Week 8 Homework Assignments: MATLAB App Designer and Symbolic Math

Global Requirements

- All deliverables shall be added, committed, and pushed to your Week9 folder in your repository.
- Include your name and the names of anyone who assisted you in the following format:

```
% Student: Firstname Lastname
% Assisted by: Firstname Lastname, etc.
```

- Provide comments explaining each part of your code.
- Ensure proper functionality and usability of your MATLAB apps.
- Ensure inputs and outputs are handled properly and tested with different scenarios.
- For symbolic operations, use MATLAB's symbolic toolbox functions (e.g., sym, diff, int, etc.).

1. Function Plotter with Symbolic Expressions

Task

Create an app using MATLAB App Designer that takes a user-input symbolic expression and plots the function over a given range.

App Components:

1. Input Field for Symbolic Expression:
   - The user should be able to input a symbolic expression (e.g., `x^2 + 3*x + 2`).
2. Input Fields for Plot Range:
   - Two input fields to specify the start and end of the range for the x-axis.
3. Plot Button:
   - A button that, when pressed, generates and displays the plot of the symbolic expression over the specified range.
4. Axes Element:
   - An axes component where the plot of the symbolic function will be displayed.

Instructions:

1. Create the App:
   - Use MATLAB App Designer to design the user interface.
   - Use the sym function to convert user input into a symbolic expression.
   - Use `fplot` to plot the symbolic expression over the user-defined range.
2. Error Handling:
   - Ensure your app handles invalid symbolic expressions and displays an error message using a dialog box.
3. Testing:
   - Test the app with various expressions, including polynomials, trigonometric, and exponential functions.

Example Output

When running your app, the user should be able to input a function like x^3 + 2*x - 5, define the range as -10 to 10, and see the corresponding plot displayed on the axes.

Deliverables

```
1.  Submit the .mlapp file for your app (e.g., FunctionPlotter.mlapp).
2.  Include comments explaining each part of your code and UI components.
```

2: Symbolic Differentiation and Integration App

Task

Create an app using MATLAB App Designer that allows users to input a symbolic expression and choose whether to differentiate or integrate the expression symbolically.

App Components:

1. Input Field for Symbolic Expression:
   - The user should be able to input a symbolic expression (e.g., x^2 + 3*x).
2. Radio Buttons for Operation:
   - The user selects either "Differentiate" or "Integrate."
3. Variable Dropdown:
   - A dropdown menu to select the variable with respect to which the differentiation or integration will be performed (e.g., x, y).
4. Result Display:
   - Display the resulting symbolic expression after the chosen operation.
5. Plot Button:
   - A button to plot both the original and the derived/integrated function over a given range.

Instructions:

1. Create the App:
   - Use MATLAB App Designer to design the user interface.
   - Use diff for symbolic differentiation and int for symbolic integration.
   - Allow users to select the variable from a dropdown and apply the operation accordingly.
2. Plot the Functions:
   - Plot both the original function and the derived or integrated function on the same axes over a user-defined range.
3. Error Handling:
   - Display an error message if the user input is not a valid symbolic expression.
4. Testing:
   - Test the app with various symbolic expressions and variables, and ensure both differentiation and integration operations work correctly.

Example Output

The app should allow a user to input `x^2 + 2*x + 1`, choose "Differentiate", and display the result `2*x + 2`. The plot should show both the original function and its derivative.

Deliverables

1. Submit the `.mlapp` file for your app (e.g., `SymbolicDiffIntApp.mlapp`).
2. Include comments explaining each part of your code and UI components.

3. Symbolic Fourier Series Visualizer

Task

Create an app using MATLAB App Designer that computes and visualizes the Fourier series expansion of a user-defined periodic symbolic function.

App Components:

1. Input Field for Periodic Symbolic Function:
   - The user should be able to input a periodic function (e.g., sin(x) or square(x)).
2. Input Field for Fourier Series Terms:
   - An input field for the number of terms to include in the Fourier series expansion.
3. Result Display:

- Display the resulting Fourier series expression.

4. Plot Button:
   - A button to plot both the original function and its Fourier series approximation on the same axes over a user-defined range.
5. Axes Element:
   - An axes component to visualize the original function and the Fourier series approximation.

Instructions:

1. Create the App:
   - Use MATLAB App Designer to design the user interface.
   - Use fourier to compute the symbolic Fourier series.
   - Allow the user to specify the number of terms in the Fourier series approximation and the plot range.
2. Plot the Functions:
   - Plot both the original periodic function and its Fourier series approximation on the same axes.
3. Error Handling:
   - Display an error message if the input is not a valid periodic function or if the number of terms is invalid.
4. Testing:
   - Test the app with various periodic functions like sin(x), cos(x), and square(x).

Example Output

The user should be able to input a function like square(x) and request a Fourier series approximation with 10 terms. The plot should show both the original function and the approximation.

Deliverables

1. Submit the `.mlapp` file for your app (e.g., `FourierSeriesVisualizer.mlapp`).
2. Include comments explaining each part of your code and UI components.

Definition of Done

Your Week9 folder shall contain at minimum the following files:

- `FunctionPlotter.mlapp`
- `SymbolicDiffIntApp.mlapp`
- `FourierSeriesVisualizer.mlapp`

Ensure that each app is well-documented, functions correctly, and follows good coding practices.

Additional Instructions

- **Testing**:
  - While explicit test scripts are not provided, test each app by inputting various symbolic expressions.
  - Ensure that invalid inputs are handled gracefully with error messages.
- **Plotting**:
  - For assignments involving plots, ensure the plots are properly labeled with titles, axis labels, and legends where appropriate.
  - Use grid lines and appropriate line styles for clarity where needed.
- **Have Fun**:
  - There is no requirement on the appearance of your applications
  - Apply any layout, coloring, or basic user experience (UX) that you feel benefits the user.

Tips

- **Handling Errors with Symbolic Expressions**: Use `try-catch` to handle unknown input errors for symbolic expressions. For example:

```
try
    expr = sym(input);
    % Your code to process and plot the expression
catch
    errordlg('There was an error with your input, please try again.');
end
```

- **Use Error Dialogs**: Instead of printing errors to the console, use `errordlg` to show error messages to the user in a dialog box. This enhances the user experience in App Designer.

- **Debugging**: Use `disp` or `fprintf` statements to help debug and trace variable values or flow of execution. For example, you can print key variables before plotting:

```
fprintf('Plotting function: %s\n', char(expr));
```

- **The Fourier Series**:
The Fourier series approximates a periodic function as the sum of sine and cosine terms. A typical Fourier series representation for a periodic function

$$f(x)$$

with period

$$T$$

is:

$$f(x) \approx \frac{a_0}{2} + \sum_{n=1}^{N} \left(a_n \cos\left(\frac{2\pi n x}{T}\right) + b_n \sin\left(\frac{2\pi n x}{T}\right) \right)$$

Where:

$$a_n$$

and

$$b_n$$

are the Fourier coefficients.

$$N$$

is the number of terms (the approximation accuracy improves as

$$N$$

increases).

You can use MATLAB's symbolic capabilities to calculate the Fourier coefficients and sum them up to get the approximation. Look into using the `symsum` function to sum terms in a symbolic expression.

- **Using `matlabFunction`**: When working with symbolic expressions, you can't plot them directly using standard plotting functions. First, convert symbolic expressions into function handles using `matlabFunction`. For example:

```
f_handle = matlabFunction(expr);
```

This allows you to pass the symbolic expression to plotting functions like `fplot` or use it in other numerical computations. For instance:

```
fplot(f_handle, [x_start, x_end]);
```

- **Using `fplot` for Symbolic Functions**:
  The `fplot` function simplifies plotting symbolic expressions. It automatically handles the evaluation and range of the plot. If you've used `matlabFunction` to convert your symbolic expression into a function handle, you can pass it directly to `fplot`. For example:

  ```
  fplot(@(x) sin(x), [-pi, pi]);  % for predefined functions
  fplot(f_handle, [-10, 10]);     % for symbolic expressions
  ```

  The advantage of `fplot` is that it automatically adjusts the plot resolution and avoids the need to manually evaluate the function at discrete points.

- **Symbolic Summation with `symsum`**:
  The `symsum` function is handy for computing symbolic sums like those in Fourier series. For instance, if you want to sum terms from (n = 1) to (N), you can write:

  ```
  syms n x;
  series = symsum((1/n)*sin(n*x), n, 1, 10);  % Sum the first 10 terms of a Fourier-like series
  ```

  This will give you the symbolic representation of the sum, which you can convert to a function and plot.