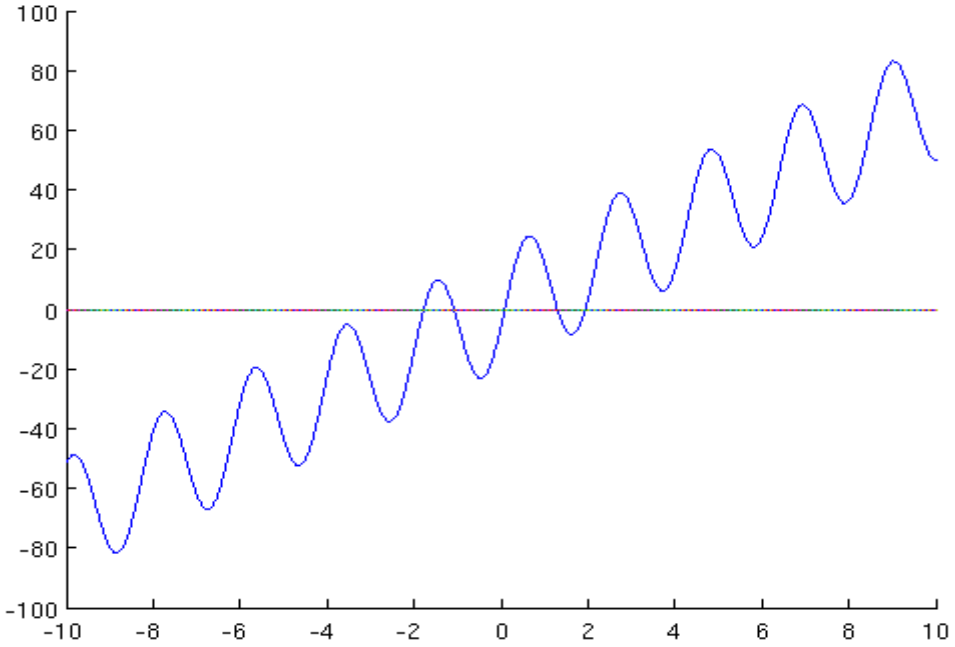
Deducing the Roots

The task is to find the roots to the equation $7x - 20\cos(3x - 5)$.

a) First, we graph the function above to determine approximately where the roots are.



It's obvious that $\overline{x} = (-2, -1, 0.01, 1, 2)$ are approximately the roots.

b) With paper and pen, we perform 3 iterations with the Newton-Rhapson method on the initial value of 1.5, the resulting iteration is: $x_0 = 1.5, x_1 = 1.176, x_2 = 1.2948, x_3 = 1.3006.$

c) In appendix A there's a Matlab program that calculates the roots automatically and outputs the error together with the constant K in the definition of quadratic convergence and the number of iterations it took to converge. The results are the following:

initial value	final value	the constant K	no. of iterations
-2	-1.775068882661	1.0351578249900	5
-1	-1.080616604050	0.7012213915084	4
0.01	0.0858526853563	0.0403799047283	4
1	1.3006044981914	0.8825160676646	5
2	1.9412645505156	1.1984589322514	4

d) Now we show that the previous equation is equivalent to the fix-point iteration where

$$F(x) = (7/8)x + (5/14)cos(3x - 5)$$

To do this we set $F(x) = x,$ and deduce the previous equation.

$$(7/8)x + (5/14)cos(3x-5) = x$$

$$(5/14)cos(3x-5) - x/8 = 0$$

$$7x - 20cos(3x-5) = 0$$

Theoretically, this fix-point iteration should converge when $|F'(a)|<1,$ where a is the root. In appendix A, we have a program that calculates $|F'(a)|$ for all values that we got in the previous exercise, and the results are the following:

a	-1.77506888	-1.08061660	0.085852685	1.300604498	1.941264550
$\|F'(a)\|$	0.035435735	1.866840126	0.195944761	1.828982108	0.088862137

So theoretically, the first, the third and the fifth values should converge.

e) In appendix A we also have a program that attempts to calculate the roots with fix-point iteration, but halts after 100 iterations. We subtracted the results with the results we got with the Newton-Rhapson method to determine the error, and the resulting vector is:

$$\overline{x}_{error} = (0,-1.166,0,1.214,0)$$

So the theory is correct.

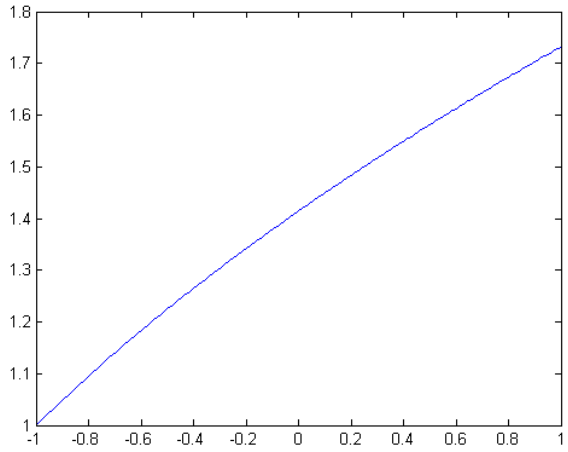
Numerical Approximation of Integrals

In this task, we are supposed to approximate the integral:

$$I = \int_{-1}^{1} \sqrt{x+2}\,dx$$

Part A – By approximation and Inspection

a) The graph of $\sqrt{x+2}$ is shown below. The area under the graph seems to be 0.7 as it looks similar to a triangle $0.7*2/2 = 0.7$.



Figur 1: The graph of the function (the integrand)

b) The actual value of the integral is 2.7974349485 .

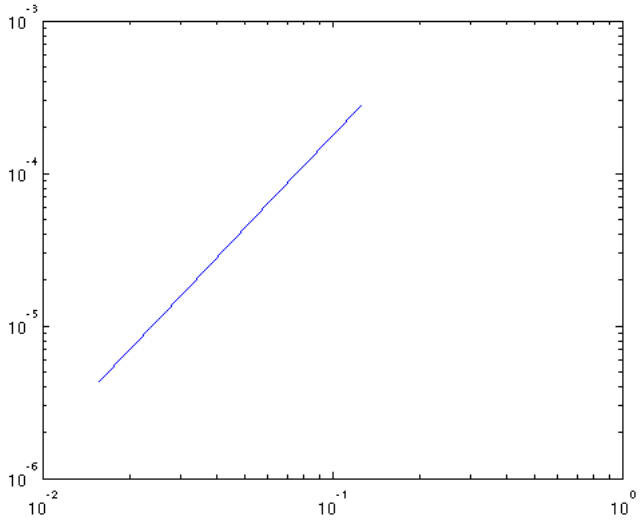
c) The table below illustrates the different values obtained using Trapezoidal rule as well as those obtained when extrapolation was used. The error for each of these approximations is included also. The power that the step length has to be raised to for trapezoidal rule is 1.992268898798353. When Trapezoidal rule is used together with one extrapolation, the exponent is 3.928868876022998 . This means that Trapezoidal rule has an 2nd order of accuracy, but when it is combined with one extrapolation, it increases to 4th order. The exponent was found by taking the logarithm from both sides and obtaining a linear relation. It turns out that the exponent is simply the slope of a straight line, which we can find using **polyfit**, for example.

h	T(h)	$E_t = T(h) - I$	$T_{extr}(h)$	$E_{T_{extr}} = T_{extr}(h) - I$
1.0000000000	2.7802389662	0.0171959823	2.7973354564	9.9492130303E-05
0.5000000000	2.7930613338	0.0043736147	2.7974277911	7.1574078184E-06
0.2500000000	2.7963361768	0.0010987717	2.7974344804	4.6811995924E-07
0.1250000000	2.7971599045	0.0002750440	2.7974349188	2.9651640254E-08
0.0625000000	2.7973661653	0.0000687832	2.7974349466	1.8862764684E-09

Figur 2: Relation between the step length, the approximation and the error for trapezoidal rule with and without one extrapolation.

Part B – Investigating the Truncation Error

By using **loglog**, we obtained the following graph (for trapezoidal rule):



Figur 3: Error vs. steplength (logarithmic scale)

In order to find how the truncation error E_T depends on the step length h, we need to take the natural logarithm of both sides, i.e.

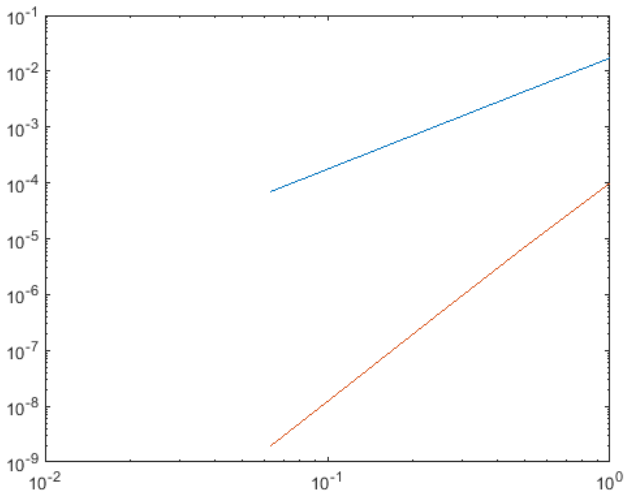
$$E_T = Ch^p$$
$$\ln(\mathrm{E}_T) = \ln(\mathrm{Ch}^p)$$
$$\ln(\mathrm{E}_T) = \ln C + p \ln h$$

As it can be seen, p is the slope of the graph when the natural logarithm of the truncation error is graphed as a function of the natural logarithm of the step length. Thus, it can be found by,

$$\ln(E_T) = \ln C + p \ln h$$
$$p = \frac{\ln(E_T) - \ln C}{\ln h}$$

For Trapezoidal rule, we obtained the p value to be close to $2(\approx 1.999801614352136)$, hence quadratic order of accuracy[1]. When Trapezoidal rule was used in combination with one extrapolation, the order of accuracy increased to $3(\approx 3.273758466097475)$, hence cubic. Juxtaposing these results with those obtained in *Part A*, it is observed that the order of accuracy for Trapezoidal rule together with one interpolation has decreased from 4th to 3rd order of accuracy. No significant change in the order of accuracy is observed for Trapezoidal rule with not extrapolation. The relationship between the error and the step length is shown in the diagram on next page.

[1] Alltså, noggrannhetsordning.



Figur 4: Error vs. steplength (logarithmic scale). Blue line represents error in Trapezoidal rule. The orange line represents error in Trapezoidal rule togehter with one extrapolation.

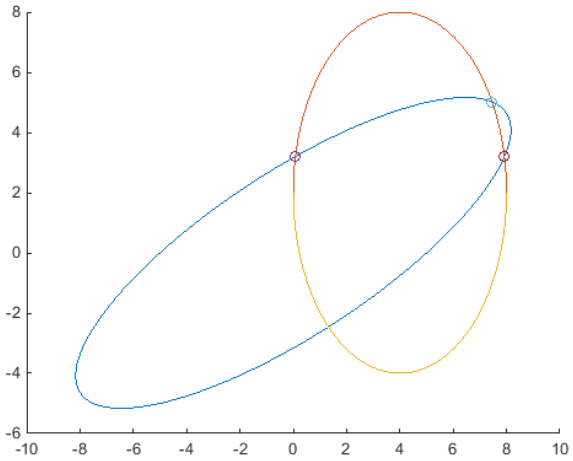
Please see Appendix B for MATLAB code.

Intersection between Ellipses – Nonlinear Systems of Equations

We are given two ellipses, defined as:

$$\begin{cases} 0.4x^2 + y^2 - xy = 10 \\ \dfrac{(x-4)^2}{4^2} + \dfrac{(y-2)^2}{6^2} = 1 \end{cases}$$

The first one is a rotated ellipse, so it is a good idea to use polar coordinates. These ellipses are graphed in the figure below. The circles indicate an intersection point. Unfortunately, one of the intersection points was not obtained as it suddenly converged to another intersection point.



Figur 5: A graph of the ellipses. The circles indicate a coordinate of intersection found numerically.

The intersection points are:

$$\begin{pmatrix} 0.081181927103024 \\ 3.202712312308590 \end{pmatrix}, \begin{pmatrix} 7.920295437725071 \\ 3.191813138716193 \end{pmatrix}, \begin{pmatrix} 7.457829532922956 \\ 5.016205254359505 \end{pmatrix}, \begin{pmatrix} 7.920294646792740 \\ 3.191811520359849 \end{pmatrix}$$

Now, let us take a look at the way our initial guess converges to these points. The tables below only portray the error, not the actual value at that point.

0.205783149
0.011769156
0.003252043
0.000908172
0.000253464
7.08E-05
1.97E-05
Actual value: 8.118193e-02, 3.202712e+00

0.714487
0.374042
0.185367
0.102139
0.052446
0.027703
0.01438
0.007529
0.003924
0.00205
0.001069
0.000558
0.000291
0.000152
7.94E-05
4.14E-05
Actual value: 7.457830e+00, 5.016205e+00

0.033304
0.082172
0.2066
0.549206
1.704017
9.395806
49.29854
11.30768
9.46478
5.866589
2.861252
1.001588
0.194858
0.036926
0.005896
0.000869
0.000122
1.70E-05
Actual value: 7.920295e+00, 3.191813e+00
Note that this iteration does not actually converge to the point it is supposed to converge to, but rather to another nearby point, hence the increase in the error at the beginning.

0.088545
0.016714
0.002512
0.00036
5.04E-05
7.01E-06
Actual value: 7.920295e+00 3.191812e+00

In conclusion, most of these points hint at quadratic convergence.

Please see Appendix C for MATLAB code.

Second Order Differential Equation Approximation

The task is to solve the equation

$$y''(x)+\pi y(x)e^{x/3}(2y'(x)sin(\pi x)+\pi y(x)cos(\pi x)) = y(x)/9, y(0)=1, y'(0)= -1/3$$

a) First we rewrite this equation to a system of first order ODE:s, by making the substitution $u_1 = y(x)$ and $u_2 = \frac{dy}{dx}$ which yields the following system:

$$\begin{cases} \frac{du_1}{dx} = u_2 \\ \frac{du_2}{dx} = \frac{u_1}{9} - \pi u_1 e^{x/3}(2u_2 \sin(\pi x) + \pi u_1 \cos(\pi x)) \end{cases}$$

with the initial values

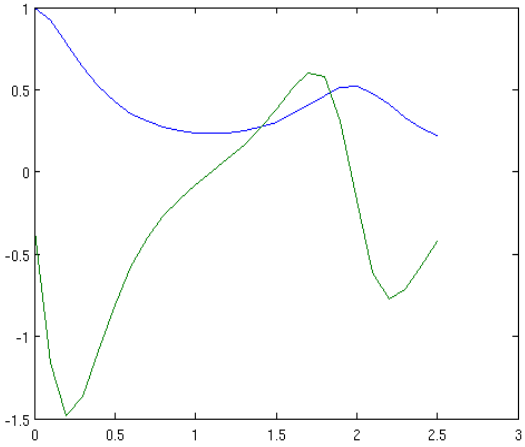
$$u_1(0) = 1$$

$$u_2(0) = -1/3$$

In appendix D there is a matlab program that solves this equation with RK4 in the interval $0 \le x \le 2.6$ and the sidestep h is either 0.1 or 0.2. To compare the difference between the two solutions we note the value of $y(2.6)$:

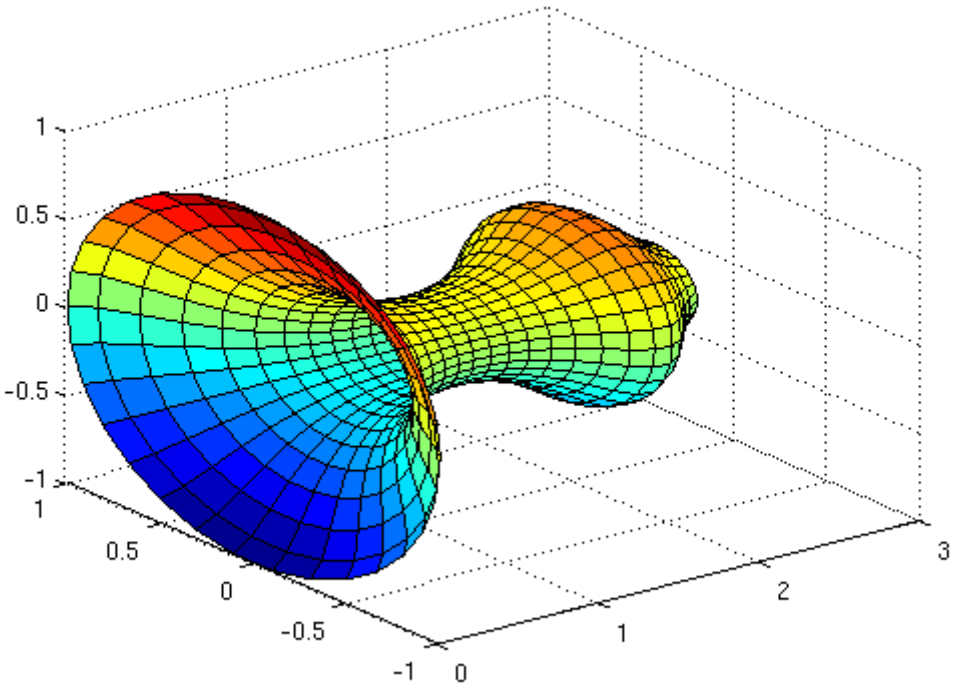
h	y(2.6)
0.1	0.219431
0.2	0.305364

Also for $h = 0.1$, we graph the solution curve, shown below:



Blue = y(x), Green = y'(x)

b) Lastly, we rotate the solution curve for y(x) around the x-axis to obtain a 3D figure. The code for this is in appendix D, and the figure itself is shown below.



Appendix A

```
x = -10:0.01:10; % x values
y = @(x) 7*x-20*cos(3*x-5); % the function

hold on
plot(x,y(x));
plot(x, 0); %
hold off


format long, xstart = [-2 -1 0.01 1 2];
solution = [];
for i =1:5
   x = xstart(i); dx=x; iter = 0;
   disp(i)
   while abs(dx/x) > 0.5e-9
      dx = -(7*x-20*cos(3*x-5))/(7-60*sin(5-3*x));
      disp([x dx]), x=x+dx; iter= iter +1;
   end
   solution =[solution; x];
   disp('No. of iterations')
   disp(iter)
   K = abs ((180*cos(5-3*x))/(2*(7-60*sin(5-3*x) )));
   disp('K value')
   disp(K)
   disp('-------------------------')

end


xp = @(x) abs (7/8 - (15.*sin(3.*x - 5))./14);
roots =[-1.775068882661698 -1.080616604050983 0.085852685356389 1.300604498191414
1.941264550515620];

xp(roots)

% conclusion, the 1st, 3rd, and 5th roots will converge because this
% function is less than 1 (function xp).


f = @(x) (7./8).*x+(5./14).*cos(3.*x-5); % our function
y=xstart;

for i=1:100
   y = f(y);
end
```

solution
y'
solution-y'

Appendix B

```
%% Exercise 2 - Numerical Approximation of Integrals

%%
% *Part A*

% A)
f = @(x) sqrt(x+2);
x = -1:0.01:1;
plot(x,f(x))
% we think it is approx. the area of a triangle, 0.7*2/2 = 0.7.

% B)
% The integral is equal to
xTheTruth = 2.7974349485;

% C)
T = @(h) h*(sum(f(-1:h:1)) - ((f(-1)+f(1))/2))

headings = {'h';'T(h)';'E_t = T(h)-I';'T_extr(h)';'E_T_extr = T_extr(h)-
I'};
a = {'h'};
h =  [1 0.5 0.25 0.125 0.0625];

ThRow = [];
ErTh = [];
ExTh = [];
TxTr =[];
for i = h
    temp = T(i);
    ThRow = [ThRow; temp];
    ErTh =[ErTh;abs(temp-xTheTruth)];
    temp2= (T(i/2)+(T(i/2)-T(i))/3);
    ExTh = [ExTh; temp2];
    TxTr = [TxTr; abs(temp2 - xTheTruth)];
end
h=h'
tab = table(h,ThRow,ErTh,ExTh,TxTr);
tab

% finding the exponent (in the error)

loglog(h,ErTh)

% We use log to find the slope of the line = k (using log laws).
coefficients = polyfit(log(h), log(ErTh), 1);
k = coefficients(1)
% This is approx equal to 2. Hence quadratic.

% This is for the extrapolation error.
```

```
coefficients = polyfit(log(h), log(TxTr), 1);
k = coefficients(1)

hold on
loglog(h,TxTr)
% which is approx = 4. = fourth degree

%%
% *Part B*
t = [1/8 1/16 1/32 1/64];

ThRow = [];
ErTh = [];
ExTh = [];
TxTr =[];
for i = t
    temp = T(i);
    ThRow = [ThRow; temp];
    ErTh =[ErTh;abs(temp-xTheTruth)];
    temp2= (T(i/2)+(T(i/2)-T(i))/3);
    ExTh = [ExTh; temp2];
    TxTr = [TxTr; abs(temp2 - xTheTruth)];
end

loglog(t,ErTh);


coefficients = polyfit(log(t'), log(ErTh), 1);
k = coefficients(1)
% This is approx equal to 2. Hence quadratic.

% This is for the extrapolation error.
coefficients = polyfit(log(t'), log(TxTr), 1);
k = coefficients(1)

% the first is same as before, but for the extrapolation one it is closer
% to 3 than 4.
```

Appendix C

```
%% Exercise 3 - Intersection between Elipses
clear
x = 0:0.01:2*pi;
x2 = 0:0.01:8;
r1 = @(a) sqrt(10./(0.4.*cos(a).^2 + sin(a).^2  - cos(a).*sin(a)))
f1 = @(y) sqrt(16 *(1- ((y-2).^2)/36 )) +4
f2 = @(x) sqrt(36 *(1 - ((x-4).^2)./16 )) +2

f3 = @(x) (3/2)*sqrt(-(x-8).*x)+2
f4 = @(x) 2-(3/2).*sqrt(-(x-8).*x)


hold on
polar(x,r1(x))
plot(x2,f3(x2))
plot(x2,f4(x2))
```

```
init = [0.1 3; 1.3 -2.5; 7 4.8; 8 3.3]';
J = @(v, y) [0.8*v-y 2*y-v; 2*(v-y)/16 2*(y-2)/36];
f = @(v, y) [0.4*v^2 + y^2 - v*y - 10; ((v-4)^2)/16 + ((y-2)^2)/36 - 1];

answers = [];
error1 = [];
error2 = [];
error3 = [];
error4 = [];

for i = 1:4
    x0 = [init(1, i) init(2, i)]';
    iter = 0;
    dxnorm = 1;

    while dxnorm > 0.5e-4 & iter < 20
        fval = f(x0(1), x0(2));
        jval = J(x0(1), x0(2));
        dx = - jval\fval;
        x0 = x0 + dx;
        dxnorm = norm(dx, inf); iter = iter + 1;
        if (i == 1)
            error1 = [error1; dxnorm];
        end
        if (i == 2)
            error2 = [error2; dxnorm];
        end
        if (i == 3)
            error3 = [error3; dxnorm];
        end
        if (i == 4)
            error4 = [error4; dxnorm];
        end
    end

    answers = [answers; x0'];
end
answers = answers';


table(error1)

%disp(answers(1, 1) +''+ answers(2, 1))

fprintf('Actual value: %d %d\n', answers(1, 1), answers(2, 1))
table(error2)
fprintf('Actual value: %d %d\n', answers(1, 2), answers(2, 2))
fprintf('Note that this iteration does not actually converge to the point
it is supposed to converge to, but rather to another nearby point, hence
the increase in the error at the beginning.\n')
table(error3)
fprintf('Actual value: %d %d\n', answers(1, 3), answers(2, 3))
table(error4)
fprintf('Actual value: %d %d\n', answers(1, 4), answers(2, 4))

fprintf('\n')
disp('It is easy to see that the error decreases quadratically for each
iteratoion')
```

```
plot(answers(1, 1), answers(2, 1), 'o')
plot(answers(1, 2), answers(2, 2), 'o')
plot(answers(1, 3), answers(2, 3), 'o')
plot(answers(1, 4), answers(2, 4), 'o')
```

Appendix D

```
ds2 = @(t,s) s(1)/9 - s(1)*pi*exp(t/3)*(2*s(2)*sin(pi*t)+pi*s(1)*cos(pi*t))
f = @(t,s) [s(2); ds2(t,s)]'

tslut = 2.6;
t = 0;

 h= 0.1;
 Y = [];
 T = [];
 y = [1 -1/3]
 while t < tslut
    f1 = f(t,y);
    f2 = f(t+h/2,y+h*f1/2);
    f3 = f(t+h/2,y+h*f2/2);
    f4 = f(t+h,y+h*f3);

    T=[T;t];
    Y=[Y;y];

    y = y +(h/6)*(f1+2*f2+2*f3+f4);
    t = t+h;

 end

L = Y(:,1)
L(end) % 0.219431472260047 för h=0.1
L(end) % 0.305364907441336 för h=0.2

figure
plot(T,Y)

phi = 0:2*pi/30:2*pi;

X = T*ones(size(phi));
Y2 = Y(:,1)*cos(phi);
Z = Y(:,1)*sin(phi);
surf(X,Y2,Z)
```