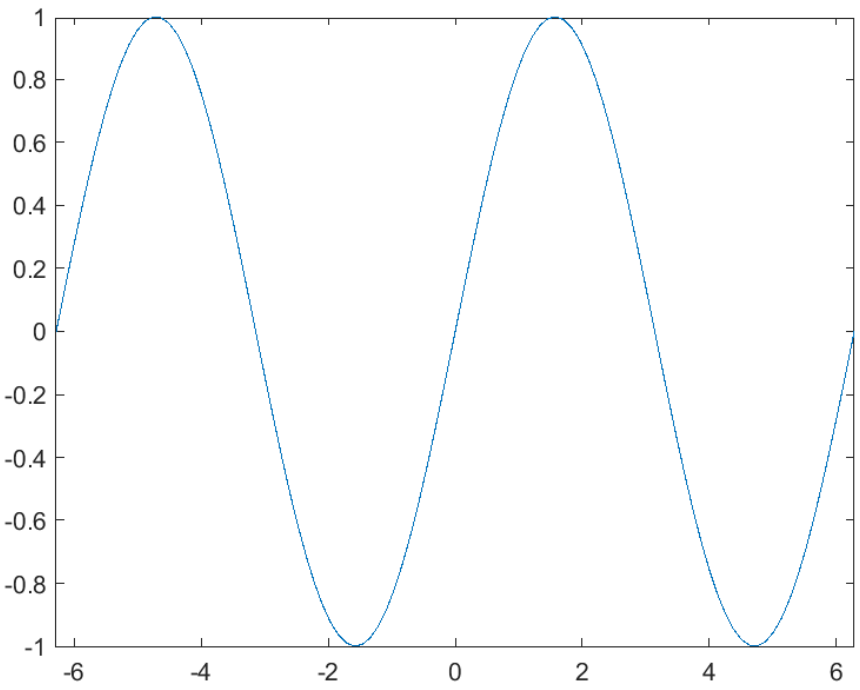


Figure 2.2: sin(t)

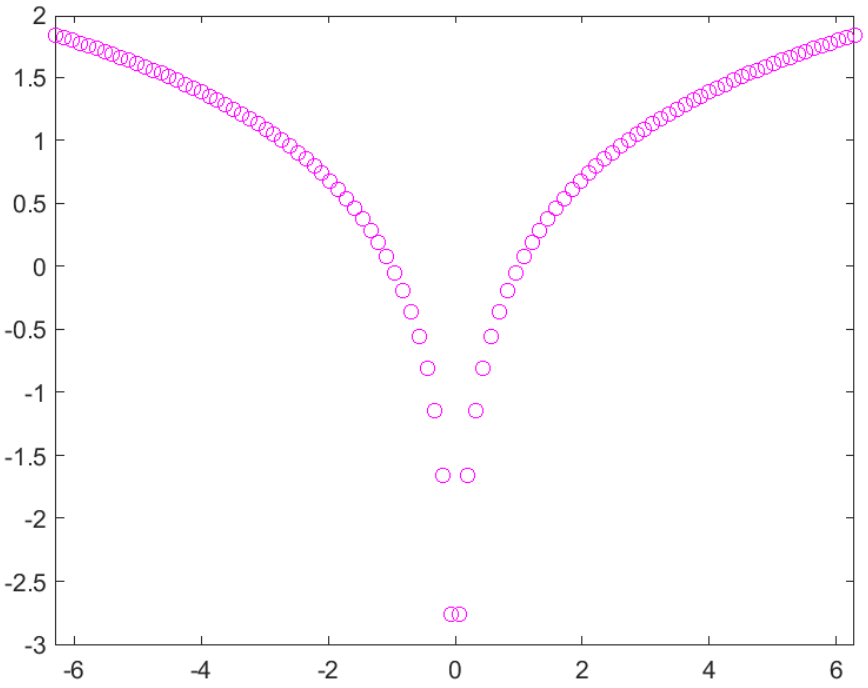


Figure 2.3:

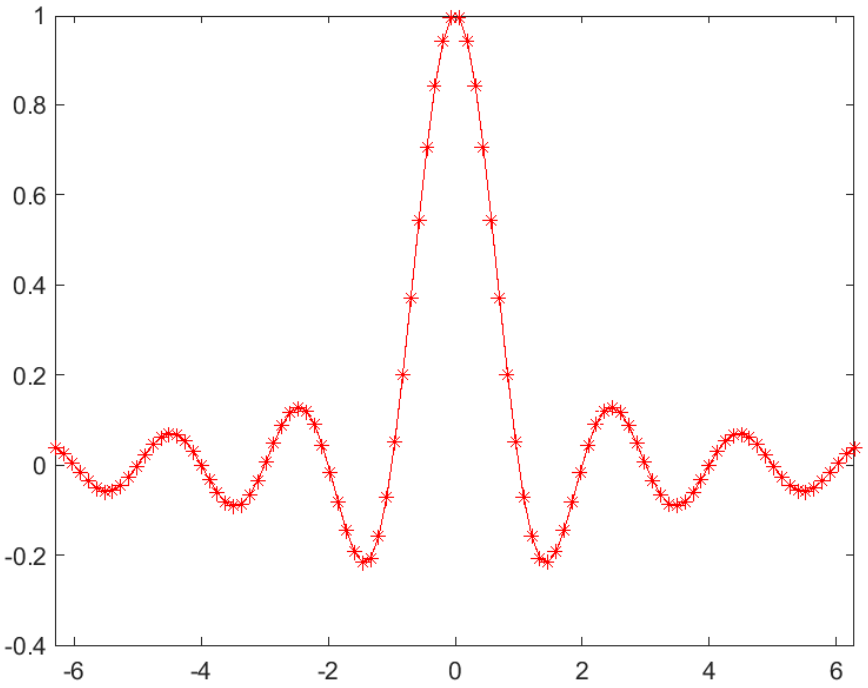


Figure 2.4: sinc(t)

With the above example we demonstrated how to add multiple figures, one per line, with different captions. You can adjust the size of the figures by accessing the `imageOption` property and setting the size as in the code above. In this case it is important to set the numbe of images per line to one or else it won't place them in the manner above. If you don't want to add a caption to a specific figure in the array, just pass an empty string in the right place as in the example above.

2.9 Pretty output

2.9.1 Listing and Latex

We can utilyze the `addMatlabOutput` function to display MATLAB console output in our document, it is possible to display it as same as in the console or in latex mode. The collor of the latex mode output can be set by accessing the `latexOutputColor` property with the `set` function. The following code was used to generate the matrices used in the following examples.

```
M1 = logical(eye(6));
M2 = flipud(M1);
```

If we use want the MATLAB console output we will use the following function and pass the variable names as a cell array of strings

```
report.addMatlabOutput({'M1','M2'});
```

```
    M1 =                    M2 =

     1  0  0  0  0  0          0  0  0  0  0  1
     0  1  0  0  0  0          0  0  0  0  1  0
     0  0  1  0  0  0          0  0  0  1  0  0
     0  0  0  1  0  0          0  0  1  0  0  0
     0  0  0  0  1  0          0  1  0  0  0  0
     0  0  0  0  0  1          1  0  0  0  0  0
```

If we want the same variables displayed but in latex mode then we will use the following syntax

```
report.addMatlabOutput({'M1','M2'}, true);
```

$$M1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad M2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Now with a different collor, standard black is also possible

```
report.set('latexOutputColor','blue');
report.addMatlabOutput({'M1'}, true);
report.set('latexOutputColor','green');
report.addMatlabOutput({'M2'}, true);
```

$$M1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

2.9.2 Big matrices

Another example would be wanting to add a matrix whose size is too big to fit into the document in a way that we dont just have a bunch of numbers on the screen, this is easy with the same function as before, lets use the following code

```
bigM = rand(100);
report.set('prettyMaxSize', '[8,6]');
report.addMatlabOutput({'bigM'});
```

```
    bigM =

     0.1372  0.9040  0.7551  0.6360  0.1408   >   0.5046
     0.9054  0.6330  0.0891  0.3407  0.6849   >   0.5851
     0.9564  0.3364  0.5850  0.4060  0.0075   >   0.5640
     0.6907  0.0465  0.4056  0.3553  0.3669   >   0.6797
     0.4376  0.0176  0.3416  0.4701  0.5801   >   0.2158
     0.5346  0.9819  0.4508  0.4396  0.1001   >   0.0153
     0.8259  0.4560  0.9898  0.4382  0.7439   >   0.2135
      |       |       |       |       |       >    |
     0.2720  0.5532  0.5830  0.0692  0.7903   >   0.3126
```

Or if we want it in latex style, this time in black color

```
report.set('latexOutputColor','black');
report.addMatlabOutput({'bigM'}, true);
```

$$\texttt{bigM} = \begin{bmatrix} 0.1372 & 0.9040 & 0.7551 & 0.6360 & 0.1408 & \cdots & 0.5046 \\ 0.9054 & 0.6330 & 0.0891 & 0.3407 & 0.6849 & \cdots & 0.5851 \\ 0.9564 & 0.3364 & 0.5850 & 0.4060 & 0.0075 & \cdots & 0.5640 \\ 0.6907 & 0.0465 & 0.4056 & 0.3553 & 0.3669 & \cdots & 0.6797 \\ 0.4376 & 0.0176 & 0.3416 & 0.4701 & 0.5801 & \cdots & 0.2158 \\ 0.5346 & 0.9819 & 0.4508 & 0.4396 & 0.1001 & \cdots & 0.0153 \\ 0.8259 & 0.4560 & 0.9898 & 0.4382 & 0.7439 & \cdots & 0.2135 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0.2720 & 0.5532 & 0.5830 & 0.0692 & 0.7903 & \cdots & 0.3126 \end{bmatrix}$$

2.9.3 Graphic output for arrays

In this part we will see how to create images from matrices. We will use the previously created `M1` and `M2`.

```
report.set('arrayImageColor',1,'arrayColorbar',1);
imageNames = report.createArrayImage({'M1','M2'});
report.setDefault('array');
report.set('imagePerLine', 2, 'imageOption', 'width=0.4\textwidth');
report.addFigure(imageNames,'Arrays as images');
```

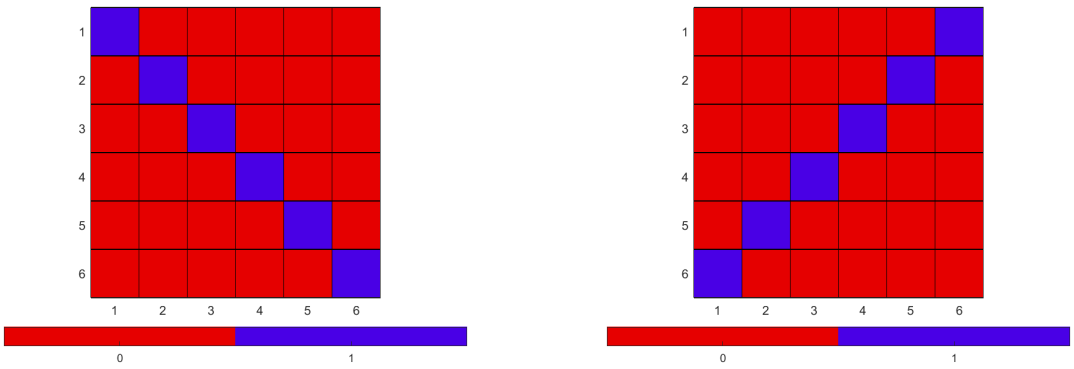


Figure 2.5: Arrays as images

To make one black and white image without the colorbar we will execute the following lines of code. Note how we passed the matrices variable names to the function, we enclosed the cell array with an additional pair of curly brackets to get one figure containing both arrays as images.

```
report.set('arrayImageColor',0,'arrayColorbar',0);
imageNames = report.createArrayImage({{'M1','M2'}});
report.setDefault('array');
report.addFigure(imageNames,'Arrays as image in bw');
```

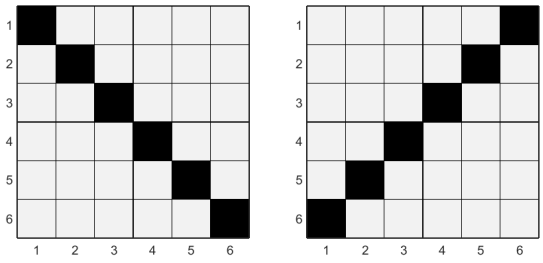


Figure 2.6: Arrays as image in bw

We can also generate images from matrices and arrays of random values, it would look like this for the previously generated matrix `bigM`.

```
report.setDefault('image');
report.set('arrayImageColor', 1, 'arrayColorbar', 1);
imageNames = report.createArrayImage({'bigM'});
report.setDefault('array');
report.addFigure(imageNames, 'Random values as image');
```

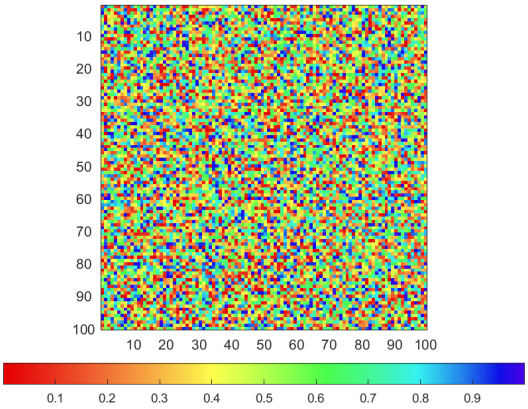


Figure 2.7: Random values as image

Using the previously presented functions some really nice representations can be made. Next you can see an image representation of a magic square, whose sum of every row, column and diagonal is the same.

```
M3 = magic(20);

report.set('arrayImageColor', 1, 'arrayColorbar', 1);
magicM = report.createArrayImage({'M3'});
report.set('imageOption', 'width=0.8\textwidth');

report.addFigure(magicM, 'Magic␣matrix␣20x20');
```

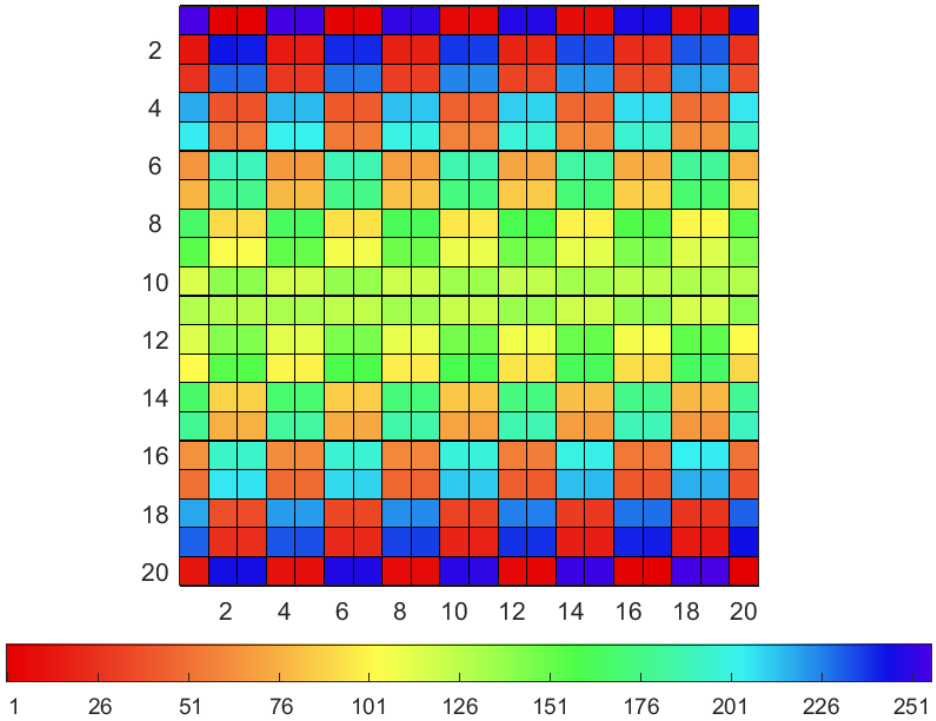


Figure 2.8: Magic matrix 20x20

2.10 Function descriptions

To get a good overview of what parameters a function takes in and which it returns back, what types od data each parameter is and a short description can be nicely structured as a table which holds all of that information. You can use the following methods to showcase a function more clearly.

```
myfun = 'exampleFunction.m';
pre = 'This part of the text comes before the table';
typ = {'logical','double','double','string','double'};
desc = {
    '1-D array', ...
    '2-D array', ...
    'scalar', ...
    '1-D array' ...
    'scalar'
    };
post = ['this text will be displayed after '...
                    'the function description'];
report.addFunctionDescription(myfun,typ,desc,pre,post);
```

This part of the text comes before the table

```
function [out1, out2] = exampleFunction(in1, in2, in3)
```

InOut	Name	Type	Description
Input	in1	logical	1-D array
Input	in2	double	2-D array
Input	in3	double	scalar
Output	out1	string	1-D array
Output	out2	double	scalar

this text will be displayed after 'the function description

The type and description can also be extracted from the MATLAB help output for that function

```
function [M1, M2] = matFun(n, V)
```

InOut	Name	Type	Description
Input	n	double	scalar, determines the size of the output matrices;
Input	V	double	1-D vector of length n
Output	M1	double	Sum of matrices with V on main and anti diagonal
Output	M2	double	Repeated vector V

Just by passing empty strings for `type` and `desc` it will get the data from the help output

2.11 Equations

There is also the possibility to add equations in latex syntax, like in the following examples. You can choose between the usual latex equation environments, if you want just one equation

pass `'equation'` as option, `'align'` can display more than one equation and can also align all of them at the `&` charakter. If you dont want to enumerate your equations, please add an asterix to the option, like in the example. It is also possible to type the latex code directly into a text file and add it with the paragraph function.

```
report.addEquation(['\nabla␣E␣&=␣\frac{\rho}{\epsilon_0}\\'...
    '\nabla␣B␣&=␣0\\'...
    '\nabla␣\times␣E␣&=␣-\frac{\partial␣B}{\partial␣t}\\'...
    '\nabla␣\times␣B␣&=␣\mu_0(J+'...
    '\epsilon_0\frac{\partial␣E}{\partial␣t})'],'align*');
```

$$\begin{aligned}
\nabla E &= \frac{\rho}{\epsilon_0} \\
'\nabla B &= 0 \\
'\nabla \times E &= -\frac{\partial B}{\partial t} \\
'\nabla \times B &= \mu_0(J +' \epsilon_0 \frac{\partial E}{\partial t})
\end{aligned}$$