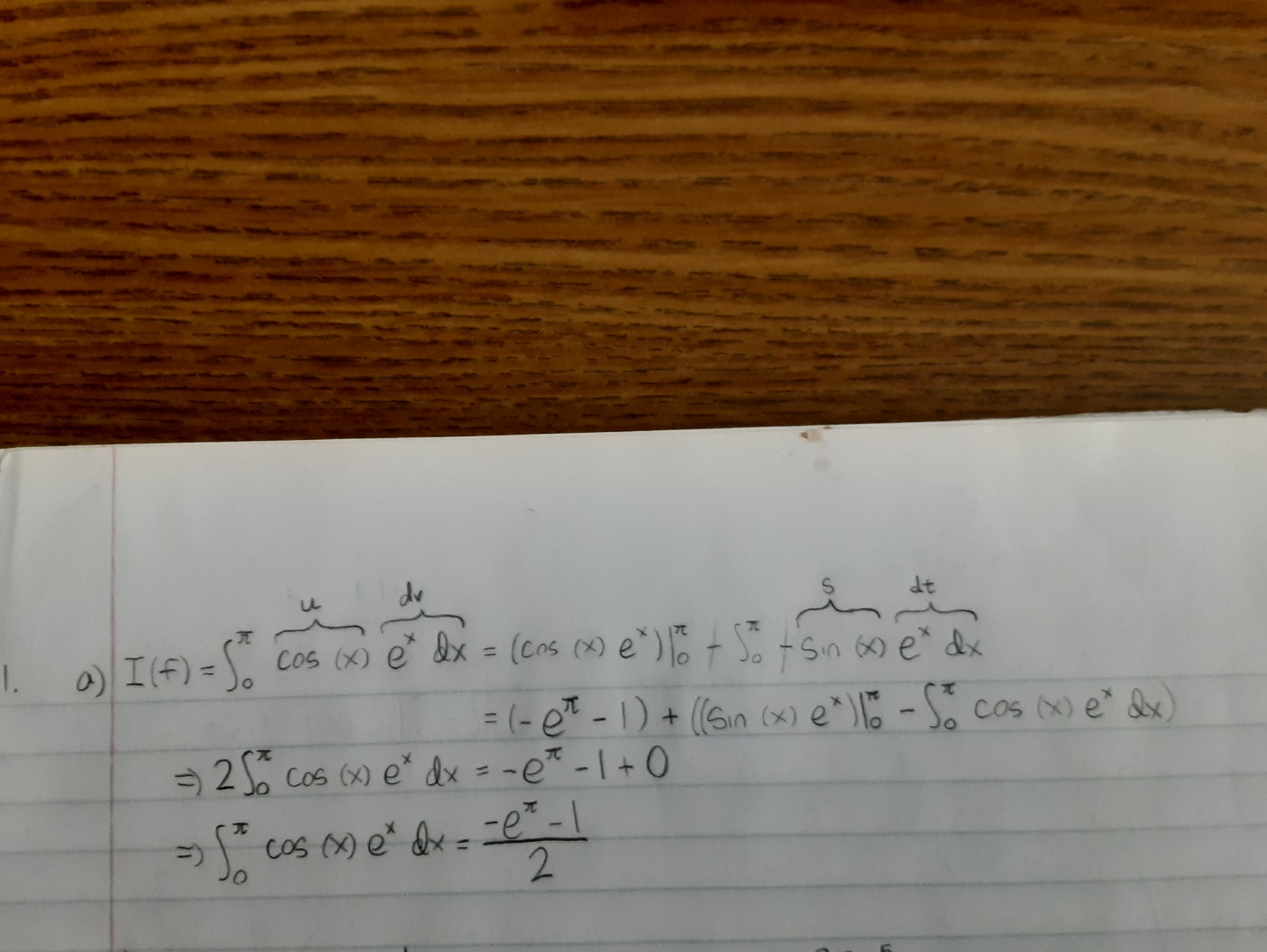
Mark Moreno

Homework 5

1.

a)



b)

Storage matrix:

1 -17.3892593301323 5.31891301374262

2 -13.3360228473715 1.26567653098186

3 -12.3821624297556 0.311816113365946

4 -12.1480040998968 0.0776577835071972

5 -12.0897421170142 0.0193958006245651

6 -12.0751940992021 0.00484778281250620

7 -12.0715581891024 0.00121187271271772

8 -12.0706492800054 0.000302963615787633

9 -12.0704220570084 7.57406187865684e-05

Code:

Storage\_Matrix = zeros( 9, 3 );

for i = 1 : 9

x = 0 : pi / ( 2^i ) : pi;

y = exp( x ) .\* cos( x );

I = ( sum( y ) - ( exp( 0 ) \* cos( 0 ) ./ 2 ) - ( exp( pi ) \* cos( pi ) ./ 2 ) ) \* pi ./ ( 2^i );

z = I - ( -exp( pi ) - 1 ) ./ 2;

error = abs( z );

Storage\_Matrix( i, 1 ) = i;

Storage\_Matrix( i, 2 ) = I;

Storage\_Matrix( i, 3 ) = error;

if i > 1000

fprintf( 'Error: Too many iterations.\n' );

break;

end

end

%The storage matrix indicates that I(f) is converging to about -12.07 or

%so, which is roughly equal to what I calculated in a), ((-e^pi) - 1)/2.

%The error is decreasing by a factor of 4 each time we double the

%subintervals (half their length), which indicates the order of convergence

%is h^2, since O((h/2)^2) = 1/4 \* O(h^2).

c)

Storage matrix:

1 -11.5928395534215 0.477506762968133

2 -11.9849440197846 0.0854022966050643

3 -12.0642089572169 0.00613735917269231

4 -12.0699513232772 0.000394993112390551

5 -12.0703214560533 2.48603363122157e-05

6 -12.0703447599314 1.55645818722405e-06

7 -12.0703462190691 9.73205445120584e-08

8 -12.0703463103064 6.08319083994502e-09

9 -12.0703463160094 3.80206088834711e-10

Code:

Storage\_Matrix = zeros( 9, 3 );

for i = 1 : 9

x = 0 : 2 \* pi / ( 2^i ) : pi; %odd xsubi's

y = 2 .\* exp( x ) .\* cos( x );

v = pi / ( 2^i ) : 2 \* pi / ( 2^i ) : pi; %even xsubi's

w = 4 .\* exp( v ) .\* cos( v );

I = ( sum( y ) + sum( w ) - ( exp( 0 ) \* cos( 0 ) ) - ( exp( pi ) \* cos( pi ) ) ) \* pi ./ ( 3 \* ( 2^i ) );

z = I - ( -exp( pi ) - 1 ) ./ 2;

error = abs( z );

Storage\_Matrix( i, 1 ) = i;

Storage\_Matrix( i, 2 ) = I;

Storage\_Matrix( i, 3 ) = error;

if i > 1000

fprintf( 'Error: Too many iterations.\n' );

break;

end

end

%The convergence in this case is again to roughly ((-e^pi) - 1)/2. The

%convergence is quicker here with the Simpson Rule, so we can tell what

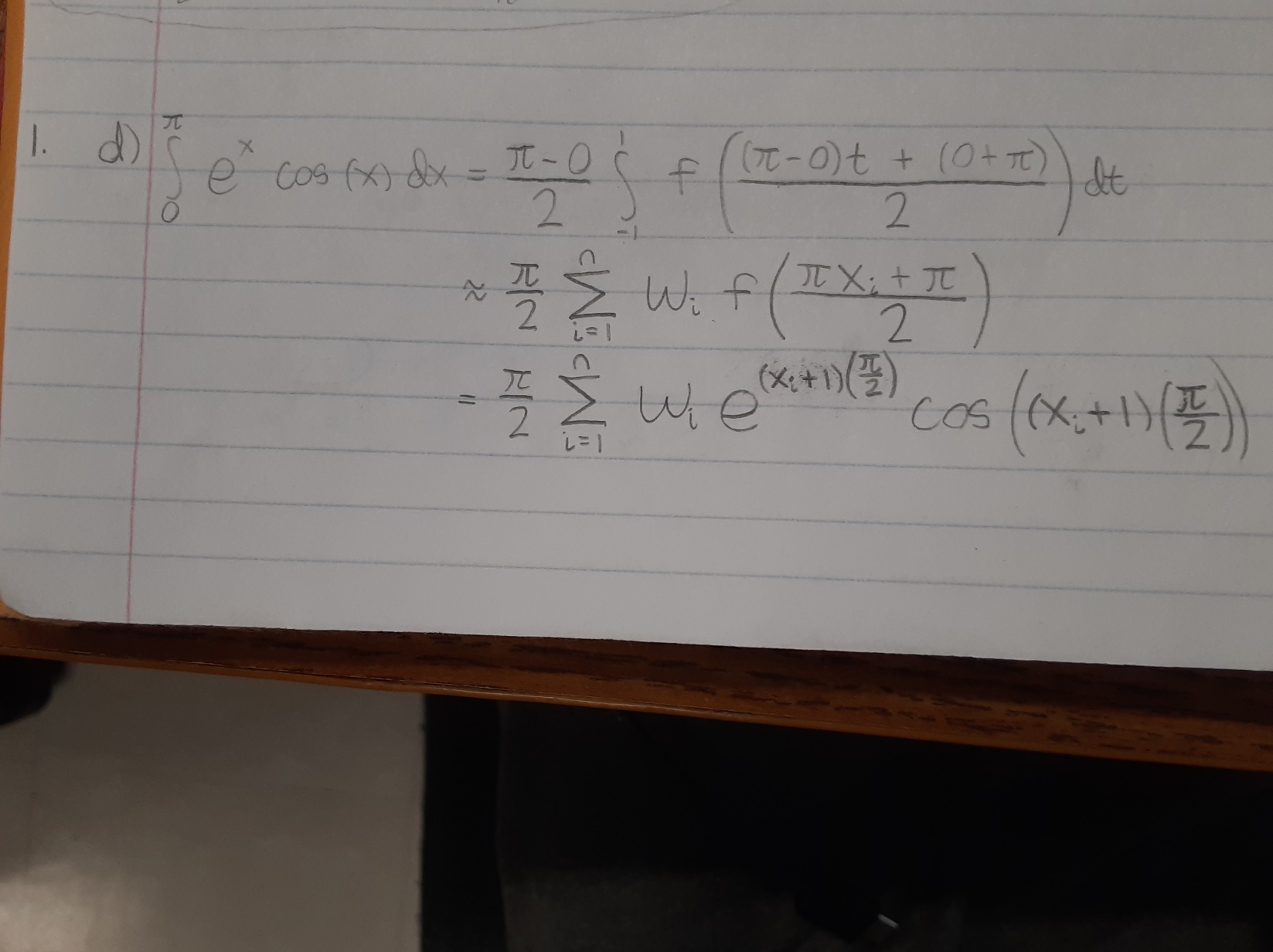
%I(f) is converging to with greater accuracy (about -12.0703). The error is

%decreasing by about a factor of 16 each time we double the number of

%subintervals (half their length, which indicates that the order of

%convergence is h^4, since O((h/2)^4) = 1/16 \* O(h^4).

d)



Command window:

>> Mark\_Moreno\_hw5\_1\_d

I =

0.0000 -12.3362 -12.1274 -12.0702 -12.0703 -12.0703

error =

12.0703 0.2659 0.0571 0.0002 0.0000 0.0000

>>

Code:

I1 = 2 \* exp( (0 + 1) \* pi / 2 ) \* cos( (0 + 1) \* pi / 2 ) \* pi / 2;

I2 = ( ( 1 \* exp( (-.577350269 + 1) \* pi / 2 ) \* cos( (-.577350269 + 1) \* pi / 2 ) ) +...

( exp( (.577350269 + 1) \* pi / 2 ) \* cos( (.577350269 + 1) \* pi / 2 ) ) ) \* pi / 2;

I3 = ( ( .555555556 \* exp( (-.774596669 + 1) \* pi / 2 ) \* cos( (-.774596669 + 1) \* pi / 2 ) ) +...

( .888888889 \* exp( (0 + 1) \* pi / 2 ) \* cos( (0 + 1) \* pi / 2 ) ) +...

( .555555556 \* exp( (.774596669 + 1) \* pi / 2 ) \* cos( (.774596669 + 1) \* pi / 2 ) ) ) \* pi / 2;

I4 = ( ( .347854845 \* exp( (-.861136312 + 1) \* pi / 2 ) \* cos( (-.861136312 + 1) \* pi / 2 ) ) +...

( .652145155 \* exp( (-.339981044 + 1) \* pi / 2 ) \* cos( (-.339981044 + 1) \* pi / 2 ) ) +...

( .652145155 \* exp( (.339981044 + 1) \* pi / 2 ) \* cos( (.339981044 + 1) \* pi / 2 ) ) +...

( .347854845 \* exp( (.861136312 + 1) \* pi / 2 ) \* cos( (.861136312 + 1) \* pi / 2 ) ) ) \* pi / 2;

I5 = ( ( .236926885 \* exp( (-.906179846 + 1) \* pi / 2 ) \* cos( (-.906179846 + 1) \* pi / 2 ) ) +...

( .478628670 \* exp( (-.538469310 + 1) \* pi / 2 ) \* cos( (-.538469310 + 1) \* pi / 2 ) ) +...

( .652145155 \* exp( (0 + 1) \* pi / 2 ) \* cos( (0 + 1) \* pi / 2 ) ) +...

( .478628670 \* exp( (.538469310 + 1) \* pi / 2 ) \* cos( (.538469310 + 1) \* pi / 2 ) ) +...

( .236926885 \* exp( (.906179846 + 1) \* pi / 2 ) \* cos( (.906179846 + 1) \* pi / 2 ) ) ) \* pi / 2;

I6 = ( ( .171324492 \* exp( (-.932469514 + 1) \* pi / 2 ) \* cos( (-.932469514 + 1) \* pi / 2 ) ) +...

( .360761573 \* exp( (-.661209386 + 1) \* pi / 2 ) \* cos( (-.661209386 + 1) \* pi / 2 ) ) +...

( .467913935 \* exp( (-.238619186 + 1) \* pi / 2 ) \* cos( (-.238619186 + 1) \* pi / 2 ) ) +...

( .467913935 \* exp( (.238619186 + 1) \* pi / 2 ) \* cos( (.238619186 + 1) \* pi / 2 ) ) +...

( .360761573 \* exp( (.661209386 + 1) \* pi / 2 ) \* cos( (.661209386 + 1) \* pi / 2 ) ) +...

( .171324492 \* exp( (.932469514 + 1) \* pi / 2 ) \* cos( (.932469514 + 1) \* pi / 2 ) ) ) \* pi / 2;

I = [ I1 I2 I3 I4 I5 I6 ]

z = [ (I1 - (( -exp( pi ) - 1 ) ./ 2)) (I2 - (( -exp( pi ) - 1 ) ./ 2)) ...

(I3 - (( -exp( pi ) - 1 ) ./ 2)) (I4 - (( -exp( pi ) - 1 ) ./ 2)) ...

(I5 - (( -exp( pi ) - 1 ) ./ 2)) (I6 - (( -exp( pi ) - 1 ) ./ 2)) ];

error = abs( z )

%The convergence with Gauss-Legendre Quadrature is far quicker than with

%any of the other methods. The error for the 4th iteration is more than 500

%times better than the 3rd, even though the 3rd is less than 5 times better

%than the 2nd.

e)

%The accuracy of the Fundamental Theorem of Calculus was obviously the best

%(being exact), but Gauss-Legendre quadrature (whose error dropped

%below 10^-4 on the 5th iteration) was also quite accurate. The Simpson

%Rule's accuracy is also quite good. In fact, its error drops below 10^-4

%even faster than Gauss-Legendre quadrature's error, although it only does

%so because it started out much closer to the true solution. (It is

%sometimes difficult to compare the relative accuracies of different

%methods for this reason--do you give credit to the Simpson Rule for

%starting closer? I suppose you could, but Gauss-Legendre quadrature

%appears to eventually converge to the true solution much faster, as the

%Simpson rule only converges toward the true solution by a factor of 16 per

%iteration.) And the Trapezoidal Rule's accuracy was the worst no matter

%how you look at it: it converges toward the true solution at a far slower

%rate (a factor of 4 per iteration) than even the Simpson Rule, and its

%error didn't drop below 10^-4 until its 8th iteration (whereas it was the

%5th for Gauss-Legendre quadrature and the 4th for Simpson's Rule.)

%As far as effort goes, I had the most trouble with Gauss-Legendre

%quadratue, as I wasn't able to set up any kind of loop (and any loop would

%have been a lot of trouble anyway, as I still would have had to manually

%insert all 42 parameters into the code somehow.) The Simpson Rule was the

%next hardest, as I had to set up 2 functions, each with its own increment

%vector, to make the for-loop work. I might have put more effort into the

%Trapezoidal Rule, though, even though it was essentially a simpler version

%of the Simpson Rule, just because I did that one first. And while

%integration by parts is hard, I would say it's definitely less "effort"

%than coding because I know within a reasonable amount of time I'll either

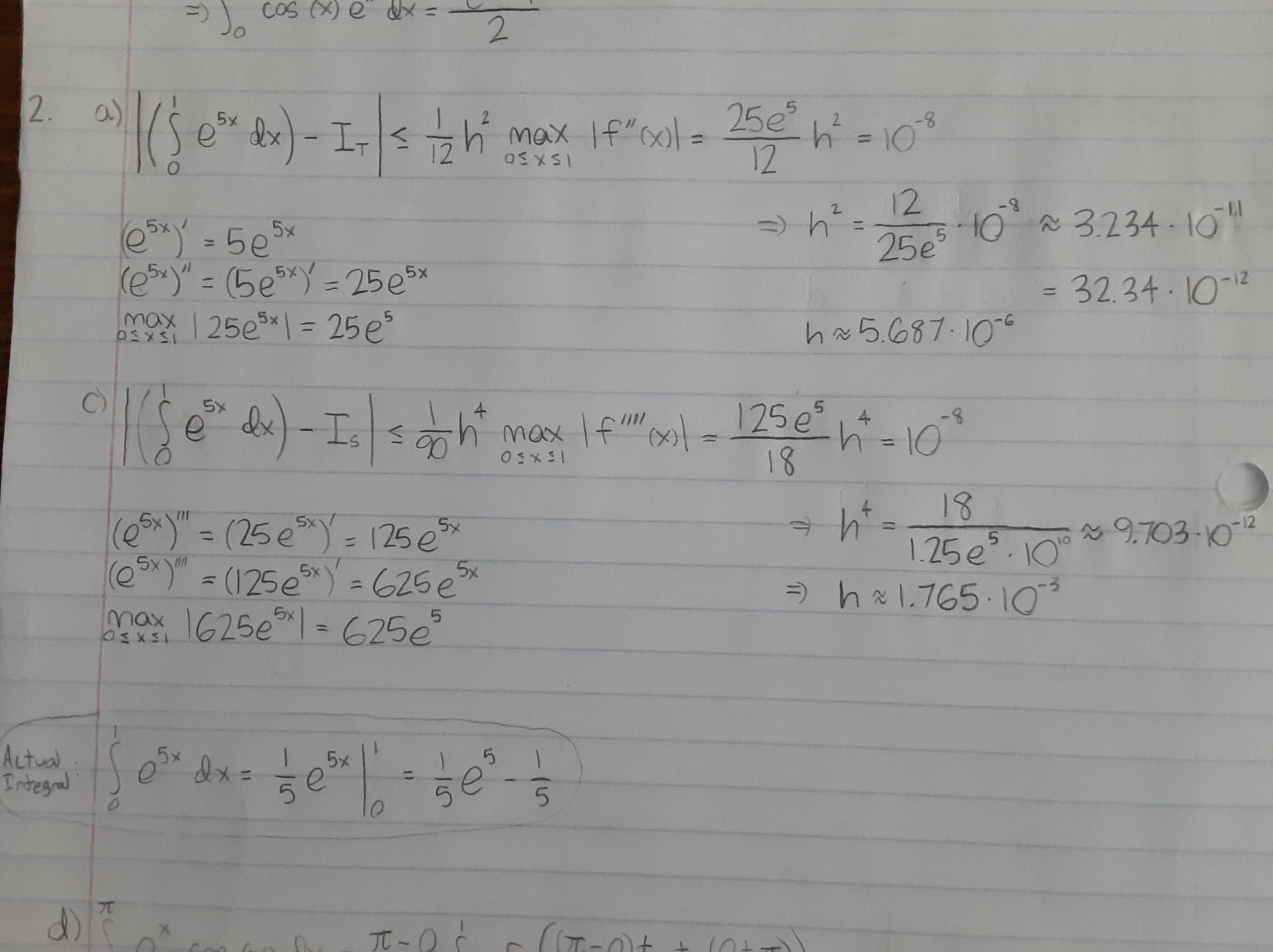
%get the right answer or realize it won't work and I have to try to find

%another method. With coding, all I know is within a reasonable amount of

%time I'll go home, whether I've made any progress or not.

2.

a)



b)

Storage matrix:

1 29.4826711306806 3.93101652811367e-05

2 29.4826416480586 9.82754330891567e-06

3 29.4826361883125 4.36779717105651e-06

4 29.4826342774012 2.45688588762505e-06

5 29.4826333929223 1.57240701526007e-06

6 29.4826329124647 1.09194932917944e-06

7 29.4826326227638 8.02248433018349e-07

8 29.4826324347368 6.14221477235333e-07

9 29.4826323058262 4.85310891917834e-07

10 29.4826322136172 3.93101874607282e-07

11 29.4826321453928 3.24877525770262e-07

12 29.4826320935027 2.72987396243707e-07

13 29.4826320531199 2.32604577377060e-07

14 29.4826320210775 2.00562212171462e-07

15 29.4826319952272 1.74711857425791e-07

16 29.4826319740706 1.53555305359987e-07

17 29.4826319565367 1.36021412799892e-07

18 29.4826319418429 1.21327598634480e-07

19 29.4826319294077 1.08892336925237e-07

20 29.4826319187906 9.82752759171035e-08

21 29.4826319096540 8.91386626733492e-08

22 29.4826319017347 8.12193867716360e-08

23 29.4826318948256 7.43102717137845e-08

24 29.4826318887621 6.82467415913379e-08

25 29.4826318834114 6.28961274173889e-08

26 29.4826318786664 5.81510732899915e-08

27 29.4826318744387 5.39233653285010e-08

28 29.4826318706558 5.01404642250236e-08

29 29.4826318672575 4.67422118788363e-08

30 29.4826318641932 4.36779039603152e-08

31 29.4826318614207 4.09053697580930e-08

32 29.4826318589043 3.83889791066849e-08

33 29.4826318566128 3.60974858892860e-08

34 29.4826318545207 3.40054064906781e-08

35 29.4826318526052 3.20898756456245e-08

36 29.4826318508473 3.03320142336361e-08

37 29.4826318492299 2.87146200150801e-08

38 29.4826318477386 2.72232725251342e-08

39 29.4826318463602 2.58448658030375e-08

40 29.4826318450842 2.45689228961510e-08

41 29.4826318439004 2.33850734332464e-08

42 29.4826318428000 2.22846594510884e-08

43 29.4826318417755 2.12601989346695e-08

44 29.4826318408204 2.03050909419744e-08

45 29.4826318399277 1.94123970231885e-08

46 29.4826318390930 1.85776301009355e-08

47 29.4826318383108 1.77954788682655e-08

48 29.4826318375772 1.70618612571616e-08

49 29.4826318368880 1.63726383561880e-08

50 29.4826318362395 1.57241615283965e-08

51 29.4826318356288 1.51134962322885e-08

52 29.4826318350531 1.45377434535021e-08

53 29.4826318345096 1.39942493149192e-08

54 29.4826318339962 1.34808573193368e-08

55 29.4826318335103 1.29949846439104e-08

56 29.4826318330505 1.25352244140231e-08

57 29.4826318326144 1.20990968355272e-08

58 29.4826318322008 1.16854828036139e-08

59 29.4826318318081 1.12927480699909e-08

60 29.4826318314347 1.09194289166226e-08

61 29.4826318310799 1.05645305836788e-08

62 29.4826318307418 1.02264294810084e-08

63 29.4826318304200 9.90464243955103e-09

0 0 0

Code:

Storage\_Matrix = zeros( 64, 3 );

for i = 1 : 64

x = 0 : 1/(1250\*i) : 1;

y = exp(5.\*x);

I = ( sum(y) - ( exp(0) ./ 2 ) - ( exp(5) ./ 2 ) ) ./ (1250.\*i);

z = I - ((exp(5)./5) - (1/5));

error = abs(z);

Storage\_Matrix( i, 1 ) = i;

Storage\_Matrix( i, 2 ) = I;

Storage\_Matrix( i, 3 ) = error;

if error < 10^-8

break

end

if i > 64

fprintf( 'Error: Too many iterations.\n' );

break;

end

end

%Our storage matrix indicates that the error drops below 10^-8 between i = 62

%(h = 1/77500) and i=63 (h = 1/78750), so at around 1.28 \* 10^-5. In a), I found

%that the error for h = 5.687 \* 10^-6 (about twice as small) can be

%no more than 10^-8; however, this is only the case when f''(c) happens to

%be f''(1), the maximum second derivative on the interval, and c can be any

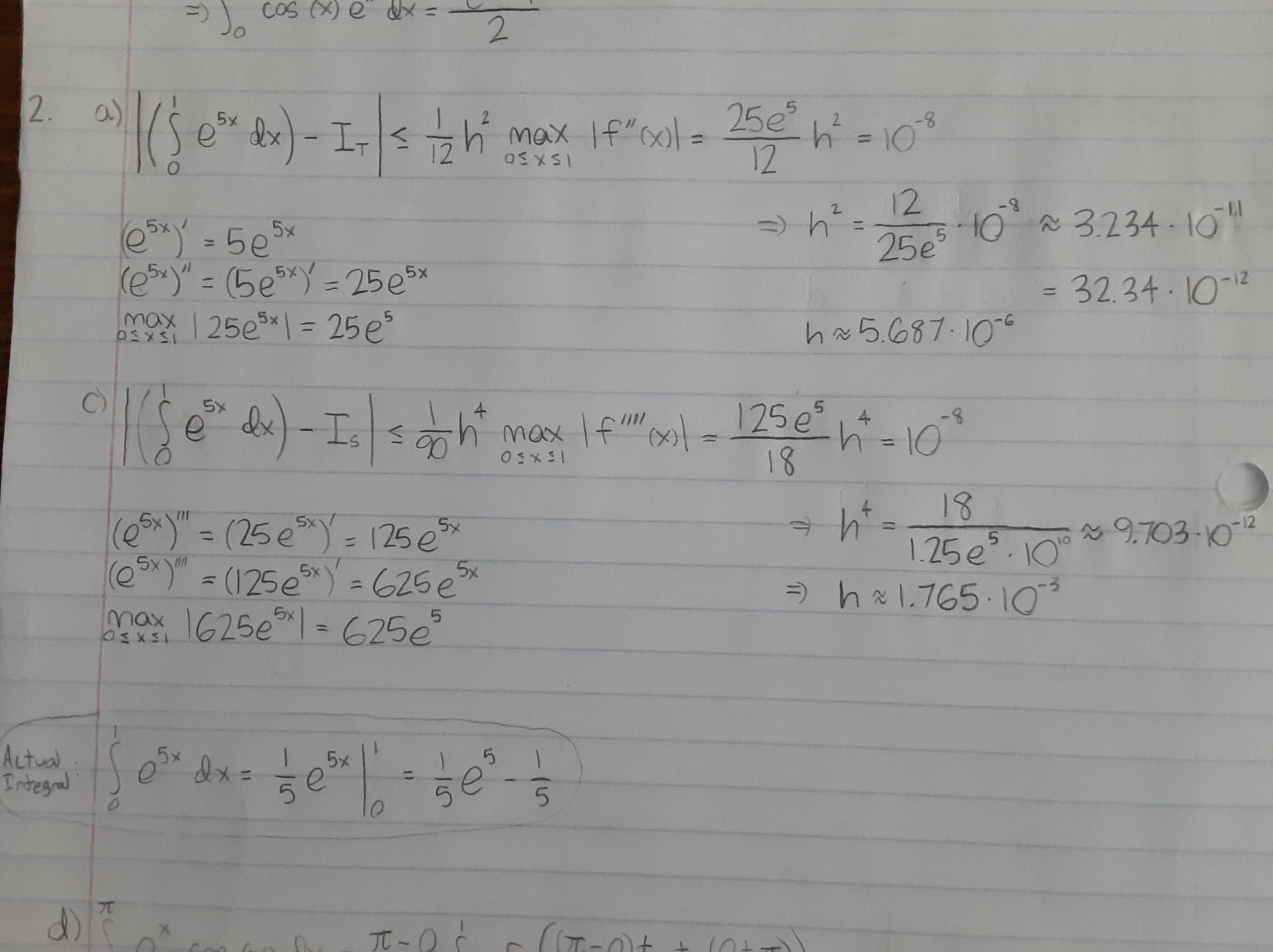
%number between 0 and 1. The maximum possible error for h = 1.3 \* 10^-5

%should be about 2^2 = 4 times larger than for h = 5.687 \* 10^-6,

%so while it happened to be only the same here, it could have in theory

%been up to about 4 times larger (4 \* 10^-8).

c) (same picture as a))



d)

Storage matrix:

1 29.5555355921321 0.0729037716167404

2 29.4874684504644 0.00483662994906808

3 29.4835981124149 0.000966291899604954

4 29.4829387857644 0.000306965249091462

5 29.4827577868215 0.000125966306164571

6 29.4826926294366 6.08089212761342e-05

7 29.4826646636300 3.28431147202934e-05

8 29.4826510801519 1.92596366197506e-05

9 29.4826438474689 1.20269536054707e-05

10 29.4826397129289 7.89241355292347e-06

11 29.4826372119130 5.39139771404962e-06

12 29.4826356276172 3.80710185865496e-06

13 29.4826345848011 2.76428578516175e-06

14 29.4826338758070 2.05529167374152e-06

15 29.4826333802265 1.55971113713349e-06

16 29.4826330254086 1.20489331223439e-06

17 29.4826327659877 9.45472386604251e-07

18 29.4826325727753 7.52259939673650e-07

19 29.4826324264906 6.05975273515469e-07

20 29.4826323140970 4.93581662652787e-07

21 29.4826322265940 4.06078665804444e-07

22 29.4826321576503 3.37135013239731e-07

23 29.4826321027365 2.82221154890294e-07

24 29.4826320585616 2.38046311551443e-07

25 29.4826320227014 2.02186107856051e-07

26 29.4826319933466 1.72831235545345e-07

27 29.4826319691309 1.48615619366410e-07

28 29.4826319490117 1.28496356666119e-07

29 29.4826319321847 1.11669383073831e-07

30 29.4826319180241 9.75088099153254e-08

31 29.4826319060387 8.55233430741009e-08

32 29.4826318958394 7.53240669837396e-08

33 29.4826318871162 6.66008936889284e-08

34 29.4826318796200 5.91047104592235e-08

35 29.4826318731494 5.26341104034600e-08

36 29.4826318675406 4.70252530249127e-08

37 29.4826318626593 4.21439843023563e-08

38 29.4826318583953 3.78799711597821e-08

39 29.4826318546572 3.41418342486577e-08

40 29.4826318513690 3.08536378668123e-08

41 29.4826318484673 2.79519447587973e-08

42 29.4826318458988 2.53834748775716e-08

43 29.4826318436187 2.31033929765090e-08

44 29.4826318415890 2.10736352812546e-08

45 29.4826318397773 1.92619751260281e-08

46 29.4826318381562 1.76408931906735e-08

47 29.4826318367021 1.61868030090773e-08

48 29.4826318353948 1.48794931931207e-08

49 29.4826318342169 1.37015483403502e-08

50 29.4826318331532 1.26378871811994e-08

51 29.4826318321908 1.16754605983260e-08

52 29.4826318313183 1.08029318823810e-08

53 29.4826318305257 1.00104280420510e-08

54 29.4826318298046 9.28925913967760e-09

0 0 0

0 0 0

0 0 0

0 0 0

0 0 0

0 0 0

0 0 0

0 0 0

0 0 0

0 0 0

Code:

Storage\_Matrix = zeros( 64, 3 );

for i = 1 : 64

x = 0 : 1/(3\*i) : 1; %odd xsubi's

y = 2.\*exp(5.\*x);

v = 1/(6\*i) : 1/(3\*i) : 1; %even xsubi's

w = 4.\*exp(5.\*v);

I = ( sum(y) + sum(w) - exp(0) - exp(5) ) ./ (18.\*i);

z = I - ((exp(5)/5) - (1/5));

error = abs(z);

Storage\_Matrix( i, 1 ) = i;

Storage\_Matrix( i, 2 ) = I;

Storage\_Matrix( i, 3 ) = error;

if error < 10^-8

break

end

if i > 64

fprintf( 'Error: Too many iterations.\n' );

break;

end

end

%Our storage matrix indicates that the error drops below 10^-8 at about i =

%53 (h = 1/318), so at around 3.14 \* 10^-3. In c), I found

%that the error for h = 1.765 \* 10^-3 (about twice as small) can be

%no more than 10^-8; however, this is only the case when f''(c) happens to

%be f''(1), the maximum second derivative on the interval, and c can be any

%number between 0 and 1. The maximum possible error for h = 3.14 \* 10^-3

%should be about 2^4 = 16 times larger than for h = 1.765 \* 10^-3,

%so while it happened to be only the same here, it could have in

%theory been up to 16 times larger, or about 5.024 \* 10^-2.

3.

Plot for a) (I was able to open this; hopefully you can too. I’ll also email it to you separately.)



Storage matrix for b)

1 484.121321163310

2 484.003232658175

3 483.986641229635

4 483.985923475464

5 483.985693485503

6 483.985612839696

7 483.985578079047

8 483.985561245848

9 483.985552290256

10 483.985547174260

11 483.985544081134

12 483.985542122495

13 483.985540833650

14 483.985539957583

15 483.985539345331

16 483.985538907045

17 483.985538586637

18 483.985538348025

19 483.985538167384

20 483.985538028603

21 483.985537920563

22 483.985537835443

23 483.985537767648

24 483.985537713112

25 483.985537668844

26 483.985537632607

27 483.985537602715

28 483.985537577880

29 483.985537557109

30 483.985537539631

31 483.985537524837

32 483.985537512248

33 483.985537501482

34 483.985537492229

35 483.985537484243

36 483.985537477321

37 483.985537471296

38 483.985537466034

39 483.985537461420

40 483.985537457362

41 483.985537453781

42 483.985537450611

43 483.985537447797

44 483.985537445292

45 483.985537443056

46 483.985537441056

47 483.985537439261

48 483.985537437648

49 483.985537436194

50 483.985537434881

51 483.985537433694

52 483.985537432617

53 483.985537431639

54 483.985537430749

55 483.985537429938

56 483.985537429197

57 483.985537428519

58 483.985537427899

59 483.985537427330

60 483.985537426807

61 483.985537426326

62 483.985537425882

63 483.985537425473

64 483.985537425096

Code:

% a)

L = .01 : .01 : 4;

%hc = alpha \* ksubb \* T = 2 \* 7000 \* 1.380649 \* 10^-23 = 1.9329086 \* 10^-19

E = 8 .\* pi .\* 1.9329 .\* (10^-19) ./ ( (L.^5) .\* (exp(2./L) - 1) );

plot( L, E )

title( 'Spectral Energy Density' );

xlabel( 'wavelength' );

ylabel( 'energy' );

% b)

Storage\_Matrix = zeros( 64, 2 );

for i = 1 : 64

x = .01 : 13/(100\*i) : .4; %odd xsubi's

y = 100 .\* 2 .\* 8 .\* pi .\* 1.9329086 ./ ((x.^5) .\* (exp(2./x) - 1)); %I'm

%multiplying the entire loop by 10^21 so that I can get 6 significant digits

v = ( .01 + 13/(200\*i) ) : 13/(100\*i) : .4; %even xsubi's

w = 100 .\* 4 .\* 8 .\* pi .\* 1.9329086 ./ ((v.^5) .\* (exp(2./v) - 1));

I = ( sum(y) + sum(w) -...

( 100 \* 8 \* pi \* 1.9329086 / ( (.01^5) \* (exp(2/.01) - 1) ) ) -...

( 100 \* 8 \* pi \* 1.9329086 / ( (.4^5) \* (exp(2/.4) - 1) ) ) ) / (600\*i/13);

Storage\_Matrix( i, 1 ) = i;

Storage\_Matrix( i, 2 ) = I;

if i > 64

fprintf( 'Error: Too many iterations.\n' );

break;

end

end

%The storage matrix indicates that E(.01,.4) is converging to about

%483.9855, which we then have to divide by 10^21 (since I multiplied the

%loop by 10^21) to get the correct answer, which is 4.839855 \* 10^-19.

4.

