Assignment 1: Fun with Roundoff Error

Gregory Smetana
ID 1917370
ACM 106a

October 10, 2013

1 Evaluation of negative exponentials

The function e^x has Taylor series expansion

$$e^x = \sum_{n=0}^{d} \frac{x^n}{n!} + O(x^{d+1}) \tag{1}$$

a)

A truncated Taylor series with $d = 10, 100, 200, 1000$ was used to evaluate e^{-x} at $x = 1, 5, 10, 15, 20$. The results are displayed in Table 1.

x	$d = 10$	$d = 100$	matlab
1	3.678794642857144e-01	3.678794411714424e-01	3.678794411714423e-01
5	8.640390762786612e-01	6.737946999086907e-03	6.737946999085467e-03
10	1.342587301587302e+03	4.539992943405057e-05	4.539992976248485e-05
15	9.377044475446430e+04	3.059054877424794e-07	3.059023205018258e-07
20	1.859623680776014e+06	4.173637499437795e-09	2.061153622438558e-09

x	$d = 200$	$d = 1000$	matlab
1	3.678794411714424e-01	3.678794411714424e-01	3.678794411714423e-01
5	6.737946999086907e-03	NaN	6.737946999085467e-03
10	4.539992943405057e-05	NaN	4.539992976248485e-05
15	3.059054877424794e-07	NaN	3.059023205018258e-07
20	4.173637499437795e-09	NaN	2.061153622438558e-09

Table 1: Values of e^{-x} calculated with Equation 1, reference matlab exp(-x)

The Taylor series in Equation 1 is about $x = 0$, so it makes sense that the approximations become worse with increasing x. The approximation at $x = 1$ has at least 7 correct digits for all values of d. The approximation using 10 terms is terrible for other values of x, with no correct digits and values becoming very large. There are no significant differences between the $d = 100$ and $d = 200$ cases, and the approximations have least 5 correct digits until $x = 20$, when there are none. For values of $x > 1$, the $d = 1000$ case returns NaN.

The errors are caused by floating point naivete. Due to the negative value of x and the power of n in the numerator, terms in the series alternate sign. The subtraction process is a cause of floating point round-off error. Additionally, the $n!$ in the denominator increases extremely rapidly with d, which caused the erroneous NaN results.

There are n floating point operations in the computation of both $(-x)^n$ and $n!$, so each term requires $O(n)$ operations. The sum $\sum_{n=0}^{d} n$ leads to an estimate that the number of operations to approximate the function using Equation 1 is $O(d^2)$.

The matlab `tic-toc` function was used to compute the time required to compute the approximations using Equation 1, with the results displayed in Table 2. The times are mostly independent of x, which was expected. However, the results are not consistent with the analysis that doubling the value of d from 100 to 200 should quadruple the number of operations and time taken. This may be due to built-in matlab efficiencies. There also may be errors in the timing, as evidenced by the discrepancy with the $x = 1$, $d = 10$ case.

x	$d = 10$	$d = 100$	$d = 200$	$d = 1000$
1	2693	2446	5057	27284
5	287	2660	5229	25319
10	265	2444	4959	24977
15	263	2444	4915	25028
20	264	2460	5055	24920

Table 2: Time in microseconds required to compute the approximations using Equation 1

b)

The Taylor series may be computed without explicitly computing each term in the series or the coefficient $1/n!$. The modified expression may be written in a loop in matlab as Equation 2. This algorithm, which resembles Horner's method for polynomial evaluation, is mathematically equivalent to Equation 1. The resulting approximations are displayed in Table 3.

```
exp = 0;
for i=d:-1:1
    exp = (exp*x/i + 1);
end
```
(2)

The results of the new approximations at $x = 1$ have at least 7 correct digits for all values of d. The approximation using 10 terms is poor for other values of x, with no correct digits and nonsense large negative results. There is no significant differences between the cases of higher values of d, and the approximations have least 3 correct

x	$d = 10$	$d = 100$	matlab
1	3.678791887125220e-01	3.678794411714423e-01	3.678794411714423e-01
5	-1.827105379188711e+00	6.737946999083810e-03	6.737946999085467e-03
10	-1.413144620811287e+03	4.539992987351305e-05	4.539992976248485e-05
15	-6.513894419642857e+04	3.058273037037651e-07	3.059023205018258e-07
20	-9.622458077601411e+05	1.551753392448063e-09	2.061153622438558e-09

x	$d = 200$	$d = 1000$	matlab
1	3.678794411714423e-01	3.678794411714423e-01	3.678794411714423e-01
5	6.737946999083810e-03	6.737946999083810e-03	6.737946999085467e-03
10	4.539992987351305e-05	4.539992987351305e-05	4.539992976248485e-05
15	3.058273037037651e-07	3.058273037037651e-07	3.059023205018258e-07
20	1.551753392448063e-09	1.551753392448063e-09	2.061153622438558e-09

Table 3: Values of e^{-x} calculated with using Equation 2, reference matlab exp(-x)

digits until $x = 20$, when there are none. The $d = 1000$ cases do not return NaN, as in the part a).

The reason the accuracy has not improved is that the modification has not removed the key issue, which is the floating point subtraction that occurs during each step. However, by not explicitly computing each term or the coefficient $1/n!$, the algorithm is more robust and will not return NaN at higher values of d.

There are 3 floating point operations in each cycle of the loop of Equation 2. The loop is repeated d times, which leads to an estimate that the number of operations to approximate the function is $O(d)$.

The matlab `tic-toc` function was used to compute the time required to compute the approximations using Equation 2, with the results displayed in Table 4. The times are much faster than those in Table 2, which matches the expectation that the modified algorithm requires fewer operations. Doubling the value of d from 100 to 200 approximately doubles the amount of time required.

x	$d = 10$	$d = 100$	$d = 200$	$d = 1000$
1	656	5	3	14
5	0	2	3	14
10	1	1	3	13
15	1	2	4	14
20	0	2	3	14

Table 4: Time in microseconds required to compute the approximations using Equation 2

c)

An alternative method to computing e^{-x} is to consider the identity $e^{-x} = 1/e^{x}$. The results of using this identity with the functions from parts a) and b) are shown in Table 5 and Table 6

x	$d = 10$	$d = 100$	matlab
1	3.678794448678090e-01	3.678794411714423e-01	3.678794411714423e-01
5	6.831506312973185e-03	6.737946999085469e-03	6.737946999085467e-03
10	7.786764066679420e-05	4.539992976248486e-05	4.539992976248485e-05
15	2.582229688676605e-06	3.059023205018258e-07	3.059023205018258e-07
20	1.906406978839908e-07	2.061153622438558e-09	2.061153622438558e-09
x	$d = 200$	$d = 1000$	matlab
1	3.678794411714423e-01	3.678794411714423e-01	3.678794411714423e-01
5	6.737946999085469e-03	NaN	6.737946999085467e-03
10	4.539992976248486e-05	NaN	4.539992976248485e-05
15	3.059023205018258e-07	NaN	3.059023205018258e-07
20	2.061153622438558e-09	NaN	2.061153622438558e-09

Table 5: Values of e^{-x} calculated with Equation 1 and identity $e^{-x} = 1/e^{x}$, reference matlab exp(-x)

x	$d = 10$	$d = 100$	matlab
1	3.678794821625896e-01	3.678794411714423e-01	3.678794411714423e-01
5	6.959452863649537e-03	6.737946999085467e-03	6.737946999085467e-03
10	9.914169866623816e-05	4.539992976248485e-05	4.539992976248485e-05
15	4.379188111818032e-06	3.059023205018257e-07	3.059023205018258e-07
20	4.126093093478530e-07	2.061153622438557e-09	2.061153622438558e-09
x	$d = 200$	$d = 1000$	matlab
1	3.678794411714423e-01	3.678794411714423e-01	3.678794411714423e-01
5	6.737946999085467e-03	6.737946999085467e-03	6.737946999085467e-03
10	4.539992976248485e-05	4.539992976248485e-05	4.539992976248485e-05
15	3.059023205018257e-07	3.059023205018257e-07	3.059023205018258e-07
20	2.061153622438557e-09	2.061153622438557e-09	2.061153622438558e-09

Table 6: Values of e^{-x} calculated with Equation 2 and identity $e^{-x} = 1/e^{x}$, reference matlab exp(-x)

The results using the identity $e^{-x} = 1/e^{x}$ are much better than those of parts a) and b). In both Table 5 and Table 6, the results for $d = 10$ at $x > 1$ do not contain more than one correct digit, but the results do not become increasingly large. The values for the other values of d are quite good, correctly matching 14-15 digits. The $d = 1000$ case in Table 5 still features NaNs due to the use of the factorial, but the function used in part a) still seems to be slightly more accurate than that of part b).

The times to calculate the approximations using the identity and functions from parts a) and b) are displayed in Table 7 and Table 8. The times for all approximations match those of parts a) and b), which is expected since the number of operations remains the same.

x	$d = 10$	$d = 100$	$d = 200$	$d = 1000$
1	325	3608	4887	25374
5	263	2426	5097	25406
10	264	2436	5086	24999
15	262	2421	4878	24806
20	268	2434	4856	24755

Table 7: Time in microseconds required to compute the approximations using Equation 1 and identity $e^{-x} = 1/e^x$

x	$d = 10$	$d = 100$	$d = 200$	$d = 1000$
1	33	4	3	14
5	1	2	3	14
10	1	2	3	14
15	1	2	3	14
20	1	2	3	15

Table 8: Time in microseconds required to compute the approximations using Equation 2 and identity $e^{-x} = 1/e^x$

2 The Quadratic Formula

The roots of the quadratic polynomial $p(x) = ax^2 + 2bx + c = 0$, $a \neq 0$ are given by

$$x_\pm = \frac{-b \pm \sqrt{b^2 - ac}}{a} \tag{3}$$

a)

Equation 3 was used directly to calculate the root x_+ with the coefficients

$$(a, b, c) = (1, 10^k, 1), \quad k = 1, 3, 5, 7, 9 \tag{4}$$

The values for the roots are displayed in Table 9 and the value of the polynomial at the calculated roots are displayed in Table 10

k	x_+
1	-5.01256e-02
3	-5.00000e-04
5	-4.99999e-06
7	-5.02914e-08
9	0.00000e+00

Table 9: Roots calculated using Equation 3

As $b^2 >> ac$, the value in the numerator of Equation 3 approaches zero. However, by definition, the value of $p(x_+)$ should remain equal to zero.

The results of Table 9 and Table 10 are not correct. The value of x_+ should not actually equal zero, as it does for $k = 9$, which leads to the value of the polynomial becoming $p(0) = c = 1$. This is caused by increasing round-off error in the numerator as $b \to \infty$. The subtraction of similar, large numbers leads to errors in the evaluation of x_+, illustrated by the increasing value of $p(x_+)$ with k in Table 10.

k	$p(x_+)$
1	-2.44249e-15
3	-8.72711e-11
5	1.11668e-06
7	-5.82838e-03
9	1.00000e+00

Table 10: Value of $p(x)$ at the roots calculated at in Table 9

b)

To reduce the round off error by removing the catastrophic cancellation in the numerator, the equation may be manipulated by multiplication of 1,

$$x_+ = \frac{-b + \sqrt{b^2 - ac}}{a} \left(\frac{-b - \sqrt{b^2 - ac}}{-b - \sqrt{b^2 - ac}} \right) \tag{5}$$

$$x_+ = \frac{c}{-b - \sqrt{b^2 - ac}} \tag{6}$$

The values calculated for the roots using the modified formula are displayed in Table 11 and the values of the polynomial evaluated at the new roots are displayed in Table 12

k	x_+
1	-5.01256e-02
3	-5.00000e-04
5	-5.00000e-06
7	-5.00000e-08
9	-5.00000e-10

Table 11: Roots calculated using Equation 6

k	$p(x_+)$
1	1.11022e-16
3	0.00000e+00
5	0.00000e+00
7	-2.22045e-16
9	0.00000e+00

Table 12: Value of $p(x)$ at the roots calculated at in Table 11

The results using the modified formula Equation 6 are much better than those using Equation 3. The value of x_+ approaches zero as expected, but does not actually equal zero. The value of $p(x_+)$ remains small as k increases.

A Smetana_Gregory_1917370_A1_P1.m

```
clear all;
close all;
clc;

d = [10, 100, 200, 1000];
x = [-1,-5, -10, -15, -20];
%% part a)
approx_a = zeros(5,4);
tictoc_a = zeros(5,4);

for i = 1:5
    for j = 1:4
        tic
        approx_a(i,j) = exp_p1a(x(i), d(j));
        tictoc_a(i,j) = toc;
    end
end
tictoc_a=tictoc_a*10^6;
% generate table of approximations
latextable([cellstr(num2str(abs(x'))), num2cell([approx_a(:,1),approx_a(:,2),exp(x')])], ...
            'Horiz',{'$x$','$d=10$','$d=100$','matlab'},...
            'name','report/table1a_approx1.tex',...
            'format','%10.15e')

latextable([cellstr(num2str(abs(x'))), num2cell([approx_a(:,3),approx_a(:,4),exp(x')])], ...
            'Horiz',{'$x$','$d=200$','$d=1000$','matlab'},...
            'name','report/table1a_approx2.tex',...
            'format','%10.15e')

% generate table of times
latextable([cellstr(num2str(abs(x'))), num2cell(tictoc_a)], ...
            'Horiz',{'$x$','$d=10$','$d=100$','$d=200$','$d=1000$'},...
            'name','report/table1a_tictoc.tex',...
            'format','%4.0f')

%% part b

approx_b = zeros(5,4);
tictoc_b = zeros(5,4);


for i = 1:5
    for j = 1:4
        tic
        approx_b(i,j) = exp_p1b(x(i), d(j));
        tictoc_b(i,j) = toc;
    end
end
tictoc_b=tictoc_b*10^6;
```

```
% generate table of approximations
latextable([cellstr(num2str(abs(x'))), num2cell([approx_b(:,1),approx_b(:,2),exp(x')])], ...
            'Horiz',{'$x$','$d=10$','$d=100$','matlab'},...
            'name','report/table1b_approx1.tex',...
            'format','%10.15e')

latextable([cellstr(num2str(abs(x'))), num2cell([approx_b(:,3),approx_b(:,4),exp(x')])], ...
            'Horiz',{'$x$','$d=200$','$d=1000$','matlab'},...
            'name','report/table1b_approx2.tex',...
            'format','%10.15e')

% generate table of times
latextable([cellstr(num2str(abs(x'))), num2cell(tictoc_b)], ...
            'Horiz',{'$x$','$d=10$','$d=100$','$d=200$','$d=1000$'},...
            'name','report/table1b_tictoc.tex',...
            'format','%4.0f')

 %% part c

approx_ac = zeros(5,4);
tictoc_ac = zeros(5,4);

for i = 1:5
    for j = 1:4
        tic
        approx_ac(i,j) = exp_p1a(-x(i), d(j));
        approx_ac(i,j) = 1/approx_ac(i,j);
        tictoc_ac(i,j) = toc;
    end
end
tictoc_ac=tictoc_ac*10^6;

% generate table of approximations
latextable([cellstr(num2str(abs(x'))), num2cell([approx_ac(:,1),approx_ac(:,2),exp(x')])], ...
            'Horiz',{'$x$','$d=10$','$d=100$','matlab'},...
            'name','report/table1ac_approx1.tex',...
            'format','%10.15e')

latextable([cellstr(num2str(abs(x'))), num2cell([approx_ac(:,3),approx_ac(:,4),exp(x')])], ...
            'Horiz',{'$x$','$d=200$','$d=1000$','matlab'},...
            'name','report/table1ac_approx2.tex',...
            'format','%10.15e')

% generate table of times
latextable([cellstr(num2str(abs(x'))), num2cell(tictoc_ac)], ...
            'Horiz',{'$x$','$d=10$','$d=100$','$d=200$','$d=1000$'},...
            'name','report/table1ac_tictoc.tex',...
            'format','%4.0f')

approx_bc = zeros(5,4);
tictoc_bc = zeros(5,4);
```

```
for i = 1:5
    for j = 1:4
        tic
        approx_bc(i,j) = exp_p1b(-x(i), d(j));
        approx_bc(i,j) = 1/approx_bc(i,j);
        tictoc_bc(i,j) = toc;
    end
end
tictoc_bc=tictoc_bc*10^6;

% generate table of approximations
latextable([cellstr(num2str(abs(x'))), num2cell([approx_bc(:,1),approx_bc(:,2),exp(x')])], ...
            'Horiz',{'$x$','$d=10$','$d=100$','matlab'},...
            'name','report/table1bc_approx1.tex',...
            'format','%10.15e')

latextable([cellstr(num2str(abs(x'))), num2cell([approx_bc(:,3),approx_bc(:,4),exp(x')])], ...
            'Horiz',{'$x$','$d=200$','$d=1000$','matlab'},...
            'name','report/table1bc_approx2.tex',...
            'format','%10.15e')

% generate table of times
latextable([cellstr(num2str(abs(x'))), num2cell(tictoc_bc)], ...
            'Horiz',{'$x$','$d=10$','$d=100$','$d=200$','$d=1000$'},...
            'name','report/table1bc_tictoc.tex',...
            'format','%4.0f')
```

B exp_p1a.m

```
% Smetana_Gregory_1917370_A1_P1

function [ exp ] = exp_p1a( x, d )
%exp_p1a Computes exp(x) by summing d terms of Taylor series
exp=0;
for n=0:d
    exp = exp + x^n/factorial(n);
end
end
```

C exp_p1b.m

```
% Smetana_Gregory_1917370_A1_P1


function [ exp ] = exp_p1b( x, d )
%exp_p1a Computes exp(x) through recursion

exp = 0;
for i=d:-1:1
    exp = (exp*x/i + 1);
```

```
end

end
```

D Smetana_Gregory_1917370_A1_P2.m

```
clear all;
close all;
clc;

k=[1,3,5,7,9];

a=1;
b=10.^k;
c=1;

%% part a
xp = quad_p2a(a,b,c);
latextable([cellstr(num2str(abs(k'))), num2cell(xp')], ...
            'Horiz',{'$k$','$x_+$'},...
            'name','report/table2a_roots.tex',...
            'format','%10.5e')

latextable([cellstr(num2str(abs(k'))), num2cell((a*xp.^2 +2*b.*xp +c )')], ...
       'Horiz',{'$k$','$p(x_+)$'},...
       'name','report/table2a_value.tex',...
       'format','%10.5e')

%% part b
xp = quad_p2b(a,b,c);
latextable([cellstr(num2str(abs(k'))), num2cell(xp')], ...
            'Horiz',{'$k$','$x_+$'},...
            'name','report/table2b_roots.tex',...
            'format','%10.5e')

latextable([cellstr(num2str(abs(k'))), num2cell((a*xp.^2 +2*b.*xp +c )')], ...
       'Horiz',{'$k$','$p(x_+)$'},...
       'name','report/table2b_value.tex',...
       'format','%10.5e')
```

E quad_p2a.m

```
% Smetana_Gregory_1917370_A1_P2

function [ xp ] = quad_p2a( a, b, c )
%QUAD_P2A Computes the positive root of quadratic polynomial
% ax^2 + 2bx + c = 0 using the standard formula

xp = (-b + sqrt(b.^2 -a*c))/a;
```

```
end
```

F quad_p2b.m

```
% Smetana_Gregory_1917370_A1_P2

function [ xp ] = quad_p2b( a, b, c )
%QUAD_P2A Computes the positive root of quadratic polynomial using
% ax^2 + 2bx + c = 0 using a modified formula

xp = c./(-b - sqrt(b.^2 -a*c));

end
```