Et billede, der indeholder tekst

Automatisk genereret beskrivelse

We know that B must lie in the range

Thus:

Thus A) must be correct.

Et billede, der indeholder tekst

Automatisk genereret beskrivelse

Writing :

2) is the answer.

Et billede, der indeholder tekst

Automatisk genereret beskrivelse

Second interpolation-error theorem

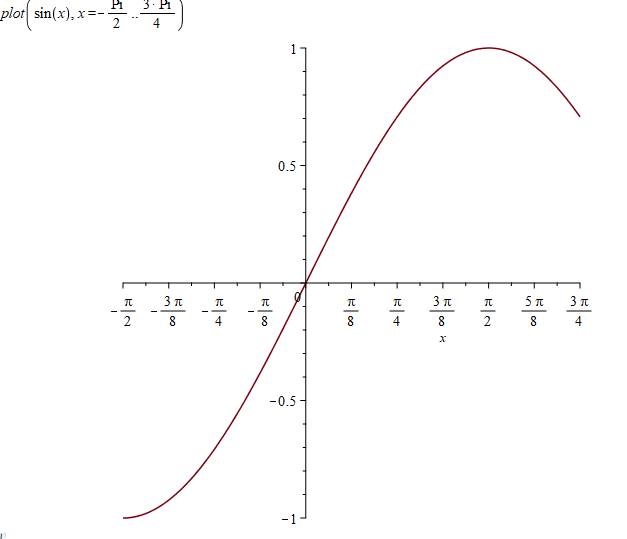
We use interpolation-degree n. .

We know that the derivative takes the largest value at ;

hmm.. Try and calculate the function-values. It seems they all have way to high accuracy, and thus n = 21 (1) is the right answer.

Et billede, der indeholder tekst

Automatisk genereret beskrivelse





Thus, secant method would fail (start-guess is horizontal secant). Answer 2 is correct.

Et billede, der indeholder tekst

Automatisk genereret beskrivelse

Global error is .

We thus know:

This constant is the proportionality. Thus:

This is answer 2.

Et billede, der indeholder tekst

Automatisk genereret beskrivelse

We have the error-term:

with . Thus, it must be between 0 and pi/2.

We have the function:

This differentiated four times is:

The max of this function is when , as , and the functionvalue thus is bounded by .

Now, we can use an inequality:

Which is solved for :

This gives us:

As we are using the composite simpsons rule we need equal of subintervals. Thus:

Et billede, der indeholder tekst

Automatisk genereret beskrivelse

Using MySimpson:



We get , and an error with the magnitude which is in the bound.

Probably we could use way smaller n in practice, as theoretical upper bound has big overhead for small n.

Et billede, der indeholder tekst

Automatisk genereret beskrivelse

The basis-functions are:

Thus, matrix takes form:

Which gives us the complete normal equation:

Finding :

Using matlab. Note: we do not explicitly find the inverse, but solve the system , as this is more numerically stable:



Which gives:

Et billede, der indeholder tekst

Automatisk genereret beskrivelse

F = c(1)\*sin(pi\*x)+c(2)\*cos(pi\*x);

err = abs(y-F)

errSum = sum(err)

gives us the errsum (quite bad for such a small interval).

Et billede, der indeholder tekst

Automatisk genereret beskrivelse

The relative error of the solution is bound by:

Giving us:

Also, for the perturbation in :

We know

Multiplying the two equation gives the relative error in c:

Using

And we are given the relative error of the perturbed data. Thus:

The upper bound for the error in c is 1.5%.

Et billede, der indeholder tekst

Automatisk genereret beskrivelse

The Euclidean distance may be written as:

The hint suggests we write , and thus we wish to minimize:

The mission is now to find a root for .



We have



Start-guess is newer part of first iteration, brother.

Et billede, der indeholder tekst

Automatisk genereret beskrivelse

It must be positive definite and be real and symmetric.

Et billede, der indeholder tekst

Automatisk genereret beskrivelse

Calculating would cost Lops. Also, we know, that calculating the Cholesky factorization takes

Lops. Thus, we would have

In total for strategy one, and the dominating term is .   
  
In strategy 2 we would first use the Cholesky-factorization:

Which would give us a lower triangular with only half the elements. Now we solve the system with half as many elements, but need to solve four triangular systems (2 with L and 2 with LT). This costs in total Lops. In total:

Which thus has dominating term . This Is at least 6 times smaller.

Et billede, der indeholder tekst

Automatisk genereret beskrivelse

Introducing the following dictionary of variables:

|  |  |  |  |
| --- | --- | --- | --- |
| Old variable | New variable | Initial value | Differential eq. |
|  |  |  |  |
|  |  |  |  |

Now, we need to implement a Matlab-function which can write rhs of diff.eq. column:



Et billede, der indeholder tekst

Automatisk genereret beskrivelse

Thus, we don’t hit with any of the guesses.

Et billede, der indeholder tekst

Automatisk genereret beskrivelse

The secant-method gives:

Giving a new shooting:



Which \*almost\* hits zero but still has a small error. This could be caused by 1. machine precision or 2. is not a completely linear function (and thus secant method needs more iterations).

ics1 = [1,-1];

ics2 = [1,-5];

tspan = [0,2];

[t\_sol\_1,x\_sol\_1] = ode45(@odefun,tspan,ics1);

[t\_sol\_2,x\_sol\_2] = ode45(@odefun,tspan,ics2);

z1 = -1;

z2 = -5;

f1 = x\_sol\_1(end,1)-0;

f2 = x\_sol\_2(end,1)-0;

%secant

z\_root = z2-((z2-z1)/(f2-f1))\*f2;

[t\_sol\_3,x\_sol\_3] = ode45(@odefun,tspan,[1,z\_root]);

plot(t\_sol\_3,x\_sol\_3(:,1))

hold on

plot(t\_sol\_2,x\_sol\_2(:,1),'.')

plot(t\_sol\_1,x\_sol\_1(:,1),'.')

plot(2,0,'rx');

legend('shot 3','shot 2','shot 1','goal')

function [X] = odefun(t,x)

X = [x(2);-4\*x(2)-4\*x(1)+exp(-2\*t)];

end

Et billede, der indeholder tekst

Automatisk genereret beskrivelse

We have a system matrix:

And

Gaussian elimination with partial pivoting:   
We want to reduce the system to a upper triangular.

Swap row 1 and 2 and 2 and 3:

The operations are thus carried out by the permutation matrix, where row 1 becomes row 3, row 2 becomes row 1 and row 3 becomes row 2

And with matrix-operations:

Now we kill column 0 in r2 and r3:

Tada, now it is upper triangular. The same operations must be applied to b:



Doing back-substitution:

Thus the solution is:

Et billede, der indeholder tekst

Automatisk genereret beskrivelse

The basis-functions are:

Thus the coefficient matrix is:

The normal equation is:

Which gives the system matrix:

And the right hand site:





Et billede, der indeholder tekst

Automatisk genereret beskrivelse

Yes we could. In every evaluation we are solving a big system of equations with gaussian elimination, which costs LOps for the elimination on every iteration and following LOps for backward/forward-sub for getting the coefficients.. As long as the left-hand-side is the same, we could save some computing power by using a LU-factorization of the system-matrix. The factorization would cost us (when using LU) or LOps when using Cholesky, but can be reused in every data-fit. Afterwards we would use forward/backward-sub which still costs LOps. Thus we would save the gaussian elimination part.

Et billede, der indeholder tekst

Automatisk genereret beskrivelse

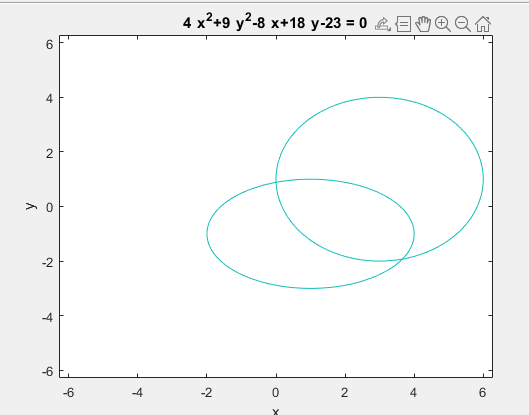
Et billede, der indeholder tekst

Automatisk genereret beskrivelse

The Jacobian may simply be found as:

**Et billede, der indeholder tekst

Automatisk genereret beskrivelse**

****

I guess I would choose and ? (must be intersection).

Et billede, der indeholder tekst, orange, mørk, luk

Automatisk genereret beskrivelse

X0 = [4;-2];

x = Newtonsys(@funFdF\_1,X0,10)

Sidste kolonne I x er dit svar.

Et billede, der indeholder tekst

Automatisk genereret beskrivelse

We have:

We calculate :

is:

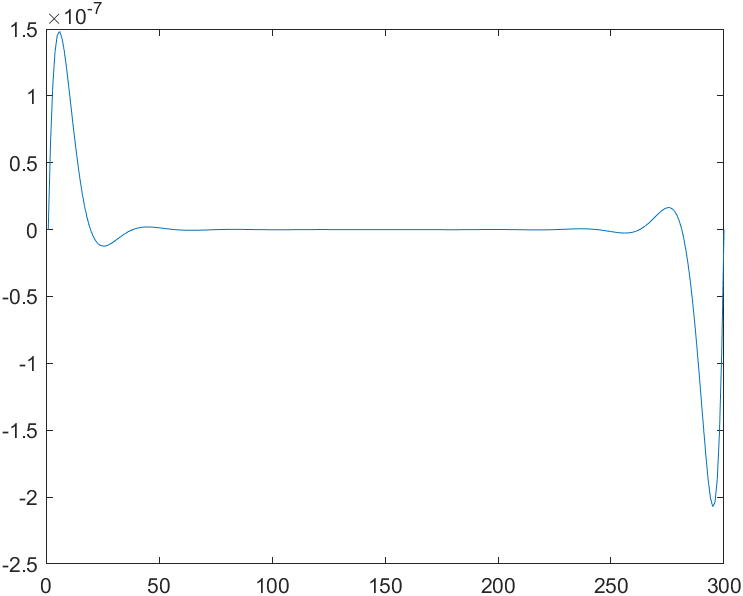
We know that serves upper bound to the function on the whole interval. The largest value the derivative takes is at the point :

Thus, we can find a value:

!(REMEMBER n and not n+1)!

Et billede, der indeholder tekst

Automatisk genereret beskrivelse



The magnitude is smaller during the whole sample.

Et billede, der indeholder tekst

Automatisk genereret beskrivelse

We have the new dictionary:

|  |  |  |  |
| --- | --- | --- | --- |
| Old variable | New variable | Initial value | Differential eq. |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Thus the system becomes:







In each iteration there are 4 function evaluations. Thus multiply these two numbers.

Et billede, der indeholder tekst

Automatisk genereret beskrivelse

For :

for :

Et billede, der indeholder tekst

Automatisk genereret beskrivelse

A3

Et billede, der indeholder tekst

Automatisk genereret beskrivelse

B4)

Et billede, der indeholder tekst

Automatisk genereret beskrivelse



C2)

Et billede, der indeholder tekst

Automatisk genereret beskrivelse

D2).

Et billede, der indeholder tekst

Automatisk genereret beskrivelse

We know that:

Thus, the worst result occurs at , and our derivative-function is bounded by:

We can calculate :

Thus, we wish that:

This is solved for :

Which means, that the smallest is 72 (must be equal).

E3.

Et billede, der indeholder tekst

Automatisk genereret beskrivelse

A1.

Et billede, der indeholder tekst

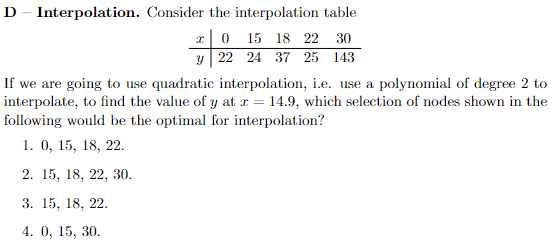
Automatisk genereret beskrivelse

B2).

Et billede, der indeholder tekst

Automatisk genereret beskrivelse

C2.



3 – because we want close points

Et billede, der indeholder tekst

Automatisk genereret beskrivelse

E4).

Et billede, der indeholder tekst

Automatisk genereret beskrivelse

Se flop-count I PDF og beregn hasithged.

Et billede, der indeholder tekst

Automatisk genereret beskrivelse

I would use cardinal-polynomials.

And thus, the interpolating polynomial becomes:

Which is a polynomial of order 2.



Second interpolation error theorem is applied:

We need an upper bound for . This is largest when , and thus:

Et billede, der indeholder tekst

Automatisk genereret beskrivelse

Introducing the dictionary:

|  |  |  |  |
| --- | --- | --- | --- |
| Old variable | New variable | Initial value | Differential eq. |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Thus, the linearized system:

With .



Et billede, der indeholder tekst

Automatisk genereret beskrivelse

tspan = [0,2\*pi];

n = 50;

h = 2\*pi/50;

z0 = [0.4;0;0;2];

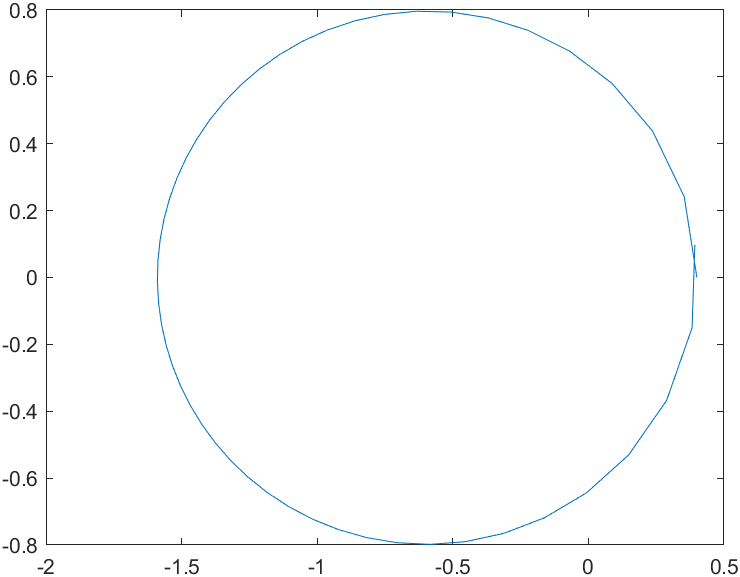
[t\_sol,x\_sol] = MyRK4System(@odefun,tspan,z0,n)

function [Z] = odefun(t,z)

Z = [z(3);z(4);-z(1)/(z(1)^2+z(2)^2)^(3/2);-z(2)/(z(1)^2+z(2)^2)^(3/2)];

end

The solution becomes:



plot(x\_sol(1,:),x\_sol(2,:))

Et billede, der indeholder tekst

Automatisk genereret beskrivelse

3.1)

We know, that the fourth derivative is:

This value is the largest at , as . Thus, we know that the error has to be bounded by:

Thus, we can solve for which gives us:

As we are using the composite Simpsons rule, n must be even. Thus, the smallest number of subintervals is .

3.2)

a = 1;

b = 3;

fun = @(x)x\*log(x);

A = MySimpson(fun,a,b,n)

Gives

Et billede, der indeholder tekst

Automatisk genereret beskrivelse

Et billede, der indeholder tekst

Automatisk genereret beskrivelse