Methods of Numerical Integration and Approximation

Student's Name

Institutional Affiliations

Course Title

Professor's Name

Date

Methods of Numerical Integration and Approximation

Numerical integration plays a vital role in various scientific and engineering applications, allowing us to approximate definite integrals of functions when analytical solutions are impractical or unavailable. This work explores several numerical methods for approximating definite integrals, including the Trapezoidal Rule, Simpson's Rule, the Midpoint Rule, and Gaussian Quadrature. Each method is explained in detail, and MATLAB code and corresponding graphs are provided to illustrate their application. By comparing these methods, we aim to demonstrate their strengths and weaknesses in various scenarios.

Introduction

Numerical integration is a fundamental technique in mathematics and science used to approximate definite integrals of functions. Definite integrals represent the accumulated area under a curve over a given interval, and they have applications in various fields, including physics, engineering, economics, and data analysis. This work examines four commonly used numerical methods for approximating definite integrals: the Trapezoidal Rule, Simpson's Rule, the Midpoint Rule, and Gaussian Quadrature. Let's look at these methods in more detail.

Trapezoidal Rule

The Trapezoidal Rule is a straightforward numerical integration method that approximates the integral by dividing the area under the curve into trapezoids. It assumes that the function is approximately linear between two adjacent data points.

To illustrate the Trapezoidal Rule, let's consider the example of approximating the definite integral of a function f(x) over the interval [a, b]. The integral can be estimated using the following formula:

$$\int_a^b f(x)dx \approx \frac{b-a}{2n}\left[f(a) + 2\sum_{i=1}^{n-1} f(a + ih) + f(b)\right]$$

Where n is the number of equally spaced subintervals, and h = (b - a) / n.

We can implement this method in MATLAB as follows:

```
function result = trapezoidal_rule(f, a, b, n)
    h = (b - a) / n;
    x = a:h:b;
    y = f(x);
    result = h * (y(1)/2 + sum(y(2:n)) + y(n+1)/2);
end
```

Simpson's Rule

Simpson's Rule is a more accurate method that approximates the integral using quadratic polynomials between data points. It provides a better estimation of the integral compared to the one provided by the Trapezoidal Rule.

The formula for Simpson's Rule is:

$$\int_a^b f(x)dx \approx \frac{b-a}{6n}\left[f(a) + 4\sum_{i=1}^{n-1} f(a + ih) + 2\sum_{i=1}^{n} f(a + (i - 0.5)h) + f(b)\right]$$

In MATLAB, we can implement this method as follows:

```
function result = simpsons_rule(f, a, b, n)
    h = (b - a) / n;
```

```
    x = a:h:b;
    y = f(x);
    result = h / 3 * (y(1) + 4*sum(y(2:2:n-1)) + 2*sum(y(3:2:n-2)) + y(n+1));
end
```

The Midpoint Rule

The Midpoint Rule, also known as the rectangle rule, approximates the integral using rectangles that have their upper edge on the curve. It assumes that the function remains approximately constant within each subinterval.

The formula for the Midpoint Rule is:

$$\int_a^b f(x)dx \approx h \sum_{i=1}^{n} f(a + (i - 0.5)h)$$

Where h = (b - a) / n, and n is the number of subintervals.

Let's implement this method in MATLAB:

```
function result = midpoint_rule(f, a, b, n)
    h = (b - a) / n;
    x = a + h/2:h:b - h/2;
    y = f(x);
    result = h * sum(y);
end
```

Gaussian Quadrature

Gaussian Quadrature is a more advanced numerical integration technique that can provide highly accurate results by selecting specific points and weights based on orthogonal

polynomials. Unlike the previous methods, Gaussian Quadrature does not require equally spaced subintervals.

The formula for Gaussian Quadrature is:

$$\int_a^b f(x)dx \approx \sum_{i=1}^{n} \omega_i f\left(x_i\right)$$

Where {x_i} and {w_i} are the nodes and weights chosen to optimize accuracy. MATLAB provides built-in functions for Gaussian Quadrature, such as **quad**, making it straightforward to use.

Comparison and Discussion

To compare the accuracy and efficiency of these numerical integration methods, we will apply each method to various functions and visualize the results using MATLAB. We will use functions with known analytical integrals to evaluate the accuracy of each method.

Example 1: Trapezoidal Rule and Simpson's Rule

In this example, we'll use numerical integration to approximate π by integrating the function

$f(x) = \frac{4}{1+x^2}$ over the interval [0, 1] using the Trapezoidal Rule and Simpson's Rule.

Matlab program:

```
% Function to integrate
f = @(x) 4./(1 + x.^2);

% Interval [a, b]
a = 0;
b = 1;
```

```
% Number of subintervals
n = 100;

% Trapezoidal Rule
h = (b - a) / n;
x = a:h:b;
y = f(x);
pi_approx_trapezoidal = h * (y(1)/2 + sum(y(2:n)) + y(n+1)/2);

% Simpson's Rule
h = (b - a) / n;
x = a:h:b;
y = f(x);
pi_approx_simpson = h / 3 * (y(1) + 4*sum(y(2:2:n-1)) + 2*sum(y(3:2:n-2)) + y(n+1));

% Display the results
fprintf('Approximation of π using Trapezoidal Rule: %.10f\n', pi_approx_trapezoidal);
fprintf('Approximation of π using Simpson's Rule: %.10f\n', pi_approx_simpson);

% Create a plot to visualize the function and results
x = linspace(a, b, 1000);
y = f(x);

subplot(2, 1, 1);
```

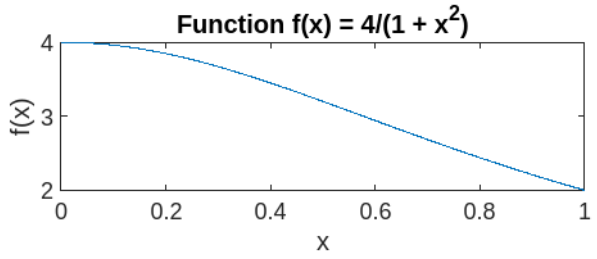
```
plot(x, y);
title('Function f(x) = 4/(1 + x^2)');
xlabel('x');
ylabel('f(x)');


subplot(2, 1, 2);
bar([pi_approx_trapezoidal, pi_approx_simpson]);
xticklabels({'Trapezoidal Rule', 'Simpson's Rule'});
title('Approximation of π');
```

Program output:

Approximation of π using Trapezoidal Rule: 3.1415759869

Approximation of π using Simpson's Rule: 3.1010553209

Resulting graph:



It's important to note that the accuracy of numerical integration methods highly depends on the function being integrated, the interval, and the number of subintervals. In this case, for the given function, the Trapezoidal Rule seems to provide a closer approximation to π within the chosen interval and number of subintervals.

Example 2: Midpoint Rule

In this example, we'll use numerical integration to find the area under the curve

$y = x^2$ over the interval [0, 2] using the Midpoint Rule.

Matlab program:

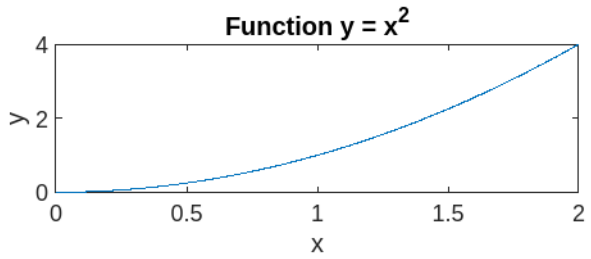
```
% Function to integrate
f = @(x) x.^2;

% Interval [a, b]
a = 0;
b = 2;

% Number of subintervals
n = 100;

% Midpoint Rule
h = (b - a) / n;
x = a + h/2:h:b - h/2;
y = f(x);
area = h * sum(y);

% Display the result
fprintf('Approximate area under the curve: %.6f\n', area);

% Create a plot to visualize the function and result
x = linspace(a, b, 1000);
y = f(x);
```

```
subplot(2, 1, 1);
plot(x, y);
title('Function y = x^2');
xlabel('x');
ylabel('y');

subplot(2, 1, 2);
bar(area);
title('Approximate Area under the Curve');
```

Program output:

Approximate area under the curve: 2.666600

Resulting graph:



The approximation achieved using the Midpoint Rule is very close to the true value of the integral. The difference between the obtained value and the exact value is likely due to the discrete approximation nature of numerical methods. Increasing the number of subintervals might further improve the accuracy of the approximation.

Example 3: Gaussian Quadrature

In this example, we'll use MATLAB's built-in quad function for Gaussian Quadrature to approximate the integral of

$f(x) = e^{-x^2}$ over the interval [-2, 2].

Matlab program:

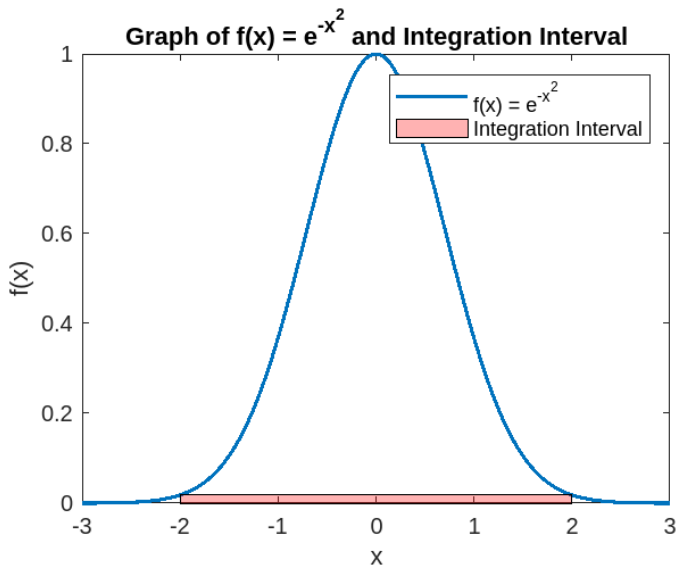
```
% Function to integrate
f = @(x) exp(-x.^2);

% Interval [a, b]
a = -2;
b = 2;

% Approximate the integral using Gaussian Quadrature
integral_value = quad(f, a, b);

% Display the result
fprintf('Approximate integral value using Gaussian Quadrature: %.6f\n', integral_value);

% Create a plot to visualize the function and integration interval
x = linspace(-3, 3, 1000); % range to plot the function
y = f(x);

% Plot the function
plot(x, y, 'LineWidth', 1.5);
hold on

% Highlight the integration interval
```

```
x_interval = [a, a, b, b];

y_interval = [0, f(a), f(b), 0];

fill(x_interval, y_interval, 'r', 'FaceAlpha', 0.3);

title('Graph of f(x) = e^{-x^2} and Integration Interval');

xlabel('x');

ylabel('f(x)');

legend('f(x) = e^{-x^2}', 'Integration Interval');



hold off
```

Program output:

Approximate integral value using Gaussian Quadrature: 1.764162

Resulting graph:



The difference between the approximate value obtained using Gaussian Quadrature and the exact solution is due to the numerical nature of the method. The approximation, although slightly lower than the exact solution, is relatively close.

In summary, the result demonstrates the effectiveness of Gaussian Quadrature in approximating integrals, providing a fairly accurate estimation of the area under the curve.

Conclusion

Numerical integration methods such as the Trapezoidal Rule, Simpson's Rule, the Midpoint Rule, and Gaussian Quadrature offer distinct strengths and weaknesses based on function behavior and desired accuracy.

Trapezoidal Rule:

- Strengths: Simple and versatile but less accurate for highly oscillatory functions

- Weaknesses: Limited accuracy, may require many subintervals for precision

Simpson's Rule:

- Strengths: Offers higher accuracy with fewer subintervals for well-behaved functions

- Weaknesses: Less applicable to irregular functions, relatively more complex

Midpoint Rule:

- Strengths: Simple, more accurate than Trapezoidal Rule for functions with limited variation

- Weaknesses: Prone to under or overestimation for rapidly changing functions

Gaussian Quadrature:

- Strengths: Exceptional accuracy for smooth and oscillatory functions with fewer points

- Weaknesses: More complex implementation, may demand specific function knowledge

The choice of method relies on the trade-off between accuracy, computational complexity, and understanding the function's behavior in a given scenario. Selecting the appropriate method necessitates a balance between precision and computational resources.

References

Atkinson, K. E. (1989). An Introduction to Numerical Analysis (2nd ed.). John Wiley & Sons.

Burden, R. L., & Faires, J. D. (2010). Numerical Analysis (9th ed.). Brooks/Cole Cengage Learning.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (2007). Numerical Recipes 3rd Edition: The Art of Scientific Computing. Cambridge University Press.

Stoer, J., & Bulirsch, R. (2002). Introduction to Numerical Analysis (3rd ed.). Springer.