Project 1:
Power series expansion of A cos(ωt)

Overview: In this project, you will apply calculus to rewrite a sinusoidal function of time as a truncated power series using the Taylor series expansion. Along the way, you will learn about robust methods of developing scripts. Once you are finished, you will also learn about the limitations of a power series expansion. It is **strongly** recommended that you complete each phase before moving on the next phase. This means you need to get started early.

You should read the **entire** assignment before starting the Phase 1, so that you know what to expect.

Consider the function $f(t) = 12\cos(40t)$, where the coefficient of t is the angular frequency in rad/s, and t is in seconds. We can rewrite this function as an infinite power series:

$$f(t) = \sum_{n=0}^{\infty} a_n \cdot t^n$$

Phase 1. Getting a relatively simple script working (as quickly as possible).

[20pts] Plot a truncated power series for $f(t)$, from $t = 0$ to $t = 0.2$s, starting with the first non-zero term, then the sum of the first two non-zero terms, etc., up to a sum of the first 6 non-zero terms.

DESIGN SPECIFICATIONS

1. You should find a general expression for the non-zero coefficients, a_n. (Include your hand calculation when you upload your solution.)
2. You should use an array for the coefficients (a_n). These coefficients should be output to the Command Window in a way that is relatively easy to read and understand, even though the range of values will be from "normal" to very large.
3. You should create 6 functions to plot. These should be defined as efficiently as possible **without** using a FOR loop. (Using a FOR loop will come in a later phase.)
4. The chart should have a meaningful title, a grid, and proper axis labels.
5. Change the vertical axis to cut off some of the very high and very low values, but still make sense in terms of the given function.
6. For now, you should not be concerned with font sizes or a legend. (These are coming next.)

For Spec #1, you should use a Taylor series expansion about $t = 0$, i.e.,

$$f(t) = \sum_{n=0}^{\infty} \frac{f^{(n)}(0)}{n!} \cdot t^n$$

where $f^{(n)}(0)$ is the nth derivative of $f(t)$ evaluated at $t = 0$. (DO NOT change the definition of n. You should end up with one general expression for n = even, and another for n = odd.)

For Specs #2 and #3, you should set up an array of n values corresponding to the non-zero coefficients. Note that there is an efficient way of creating the array of n values. Use dot operations to create the array of a_n values. Then, you should use the `format shortG` command near the beginning of the script, so that the output is meaningful for this wide range of values.

(continued)

Phase 1. (continued)

Also for Spec #3, you will need to think about an efficient way of defining the functions (without using a FOR loop). It will help to have the array of *n* values, as you will be using it three times, once for defining the coefficients, once for the using the coefficients in the six functions you are creating, and once for the exponent of *t* (also when you are creating the six functions).

For Spec #4, a meaningful title should include the function being approximated by the power series and the number of non-zero terms in the truncated series. The "context" should mention "Phase 1", so that this can be changed as you proceed through this project. You might need a multiline title.

For Spec #5, we are expecting the truncated series to eventually become something that looks like the given function, 12 cos(40*t*). Therefore, use the `AXIS` or `YLIM` command to choose Y limits from –15 to +15, or something similar. In other words, without changing the Y limits, it can be hard (or impossible) to tell whether your figure is correct. But with one of these commands, you should just start to see the last function looking like the given function, at least for the first 0.1s or so.

UPLOAD: Hand calculation, script, Command Window, one figure (with all 6 graphs).

Phase 2. Adding features to improve the output.

[10pts] Once you have a functioning script that is perfect, rename it (so that you can keep the old one), and add the following features to make the output much nicer…

A. Change the format of the coefficients to be easier to understand by making the output a table with two columns, one with a list of *n* values and the other with a list of coefficients, and appropriate column headings.

B. Switch to ms for the given time limits, and plot in ms, while still using seconds to calculate functions to plot.

C. Add a horizontal axis.

D. All of the graphs should be thicker, with the last graph noticeably thicker than the rest, so that it's easier to tell which one it is. The grid should be darker than the default, without being too dark.

E. Make the font sizes appropriate for the size of the chart.

F. Add a legend. Use the last "n" value in each truncated series to label each graph (with a short but meaningful phrase).

G. Position the legend so that it will NEVER overlap any of the plotted graphs.

For feature A, use the `TABLE` command, with appropriate "variable names" as column headings. Choose a meaningful name for the table.

For feature B, switch the beginning / ending times in LINSPACE to be in ms. You will still need an array in seconds to compute the terms of the power series, so think about how to do this efficiently.

(continued)

Phase 2. (continued)

For feature C, plot a black, horizontal line first (so that it's underneath everything), for example…

```
plot([0,200], [0,0], 'k', 'LineWidth', 1)
```

For features C and D, you will need to use the `HOLD ON` and `HOLD OFF` commands, so that you can have three separate `PLOT` commands, one for the horizontal axis, one for the first 5 functions, and one for the last function. To make the grid darker, use…

```
ax = gca; ax.GridAlpha = 0.5;
```

For feature E, it helps to change all of the font sizes first, using `ax.FontSize = 16`, then, use the `FontSize` parameter to change the font sizes for the title, legend, and axis labels. This will make the font size equal to 16 points for the numbers along the axes, and then you can set the rest of them to whatever you like, as long as it comes after this line of code. Typically, the axis labels and legend text would be about 20pt, and the title is the largest, about 24pt. (Scale these up or down for your figure.)

For feature F, there is a way to easily append text to a string using a + sign. For instance…

```
"n = " + n;
```

… returns a row array of useful text strings. (You should make it more meaningful than this.) You will also need to skip the x-axis you have plotted for feature C. To do this, define the two other plots, e.g., as `p1` and `p2`, then include these as a column array in your `LEGEND` command, that is…

```
p1 = plot(tms, f1, tms, f2, ...);
p2 = plot(tms, f6, ...);
legend([p1; p2], legendText, ...)
```

Note the semicolons in the first two lines.

For feature G, find a parameter for the legend that will put it just outside the frame.

Note that Spec #4 from Phase 1 must still be met, which means you should change the "context" to refer to "Phase 2". Also, update the brief description near the beginning of your script to refer to this phase.

UPLOAD: New script, Command Window, figure.

Phase 3. Making the script more robust and more general.

[10pts] Let's add a specification that will help make the script easier to generalize in a later phase. Let's also add a check that we have not broken anything that was working before…

6. You should not "hardwire" or "hardcode" anything.
7. You should check that the results have not changed.

As before, rename your script before you start editing, so that you can keep the old one.

For Spec #6, define variables for the amplitude and angular frequency of the given sinusoid (i.e., `A` and `w`, as these are given), the number of non-zero terms in the truncated power series, the limits for the time axis (now in ms), and the number of points to plot (or intervals).

Other parameters will depend on these, such as the maximum value of *n* (which depends on the number of non-zero terms in the truncated series), and the limits of the vertical axis (which depend on the amplitude of the given sinusoid).

(continued)

Phase 3. (continued)

The title should also be affected, that is, you should not hardwire the number of non-zero terms that are mentioned in the title, or the values of A and ω. You will need something like `SPRINTF` and `%g` to do this. (You may instead use `STRCAT` and `NUM2STR`, if you prefer.) Recall also that you should "continue" a line to the next line using three dots (…) when it won't fit on one line. Update the "context" as well.

For Spec #7, you can do a visual check that the figure looks the same as before and add a comment near the end of your script. In other words, the output from this phase should look like the output from Phase 2.

UPLOAD: New script, Command Window, figure.

Phase 4. Making the script more efficient and easier to scale to any number of terms.

[10pts] The section of code we are using to define the 6 functions to plot is inefficient and will not scale. That is, what if we wanted to look at 20 or 30 non-zero terms? I don't think anyone would want to deal with 20 or 30 lines of code, even if they just had to copy the previous line and edit it.

Therefore, let's change the 3rd specification…

3b. Use a `FOR` loop to create the 6 functions to plot.

Now that everything is written using parameters (having made the script more robust in Phase 3), it should be relatively straightforward to add an appropriate section to your script.

Let's use `f` to store the truncated series. Start by initializing `f` as an array of zeros that matches the dimensions of the time arrays. Then, the first line in your `FOR` loop would be something like…

```
f = f + [expression for the next term in the series];
```

The next line (within the `FOR` loop) should be a `PLOT` function, which is why you don't need to define 6 functions separately. That is, you define the function, then plot it and use it in the next iteration of the loop, when you redefine it. Remove all non-essential parts from the `FOR` loop, e.g., `HOLD ON` and `HOLD OFF` can be outside the `FOR` loop.

You will need to think about exactly how you are going to make the line thickness of the last function thicker than the rest. (And check the legend to make sure it's correct as well.)

I also recommend that you introduce a new variable, for example, `k`, that runs from 1 to the number of non-zero terms. You will use `k` multiple times within the `FOR` loop.

You won't be able to use `p1` and `p2` any more, but you can use an array to keep track of the plots, so that the legend is still correct, for instance…

```
p(k) = plot(tms, f, ...);
```

… then replace `[p1; p2]` in the `LEGEND` command with simply `p`, i.e., no brackets are needed. Make sure to initialize `p` as a column vector just before the `FOR` loop.

As a better check (Spec #7), compare the new code to the old code (from Phase 3). Specifically, construct an expression that should equal zero for the last function. In other words, don't remove the "inefficient" section of code that you needed for Phase 3. Instead, add a new section that uses a `FOR` loop, and use the last function in the new section to compare to last function in the old section.

UPLOAD: New script, Command Window, figure.

Phase 5. Letting the user define the parameters.

[10pts] To finish the script, we need some new specifications…

8. Allow the user to input parameters.
9. Make it relatively easy for someone to understand what they are inputting.
10. In your prompts, include units wherever appropriate.
11. Skip the check of "old" vs. "new" when the number of non-zero terms is not equal to 6.
12. Add a new calculation to compare the last function to the exact (given) function. A meaningful value is the average magnitude of the deviation. Include the value in the title, robustly.

It will become too tedious to convert all of the "givens" (from Phase 3) into user-defined variables, although, after completing this phase, you will be able to easily update your script, if desired. Instead, to complete this project, you should only allow the user to choose the number of points (or intervals) being plotted, the number of non-zero terms (currently equal to 6), and the beginning and ending times (currently equal to 0ms and 200ms). Make sure that the prompts are appropriate and that the output is easy to read. Suppress output for the INPUT statements.

For Spec #11, use an IF statement.

For Spec #12, define a function (the "average deviation") that should be an array of small numbers, if the approximation is a good one. Make each element of the array positive, so that negative values do not cancel positive values by accident when you sum them. Find the average, so that the result does not depend on the number of points or intervals used to plot. Choose an appropriate name, so that the output is meaningful. Update the title appropriately.

A visual check is again useful here. Update the "context". Comment at the end of the script.

UPLOAD: New script, Command Window, figure.

Phase 6. Understanding the Taylor series.

[20pts] Now that we have a robust, efficient code, with user-defined inputs, we can use it to explore the Taylor series.

(a) Change the number of non-zero terms until the last function "looks like" the given function over the first 200ms, by finding the smallest number of non-zero terms corresponding to an average magnitude of deviation less than 0.05. What is that number of non-zero terms? (Include the Command Window and corresponding figure.)

(b) Verify that the average magnitude of deviation does not change appreciably when you double the number of intervals being plotted. (Use the same number of non-zero terms as you used in (a). Include only the Command Window.)

(c) Imagine that the beginning time is –200ms instead of 0ms (but use the same ending time). Predict what you think will be the average magnitude of deviation, using the same number of non-zero terms as you used in (a) and (b). For instance, do you think the average deviation will be higher, lower, or the same as it was in (a) and (b)? Why do you think so?

(continued)

Phase 6. (continued)

(d) Check your prediction in (c) by running your script. What happened? Why do you think this happened? (Include the Command Window and figure.)

(e) Imagine that the beginning time is 0ms, the ending time is 400ms, and the time about which the Taylor Series is expanded is t_0 = 200ms (instead of t_0 = 0, as in Phase 1). What do you think the average magnitude of deviation will be? Why? (There is no output possible, as we have not set up this project to make t_0 a variable or a user input.)

(f) Keep the beginning time as 0ms and the ending time as 400ms, then run your script with the same number of non-zero terms as (a), (b), and (d). What happens during the second 200ms? Why do you think this happens? What is the significance of choosing t_0 = 0 to expand the Taylor Series in Phase 1? What happens the farther you get from t_0? (Include the Command Window and figure.)

(g) Change the number of non-zero terms until the first 400ms again "looks like" the given function. How many non-zero terms do you need now? (Include the Command Window and figure.)

Do not edit your script from Phase 5, except to update the context and description near the beginning, to update the title to refer to this phase, and to answer the questions above at the end of the script. Make sure to run it five times as needed to answer the questions. Make sure to make a prediction in part (c) **before** running the script in part (d).

UPLOAD: New script, answers to 7 questions, Command Window (with all 5 runs, including tables of coefficients), 4 figures.

Summary

Here is a table showing what you should upload to Moodle for each phase.

Phase	Hand calculation	Script?	Command Window?	Number of Figure(s)	Answers to Questions
1	Derivation of a_n	yes	yes	1	—
2	—	yes	yes	1	—
3	—	yes	yes	1	—
4	—	yes	yes	1	—
5	—	yes	yes	1	—
6	—	yes	yes	4	7