

Problem 1.

In this problem we will use Taylor approximation to approximate the integral

$$I = \int_1^2 \frac{e^x - 1}{x} dx$$

Denote $f(x) = \frac{e^x - 1}{x}$.

Part A

Derive a formula for the n 'th Taylor polynomial around x_n called $p_n(x)$ of f . Use the summation symbol to write $p_n(x)$.

Solution

We would like to construct the power series, first

$$e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!} = 1 + \sum_{k=1}^{\infty} \frac{x^k}{k!} \implies e^x - 1 = \sum_{k=1}^{\infty} \frac{x^k}{k!}$$

Then divide by x to obtain

$$\frac{e^x - 1}{x} = \frac{1}{x} \sum_{k=1}^{\infty} \frac{x^k}{k!} = \sum_{k=1}^{\infty} \frac{x^{k-1}}{k!}$$

Which generates a series representation of f . To construct p_n , we truncate the function to obtain an n 'th degree polynomial.

$$p_n(x) = \sum_{k=1}^{n+1} \frac{x^{k-1}}{k!} = \sum_{k=0}^n \frac{x^k}{(k+1)!}$$

The reason we sum to $n+1$ is because when $k = n+1$, $k-1 = (n+1)-1 = n$ resulting with an n 'th degree polynomial. We can shift the indices (done in the right summation) to get a more compact answer. Both formats are acceptable.

Part B

Write the integral $I_n = \int_1^2 p_n(x) dx$ in sigma-notation format without the \int .

Solution

As our series is finite, we can first interchange the summation and integral operation (note: this may not hold for arbitrary infinite summations)

$$I_n = \int_1^2 p_n(x) dx = \int_1^2 \sum_{k=1}^{n+1} \frac{x^{k-1}}{k!} dx = \sum_{k=1}^{n+1} \int_1^2 \frac{x^{k-1}}{k!} dx = \sum_{k=1}^{n+1} \frac{x^k}{k(k!)} \Big|_{x=1}^{x=2} = \sum_{k=1}^{n+1} \frac{2^k - 1}{k(k!)} = \sum_{k=0}^n \frac{2^{k+1} - 1}{(k+1)((k+1)!)}$$

Both 0 or 1 based indexing are acceptable.

Part C

How large should n be such that I_n approximates I with error less than $\epsilon = 10^{-5}$.

Solution

Let $R_n(x)$ be the remainder of the n 'th Taylor expansion of $f(x)$:

$$f(x) = p_n(x) + R_n(x)$$

We have

$$I = \int_1^2 f(x)dx = \int_1^2 p_n(x)dx + \int_1^2 R_n(x)dx = I_n + \int_1^2 R_n(x)dx.$$

Thus,

$$I - I_n = \int_1^2 R_n(x)dx.$$

We want to find a bound for the quantity $|I - I_n|$ that only depends on n . By triangle inequality,

$$|I - I_n| = \left| \int_1^2 R_n(x)dx \right| \leq \int_1^2 |R_n(x)|dx. \quad (*)$$

The problem becomes how to bound the error term $|R_n(x)|$. It is difficult to apply Lagrange's theorem to function $f(x)$ to bound $R_n(x)$. The reason is that the derivatives f', f'', f''', \dots get complicated very quickly. Instead, we will work through the function $g(x) = e^x$ because we can easily apply Lagrange's theorem to this function. We have

$$g(x) = e^x = 1 + \underbrace{\frac{x}{1!} + \frac{x^2}{2!} + \dots + \frac{x^n}{n!}}_{q_n(x)} + \underbrace{\frac{x^{n+1}}{(n+1)!} + \dots}_{r_n(x)}$$

Here $q_n(x)$ is the n 'th Taylor polynomial of $g(x)$, and $r_n(x)$ is the corresponding error term. Let us subtract 1 from both sides:

$$e^x - 1 = \frac{x}{1!} + \frac{x^2}{2!} + \dots + \frac{x^n}{n!} + r_n(x).$$

Let us divide both sides by x :

$$\frac{e^x - 1}{x} = \frac{1}{1!} + \frac{x}{2!} + \dots + \frac{x^{n-1}}{n!} + \frac{r_n(x)}{x}.$$

This is a sum of an $(n-1)'$ st polynomial and an error term. Hence,

$$\begin{aligned} p_{n-1}(x) &= \frac{1}{1!} + \frac{x}{2!} + \dots + \frac{x^{n-1}}{n!}, \\ R_{n-1}(x) &= \frac{r_n(x)}{x}. \end{aligned}$$

Note that $R_{n-1}(x)$ is the error term that comes from f , while $r_n(x)$ is the error term that comes from g . They are related to each other by $R_{n-1}(x) = \frac{r_n(x)}{x}$. By replacing n by $n+1$, we can write

$$R_n(x) = \frac{r_{n+1}(x)}{x}.$$

Now we can use Lagrange's theorem for function g in order to write r_{n+1} . It says that

$$r_{n+1}(x) = \frac{g^{(n+2)}(c)}{(n+2)!} x^{n+2} = \frac{e^c}{(n+2)!} x^{n+2}$$

for some c in between 0 and x . Thus,

$$R_n(x) = \frac{r_{n+1}(x)}{x} = \frac{e^c}{(n+2)!} x^{n+1}.$$

Since we are integrating f over the interval $[1, 2]$, we only interested in $x \in [1, 2]$. Since c is in between 0 and x , it is in between 0 and 2. Thus,

$$|R_n(x)| = \frac{e^c}{(n+2)!} x^{n+1} \leq \frac{e^2}{(n+2)!} 2^{n+1} < \frac{9}{(n+2)!} 2^{n+1}.$$

We then substitute this number into (*) and get

$$|I - I_n| \leq \int_1^2 |R_n(x)| dx \leq \int_1^2 \frac{9}{(n+2)!} 2^{n+1} dx = \frac{9}{(n+2)!} 2^{n+1}.$$

In order for $|I - I_n|$ to be less than 10^{-5} , we only need to choose a large n such that

$$\frac{9}{(n+2)!} 2^{n+1} < 10^{-5}.$$

By calculator, we see that $n = 11$ will do it.

Part D

With a value of n found in part C, write a Matlab code to compute I_n . (Hint: use the `int` function to approximate I)

Solution

A simple code to compute I_n with $n = 11$ is as follows:

```
n = 11;
s = 0;
for k = 0:n
    s = s + (2^(k+1)-1)/(k+1)/factorial(k+1);
end
s
```

For those who are interested in learning more functions in Matlab, a more decorated code is as follows.

```
format long % Only need this once

% Now find the value for the integral using built in functions
f = @(x) (exp(x) - 1)./x;
integral_numerical = integral(f,1,2);
syms x
objective_function = (exp(x) - 1)/x;
% default behavior for int is to integrate wrt x if it finds x
integral_exact = int(objective_function, [1 2]);
disp(strcat('The symbolic integral result is I=', num2str(double(
    integral_exact),10)))
disp(strcat('The approximation using the integral function is I=', num2str
    (integral_numerical,10)))

num_samples = 12;

approximation_I = zeros(num_samples,1);
error_I = zeros(num_samples, 1);
for poly_degree = 1:num_samples
    approximation_I(poly_degree) = series_approx(poly_degree); %Get the
    approximation
    error_I(poly_degree) = abs(approximation_I(poly_degree) -
        integral_exact); % find the error
end

%Now we look at the vector of errors to find the first index
error_tol = 10^(-5);
[~, best_index] = max(error_I <= error_tol);
```

```

disp(strcat('The smallest n that satisfies the error bound is n=', num2str(
    (best_index)))
disp(strcat('The first I_n that satisfies the error bound is I_', num2str(
    best_index), '=', num2str(approximation_I(best_index),10)))

function sum_running = series_approx(degree)
sum_running = 0;
    for index = 1:degree
        sum_running = sum_running + (2^(index) - 1)./(index.*factorial(
            index));
    end
end

```

This prints to console the following output.

```

The symbolic integral result is I=2.365969359
The approximation using the integral function is I=2.365969359
The smallest n that satisfies the error bound is n=10
The first I_n that satisfies the error bound is I_10=2.365963868

```

As $10 \leq 11$, the bound we derived from the error formula is quite good.

Problem 2.

Let us consider the following toy model of the IEEE double precision floating point format. This toy model makes it simpler to demonstrate how addition and multiplication of floating-point numbers work. The sequence of 8 bits

$$c_0 b_1 b_2 b_3 b_4 a_1 a_2 a_3$$

represents a number $x = \sigma \bar{x} 2^e$ where σ, \bar{x}, e are determined as follows:

$$\sigma = \begin{cases} 1 & \text{if } c_0 = 0 \\ -1 & \text{if } c_0 = 1 \end{cases}$$

$$E = (b_1 b_2 b_3 b_4)_2$$

- If $1 \leq E \leq 14$ then

$$e = E - 7$$

$$\bar{x} = (1.a_1 a_2 a_3)_2$$

- if $E = 0$ then $e = -6$ and $\bar{x} = (a_1 a_2 a_3)_2$
- if $E = 15$ then $x = \pm\infty$ (depending on the sign σ)

Part A

Find the dynamic range of this floating point format.

Solution

The largest number M can be represented with $e = 7$ and $\bar{x} = 1.111$, so $M = 1.111_2 \times 2^7 \approx 2^8$. The smallest number m can be represented with $e = -7$ and $\bar{x} = 0.001$, so $m \approx 1 \times 2^{-10}$. Then the dynamic range D_r is $D_r = \frac{2^8}{1 \times 2^{-10}} = 2^{18}$.

Part B

What numbers are represented by the sequences 11001001, 00000000, 11111000?

Solution

11001001. We decompose this into the terms

- $c_0 = 1 \implies \sigma = -1$
- $E = 1001_2 \implies E_{10} = 8 + 1 = 9 \implies e = 2$
- $\bar{x} = 1.001_2$

Then we can construct the base 2 value of this bit-sequence as $-1.001_2 \times 2^2$.

0000000. We decompose this into the terms

- $c_0 = 0 \implies \sigma = 1$
- $E = 0000_2 \implies E_{10} = 0 \implies e = -6$
- $E = 0 \implies \bar{x} = 0.000_2$

Then we can construct the base 2 value of this bit-sequence as $0.000_2 \times 2^{-6}$.

11111000. We decompose this into the terms

- $c_0 = 1 \implies \sigma = -1$
- $E = 1111_2 \implies E_{10} = 8 + 4 + 2 + 1 \implies e = \infty$

Having obtained an $E = 15$ and $\sigma = -1$, we can stop here as it must be that $\bar{x} = -\infty$. We do not need to extract the values of \bar{x} from the bit sequence $a_1a_2a_3$ as the values of the bit sequence lack meaning in this context.

Problem 3.

There are only 256 different sequences of 8 bits. Thus, the sequence of 8 bits in problem 2 cannot represent precisely every real number. It can represent precisely only 242 real numbers and $\pm\infty$ (14 possible binary encodings). However any real number can be represented *approximately* by a bit sequence. The principle is simple: given a real number x we look for the number y among those 256 numbers that is closest to x . Then x is represented by the bit sequence that *exactly* represents y .

The method is as follows:

- Write x in binary form. For example, $6.3 = (110.010011001\dots)_2$.
- Shift the binary point to the form $1.c_1c_2c_3\dots_2$ by choosing an exponent $-6 \leq e \leq 7$ (an integer). For example $6.3 = (1.10010011001\dots) \times 2^2$.
- Round the mantissa to 3 digits after the binary point (the dot). For example $6.3 \approx (1.101_2) \times 2^2$.
- Decompose into pieces to find the values of \bar{x}, σ, e . Then construct the bit sequence encoding σ, e, \bar{x} according to the floating point format.

Note that in the second step, it may be impossible to choose an e that satisfies $-6 \leq e \leq 7$. If e is 'too big', $\pm\infty$ is recorded depending on σ . If $e < -6$, we can shift the decimal one digit to the left (and potentially lose one value of precision), from $(1.c_1c_2c_3\dots)_2$ to $(0.1c_1c_2\dots)_2$. The new exponent is $e + 1$. If $e = -6$ we can proceed. If $e < -6$ still, then we declare the number 'too close to zero' for this floating point format and thus is best approximated by 0.

Part A

Represent the decimal numbers 1, 5.5, 12.9, 1000, and 0.0001 in the floating point format $x = \sigma\bar{x}e^2$ and bit sequence described in problem 2.

Solution

1. $1 = 1.000_2 \times 2^0$. Here $e = 7, \sigma = 1$, and $\bar{x} = 1.000$. The bit sequence is 00111000.
2. $5.5 = 1.011_2 \times 2^2$. Here $E = 9, \sigma = 1$, and $\bar{x} = 1.011_2$. The corresponding bit sequence is 01001011.
3. $12.9 = 1.100111001100110011 \dots_2 \times 2^3$. As this contains more than 4 values in the binary expansion, we approximate this sequence with $12.9 \approx 1.1010_2 \times 2^3$. Here $E = 10, \sigma = 1$, and $\bar{x} = 1.010_2$. The corresponding bit sequence is 01010010.
4. $1000 = 1111101000_2 \times 2^0 = 1.111101000_2 \times 2^{10}$. Here $E = 10 + 7 = 17 > 14$. So as $\sigma = 1$ we approximate this value with $+\infty$. So $1000_{10} \approx \infty_2$. There are multiple corresponding bit sequences that we can use. There is some ambiguity in our choice of bit-wise encoding. Valid bit sequences are of the form 01111*** where * denotes either a 0 or a 1.
5. $0.0001 = 1.10100011011011100 \dots_2 \times 2^{-14}$. As this contains more than 4 values in the binary expansion, we approximate this sequence with $0.0001 \approx 1.101_2 \times 2^{-14}$. As $-14 + 1 < -6$, it is not possible to increment the exponent ($-14 \rightarrow -13$) to fit the format. $E < 0$ is required, so this number is *too close to 0* for this floating point format. So 0.0001 is approximately 0×2^{-13} , with the bit sequence 0000000. The last three bits $a_1 a_2 a_3$ are set to zero to ensure that when the bit sequence is converted back to a binary number we do not 'acquire' any new values.

Part B

Find the smallest number larger than 5.5 that can be represented precisely by the floating point format in Problem 2. Repeat this for 12.9 and 100.25.

Solution

Our strategy here is to find the smallest floating point number larger than our target number (5.5, 12.9 or 100.25). To do this, we compute the approximation of our target number and round up to the 'next' floating point number (floating point numbers are finite in \mathbb{R} , therefore not dense). Then we will convert this number from binary to decimal. We will denote our target as x .

1. Let $x = 5.5$. From above, we know that the binary encoding of x is 101.1_2 . As 5.5 can be exactly represented in floating point format, we do not need to truncate the representation. To round up, we add 0.1_2 to 101.1_2 to obtain $110.0_2 = 110_2$. Then $y = 2^2 + 2^1 + 0 = 6$. 6 is the next floating point number.
2. Let $x = 12.9$. From above, we know that $x_2 \approx 1100.1110011 \dots_2 = 1.1001110011 \dots_2 \times 2^3$. Then the next smallest number is $1.101_2 \times 2^3 = 1101_2 = 1 + 4 + 8 = 13$. 13 is the next smallest floating point number that can be represented exactly in this floating point format.
3. Let $x = 100.25$. We compute $x = 1.10010001_2 2^6$. We round up to the next floating point number $1.101_2 \times 2^6 = 104$. 104 is the next smallest floating point number that can be represented exactly in this floating point format.