Katherine Olson

998992204

MAT 128A

Due 11/23/16

MAT 128A Programming Project III

This project will be estimating the integral of the function $\frac{2}{\sqrt{\pi}} exp(-x^2)\ dx$ from 0 to b where b equals 0.4, 0.8, 1.2, 1.6 and 2.0 using three different methods. Using the algorithm from class, the algorithm was way too slow to ever reach a tolerance of 10^{-12}. Trying the idea from TA office hours to vectorize the sum, it was much faster but starting at 10^{-6} there was not enough memory. So, I will be using tol = 10^{-5}.

For b = 0.4

What we are estimating: I = 4.283923550466685e-01

The composite trapezoidal rule:

T_j	E_j	$\|I - T_j\|$	j	nfcount
7.420963079166081e-01	5.635131409646258e-06	3.137039528699396e-01	2.200000000000000e+01	4.194283000000000e+06

The composite Simpson's rule:

S_j	E_j	$\|I - S_j\|$	j	nfcount
4.283915305082495e-01	8.029845141995414e-07	8.245384189864424e-07	2.300000000000000e+01	8.388587000000000e+06

The spline-based approach:

	I_{spline}	$\|I - I_{spline}\|$
Equally-spaced points	4.283923550466685e-01	0
Chebyshev points	4.283923550466685e-01	0

The results are very accurate for spline with both types of points. The results are accurate for composite Simpson's rule. And the results are least accurate for the composite trapezoidal rule, but still close to the result we are looking for.

For b = 0.8

What we are estimating: I = 7.421009647076605e-01

The composite trapezoidal rule:

T_j	E_j	$\|I - T_j\|$	j	nfcount
9.763436265350898e-01	5.768489455881820e-06	2.342426618274293e-01	2.300000000000000e+01	8.388586000000000e+06

The composite Simpson's rule:

S_j	E_j	$\|I - S_j\|$	j	nfcount
7.420949535755790e-01	5.799689304097910e-06	6.011132081540360e-06	2.100000000000000e+01	2.097133000000000e+06

The spline-based approach:

	I_{spline}	$\|I - I_{spline}\|$
Equally-spaced points	7.421009647076605e-01	0
Chebyshev points	7.421009647076605e-01	0

The results are the same for spline and trapezoidal and have improved slightly for trapezoidal.

For b = 1.2

What we are estimating: I = 9.103139782296353e-01

The composite trapezoidal rule:

T_j	E_j	$\|I - T_j\|$	j	nfcount
9.993044800335483e-01	8.480541422735872e-06	8.899050180391299e-02	2.300000000000000e+01	8.388586000000000e+06

The composite Simpson's rule:

S_j	E_j	$\|I - S_j\|$	j	nfcount
9.103049887828557e-01	8.670465913003985e-06	8.989446779628096e-06	2.100000000000000e+01	2.097133000000000e+06

The spline-based approach:

	I_{spline}	$\|I - I_{spline}\|$
Equally-spaced points	9.103139782296351e-01	2.220446049250313e-16
Chebyshev points	9.103139782296351e-01	2.220446049250313e-16

The results are the same for spline and trapezoidal, but slightly less accurate for spline, probably due to rounding. The results have improved a lot for trapezoidal, but it is still the least accurate.

For b = 1.6

What we are estimating: I = 9.763483833446440e-01

The composite trapezoidal rule:

T_j	E_j	$\|I - T_j\|$	j	nfcount
9.999891606831663e-01	5.844150900694936e-06	2.364077733852232e-02	2.400000000000000e+01	1.677719300000000e+07

The composite Simpson's rule:

S_j	E_j	$\|I - S_j\|$	j	nfcount
9.763421276799981e-01	6.060524101769716e-06	6.255664645959946e-06	2.200000000000000e+01	4.194284000000000e+06

The spline-based approach:

	I_{spline}	$\|I - I_{spline}\|$
Equally-spaced points	9.763483833446440e-01	0
Chebyshev points	9.763483833446440e-01	0

The results are back to perfect for spline and about the same for Simpson's. The results have improved for trapezoidal, but it is still the least accurate.

For b = 2

What we are estimating: I = 9.953222650189527e-01

The composite trapezoidal rule:

T_j	E_j	$\|I - T_j\|$	j	nfcount
9.99994045943	7.20078179217	4.67178092425	2.4000000000000	1.6777193000000

2080e-01	1540e-06	5242e-03	00e+01	00e+07

The composite Simpson's rule:

S_j	E_j	$\|I - S_j\|$	j	nfcount
9.95314482077 0100e-01	7.53657361845 7017e-06	7.78294194270 2237e-06	2.2000000000000 00e+01	4.1942840000000 00e+06

The spline-based approach:

	I_{spline}	$\|I - I_{spline}\|$
Equally-spaced points	9.953222650189526e-01	1.110223024625157e-16
Chebyshev points	9.953222650189526e-01	1.110223024625157e-16

The results are not quite perfect for spline, probably because of rounding. The results for trapezoidal have really improved, but Simpson's is still more accurate.

Code Appendix

The main script

```
format long e;
clear;
fx = @(x)(2 / sqrt(pi)) * exp(-x.^2);
a = 0;
b = [0.4, 0.8, 1.2, 1.6, 2.0];
b = b(1); % Change this for all of the different b's
tol = 10^(-5);
real = erf(b);
trap = CompTrap2(a, b, fx, tol, 30);
errorT = abs(real - trap);
simp = CompSimp2(a, b, fx, tol, 30);
errorS = abs(real - simp(1));
n = simp(4) - 1;
equal = linspace(a, b, n + 1);
for i = 0:n
    cheby(i + 1) = 5 * cos((pi * (n - i)) / n);
end
eq = Spline(a, b, equal, fx)
cheb = Spline(a, b, cheby, fx);
errorSE = abs(real - eq);
errorSC = abs(real - cheb);
```

The Simpson Function

```
function [ Results ] = CompSimp2( a, b, fx, tol, jmax )
% This program that implements the version of the composite
Simpson's rule presented in class.
h = b - a;
B = 2 * fx((a + b) / 2);
A = fx(a) + B + fx(b);
nfcount = 3;
S = (h / 6) * (A + B);
for j = 2:(jmax + 1)
    h(j) = h(j - 1) / 2;
    sumR = 0;
    sumR = sum(fx(      (   a +    (2 * (2:2^(j - 1)) - 1)  *
(h(j) / 2)    )    ));
    nfcount = nfcount + length(2:(2^(j - 1)));
    B(j) = 2 * sumR;
    A(j) = A(j - 1) + B(j);
    S(j) = (h(j) / 6) * (A(j) + B(j));
    Ej = (16 / 15) * abs(S(j) - S(j - 1));
    Results = [S(j), Ej, j, nfcount];
    if(Ej <= tol)
        return;
    end

end % for j

end
```

<u>The Trapezoidal Function</u>

```
function [ Results ] = CompTrap( a, b, fx, tol, jmax )
% This function implements the version of the composite
trapezoidal rule
% presented in class.
nfcount = 0;
h = b - a;
T = (h / 2) * (fx(a) + fx(b));
nfcount = nfcount + 2;
for j = 2:(jmax + 1)
    h(j) = h(j - 1) / 2;
    sumR = 0;
    sumR = sum(fx(      (   a +    (2 * (2:2^(j - 1)) - 1)  *
h(j)   )    ));
    nfcount = nfcount + length(2:2^(j - 1));
    T(j) = 0.5 * T(j - 1) + h(j) * sumR;
    Ej = (4 / 3) * abs(T(j) - T(j - 1));
    Results = [T(j), Ej, j, nfcount];
```

```
    if(Ej <= tol)
        return;
    end
  end % for j
end
```

<u>The Spline Function</u>

```
function [ I ] = Spline(a, b, xn, fx)
% Computes an approximation to I by integrating a not-a-knot
spline.
% xn is length n + 1.
  n = length(xn) - 1;
  for i = 1:(n + 1)
    yn(i) = fx(xn(i));
  end
  j = 1;
  pp = spline(xn, yn);
  coefs = pp.coefs;
  for i = pp.order:-1:1
      icoefs(:, j) = coefs(:, j) / i;
      j = j + 1;
  end
  icoefs(:, j) = zeros(1, length(icoefs(:, 1)));
  ppd = mkpp(pp.breaks, icoefs);
  %I = ppval(ppd, b) - ppval(ppd, a)
  I = integral(@(x)ppval(pp,x), a, b)
end
```