NUMERICAL ANALYSIS PROJECT #2

XINGLU WANG
Student Number: 3140102282
College of Information Science & Electronic Engineering

CONTENTS

LIST OF FIGURES

* *College of Information Science & Electronic Engineering, ZheJiang University, China*

1 PROBLEM AND ANALYSIS

The question is to find results for

$$R = \int_0^{\infty} B\log_2(1+\frac{ph}{N_0})\frac{1}{2\sigma^2}e^{-\frac{h}{2\sigma^2}}$$

It has been prove that there is not closed, explicit, or analytical expression for integration like $\frac{\int e^x}{x}dx$ and $\int \ln(x+1)e^x dx$. To find ergodic capacity for user, we can use numerical method. Before applying quadrature method, we need to observe intergrand,

- The physical parameter is expressed in dB. Take care when multiply between dB.
- The coefficient can be quite huge or small. Make some transformation
- It is an improper integration, to be specific, it is an infinite integration. We should use some special technique.

Since the coefficient on $e^{-\frac{h}{2\sigma^2}}$ is small, substitute $\frac{h}{2\sigma^2}$ with h,

$$f_R(h) = 1.4427 \times 10^6 \ln(1.0298 \times 10^7 h + 1.0)e^{-h}$$

For convenience, the formula is rewritten as

$$f_R(h) = K_1 \ln(1+Kx)e^{-x}dx, \tag{1}$$

where $K_1 = 1.4427 \times 10^6$, $K = 1.0298 \times 10^7$.
Integrate by parts, we get

$$\begin{aligned} R &= \int_0^{\infty} f_R(h) && (2)\\ &= -\ln(kx+1)e^{-x}\big|_0^{\infty} + K_1 K \int_0^{\infty} \frac{1}{1+Kx}e^{-x}dx && (3)\\ &= K_1 K \int_0^{\infty} \frac{1}{1+Kx}e^{-x}dx && (4) \end{aligned}$$

2 GUASS LAGUERRE RULE

2.1 Basic idea

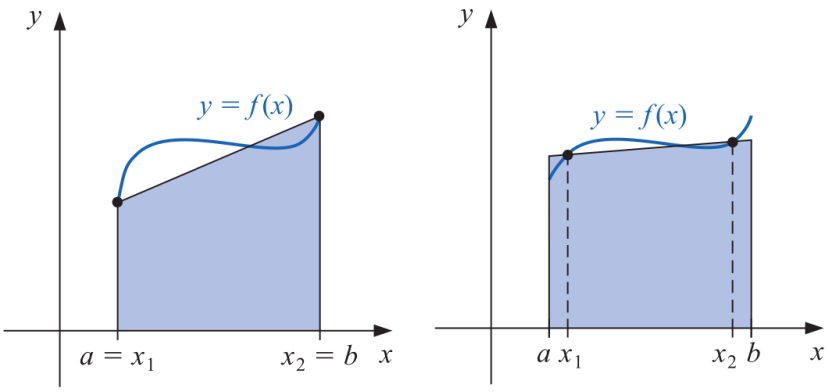
All the Newton-Cotes formulas use values of the function at equally-spaced points. It has both advantage and disadvantage.Now focus on its disadvantage, compare Trapezoidal rule and Gaussian quadrature. 1b on the following page Apparently, Gaussian quadrature'S local error is more small. How can we get C_i and x_i? We can express it via optimization model.
Decision variables:

$\{C_i\}$	$i \in (1,n)$	weights
$\{x_i\}$	$i \in (1,n)$	nodes

Objective

$$\min \quad \left| \int_a^b f(x) - \sum_{i=1}^{n} c_i f(x_i) \right|$$

For every $f(x)$, polynomial and non-polynomial, C_i and x_i should achieve optimal. It is an open question now. To simply the question, we assume $f(x)$



(a) Trapezoidal rule **(b)** Forest landscape.

Figure 1: Find optimal C_i and x_i.

is $P_n(x)$. Then the best choice of these values produces the exact result for the largest class of polynomials, that is, the choice that gives the greatest degree of precision.
Decision variables:

$\{C_i\}$ $i \in (1, n)$ weights
$\{x_i\}$ $i \in (1, n)$ nodes

Objective

$$\max \quad degree(P(x))(\text{measurement precision})$$

Constrains:

$P(x)$ is any polynomial whose degree is smaller than the given optimization goal.

2.2 Proof

Since $2n$ there are parameters to choose, $\max degree(P(x)) = 2n - 1$. The method to find C_i and x_i is shown below.

Definition 1. The Laguerre polynomial $\{L_0(x), L_1(x)...\}$ is an orthogonal set on $[0, \infty)$ and satisfy $\int_0^\infty e^{-x} L_i(x) L_j(x) dx = 0$, for $i \neq j$.

Theorem 2. Gaussian Laguerre rule Suppose that $x_1, x_2, ..., x_n$ are the roots of the nth Laguerre polynomial $P_n(x)$ and that for each $i = 1, 2, ..., n$, the numbers C_i are defined by

$$C_i = \int_0^\infty e^{-x} \prod_{j=1, j\neq i}^{n} \frac{x - x_j}{x_i - x_j} dx$$

if $P(x)$ is any polynomial of degree less than $2n$, then

$$\int_0^\infty P(x) e^{-x} dx = \sum_{i=1}^{n} C_i P(x_i)$$

which means the guassian Laguerre rule has maesurement of precision of $2n - 1$
proof Let us first consider the degree of $P(x)$ is less than n. Use Lagrange interpolation

$$P(x) = \sum_{i=1}^{n} P(x_i) L_i(x_i) + \frac{f^{(n+1)}(\xi(x))}{(n+1)!} \prod_{i=1}^{n} (x - x_i)$$

Since what we consider is just polynomial, we can see the error term is 0. Then

$$\int_0^\infty e^{-x}P(x)dx = \sum_{i=1}^{n}\left[P(x_i)\int_0^\infty e^{-x}\prod_{\substack{j=1\\ j\neq i}}^{n}\frac{x-x_j}{x_i-x_j}dx\right] = \sum_{i=1}^{n}C_iP(x_i)$$

Then, let us consider $P(x)$ ($n \leqslant degree(P(x)) < 2n$). Divide $P(x)$ by the nth laguerre polynomial $P_n(x)$. Then we get an $R(x)$ as remainder.

$$P(x) = Q(x)P_n(x) + R(x).$$

where $degree(Q(x)) < n$ and $degree(R(x)) < n$. According to the orthogonality of laguerre polynmial,

$$\int_0^\infty e^{-x}P_n(x)Q(x)dx = 0$$

Note that x_i is a root of $P_n(x)$, so

$$P(x_i) = Q(x_i)P_n(x_i) + R(x_i) = R(x_i)$$

Here $degree(R(x)) < n$, and fall into the case one, so

$$\int_0^\infty e^{-x}R_n(x) = \sum_{i=1}^{n}C_iR(x_i) = \sum_{i=1}^{n}C_iP(x_i)$$

Putting these facts together verifies that the formula is exact for the polynomial $P(x)$

$$\int_0^\infty e^{-x}P(x)dx = \sum_{i=1}^{n}C_iP(x_i)$$

2.3 Implement and Application

The flow chart of my algrithom is shown in 2 on the next page My focus is implementing the quadrature rather than roots_finding, the algorithm **Main2(symbolic_compute).m** implemented by myself will be shown below. And the file **Main1(use GUN).m** uses the code distributed under the GNU LGPL license. (http://people.sc.fsu.edu/~jburkardt/m_src/gen_laguerre_rule/gen_laguerre_rule.html)

```
function Main2_symbolic_compute()
syms x
k1=sym(1000000/log(2));
k=sym(4600000*10^(7/20));
%f=exp(-x)/(k*x + 1);
n=4;

%Laguerre polynomial
p=polelague(n);
%Roots
xi=real(solve(poly2sym(p)) );

%Coeficients
warning off;
P=LagInter(xi);
for i=1:n
```

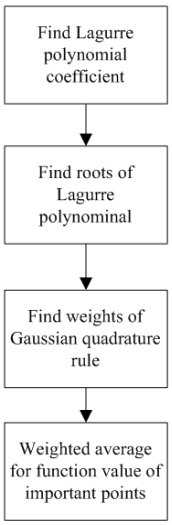


Figure 2: flow chart

```
17      wi(i)=int(exp(-x)*P(i),x,0,inf);
18  end
19  %double(wi')
20
21  % double(xi)
22  double(k1*k*sum(wi'.*subs(1/(1+k*x),x,xi)))
23  end
24
25
26  function p=polelague(n)
27  % p=polegend(n)
28  % Almacena en las filas de la matriz p los coefs de los ...
        polinomios de Legendre
29  p(1,1)=1;
30  p(2,1:2)=[-1 1];
31  for k=2:n
32     p(k+1,1:k+1)=((2*(k-2)*[0 p(k,1:k)]+3*[0 ...
           p(k,1:k)]-[p(k,1:k) 0]-(k-1).^2*[0 0 p(k-1,1:k-1)]));
33  end
34  p=p(n+1,:);
35  end
36
37  function [P]=LagInter(xn)
38  syms total L_up L_down x
39  %sizen=size(xn,1);
40  total=prod(x-xn);
41  L_up=total./(x-xn);
42  L_down=subs(L_up,x,xn);
43  L_down=L_down(L_down~=0);
44  P=L_up/L_down;
45  end
```

First, let us see a successful example 3 on the following page. We can see for

$$\int_0^\infty e^{-x}\frac{1}{1+x}dx$$

The result is the same as symbolic computation in matlab. And the rate of convergence is acceptable. It converges when degree of Laguerre poly

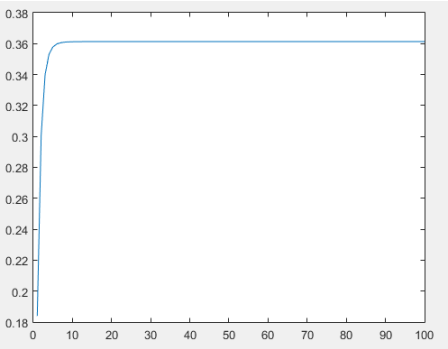


Figure 3: successful example

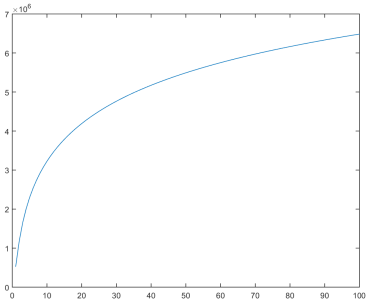


Figure 4: Gauss_Laguerre fail

$n = 5$. But there are some problem occured in the progress of integrating in the following.

$$\int_0^{\infty} e^{-x} \frac{1}{1+x} dx$$

where $k = 1.029811723741436e+07$, it converges and computes slowly, refer to 4. Only when $n = 10000$, I get $2.179759274595623e+07$, relatively close to $2.246313340780598e+07$ computed by symbolic tool, and the time costed as high as b refer to 5

I guess it is because k is too big, so I have made many substitution and transforation first.

```
[wi,xi,¬]=gen_laguerre_rule(50,0,0,1);
double(k1*k*sum(wi.*subs(1/(1+k*x),x,xi)))
```

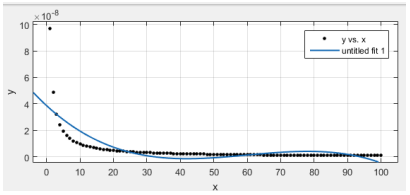
```
[wi,xi,¬]=gen_laguerre_rule(50,0,0,1/k);
double(k1*sum(wi.*subs(1/(1+x),x,xi)))
```

Profile Summary

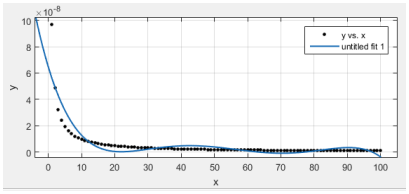
Generated 24-Jan-2016 19:36:04 using cpu time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
Main	1	266.168 s	0.050 s	
gen_laguerre_rule	501	193.538 s	0.010 s	
gen_laguerre_rule>cgqf	501	193.518 s	0.000 s	
gen_laguerre_rule>cdgqf	501	193.356 s	0.011 s	
gen_laguerre_rule>sgqf	501	193.265 s	0.030 s	
gen_laguerre_rule>imtqlx	501	193.235 s	191.945 s	
mupadmex (MEX-file)	266789	60.162 s	59.438 s	

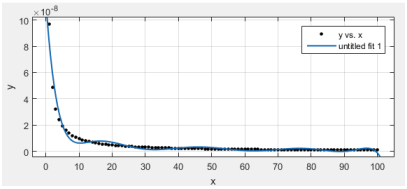
Figure 5: runtime



(a) degree=3



(b) degree=7



(c) degree=13

Figure 6: Rungeś phenomenon

```
double(int(exp(-x)/x,x,1/k,inf)*exp(1/k)/k*k*k1)
[wi,xi,¬]=gen_laguerre_rule(50,0,1/k,1);
double(exp(1/k)/k*k*k1*sum(wi.*subs(1/x,x,xi)))
```

Then I try to implement all algorithm via symbol computation, which means immense accuracy, but the time-consuming is great and I only wait for the result $2.138289534346160e+07$

2.4 Analysis

It is apparently this algorithm will converge slowly if n is huge, but why it suffers big error when n is small? We can illustrate it by 6b. When k is quite hugh the influence of Rungeś phenomenon will be amplified.
To sum up, Gaussian quadrature as above will only produce accurate results if the function $f(x)$ is well approximated by a polynomial function within the range $(0, \infty]$.
Finally, I guess there are still many other objective function and may result other methods to improve its performance.

strength	weakness
Accurate when condition is satisfied	Invoking f(x) be well approximated by a polynomial function
Tabular for laguerre ploy can be made previously	Accurate when f(x) is polynomial, and I can not decided the accuracy for f(x) in other form.
	Without error bound.
	Cannot use composite method to be more accuracy.

3 ADAPTIVE QUAD

Since Gauss Laguerre method on the book is not suitable for this question, I must quest for more accurate and fast method.

3.1 Infinite to Finite

A one-parameter family of transformations can be made by splitting the range $[0, \infty)$ into two intervals $[0, S]$ and $[S, \infty)$. And apply $t = x/S$, we get

$$\int_0^\infty f(x)dx = \int_0^\infty \frac{e^{-x}}{1+kx}dx \tag{1}$$

$$= S\int_0^1 [f(St) + t^{-2}f(S/t)]dt \tag{2}$$

$$= S\int_0^1 \left(\frac{e^{-St}}{kSt+1} + \frac{e^{-\frac{S}{t}}}{t(kS+t)}\right)dt \tag{3}$$

$$= \text{def as} = S\int_0^1 (f_1(x) + f_2(x))dx \tag{4}$$

3.2 Why choose it?

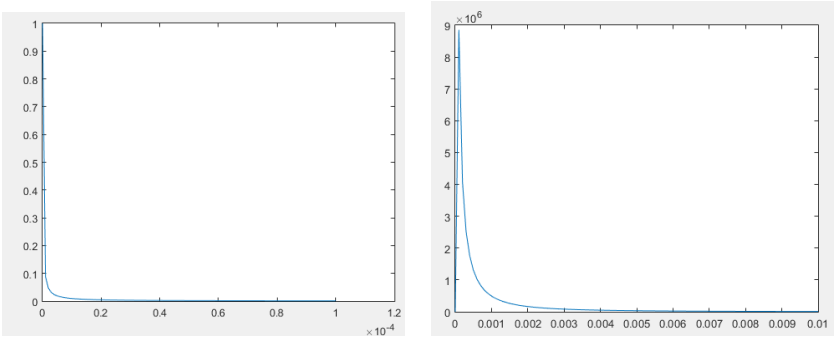
Observe $f(x), f_1(x), f_2(x)$, (refer to 7c on the following page) $f_2(x)$ change rapidly in a very small range, $f_x(s)$ contain its function value mainly on a range very close to 0. Thus, $f_1(x)$ is the main parts of integration. To sum up,

- We should not abandon $f_2(x)$ for accuracy purpose. The sensitivity of S should be checked.
- Why Romberg or other Newton-Cotes formulas which use values of the function at equally-spaced points fail. Because they can not focus on the small region, wasting time on trivial things?
- Why Gaussian Laguerre fail? Because it cannot divided and conquer, cannot iterate from last result to a more accurate result.

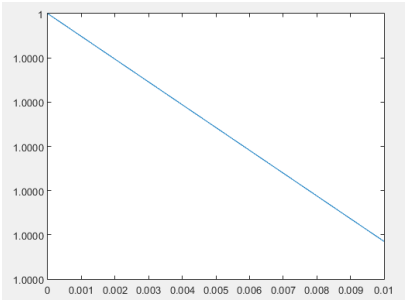
3.3 Implement and Results

A rather accurate results $2.246313340780605e+07$ compared to $2.246313340780598e+07$ with relative error of $3.150963598256511e-15$ can be resulted quite fast in just 7.111s when $S = 0.0001$

```
function y = adapt_simp ( f, a, b, TOL )

fa = feval ( f, a );
fc = feval ( f, (a+b)/2 );
fb = feval ( f, b );
sab = (b-a)*(fa + 4*fc + fb)/6;
[approx eest nfunc] = as ( sab, fa, fc, fb, f, a, b, TOL );
y = approx;
return

function [app, est, nf] = as ( sab, fa, fc, fb, f, a, b, TOL )

c = (a+b)/2;
fd = feval ( f, (a+c)/2 );
fe = feval ( f, (c+b)/2 );
sac = (c-a)*(fa + 4*fd + fc)/6;
scb = (b-c)*(fc + 4*fe + fb)/6;

errest = abs ( sab - sac - scb );
```



(a) $f(x)$ **(b)** $f_2(x)$



(c) $f_1(x)$

Figure 7: Change rapidly

```
20  if ( errest < (10.0*TOL) )
21     app = sac+scb;
22     est = errest / 10.0;
23     nf = 2;
24     return;
25  else
26     [a1 e1 n1] = as ( sac, fa, fd, fc, f, a, c, TOL/2.0 );
27     [a2 e2 n2] = as ( scb, fc, fe, fb, f, c, b, TOL/2.0 );
28     app = a1 + a2;
29     est = e1 + e2;
30     nf = n1 + n2 + 2;
31     return;
32  end;
```

3.4 Sensitivity of S

Refer to 8 on the next page, we can see from 1 down to 0.00001 the results is more and more close to symbolic answer. Meanwhile ,the costs of time also increase. Overall, compared to its acuuracy, this kind of slightly unstable is acceptable.

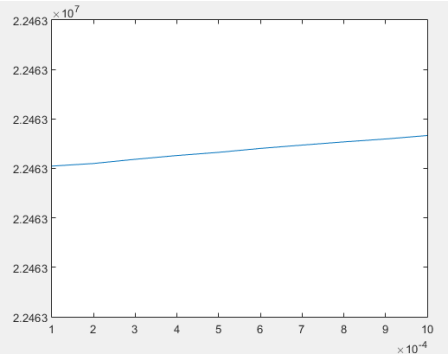


Figure 8: Sensitivity of S

APPENDIX

Weights and nodes of Laguerre polynomial

i	xi	wi
1	3.570039E-02	8.841211E-02
2	1.881623E-01	1.768147E-01
3	4.626943E-01	2.113631E-01
4	8.597730E-01	1.940812E-01
5	1.380011E+00	1.464343E-01
6	2.024209E+00	9.332680E-02
7	2.793369E+00	5.093220E-02
8	3.688703E+00	2.397619E-02
9	4.711641E+00	9.774625E-03
10	5.863851E+00	3.457940E-03
11	7.147248E+00	1.062247E-03
12	8.564017E+00	2.832717E-04
13	1.011663E+01	6.550941E-05
14	1.180789E+01	1.311607E-05
15	1.364093E+01	2.268453E-06
16	1.561929E+01	3.379626E-07
17	1.774691E+01	4.322821E-08
18	2.002823E+01	4.728494E-09
19	2.246825E+01	4.403174E-10
20	2.507256E+01	3.472441E-11
21	2.784748E+01	2.305382E-12
22	3.080015E+01	1.279773E-13
23	3.393866E+01	5.894177E-15
24	3.727225E+01	2.232218E-16
25	4.081149E+01	6.880336E-18
26	4.456860E+01	1.705604E-19
27	4.855776E+01	3.353712E-21
28	5.279561E+01	5.146200E-23
29	5.730186E+01	6.044763E-25
30	6.210018E+01	5.310585E-27
31	6.721937E+01	3.392528E-29
32	7.269516E+01	1.521735E-31
33	7.857280E+01	4.585292E-34
34	8.491123E+01	8.762159E-37
35	9.178987E+01	9.827416E-40
36	9.932081E+01	5.801152E-43
37	1.076724E+02	1.530909E-46
38	1.171223E+02	1.381986E-50
39	1.282018E+02	2.566634E-55
40	1.422800E+02	2.700361E-61

```
close all;format long;clear
% syms G pho B No d p sigma2
%% prepare parameter
G=sym(-128.1);
pho=sym(3.76);
B=sym(1e6);
N0=sym(-114);
d=0.1;
p=23;

% G=-128.1;
% pho=3.76;
% B=1e6;
% N0=-114;

G = dB2W(G);
sigma2=G*d^(-pho);
N0 = dB2W(N0-30);
%% using matlab symbolic tool
syms h f(h) dR(h)

f=exp(-h);
dR=B*log2(1+2*p*h*sigma2/N0)*f;
res=int(dR,h,0,inf);
double(res);

k1=sym(1000000/log(2));
k=sym(4600000*10^(7/20));
syms x f(x)
f=k1*k/(1+k*x)*exp(-x);
res=int(f,x,0,inf);
double(res);

f=1/(1+k*x)*exp(-x);
res=k1*k*int(f,x,0,inf);
double(res);

%% for illustration
% xx=sym(0:0.01:2)
% plot(xx,subs(f,h,xx));
% hold on;
% plot(xx,subs(log2(1+2*p*h*sigma2/N0),h,xx));
% plot(xx,subs(dR,h,xx));
% hold off;

%%  My algrithm starts
[wi,xi,¬]=gen_laguerre_rule(50,0,0,1);
double(k1*k*sum(wi.*subs(1/(1+k*x),x,xi)))


% [wi,xi,¬]=gen_laguerre_rule(50,0,0,1/k);
% double(k1*sum(wi.*subs(1/(1+x),x,xi)))
%
% double(int(exp(-x)/x,x,1/k,inf)*exp(1/k)/k*k*k1)
% [wi,xi,¬]=gen_laguerre_rule(50,0,1/k,1);
% double(exp(1/k)/k*k*k1*sum(wi.*subs(1/x,x,xi)))

inti=1;
endi=1000;
resd(1)=0;
for n=inti:endi
[wi,xi,¬]=gen_laguerre_rule(n,0,0,1);
%  * the ORDER (number of points) in the rule;
%  * ALPHA, the exponent of |X|;
%  * A, the left endpoint of integration;
%  * B, the scale factor in the exponential;
```

```
%  * FILENAME, the root name of the output files.
% The generalized Gauss-Laguerre quadrature rule is used as ...
    follows:
%
%         Integral ( a <= x < +oo ) |x-a|^alpha * exp(-b*(x-a)) ...
    f(x) dx
%
% is to be approximated by
%         Sum ( 1 <= i <= order ) w(i) * f(x(i))

resd(n)=sum(wi.*subs(1/(1+k*x),x,xi));
end
plot(inti:endi,double(resd));
double(resd(end)*k1*k)


%% for test
% syms x;
% f=exp(-x)/(1+x);
%
% res=double(int(f,x,0,inf))
%
% for n=1:100
% [wi,xi,¬]=gen_laguerre_rule(n,0,0,1);
% res(n)=sum(wi.*subs(1/(x+1),x,xi));
% end
% plot(1:100,double(res))
```

```
format long;clear;close all
% syms G pho B No d p sigma2
%% prepare parameter
G=sym(-128.1);
pho=sym(3.76);
B=sym(1e6);
N0=sym(-114);
d=0.1;
p=23;

G = dB2W(G);
sigma2=G*d^(-pho);
N0 = dB2W(N0-30);
%% using matlab symbolic tool
syms h f(h) dR(h)

f=exp(-h);
dR=B*log2(1+2*p*h*sigma2/N0)*f;
res=int(dR,h,0,inf);
double(res);

k1=sym(1000000/log(2));
k=sym(4600000*10^(7/20));
syms x f(x)
f=k1*k/(1+k*x)*exp(-x);
res=int(f,x,0,inf);
double(res);

f=1/(1+k*x)*exp(-x);
res=k1*k*int(f,x,0,inf);
res=double(res);

%%  My algrithm starts
memo(1)=0;
for S=0.0001:0.0001:0.001
syms t
k=double(k);k1=double(k1);
```

```
true=double(k*k1*S*int(exp(-S*t)/(1+S*k*t)+exp(-S/t)/(t*(k*S+t)),0,1));
f1=exp(-S*t)/(1+S*k*t);%double(int(f1,0,1));
f2=exp(-S/t)/(t*(k*S+t));%double(int(f2,0,1));

% f=exp(-x)/(1+k*x);
% xx=0:0.000001:0.0001;
% plot(xx,double(subs(f,x,xx)));

f1=matlabFunction(f1);
f2=matlabFunction(f2);

% xx=0:0.0001:0.01;
% plot(xx,double(subs(f1,t,xx)));
% figure
% xx=1e-30:0.0001:0.01;
% plot(xx,double(subs(f2,t,xx)));

r1 = adapt_simp ( f1, 0, 1,  10^-16);
r2 = adapt_simp ( f2, 10^-18, 1, 10^-16);
res=k*k1*S*(r1+r2)

error=abs(res-true)/true
memo(end+1)=res;
end
plot(0.0001:0.0001:0.001,memo(2:end))
```