MTH 374 Numerical Analysis

Cason Konzer

Homwork #1

Background Functions

machine_2_decimal

```
% function [output_decimal] = machine_2_decimal(input_machine)
% % This function converts a machine number to decimal.
% % input_machine = '0100000000111011100100010000000000000000000000000000000000000000';
%
% if isa(input_machine, 'double')
%     input_machine = num2str(input_machine);
% end
%
% binary_s = input_machine(1);
% binary_c = input_machine(2:12);
% binary_f = input_machine(13:64);
%
% s = binary_s;
% %disp(['s: ', s])
% %disp(' ');
%
% c = 0;
% for i = strlength(binary_c):-1:1
%     c = c + (str2double(binary_c(i)) * 2^(11-i));
%     %disp(['i: ', num2str(i), ', bin: ', num2str(binary_c(i)), ', c: ', num2str(c)])
% end
% %disp(' ');
%
% f = 0;
% for i = 1:strlength(binary_f)
%     f = f + (str2double(binary_f(i)) * (1 / 2^i));
%     %disp(['i: ', num2str(i), ', bin: ', num2str(binary_f(i)), ', f: ', num2str(f)])
% end
%
% output_decimal = (-1)^s * 2^(c-1023) * (1+f);
%
% end
```

chop_decimal

```
% function [output_chopped] = chop_decimal(input_decimal, k)
% % This function chops a decimal with k digits
%
```

```
% if isa(input_decimal, 'string')
%     input_decimal = str2double(input_decimal);
%     %disp('coverted to double');
% end
%
% if isa(input_decimal, 'char')
%     input_decimal = str2double(input_decimal);
%     %disp('coverted to double');
% end
%
% n = 0;
% %disp(n)
%
% while abs(input_decimal) >= 1.0000000000000000000000000000000000000
%     input_decimal = input_decimal / 10;
%     n = n + 1 ;
% %     disp(n);
% %     disp(input_decimal);
% end
%
% while abs(input_decimal) < 0.0999999999999999999999999999999999999
%     input_decimal = input_decimal * 10;
%     n = n - 1 ;
% %     disp(n);
% %     disp(input_decimal);
% end
%
% extra_digit = num2str(input_decimal,k+1);
%
% %output_chopped = [extra_digit(1:k+2), ' x 10^', num2str(n)];
% output_chopped = str2double(extra_digit(1:k+2)) * 10^n;
% end
```

round_decimal

```
% function [output_rounded] = round_decimal(input_decimal, k)
% % This function rounds a decimal with k digits
%
% if isa(input_decimal, 'string')
%     input_decimal = str2double(input_decimal);
%     %disp('coverted to double');
% end
%
% if isa(input_decimal, 'char')
%     input_decimal = str2double(input_decimal);
%     %disp('coverted to double');
% end
%
% n = 0;
% %disp(n)
```

```
%
% while abs(input_decimal) >= 1.000000000000000000000000000000000000000
%     input_decimal = input_decimal / 10;
%     n = n + 1 ;
% %     disp(n);
% %     disp(input_decimal);
% end
%
% while abs(input_decimal) < 0.0999999999999999999999999999999999999999
%     input_decimal = input_decimal * 10;
%     n = n - 1 ;
% %     disp(n);
% %     disp(input_decimal);
% end
%
% to_chop = input_decimal + 5*10^(-k-1);
% rounded_extra_digit = num2str(to_chop,k+1);
%
% %output_rounded = [rounded_extra_digit(1:k+2), ' x 10^', num2str(n)];
% output_rounded = str2double(rounded_extra_digit(1:k+2)) * 10^n;
% % end
```

calc_errors

```
% function [actual_error,absolute_error,relative_error] = calc_errors(input_p,input_pstar)
% % This function calculates actual, absolute, and relative errors in machine
% % operations.
%
% actual_error = input_p - input_pstar;
% absolute_error = abs(input_p - input_pstar);
% relative_error = (input_p - input_pstar) / input_p;
% end
```

arctan_series

```
% function [output_tan_approx] = arctan_series(input_x, input_N)
% % This function calculates the summation of n terms of the arctan series.
%
% output_tan_approx = 0;
%
% for n = 1:input_N
%     term = (-1)^(n+1) * input_x^(2*n-1) / (2*n-1);
%     output_tan_approx = output_tan_approx + term;
% end
%
% end
```

arctan_error

```
% function [output_error_lim] = arctan_error(input_x,input_N)
```

```
% % This function calculates worst case error for of n terms of the arctan
% % series evaluated at x.
%
% itter = 2*input_N + 1;
% output_error_lim = 4 * abs(input_x)^itter / itter;
%
% end
```

Start Assignment

Problem 1 (Q1.2.1)

a.

```
[actual_error,absolute_error,relative_error] = calc_errors(pi, 22/7)
```

```
actual_error =
  -0.001264489267350
absolute_error =
   0.001264489267350
relative_error =
     4.024994347707008e-04
```

b.

```
[actual_error,absolute_error,relative_error] = calc_errors(pi, 3.1416)
```

```
actual_error =
    -7.346410206832132e-06
absolute_error =
     7.346410206832132e-06
relative_error =
     2.338434996796174e-06
```

c.

```
[actual_error,absolute_error,relative_error] = calc_errors(exp(1), 2.718)
```

```
actual_error =
     2.818284590451192e-04
absolute_error =
     2.818284590451192e-04
relative_error =
     1.036788960197272e-04
```

d.

```
[actual_error,absolute_error,relative_error] = calc_errors(sqrt(2), 1.414)
```

```
actual_error =
     2.135623730952219e-04
absolute_error =
     2.135623730952219e-04
relative_error =
     1.510114022219229e-04
```

Problem 2 (Q1.2.6)

a.

```
p = 133 + 0.921
```

```
p =
     1.339210000000000e+02
```

```
p_star = round_decimal(round_decimal(133,3) + round_decimal(0.921,3),3)
```

```
p_star =
   134
```

```
[actual_error,absolute_error,relative_error] = calc_errors(p, p_star)
```

```
actual_error =
  -0.079000000000008
absolute_error =
   0.079000000000008
relative_error =
     5.899000156809443e-04
```

b.

```
p = 133 - 0.499
```

```
p =
     1.325010000000000e+02
```

```
p_star = round_decimal(round_decimal(133,3) - round_decimal(0.499,3),3)
```

```
p_star =
   133
```

```
[actual_error,absolute_error,relative_error] = calc_errors(p, p_star)
```

```
actual_error =
  -0.498999999999995
absolute_error =
   0.498999999999995
relative_error =
   0.003766009313137
```

c.

```
p = (121 - 0.327) - 119
```

```
p =
   1.673000000000002
```

```
p_star = round_decimal(round_decimal(round_decimal(121,3) - round_decimal(0.327,3),3) ...
     - round_decimal(119,3),3)
```

```
p_star =
     2
```

```
[actual_error,absolute_error,relative_error] = calc_errors(p, p_star)
```

```
actual_error =
  -0.326999999999998
absolute_error =
```

```
   0.326999999999998
relative_error =
   0.195457262402868
```

d.

```
p = (121 - 119) - 0.327
```

```
p =
   1.673000000000000
```

```
p_star = round_decimal(round_decimal(round_decimal(121,3) - round_decimal(119,3),3) ...
    - round_decimal(0.327,3),3)
```

```
p_star =
   1.670000000000000
```

```
[actual_error,absolute_error,relative_error] = calc_errors(p, p_star)
```

```
actual_error =
   0.003000000000000
absolute_error =
   0.003000000000000
relative_error =
   0.001793185893604
```

Problem 3 (Q1.2.21)

$y - y_0 = m(x - x_0)$

a.

at the x intercept,

$0 - y_0 = m(x - x_0)$,

thus $x = x_0 - \frac{y_0}{m}$.

We know $m = \frac{\text{rise}}{\text{run}} = \frac{y_1 - y_0}{x_1 - x_0}$,

thus $x = x_0 - \frac{(x_1 - x_0)y_0}{y_1 - y_0}$. Eq. 2

Last as $x_0(y_1 - y_0) - (x_1 - x_0)y_0 = x_0 y_1 - x_1 y_0$,

$x = \frac{x_0 y_1 - x_1 y_0}{y_1 - y_0}$. Eq. 1

b.

```
x0 = 1.31;
y0 = 3.24;
x1 = 1.93;
y1 = 4.76;

p1 = (x0*y1 - x1*y0)/(y1 - y0);
p2 = x0 - ((x1 - x0)*y0 / (y1 - y0));
```

```
%curious
[actual_error,absolute_error,relative_error] = calc_errors(p1, p2)
```

```
actual_error =
    -1.786765180256111e-16
absolute_error =
     1.786765180256111e-16
relative_error =
     1.543115382948396e-14
```

```
p1_star = round_decimal( ...
    round_decimal( ...
    (round_decimal( ...
    round_decimal(x0,3)*round_decimal(y1,3),3) - ...
    round_decimal(round_decimal(x1,3)* ...
    round_decimal(y0,3),3)),3)/ ...
    round_decimal((round_decimal(y1,3) - round_decimal(y0,3)),3),3);
p2_star = round_decimal( ...
    round_decimal(x0,3) - round_decimal( ...
    (round_decimal( ...
    round_decimal( ...
    (round_decimal(x1,3) - round_decimal(x0,3)),3)* ...
    round_decimal(y0,3),3)/round_decimal( ...
    (round_decimal(y1,3) - round_decimal(y0,3)),3)),3),3);
```

```
[actual_error,absolute_error,relative_error] = calc_errors(p1, p1_star)
```

```
actual_error =
  -0.005078947368422
absolute_error =
   0.005078947368422
relative_error =
   0.438636363636387
```

```
[actual_error,absolute_error,relative_error] = calc_errors(p2, p2_star)
```

```
actual_error =
  -0.002578947368421
absolute_error =
   0.002578947368421
relative_error =
   0.222727272727293
```

Eq.2 is better as it substitutes a costly multiplication with a subtraction.

e.g. Eq.1. costs 2 multiplications, 2 subtractions and a division

but Eq.2. costs 1 multiplication, 3 subtractions and a division.

Problem 4 (Q1.3.1)

a.

```
sum = 0;
for i = 1:10
    sum = sum + (1/i^2)
end
```

```
sum =
     1
sum =
   1.250000000000000
sum =
   1.361111111111111
sum =
   1.423611111111111
sum =
   1.463611111111111
sum =
   1.491388888888889
sum =
   1.511797052154195
sum =
   1.527422052154195
sum =
   1.539767731166541
sum =
   1.549767731166541
```

```
sum_increasing = 0;
for i = 1:10
    sum_increasing = chop_decimal(sum_increasing + chop_decimal((1/i^2),3),3)
end
```

```
sum_increasing =
     1
sum_increasing =
   1.250000000000000
sum_increasing =
   1.360000000000000
sum_increasing =
   1.420000000000000
sum_increasing =
   1.460000000000000
sum_increasing =
   1.480000000000000
sum_increasing =
   1.500000000000000
sum_increasing =
   1.510000000000000
sum_increasing =
   1.520000000000000
sum_increasing =
   1.530000000000000
```

```
sum_decreasing = 0;
for i = 10:-1:1
    sum_decreasing = chop_decimal(sum_decreasing + chop_decimal((1/i^2),3),3)
end
```

```
sum_decreasing =
   0.010000000000000
sum_decreasing =
```

```
   0.022300000000000
sum_decreasing =
   0.037900000000000
sum_decreasing =
   0.058300000000000
sum_decreasing =
   0.086000000000000
sum_decreasing =
   0.126000000000000
sum_decreasing =
   0.188000000000000
sum_decreasing =
   0.299000000000000
sum_decreasing =
   0.549000000000000
sum_decreasing =
   1.540000000000000
```

The decreasing i method is more accurate as you start with smaller values which of which are continuously added as the summation continues.

This is akin to the rounding method.

b.

```
sum = 0;
for i = 1:10
    sum = sum + (1/i^3)
end
```

```
sum =
     1
sum =
   1.125000000000000
sum =
   1.162037037037037
sum =
   1.177662037037037
sum =
   1.185662037037037
sum =
   1.190291666666667
sum =
   1.193207118561710
sum =
   1.195160243561710
sum =
   1.196531985674193
sum =
   1.197531985674193
```

```
sum_increasing = 0;
for i = 1:10
    sum_increasing = chop_decimal(sum_increasing + chop_decimal((1/i^3),3),3)
end
```

```
sum_increasing =
     1
sum_increasing =
   1.120000000000000
```

```
sum_increasing =
   1.150000000000000
sum_increasing =
   1.160000000000000
sum_increasing =
   1.160000000000000
sum_increasing =
   1.160000000000000
sum_increasing =
   1.160000000000000
sum_increasing =
   1.160000000000000
sum_increasing =
   1.160000000000000
sum_increasing =
   1.160000000000000
```

```
sum_decreasing = 0;
for i = 10:-1:1
    sum_decreasing = chop_decimal(sum_decreasing + chop_decimal((1/i^3),3),3)
end
```

```
sum_decreasing =
     1.000000000000000e-03
sum_decreasing =
   0.002370000000000
sum_decreasing =
   0.004320000000000
sum_decreasing =
   0.007230000000000
sum_decreasing =
   0.011800000000000
sum_decreasing =
   0.019800000000000
sum_decreasing =
   0.035400000000000
sum_decreasing =
   0.072400000000000
sum_decreasing =
   0.197000000000000
sum_decreasing =
   1.190000000000000
```

The decreasing i method is more accurate as you start with smaller values which of which are continuously added as the summation continues.

This is akin to the rounding method.

Problem 5 (Q1.3.3)

a.

```
for n = 1:100000
    if arctan_error(1,n) < 10^(-3)
        disp(["n: " n ", max error " arctan_error(1,n)])
        break
    end
end
```

```
"n: "    "2000"    ", max error "    "0.00099975"
```

```
for n = 1:100000
    pi_star = 4*(arctan_series((1),n));
%     disp(pi_star)
    if abs(pi-pi_star) < 10^(-3)
        disp(["n: " n ", pi*: " pi_star ", absolute_error: " abs(pi-pi_star)])
        break
    end
end
```

```
"n: "    "1000"    ", pi*: "    "3.1406"    ", absolute_error: "    "0.001"
```

2000 (2 * 10^3) terms must be summed to guarentee the condition, although we meet the condition earlier.

b.

```
for n = 1:100000000000
    if arctan_error(1,n) < 10^(-10)
        disp(["n: " n ", max error " arctan_error(1,n)])
        break
    end
end
```

```
"n: "    "20000000000"    ", max error "    "1e-10"
```

```
% for n = 1:100000000000
%     pi_star = 4*(arctan_series((1),n));
% %     disp(pi_star)
%     if abs(pi-pi_star) < 10^(-10)
%         disp(["n: " n ", pi*: " pi_star ", absolute_error: " abs(pi-pi_star)])
%         break
%     end
% end
```

20000000000 (2 * 10^10) terms must be summed to guarentee the condition.

Problem 6 (Q1.3.6)

for

$h \approx 0, \sin(h) = O(h)$

a.

$\lim_{n\to\infty} \frac{1}{n} \approx 0$, thus $\sin\left(\frac{1}{n}\right) = O\left(\frac{1}{n}\right)$

b.

$\lim_{n\to\infty} \frac{1}{n^2} \approx 0$, thus $\sin\left(\frac{1}{n^2}\right) = O\left(\frac{1}{n^2}\right)$

Problem 7 (Q1.3.11)

$F_{n+1} = F_n + F_{n-1}$

$$\frac{F_{n+1}}{F_n} = 1 + \frac{F_{n-1}}{F_n} = 1 + \frac{1}{\frac{F_{n+1}}{F_n}}$$

$$x = 1 + \frac{1}{x}, x^2 - 1 - x = 0$$

$$x = \frac{-(-1) \pm \sqrt{(-1)^2 - 4(1)(-1)}}{2(1)} = \frac{1 \pm \sqrt{5}}{2}$$

as $n \geq 0, x = \frac{1 + \sqrt{5}}{2}$

Problem 8 (Q1.3.3)

```
% A = ["a0", "a1", "...", "an"]
A = [1, 3, 6]; % Given coefficients
x0 = 2; % Given x0
Px = 0;
for i = length(A):-1:2 % length A = n-1
    Px = (Px + A(i)) * x0
end
```

```
Px =
    12
Px =
    30
```

```
Px = Px + A(1)
```

```
Px =
    31
```