(1.22) Taylor series expansion of the function is:

$$f(x) = e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!} + \ldots \qquad (1.21)$$

Use Eq. (1.21) to calculate the value of e^{-2} for the following cases. Use decimal numbers with six significant numbers (apply rounding at each step). In each case calculate also the true relative error. Use MATLAB with format long to calculate the true value of e^{-2}.
(*a*) Use the first four terms. (*b*) Use the first six terms. (*c*) Use the first eight terms.

Solution

(a) Using the first four terms and 6 significant figures,

$$e^{-2} = 1 - 2 + 2 - \frac{8}{6} = 1 - 1.33333 = -0.33333$$

Before calculating the true relative error, note that the sign of the answer is completely wrong since $e^x > 0$ for all real values of x. Use of format long in MATLAB yields:

```
>> format compact
>> format long
>> exp(-2)
ans =
   0.13533528323661
```

Therefore, the true relative error is:

$$\text{True Relative Error} = \left| \frac{0.13533528323661 - (-0.33333)}{0.13533528323661} \right| = 3.46$$

or 346%!

(b) With the first 6 terms and retaining 6 significant figures,

$$e^{-2} = 1 - 2 + 2 - \frac{8}{6} + \frac{16}{24} - \frac{32}{120}$$
$$= 1 - 1.33333 + 0.666667 - 0.266667 = 0.0666700$$

which yields a true relative error of $\left| \frac{0.13533528323661 - 0.0666700}{0.13533528323661} \right| = 0.50737$ or less than 50.74%.

(c) With the first 8 terms and retaining 6 significant figures,

$$e^{-2} = 1 - 2 + 2 - \frac{8}{6} + \frac{16}{24} - \frac{32}{120} + \frac{64}{720} - \frac{128}{5040}$$
$$= 1 - 1.33333 + 0.666667 - 0.266667 + 0.0888889 - 0.0252968$$
$$= 0.130262$$

which yields a true relative error of $\left|\frac{0.13533528323661-0.130262}{0.13533528323661}\right|=0.03749$ or less than 3.75%. It can be clearly seen from the answers of parts (a), (b), and (c), that as more terms are retained, the better the accuracy.

1.25 Develop an algorithm for adding all prime numbers between 0 and a given number.

Solution

(1) Start with $sum = 0$. If the given integer $x = 1$, then $sum = 0$ and stop. If the given integer $x = 2$, then $sum = 1$ and stop.
(2) Given an integer $x > 2$, for $i = 2$ to $x - 1$, find out if i is a prime number using for example, the algorithm from problem (1.19).
(3) If i is a prime number, then set $sum = sum + 1$.
(4) Set $sum = sum + 3$ (accounting for the trivial prime numbers 1 and 2), and stop.

1.30 Write a MATLAB program in a script file that determines whether or not a given integer is a prime number by implementing the algorithm developed in Problem 1.24. The program should start by assigning a value to a variable x. When the program is executed, a message should be displayed that states whether or not the value assigned to x is a prime number. Execute the program with $x = 79$, $x = 126$, and $x = 367$.

Solution

The following MATLAB code implements the algorithm from Problem 1.24:

```
clear; clc;
x=input('please enter a value for x\n');flag=0;
if(x==1) flag=1;
end
if(x==2) flag=1;
end
if(x~=1 & x~=2)
    for i=2:x-1
        remainder=rem(x,i);
        if(remainder==0)
            flag=0;
            break
        else
            flag=1;
        end
    end
end
if(flag==1)
    fprintf('x=%5f IS a prime number\n',x);
end
if(flag==0)
    fprintf('x=%5f is NOT a prime number\n',x);
end
```

When executed for the numbers given in this problem, the following is displayed in the Command Window:

```
please enter a value for x
79
x=79.000000 IS a prime number
>>
please enter a value for x
126
x=126.000000 is NOT a prime number
>>
please enter a value for x
367
x=367.000000 IS a prime number
```

1.31 Write a user-defined MATLAB function that adds all prime numbers between 0 and a given number by implementing the algorithm developed in Problem 1.25. Name the function `sp = sumprime(int)`, where the input argument `int` is a number larger than 1, and the output argument `sp` is the sum of all the prime numbers that are smaller than `int`. Use the function to calculate the sum of all the prime numbers between 0 and 30.

Solution

The following MATLAB built-in function implements the algorithm from Problem 1.25:

```
function sp = sumprime(int)
sum=0; flag=0;
if(int==1)
    sum=0; flag=1;
end
if(int==2)
    sum=1; flag=1;
end
if(flag~=1)
    for i=2:int-1
%Check if i is a prime number
        for j=2:i-1
            remainder=rem(i,j);
            if(remainder==0)
                flag=0;
                break
            else
                flag=1;
            end
        end
        if(flag==1)
            sum=sum+i;
        end
    end
```

```
          117877
>>
```

1.39 The value of π can be calculated with the series:

$$\pi = 4\sum_{n=1}^{\infty}(-1)^{n-1}\frac{1}{2n-1} = 4\left(1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\frac{1}{9}-\frac{1}{11}+\ldots\right) \quad (1.1)$$

Write a MATLAB program in a script file that calculates the value of π by using n terms of the series and calculates the corresponding true relative error. (For the true value of π, use the predefined MATLAB variable `pi`.) Use the program to calculate π and the true relative error for:
(*a*) $n = 10$. (*b*) $n = 20$. (*c*) $n = 40$.

Solution

The MATLAB script file is given below:

```
clear all; clc;
n=input('Please enter the number of terms of the series desired:\n');
total=0;
for i=1:n
    total = total + (((-1)^(i-1))/(2*i-1));
end
num_pi=4*total; true_pi = pi;
true_relative_error = abs((true_pi - num_pi)/true_pi);
percent = true_relative_error*100;
fprintf('For n=%3i, the calculated value of pi is %9.5f\n',n,num_pi)
fprintf('The true relative error is %9.5e or %6.3f per-
cent\n',true_relative_error,percent)
```

When executed for the values given in the problem, the display in the Command Window is:

(*a*)
```
Please enter the number of terms of the series desired:
10
For n= 10, the calculated value of pi is   3.04184
The true relative error is 3.17524e-002 or  3.175 percent
```

(*b*)
```
Please enter the number of terms of the series desired:
20
```

```
For n= 20, the calculated value of pi is   3.09162
The true relative error is 1.59056e-002 or  1.591 percent
```

(*c*)

```
Please enter the number of terms of the series desired:
40
For n= 40, the calculated value of pi is   3.11660
The true relative error is 7.95650e-003 or  0.796 percent
```

As can be seen, the convergence of this series is extremely slow, and many more terms are required before accuracy can be obtained to three decimal places:

```
Please enter the number of terms of the series desired:
2000
For n=2000, the calculated value of pi is   3.14109
The true relative error is 1.59155e-004 or  0.016 percent
```

(1.40) The Taylor's series expansion for is:

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{(2n+1)}$$

where x is in radians. Write a user-defined function that determines sinx using Taylor's series expansion. For function name and arguments, use `y=sinTaylor(x)`, where the input argument x is the angle in degrees and the output argument y is the value for sinx. Inside the user-defined function, use a loop for adding the terms of the Taylor's series. If a_n is the nth term in the series, then the sum S_n of the n terms is $S_n = S_{n-1} + a_n$. In each pass, calculate the estimated error E given by $E = \left|\frac{S_n - S_{n-1}}{S_{n-1}}\right|$. Stop adding terms when $E \le 0.000001$. Use `sinTaylor` for calculating: (*a*) $\sin 65^o$. (*b*) $\sin 195^o$. Compare the values calculated using `sinTaylor` with the values obtained by using MATLAB's built-in `sind` function.

Solution

```
% Problem (1.40), second edition
function y=sinTaylor(x)
format long
xrad=x*pi/180; sum=0;
for i=1:100
    n=i-1;
    fact=factorial(2*n+1);
    sum=sum+(((-1)^n)*(xrad^(2*n+1))/fact);
    S(i)=sum;
    if i>=2
        E=abs((S(i)-S(i-1))/S(i-1));
        if E<=0.000001
            break
        end
    end
end
y=sum;
%*********************************************************************
function a=factorial(n)
fact=1; prod=1;
for i=1:n-1
    fact=prod*(i+1);
    prod=fact;
end
a=fact;
```

When executed, the program yields:

```
>> y=sinTaylor(65)
y =
```

```
  0.90630778621376
>> sind(65)
ans =
  0.90630778703665
>> y=sinTaylor(195)
y =
 -0.25881904793355
>> sind(195)
ans =
 -0.25881904510252
>>
```

As can be seen, the values compare very well with the MATLAB built-in function values, to the 8th decimal place.