Lösningar Utvalda Wiki VT18

Solutions block 0:

Solution 1-0-01

Ett flyttal är en approximation av ett reellt tal. Flyttalet består av ett tecken, en mantissa och en exponent. Flyttalet kan per definition beskrivas som $\boldsymbol{x = \pm m*2^n}$. Där $\boldsymbol{m = 1.bb...b_2}$ som är ett tal i intervallet [1,2) och **n** *som* är ett heltal. Antalet siffror som används i mantissan och exponenten avgörs av flyttalets precision. Att flyttalet har enkel precision innebär att talet kan beskrivas med 32 bittar och dubbel precision innebär att talet kan beskrivas med 64 bittar vilket även innebär att mantissan består av 52 binära siffror.

Moderator: Korrekt.

Solution 1-0-02

Regeln är: kancellation uppstår när vi subtraherar två nästan lika tal.

a) Om x ungefär y

b) Ingen subtraktion eller addition. Det uppstår aldrig.

c) Om x ungefär -y. Det är samma sak som att addera x och -y

d) Kancellation uppstår när x ungefär y, eller x ungefär -y.

Moderator: Korrekt. **Utmärkt!**

Solution 1-0-03

Vi har att:

$9.25 \ = 8 + 1 + 0.25 = \ 2^3 \ + \ 2^0 \ + \ 2^{-2}$

Vilket ger att flyttalsrepresentationen är

$+1.0010100...0_2 \cdot 2^3$

Ty detta är ekvivalent med

$(1 \cdot 2^0 + 0 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} + 0 \cdot 2^{-4} + 0 \cdot 2^{-5}) \cdot 2^3 = 2^3 + 2^{-3} \cdot 2^3 + 2^{-5} \cdot 2^3 = 2^3$ -

Moderator: Korrekt!

Solution 1-0-04

a. Absolutfelet e_x är differensen mellan det exakta värdet på x och approximationen x_{approx} av x, alltså:

$e_x = x - x_{approx}$

Vi definierar också för e_x en absolutfelgräns E_x ,

$|e_x| \leq E_x$, som ger en begränsning till absolutfelet.

b. Relativfelet r_x visar hur bra ett resultat är i relation till resultatets storlek. Vi definierar det som följande:

$r_x = \frac{e_x}{x} = \frac{x - x_{approx}}{x}$

Vi definierar också för r_x en relativfelsgräns R_x ,

$|r_x| = \frac{|r_x|}{|x|} = \frac{|x - x_{approx}|}{|x|} \leq R_x$, som ger en begränsning till relativfelet.

Moderator: Korrekt.

Solution 1-0-05

Vi får

$e_z = z - \tilde{z} = x - y - (\tilde{x} - \tilde{y}) = x - y - \tilde{x} + \tilde{y} = x - \tilde{x} - (y - \tilde{y}) = e_x - e_y$

detta ger

$|e_z| = |e_x - e_y| \leq |e_x| + |e_y| \leq E_x + E_y$

V.S.V

Moderator: Korrekt.

Solution 1-0-06

$1.25_{10} = 1_{10} + 0.25_{10}$

$1_{10} = 1_2$ and $0.25_{10} = 0.01000..._2$

To convert a fractional number of base 10 to binary use the following method using 0.25_{10} as an example:

Continually multiply the fractional part of the number by two

0.25 * 2 = **0**.5

0.5 * 2 = **1**.0

0.0 * 2 = **0**.0 (here we see the pattern will be repeating zeros)

The integer part of the products is the binary representation of the number, in this case 0.010000...

Therefore $1.25_{10} = 1.01000..._2$ which represented as a normalized floating point number is

$+1.010...0_2 \cdot 2^0$

Moderator: Correct

Solution 1-0-07

Maskinnoggrannhet innebär maskinens förmåga att representera ett tal med en specificerad noggrannhet.

Noggrannheten beräknas genom att subtrahera 1 med det högsta flyttalet $0 < x < 1$ som maskinen kan representera. Denna noggrannhet beror på antalet bitar i mantissan.

Moderator: Bra. Den vanliga definitionen av maskinnoggrannhet är avståndet mellan 1 och det närmsta tal större än ett.

Solution 1-0-10

Antag att relativfelet i x är r_x och relativfelet i y är r_y. Då gäller

$$\frac{\tilde{x}-x}{x} = r_x$$

dvs $\tilde{x} = x(1+r_x)$ och motsvarande för y: $\tilde{y} = y(1+r_y)$.

Vi säger att $\tilde{z} = \tilde{x}\tilde{y}$ är en approximation av $z = xy$. Relativfelet blir

$$r_z = \frac{\tilde{z}-z}{z} = \frac{\tilde{x}\tilde{y}-xy}{xy} = \frac{x(1+r_x)y(1+r_y)-xy}{xy}$$

som kan förenklas ytterligare

$$r_z = \frac{(1+r_x)(1+r_y)-1}{1} = r_x + r_y + r_x r_y \approx r_x + r_y$$

där vi försummat termen $r_x r_y$ som är väldigt liten om r_x och r_y är små. Därmed är

$$|r_z| \approx |r_x + r_y| \leq R_x + R_y$$

Moderator: Korrekt. (Jag skrev den.)

Solution 1-0-13

$E_x = |x - \tilde{x}| = 0.1$. Beräkna derivatan $f'(x) = e^x - 3x^2$ för att kunna uppskatta absolutfelgränsen för y.

Felfortplantningsformeln ger $|y - \tilde{y}| \leq E_y \approx E_x|f'(\tilde{x})| = 0.1|e^{(2.1)} - 3(2.1)^2| \approx 0.506$.

I Matlab tar vi skillnaden mellan de beräknade värdena på y och $\tilde{y}$:

```
fp = @(x)exp(x) - 3 * x^2;
x = 2;
xtilde = 2.1;
y = fp(x);
ytilde = fp(xtilde);
ey = abs(y - ytilde)
```

Detta ger utskriften

```
ey = 0.452886186363001
```

Det beräknade värdet på felet i y är som förväntat mindre än absolutfelgränsen E_y, men i samma storleksordning.

Moderator: Korrekt. (Jag löste den.)

Solution 1-0-15

Använd allmänna felfortplantningsformeln: $E_f = \left|\frac{\partial f}{\partial x}\right|_{x=\tilde{x},y=y} E_x + \left|\frac{\partial f}{\partial y}\right|_{x=x,y=\tilde{y}} E_y$.

De partiella derivatorna av funktionen $f(x,y)$ blir

$\frac{\partial f}{\partial x} = -\sin(x)e^{x+y} + \cos(x)e^{x+y}$, $\frac{\partial f}{\partial y} = \cos(x)e^{x+y} + 3\cos(3y)$.

Med $\tilde{x} = x + E_x = 1.2 + 0.1 = 1.3$ och $\tilde{y} = y + E_y = 0.9 + 0.05 = 0.95$får vi

$E_f = \left|-\sin(1.3)e^{1.3+0.9} + \cos(1.3)e^{1.3+0.9}\right| \cdot 0.1 + \left|\cos(1.2)e^{1.2+0.95} + 3\cos(3 \cdot 0.95)\right| \cdot 0.05$
$\approx 0.628 + 0.012 = 0.64$.

Moderator: Korrekt (jag skrev den).

Solution 1-0-17

Kancellation uppstår i det här fallet då x är nästan lika med noll. Om x är nästan lika med noll kommer vi subtrahera två nästan lika tal, 1 och något tal lite större än 1.

Vi kan skriva om uttrycket för att undvika kancellation genom att:

1)Multiplicera nämare och täljare med uttryckets konjugat: $\frac{\left(1-\sqrt{1+x^2}\right)\left(1+\sqrt{1+x^2}\right)}{\left(1-\sqrt{1+x^2}\right)}$

2) Förenkla sen uttrycket: $\frac{1-\left(1+x^2\right)}{\left(1+\sqrt{1+x^2}\right)}$

3) Med förenkling får vi till slut uttrycket: $\frac{-x^2}{1+\sqrt{1+x^2}}$

Moderator: Korrekt!

Solution 1-0-19

a) När x och y är väldigt lika värden. Tex om $x = 4$ och $y = 3.99999999$.

b) Om $x + y$ är väldigt nära värdet -4. Tex om $x = -2.0000001$ och $y = -2$

c) Om $x * y$ är väldigt nära värdet 9. Tex om $x = y = 3.0001$

Moderator: Svaret på b kräver lite närmare analys (dvs överkurs). Det är inte självklart vilken ordning operationerna i a+b+c genomförs. Kompilatorer/processorer har ibland ändrat ordning på operationerna för att öka prestanda. (Kuriosa: Det ledde till ett säkerhål som rapporterades först januari 2018.)

Solution 2-0-01

(Absloutfelet) e_x =0.03 km

(Relativtfel) R_x=0.03/23.5=0.00127659 km

AbsloutFel är bättre eftersom den är större så att vi är säkra att felet inte kommer och blir större än absloutfelet

Moderator: Korrekta beräkningar! Vid numeriska beräkningar används oftast det relativa felet då detta ger en rättvisare bild av felet i förhållande till vad som beräknas.

Solution 2-0-02

Låt $\tilde{x}$ och $\tilde{y}$ vara approximationer för x och y, och låt E_x och E_y vara felgränser. Dvs. $|\tilde{x} - x| \leq E_x$ och $|\tilde{y} - y| \leq E_y$. Vill nu bestämma felgränsen E_z för $\tilde{z} = \tilde{x} + \tilde{y}$.

$$|\tilde{z} - z| = |(\tilde{x} + \tilde{y}) - (x + y)| = |(\tilde{x} - x) + (\tilde{y} - y)| \leq E_x + E_y = E_z$$

Moderator comment: Nice solution!

Solution 2-0-03

654 Till binärsystem är 1010001110_2

0.321 Till Binärsystem är 0.01010010001011010001_2

654.321= $1010001110.01010010001011010001_2$

$654.321*2^0=654.321$

$1010001110.01010010001011010001*2^0=654.321$

$0.10100011100101001000101101000 1 * 2^{10}=654.321$

$0.10100011100101001000101101000 1 * 2^{1010}=654.321$

I 8 bitar=

Sign	Exponent	Mantissa
0(Positive)	101	1010

I 16 bitar (Halv)=

Sign	Exponent	Mantissa
0 (Positive)	10100	1010001110

Moderator: Konverteringen till binär form är korrekt. Tänk på att mantissans ledande term är 1 och ej 0 så att exponenten blir 9 istället för 10.

På s.11 i Sauer beskrivs hur talet kan representeras på en maskin. Mantissan representeras då som följden $bbb...b$ efter decimalpunkten 1. och exponenten som det tal du får om du i det första fallet (8 bitars representation) adderar $2^2 - 1$ till exponenten 9 så att exponenten i binär form blir 110 och i det andra fallet adderar $2^4 - 1$ så att exponenten blir 11000.

Solution 2-0-11

Att effektivt lösa ett linjärt ekvationssystem resulterar i cirka $\frac{2}{3}n^3$ flops (Sauer kap 2.1.2). Detta ger oss approximationen $t(n) = \frac{2}{3}n^3 * K$ för tiden t det tar att lösa ett ekvationssystem i n variabler, där K är en konstant som beror på hård- och mjukvara. Vi har vidare att $t(100) = \frac{2}{3} * 100^3 * K = 10$ och således får vi approximationen
$t(500) = \frac{2}{3}(5 * 100)^3 * K = 5^3 * t(100) = 125 * 10 = 1250$

Svar: 1250 sekunder

Solution 2-0-20

$f(x_0, y_0)$ without any errors:

$f = f(2,\ 3) = -0.6956$

$f(x_0, y_0)$ with maximum lower error:

$f_{\text{min}} = f(1.7,\ 2.8)\ =\ -0.1277$

$f(x_0, y_0)$ with maximum upper error:

$f_{\text{max}} = f(2.3,\ 3.2) = 0.2142$

The above gives following:

Lower uncertainty

$|f - f_{\text{min}}| = |-0.6956 - (-1.127)| = 0.4321$

Upper uncertainty

$|f - f_{\text{max}}| = |-0.6956 - 0.2142| = 0.9097$

Therefore, the lower and upper uncertainties in the value of $f(x_0, y_0)$ are 0.4321 and 0.9097, respectively.

Moderator: Correct. You did a detailed analysis based on the specific functions. For more complicated functions we normally use AFFF:

$$E_f \approx |\frac{\partial f}{\partial x}|E_x + |\frac{\partial f}{\partial y}|E_y$$

In our case $\frac{\partial f}{\partial x} = -\sin(x) + y\cos(xy) \approx 1.97$ and $\frac{\partial f}{\partial y} = x\cos(xy) \approx 1.92$

Therefore,

$$E_f \approx 1.97 * 0.3 + 1.92 * 0.2 = 0.975$$

which is consistent with your (more exact) bound 0.9097.

Solution 2-0-22

Kancellation uppstår då två nästan lika stora tal subtraheras eller någon analog uträkning utförs, vilket leder till ett stort relativt fel.

I det här fallet kan vi se fyra situationer då detta kan uppstå.

1) I den första termen uppstår kancellation då $x \approx y$ i nämnaren.

2) Då den första och den andra termen adderas uppstår kancellation om $y \approx x+(4x)^{-1}$.

3) Då den tredje termen subtraheras från den andra termen uppstår kancellation om $y \approx 20x^2+8x$.

4) Om hela uttrycket är nästintill noll förekommer också kancellation. Det finns två punkter med omnejd där detta uppstår, (-0.40, -0.22) och (0.29,2.15).

Solution 3-0-01

Bevis för detta finns på sida 6 i block0.pdf

Absolutfel adderas vid addition och subtraktion.

Sida 6 säger att absolutfelsgränserna adderas. Om vi har absolutfelen e_x, e_y, e_z och absolutfelsgränserna E_x, E_y, E_z så att $\begin{cases} |e_x| \leq E_x \\ |e_y| \leq E_y \\ |e_z| \leq E_z \end{cases}$, så säger sida 6 att:

$|e_z| \leq E_x + E_y$

För multiplikation får vi att relativfelsgränserna adderas istället. Om vi har relativfelen r_x, r_y, r_z och relativfelsgränserna R_x, R_y, R_z så att $\begin{cases} |r_x| \leq R_x \\ |r_y| \leq R_y \\ |r_z| \leq R_z \end{cases}$, så säger sida 6 att:

$|e_z| \leq R_x + R_y + R_x R_y \approx R_x + R_y$

Detta ger en relativfelgräns för z som är ungefär lika med summan av relativfelgränserna för x och y. OBS Detta gäller eftersom vi antar att R_x, R_y är mycket små, annars är $R_x R_y$ ej försumbar.

Moderator: Korrekt! Bra!

Solution 3-0-04

Vi vet att $x = 0.94$. Notera att detta x är en approximation eftersom vi har fått givet en felgräns på detta värde.

Enligt uppgift är $E_x = 0.02$.

Enligt sida 18 i pres_block0.pdf kan vi använda formeln för AFFF (Almänna felfortplantningsformeln):

$E_y \approx E_x|F'(\tilde{x})|$, där $F(x) = cos(x) + e^x$

Jag anser att 0.02 är en ganska liten felgräns (detta var kravet för att få använda formeln)

Vi deriverar $F(x)$ term för term:

$F'(x) = -sin(x) + e^x$

Detta ger en absolutfelgräns för y som är

$E_y = E_x|F'(\tilde{x})| = 0.02 \cdot |-sin(0.94) + e^{0.94}| \approx 0.035048466358483$

Svar: En felgräns för $y(x) = cos(x) + e^x$ är ca 0.035

Moderator: Korrekt!

Solution 3-0-05

$123.456_{10} = 1111011.0111_2... = 1.1110110111_2... \times 2^6 = 1.1110110111_2... \times 2^{110_2}$

8-bitar:

Tecken: 0 (pos). Exponent: 110. Mantissa: 1110.

Halv:

Tecken: 0 (pos). Exponent: 00110. Mantissa: 1110110111.

Moderator: Korrekt!

Solution 3-0-06

Givet:

$x = 3$

$E_x = 0.1$

E_x uppskattas vara litet då det i uppgiften anges att AFFF ska användas för att lösa uppgiften. När det är uppfyllt och y=f(x) gäller:

$E_y \approx E_x|f\prime(\tilde{x})|$ där $f(x) = sin(\pi x)$ & $f\prime(x) = \pi cos(\pi x)$

Absolutfelgränsen för y är då lika med

$E_y \approx 0.1 \cdot |\pi cos(3\pi)| \approx 0.314159265358979$

Moderator: Bra!

Solution 3-0-07

Vi gör uppskattning med O-kalkyl. Gauseliminering kräver O(n^3) flops. Det betyder att för stora n är tidsåtgången ungefär

$a * n^3$

för någon konstant a. Ett problem med n=1000 krävde enligt uppgift 5 minuter, dvs

$5 \text{ minuter} \approx a \cdot 1000^3$

Tidsåtgången för det större systemet blir uppskattningvis

$a * 10000^3 = a * 1000^3 * 10^3 \approx 10^3 * 5 \text{ minuter}$

Med andra ord: 5000 minuter som är ungefär 83 timmar. (Oj!)

Moderator: Tydligt och korrekt!

Solution 3-0-09

a) Låt z vara den punkt Anna är efter att hon har sprungit fram och tillbaka, alltså z= x-y = 1000-990 = 10

Anna är således 10m i den riktning hon sprang först.

b) Om stegmätaren gör i snittett fel på 0.01 m för varje meter så innebär det att felgränserna blir$E_x = 1000 \cdot 0.01 = \ 10 \ och \ E_y = 990 \cdot 0.01 = 9.9$

Det följer sig av definition av relativfelgränser att relativfelgränsen blir

$|r_z| = \frac{|z - \tilde{z}|}{|z|} \leq \frac{E_x + E_y}{|z|} = \frac{10+9.9}{|10|} = \frac{19.9}{10} = 1.99$

vilket är inte särskilt bra, dvs nästan 200% osäkerhet.

c) Kancellation uppkommer när två nästan lika tal subtraheras med varandra, i detta exempel kan det uppstå när operationen z=x-y utförs.

Moderator: Bra!

Solution 3-0-19

Det följer direkt från definitionen av relativfel, att talet som efterfrågas är

$r_v = \frac{l \cdot l \cdot l - V}{V}$,

där den faktiska sidolängden betecknats med l, och den önskvärda volymen betecknats med V. substitution av de angivna värdena ($l := 3.02,\ V := 3 \cdot 3 = 9$) ger då relativfelet

$r_v = \frac{3.02^3 - 9}{9} = \frac{9.124 - 9}{9} = \frac{0.124}{9} = 0.012\overline{7}.$

Svar: $0.012\overline{7}$

Moderator: Bra. Du har gjort en detaljerad felanalys. För mer komplicerade funktioner behöver man använda AFFF, som i detta fall blir om vi sätter $f(x) = x^3$:

$$E_f \approx |\frac{\partial f}{\partial x}| E_x = 3 * (3^2) * 0.02 = 0.54$$

En relativfelgräns kan man få från relationen

$$R_f = \frac{E_f}{f} \approx 0.02.$$

Din mer detaljerade analys gav alltså en aningen bättre feluppskattning.

Solution 3-0-22

Den naiva implementationen av Gauss-eliminering har en tidskomplexitet av n^3, då n i n x n matrisen multipliceras med 4 bör beräkningstiden uppskattas bli ungefär 4^3 = 64 gånger så lång. Det är en rimlig uppskattning, men tyvärr blir det oftast inte så i praktiken eftersom det beror på processen, och det tillkommer en viss tidskostnad när man kör programmet för första gången. Skulle man köra programmet många gånger och sedan ta ett medelvärde på hur mycket extra tid det tog bör man rimligtvis komma ganska nära 64.

Solution 4-0-01

A single precision floating point number consists of 32 bits and is used to represent a real number. It is merely an approximation as computers does not have the memory required to represent all real numbers. A double precision floating point number on the other hand consists of 64 bits and can therefore represent a number with a higher degree of precision than the previously mentioned type. A half precision floating point number consists of 16 bits and does not have the same precision as the other two types, but it requires less memory instead.

Moderator: Correct!

Solution 4-0-03

Det beror på att 0.1 och 0.2 är flyttal och därför kommer att uppskattas som flyttal enligt IEEE 754-standard. Uppskattningen innebär att de riktiga värdena på talen representerade i datorn får ett relativfel $r \leq \frac{\varepsilon_{mach}}{2}$. När talen sedan adderas sker en uppskattning av summan på samma sätt och operationen resulterar inte i 0.3.

Moderator: Korrekt!

Solution 4-0-08

Kancellation uppstår då differensen av två tal är nära 0. Uttrycket x^2 - x blir nästan 0 då x är nära 0 eller 1. Ytterligare blir uttrycket (x^2 - x) - 2 nära 0 då x^2 - x är nära 2, alltså då x nästan är 2.

Svar: Kancellation uppstår då runt 0, 1 och 2.

Moderator: Korrekt lösning, rätt tänkt!

Solution 4-0-16

Problem 4 0 16

Volymen av en låda ska bestämmas. Vi mäter lådan till h~=0.5, b~=3, d~=2, så volymen uppskattas till 3 m^3.

a) Hur stor osäkerhet (absolutfel) har vi volymen om vi har mätfel som motsvarar absolutfel 0.01 m.

$$E_f = \left|\frac{\partial f}{\partial x}\right| E_x + \left|\frac{\partial f}{\partial y}\right| E_y + \left|\frac{\partial f}{\partial z}\right| E_z$$

$$E_x = E_y = E_z = 0,01m$$

$$f = x \cdot y \cdot z$$

$$\left|\frac{\partial f}{\partial x}\right| = y \cdot z$$

$$\left|\frac{\partial f}{\partial y}\right| = x \cdot z$$

$$\left|\frac{\partial f}{\partial z}\right| = x \cdot y$$

$$E_f = 0,01(y \cdot z + x \cdot z + x \cdot y)\Big|$$

$$= 0,01(6 + 1 + 1,5)$$

$$= 0,085$$

Svar osäkerheten är $E_f = 0,085$

b) Hur stor osäkerhet (absolutfel) har vi volymen om vi har mätfel som motsvarar relativfel 0.1

(enligt samma beräkningar fast $10x$ större fel)

Svar: Osäkerheten är $E_f = 0,85$

Solution 4-0-19

a. kvadratisk matris, (m x m)

Då löser backslash ett linjärt ekvationssystem m.h.a gausselimination

b. rektangulär matris

Då löser backslash ett minsta kvadrat problem.

Solutions block 1:

Solution 1-1-01

Både Newton-Raphsons metod och sekantmetoden används för att approximera nollställen till funktioner iterativt. De huvudsakliga skillnaderna kan man se när man observerar de olika iterationsformlerna.

Newton-Raphsons:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

Sekantmetoden:

$$x_{n+1} = x_n - \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} f(x_n)$$

Man kan se att i sekantmetoden måste man gissa två startvärden x_0 och x_1 medans man med Newton-Raphsons metod endast behöver gissa på ett startvärde. Ytterligare så måste man derivera funktionen $f(x)$ när man använder Newton-Raphsons metod.

Den största fördelen med att använda sekantmetoden över Newtons-Raphsons metod är att man inte är tvungen att derivera funktionen man vill approximera nollställena till funktionen $f(x)$. Detta kan visa sig mycket fördelaktigt då funktionen $f(x)$ inte alltid är deriverbar.

En av nackdelarna med sekantmetoden är att den är lite svårare att implementera än Newton-Raphsons metod då vi behöver gissa två startvärden istället för ett. Ytterligare en nackdel är att sekantmetoden inte konvergerar lika snabbt som Newton-Raphsons metod. Newton-Raphson har konvergensordning $\alpha = 2$ (kvadratisk) medans sekantmetoden har $\alpha = \frac{1+\sqrt{5}}{2} \approx 1.618$ (vilket även är det gyllene snittet)

Moderator: En korrekt och utförlig lösning. Bra! Tack!

Solution 1-1-04

Newtons metod brukar konvergera snabbare på grund av kvadratisk konvergens. Ett tillfälle man vill hellre använda Sekantmetoden är när den angivna funktionen när derivatan är svårt att beräkna för hand eller kräver mycket beräkningar med dator.

Moderator: Korrekt! (Jag tog bort "ej deriverbar" eftersom det strikt betyder att f'(x) inte existerar och då fungerar i regel varken Newton eller sekant.)

Solution 1-1-09

Genom att bryta ut y i den andra ekvationen får vi $y = 1 - cos(x) - x^2$.

Om vi sedan sätter in detta i den första ekvationen och flyttar över 0.5 får vi
$cos(x) + sin(1 - cos(x) - x^2) - 0.5 = 0$

Genom att plotta vänsterledet ser vi att det finns en lösning mellan x = 0.5 och x = 1. Vi sätter således $x_0 = 0.5$ och $x_1 = 1$ och applicerar sekantmetoden på vänsterledet vilket ger oss roten x = 0.7092. Genom att bakåtsubstituera x får vi
$y = 1 - cos(0.7092) - 0.7092^2 = -0.2619$.

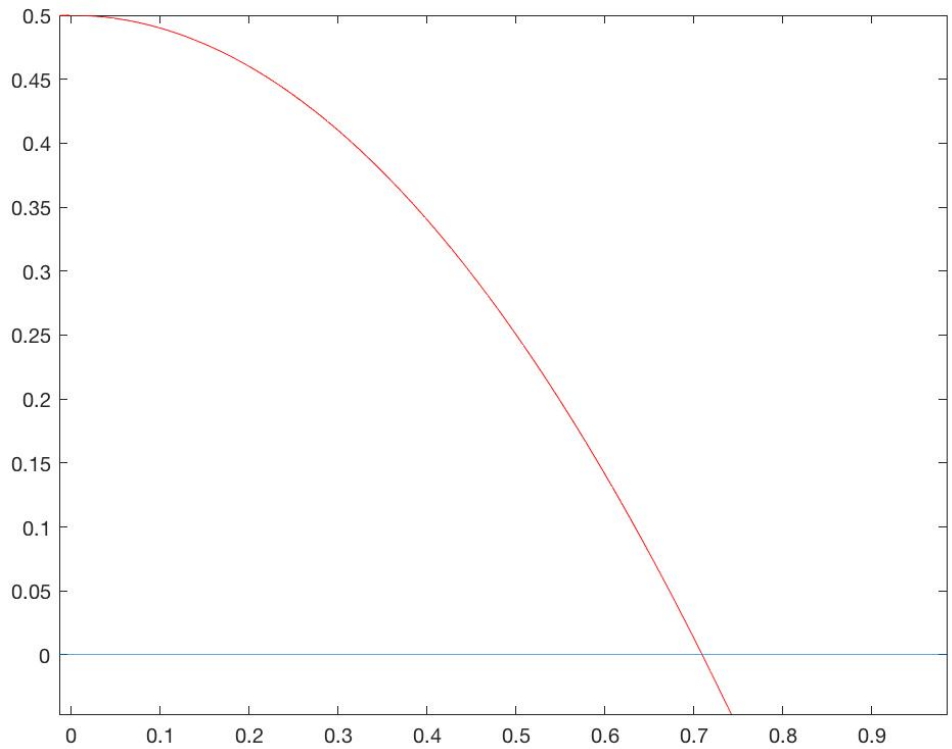
Moderator: Korrekt. Klistra gärna in bilden på figuren du plottade, så blir det tydligare. (Wiki bonus även utan.)

Solution 1-1-11

På liknande sätt som i upp. 1-1-09 elimineras y för att erhålla ekvationen

$\cos(x) + \sin\left(1 - x^2 - \cos(x)\right) - 0.5 = 0$

som plottas för att få fram startvärdena $x_0 = 0.5\ och\ x_1 = 1$



Med hjälp av Newtons metod för flera variabler fås $x = 0.70921$, $y =$ -0.26186

Detta efter 8 iterationer Newtons metod för flera variabler.

Med sekantmetoden erhölls 7 iterationer.

Matlab kod:

```
%Setup
f = @(x) [cos(x(1)) + sin(x(2)) - 0.5;
          cos(x(1)) + x(1).^2 + x(2) - 1];
J = @(x) [-sin(x(1)), cos(x(2));
          -sin(x(1)) + 2*x(1), 1];

h = inf;
TOL = 1e-10;
xv = [0.5;1];
counter = 1;
while(norm(h)>TOL)
   h = J(xv)\f(xv);
   xv = xv - h;
   counter = counter + 1;
end
```

Moderator: Korrekt!

Solution 1-1-18

Vi kan göra taylorutveckling i flera variabler med hjälp av den såkallade Jacobmatrisen, eftersom att det är en matris som generaliserar derivatabegreppet till flera variabler.

För funktioner som går att derivera tillräckligt ofta kan vi utveckla följande uttryck:

1) $f(x+h) = f(x) + J(x)h + O\left(\|h\|^2\right)$

Med hjälp av den ovanstående utvecklingen (1) kan man härleda Newtons metod i flera variabler. Som en början antar vi att $x_0 \epsilon \mathbb{R}$ är givet och är en approximation till lösningen x_x. Då blir korringen: h = x_x - x_0 och vi kan då ersätta h i Taylorutvecklingen ovan med högerledet (x_x - x_0):

2)$0 = f(x_x) = f(x_0 + x_x - x_0) = f(x_0) + J(x_0)(x_x - x_0) + O\left(\|x_x - x_0\|^2\right)$

Vi kan då göra en ny och förbättrad approximation x_1, med hjälp av uttryck (2) , som uppfyller ekvation (2). För det behöver vi bara försumma den kvadratiska termen och ersätta x_x med x_1:

3)$0 = f(x_0) + J(x_0)(x_1 - x_0)$

Eftersom att för att vi tar bort den kvadratiska termen för att definera x_1 kan vi förvänta oss att x_1 är en ganska bra approximation för x_x om det kvadratiska uttrycket ($\|x_x-x_0\|^2$) är litet, eftersom att även x_0 är en approximation av x_x.

Vi antar att jacobmatrisen J(x_0) är inverterbar och kan då skriva om ekvation (3) på följande sätt:

4) $x_1 = x_0 - J(x_0)^{-1} f(x_0)$

Vilket betyder att vi kan beräkna x_1 givet x_0.

Moderator: Korrekt.

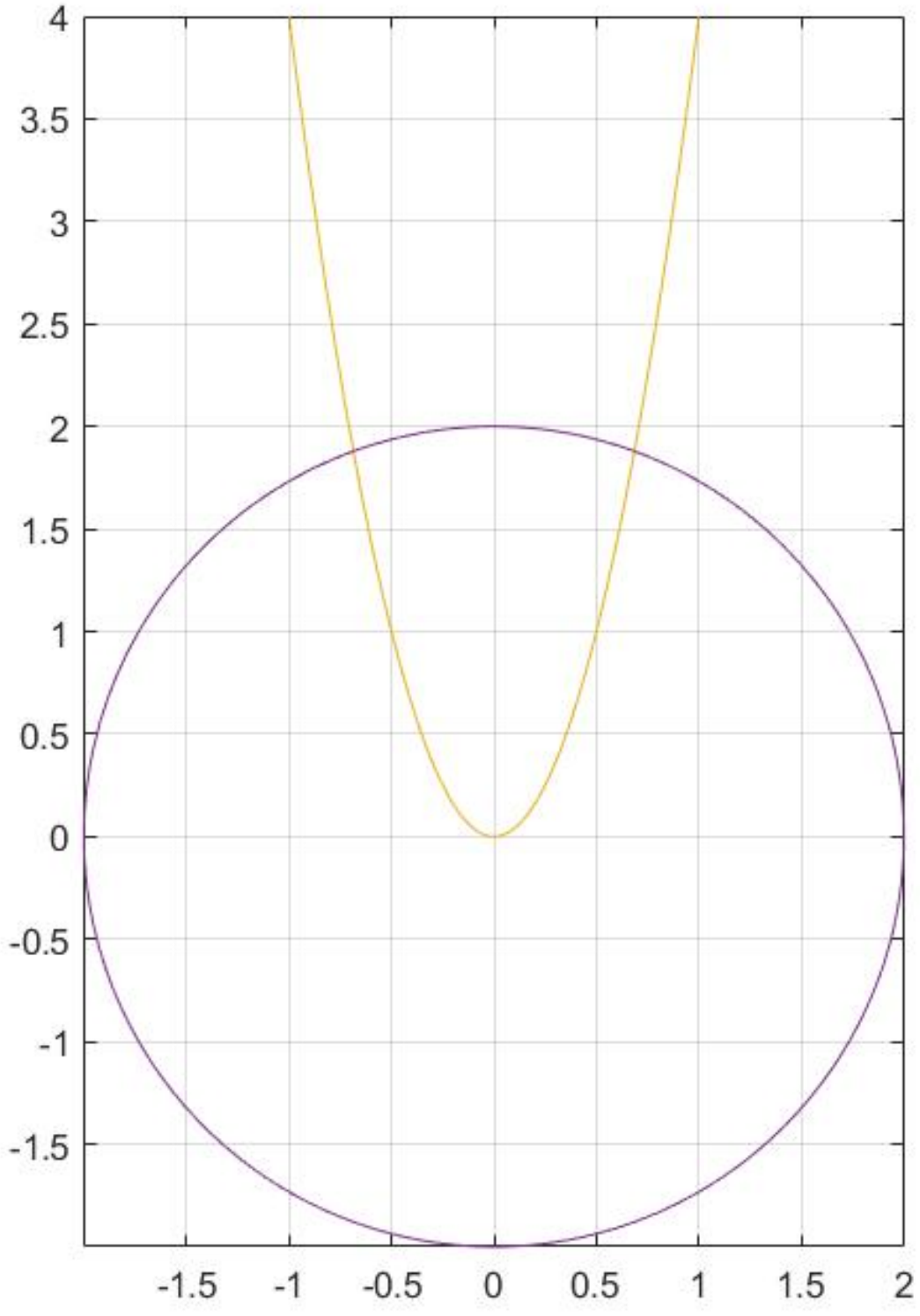
Solution 2-1-02

Efter omskrivning erhålles ekvationerna:

$$\begin{bmatrix} y - 4x^2 \\ x^2 + y^2 - 4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \text{ med tillhörande Jacobian } \begin{bmatrix} -8x & 1 \\ 2x & 2y \end{bmatrix}.$$

Kurvorna skär varandra i första och andra kvadranten. Med gissningen $(x_0, y_0) = (1, 1)$ för roten i den första kvadranten erhålles efter 6 iterationer lösningen $(x, y) = (0.685365311847480, 1.878902442735175)$ med toleransen 10^{-10}.

På grund av kurvornas symmetri erhålles den andra lösningen i punkten $(x, y) = (-0.685365311847480, 1.878902442735175)$.



Moderator: Snygg lösning!

Solution 2-1-04

The following code was used to inspect the problem:

```
% Jakobimatrix

J = @(x) [-sin(x(1)), -2*x(2);

          cos(x(1)), 0];

f = @(x) [cos(x(1)) - x(2)^2; sin(x(1)) - x(1)]; % The two functions


TOL = 1e-10; % Tolerance

xv = [pi;0]; % Start guess

h = inf;

while (h > TOL)

    h = J(xv)\f(xv); % <-- A problem immediately arises on the first iteratior

    xv = xv - h;

end
```

On the third line from the bottom, I have placed a comment that highlights the issue. Namely, the matrix J(xv), which is the Jakobimatrix, is

```
 -0.000000000000000 0
 -1.000000000000000 0
```

whereas the matrix f(xv) is

```
-1.000000000000000
 -3.141592653589793
```

meaning that we are trying to solve the equations

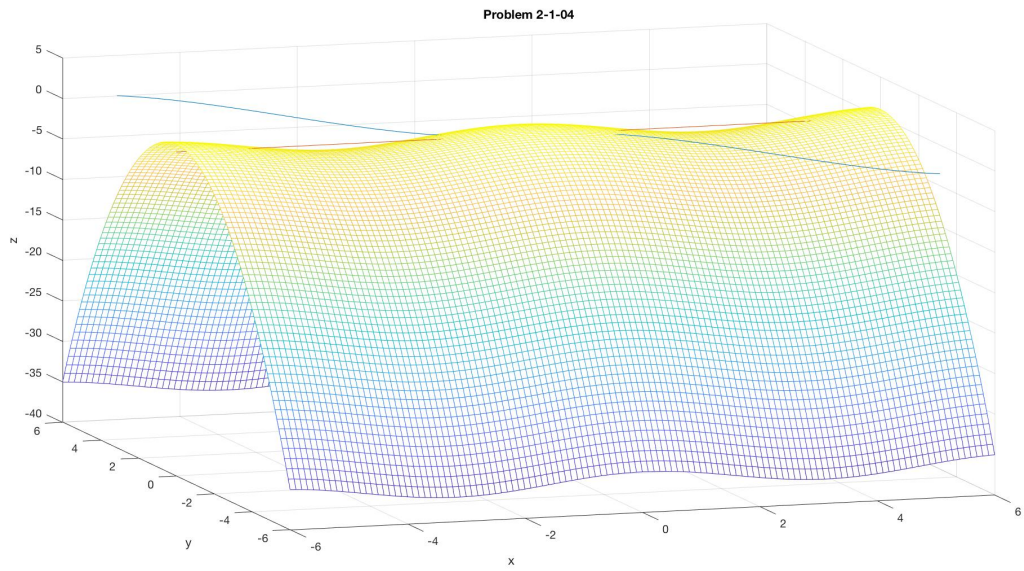
-0.000000000000000*a + 0*b = -1.000000000000000
-1.000000000000000*a + 0*b = -3.141592653589793

where the first equation obviously cannot be solved, since 0 multiplied with any number cannot possibly result in a number other than 0, and thus the system as a whole is also

unsolvable and can neither be approximated (although MatLab will try to solve the system using very large numbers, that are off from the correct solutions to the original equation system, which are (0, 1) and (0, -1)).

<u>As such, the starting guess of (π, 0) is a bad idea since we will end up with an unsolvable linear equation system that will not yield a good approximate solution (if any) if we use that starting guess with Newton's method.</u>

Also, here's a figure that I made:



The most noticeable function is $z = \cos(x) - y^2$, which covers three dimensions. The blue line is the function $y = \sin(x) - x$, and the red line shows the function $y = 0$. This may be confusing since the variables are a bit mixed (they aren't all functions of z), but it still provides an overview of the problem.

Moderator: Nice. Thanks for the extra figure and reasoning!

Solution 2-1-05

Vi vill approximera x då funktionen $f(x) = 0$.

Vi låter $f(x) \approx P(x) = f(a) + f'(a)(x-a) = 0$.

Där $a = x0$ och $x0$ är vår startgissning.

$P(x1) = 0$ ger $f(x0) + f'(x0)(x1 - x0) = 0 \iff x1 = x0 - \frac{f(x0)}{f'(x0)}$ där $f(x1) \approx 0$.

Detta är iterations-formeln som upprepas i Newtons metod som (oftast) ger ett bättre värde på $x1$ efter varje iteration.

Moderator: Korrekt. Jag skulle ersätta *x0* med x_0. Kom ihåg: I programmering kan en variabel heta x0, men i matematik betyder x0 = x*0=0. Nu är CANVAS-editorn inte perfekt.

Solution 2-1-09

Kör följande Matlab-kod:

```
clear
% ekvationssystemet
f=@(x) [x(1)^3-6*x(2) + 3;x(1) + x(2)^5];
% Jacobimatrisen
J=@(x) [3*x(1) , -6;1, 5*x(2)^4];

h=inf;
TOL=1e-15;
x=[0;0];
while (norm(h)>TOL)
 h=J(x)\f(x);
 x=x-h;
end
```

Då får vi att (x, y) = (-0.0312, 0.5000) är den enda reella lösningen till ekvationssystemet.

Solution 2-1-16

För att hitta roten till $f(x) = x^3 + 2x - 2$ används Sekantmetoden med startgissningarna $x_0 = 0 \; och \; x_1 = 1$.

Sekantmetoden: $x_{n+1} = x_n - \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} f(x_n)$

Detta ger:

$x_2 = 1 - \frac{1-0}{1+2} = \frac{2}{3}$

Kör jag sekantmetoden i MATLAB får jag fram följande resultat:

n	x_n
0	0.000000000000000
1	1.000000000000000
2	0.666666666666667
3	0.756756756756757
4	0.771836828762388
5	0.770909015402272
6	0.770916997059248

Roten till funktionen är alltså i punkten (0.770916997059248, 0)

Moderator: Ok.

Solution 2-1-18

```
% sekantmetoden för att hitta skärningen mellan två funktioner
% wiki uppgift 2-1-18

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Våra värden
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% FUNKTIONEN
f=@(x) exp(cos(x/4)) + 5/(exp(x^2)) - (x^5)/100 - (5*x^2)/7;
%Vi sätter funktionerna kring ett likamedtecken och flyttar över en
%funktion till den andra sidan så att de tillsammans blir likamed 0


% TVÅ INITIALA GISSNINGSPUNKTER

x1 = 4;
x2 = 5;

% TOLERANS

tol = 1e-6;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Implementation av sekantmetoden
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Funktionsvärdet vid x1

f1 = f(x1);

% WHILE LOOP

%dx säger hur mycket x har förändrats
dx = inf;
%detta ser till att vår loop alltid körs den där första gången oavsett
%vilken tolerans vi har

iter = 0;

while abs(dx) > tol


%räkna iterationer
iter = iter + 1;
```

```
%Evaluera funktionen i x2
f2 = f(x2);

%räkna ut dx för att ser hur långt ut vår nya x-gissning är
dx = (x2-x1)*f2/(f2-f1);

%Vår "andra" punkt blir nu den första
x1 = x2;
f1 = f2;

%Uppdatera positionen av den andra punkten
x2 = x2 - dx;

end
```

RESULTAT

Roten fås (med bra startvärden) genom att köra programmet och sedan skriva in x2 eller x1 i kommandofönstret

Startvärde x1	Startvärde x2	Rot
4	5	1.8376
-2	0	-1.9851
-4	-5	-3.9156

Moderator: Okej.

Solution 3-1-05

```
x(1)=-1;
x(2)=-1.1;
for i=2:8
   x(i+1)=x(i)-f(x(i))*(x(i)-x(i-1))/(f(x(i))-f(x(i-1)));
end
```

Moderator: Korrekt svar. Kom ihåg att det är bättre göra while-slingor med termineringsvillkor abs(h)>TOL, än for-slingor. (Nu var uppgiften formulerad med for-slinga, så helt rätt.)

Solution 3-1-06

Vi har följande ekvationer:

$$f_1(x,y) = \left(\frac{x}{5}\right)^2 + \left(\frac{y}{10}\right)^2 - 1 \;=\; 0$$

$$f_2(x,y) = y - \frac{2}{3}x + \sin(2x) - 5 = 0$$

Som ger följande ekvationssystem

$$\mathbf{f(x)} = (f_1(x,y),\; f_2(x,y))^T = \mathbf{0}$$

Vi söker efter punkterna $\mathbf{x} = (x,y)$ som uppfyller det här ekvationssystemet.

Vi hittar Jacobianen:

$$J\left(\mathbf{x}\right) = \begin{bmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} \end{bmatrix} = \begin{bmatrix} \frac{2x}{25} & \frac{y}{50} \\ -\frac{2}{3} + 2\cos 2x & 1 \end{bmatrix}$$

Därefter kan vi använda Newtons formeln för att hitta lösningarna:

$$\mathbf{x}_n = \mathbf{x}_{n-1} - J(\mathbf{x}_{n-1})^{-1}\mathbf{f}(\mathbf{x}_{n-1})$$

Vi använder MATLAB för att lösa systemet. En snabb plot visar att skärningspunkterna ligger ungefär vid $(-4,1)$ och $(4,7)$, så dessa punkter blir våra startgissningar.

Genom att använda MATLAB-koden nedan får vi lösningarna $\mathbf{x}_1 = (-4.96, 1.21)$ och $\mathbf{x}_2 = (3.77, 6.56)$. Jag antar att författaren menar att öronen ligger vid skärningspunkterna mellan $y = 2$ och ellipsen. I sådana fall är det endast ena örat som täcks, eftersom dess y-värde är mindre än 2.

```
f=@(x) [(x(1)/5)^2+(x(2)/10)^2-1; x(2)-(2/3)*x(1)+sin(2*x(1))-5];
J=@(x) [2*x(1)/25 , x(2)/50; (-2/3)+2*cos(2*x(1)), 1];

TOL=1e-15; % Toleransen
h=inf;
x=[7;4]; % Startgissning
while (norm(h)>TOL)
    h=J(x)\f(x);
    x=x-h;
    norm(h);
end
```

Moderator: Korrekt och kul

Solution 3-1-07

Enligt hänvisningar skall Ramanujans approximation användas. Dock används följande variant av approximation av omkretsen:

$$O \approx \pi \left[3(a+b) - \sqrt{10ab + 3(a^2+b^2)}\right]$$

Denna variant är inte *den* approximation som hänvisas till i uppgiften. Detta är dock formel (49) ur ”Collected Papers of Srinivasa Ramanujan”, kap. 6, pg. 39 (följande länk är den samma som på den hänvisade wikipedia-sidan https://books.google.com/books?id=oSioAM4wORMC&pg=PA39). De två formlerna har olika felgränser --- något som är att ha i åtanke vid beräkning.

Anledningen till användandet av denna formel är då formeln ovan är att den är betydligt enklare att derivera med avseende på b.

$$\frac{\mathrm{d}}{\mathrm{d}b} O = \pi \left[3 - \frac{10a+6b}{2\sqrt{10ab+3(a^2+b^2)}}\right]$$

För att använda Newtons metod gör vi följande:

$$O - 600 \approx \pi \left[3(a+b) - \sqrt{10ab + 3(a^2+b^2)}\right] - 600$$

Och letar efter rötter. I Matlab itererar vi tills vi får ett fel mindre än e-10. Matlabkoden är straight forward (se till exempel föreläsningsfilerna), och får att:

$$b \to 90.876904966299691$$

Vilket säger att vi ungefär behöver en 181,7538 m bred plan. ■

Moderator: Utmärkt! Det är viktigt med källkontroll. Bra!

Solution 3-1-11

a) Detta är ett egenvärdesproblem eftersom om vi definierar $\lambda = x_4$ och låter

$$\mathbf{z} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

så betyder ekvationen som innehåller matrisen A att

$$A\mathbf{z} = \lambda z$$

som är definitionen av egenvärden. Det andra villkoret betyder att

$$\|z\|^2 = x_1^2 + x_2^2 + x_3^2 = 1$$

dvs z är egenvektorn som är normaliserad

b) För att genomföra Newtons metod i flera variabler behöver vi Jacobianen som blir (efter lite räkningar)

$$J(\mathbf{x}) = \begin{bmatrix} A - \lambda I & -z \\ 2z^T & 0 \end{bmatrix}$$

Om vi nu genomför Newtons metod i flera variabler får vi:

```
>> J=@(x) [A-x(4)*eye(3), -x(1:3); x(1:3)',0];
>> f=@(x) [A*x(1:3)-x(4)*x(1:3);x(1)^2+x(2)^2+x(3)^2-1];
>> h=inf; TOL=1e-10;
>> x=[1;1;1;100];
>> while (norm(h)>TOL);
      h=J(x)\f(x);
   x=x-h;
end
>> norm(f(x))
ans =
    0
>> z=x(1:3); lambda=x(4);
>> A*z-lambda*z
ans =
    0
    0
    0
>> lambda
lambda =
  99.9489
```

```
>> eig(A)
ans =
   1.0378
   5.0133
  99.9489
```

Funktionen eig som returnerar matrisens egenvärden stämmer överrens med vår metod.

Solution 3-1-16

Moderator: Lösningen som var här är korrekt (och räknas till wiki-bonus). Jag tog bort den eftersom den låg för nära labbuppgifterna.

Solution 3-1-22

Vilket ypperligt tillfälle att pröva min tanke! Jag vill veta om följande resonemang är korrekta (jag frågade när jag skulle redovisa en av labbarna, men vi hade inte tid att gå igenom det då):

Fall 1:

Newton-Rahpson fungerar inte när vi har en punkt, x, som ger en avtagande derivata på ett intervall $[a,\ \infty]$. Där a är vald så att $\lim_{a\to\infty} f(x) = \infty$. Om punkten istället hade gett en ökande derivata på intervallet, så skulle nästkommande varv vara närmare roten --- och Newton skulle *fungera*. Motsatsen misslyckas också; om punkten ger en ökande derivata på ett intervall som funktionen är ”tillslut-avtagande” (en: ultimately decreasing).

Fall 2:

Newton-Rahpson misslyckas även om vi gissar en kritisk punkt ($x_0 \implies f'(x_0) = 0$). Eftersom tangenten i denna punkt kommer vara parallel med x-axeln (och/eller att NR i sådana fall leder till en division med noll).

Solution 4-1-01

Vi använder matlab för att iterera fram roten:

Editor - /afs/kth.se/home/a/m/amiksits/Dropbox/Nummen SF1547/LabA1/wiki4-1-01.m

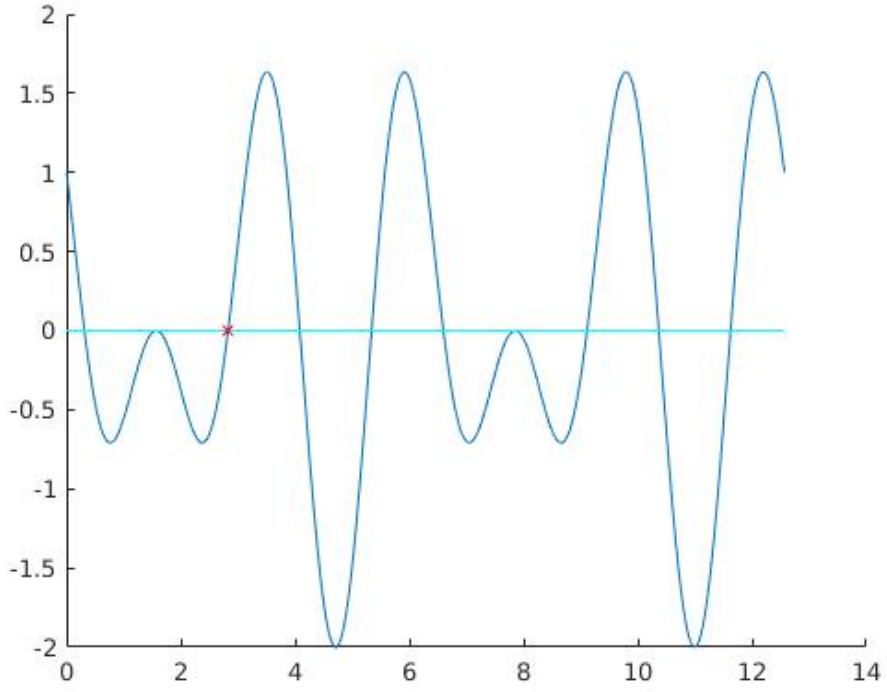
LabA1.m | wiki4-1-01.m

```
% Newton-metoden för att hitta cos(2x) + sin(3x) = pi

clc
clear
close all

y = @(x) cos(2*x) - sin(3*x);
xval = linspace(0,4*pi,1000);
g = zeros(1, length(xval));

hold on
plot(xval, y(xval))
plot(xval, g, 'c')

yprim = @(x) -2*sin(2*x) - 3*cos(3*x);
n_iterations = 20;
x = zeros(1,n_iterations);


x(1) = pi; %Sök rot närmast pi --> börja på pi

for k=2:n_iterations
    x(k) = x(k-1) - y(x(k-1))/yprim(x(k-1));
end


display(['Min rot blir x = ', num2str(x(n_iterations)/pi), '*pi och funktionens värde blir då

plot(x(n_iterations), y(x(n_iterations)), 'xr')
```

Command Window

```
Min rot blir x = 0.9*pi och funktionens värde blir då y = -2.2204e-16 efter 20 iterationer
>>
```

och får som framgår ur texten roten x = 0.9pi. I grafen ser det ut som nedan:



Funktionsvärdet är väldigt nära noll enligt utskrift i matlab, varför det verkar vara en bra approximation av roten.

Moderator: Korrekt. I första föreläsningen om Newtons metod använde vi också for-loop. While-loop med tolerans som termineringsvillkor är att föredra, för det ger mer automatik. Använd hellre while-loop i framtiden / labbarna.

Solution 4-1-04

Vi kan se att $10x - 1 = 0 \iff x = \frac{1}{10}$, så söker antalet steg i Newtons metod som ger $\frac{1}{10}$.

Sätter $f(x) = 10x - 1$ vilket ger $f'(x) = 10$

Startgissningen är $x_0 = 0$, därav är tangentens funktion i x_0 enligt enpunksformeln

$q(x) = f(x_0) + f'(x_0)(x - x_0)$

$f(x_0) = f(0) = -1$

$f'(x_0) = f'(0) = 10$

Söker ett t så att $q(t) = 0$, dvs

$$f(x_0) + f'(x_0)(t - x_0) = 0 \iff -f(x_0) = f'(x_0)(t - x_0) \iff -\frac{f(x_0)}{f'(x_0)} = t - x_0$$

Alltså hittas rätt rot till $10x - 1 = 0$ redan efter 1 iteration

Eftersom derivatan är konstan, dvs att den inte beror på x, kommer valet av startvärde inte spela någon roll. Alla tangentlinjer för alla x-värden kommer ha funktionen $q(x)$ vilket ger ett m värde (i funktionen för en rät linje $y = kx + m$) som -1, och då $f'(x) = 10 \forall x$ kommer $t = \frac{1}{10}$ efter första iterationen oberoende av vad valet av startvärde är.

Moderator: Korrekt. Utmärkt!

Solution 4-1-06

Figuren längst till höger motsvarar kvadratisk konvergens där x-axeln är antalet iterationer och y-axeln är felet. Detta kan motiveras genom att observera y-värdet för varje iteration (det ökar kvadratiskt, exempelvis $f(5) = 10^{-5}$, $f(6) = 10^{-10}$)

Moderator: Korrekt!

Solution 4-1-08

$x =$ -3.894523972300119 erhölls med följande MATLAB-kod:

```
f = @(x) 2.*x.*cos(x) + 3.*x + 6;
fprim = @(x) 2.*cos(x) - 2.*x.*sin(x) + 3;

TOL = 1e-16;
h = inf;
x = -5;

while (abs(h) > TOL)
 h = f(x)/fprim(x);
 x = x - h;
end

x
```

Moderator: lösningen är korrekt! Visa gärna med en graf att lösningen är rimlig.

Solution 4-1-10

• h har inte in introducerats vilket gör att loopen inte kommer fungera.

• I while-loopen ska villkoret vara $|h| > TOL$

• x har inte introducerats, så vi har ingen startgissning att ta funktionsvärden av eller att korrigera.

Moderator: Bra. Det finns visst ett tredje fel. Uppdatera gärna lösningen. - **Redigerad**

Solution 4-1-11

Om man väljer startgissningen $x_0 = 1$ så blir derivatan $f'(x_0) = f'(1) = 6(1-1) = 0$. Newtons metod ges av $x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$. Men om derivatan är 0 så är Newtons metod odefinierad ty division med 0. Därav kan inte Newtons metod användas tillsammans med startvärdet $x_0 = 1$.

Moderator: Korrekt! Skulle du kunna visa vad som sker med en bild?

Solution 4-1-13

När vi använder oss av Newtons metod har vi formeln $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$ för att beräkna våra x-värden när vi itererar.

I vårt fall har vi att $f(x) = \ln(x)$ och $f'(x) = \frac{1}{x}$, vilket ger oss:

$$x_{n+1} = x_n - \frac{\ln(x_n)}{\frac{1}{x_n}} = x_n - x_n \cdot \ln(x_n) = x_n \cdot (1 - \ln(x_n))$$

Eftersom $f(e) = 1$ och eftersom $f(x)$ är en växande funktion kommer $f(x_0) \geq 1$ för $x_0 \geq e$.

$$\ln(x_0) \geq 1 \implies 1 - \ln(x_0) \leq 0 \implies x_1 = x_0 \cdot (1 - \ln x_0) \leq x_0 \cdot 0 = 0$$

För att räkna ut nästa x-värde i iterationen kommer vi behöva värdet av $f(x_1)$, men då $x_1 \leq 0$ och $\ln(x)$ bara är definierade för $x > 0$, kommer vi inte kunna räkna ut x_2. Därför kan vi inte nå ett nollställe om vi sätter $x_0 \geq e$.

Moderator: Utmärkt lösning och bra förklarat!

Solution 4-1-19

Lösningarna är x = 2.9814; y = 0.5279 och x = -2.9814, y = 9.4721. Det hittades genom att skriva om funktionerna och använda Newtons metod för flervariabel och sedan testa olika startvärden.

```
f=@(x) [3*x(1)*x(2) + x(2)^2 - 5; 2*x(2) + 3*x(1) - 10];
J=@(x) [3*x(2), 3*x(1) + 2*x(2); 3, 2];
```

Moderator: Du visar förståelse, även om svaret inte är fullständigt. Räknas till wiki-bonus. Lägg lite mer energi på att göra svaret fullständigt nästa gång, tack, då har andra studenter mer nytta av den.

Solutions block 2:

Solution 1-2-02

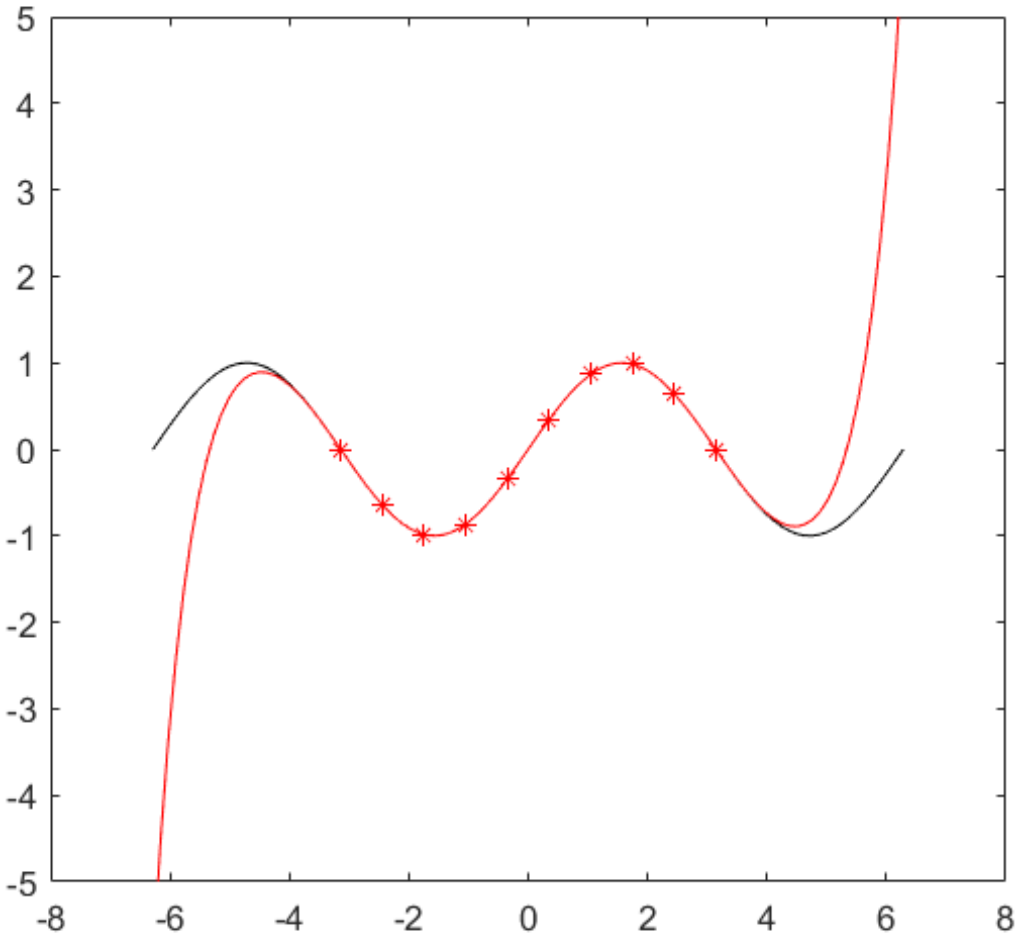
Då n = 3 krävs fyra punkter: -pi, -pi/3, pi/3 och pi. Vandermondematrisen (med formatet $1 \quad x_i \quad x_i^2 \quad x_i^3$) blir då

$$\begin{bmatrix} 1 & -\pi & \pi^2 & -\pi^3 \\ 1 & -\frac{\pi}{3} & \left(-\frac{\pi}{3}\right)^2 & \left(-\frac{\pi}{3}\right)^3 \\ 1 & \frac{\pi}{3} & \left(\frac{\pi}{3}\right)^2 & \left(\frac{\pi}{3}\right)^3 \\ 1 & \pi & \pi^2 & \pi^3 \end{bmatrix}$$

Kalla denna matris A. Vi vill finna koefficienterna $a_0,\ a_1,\ a_2,\ a_3$. En vektor bestående av dessa fyra okända kallar vi c. Högerledet f i det linjära ekvationssystemet A c = f är sin(x) för de fyra punkterna.

För detta och högre ordningens polynom kan man beräkna interpolationspolynomet med följande matlabprogram, där brytpunkterna gör att vi kan stanna och titta på resultatet för olika polynomgrad innan nästa beräkning görs:

```
%%
%Interpolationsexempel:
%Approximera sin(x)
figure(1)
t = linspace(-2*pi,2*pi);

for k = 3:11 %Approximerar sin(x) med högre ordningens polynom,
    %från grad 3 till 11.
    plot(t,sin(t),'k');
    hold on
    ylim([-5,5])
    x = linspace(-pi,pi,k+1); %Tar fram k+1 jämnt fördelade punkter i
    % intervallet.
    plot(x,sin(x),'b+');
    A =ones(k+1,1);
    for l = 1:k
        A = [A x'.^l]; %Skapar vandermondematrisen
    end
    f = sin(x)'; %Skapar högerledet till ekvationssystemet
    c = A\f; %Löser systemet för att finna polynomets koefficienter
    c1 = c(end:-1:1); %Vänder på vektorn för att enklare kunna plotta
    %polynomet med hjälp av polyval.
    plot(x,polyval(c1,x),'r*')
    plot(t,polyval(c1,t),'r')
    hold off

end
```



Bilden illustrerar approximationen av grad 9.

Moderator: korrekt!

Solution 1-2-04

Med Newton-ansatsen fås den triangulära matrisen A:

```
A =

     1     0     0
     1     1     0
     1     2     2
```

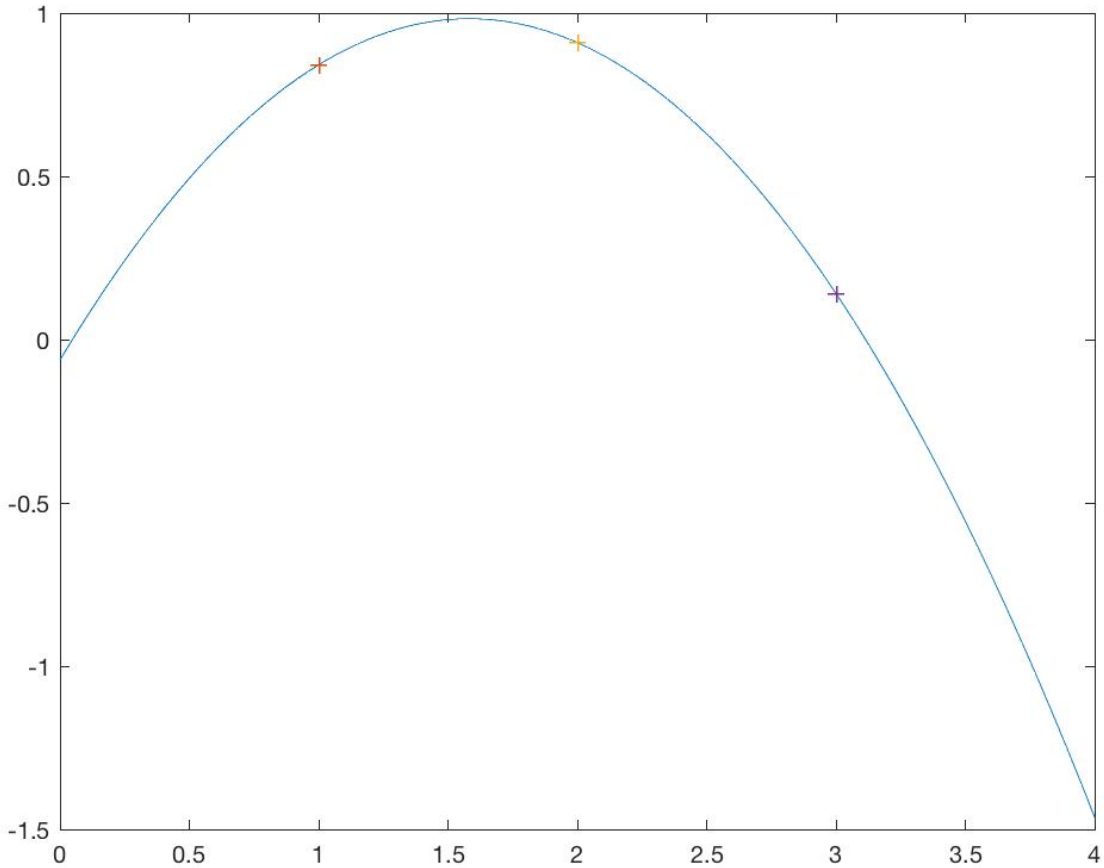
på detta sätt löses konstanterna $d_0,\ d_1\ och\ d_2$

med hjälp av Matlab:

```
%  (1 , 0.8415) , (2 , 0.9093) och  (3 , 0.1411)

%  x0 = 1;
%  x1 = 2;
%  x2 = 3;
% P = d0 + d1(x-x0) + d2(x-x0)(x-x1)
hold off
xv=[1,2,3]';
xxv = 0:0.01:4;
n = length(xv);
A = [ones(n,1), xv-xv(1), (xv-xv(1)).*(xv-xv(2))];
b = [0.8415, 0.9093, 0.1411]';
d = A\b;
pv = d(1) + d(2)*(xxv-xv(1)) + d(3)*(xxv-xv(1)).*(xxv-xv(2));
plot(xxv,pv)
```

och ger funktionen $p\left(x\right)\ =\ 0.8415\ +\ 0.0678\left(x-x_0\right)\ -\ 0.4180\left(x-x_0\right)\left(x-x_1\right)$

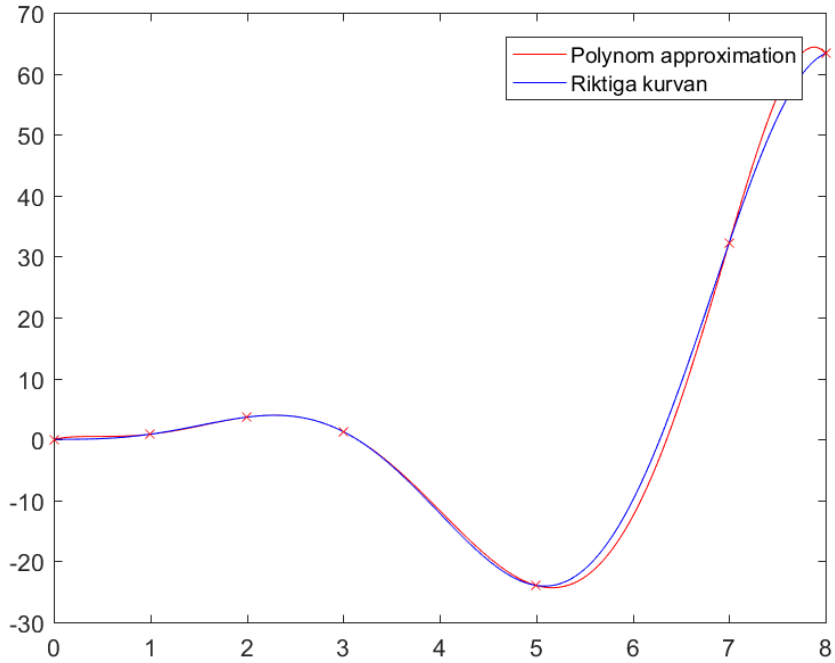
Plottar de givna punkterna ihop med funktionen som fås och ser att den går igenom de givna punkterna.



Moderator: Korrekt.

Solution 1-2-05

Grafen visar kurvanpassningen med Vandermondematrisen och Newton ansatsen. Trots ett högt gradtal så uppstod inte Runges fenomen. Oscillationer uppstod vid slutet av intervallet när gradantalet ökade till 10. Den bifogade koden, visar hur detta implementerades. Det finns möjlighet att utveckla hur Newton matrisen kan skapas men har ännu inte hittat en bra algoritm för detta.




```
x=[0 1 2 3 5 7 8]';
n=6; %degree for polynomial
for k=1:n+1; %vandermonde matrix
    A(:,k)=x.^(k-1);
end
f=x.^2.*sin(x);
c=A\f; %solving linear eq. sys.
p=0;
X=x(1):0.01:x(7);
for i=n+1:-1:1
   p=p.*X+c(i); %Horners alg.
end
%plot(X,p),
%P=c(1)+c(2)*X+c(3)*X.^2+c(4)*X.^3+c(5)*X.^4+c(6)*X.^5+c(7)*X.^6;
plot(X,p,'r'),hold on
plot(x,f,'rx')
%newton
col0=ones(7,1);col1=(x-x(1));col2=(x-x(2)).*(x-x(1));
col3=(x-x(3)).*(x-x(2)).*(x-x(1));col4=(x-x(4)).*(x-x(3)).*(x-x(2)).*(x-x(1));
col5=(x-x(5)).*col4; col6=(x-x(6)).*col5;
A=[col0 col1 col2 col3 col4 col5 col6]
d=A\f;
X=x(1):0.01:x(7);
P=d(1)+d(2).*(X-x(1))+d(3).*(X-x(2)).*(X-x(1))+...
     d(4).*(X-x(3)).*(X-x(2)).*(X-x(1))+...
     d(5).*(X-x(4)).*(X-x(3)).*(X-x(2)).*(X-x(1))+...
     d(6).*(X-x(5)).*(X-x(4)).*(X-x(3)).*(X-x(2)).*(X-x(1))+...
     d(7).*(X-x(6)).*(X-x(5)).*(X-x(4)).*(X-x(3)).*(X-x(2)).*(X-x(1));
%size(A);
plot(X,P,'b--')
hold off
```

Moderator: Okej. Korrekt implementation. Runges fenomen skulle troligtvis uppstå vid (ännu) högre gradtal.

Solution 1-2-06

Runges fenomen uppstår när man försöker interpolera ett polynom med högt gradtal givet ett antal mätpunkter i planet. Det uppstår då stora svängningar i kurvan mellan interpolationspunkterna. Ett sätt att minska fenomenet är att istället för jämnt spridda punkter använda sig av nollställen till ortogonala polynom som interpolationspunkter.

Moderator: Bra. Att använda sig av nollställen till ortogonala polynom, t.ex. Chebyshevpolynom, ingår (tyvärr?) inte i kursen. En annan lösning är att dela upp intervallet i delinterval och göra styckvis interpolation med lägre polynomgrad.

Solution 1-2-09

För stora n, dvs ett polynom av högt gradtal så är Newton ansatsen bättre att använda då det ger mycket mindre avrundningsfel än vandermondetekniken. Grafiskt syns detta tydligt då Newton metoden skulle ge en fin kurva och vandermonde skulle också följa denna kurva men se väldigt hackig ut.

Moderator: Korrekt. Den andra viktiga anledningen är att det går snabbare att lösa ett triangulärt system.

Solution 1-2-10

Vi söker värden på variablerna $a, b \in \mathbb{R}$ så att $y = a + bx$ bildar en rät linje anpassad efter våra datapunkter. Med vår mätdata får vi ekvationerna:

$$\begin{cases} a + 1b = 1 \\ a + 2b = 3 \\ a + 3b = 4 \\ a + 4b = 9 \\ a + 5b = 13 \end{cases}$$

Detta kan vi skriva som $A\vec{c} = \vec{y}$, vilket blir:

$$\begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \\ 1 & 5 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \\ 4 \\ 9 \\ 13 \end{bmatrix}$$

Minstakvadratmetoden, som går ut på att multiplicera båda leden med $A^\top$ (A' i *Matlab*), ger oss ekvationen $A^\top A\vec{c} = A^\top \vec{y}$, som vi kan lösa med \-operatorn i *Matlab*: cv=(A'*A)\(A'*yv). Detta ger oss $\vec{c} = \begin{bmatrix} -3 \\ 3 \end{bmatrix}$, dvs. $a = -3$ och $b = 3$, så ekvationen för vår linje blir $y = 3x - 3$.

Residualvektorn, dvs. felet, erhålls genom att räkna ut $A\vec{c} - \vec{y}$, vilket blir $\begin{bmatrix} -1 \\ 0 \\ 2 \\ 0 \\ -1 \end{bmatrix}$.

Moderator: Korrekt!

Solution 1-2-12

a) Gausseliminering för att det blir ett linjärt ekvationsystem med lika många ekvationer som obekanta, dvs backslash

b) Flera ekvationer än obektanta + linjärt beroende på de okända variablerna $\Rightarrow$ Backslash

c) Samma som b

Moderator: Ok.

Solution 1-2-13

Ställ upp en ekvation för varje datapunkt:

$$y_i = a + bt_i + t_i^2, \; i = 1, 2, 3, 4.$$

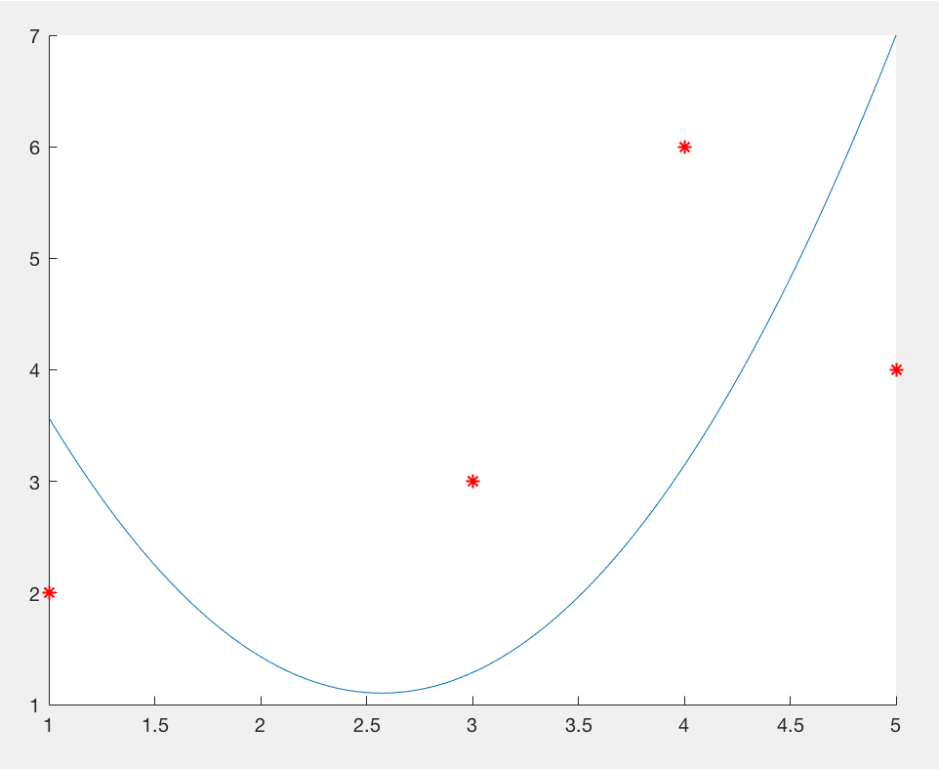
$$\begin{aligned} 2 &= a + b \cdot 1 + 1^2, \\ 3 &= a + b \cdot 3 + 3^2, \\ 6 &= a + b \cdot 4 + 4^2, \\ 4 &= a + b \cdot 5 + 5^2, \end{aligned}$$

Flytta över (den kända) t^2-termen i vänsterledet, detta ger systemet

$$\begin{bmatrix} 1 & 1 \\ 1 & 3 \\ 1 & 4 \\ 1 & 5 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} 2 - 1^2 \\ 3 - 3^2 \\ 6 - 4^2 \\ 4 - 5^2 \end{bmatrix} = \begin{bmatrix} 1 \\ -6 \\ -10 \\ -21 \end{bmatrix}.$$

Matlabkod:

```
% definiera data som kolumnvektorer
t = [1 3 4 5]';
y = [2 3 6 4]';

% Definiera överbestämt ekvationssystem A*x = b
% motsvarande ekvationerna y_i = a + b*t_i + t_i^2.
A = [ones(4,1) t];
b = y - t.^2;

% Minstakvadratlösning
x = A\b;

% koefficienterna a,b är komponenter i lösningsvektorn x
a = x(1);
b = x(2);

% evaluera polynomet p(t) = a + bt + t^2
% med en finfördelad tidsvektor tvec
tvec = linspace(t(1),t(end));
p = x(1) + x(2)*tvec + tvec.^2;

% plotta datapunkter och polynom
figure
hold on
plot(t,y,'r*')
plot(tvec,p)
```



Polynomet i blått och datapunkterna i rött.

Moderator: Korrekt (jag löste den).

Solution 2-2-02

a1 =4.7794e-13

a2 =6.2062e-12

b1 =9.8339

b2 =5.4435e-15

I program (a) har matrisen A (100,100) kolumner är lika mycket som rader så det spelar inte så stort roll om vi använder backslash eller minstakvadratmetod.

Men i program (b) har matrisen A(100,5) flera rader än kolumner så har blir ett överbestämt linjärt ekvationssystem o då använder man minstakvadratmetod p.g.a. det inte finns exakt en lösning och vi kan hitta att närmaste lösning är b2 som använder minstakvadratmetoden. OBS!!Backslash operatör kommer att utföra minstkvadratsmetod för att hitta närmaste värde.

Moderator: Korrekt! Du har dock inte svarat på vad backslashoperatorn gör i respektive fall. Komplettera gärna med att tydligare redogöra för detta.

Solution 2-2-12

a) Newton's metod (lika många ekvationer som obekanta)

b) Gauss-Newton (olinjärt beroende på en obekant konstant och fler ekvationer än obekanta)

c) Samma som b

Moderator: Ok.

Solution 3-2-01

a) Programmet anpassar andragradskurvor efter en $cos(x)$ kurva. Den första delen av programmet anpassar en kurva så att den går igenom 3 punkter på kurvan. I mitten ligger lite kod som ritar kurvor. Den sista delen av programmet anpassar en kurva så att den går (nästan) igenom 4 punkter på kurvan och ritar kurvan.

b) Att anpassa en polynomisk kurva efter punkter är ekvivalent med att lösa ett ekvationssystem. Anta att vi har punkter $(x_1,\ y_1)\ ,\ (x_2,\ y_2)\ ,\ \ldots\ (x_n,\ y_n)$ som ska anpassas efter kurvan $p\,(x) = c_0 + c_1 x + c_2 x^2$. Vi kan skriva detta så här:

$$\begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ 1 & x_3 & x_3^2 \\ \vdots & & \\ 1 & x_n & x_n^2 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix}$$

Vi ser att om $n = 3$ d.v.s vi har 3 punkter att anpassa efter, så blir matrisen till vänster (vandermondematris) kvadratisk d.v.s. den är lika hög som den är bred. Om $n > 3$ d.v.s. vi har fler än 3 punkter, så blir matrisen rektangulär. Detta innebär att det inte finns exakta lösningar c_0, c_1, c_2 så att systemet stämmer. Vi kallar ett ekvationssystem för mer ekvationer än obestämda för "överbestämt". Vi kan försöka hitta lösningarna genom att använda minstakvadratmetoden, men de kommer inte stämma exakt (om vi inte har riktigt tur, men det går vi inte in på nu).

Eftersom den röda kurvan använder 3 punkter, finns det en exakt lösning för koefficienterna. Den gula kurvan använder 4 punkter, vi kan inte garantera att kurvan skär $cos(x)$ kurvan i alla punkter.

c) Skillnaden är att första gången vi använder den, så löser den det linjära ekvationssystemet exakt (ungefär som att skriva $c = inv(A) \cdot b$), medans andra gången vi använder den så beräknar det en minstakvadratlösning till det överbestämda ekvationssystemet, som alltså är en ungefärlig lösning till ekvationssystemet och kurvan går alltså bara nästan genom de fyra punkterna.

Moderator: Bra!

Solution 3-2-03

Vi börjar med att omformulera problemet genom att anta ett polynom:

$$P(x) = a + bx + cx^2 + dx^3$$
$$P'(x) = 0 + b + 2cx + 3dx^2$$

Med de angivna punkterna: $\vec{x} = \begin{bmatrix} 0 & \frac{\pi}{2} \end{bmatrix}^T$. Får vi följande system:

$$\begin{cases} P(0) = a + b0 + c0 + d0 & = 1 \\ P\left(\frac{\pi}{2}\right) = a + b\frac{\pi}{2} + c\frac{\pi^2}{4} + d\frac{\pi^3}{8} & = 0 \\ P'(0) = 0 + b + c0 + d0 & = 0 \\ P'\left(\frac{\pi}{2}\right) = 0 + b + c\pi + d\frac{3\pi^2}{4} & = -1 \end{cases}$$

Och vi får följande Matrisekvation:

$$A\vec{x} = \vec{b} \implies \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & \frac{\pi}{2} & \frac{\pi^2}{4} & \frac{\pi^3}{8} \\ 0 & 1 & 0 & 0 \\ 0 & 1 & \pi & \frac{3\pi^2}{4} \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ -1 \end{bmatrix}$$

Vi löser systemet med valfri metod (t.ex. med inversen och Gauss, eller backslash i MatLab). Vi får följande exakta värden på konstanterna:

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ \frac{2\pi-12}{\pi^2} \\ \frac{16-4\pi}{\pi^3} \end{bmatrix}$$

Vilket ger det interpolerande polynomet:

$$P(x) = 1 + 0 + x^2\left(\frac{2\pi-12}{\pi^2}\right) + x^3\left(\frac{16-4\pi}{\pi^3}\right)$$

Moderator: Bra! Du har ett litet skrivfel i $P'\left(\frac{\pi}{2}\right)$, men system matrisen A är korrekt. Uppdatera gärna.

EDIT 18-04-08: Rättade till skrivfelet!

Solution 3-2-05

Polynomet går igenom punkterna (0,2), (2,1), (3,-2), (5,3).

$$\begin{cases} (0,2), (2,1), (3,-2), (5,3) \\ (x_1,y_1), (x_2,y_2), (x_3,y_3), (x_4,y_4) \end{cases}$$

Ansats: $c_0 + c_1 x + c_2 x^2 + c_3 x^3$

$$\begin{cases} p(x_1) = y_1 : c_0 = 2 \\ p(x_2) = y_2 : c_0 + 2c_1 + 4c_2 + 8c_3 = 1 \\ p(x_3) = y_3 : c_0 + 3c_1 + 9c_2 + 27c_3 = -2 \\ p(x_4) = y_4 : c_0 + 5c_1 + 25c_2 + 125c_3 = 3 \end{cases}$$

Detta ger ett linjärt ekvationssystem där antalet ekvationer är lika med antalet okända variabler (Ci), i = 0,1,2,3.
Ovanstående linjära ekvationssystem kan tecknas som Ac = b där:

```
A = [1  0  0  0;...
     1  2  4  8;...
     1  3  9 27;...
     1  5  25 125];
c = [c0 c1 c2 c3]';
b = [2 1 -2 3]';
```

Svar:

```
A = [1  0  0  0;...
     1  2  4  8;...
     1  3  9 27;...
     1  5  25 125];
b = [2 1 -2 3]';
```

Moderator: korrekt. Bra. c-vektorn (som inte efterfrågades) fås genom c=A\b.

Solution 3-2-06

a) Konditionstalet för en matris A_{Newton} har oftast ett lägre konditionstal än en vandermondematris A, eftersom A_{Newton}är triangulär (i jämförelse med A, som inte nödvändigtvis behöver vara det).

b) Ett polynom som vi löser med Newtons metod har formen

$$p(x) = d_0 + d_1(x - x_0) + d_2(x - x_0)(x - x_1) + \ldots + d_n(x - x_0)(x - x_1)\cdots(x - x_{n-1})$$

För att lösa systemet givet punkterna $(x_0, p(x_0)), (x_1, p(x_1)) \ldots (x_n, p(x_n))$ kan man lösa följande ekvation

$$\begin{bmatrix} p(x_0) \\ p(x_1) \\ \vdots \\ p(x_n) \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & (x_1 - x_0) & 0 & \cdots & 0 \\ 1 & (x_2 - x_0) & (x_2 - x_0)(x_2 - x_1) & \ddots & 0 \\ \vdots & \vdots & \vdots & \ddots & 0 \\ 1 & (x_n - x_0) & (x_n - x_0)(x_n - x_1) & \cdots & (x_n - x_0)\cdots(x_n - x_{n-1}) \end{bmatrix}}_{A_{Newton}} \begin{bmatrix} d_0 \\ d_1 \\ \vdots \\ d_n \end{bmatrix}$$

Notera då att matrisen A_{Newton} blir undertriangulär.

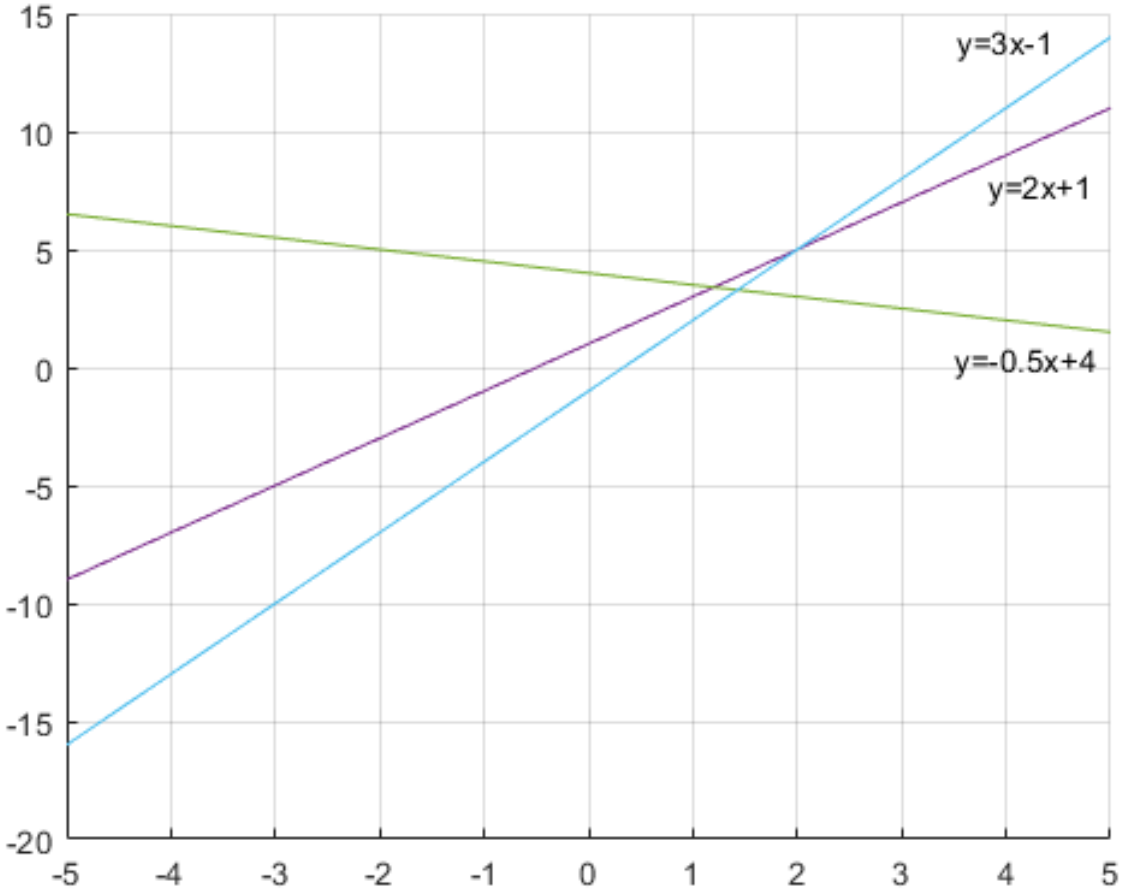
Moderator: Korrekt. Bra. Tillägg (som inte efterfrågades): Newton-ansatsen är bättre än Vandermonde av två anledningar. Det går även oftast snabbare att lösa ett ekvationssystem med en triangulär matris.

Solution 3-2-09

a)

$$\begin{bmatrix} 2 & 1 \\ 0.5 & 1 \\ -3 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1 \\ 4 \\ -1 \end{bmatrix}$$

b)



Det är ett linjärt överbestämt ekvationssystem (fler ekvationer än obekanta), och faktumet att linjerna inte är parallella men inte heller skär varandra i samma punkt visar att det inte finns någon exakt lösning.

c)

Använder vi backslashoperatorn på ekvationssystemet får vi ut vektorn $\binom{a}{b}$ från en minstakvadratlösning på formen $y = ax + b$, alltså en linje som är anpassad till systemet.

Moderator: Korrekt!

Solution 4-2-08

Problemet kan formuleras som ett olinjärt minstakvadratproblem:

$$\mathbf{f}(\mathbf{x}) \approx 0$$

där $\mathbf{x}$ är en vektor som innehåller cirkelns mittpunkt och radie

$$\mathbf{x} = \begin{bmatrix} x_c \\ y_c \\ r \end{bmatrix}.$$

Vi låter varje rad i $\mathbf{f}$ vara en punkt och motsvarar cirkelns ekvation för den punkten. Om vi kallar m för antalet punkter:

$$\mathbf{f}(\mathbf{x}) = \begin{bmatrix} (x_1 - x_c)^2 + (y_1 - y_c)^2 - r^2 \\ (x_2 - x_c)^2 + (y_2 - y_c)^2 - r^2 \\ \vdots \\ (x_m - x_c)^2 + (y_m - y_c)^2 - r^2 \end{bmatrix}.$$

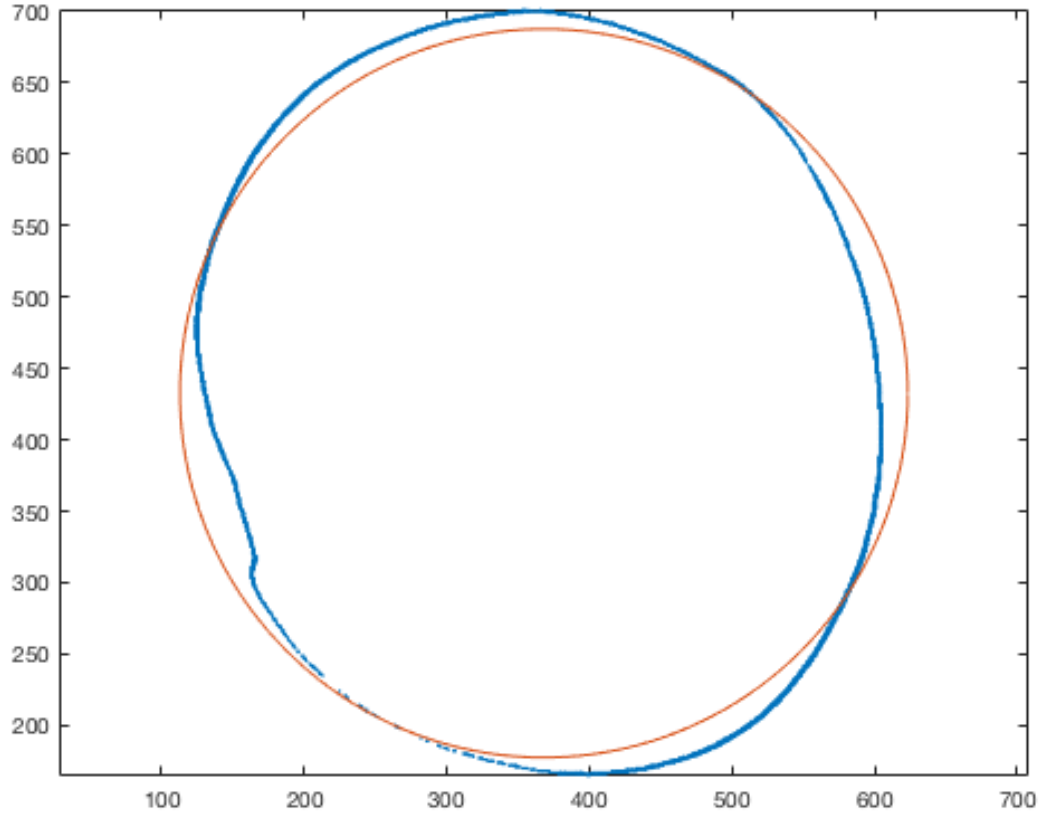
Vi behöver Jacobianen för funktionen, som blir en $m \times 3$-matris:

$$J(\mathbf{x}) = \begin{bmatrix} -2(x_1 - x_c) & -2(y_1 - y_c) & -2r \\ -2(x_2 - x_c) & -2(y_2 - y_c) & -2r \\ \vdots & \vdots & \vdots \\ -2(x_m - x_c) & -2(y_m - y_c) & -2r \end{bmatrix}$$

Löste uppgiften med följande kod:

```
rv = ones(length(xv),1); % Vektor med ettor.


f=@(tv) (xv-tv(1)).^2 + (yv-tv(2)).^2-tv(3).^2;
J=@(tv) [-2*(xv-tv(1)), -2*(yv-tv(2)), -2*rv*tv(3)];


x=[300;450;100];
TOL=1e-10;
h=inf;
iter = 0;
while (norm(h)>TOL)
   h=J(x)\f(x);
   x=x-h;
   norm(h)
   iter = iter +1;
end
```

Det gav den orangea cirkeln med mittpunkt (368.1924, 432.6262) och radie = 255.1041. Den blå cirkeln är de utritade punkterna som använts.



Moderator: Bra! Det är korrekt. För att göra lösningen lättare att förstå för andra studenter har jag lagt till beskrivningen av f och Jacobianen.

Solutions block 3:

Solution 1-3-02

a) Den blåa kurvan representerar centraldifferens. Centraldifferens har nogranhetsordning $O\left(h^2\right)$, vilket innebär att felet (om man bortser från avrundningsfel) har lutning 2 i en graf med log-log skala, vilket vi kan se att den röda kurvan inte har och den blåa har.

b)

Framåtdifferens: $d_f(h) = \frac{e^{2+h}-e^2}{h}$

Centraldifferens: $d_c(h) = \frac{e^{2+h}-e^{2-h}}{2h}$

När vi i dessa har att relativfelet hos $2+h$ (och $2-h$) till 2 är mindre än $\frac{\epsilon_{mach}}{2}$, dvs om $\frac{|2-2+h|}{2} < \frac{\epsilon_{mach}}{2}$, så kommer $2+h$ (och $2-h$) att avrundas till 2 i deras flyttalsrepresesentationer.

detta ger $d_f(h) = \frac{e^{2+h}-e^2}{h} = \frac{e^2-e^2}{h} = \frac{0}{h} = 0$

samt $d_c(h) = \frac{e^{2+h}-e^{2-h}}{h} = \frac{e^2-e^2}{h} = \frac{0}{h} = 0$

vilket ger ett absolutfel hos metoderna på $|e^2 - 0| = e^2 \approx 7.389$ (vilket alltså är det värde graferna i bilden antar).

Storleken av h som ger detta: $R_2 = \frac{|2+h-2|}{2} \leq \frac{\epsilon_{mach}}{2} \Leftrightarrow |h| \leq \epsilon_{mach} \approx 10^{-16}$

Moderator: Perfekt svar!

Solution 1-3-08

Taylor utveckling av en funktion:

$$f(z) = f(a) + f'(a)(z-a) + \frac{f''}{2}(z-a)^2 + \mathcal{O}((z-a)^3)$$

Med z=x+2h och a=x ger det:

$$f(x+2h) = f(x) + f'(x)(2h) + \frac{f''}{2}(2h)^2 + \mathcal{O}(h^3)$$

med z=x-2h och a=x ger det:

$$f(x-2h) = f(x) + f'(x)(-2h) + \frac{f''}{2}(-2h)^2 + \mathcal{O}(h^3)$$

Genom att skriva om den första ekvationen så att vi har endast $f(x)$ på vänstra ledet och därefter sätter in i den andra ekvationen så får vi:

$$f(x-2h) = f(x+2h) - 2hf'(x) - \frac{f''}{2}(2h)^2 - 2hf'(x) + \frac{f''}{2}(2h)^2 + \mathcal{O}(h^3)$$

Detta kan förenklas till:

$$f'(x) = \frac{f(x+2h)-f(x-2h)}{4h} + \mathcal{O}(h^2)$$

Dock så innehåller inte detta f(x). Jag vet inte hur denna term ska bibehållas. Svara gärna ifall det går så jag vet om jag måste försöka lösa denna uppgift på annat sätt.

Solution 1-3-12

Richardsonextrapolation har formeln $R(h) = \frac{2^n F\left(\frac{h}{2}\right) - F(h)}{2^n - 1}$ för en numerisk metod $F(h)$ med noggrannhetsordningen n. Som vi ser här halveras steglängden när Richardsonextrapolation används. Detta leder till att noggrannhetsordningen ökar med minst ett. Det känns nog naturligt approximationen blir bättre om man halverar steglängden och därmed tar fler steg.

För att ge en matematisk förklaring till varför noggrannhetsordningen ökar skriver vi ut vår metod $F(h)$ med noggrannhetsordningen n som approximerar något Q som $Q = F(h) + Ch^n + O\left(h^{n+1}\right)$, som kan skrivas om till: $F(h) = Q - Ch^n + O\left(h^{n+1}\right)$.

Vi vet att detta gäller under antagandet att C nästan är konstant för de relevanta steglängderna (se Sauer 5.1.3 och pres_block3.pdf för motiveringar och exempel). Då blir Richardsonextrapolationen av $F(h)$:

$$\frac{2^n F\left(\frac{h}{2}\right) - F(h)}{2^n - 1} = \frac{2^n\left(Q - C\left(\frac{h}{2}\right)^n + O\left(h^{n+1}\right)\right) - \left(Q - Ch^n + O\left(h^{n+1}\right)\right)}{2^n - 1} = \frac{(2^n - 1)Q - \frac{2^n C h^n}{2n} + Ch^n + (2^n - 1)O\left(h^{n+1}\right)}{2^{n-1}}$$

Som vi ser försvinner termerna med Ch^n och efter divisionen återstår enbart $Q + O\left(h^{n+1}\right)$. Därmed har noggrannhetsordningen blivit minst en grad högre efter Richardsonextrapolationen.

Solution 2-3-04

En funktion som heter simpson

```
function [I,felet]=simpson(f,a,b,n)
%programmet ber¨aknar integralen av funktionen f p?a intervallet [a,b]
%med Simpsons formel. n ¨ar antalet steg och m?aste vara ett j¨amnt heltal.
%f kan anges som en m-filsfunktion eller som en anonym funktion.
if not(n/2==floor(n/2))
error('The number of N has to be even');
else
h=(b-a)/n;
S=f(a);
for i=1:2:n-1
x(i)=a+h*i;
S=S+4*f(x(i));
end
for i=2:2:n-2
x(i)=a+h*i;
S=S+2*f(x(i));
end
S=S+f(b);
I=h*S/3;
felet=abs(I-(6*sin(3) - 7*cos(3) - 2));
end
```

Om vi kör det

[I felet]=simpson(f,0,3,4);

där h=0.75

så får vi 5.762119239822489 med fel som är 0.014548284739832

b)-

För att h=0.1 --> n=30

f=@(x) (x^2)*sin(x);
[I felet]=simpson(f,0,3,30);

Så får vi 5.776664406730492 med fel som är 3.117831829158035e-06

c)-

h=0.21428571 --> n=14 och detta ger noggranhete med 4 decimala

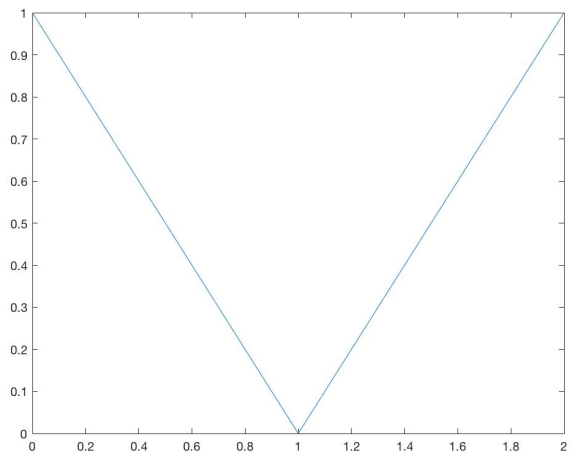
Svaret 5.776600001997786 och felet är 6.752256453523842e-05

Moderator: Lösningen är nästan korrekt, bra! Tänk på att du inte kan jämföra det numeriska resultatet med ett analytiskt resultat som är avrundat (5.77667). Använd istället uttrycket för det analytiska resultatet när du ska beräkna felet så att du får med så många siffror som möjligt. Ett tips är att en funktion i matlab kan returnera mer än en output, om du skriver [I, felet] = simpson(f,a,b,n).

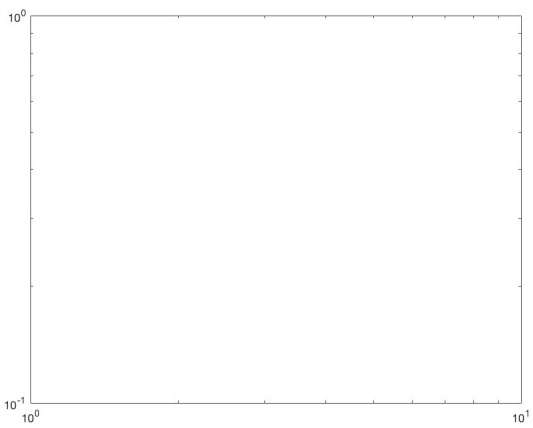
Solution 2-3-08

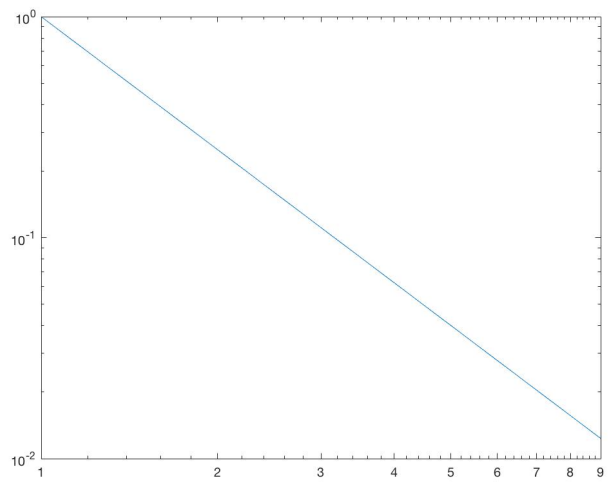
a) Det exakta värdet är 1.

Här är en figur över integranden för intervallet [0,2]:

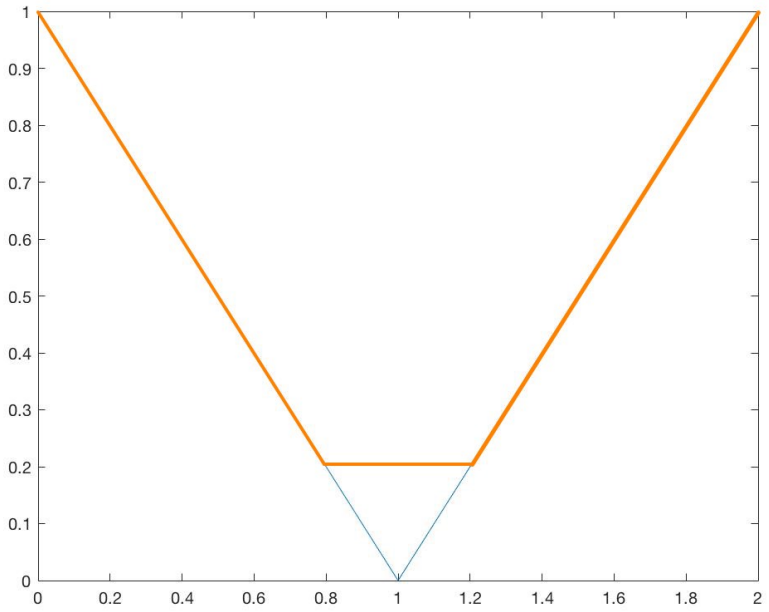


Figurer för **b)** respektive **c)** följer nedan. m-värdena är på x-axeln och felet på y-axeln. Båda ritades med loglog, även om den vänstra (som visar $m = 2, 4, 6, \ldots$) inte blir så intressant i det fallet, eftersom felet alltid blir 0 för jämna värden på m.





Förklaring till beteendet, alltså att felet alltid blir 0 om m är jämnt men inte udda, har helt att göra med sättet ingranden ser ut på över intervallet [0,2], nämligen som två trianglar av exakt samma storlek. Eftersom trapetsregeln rent visuellt delar upp området mellan x-axeln och funktionskurvan för integranden i trapetser vars areor sedan adderas, så kommer ju trapetsregelen för jämna m att ge trapetser som passar in perfekt på integranden, som har en symmetri kring $x = 1$, alltså mitten av intervallet vi integrerar över, [0, 2]. Men för udda m delas de två trianglarna in i ett udda antal delar, vars bredd är lika stora, vilket ger att trapetserna inte passar figuren längre. Se bild nedan.



I denna figur är $m = 5$, och om man beräknar arean mellan x-axeln och de orangea linjerna ovan får man samma svar som trapetsregeln skulle ge, där felet alltså uppstår vid mitten-partitionen och i detta fall blir 0.04.

<u>Svar</u>: *För att sammanfatta beror alltså beteendet på att trapetsregelns trapetser passar integranden i denna uppgift perfekt om vi delar in intervallet i jämna delar, men inte udda, vilket beror på att vi över intervallet i fråga har två symmetriska trianglar av samma storlek, alltså ett jämnt antal. Således kan vi inte dela in trianglarna i ett udda antal trapetser med samma bredd utan att få ett fel, men ett jämnt antal går bra.*

Moderator: Bra! Korrekt.

Solution 2-3-10

Bakåtdifferens är :

$f'(x) \approx \frac{(f(x)-f(x-h))}{h}$

Först taylorutveckling kring x:

$f(x-h) \;=\; f(x) - f'(x) \cdot h + \frac{f''(c)}{2} \cdot h^2$ där $c \in [x,\, x+h]$.

Skriver ihop täljaren:

$$f(x) \;-\; f(x-h) \;= f(x) - \left(f(x) - f'(x) \cdot h + \frac{f''(c)}{2} \cdot h^2\right) \;= f'(x) \cdot h - \frac{f''(c)}{2} \cdot h^2$$

$$\Longrightarrow\; f'(x) \;=\; \frac{\left(f'(x) \cdot h - \frac{f''(c)}{2} \cdot h^2\right)}{h} \;=\; f'(x) + \frac{f''(c)}{2} \cdot h$$

Detta visar alltså att noggranhetsordningen är 1, dvs O(h)

Solution 3-3-01

Taylorutveckling runt x:

$$f\left(x+h\right)=f\left(x\right)+f'\left(x\right)h+\frac{f''(x)}{2!}h^2+\frac{f'''(c_1)}{3!}h^3,\ c_1\in\left[x,\ x+h\right]$$

$$f\left(x-h\right)=f\left(x\right)-f'\left(x\right)h+\frac{f''(x)}{2!}h^2-\frac{f'''(c_2)}{3!}h^3,\ c_2\in\left[x-h,\ x\right]$$

Centraldifferens:

$$\frac{f(x+h)-f(x-h)}{2h}=\frac{f(x)+f'(x)h+\frac{f''(x)}{2!}h^2+\frac{f'''(c_1)}{3!}h^3-f(x)+f'(x)h-\frac{f''(x)}{2!}h^2+\frac{f'''(c_2)}{3!}h^3}{2h}=f'\left(x\right)+\left(\frac{f'''(c_1)+f'''(c_2}{3!\cdot 2}\right.$$

Noggrannhetsordningen blir 2.

Moderator: Bra! Vad blir då noggrannhetsordningen? Korrigerad.

Solution 3-3-02

Vi har att $f(x)=sin(x)$. Därför är $f'(x)=cos(x)$ och $f''(x)=-sin(x)$.

Som bekant är $-sin(c)$ negativt och växande då $c\in[2,2.1]$, vilket medför att det absoluta maxvärdet (jag antar att det är detta som söks) inom intervallet finns då $c=2$.

Därför blir det maximala felet lika med $\frac{|f''(2)|}{2!}0.1^2=0.005\,|-sin(2)|\approx 0.00455$.

Solution 3-3-07

Taylorutvecklar och får:

$$f(x-h) = f(x) - f'(x) * h + \frac{f''(x)}{2} * h^2 + O(h^3)$$

$$f(x+h) = f(x) + f'(x) * h + \frac{f''(x)}{2} * h^2 + O(h^3)$$

Alltså:

$$a * f(x-h) + b * f(x) + c * f(x+h) =$$

$$a * (f(x) - f'(x) * h + \frac{f''(x)}{2} * h^2 + O(h^3)) + b * f(x) + c * (f(x) + f'(x) * h + \frac{f''(x)}{2} * h^2$$

Vi kan formulera detta som ett ekvationssystem för att eliminera så många dominerande feltermer som möjligt:

$$\begin{bmatrix} f(x) & f(x) & f(x) \\ -f'(x) * h & 0 & f'(x) * h \\ f''(x) * h^2 & 0 & f''(x) * h^2 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 0 \\ f'(x) \\ 0 \end{bmatrix}$$

Vi tar endast med tre rader, ty vi vet att om vi lägger till en till rad så blir systemet överbestämt och vi kommer inte kunna lösa det exakt, d.v.s. vi kommer inte att kunna eliminera några flera termer. Lösning av ovan system ger $a = \frac{-1}{2h}, \quad b = 0, \quad c = \frac{1}{2h}$.

Insättning ger:

$$\frac{-1}{2h} * f(x-h) + 0 * f(x) + \frac{1}{2h} * f(x+h) =$$

$$\frac{-f(x)+f(x)+f'(x)*h+f'(x)*h-f''(x)/2+f''(x)/2+O(h^3)}{2h} = \frac{2f'(x)*h+O(h^3)}{2h} = f'(x) + O(h^2)$$

Alltså ges den bästa derivataapproximationen av:

$f'(x) \approx \frac{f(x+h)-f(x-h)}{2h}$ med en dominerande felterm $O(h^2)$. Vi kan observera att detta är den vanligt förekommande centraldifferens-approximationen av en förstaderivata.

Solution 3-3-08

Vi vet att $f(x)$ går igenom punkterna $(x_0, f(x_0)), (x_1, f(x_1)), (x_2, f(x_2))$. Vi kan därför lösa problemet med följande ekvationssystem.

$$\begin{bmatrix} f(x_0) \\ f(x_1) \\ f(x_2) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & (x_1 - x_0) & 0 \\ 1 & (x_2 - x_0) & (x_2 - x_0)(x_2 - x_1) \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \end{bmatrix}$$

Utför vi multiplikationen på HL får vi

$$\begin{bmatrix} f(x_0) \\ f(x_1) \\ f(x_2) \end{bmatrix} = \begin{bmatrix} \alpha_0 \\ \alpha_0 + \alpha_1(x_1 - x_0) \\ \alpha_0 + \alpha_1(x_2 - x_0) + \alpha_2(x_2 - x_1)(x_2 - x_0) \end{bmatrix} = \begin{bmatrix} \alpha_0 \\ \alpha_0 + \alpha_1 h \\ \alpha_0 + \alpha_1 2h + \alpha_2 2h^2 \end{bmatrix}$$

Därefter kan vi väldigt lätt lösa ut $\alpha_0, \alpha_1, \alpha_2$ genom vanlig substitution, som jag hoppar över här. Vi får då konstanterna

$$\begin{cases} \alpha_0 = f(x_0) \\ \alpha_1 = \dfrac{f(x_1) - f(x_0)}{h} \\ \alpha_2 = \dfrac{f(x_2) - 2f(x_1) + f(x_0)}{2h^2} \end{cases}$$

Sätter vi in de här konstanterna i $f(x)$ får vi det interpolerande polynomet.

Solution 3-3-11

Framåtdifferens av f(x):

$$f'(x) \approx g(x) = \frac{f(x+h)-f(x)}{h} \text{ (*)}$$

Bakåtdifferens av g(x):

$$g'(x) \approx f''(x) = \frac{g(x)-g(x-h)}{h} \text{ (**)}$$

(**) tillsammans med (*):

$$g(x) = \frac{f(x+h)-f(x)}{h},\ g(x-h) = \frac{f(x)-f(x-h)}{h}$$

$$f''(x) = \frac{\frac{f(x+h)-f(x)}{h} - \frac{f(x)-f(x-h)}{h}}{h} = \frac{f(x+h)-2f(x)+f(x-h)}{h^2}$$

Vilket är densamma som centraldifferensformeln för andraderivatan.

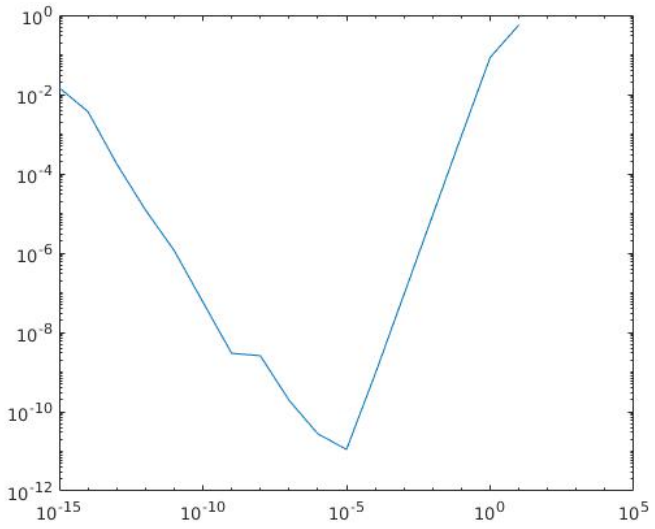
Solution 4-3-01

a)

Det går att hitta två programmeringsfel i programmet. Det första är att fpapprox är definierad att ta en variabel x0, men använder aldrig den utan istället använder en variabel x. Det andra felet är att h i nämnaren i fpapprox saknar koefficienten 2. Det korrigerade programmet är:

```
>> hv=10.^(-15:1);
>> f=@(x) sin(x); fp=@(x) cos(x);
>> fpapprox=@(x,h) (f(x+h)-f(x-h))./(2.*h);
>> errv=abs(fp(1)-fpapprox(1,hv));
>> loglog(hv,errv);
```

Och ger plotten:



b)

I grafen ser man att felet först minskar och sedan ökar. I problemet står det att diskretiseringsfelet blir stort när h blir litet, men det är inte sant. Grafen visar i själva verket totalfelet, alltså diskretiseringsfel och avrundningsfel tillsammans. Detta är anledningen till varför felet växer när h blir mindre. För små värden på h uppstår kancellation i subtraktionen $f(x+h)-f(x-h)$ eftersom talen blir mer lika ju mindre h blir.

Svaret är alltså att totalfelet blir stort när h blir litet på grund av att avrundningsfelet blir stort, trots att diskretiseringsfelet fortsätter avta.

Moderator: Korrekt. Bra!

Solution 4-3-02

För att minimera totalfelet vill vi minimera

$$E(h) = \frac{|f''(x)|}{2} + 2\frac{\epsilon_{mach}}{h}|f(x)|$$

vilket ger

$$h = \sqrt{\frac{4[f(x)]}{[f''(x)]}\epsilon_{mach}} = 8.688795409866125\text{e-08}$$

Används sedan framåtdifferensen i punkten $x = -2$ ges

$$d = \frac{f(x+h)-f(x)}{h} = 1.000000087650228$$

vilket evaluerats analystisk till 1.

Moderator: Något är knas med första formeln. Korrigera gärna.

Solution 4-3-04

Beräknar först värdet analytiskt för att få en bild av vad ett rimligt svar är.

$$\int_0^1 2x^2 + 3x + 1\, dx = \left[\frac{2}{3}x^3 + \frac{3}{2}x^2 + x\right]_0^1 = \frac{2}{3} + \frac{3}{2} + 1 = \frac{19}{6} = 3.166...$$

Använder sedan följande matlabkod för att beräkna integralen numeriskt.

Följande funktioner används:

trapets:

```
function [integral] = trapets(f, a, b, h)
integral = (f(a) + f(b))/2;
for x=a+h:h:b-h
    integral = integral + f(x);
end
integral = h * integral;
end
```

Simpson:

```
function integral=simpson(f,a,b,m)
xv=a:m:b;
integral=f(a)+f(b);
for k = 2:2:length(xv)-1
    integral = integral+4*f(xv(k));
    integral = integral+2*f(xv(k+1));
end
integral = integral - 2*f(xv(end));
integral=integral*m/3;
```

Huvudprogrammet:

```
format long;

% ∫2x^2+3x+1 dx
% Analytiskt värde 19/6
integral = 19/6

f = @(x)(2*x^2+3*x+1);
a = 0;
b = 1;
h = 0.5;

% Använder trapetsfunktionen
integralt = trapets(f, a, b, h)

% Använder Simpsons
integrals = simpson(f, a, b, h)

% Räknar ut absolutfelen

dift = abs(integralt-integral)
difs = abs(integrals-integral)
```

Vilket ger följande svar:

$integral_t = 3.250..., dif_t = 0.0833$

$integral_s = 3.166..., dif_s = 0$

Anledningen att Simpson beräknar integralen exakt är på grund utav hur metoden funkar. Simpson är Richardson-extrapolation av trapetsmetoden, den anpassar flergradskurvor till funktionen. Då funktionen är en flergradskurva, kommer simpson kunna göra en exakt approximering, till skillnad från om det inte var det. Det är därför den kan beräkna exakt. Trapetsregeln funkar istället genom att anpassa rät linjer till funktioner, vilket oftast inte kommer ha samma noggrannhet.

Solution 4-3-05

Taylorutveckling för $f'(x) \approx a \cdot f(x) + b \cdot f(x+h) + c \cdot f(x+2h)$ av för f(x+h) och f(x+2h) av grad 2 runt punkten x evaulerat i punkterna (x+h) och (x+2h) ger

$$f'(x) \approx a \cdot f(x) + b\left(f(x) + h \cdot (f'(x)) + \frac{h^2 f''(x)}{2} + O\left(h^3\right)\right) + c \cdot \left(f(x) + 2 \cdot h \cdot f'(x) +\right.$$

Detta ger ekvationssystemet

$a + b + c = 0$

$h \cdot b + 2h \cdot c = 1$

$\frac{h^2}{2} \cdot b + 2h^2 \cdot c = 0$

Som har lösningen $a = -\frac{3}{2h},\ b = \frac{2}{h},\ c = -\frac{1}{2h}$

Derivatan kan därför approximeras som

$$f'(x) = \frac{\left(-\frac{3}{2} \cdot f(x) + 2 \cdot f(x+h) - \frac{1}{2} f(x+2h)\right) + O\left(h^3\right)}{h} = \frac{\left(-\frac{3}{2} \cdot f(x) + 2 \cdot f(x+h) - \frac{1}{2} f(x+2h)\right)}{h} + O\left(h^2\right)$$

Detta stämmer med det som står på Wikipediasidan under rubriken "Forward and Backward finite difference".

Solution 4-3-06

Med trapetsregeln delar vi in vår graf i delintervall och försöker interpolera varje intervall till en rät linje med hjälp av de två ändpunkterna på intervallet, och approximerar därefter integralen till summan av integralerna av våra räta linjer i respektive delintervall. Eftersom funktionen de försöker integrera är den räta linjen $f(x) = x$, kommer approximationer av integralerna som B gör för varje delintervall bli exakt. Därför kommer B lyckas räkna ut integralen korrekt.

Med Simpsons regel delar vi in grafen i delintervall och försöker interpolera varje intervall till ett andragradspolynom med hjälp av intervallets två ändpunkter och en punkt inom intervallet, och approximerar därefter integralen till summan av integralerna av polynomen i respektive delintervall. Även om $f(x)$ inte är ett andragradspolynom, kommer även A:s approximationer för integralerna av delintervallen vara exakta. Detta gäller eftersom vi kommer få fram den exakta funktionen om vi polynominterpolerar en funktion som är ett polynom av högst grad 2, med hjälp av 3 punkter på grafen och Newton-ansatsen (vilket vi gör när vi använder Simpsons regel).

Hade de istället försökt lösa $\int_0^{10} x^2 dx$, hade de fått olika resultat. A kommer fortfarande lyckas lösa integralen exakt, eftersom funktionen fortfarande är ett polynom som inte har högre grad än 2. B, däremot, skulle använda sig av approximerade integraler för varje delintervall, eftersom vår funktion är ett andragradspolynom, och inte en rät linje.

Moderator: snygg lösning!

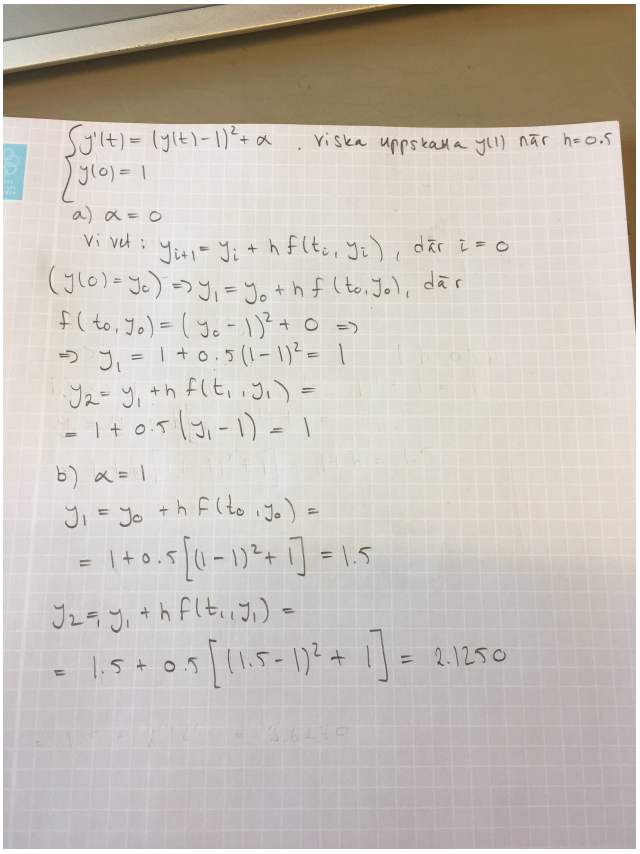
Solution 4-3-13

```
f = @(x) sin(exp(0.1*x^2))*cos(exp(sqrt(x)));
a = 0;
b = 5;
h = 0.1;
m = (b - a)/h;

sum = 0;
for i=1:m-1
sum = sum + f(a+i*h);
end
h*(1/2*f(a)+1/2*f(b) + sum)
```

Detta program approximerar den sökta integralen med steglängd 0.1 och får resultatet -1.267841967039195

Solutions block 4:

[Solution 1-4-01]



$$\begin{cases} y'(t) = (y(t)-1)^2 + \alpha \\ y(0) = 1 \end{cases}$$

. Vi ska uppskatta $y(1)$ när $h=0.5$

a) $\alpha = 0$

Vi vet: $y_{i+1} = y_i + h f(t_i, y_i)$, där $i = 0$

$(y(0) = y_0) \Rightarrow y_1 = y_0 + h f(t_0, y_0)$, där

$f(t_0, y_0) = (y_0 - 1)^2 + 0 \Rightarrow$

$\Rightarrow y_1 = 1 + 0.5(1-1)^2 = 1$

$y_2 = y_1 + h f(t_1, y_1) =$

$= 1 + 0.5(y_1 - 1) = 1$

b) $\alpha = 1$

$y_1 = y_0 + h f(t_0, y_0) =$

$= 1 + 0.5\left[(1-1)^2 + 1\right] = 1.5$

$y_2 = y_1 + h f(t_1, y_1) =$

$= 1.5 + 0.5\left[(1.5-1)^2 + 1\right] = 2.1250$

Moderator: Lösning är ej fullständig. Hur många Euler steg behöver du ta med steglängd h=0.5 för att komma till t=1?

Svar: Ser mitt misstag, krävs två steg.

Moderator: Ok. Bra. Ladda gärna upp en ny lösning så andra ser vad som är rätt.

Svar: Det är redan gjort.

Moderator: Bra!

Solution 1-4-09

y(1) = 71.9973.

wiki1409.m

```
y=[1,1];        %begynnelsevärdet
yv=[y(1)];
h=0.1;
t=0;
tv=[t];

for i=1:10
    t=t+h;
    tv=[tv;t];
    y=y+h*mineuler(t,y);
    yv=[yv;y(1)];
end

plot(tv,yv)
t1 = yv(11)
%yv(11) är approximerade värdet för t=1, då yv(1)
%motsvarar t=0, och t har ökat med 0.1 för varje loop.
```

ommand Window

```
>> wiki1409

t1 =

   71.9973
```

Moderator: okej. Svaret är rätt även om du gjorde ett steg för mycket med Euler (eftersom vektorn yv blir längd 12 och du väljer ut näst sista värdet).

- Har ändrat i svaret. Detta borde vara mer korrekt.

Solution 1-4-12

a)
$y''(t) + y'(t) + 2y(t) + \sin(t) = 0$

blir efter införandet av nya variabler y_1=y, y_2=y':

$$\begin{cases} y_1' = y_2(t) \\ y_2' = -y_2 - 2y_1 - sin(t) \end{cases}$$

b) Jag beslöt att använda Eulers stegmetod framåt

Bifogat här är koden:

```
% Euler framåt

F=@(t, y) ([ y(2) , -2*y(1)-1*y(2)-1*sin(t)]);
a=0;
b=3.5;
h=0.1; %stepsize
m=(b-a)/h; %antal steg
t=a;
y=[-1,1]; %begynnelsevärde
yv=zeros(35,1);
for i=1:m
    y = y + h*F(t, y);
    t = t + h;
    yv(i)=y(2); %Vektor för alla steg
end
```

Det sista värdet i yv eller y(2) efter exekvering av koden är resultatet:

y'(3.5) = 0,371788711619431

Solution 1-4-13

a)

Vi definierar en ny funktion z på följande sätt:

$$\mathbf{z}\left(\mathbf{t}\right) = \begin{bmatrix} z_1(t) \\ z_2(t) \\ \vdots \\ z_9(t) \\ z_{10}(t) \end{bmatrix} = \begin{bmatrix} y(t) \\ y'(t) \\ \vdots \\ y^{(8)}(t) \\ y^{(9)}(t) \end{bmatrix}$$

Dess derivata är då:

$$\mathbf{z}'\left(\mathbf{t}\right) = \begin{bmatrix} z_1'(t) \\ z_2'(t) \\ \vdots \\ z_9'(t) \\ z_{10}'(t) \end{bmatrix} = \begin{bmatrix} y'(t) \\ y''(t) \\ \vdots \\ y^{(9)}(t) \\ y^{(10)}(t) \end{bmatrix} = \begin{bmatrix} z_2(t) \\ z_3(t) \\ \vdots \\ z_{10}(t) \\ z_1(t) + cos(t) \end{bmatrix},$$

där det sista elementet i den sista likheten ges av att $y^{(10)}(t) = y(t) + cos(t)$ från frågan.

Vi kan nu skriva upp ekvationen på standardform:

$$\mathbf{z}'\left(\mathbf{t}\right) = \begin{bmatrix} z_2(t) \\ z_3(t) \\ \vdots \\ z_{10}(t) \\ z_1(t) + cos(t) \end{bmatrix}$$

$$\mathbf{z}\left(\mathbf{0}\right) = \begin{bmatrix} z_1(0) \\ z_2(0) \\ \vdots \\ z_9(0) \\ z_{10}(0) + cos(0) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}$$

b)

$$\mathbf{z}'(\mathbf{t}) = A\mathbf{z}(t) + \mathbf{g}(t)$$

$$= \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \ddots & 0 \\ \vdots & \vdots & \ddots & \ddots & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} z_1(t) \\ z_2(t) \\ \vdots \\ z_9(t) \\ z_{10}(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ cos(t) \end{bmatrix}$$

[Solution 1-4-15]

$$\bar{u} = \begin{cases} u_1 = q_1(t) \\ u_2 = q_1'(t) \\ u_3 = q_2(t) \\ u_4 = q_2'(t) \end{cases} \qquad \begin{aligned} u_1' &= u_2 \\ u_2' &= u_1 u_3 \\ u_3' &= u_4 \\ u_4' &= 1 + u_1 u_3 \end{aligned}$$

$$u' = f(t, \bar{u}) \quad \text{där} \quad \bar{u}_0 = \begin{bmatrix} 3 \\ 0 \\ 3 \\ 1 \end{bmatrix}$$

$$\text{och} \quad f(t, \bar{u}) = \begin{bmatrix} u_2 \\ u_1 u_3 \\ u_4 \\ 1 + u_1 u_3 \end{bmatrix}$$

Moderator: Korrekt!

Solution 1-4-19

Ett steg med Euler bakåt blir:

$$\mathbf{y}_1 = \mathbf{y}_0 + h(A\mathbf{y}_1 + \mathbf{b})$$
$$\{\mathbf{y}_0 = \mathbf{0}\} \Rightarrow$$
$$\mathbf{y}_1 = hA\mathbf{y}_1 + h\mathbf{b}$$
$$\mathbf{y}_1 - hA\mathbf{y}_1 = h\mathbf{b}$$
$$(I - hA)\mathbf{y}_1 = h\mathbf{b}$$
$$\mathbf{y}_1 = (I - hA)^{-1}h\mathbf{b}$$
$$\left\{ h = \tfrac{1}{10}, A = \begin{bmatrix} 2 & 0 \\ 0 & 4 \end{bmatrix}, b = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \right\} \Rightarrow$$
$$\mathbf{y}_1 = \mathbf{y}_1 = (I - \tfrac{1}{10}\begin{bmatrix} 2 & 0 \\ 0 & 4 \end{bmatrix})^{-1}\tfrac{1}{10}\begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$$
$$\mathbf{y}_1 = (I - \begin{bmatrix} \frac{1}{5} & 0 \\ 0 & \frac{2}{5} \end{bmatrix})^{-1}\begin{bmatrix} \frac{1}{10} \\ \vdots \\ \frac{1}{10} \end{bmatrix}$$
$$\mathbf{y}_1 = \begin{bmatrix} 1 - \frac{1}{5} & 0 \\ 0 & 1 - \frac{2}{5} \end{bmatrix}^{-1}\begin{bmatrix} \frac{1}{10} \\ \vdots \\ \frac{1}{10} \end{bmatrix}$$
$$\mathbf{y}_1 = \begin{bmatrix} \frac{4}{5} & 0 \\ 0 & \frac{3}{5} \end{bmatrix}^{-1}\begin{bmatrix} \frac{1}{10} \\ \vdots \\ \frac{1}{10} \end{bmatrix}$$
$$\mathbf{y}_1 = \begin{bmatrix} \frac{5}{4} & 0 \\ 0 & \frac{5}{3} \end{bmatrix}\begin{bmatrix} \frac{1}{10} \\ \vdots \\ \frac{1}{10} \end{bmatrix}$$
$$\mathbf{y}_1 = \begin{bmatrix} \frac{5}{4} * \frac{1}{10} \\ \frac{5}{3} * \frac{1}{10} \end{bmatrix}$$
$$\mathbf{y}_1 = \begin{bmatrix} \frac{1}{8} \\ \frac{1}{6} \end{bmatrix}$$

Solution 2-4-13

Från systemet får vi

$f\,(t\,,\,y)\;=\;3y$ och $y\,(t)\;=e^{3t}$.

Metoden för Euler framåt är:

$y_{i+1} = y_i + h \cdot (t_i\,,\,y_i)$

Alltså tre steg med Euler framåt, där steglängden $h = 0.1$ ger:

$y\,(0) = y_0$

$y\,(t_1)\;\approx y_1 = y_0 + h \cdot f\,(t_0\,,\,y_0)\;=\;y_0 + h \cdot 3 \cdot y_0 = 1 + 0.1 \cdot 3 \cdot 1\;=\;1.3$

$y\,(t_2) \approx y_2 = y_1 + h \cdot f\,(t_1,\,y_1) = y_1 + h \cdot 3 \cdot y_1 = 1.3 + 0.1 \cdot 3 \cdot 1.3 = 1.69$

$y\,(t_3) \approx y_3 = y_2 + h \cdot f\,(t_2\,,\,y_2) = y_2 + h \cdot 3 \cdot y_2 = 1.69 + 0.1 \cdot 3 \cdot 1.69 = 2.197$

Nu har jag tagit fram de tre första punkterna/approximationerna av funktionen $y\,(t) = e^{3t}$.

Moderator: Fint! Korrekt.

Solution 2-4-04

Problemet på sidan är som följer:

$y' = -2.3y$

$y(0) = 1$

Enligt Elias presentation *pres_block4.pdf* utgörs stabilitetesområdet för Euler framåt då av alla $z = \lambda h$ som uppfyller $|1 + z| < 1$, där $\lambda = (-2.3)$ i detta fall. För Euler bakåt blir motsvarande $|1 - z| > 1$.

a) Vi har $|1 + \lambda h| < 1$. Detta ger att alla $h \in \left(0, -\frac{2}{\lambda}\right)$ ger stabilitet. Eftersom $\lambda = (-2.3)$ är alltså *svaret* $h \in \left(0, \frac{2}{2.3}\right)$

b) Vi har $|1 - \lambda h| > 1$. Detta ger att alla $h \in (0, \infty) \cup \left(-\infty, \frac{2}{\lambda}\right)$ ger stabilitet. Eftersom $\lambda = (-2.3)$ och $h > 0$ är alltså *svaret* $h \in (0, \infty)$.

Moderator: Korrket. Bra. Jag fixade lite i formler för tydlighet.

Solution 2-4-05

This is an ODE of order 2, so in order to solve this, we have to first re-construct the problem to be entered around a system of differential equation. Let us define the following terms:

$$\begin{cases} y_1(t) = y(t) \\ y_2(t) = y'(t) \end{cases}$$

=> $y_1'(t) = y'(t) = y_2(t)$ and
$y_2'(t) = y''(t) = sin(y(t)) + y'(t)cos(t) = sin(y_1(t)) + y_2(t)cos(t)$

We then define:

$y_{vector}(t) = \binom{y_1(t)}{y_2(t)}$ and $y'_{vector}(t) = \binom{y_2(t)}{sin(y_1(t))+y_2(t)cos(t)}$ and
$f_{vector}(t, y_{vector}(t)) = y'_{vector}(t)$

We have the start vector $y_0 = \binom{\pi}{2}$ given from the problem. Now we can use an Euler forward recursion to solve for y_2 (this will require 2 iterations):

$$y_{i+1} = y_i + h * f_{vector}(t_i, y_{vector}(t_i))$$

The first iteration gives us $y_1 = \binom{\pi}{2} + 0.1 * \binom{2}{2} = \binom{\pi+0.2}{2.2}$

and the second gives us
$y_2 = \binom{\pi+0.2}{2.2} + 0.1 * \binom{2.2}{sin(\pi+0.2)+2.2cos(0.1)} \approx \binom{\pi+0.2}{2.2} + \binom{0.22}{0.2} = \binom{\pi+0.42}{2.4}$

Solution 2-4-06

Euler framåt

På intervallet a ≤ t ≤ b har vi:

$$\begin{cases} y' = \ f(t,y) \\ y(a) = y_a \\ t \in [a,b] \end{cases}$$

Begynnelsevärdet ger oss (för $a = t_0$) att:
$y'(t_0) = f(t, y_0)$
och använder vi framåtdifferens för approximationen av $y'(t_0)$ får vi (för små h):

$$\frac{y(t_0+h) - y(t_0)}{h} \approx f(t, y_0)$$

varifrån vi kan lösa ut:

$y(t_0 + h) \approx y_0 + h * f(t, y_0)$

och y_1 är ju en approximation av $y(t_1) = y(t_0 + h)$.

Alltså:

$y_1 = y_0 + h * f(t_0, y_0)$

Resonemanget upprepas för $y_2, \ldots$.

Euler bakåt

Görs på liknande sätt som ovan, men vi använder oss istället av bakåtdifferens:

$y'(t_1) = f(t_1, y(t_1))$

vilket ger (nu används bakåtdifferensen istället)

$$\frac{y(t_1) - y(t_0)}{h} \approx f(t_1, y(t_1))$$

dvs

$y(t_1) \approx y_0 + h * f(t_1, y(t_1))$

och därmed kan y_1 som är en approximation av $y(1)$ definieras:

$0 = y_0 + h * f(t_1, y_1) - y_1$

Resonemanget upprepas för $y_2, \ldots$.

(Detta gör Euler bakåt till en implicit metod, dvs. vi får en ekvation vi måste lösa. Detta i motsats till Euler framåt som är en explicit metod, där vi bara behöver "plugga in värden" för att få våra svar.)

Moderator: Bra! Jag fixade lite småsaker (grönt).

Solution 2-4-08

Enligt villkor för stabilitet vill vi att $h\lambda$, där λ är matrisens egenvärden, ska ligga i stabilitetsområdet. Eftersom matrisen är triangulär vet vi att α och -8 är dess egenvärden. Vi behöver välja h så att båda villkoren är uppfyllda

$h\alpha \in [-4, 0]$

$-8h > -4$

Vi behöver alltså att

* $\alpha \leq 0$

* $h\alpha > -4$

* $h < 1/2$

Solution 2-4-15

Problemet motsvarar diskretisering med matrismetoden av randvärdesproblemet

$y''(x) = x^2$

$y(0) = -10$

$y(1) = 5$

Solution 2-4-16

Eulers metod kan användas för att approximativt lösa ordinära differentialekvationer. Formeln är som följer:

$y_{n+1} = y_n + h \cdot F(x_n, y_n)$

$F(x, y) = y'(x) = 10x + y(x)^2$

där h är steglängden och $x_{n+1} = x_n + h$

I det här fallet är begynnelsevärdena $y_0 = 10$ och $x_0 = 0$.

Vi beräknar det första steget y_1:

$y_1 = y_0 + h\left(10x_0 + (y_0)^2\right)$

$y_1 = 10 + 0.1\left(10 \cdot 0 + 10^2\right) = 10 + 0.1 \cdot 100 = 20$

Vi kan nu beräkna det andra steget y_2 då $x_1 = 0.1$:

$y_2 = y_1 + h\left(10x_1 + (y_1)^2\right)$

$y_2 = 20 + 0.1\left(10 \cdot 0.1 + 20^2\right) = 20 + 0.1(1 + 400) = 20 + 40.1 = 60.1$

Solution 2-4-20

Låt oss tillämpa Euler Framåt på testproblemet

$y' \; = \; \lambda y$

$y\left(0\right) = 1$

Alltså får vi

$u_{n+1} = u_n + h\lambda u_n \; , \; u_0 = 1$

och detta kan skrivas om till

$u_{n+1} = \left(1 + h\lambda\right) u_n = \left(1 + h\lambda\right)^2 u_{n-1} = \{faktorisera\} = \left(1 + h\lambda\right)^{n+1} u_0 = \{u_0 = 1\} = (1$

Enligt definitionen för absolutstabil, för att talföljden $\{u_n\}$ ska gå mot noll måste
$|1 + h\lambda| < 1$

och därmed har vi fått fram stabilitetsområdet för Euler Framåt.

Källa: Ant-Absolutstabilitet.pdf

Solution 2-4-21

Let us first assume an initial condition $y'(0) = \beta$. We can now rewrite the original problem as an initial value problem dependent on β, that is,

$$y''_\beta(x) = -5y'_\beta(x) - 4y_\beta(x),$$

$$y_\beta(0) = 1, \quad y'_\beta(0) = \beta.$$

We will now use a change of variable to transform this system into a first order ODE in standard form. Let $z(x) = \begin{pmatrix} z_1(x) \\ z_2(x) \end{pmatrix} = \begin{pmatrix} y_\beta(x) \\ y'_\beta(x) \end{pmatrix}$.

This gives us the following first order system:

$$z'(x) = \begin{pmatrix} y'_\beta(x) \\ y''_\beta(x) \end{pmatrix} = \begin{pmatrix} z_2(x) \\ -5z_2(x) - 4z_1(x) \end{pmatrix},$$

$$z(0) = \begin{pmatrix} 1 \\ \beta \end{pmatrix}$$

The next step is to define the function $F(\beta) = y_\beta(1) - y(1) = z_1(1) - 2$ and solve $F(\beta) = 0$(we use the Secant method to do this). Note that evaluating the function $F()$ at β is equivalent to computing $y_\beta(1)$ or $z_1(1)$ using $y'_\beta(0) = \beta$ as an initial condition with a suitable time-stepping scheme(we use Forward Euler for this).

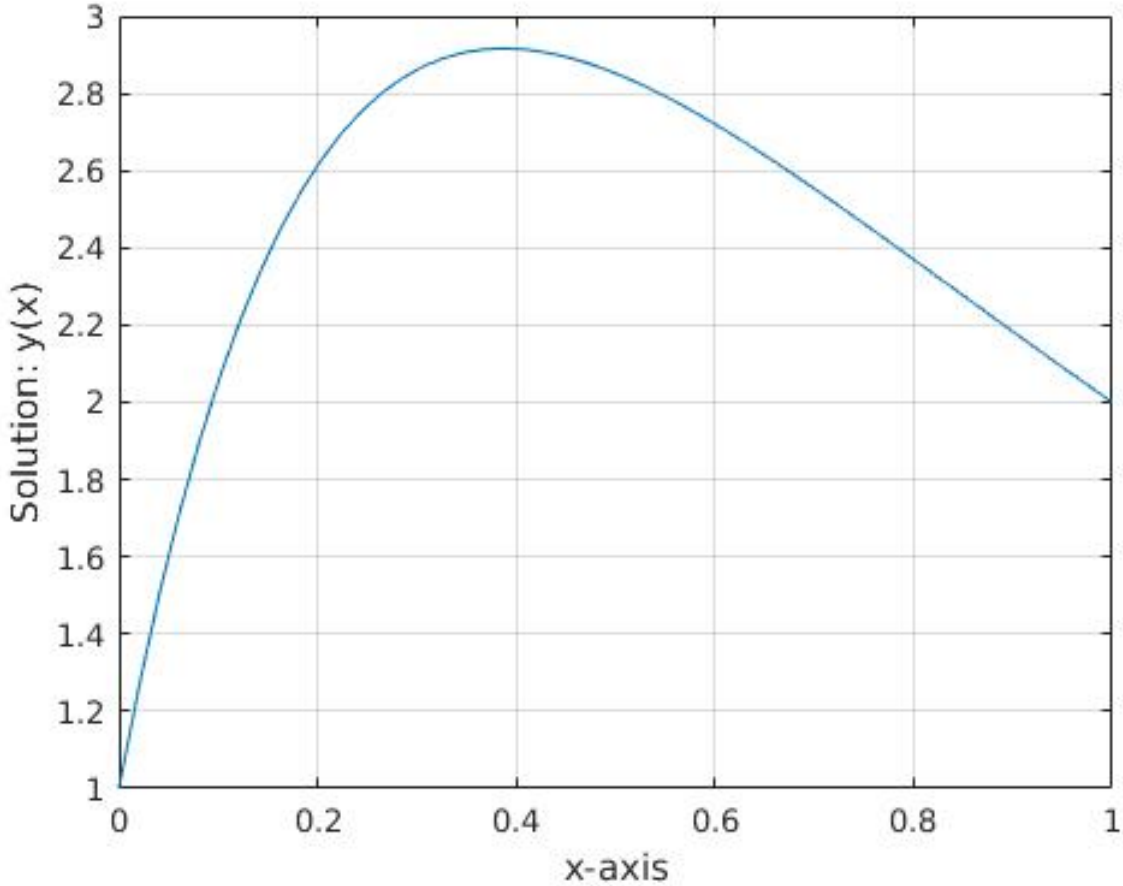
The secant method can be formulated as follows:

Start with initial guesses β_0, β_1.

Until convergence:

$$\beta_{k+1} = \beta_k - (y_{\beta_k} - 2)(\beta_k - \beta_{k-1})/(y_{\beta_k} - y_{\beta_{k-1}})$$

Using this method, we converge to $\beta = 13.0750$ and we get the following plot for our solution:



A screenshot of the MATLAB code to solve this problem is attached below:

```
%Solve a second order differential equation using the Shooting method
N = 50;
x = linspace(0,1,N);

beta0 = 0;
beta1 = 1;

tol = 1e-8;

yb = 2;
%Secant method to solve F(beta) = y_beta(1)-yb = 0
iters = 0;
while(abs(beta0-beta1)> tol)
    f0 = feuler(beta0,N);
    f1 = feuler(beta1,N);

    %beta_(k+1) = beta_k - F(beta_k)*(beta_(k-1)-beta_k)/(F(beta_(k-1))-F
    beta2 = beta1-(f1(1,end)-yb)*(beta0-beta1)/(f0(1,end)-f1(1,end));

    beta0 = beta1;
    beta1 = beta2;
    iters = iters+1;
end

%Plot solution
plot(x,f1(1,:));
grid on;
xlabel('x-axis')
ylabel('Solution: y(x)')

%Forward Euler: Takes as input the inital condition beta and return
%y_beta(1)
function z = feuler(beta,N)
    z = zeros(2,N);
    z(:,1) = [1 beta]';

    h = 1/(N-1);
    for i=2:N
       z(:,i) = z(:,i-1)+h*[z(2,i-1) -5*z(2,i-1)-4*z(1,i-1)]';
    end
end
```

Solution 2-4-22

In order to use the matrix method, we must use finite difference schemes to discretize our differential equation. Assume that we have divided the domain $[0, 1]$ into N subintervals. This means that we have $N + 1$ grid points $x_{i,i=0,\ldots,N}$ such that x_0 = 0 and x_N = 1. Also, the grid spacing $h = 1/N$.

We want to find the approximation to the solution at precisely these points, that is,

$$y_i \approx y(x_i), \quad i = 0, \ldots, N.$$

Since we have the Dirichlet boundary condition $y(x_0) = y(0) = 1$, we can set $y_0 = 1$ and hence it is not an unknown. This means that we have N unknowns are $y_1, \ldots, y_N$. To solve for these unknowns, we need N equations, which will come from the finite difference discretization.

At i=1, using central difference to approximate y'' and y' in the differential equation, we have,

$$\frac{y_0 - 2y_1 + y_2}{h^2} + 5\frac{y_2 - y_0}{2h} + 4y_1 = 0$$

$$\implies (-2 + 4h^2)y_1 + (1 + 5h/2)y_2 = (5h/2 - 1)y_0$$

At $i = 2, \ldots, N - 1$ which correspond to the other interior points,

$$\frac{y_{i-1} - 2y_i + y_{i+1}}{h^2} + 5\frac{y_{i+1} - y_{i-1}}{2h} + 4y_i = 0$$

$$\implies (1 - 5h/2)y_{i-1} + (-2 + 4h^2)y_i + (1 + 5h/2)y_{i-1} = 0.$$

At i = N which corresponds to the left boundary, we apply the Neumann boundary condition,

$$(y_N - y_{N-1})/h = 0$$

$$\implies -y_{N-1} + y_N = 0$$

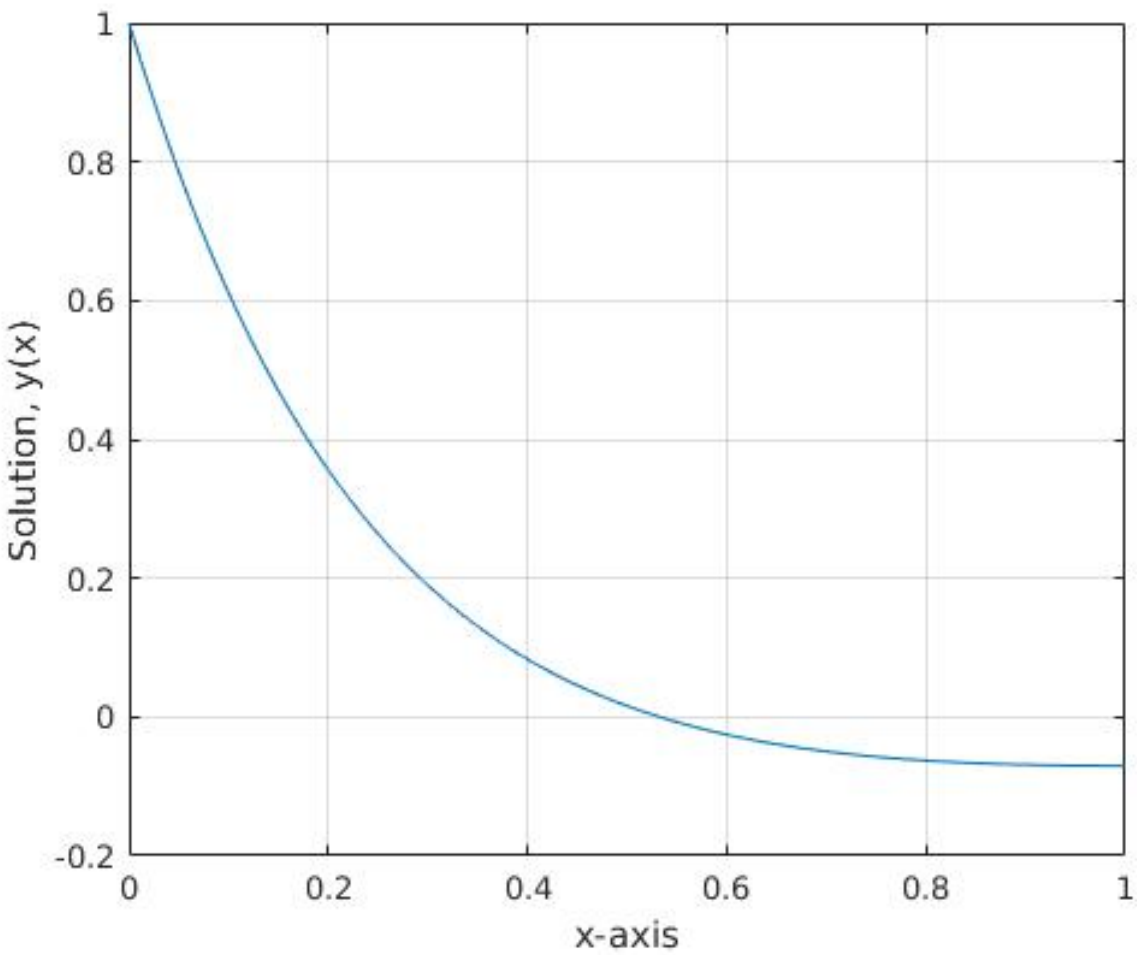
If we let $y = \begin{pmatrix} y_1 \\ \vdots \\ \vdots \\ y_N \end{pmatrix}$ and assemble all the equations derived above, we end up with the linear equation system $Ay = b$,

where $b = \begin{pmatrix} (5h/2-1)y_0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$

and $A = \begin{bmatrix} -2+4h^2 & 1+5h/2 & 0 & \dots & \dots \\ 1-5h/2 & -2+4h^2 & 1+5h/2 & 0 & \dots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & 1-5h/2 & -2+4h^2 & 1+5h/2 \\ 0 & \dots & \dots & -1 & 1 \end{bmatrix}$

Hence, we can obtain the solution by using **y = A\b** in MATLAB.

The plot of the solution looks as follows:



As we can see, the solution is very different from the solution obtained from the previous problem as the Neumann boundary condition has to be satisfied at x=1, which makes all the

difference.

We attach a screenshot of the the MATLAB code used to solve the problem:

```
% Solve differntial equation with Neumann boundary condition
% at x = 1
y0 = 1;
N = 100;
x = linspace(0,1,N+1);
h = 1/(N);

%Create the tridiagonal matrix A
A = zeros(N);
e = ones(N,1);
A = spdiags([(1-5*h/2)*e (-2+4*h^2)*e (1+5*h/2)*e],-1:1,N,N);
A(N,N-1) = -1; A(N,N) = 1; %Applying the Neumann boundary condition

%Create RHS b
%All elements of b except the first one are 0
b = zeros(N,1);
b(1) = (5*h/2-1)*y0;

%Solve
y = A\b;

%Plot
plot(x',[y0;y]);
grid on;
xlabel('x-axis');
ylabel('Solution, y(x)');
```

Solution 3-4-05

Vi inför en vektor:

$$\mathbf{y}(t) = \begin{bmatrix} y_1(t) \\ y_2(t) \end{bmatrix} = \begin{bmatrix} k(t) \\ k'(t) \end{bmatrix}$$

Vi deriverar vektorn

$$\mathbf{y}'(t) = \begin{bmatrix} k'(t) \\ k''(t) \end{bmatrix}$$

och använder ekvationen för $k''(t)$. Det betyder att

$$\mathbf{y}'(t) = \begin{bmatrix} k'(t) \\ -15k'(t) + 2k(t) + \sin(t^3) \end{bmatrix}$$

För att kunna skriva problemet på formen

$$\mathbf{y}'(t) = \mathbf{f}(\mathbf{y}(t)),$$

använder vi $k(t) = y_1(t)$ och $k'(t) = y_2(t)$, och får

$$\mathbf{y}'(t) = \begin{bmatrix} y_2(t) \\ -15y_2(t) + 2y_1(t) + \sin(t^3) \end{bmatrix}$$

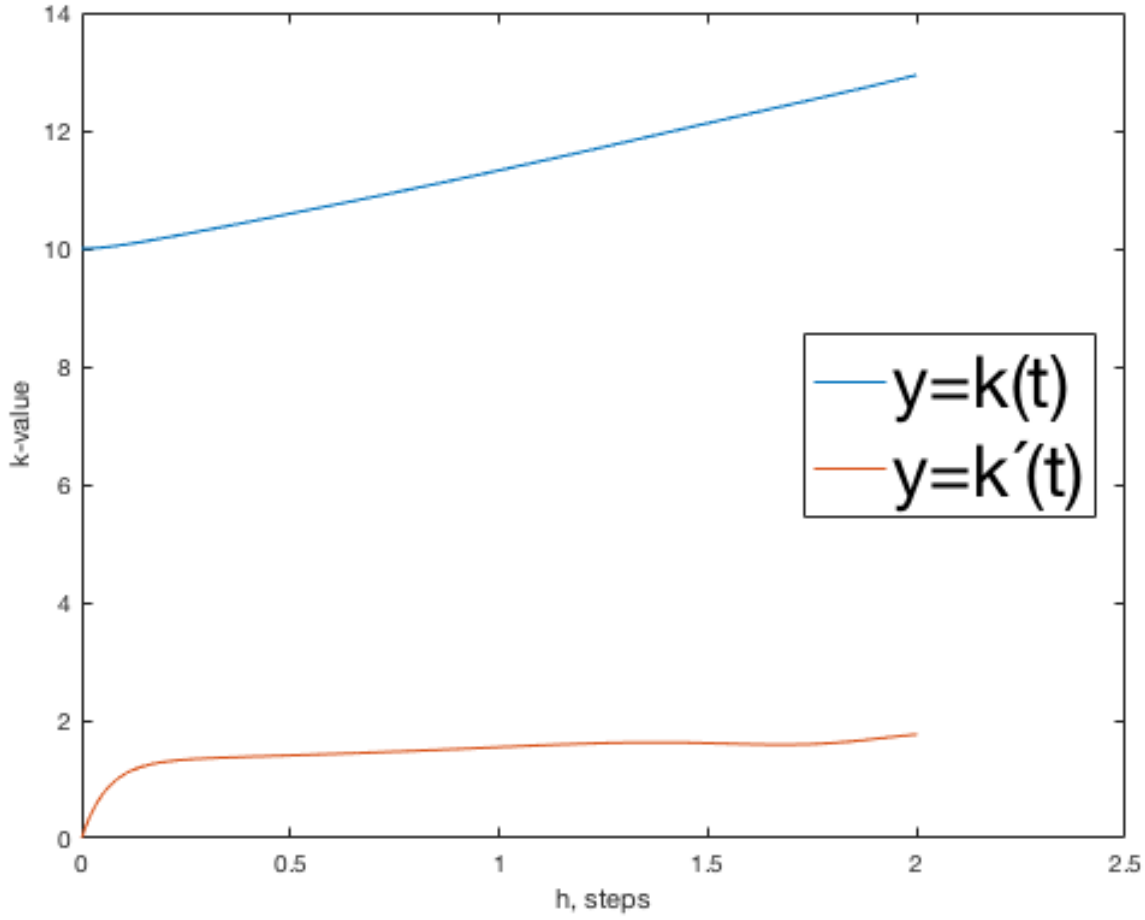
Med andra ord har vi ODE-systemet $\mathbf{y}'(t) = \mathbf{f}(\mathbf{y}(t))$ där

$$\mathbf{f}(\mathbf{y}) = \begin{bmatrix} y_2(t) \\ -15y_2(t) + 2y_1(t) + \sin(t^3) \end{bmatrix}$$

Moderator: Korrekt a-uppgift.

Solution 3-4-06

Här är de plottade funktionerna.



Vi fick detta genom att spara de värden som varje varv av Euler-framåt ger. Nu när vi bara går till 2 med en relativt liten steglängd så kan vi anta att approximationen är bra. Men om vi vill gå till större värden t.ex. $t = 50$, så kommer vi behöva ändå mindre stegvärden (finare diskretisering). Detta beror på att Euler-framåt ger fel för varje approximation, vilket kommer att ackumuleras och tillslut ge en alldeles för felaktig approximation. En finare diskretisering kommer att minimera varvens fel och på så sätt minimera det kumulativa felet.

Moderator: Flera saker ser lite knasiga ut. Det ser inte ut som att du har tagit steglängd h=0.01. Vilken funktion är $k(t)$ och vilken är $k'(t)$? Programkod, tack.

EDIT: Några slarvfel, glömde ett ´ och fel steglängd. Här är koden, den röda texten är det redigerade:

```
clc;
f = @(t, y) [y(2); -15*y(2)+2*y(1)+sin(t^3)];
% init = [y1; y2];
initial = [10; 0];
% This is a function that performes Euler forward.
[interval, k, dk] = euler(f, initial, 0.01, 2, 0);

K = plot(interval, k);
hold on;
DK = plot(interval, dk);
xlabel('h, steps')
ylabel('k-value')
lg = legend([K, DK], 'y=k(t)', 'y=k´(t)');
lg.FontSize =30;
lg.Location = 'Best';
```

Moderator: Bra. Det stämmer att Euler har en accumulering av fel. I detta fall har vi dessutom termen $\sin(t^3)$ som kommer att oscillera snabbare och snabber ju större t. Vid $t = 50$ kommer inte $h = 0.01$ vara tillräckligt litet, så metoden kan inte beakta oscillationerna.

Solution 3-4-07

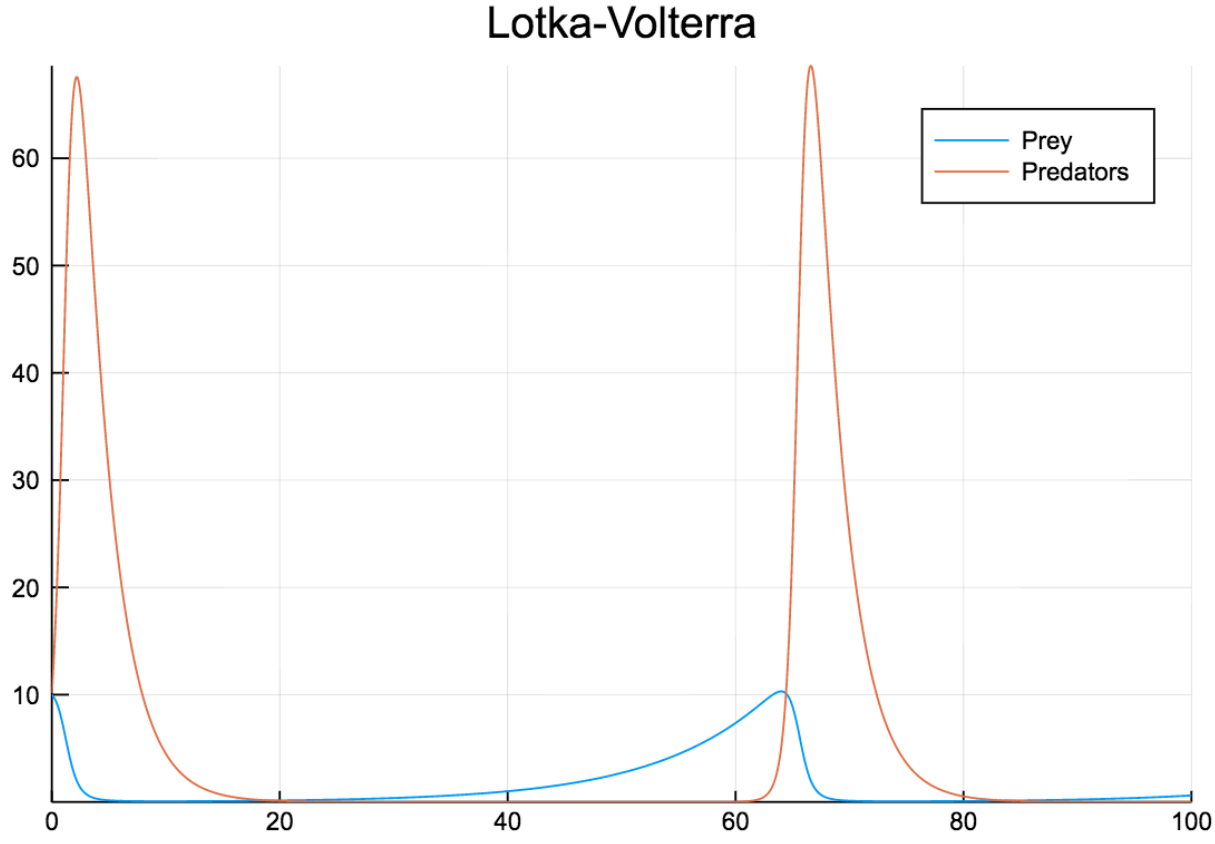
Vi ska lösa ett ODE-system i två variabler, Eulers metod går att tillämpa radvis:

$$\boldsymbol{y}_{i+1} = \boldsymbol{y}_i + h * \boldsymbol{f}(t_i, \boldsymbol{y}_i)$$

Löser med följande Julia-kod:

```
# Plotpaket
using Plots
gr()

# Konstanter
α = 0.1; β = 0.02; γ = 0.4; δ = 0.2

# Vårt system
f(t, xy) = [α*xy[1] - β*xy[1]*xy[2];
δ*xy[1]*xy[2] - γ*xy[2]]

# Förbereder Euler
h = 0.01
ts = 0:h:100
xy = [10.; 10.]
results = Array{Float64, 2}(length(ts), length(xy))

# Kör Euler och sparar resultat
it = 1
for t in ts
xy .+= h*f(t, xy)
results[it,:] .= xy
it+=1
end

# Plottar
plt = plot(title = "Lotka-Volterra")
plot!(ts, results[:,1], label = "Prey")
plot!(ts, results[:,2], label = "Predators")
display(plt)
```

Får följande graf:



Moderator: Utmärkt!

Solution 3-4-10

Vi introducerar följande notation

$$\mathbf{y}(t) = \begin{bmatrix} y_1(t) \\ y_2(t) \end{bmatrix} = \begin{bmatrix} y'(t) \\ y''(t) \end{bmatrix}$$

Vi kan då skriva om diffekvationen till följande system:

$$\begin{cases} y_1'(t) = y_2(t) \\ y_2'(t) = -9.8 + y_1(t) \end{cases}$$

Vi ska nu använda inskjutningsmetoden för att hitta ett värde för $y'(0) = s$ (som vi kallar för s). I Sauer använder de följande kod för att lösa ett liknande system (notera att vi använder oss av ode45).

```
%% Vi definierar vår funktion som löser ett system med ode45
function z = F(s)
a=0;    % Första t-värdet
b=1;    % Andra t-värdet
ya=1.5; % Första y-värdet
yb=0;   % Andra y-värdet
ydot=@(t,y) [y(2);-9.8+y(1)];   % Diffsystemet
[t,y]=ode45(ydot,[a,b],[ya,s]); % Lös varje system med ode45
z=y(end,1)-yb; % vektorn som innehåller lösningen
end
```

Den här koden löser systemet ovan för olika begynnelsevillkor, givet $y(0) = 1.5$ och $y'(0) = s$, för något givet s. Vi ska nu använda sekantmetoden för att hitta ett värde som gör att funktionen ovan blir 0 d.v.s vi minimerar skillnaden. Vi använder oss då av följande kod.

```
%% Vi letar efter punkten där F(s) = 0 m.h.a sekantmetoden
h = inf;
y1 = -1;
y2 = 8;
epsilon = 10e-10;
while(abs(h) > epsilon)
    h = F(y1)*(y1-y2)/(F(y1)-F(y2));
    temp = x1;
    y1 = y1-h
    y2 = temp
end
```

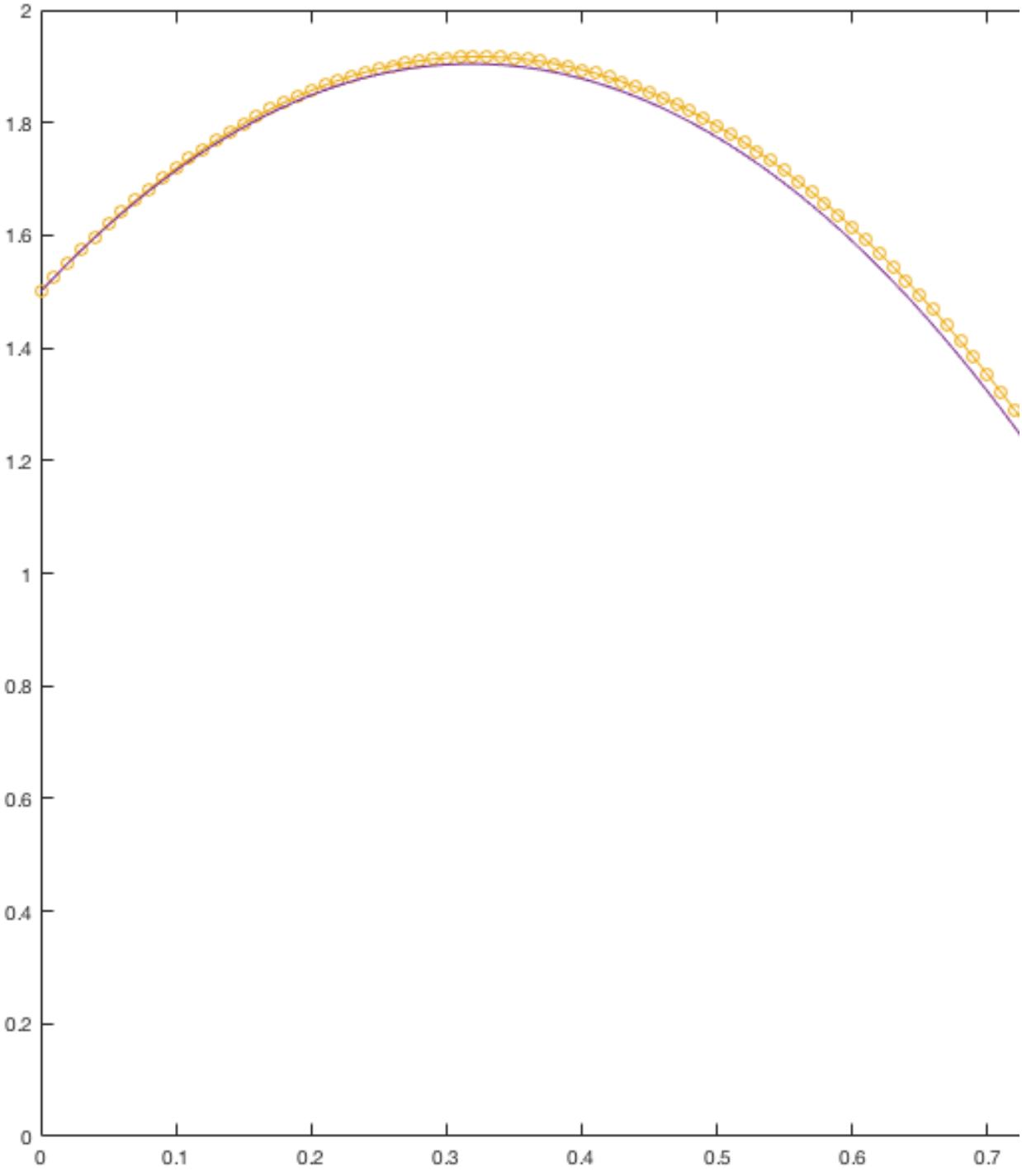
Vi finner att värdet som gör att $F(s) = 0$ är ungefär $s = 2.5592$. Därmed är våra begynnelsevillkor

$$\begin{cases} y(0) = 1.5 \\ y'(0) = 2.5592 \end{cases}$$

och vi kan äntligen lösa systemet med vanlig Euler-framåt.

```
%% M.h.a lösningen ovan kan vi formulera vår BVP

% Funktionsvektorn som innehåller derivatan
f = @(t,y) [y(2);-9.8+y(1)];

% Den riktiga lösningen, som har hittats med WolframAlpha
g = @(t) 9.8 - 5.4296*exp(-t)-2.8704*exp(t);

h = 0.01;
tv = [0:h:1];
y0 = 1.5;
y0_p = 2.5592;
y = [y0 ; y0_p];
Y = [];
i = 0;
for t=tv
   i = i+1;
   Y(:,i) = y;
   y = y+h*f(t,y);
   t = t+h;
end

zv = Y';
zv = zv(:,1);
gv = arrayfun(g, tv);
plot(tv,zv, '-o'); hold on;
plot(tv,gv);
```

Gjorde lite fulkodning med några matriser men den är ganska sent nu och jag orkar inte hitta optimala lösningen. Hur som helst, koden ovan ger oss en lösning som man kan se i bilden nedan. Jag har även ritat ut den riktiga lösningen.



Moderator: Ok. Bra!

Solution 3-4-14

Om vi skriver om ekvationen till $y''(t) = -2y(t) - \sin(t)$ kan vi enkelt omvandla den till standardform:

$$f(t) = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} = (\text{för vår ekvation}) = \begin{bmatrix} f_2 \\ -2f_1 - sin(t) \end{bmatrix}, \quad f(a) = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

Moderator: Det är i princip rätt (räknas som bonus). Det är viktigt att skilja på vad som är funktioner av vad och din användning av f gör det lite förvirrande. Det här är det vanliga sättet att skriva på: Vi definierar

$$\mathbf{z}(t) = \begin{bmatrix} y(t) \\ y'(t) \end{bmatrix} = \begin{bmatrix} z_1(t) \\ z_2(t) \end{bmatrix}$$

När vi formulerar på standarform vill vi hitta funktionen $\mathbf{f}$ så att

$$\mathbf{z}'(t) = \mathbf{f}(t, \mathbf{z}(t)). \qquad (*)$$

Vi deriverar definitionen av $\mathbf{z}(t)$:

$$\mathbf{z}'(t) = \begin{bmatrix} y'(t) \\ y''(t) \end{bmatrix}$$

och använder relationerna $z_1(t) = y(t)$ och $z_2(t) = y'(t)$:

$$\mathbf{z}'(t) = \begin{bmatrix} z_2(t) \\ -2z_1(t) - \sin(t) \end{bmatrix}$$

Därmed är (*) uppfylld där

$$\mathbf{f}(t, \mathbf{z}) = \begin{bmatrix} z_2 \\ -2z_1 - \sin(t) \end{bmatrix}$$

(Obs: f är en funktion av både t och z.)

Solution 3-4-16

Härledning av stegningsformeln för eulerbakåt som används:

$y' = -y^2 \quad y_{k+1} = y_k + h \cdot y'_{k+1} = y_k + h \cdot -y^2_{k+1}$

$h \cdot y^2_{k+1} + y_{k+1} - y_k = 0 \quad y^2_{k+1} + \frac{y_{k+1}}{h} - \frac{y_k}{h} = 0$

PQ-formeln: $y_{k+1} = -\frac{1}{2h} \pm \sqrt{\frac{1}{4h^2} + \frac{y_k}{h}}$

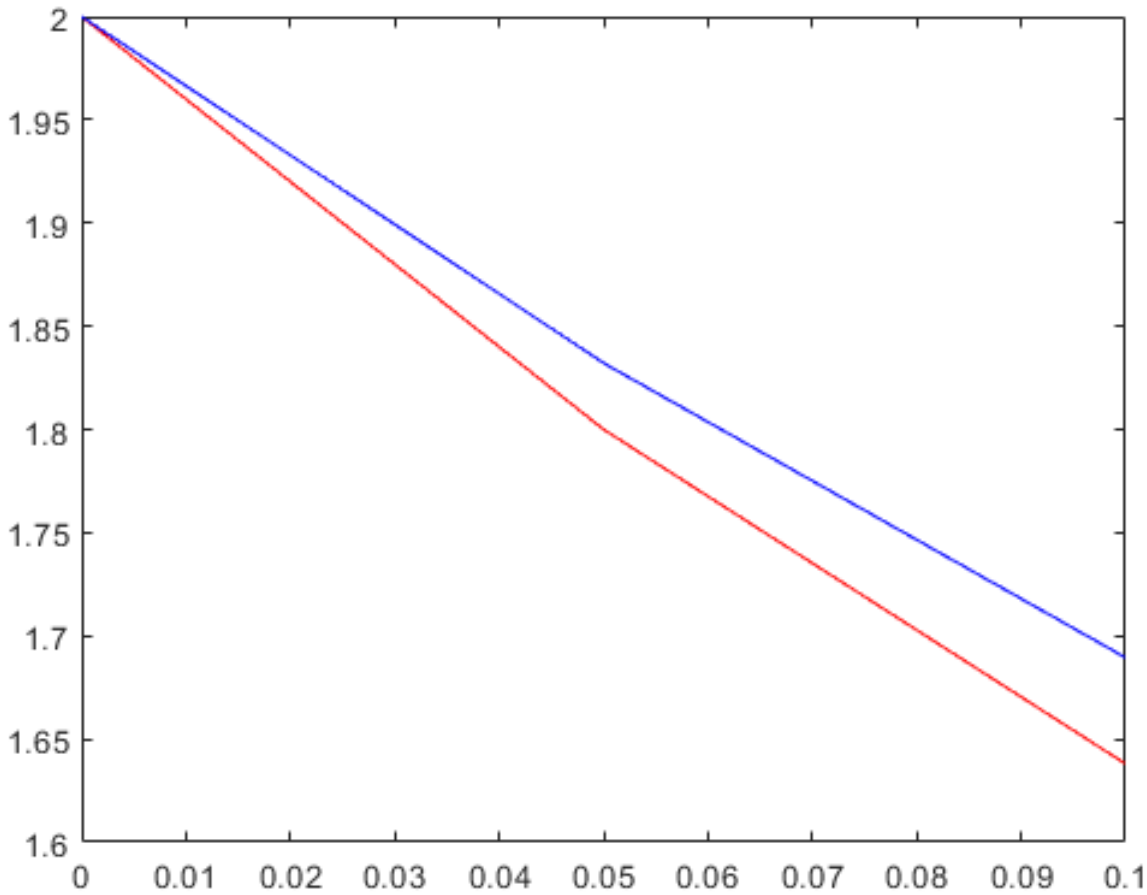
I koden tar man + rotuttrycket eftersom att man ska välja det tecken som gör så att värdet man får från ett steg från startvärdet är så nära som möjligt.

Koden som körs:

```
f=@(t,y) -1*y^2;

h=0.05;
y=2;
t=0;
yv=[2];
tv=[0];
for i=0:h:0.05
y=y+h*f(t,y);
t=t+h
yv=[yv;y];
tv=[tv;t];
end

plot(tv,yv,'r')
hold on;

% Bakåt
y=2;
t=0;
yv=[2];
tv=[0];
for i=0:h:0.05
y=(-1/(2*h))+sqrt((1/(4*h^2))+ y/h)
t=t+h;
yv=[yv;y];
tv=[tv;t];
end
```

```
plot(tv,yv,'b')
hold on;
```

Figur som erhålls:



Resultatet är att de inte skiljer sig så mycket.

Solution 3-4-20

A. Det lokala felet är det trunkeringsfel som uppkommer i ett visst steg av Eulers metod.

B. Det globala felet är det ackumulerade trunkeringsfelet i ett visst steg av Eulers metod.

C. Nej. Det lokala felet har noggrannhetsordning $O(h^2)$ medan det globala har noggrannhetsordning $O(h)$.

Solution 4-4-10

$y'(t) = g(t) \cdot y(t) = f(t, y(t))$

Med Euler bakåt blir detta

$y_{k+1} = y_k + h \cdot f(t_{k+1}, y_{k+1}) = y_k + h \cdot (g(t_{k+1}) \cdot y_{k+1})$

Detta kan skrivas om som

$y_{k+1} = \frac{y_k}{1 - h \cdot g(t_{k+1})}$

Eftersom att $t_0 = 0$ är $t_{k+1} = (k+1)\,h$, så ekvationen kan skrivas om till

$y_{k+1} = \frac{y_k}{1 - h \cdot g((k+1) \cdot h)}$

$y(0) = 1$, så en explicit formel kan skrivas som

$y_1 = \frac{y(0)}{1 - h \cdot g(h)}$

$y_2 = \frac{y_1}{1 - h \cdot g(2h)}$

...

$y_{k+1} = \frac{y_k}{1 - h \cdot g((k+1) \cdot h)}$

Moderator: Korrekt!

Solution 4-4-13

Vi löser ut A då $y_0 = 1$.

$$A = \frac{y_2 - 2y_1 + 1}{h^2} = B + \frac{1}{h^2}$$

B kan då lösas ut

$$B = \frac{y_2 - 2y_1}{h^2}$$

$$y''(x_2) = \frac{y_1 - 2y_2 + y_3}{h^2}$$

Vilket ger att $C = h^2$.

$$y''(x_4) = \frac{y_5 - 2y_4 + y_3}{h^2} = D + \frac{1}{h^2}$$

Där vi kan lösa ut $D = -y_4 + y_3$ när $x_5 = 1$.

Vidare får vi

$$\frac{y_1 - 2y_2 + y_3}{h^2} = y_2 + 100\sin(10x_2)$$

$$\frac{y_2 - 2y_3 + y_4}{h^2} = y_3 + 100\sin(10x_3)$$

$$D? + \frac{1}{h^2} = y_4 + 100\sin(10x_4)$$

Solution 4-4-14

Solution 4-4-16

Av differentialekvationen fås att $f(t, y(t)) = 10t + y(t)^2$ och att $y_0 = 10$, $h = 0.1$

Enligt Euler framåt så är $y_{i+1} = y_i + h \cdot f(t_i,\ y_i)$.

Två steg av Euler framåt är således följande:

$y_1 = y_0 + h \cdot f(t_0,\ y_0) \ = \ 10 \ + \ 0.1 \cdot \left(0 + 10^2\right) = 20$

$y_2 = y_1 + h \cdot f(t_1,\ y_1) = 20 + 0.1 \cdot \left(1 + 20^2\right) = 60.1$

Solution 4-4-18

Jag löste följande uppgift för hand enligt följande:

$$y''(t) = y(t) + x(t)^2$$

$$x'(t) = 10x(t) + \sin(y(t))$$

$$y(0) = 1,\ y'(0) = 10,\ x(0) = -1$$

används vid beräkningar

$$q_1 = y(t)$$

$$q_1' = y'(t) = q_2$$

$$q_2' = y''(t)$$

$$q_3 = x(t)$$

$$q_3' = x'(t)$$

$$y''(t) = q_1 + (q_3)^2$$

$$x'(t) = 10 \cdot q_3 + \sin(q_1)$$

$$\bar{q} = \begin{bmatrix} q_1' \\ q_2' \\ q_3' \end{bmatrix} = \begin{bmatrix} q_2 \\ q_1 + (q_3)^2 \\ 10 q_3 + \sin(q_1) \end{bmatrix}$$