Introduction to MATLAB

Jefferson Flórez, Aldo C. Martínez,* and Yazid Braik
Department of Physics, University of Ottawa - PHY3902
(Dated: September, 2020)

CONTENTS

I. PRELIMINARIES

The default desktop layout contains the panels described in Table I.

Panel	Functions
Current Folder	displays the files in the current working folder
Command Window	contains the command line, indicated by the prompt (>>), where the input commands are introduced
Workspace	displays the data imported, defined or obtained during the calculations

TABLE I. MATLAB panels.

In MATLAB, we create data arrays and use them as input in predefined functions. For example, let's define variables x and n by typing the following statements in the Command Window:

```
>> x = pi
x =
    3.1416
>> n = 2
n =
    2
```

You get in the Workspace panel the two new variables. Now, let's compute $\cos(2\pi)$:

```
>> cos(n*x)
ans =
     1
```

The result is 1. Note that this result appears in the Workspace as `ans`, short for "answer", which is the default name for an undefined variable. We can define the output result as `y` by typing

```
>> y = cos(n*x)
y =
    1
```

Now, the result appears in the Workspace as `y`. You can call this new variable to make further calculations:

```
>>  asin(y)
ans =
     1.5708
```

which is $\pi/2$. You can suppress the output of any calculation in the Command Window by ending a statement with a semicolon:

```
>>  asin(y);
```

You can also look for previous commands by pressing the up and down arrows in your keyboard.

* amart224@uottawa.ca

II. MATRICES AND ARRAYS

In comparison to other programming languages, MATLAB is particularly suitable for numerical calculations based on data arrays (indeed, MATLAB is an abbreviation for "matrix laboratory"). Therefore, any variable in MATLAB is handle as a data array, regardless of its dimension. For example, the variables `x`, `n` and `y` defined in Sec. I are 1×1 data arrays. To create higher dimension data arrays, like a row vector with two elements, we separate those elements with either a comma (`,`) or a blank space:

```
>>  a = [1 2];
```

We see in the Workspace the two values of the row vector `a` explicitly. To define a two-dimensional array (matrix), like a 3×3 matrix, we separate each row with semicolons:

```
>> b = [1 2 3; 4 5 6; 7 8 9];
```

We can also define a data array using the functions `ones`, `zeros`, or `rand`. For example, the following statement creates a 5×5 matrix with random numbers from 0 to 1.

```
>> c = rand(5,5);
```

Note that in the Workspace we have the dimensions of the variable `c` in the field "Value" instead of the actual numbers defining `c`. If you want to see the matrix `c` explicitly, you can either type `c` in the Command Window (without semicolon) or double-click this variable in the Workspace.

You can evaluate functions on a data array. For example,

```
>> a - 1;
>> cos(b);
>> exp(c);
```

You can also evaluate any matrix operation in MATLAB, like matrix transpose (`'`), matrix inversion (`inv`), and matrix multiplication (`*`):

```
>> c';
>> inv(c);
>> b * c;
```

In order to do element-wise calculations between data arrays, we use a dot in front of the operation symbol:

```
>> c.*c;
>> c.^2;
```

If the data array has complex entries, all the previous functions and operations can be evaluated as well. In the particular case of the matrix transpose, the rows and columns are not only switched, but the array elements are complex conjugated. For example, let's define matrix `d` as

```
>> d = [1+i 2-i; 3*i 1+4*i];
```

The matrix transpose of `d` is

```
>> d'
ans =
     1.0000 - 1.000i    0.0000 - 3.000i
     2.0000 + 1.000i    1.0000 - 4.000i
```

If you only want to calculate the transpose of a complex matrix without complex conjugating the matrix elements, you must use the command `.'` instead. For example, `d.'`

You can access any set of elements of a given matrix. Coming back to the matrix `b` defined above, we can get the second element in the second column in the following way:

```
>> b(2,2)
ans =
     5
```

If we want the second column of matrix `b` we should type

```
>> b(:,2)
ans =
     2
     5
     8
```

You can find the eigenvalues and eigenvectors of a matrix using the function `eig` in the following way:

```
>> [V,D] = eig(b);
```

The columns of matrix `V` are the eigenvectors of `b`, and the matrix `D` is a diagonal matrix with its non-zero elements being the eigenvalues of `b`.

III. 2D PLOTS

To plot a single variable function, you first need to define a set of values on which the function will be evaluated. For example, if we want to plot the sine function, we first define the variable `x` from 0 up to 2π in steps of 0.01:

```
>> x = 0:0.01:2*pi;
```

Then, we plot the sine functions using the command `plot`:

```
>> plot(x,sin(x))
```

If you do not specify the independent variable in the `plot` command, MATLAB uses the natural numbers 1, 2, 3,... in the horizontal axis. We can add other functions in the same plot using the command `hold on` and `hold off`:

```
>> figure(1)
>>  hold on
>> p = plot(x, x.^(1/3),'LineWidth',1,'Color',[1,0,0])
>> plot(x, x.^(1/2),'LineWidth',1,'Color',[0.7,0.3,0])
```

```
>> plot(x, x,'LineWidth',1,'Color',[0,1,0])
>> plot(x, x.^2,'LineWidth',1,'Color',[0.5,0.5,0])
>> plot(x, x.^3,'LineWidth',1,'Color',[0,0,1])
>> hold off
```

You can add a legend to the plot using the command `legend`:

```
>> legend('x^{1/3}','x^{1/2}','x','x^2','x^3')
```

We can also add axis labels, a plot title and increase the font size:

```
>> ylabel('f(x)')
>> xlabel('x axis')
>> title('Polynomials')
>> set(gca,'fontsize',18)
```

Finally, we save our plot in Portable Network Graphics (`.png`) format using the command `saveas`, as shown below. The resulting plot is shown in Fig. 1.

```
>> saveas(gca,'2Dplot.png')
```

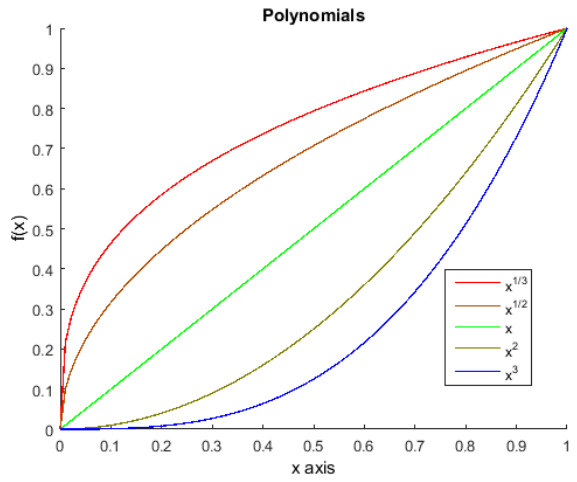

FIG. 1. A plot of one dimensional functions.

The properties of each line, like style (solid, dashed, dotted, etc), color and width, can be modified in the instruction `plot`. For example, if you modify the arguments of the `plot` command in the following way you will get a dashed, blue, and thick line for the sine function in Fig. 1:

```
>> plot(x,sin(x),'--','color',[0 0 1],...
'linewidth',4)
```

The color is specified by means of the corresponding RGB code (with each of the three numbers going from 0 to 1).

IV. WORKSPACE (`.MAT`) AND SCRIPTS (`.M`) FILES

You can save the data appearing in the Workspace using the command `save('data.mat')`, where `data` is an arbitrary name given to the file and `.mat` the extension. You can load any `.mat` file into the Workspace using the command `load('data.mat')`. To delete the current data in the Workspace you need to use the command `clear`. All saved and loaded `.mat` files must be in the Current Folder.

You can also save a set of commands in a script file and run all of them with a single enter. To open a new script file, arbitrarily named `load('script1')`, you must type `edit('script1.m')`. MATLAB will ask you if you want to create the corresponding file. After clicking "Yes", a new tab will open in a panel called Editor. You will also see your new file in the Current Folder. To run your script file, you just need to use the command `run('script1.m')`. Every time that you run your script, MATLAB saves automatically your script file. If you want to insert comments in your script file, you must use the percentage symbol (`%`) before your comment (the same as in LaTeX).

For example, the set of commands to get the 2D plot in Fig. 1 are the following:

```
clear
close all
x = 0:0.01:2*pi;
plot(x,sin(x))
hold on
plot(x,cos(x))
plot(x,tan(x))
hold off
legend('sin','cos','log')
xlabel('x')
ylabel('f(x)')
title('Trigonometric functions')
set(gca,'fontsize',18)
saveas(gca,'2Dplot.png')
```

It is recommended to start our script with the commands `clear` and `close all` in order to start from scratch our calculations and close any previous figure window.

V. 3D PLOTS

There are different commands to plot 3D functions. However, the more general one is `surf`. The following script is an example on how to plot a 3D plot using this command, it also shows a top view of the same plot as obtained by the command `pcolor`:

```
clear
close all
Z = peaks(50);

surf(Z)
hold on
h = pcolor(Z)
```

```
hold off
set(h, 'ZData',-10 + 0*Z)
set(gca,'fontsize',18)
saveas(gca,'3Dplot.png')
```

The resulting 3D plot is shown in Fig. 2.

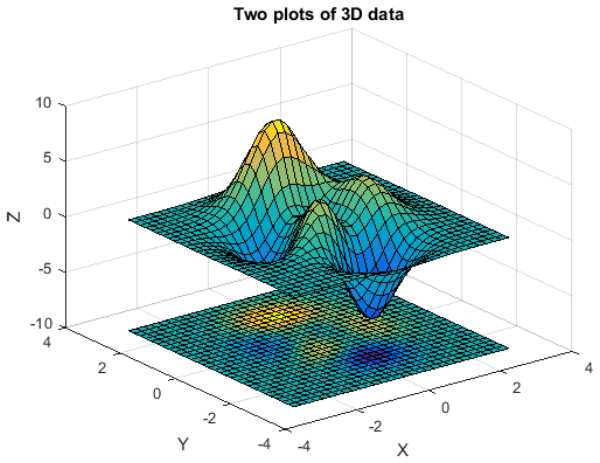

FIG. 2. A 3D plot using with a top view of itself.

VI. SOLVING SYSTEMS OF LINEAR EQUATIONS

Let's try to solve the following system of linear equation:

$$\begin{aligned} x + 2y - 3z &= 5 \\ 3x - y + z &= -8 \\ x - y + z &= 0 \end{aligned} \tag{1}$$

Note that this system of equations can be written as

$$A\mathbf{x} = \mathbf{b}, \tag{2}$$

with

$$A = \begin{bmatrix} 1 & 2 & -3 \\ 3 & -1 & 1 \\ 1 & -1 & 1 \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 5 \\ -8 \\ 0 \end{bmatrix}. \tag{3}$$

To solve this system of linear equation in MATLAB, we first define A and $\mathbf{b}$, and then introduce the command

```
>> x = A\b
```

MATLAB gives the the solution to the system of linear equation as a column vector. Note that we have used a backslash (\) in contrast to the usual slash (/).

VII. FIT NONLINEAR REGRESSION MODEL

You can fit experimental data to nonlinear functions in MATLAB. In the following example, we fit the voltage of a capacitor as a function of time according to the expression

$$V(t) = V_0 e^{-t/RC}, \tag{4}$$

with V_0 and RC the initial voltage in the capacitor and the characteristic time of the RC circuit. In MATLAB, we introduce these constants as the elements of a row vector `b`.

```
clear
close all

data = load('ExperimentalData1.txt');
x = data(:,1);
y = data(:,2);

modelfun = @(b,x) b(1)*exp(x/b(2));
b = [5 -1];
mdl = fitnlm(x,y,modelfun,b);

plot(x,y,'ob');
hold on;
plot(x,predict(mdl,x),'-b','linewidth',1.5);
legend('data','fit');
xlabel('time (s)')
ylabel('voltage (V)')
set(gca,'fontsize',18)
saveas(gcf,'NonlinearFit1.png')
```

The resulting fit is shown in Fig. 3. You can extract the fitting parameters and their respective standard deviation via the commands `mdl.Coefficients.Estimate(1)` and `mdl.Coefficients.SE(1)`, respectively, where the number 1 refers to the first parameter (`b(1)`), for example. The R^2 value can be obtained via the command `mdl.Rsquared.Ordinary`.

VIII. NUMERICAL DIFFERENTIATION AND INTEGRATION

In MATLAB we can numerical differentiate a function using the command `diff`. First, we define a vector `x` with the independent variable values, and then evaluate the function on these values to get a second vector `y`. The derivative of the function is then obtained as `diff(y)./diff(x)`. For example,

```
clear
close all
x = 0.1:0.01:10;
y = log(x);
derivative = diff(y)./diff(x);
```

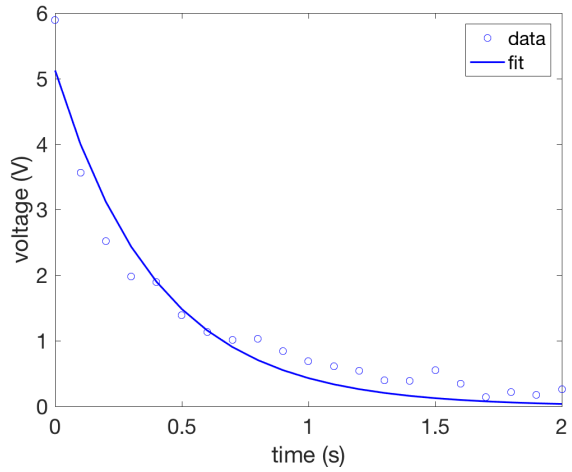

FIG. 3. Nonlinear fit to the voltage of a capacitor in an RC circuit.

```
plot(x,y,'-b','linewidth',1.5)
hold on
plot(x(2:end),derivative,'--r','linewidth',1.5)
hold off
legend('log(x)','derivative');
xlabel('x')
ylabel('y')
set(gca,'fontsize',18)
saveas(gcf,'Differentiation.png')
```

The resulting plot is shown in Fig. 4. Since the command `diff` calculates differences between adjacent elements of `x` and `y`, the variable `derivative` has one fewer element than `x` or `y`. Therefore, we plot `derivative` starting at the second value of `x`.

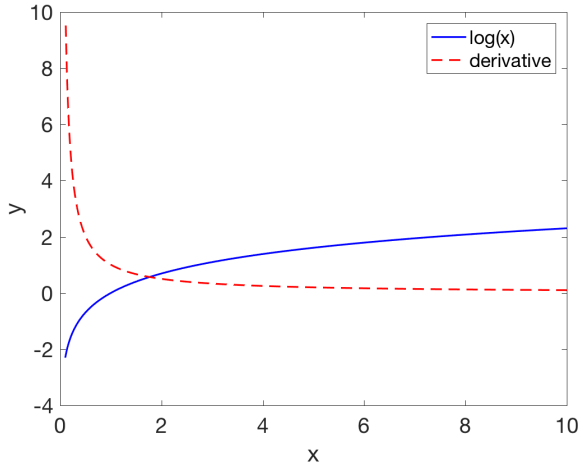

FIG. 4. Derivative of the function $\log(x)$.

You can also plot the analytic result for the derivative of $\log(x)$, $1/x$, and observe the discrepancy between the numerical and the analytic result. Such a discrepancy can be reduced by increasing the number of elements in the vector `x` within the same interval, i.e. by reducing the step.

There are two different commands to numerically calculate definite integrals in MATLAB, depending on whether we are dealing with analytic functions or vectors. In the first case, we use the command `integral` as shown below:

```
f = @(x) sin(x) ./ x
integral(f,-1,1)
```

The numerical result is 1.8922. We can use the command `format long` to display more decimal digits for the integral result.

If we want to integrate a vector instead of a function, we use the command `trapz`. For example, let's find the integral for the experimental data in Fig. 3:

```
data = load('ExperimentalData1.txt');
x = data(:,1);
y = data(:,2);
trapz(x,y)
```

The result is 2.2497.

IX. DIFFERENTIAL EQUATIONS

To solve ordinary differential equations (ODEs) in MATLAB you have several commands or solvers depending on the specific problem. However, the `ode45` is the most general and versatil one. For example, let's suppose we want to find the motion of a particle given by the following system of ODEs:

$$\begin{aligned} x' &= -x - 2y + z \\ y' &= x - 2y + 3z \\ z' &= x - y + z. \end{aligned} \tag{5}$$

Each of the three spatial coordinates x, y and z is a function of time t. We define this system of ODEs by means of a function `f` depending on two variables `t` and `x`, where `x` is now a vector with three components `x(1)`, `x(2)` and `x(3)`, referring to the three spatial coordinates x, y and z. This is,

```
f = @(t,x) [-x(1)-2*x(2)+x(3);...
            x(1)-2*x(2)+3*x(3);...
            x(1)-x(2)+x(3)];
```

The numerical solution in the interval $t \in (0, 10)$ with the initial conditions $x(0) = 1$, $y(0) = 1/2$ and $z(0) = 3$ is

```
[t,x] = ode45(f,[0 10],[0 1/2 3]);
```

Note that `x` is a three-column matrix, each of the columns describing `x(1)`, `x(2)` and `x(3)`, respectively. Also note that `t` is a column vector with the same number of rows as `x`. Let's plot the result for the three dependent variables:

```
clear
close all
plot(t,x(:,1),'-b','linewidth',1.5)
hold on
plot(t,x(:,2),'--r','linewidth',1.5)
plot(t,x(:,3),':g','linewidth',2)
hold off
legend('x(t)','y(t)','z(t)');
xlabel('t')
ylabel('spatial coordinates')
set(gca,'fontsize',18)
saveas(gcf,'ODEs2Dplot.png')
```

The resulting plot is shown in Fig. 5.

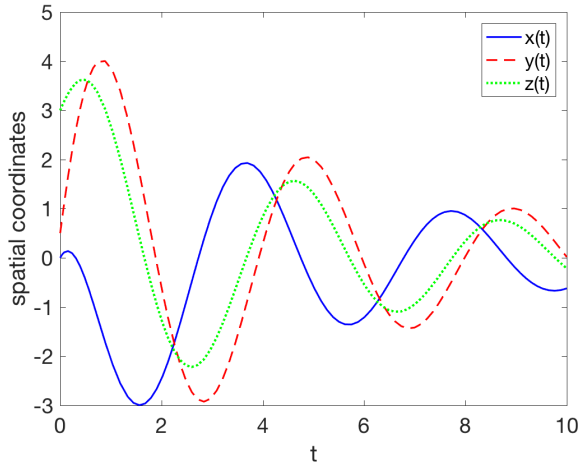

FIG. 5. Solution to the ODE system in Eq. (5).

We can plot the resulting trajectory for the particle in a 3D plot using the command `plot3`:

```
plot3(x(:,1),x(:,2),x(:,3),'-b','linewidth',1.5)
grid on
xlabel('x(t)')
ylabel('y(t)')
zlabel('z(t)')
set(gca,'fontsize',18)
saveas(gcf,'ODEs3Dplot.png')
```

Fig. 6 shows the 3D particle trajectory.

If you prefer animations, you can use the following script to visualize the particle trajectory and export it to a `.avi` file:

```
v = VideoWriter('Animation.avi');
v.FrameRate = 10;
open(v);
```

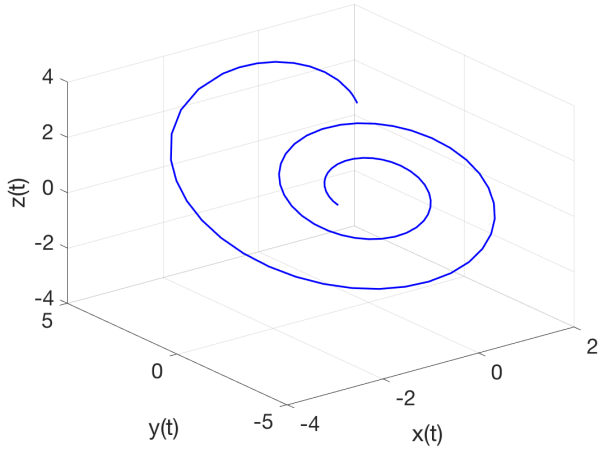

FIG. 6. Solution to the ODE system in Eq. (5).

```
for tau=1:size(t)
    plot3(x(tau,1),x(tau,2),x(tau,3),'ob',...
        'markerfacecolor', 'b')
    axis([-4 2 -5 5 -4 4]);
    grid on
    xlabel('x(t)')
    ylabel('y(t)')
    zlabel('z(t)')
    set(gca,'fontsize',18)
    frame = getframe(gcf);
    writeVideo(v,frame)
end

close(v);
```

After running your script, the video will appear in your MATLAB folder as a .avi file.

Note that we have used the loop `for` for the very first time. Its purpose is to repeat a specified number of times the instructions within the `for` and the respective `end` command. In the particular case of the previous script, the repetition number is given by the number of possible values that the variable `tau` can take, ranging from 1 up to the size of the variable `t` (in unit steps). Also, note that `tau` is used to retrieve the position coordinates $(x(t), y(t), z(t))$ at each time t.

X. EXERCISES.

Exercises are due on Wednesday October 14th at 11:59 pm. Write your answers in a latex file (e.g. https://www.overleaf.com/read/sqmzqbfkpfrf) and include your plots. Send only the pdf and your MATLAB codes (`.m` file) to amart224@uottawa.ca. The MATLAB code should be readable and easy to ejecute.

EXERCISE 0.

Create a random vector $\mathbf{x}$ of dimension 100. Create a new vector $\mathbf{y}$ whose i^{th} entry is given by $\mathbf{y}_i = \mathbf{x}_i * \mathbf{x}_{i+1}$. Note that the dimension of $\mathbf{y}$ is 99.

EXERCISE 1. REGIONS ON A PLANE.

Pick your favourite **non-degenerate** conic section (ellipse, hyperbole or parabola).

1. What is the equation that describes your conic?
2. Your conic divides the plane in an outer and inner region. Which region do you get when the equality sign is changed for an inequality?
3. Using the command `"imagesc"`, show your conic and the inner and outer regions of the plane created by it. Give your conic some width in the plot such that it is visible, e.g., if your figure were a circle draw a ring.
4. Label, edit and save your plots.

EXERCISE 2. DIFFRACTION OF A PLANE WAVE BY A CIRCULAR APERTURE.

In this exercise you will visualize the Fraunhofer (also known as far field) diffraction pattern of a plane wave, at wavelength λ, after passing by a circular aperture of radius ω. Such diffraction pattern is known as an Airy pattern and it is given by the following equation

$$I(x,y) = \left(\frac{\omega^2}{\lambda z}\right)^2 \left(\frac{J_1\left(2\pi \frac{\omega}{\lambda z}\sqrt{x^2+y^2}\right)}{\frac{\omega}{\lambda z}\sqrt{x^2+y^2}} \right)^2 . \qquad (6)$$

Where (x, y) are the coordinates of the observation plane, and z is the distance between the aperture and the observation plane. J_1 is a Bessel function of the first kind implemented in matlab with the command `"besselj"`.

1. Define in MATLAB the parameters involved in Eq. 6. Suppose $\omega = 1$ mm, a He:Ne laser i.e., $\lambda = 0.633$ µm and $z = 50$ m (which assures the far field condition).
2. Plot the Airy pattern in two dimensions (x, y). Choose a good window size to appreciate the maxima and minima. You should get a typical image of an Airy pattern e.g., https://en.wikipedia.org/wiki/Airy_disk.
3. The Airy pattern reaches its maxima at the center i.e., $(x, y) = (0, 0)$, plot the intensity at $y = 0$. You are plotting a function of one variable, so the command `'plot'` gives a good plot.
4. Label, edit and save your plots.

EXERCISE 3. NONLINEAR FITTING FROM A LASER BEAM TRANSVERSE PROFILE.

1. Import the file "Beam1.png" to your MATLAB folder, this image is a record of the 2D intensity $I(x, y)$ of a laser beam. You can download it from https://drive.google.com/drive/folders/1lWYdt61tDzPuQTFxKXAj9kj7Svof0gca?usp=sharing.
2. Which type of image is it (use the command `"image"` to visualize it)? Display the 2D image with colors with a colormap different from the default one of matlab. Use the matlab manual "help colormap" for details of the command.
3. 1D fit. Integrate the image along the x direction to obtain the intensity along y: $I(y) = \int_{-\infty}^{\infty} I(x, y)dx$. Normalize $I(y)$ to its maximum value, such that the maximum value now is equal to one.

   A Gaussian function is given by the equation

   $$G(x) = A \exp\left(-\frac{(x - x_0)^2}{2\sigma^2}\right), \qquad (7)$$

   where x_0 is the center of the distribution, σ its standard deviation and A is a constant that is unimportant for our purposes.
4. Fit a Gaussian function to $I(y)$. Add a constant to account for the background of the image.
5. Plot the experimental data and the fitted function in the same plot. Add "position (a.u.)" and "Intensity (a.u.)" as your x and y axis labels. Do not forget a legend for your plot.
6. What is the center of your fitted Gaussian? What is its standard deviation?
7. Perform a fitting for $I(x) = \int_{-\infty}^{\infty} I(x, y)dy$. Plot your fitting and experimental data in the same plot.
8. Label, edit and save your plots.
9. How could you experimentally modify the width of the beam in one direction only?

Parameter	Value	Std deviation
A		
x_0		
σ		

TABLE II. Fitting parameters.

EXERCISE 4. SOLVING THE HEAT EQUATION WITH BOUNDARY CONDITIONS.

When solving partial differential equations it is necessary to implement appropriate numerical methods. In this exercise you will solve the heat equation in one dimension with a given numerical method.
The boundary problem of the heat flow in one dimension is represented by a scalar function $u(x,t)$ that satisfies the following partial differential equation

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial^2 x}, \tag{8}$$

subject to the initial condition

$$u(x, t = 0) = f(x), \tag{9}$$

and boundary conditions

$$u(x = x_0, t) = g(t), u(x = x_f, t) = h(t). \tag{10}$$

The idea of the numerical method is to discretize space $\big(x_0, x_1 = x_0 + \Delta x, x_2 = x_0 + 2 * \Delta x, ..., x_j = x_0 + j * \Delta x, ..., x_f \big)$ and time $\big(t_0, t_1 = t_0 + \Delta t, ..., t_n = t_0 + n * \Delta t, ..., t_f\big)$. The solution is obtained at every point of time and space $u(x_j, t_n)$. In this exercise we calculate $u(x_j, t_{n+1})$ using adjacent points of space and time: $u(x_j, t_{n+1}) = u(x_j, t_n) + \frac{\Delta t}{\Delta x^2}\Big(u(x_{j+1}, t_n) - 2 * u(x_j, t_n) + u(x_{j-1}, t_n)\Big)$ where Δt and Δx are the mesh sizes in time and space respectively.

1. Get the template of a script from https://drive.google.com/drive/folders/1lWYdt61tDzPuQTFxKXAj9kj7Svof0gca?usp=sharing.

2. As parameters define $x_0 = 0$, $x_f = 1$, $g(t) = -1$, $h(t) = 0$, $f(x) = sin(\pi x) + sin(2\pi x) + sin(\frac{\pi}{2}x) - 1$ and obtain the solution for t in $[0, 0.5]$.

3. Input your code in the template following the commented sections. The comments should guide you to implement the method.

4. Your final result is your final code and a three dimensional plot of $u(x,t)$.

EXERCISE 5.

We are receiving a message with $N = 100$ characters. The transmission channel added noise to the message by performing a permutation $\mathbf{p}$, of the same dimension N as the message. Assume that the channel was characterized and $\mathbf{p}$ is given (use the command 'randperm'), create a program to reorder the message into its original form. The program has to display the original received message (i.e., the permutted message), the reordering permutation and the corrected message.
For example, if $N = 4$ we should get the message 'hope'. The transmission line performs the permutation p = [4,2,1,3], therefore the receiver gets the message 'eoph'. To obtain the original message the receiver needs to perform the permutation $p^{'}$ = [3,2,4,1].
For this message report your reasoning to create the program.