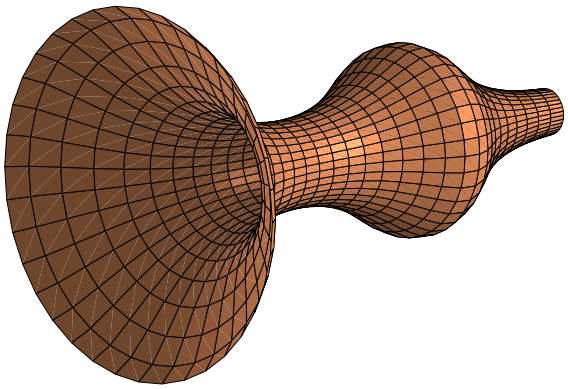
NUMERISKA ALGORITMER med MATLAB

Gerd Eriksson
NADA, KTH

1. **Idéer, begrepp och ekvationssystem**
2. **Minstakvadratmetoden**
3. **Interpolation**
4. **Bézierkurvor**
5. **Numerisk integration**
6. **Ekvationer och ekvationssystem**
7. **Maximering och minimering**
8. **Differentialekvationer**

Integrationsuppgift: Hur stor volym har luren?



Juni 2002

Innehåll

1 IDEER, BEGREPP och EKVATIONSSYSTEM **1**
- 1.1 Inledning . . . 1
- 1.2 Felbegrepp och felfortplantning . . . 2
- 1.3 Differenskvot som derivataapproximation . . . 4
  - 1.3.1 Framåtdifferenskvot och centraldifferenskvot . . . 4
  - 1.3.2 Härledning av trunkeringsfelet . . . 5
  - 1.3.3 Differenskvotformel för andraderivatan . . . 6
- 1.4 Linjära ekvationssystem . . . 6
  - 1.4.1 Gausselimination och Matlabs ekvationssystemlösare . . . 6
  - 1.4.2 Antal operationer och tidsåtgång . . . 7
- 1.5 Matrisegenskaper . . . 8
- 1.6 Tridiagonala systemmatriser . . . 9
  - 1.6.1 Effektiv algoritm för lösning av tridiagonalt system . . . 9
  - 1.6.2 Asplunds metod för beräkning av inversen till symmetriska tridiagonala matriser . . . 10
- 1.7 Vektor- och matrisnormer . . . 12
- 1.8 Experimentell störningsräkning vid ekvationssystem . . . 12

2 MINSTAKVADRATMETODEN **14**
- 2.1 Överbestämda ekvationssystem . . . 14
- 2.2 Minstakvadratlösning med normalekvationerna . . . 15
- 2.3 Linjär modellanpassning som minstakvadratproblem . . . 16
- 2.4 Val av basfunktioner vid polynomanpassning . . . 18
- 2.5 Residualanalys . . . 19
- 2.6 Experimentell störningsräkning . . . 21
- 2.7 Praktisk statistik vid minstakvadratanpassning . . . 22

3 INTERPOLATION **24**
- 3.1 Interpolationspolynom som modell till rutschbana . . . 24
- 3.2 Polynom av hög grad genom alla punkter . . . 26
- 3.3 Styckvis interpolation . . . 28
  - 3.3.1 Hermiteinterpolation . . . 28
  - 3.3.2 Interpolation med kubiska splines . . . 30
- 3.4 Hackigheter i kurvan, symptom på illakonditionering . . . 34
- 3.5 Experimentell störningsanalys . . . 37
  - 3.5.1 Tillförlitlighet i polynomkoefficienterna . . . 37
  - 3.5.2 Tillförlitlighet i de beräknade polynomvärdena . . . 38
- 3.6 Tillämpning: Polynommultiplikation på smart sätt . . . 39
- 3.7 Flerdimensionell interpolation . . . 40
  - 3.7.1 Bilinjär interpolation . . . 40
  - 3.7.2 Kvadratisk–linjär interpolation genom sex punkter . . . 41
  - 3.7.3 Bikubisk interpolation — splineyta . . . 42
- 3.8 Richardsonextrapolation . . . 44

4 BÉZIERKURVOR **46**
4.1 Parameterkurvor . . . 46
4.2 Bézierkurvor . . . 46
4.2.1 Kvadratiska bézierkurvor . . . 46
4.2.2 Kubiska bézierkurvor . . . 47
4.2.3 Bézierkurvor med givna ändriktningar . . . 48
4.2.4 Exempel: Cosinuskurva approximerad av bézierkurva . . . 48
4.2.5 Areaberäkning för kvadratisk bézierkurva . . . 50
4.3 Styckvis interpolation med bézierkurvor . . . 50
4.3.1 Styckvisa bézierkurvor med kontinuerliga tangentriktningar . 50
4.3.2 Styckvisa bézierkurvor med kontinuerliga derivator . . . 52
4.3.3 Parametriska kubiska splinekurvor . . . 53
4.4 Icke-interpolerande B-splines . . . 53

5 NUMERISK INTEGRATION **57**
5.1 Allmänt . . . 57
5.2 Trapetsregeln med släktingar . . . 57
5.2.1 Trapetsregeln och Simpsons formel . . . 57
5.2.2 Matlabs quad-funktioner . . . 60
5.2.3 Periodiska integrander – lös med trapetsregeln . . . 61
5.2.4 Integration av hermiteinterpolationspolynom och splines . . . 62
5.2.5 Integration av kubiska bézierkurvor . . . 63
5.3 Gausskvadratur – väl valda punkter . . . 64
5.4 Gauss-Kronrods metod – 15 smart valda punkter . . . 65
5.5 Integraler som fordrar förbehandling . . . 68
5.5.1 Knepig integral . . . 68
5.6 Dubbelintegraler . . . 69
5.7 APPENDIX, Parametrarna i gausskvadratur . . . 71

6 EKVATIONER OCH EKVATIONSSYSTEM **72**
6.1 Allmänt . . . 72
6.2 Polynomekvationer . . . 72
6.3 Intervallhalveringsmetoden . . . 72
6.4 Sekantmetoden . . . 73
6.5 Newton-Raphsons metod . . . 74
6.5.1 Exempel: Skärning mellan funktionskurva och rät linje . . . 74
6.5.2 Härledning av Newton-Raphsons kvadratiska konvergens . . . 76
6.6 Fixpunktsiteration . . . 77
6.7 Iterativ lösning av glesa linjära ekvationssystem . . . 78
6.8 Lösning av ickelinjära ekvationssystem . . . 79
6.9 Newtons metod för ickelinjära ekvationssystem . . . 81
6.9.1 Algoritm för Newtons metod för ett ickelinjärt system . . . 82
6.9.2 Exempel: Interpolerande sinuskurva . . . 82
6.9.3 Approximation till jacobianen . . . 84

6.10 Ickelinjär modellanpassning 84
6.10.1 Exempel: Anpassande sinuskurva 84
6.10.2 Gauss-Newtons metod 86
6.10.3 Exempel: Ellipsanpassning till givna punkter 87

7 MAXIMERING OCH MINIMERING **89**
7.1 Maximering/minimering av unimodala funktioner 89
7.1.1 Gyllenesnittetsökning 89
7.1.2 Fibonaccisökning 91
7.2 Maximering av funktioner i flera variabler 92
7.2.1 Maximering med användning av Newtons metod 92
7.2.2 Maximering utan beräkning av jacobianmatris 94

8 DIFFERENTIALEKVATIONER **95**
8.1 Allmänt om ODE 95
8.2 Eulers metod och Runge-Kuttas metod RK4 95
8.3 Rävar och kaniner – ett differentialekvationssystem 99
8.4 Högre ordningens differentialekvationer 100
8.5 Pendel med stort utslag 101
8.6 Varnande exempel 103
8.6.1 Explicita och implicita metoder, instabilitet 104
8.7 Randvärdesproblem 105
8.7.1 Inskjutningsmetoden 105
8.7.2 Finitadifferensmetoden, FDM 106
8.7.3 Linjärt randvärdesproblem med en parameter 106
8.7.4 Ickelinjärt randvärdesproblem med FDM 109

1 IDEER, BEGREPP och EKVATIONSSYSTEM

1.1 Inledning

Numeriska beräkningar kommer in i alla tekniska tillämpningar och det gäller att angripa problemen effektivt och med tillförlitliga metoder. På engelska används begreppet *Scientific Computing* för denna allt viktigare beräkningsverksamhet. En modern bok som rekommenderas som kursbok i numeriska metoder och därefter som referenslitteratur vid ingenjörsmässigt beräkningsarbete har just detta namn. Det är *M.T.Heath, Scientific Computing – An Introductory Survey*. Andra upplagan utgiven 2002 ger också hela 435 referenser som lästips för den vetgirige!

Fastän man nu har tillgång till utmärkt programvara med numeriska algoritmer för den typ av problem som behandlas i våra numeriska grundkurser, är det nödvändigt att ha kännedom om vilka idéer och numeriska redskap som bygger upp algoritmerna. Man måste kunna bedöma om den valda algoritmen passar och är effektiv för det aktuella beräkningsproblemet. Om resultatet ser konstigt ut eller om hela körningen spårar ur, kan det bero på metodegenskaper som man bör känna till och helst kunna korrigera.

Att bedöma resultatets tillförlitlighet är också viktigt. Om datorn skriver ut resultatvärdet 375.6803, kan man då vara säker på att svaret har sju siffrors noggrannhet? Nej, troligen inte. En tillförlitlighetsanalys visar kanske att ett lämpligare svar är 375.7 eller i värsta fall att inte ens första siffran i det angivna sjusiffriga värdet är tillförlitlig.

Den här lilla boken *Numeriska algoritmer med Matlab* är tänkt att tjäna som komplement till en mer omfattande lärobok, till exempel den nämnda boken av Heath eller *Peter Pohl, Grunderna i numeriska metoder* (lämplig för Numeriska metoder gk1, 2D1210, men inte heltäckande för gk2, 2D1240).

Vissa metoder presenteras på samma sätt här som i läroboken, men ofta är infallsvinklarna lite olika, här läggs tonvikten vid datortillämpningar i Matlab. Lite nytt stoff finns här såsom residualanalys vid minstakvadratproblem, flerdimensionell interpolation och kurvdesign med bézierkurvor.

Övningarna i *Exempelsamling i Numeriska metoder* av Edsberg-Eriksson-Lindberg-Pohl visar att man har stor nytta av numeriska metoder i både vardagliga och tekniska tillämpningar. I varje kapitel här finns förslag till lämpliga övningar i exempelsamlingen.

1.2 Felbegrepp och felfortplantning

Beteckna med $\tilde{x}$ en approximation (ett närmevärde) till x. **Absoluta felet** definieras som $\tilde{x} - x$ och **relativa felet** $(\tilde{x} - x)/x$.

Absolutfel och relativfel kan vara negativa, t ex gäller att den tresiffriga approximationen $\tilde{x} = 3.14$ till $x = \pi = 3.14159265358979...$ har absolutfelet -0.0016 och relativfelet $-0.0016/\pi \approx -0.0005 = -0.5$ promille.

En positiv gräns för dessa fel är det som används mest i praktiken — i fallet ovan blir absoluta felets gräns 0.002 och relativfelets gräns 0.5 promille.

Indata till tekniska och fysikaliska problem är ofta mätvärden behäftade med viss osäkerhet. Då är det naturligt att ange värdet tillsammans med osäkerhetsbeloppet på formen $g = 9.81 \pm 0.005$.

Om absolutfelets belopp i ett närmevärde är högst $0.5 \cdot 10^{-d}$ säger man att talet har d **korrekta decimaler**. Antalet **korrekta siffror** i talet är alla siffror från och med första ickenolla till och med den sista korrekta decimalen. Talen 6543.21 ± 0.005 och $0.00654321 \pm 0.5 \cdot 10^{-8}$ har båda sex korrekta siffror men två respektive åtta decimaler.

Antalet korrekta siffror utgör ett mått på relativfelets storlek. Om antalet korrekta siffror s, så gäller att relativfelet har storleksordningen 10^{-s}.
Två *korrekta siffror* motsvarar ungefär relativa felgränsen 10^{-2} alltså 1%. Beroende på första siffran kan felet variera mellan en halv och fem procent.

Exempel: Studera följande tal givna med två korrekta siffror, $\tilde{x}_1 = 1.1$, $\tilde{x}_2 = 5.5$ och $\tilde{x}_3 = 9.9$, absolutfelgränsen är som synes 0.05 i vart och ett. Relativfelen skattas i tur och ordning till: $0.05/1.1 \approx 0.05 = 5\%$, $0.05/5.5 \approx 0.01 = 1\%$ och $0.05/9.9 \approx 0.005 = 0.5\%$.

Felkällor: Fel i resultatet av en numerisk algoritm kan bero på flera orsaker:

- osäkerhet i indata som fortplantas till utdata
- avrundningsfel under beräkningarna på grund av datorns begränsade (ofta sextonsiffriga) precision
- metodfel eller trunkeringsfel på grund av approximationer i algoritmen.

Studera och lös **Ex 1.1** *och* **Ex 8.1** *i exempelsamlingen.*

Regler för felfortplantning: Vid addition och subtraktion av närmevärden adderas absoluta felgränserna; vid multiplikation och division adderas relativa felgränserna.

Exempel: $x = 48.51 \pm 0.005$, $y = 48.40 \pm 0.005$, båda har relativfel på 10^{-4}.
Ange absoluta och relativa felgränser för $z = x + y = 96.91$ och $w = x - y = 0.11$.
Absolutfelet kan för både z och w uppgå till 0.01. Relativfelet för z beräknas till $0.01/96.91 \approx 1 \cdot 10^{-4}$, medan det för w blir $0.01/0.11 \approx 0.1 = 10\%$. Relativfelen har alltså helt olika storleksordning.

Kondition: Maximala relativfelsförstoringen, alltså kvoten mellan relativfelet i utdata och relativfelet i indata, kallas *konditionstalet*. Ett högt konditionstal är inte bra, det innebär en stor förlust i antal korrekta siffror på vägen från indata till utdata. Ett konditionstal på 10^5 medför förlust på fem siffror, dvs om indata har sju siffrors precision har resultatet inte mer än två korrekta siffror.
För exemplet gäller att konditionstalet vid additionen av x och y blir 1 (samma storlek på relativfelet i indata och utdata), medan differensberäkningen i detta fall har konditionstalet $0.1/10^{-4} = 1000$.
I exemplet ovan subtraherades två nästan lika stora tal (behäftade med fel) med stor relativ noggrannhetsförlust som följd. Det vi råkade ut för kallas *kancellation*. I många beräkningsprocesser uppstår sådana kancellationer, men genom listig omformning av algoritmen kan de ofta undvikas.

Experimentell störningsräkning är ett naturligt tillvägagångssätt när man vill bedöma tillförlitligheten i ett resultat och få en uppfattning om hur känsliga utdata är för osäkerhet i givna indata. Det kommer att tillämpas i olika beräkningsalgoritmer (närmast i avsnitt 1.7).

Studera och pröva på att lösa exempelsamlingens **Ex 8.2–8.8**. *De är trevliga och allmänbildande tillämpningar på experimentell störningsräkning. Facit finns till alla exemplen.*

Teoretiska felfortplantningsformeln: Om sambandet mellan indata och utdata kan skrivas som en explicit (och deriverbar) funktion $y = f(x_1, \ldots, x_n)$, så gäller vid små indatastörningar (och om $\frac{\partial f}{\partial x_i}(\tilde{\mathbf{x}}) \neq 0$, $\tilde{\mathbf{x}} = (\tilde{x}_1, \ldots, \tilde{x}_n)$) följande felfortplantningsformel:

$$|\tilde{y} - y| \approx \left|\frac{\partial f}{\partial x_1}(\tilde{\mathbf{x}})\right| |\tilde{x}_1 - x_1| + \left|\frac{\partial f}{\partial x_2}(\tilde{\mathbf{x}})\right| |\tilde{x}_2 - x_2| + \ldots + \left|\frac{\partial f}{\partial x_n}(\tilde{\mathbf{x}})\right| |\tilde{x}_n - x_n|.$$

Enklast och kanske vanligast är den i fallet en variabel:

$$|\tilde{y} - y| \approx |f'(\tilde{x})|\, |\tilde{x} - x|.$$

Exempel: Om $y = \log x$ så gäller att gränsen för absolutfelet i y är lika med gränsen för relativfelet i x. Derivera: $y' = 1/x$; nu erhålls direkt $|\tilde{y} - y| = |\tilde{x} - x|/|x|$, vilket definitionsmässigt är relativfelets gräns.

1.3 Differenskvot som derivataapproximation

1.3.1 Framåtdifferenskvot och centraldifferenskvot

Derivatan av en funktion $f(x)$ definieras som

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}.$$

Om h är litet bör uttrycket

$$f'(a) \approx \Delta_h = \frac{f(a+h) - f(a)}{h} \tag{1}$$

vara en god approximation till derivatan för $x = a$. Uttrycket kallas framåtdifferenskvot.

Låt oss ta ett exempel med $f(x) = e^x$ och $a = 1$. Då gäller ju $f'(a) = e^a = e^1 = 2.718281828...$

Tabellen visar resultatet då vi på en räknedosa med åttasiffrig precision tillämpar differenskvotformeln Δ_h för h-värdena 1, 0.1, 0.001, 0.0001, ...

h	$(f(1+h) - f(1))/h$
1	4.6707743
0.1	2.8588418
0.01	2.7319190
0.001	2.7196400
0.0001	2.7184000
0.00001	2.7200000
0.000001	2.7000000
0.0000001	3.0000000
0.00000001	0.0000000

En jämförelse med det rätta värdet 2.7182818 visar att när h minskar så blir differenskvoten en allt bättre approximation till derivatan, men efter värdet 2.7184000 (vid h=0.0001) försämras värdena mer och mer.

I teorin borde värdena förbättras hela tiden, men i praktiken spelar den begränsade precisionen hos datorn (i detta fall en åttasiffrig räknedosa) in. Vid små h blir det kancellation. Så till exempel får vi för $h = 10^{-7}$:

$f(1+h) - f(1) = e^{1.0000001} - e^1 = 2.7182821 - 2.7182818 = 0.0000003.$

Dividera med $h = 10^{-7}$ och differenskvotsresultatet blir 3 (tabellens näst understa värde). Genom subtraktionen av nästan lika stora tal förlorades alla siffror utom den sista trean, sju siffror kancellerades!

Slutsatsen av detta är att vid numerisk derivering med hjälp av framåtdifferenskvoten Δ_h måste h väljas med omsorg, inte för stort för då är differenskvoten en dålig modell för derivatan (trunkeringsfelet är stort) och inte för litet för då inträffar kancellation som förstör resultatet.

En bättre approximation till derivatan $f'(a)$ är centraldifferenskvoten

$$f'(a) \approx D_h = \frac{f(a+h) - f(a-h)}{2h} \tag{2}$$

Resultatet med D_h och avvikelsen från rätta derivatavärdet visas här.
Bästa approximation är nu 2.7183000 (med avvikelse på $1.82 \cdot 10^{-5}$, en faktor tio bättre än med Δ_h). Även med denna formel blir det kancellation då h är mycket litet.

h	D_h	$D_h - f'(1)$
1	3.1945280	0.4762463
0.1	2.7228145	0.0045327
0.01	2.7183250	0.0000432
0.001	2.7183000	0.0000182
0.0001	2.7185000	0.0002182
0.00001	2.7200000	0.0017182

1.3.2 Härledning av trunkeringsfelet

Med taylorutveckling kan man visa hur stort trunkeringsfelet är i de båda modellerna (1) och (2) för numerisk derivering. Taylorutvecklingen lyder:

$$f(a+h) = f(a) + h\,f'(a) + \frac{h^2}{2}f''(a) + \frac{h^3}{3!}f'''(a) + \frac{h^4}{4!}f^{(4)}(a) + \ldots \tag{3}$$

Flytta över termen $f(a)$ till vänster sida och dividera allt med h så erhålls

$$\frac{f(a+h) - f(a)}{h} = f'(a) + h\,\frac{f''(a)}{2} + h^2\,\frac{f'''(a)}{3!} + h^3\,\frac{f^{(4)}(a)}{4!} + \ldots$$

På vänster sida står framåtdifferenskvoten Δ_h och vi får alltså sambandet $\Delta_h - f'(a) = c \cdot h + O(h^2)$. Trunkeringsfelet är proportionellt mot steget h (med den okända proportionalitetsfaktorn $f''(a)/2$).

För att få trunkeringsfelet för centraldifferenskvotmodellen (2) behöver vi använda oss av taylorutvecklingen för $f(a-h)$:

$$f(a-h) = f(a) - h\,f'(a) + \frac{h^2}{2}f''(a) - \frac{h^3}{3!}f'''(a) + \frac{h^4}{4!}f^{(4)}(a) - \ldots$$

Med hjälp av formel (3) kan vi uttrycka differensen $f(a+h) - f(a-h)$ i potenser av h. Som synes försvinner alla jämna potenser och det blir dubbelt upp av termerna med udda h-potenser, alltså

$$f(a+h) - f(a-h) = 2h\,f'(a) + \frac{2h^3}{3!}f'''(a) + \frac{2h^5}{5!}f^{(5)}(a) + \ldots$$

Dividera allt med $2h$ och flytta över $f'(a)$ till vänster sida, så erhålls

$$D_h - f'(a) = h^2\frac{f'''(a)}{3!} + h^4\frac{f^{(5)}(a)}{5!} + \ldots = c \cdot h^2 + O(h^4).$$

Trunkeringsfelet är ungefärligen proportionellt mot h^2. Då steget h minskas med en faktor tio ska felet avta med faktorn hundra, och det är ju uppfyllt i tabellkolumnen $D_h - f'(1)$ fram till dess att kancellationsfelet (från $h = 0.001$ och nedåt) får dominerande inverkan.

Exempelsamlingens **Ex 1.3** *är lämplig övningsuppgift, b-delen innehåller en noggrannare differensapproximation än framåtdifferenskvot för derivatan i början av en tabell; bra formel att känna till!*

1.3.3 Differenskvotformel för andraderivatan

En god numerisk approximation till andraderivatan $f''(a)$ är differenskvotformeln:

$$f''(a) \approx \frac{f(a+h) - 2f(a) + f(a-h)}{h^2} \tag{4}$$

Trunkeringsfelet är proportionellt mot h^2. Visa det med taylorutveckling!

Richardsonextrapolation är en teknik som på ett magiskt sätt kan förbättra värdet vid numerisk derivering och andra numeriska metoder. Det kommer att behandlas i avsnitt 3.8.

1.4 Linjära ekvationssystem

1.4.1 Gausselimination och Matlabs ekvationssystemlösare

Börja med att friska upp kunskaperna från kursen i linjär algebra om vektor- och matrisräkning och lösning av ekvationssystem med gausselimination.

Linjära ekvationssystem är en vanlig ingrediens i beräkningsproblem. Det kan gälla små system med bara tre obekanta, men mycket ofta uppgår antalet obekanta till hundra eller kanske flera tusen. Med vektor- och matrisbeteckningar skrivs ekvationssystemet $\mathbf{Ax} = \mathbf{b}$. Systemmatrisen $\mathbf{A}$ är en kvadratisk matris, högerledet $\mathbf{b}$ och lösningsvektorn $\mathbf{x}$ är kolumnvektorer.

Lösningen till $\mathbf{Ax} = \mathbf{b}$ kan skrivas $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$. Det är av stort teoretiskt värde, men i praktiken löser man aldrig ekvationssystem så, dvs med inversberäkning följt av matris-vektormultiplikation, eftersom det är ett onödigt beräkningskrävande sätt. Gausselimination är basmetoden för att lösa allmänna linjära ekvationssystem, och det är denna teknik som används i MATLABs inbyggda ekvationssystemlösare: `x=A\b`.

Exempel 1: I färgaffären finns fyra olika gröna färgburkar: lindblomsgrönt, gräsgrönt, mossgrönt och havsgrönt, var och en bestående av fyra grundpigment med procentfördelningen nedan. En kund vill köpa en burk illgrönt som tyvärr inte finns i affären. Enligt innehållsdeklarationen består illgrönt av 25% av alla grundpigmenten. Kan färghandlaren åstadkomma illgrönt genom lämplig blandning av de fyra burkarna? Med några MATLAB-satser löses problemet så här:

```
lind=[50 20 30  0]';       % kolumnvektor för lindblomsgrönt
gras=[30 55  5 10]';       % kolumnvektor för gräsgrönt
moss=[15 15 45 25]';       %                 mossgrönt
hav= [ 0 10 25 65]';       %                 havsgrönt
A=[lind gras moss hav]     % matris uppbyggd av fyra kolumner
wanted=[25 25 25 25]';     % önskad pigmentfördelning, illgrönt
x=A\wanted                 % Matlabs gausseliminationslösare
```

Resultatvektorn `x` med komponenterna 0.2997, 0.2476, 0.1726, 0.2801 innehåller andelarna av de olika burkarna, och färghandlaren ska alltså blanda (avrundat till två siffror) 30% lindblomsgrönt, 25% gräsgrönt, 17% mossgrönt och 28% havsgrönt för att kunden ska få sin önskade illgröna färg.

1.4.2 Antal operationer och tidsåtgång

Att lösa ett hundra gånger hundra system med dator går snabbt, men hur arbetskrävande är det att lösa ett system med tusen obekanta — eller allmännare med n obekanta då n är stort?

Antalet operationer som krävs i gausselimination är proportionellt mot kuben på antalet obekanta, alltså mot n^3. Det innebär att om man behöver lösa ett dubbelt så stort system som tidigare så kommer det att kräva 2^3 gånger fler operationer. Lösningen av ekvationssystemet kommer därmed att ta åtta gånger längre tid teoretiskt sett. Låt oss undersöka det!

Exempel 2: Konstruera några stora ekvationssystem med helfylld systemmatris och undersök tidsåtgången i datorn för att lösa systemen Vi låter matrisen bestå av enhetsmatrisen adderad med en slumptalsmatris som är lätt att åstadkomma i MATLAB.

```
n=500;
for nr=1:3
  n, A=eye(n)+rand(n); b=(1:n)'; tic, x=A\b; toc, n=2*n;
end
```

Resultatet visar att tiden ökar kraftigt då antalet obekanta fördubblas; tidsförhållandet ligger dock under det teoretiska värdet åtta, beroende på att MATLABs systemlösare är en mycket effektiv gausseliminationsalgoritm.

n	tid i s
500	0.46
1000	2.3
2000	16.0

Varför slutade vi undersökningen vid 2000 obekanta? Fördubbling till 4000 innebär dels att tidsåtgången i vår dator skulle bli ungefär två minuter, dels skulle 4000^2 alltså 16 miljoner matriselement behöva lagras!

Med den teoretiska lösningsformeln med inversberäkning, `x=inv(A)*b`, tar beräkningen för $n = 1000$ ungefär sju sekunder, alltså en onödig tredubbling av tidsåtgången för `x=A\b`.

Repetera i läroboken i linjär algebra (eller Pohls bok) hur gausselimination går till och motivering till att antalet operationer växer med kuben på antalet obekanta.
Exempelsamlingens **Ex 3.1–3.3** *leder till små linjära ekvationssystem, pröva dem!*

1.5 Matrisegenskaper

I många tillämpningar där ekvationssystem förekommer, har systemmatrisen goda egenskaper som man kan dra nytta av vid lösning av systemet.

Om matrisen är *symmetrisk*, $\mathbf{A}^T = \mathbf{A}$, gäller att också inversen $\mathbf{A}^{-1}$ är symmetrisk. Ofta kan lagring och beräkningsarbete halveras genom att symmetrin utnyttjas.

Vid *triangulär* systemmatris behövs ingen gausselimination. Om systemet är högertriangulärt löses det med bakåtsubstitution; de obekanta bestäms i ordningen x_n, $x_{n-1}, \ldots, x_1$. Systemets understa ekvation ger $x_n = b_n/a_{nn}$. Näst understa ger $x_{n-1} = (b_{n-1} - a_{n-1,n}x_n)/a_{n-1,n-1}$. Vi arbetar oss uppåt i systemet och får till slut $x_1 = (b_1 - a_{12}x_2 - a_{13}x_3 - \ldots - a_{1n}x_n)/a_{11}$.

Ett vänstertriangulärt system löses uppifrån och ned och algoritmen kallas framåtsubstitution. Antalet operationer för lösningen av triangulära system är proportionellt mot n^2. Inversmatrisen till en högertriangulär matris är högertriangulär. En vänstertriangulär matris har vänstertriangulär invers.

En *diagonaltung* matris definieras av $|a_{ii}| \geq \sum_{k=1,k\neq i}^{n} |a_{ik}|,\ i = 1, \ldots, n$, det vill säga på varje rad gäller att beloppet av diagonalelementet är större än eller lika med summan av beloppen av de utomdiagonala elementen. När systemmatrisen är diagonaltung kan gausselimination alltid utföras utan radbyten.

Om matrisen förutom att vara diagonaltung också är *gles* (de flesta matriselementen består av nollor), så använder man med fördel en iterativ metod för att lösa systemet; mer om det i kapitlet om ickelinjära ekvationssystem.

Bandmatriser är den speciella typ av glesa matriser där alla ickenollelement finns koncentrerade i ett band kring huvuddiagonalen. *Tridiagonal* matris är den vanligaste typen av bandmatris då på varje rad bara diagonalelementet och dess närmaste granne på vardera sidan är skilda från noll. Vid stora tridiagonala system är det mycket ineffektivt att lagra hela matrisen och lösa med `x=A\b`. Det gäller att utnyttja bandstrukturen.

I MATLAB finns en uppsättning kommandon och funktioner för effektiv lagring och hantering av stora glesa matriser, `sparse`-matriser. Mer om detta i senare avsnitt.

1.6 Tridiagonala systemmatriser

1.6.1 Effektiv algoritm för lösning av tridiagonalt system

En tridiagonal matris med det allmänna utseendet

$$\begin{pmatrix} d_1 & p_1 & 0 & 0 & \cdots & 0 \\ q_1 & d_2 & p_2 & 0 & \cdots & 0 \\ 0 & q_2 & d_3 & p_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & q_{n-2} & d_{n-1} & p_{n-1} \\ 0 & 0 & 0 & 0 & q_{n-1} & d_n \end{pmatrix}$$

kan beskrivas med tre vektorer, en med n komponenter för diagonalen och en vardera för super- och subdiagonalen med $n-1$ komponenter. Man vinner i lagringsutrymme; istället för att lagra n^2 element klarar man sig med $3n$.

Lösning av ett ekvationssystem med tridiagonal matris görs som vanligt med hjälp av gausselimination, vilket innebär att man skaffar nollor under diagonalen. I detta fall underlättas eliminationen avsevärt av att nollorna redan finns på plats förutom i elementen närmast under diagonalen; det är bara subdiagonalelementen q_1, q_2, .., q_{n-1} som måste elimineras.

I MATLAB kan algoritmen skrivas:

```
function x = tridia(d,p,q,b)
  % Beräknar lösningsvektorn x till tridiagonalt system
  % med diagonal d, superdiagonal p, subdiagonal q och högerled b
    n=length(b);  r=d; g=b; x=b;
    for i=2:n
      j=i-1; m=q(j)/r(j); r(i)=d(i)-m*p(j); g(i)=b(i)-m*g(j);
    end
    x(n)=g(n)/r(n);
    for i=n-1:-1:1, x(i)=(g(i)-p(i)*x(i+1))/r(i);  end
  % Vektorn x innehåller lösningen
```

Övning: Genomför gausselimination av det tridiagonala systemet och verifiera algoritmen ovan!

Observera att `tridia` inte är en standardfunktion i MATLAB utan finns med i kurskatalogen av pedagogiska skäl. (MATLABs motsvarande sparsefunktion är mer komplicerad att anropa men betydligt snabbare och kommer att utnyttjas vid behandling av randvärdesproblem senare.)

Vi vet att tiden det tar för att lösa ett helfyllt ekvationssystem $\mathbf{Ax} = \mathbf{b}$ växer med kuben på antalet obekanta. Med effektivt utförd gausselimination på ett tridiagonalt system klarar man sig betydligt lindrigare undan; arbetet och därmed tidsåtgången växer linjärt med antalet obekanta. Proportionalitetskonstantens värde är ointressant, viktigare att veta är att fördubbling av antalet obekanta leder till ungefärlig fördubbling av beräkningstiden.

Exempel 3: Utnyttja `tridia` för att lösa systemet med följande tridiagonala matris $\mathbf{A}$ och högerledsvektor $\mathbf{b}$:

$$\mathbf{A} = \begin{pmatrix} 1 & 2 & 0 & 0 & 0 \\ 2 & 5 & 2 & 0 & 0 \\ 0 & 2 & 5 & 2 & 0 \\ 0 & 0 & 2 & 5 & 2 \\ 0 & 0 & 0 & 2 & 5 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 5.7 \\ 11.5 \\ 1.6 \\ 6.8 \\ 9.4 \end{pmatrix}$$

Skapa tre vektorer för diagonalen, super- och subdiagonalen och en vektor för högerledet. Sedan är det bara att anropa tridia.

```
dia=[1 5 5 5 5]';
sup=[2 2 2 2]';  sub=sup;
b=[5.7 11.5 1.6 6.8 9.4]';
x=tridia(dia,sup,sub,b)
```

Lösningsvektorn blir. $x_1 = 1.5$, $x_2 = 2.1$, $x_3 = -1.0$, $x_4 = 1.2$, $x_5 = 1.4$. Tillförlitligheten i resultatet ska vi strax titta närmare på.

Inversmatrisen till en tridiagonal matris har tyvärr inte samma trevliga tridiagonala egenskap utan är i allmänhet helfylld utan en enda nolla.

Mycket vanligt förekommande är *symmetriska tridiagonala matriser*, och i vissa problem kommer man inte ifrån att beräkna inversen som i detta fall alltså är symmetrisk och helfylld.

1.6.2 Asplunds metod för beräkning av inversen till symmetriska tridiagonala matriser

I kursen i linjär algebra har du lärt dig att beräkna inversen till $n \times n$ matrisen $\mathbf{A}$ genom att lösa n stycken ekvationssystem med $\mathbf{A}$ som systemmatris och

med de n enhetsvektorerna $\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_n$ som högerledsvektorer. De erhållna lösningsvektorerna utgör i tur och ordning inversens kolumner.

KTH-matematikern Edgar Asplund visade i början på 1960-talet att för symmetriska tridiagonala matriser finns det en genväg, det räcker att lösa två ekvationssystem – med första respektive sista enhetsvektorn som högerled. Det leder fram till första och sista kolumnen i $\mathbf{C} = \mathbf{A}^{-1}$ och därmed också första och sista raden (av symmetriskäl).

Inversmatrisens element i och ovanför diagonalen räknas nu lätt ut enligt följande algoritm: börja med att beteckna det kända elementet c_{1n} högst upp till höger med α. Elementet c_{ik} bestäms ur det korsvisa sambandet $c_{ik}\,\alpha = c_{in}\,c_{1k}$ (produkten av sista elementet på i-te raden och element nummer k på första raden). Elementen under diagonalen erhålls därefter ur symmetrin.

Exempel 4: Bestäm inversen till matrisen i Exempel 3.

Först ska $\mathbf{A}\mathbf{c}_1 = \mathbf{e}_1$ och $\mathbf{A}\mathbf{c}_5 = \mathbf{e}_5$ lösas, och det görs med hjälp av `tridia` (vektorerna `dia`, `sup` och `sub` är samma som tidigare):

```
c1=tridia(dia,sup,sub,[1 0 0 0 0]')
c5=tridia(dia,sup,sub,[0 0 0 0 1]')
```

Vektorn $\mathbf{c}_1$ placeras in i första kolumnen och i första raden, och vektorn $\mathbf{c}_5$ i sista kolumnen och sista raden.

$$\mathbf{C} = \begin{pmatrix} 341 & -170 & 84 & -40 & 16 \\ -170 & \times & \times & \times & -8 \\ 84 & \times & \times & \times & 4 \\ -40 & \times & \times & \times & -2 \\ 16 & -8 & 4 & -2 & 1 \end{pmatrix}, \quad \alpha = 16, \qquad \begin{aligned} c_{22}\,\alpha &= -8 \cdot (-170) \\ c_{23}\,\alpha &= -8 \cdot 84 \\ c_{24}\,\alpha &= -8 \cdot (-40) \\ c_{33}\,\alpha &= 4 \cdot 84 \\ c_{34}\,\alpha &= 4 \cdot (-40) \\ c_{44}\,\alpha &= -2 \cdot (-40) \end{aligned}$$

Nu är hela övre delen av inversen klar och eftersom matrisen är symmetrisk är hela inversmatrisen $\mathbf{C} = \mathbf{A}^{-1}$ känd.

$$\mathbf{C} = \begin{pmatrix} 341 & -170 & 84 & -40 & 16 \\ -170 & 85 & -42 & 20 & -8 \\ 84 & \times & 21 & -10 & 4 \\ -40 & \times & \times & 5 & -2 \\ 16 & -8 & 4 & -2 & 1 \end{pmatrix} = \begin{pmatrix} 341 & -170 & 84 & -40 & 16 \\ -170 & 85 & -42 & 20 & -8 \\ 84 & -42 & 21 & -10 & 4 \\ -40 & 20 & -10 & 5 & -2 \\ 16 & -8 & 4 & -2 & 1 \end{pmatrix}$$

Produkten av $\mathbf{A}$ med den erhållna inversmatrisen ska bli enhetsmatrisen, om alla räkningar gjorts korrekt. Det är en bra kontroll.

1.7 Vektor- och matrisnormer

Maxnormen $\|\mathbf{x}\|_\infty$ och euklidiska normen $\|\mathbf{x}\|_2$ för en vektor definieras av

$$\|\mathbf{x}\|_\infty = \max_{1\le i\le n} |x_i|, \quad \|\mathbf{x}\|_2 = \sqrt{x_1^2 + x_2^2 + \ldots + x_n^2}$$

En vektornorm är en reellvärd funktion med följande egenskaper:

$\|\mathbf{x}\| > 0$ för varje nollskild vektor $\mathbf{x}$ och $\|\mathbf{x}\| = 0$ bara om $\mathbf{x} = 0$.

$\|\alpha\mathbf{x}\| = |\alpha|\,\|\mathbf{x}\|$, där α är ett reellt tal.

$\|\mathbf{x} + \mathbf{y}\| \le \|\mathbf{x}\| + \|\mathbf{y}\|$ (triangelolikheten).

Till varje vektornorm hör en matrisnorm genom definitionen

$$\|\mathbf{A}\| = \max_{\mathbf{x}\neq 0} \frac{\|\mathbf{A}\mathbf{x}\|}{\|\mathbf{x}\|}.$$

De tre egenskaperna ovan gäller för matrisnormen också. Därtill kommer följande användbara olikheter: $\|\mathbf{A}\mathbf{x}\| \le \|\mathbf{A}\|\,\|\mathbf{x}\|$ och $\|\mathbf{A}\mathbf{B}\| \le \|\mathbf{A}\|\,\|\mathbf{B}\|$.

Den till $\|\mathbf{x}\|_\infty$ hörande maxnormen för en matris är $\|\mathbf{A}\|_\infty = \max_{1\le i\le n} \sum_{k=1}^{n} |a_{ik}|$.

Maxnormen för en matris bildas alltså av den "tyngsta" raden i $\mathbf{A}$, dvs den rad där summan av elementens absolutbelopp är störst. Den euklidiska normen för en matris är mycket krångligare att beräkna; den utgörs av kvadratroten ur största egenvärdet till matrisen $\mathbf{A}^T\mathbf{A}$.

Exempel 5. Beräkna normerna $\|\mathbf{b}\|_\infty$, $\|\mathbf{x}\|_\infty$, $\|\mathbf{A}\|_\infty$ och $\|\mathbf{A}^{-1}\|_\infty$ för vektorerna och matrisen i Exempel 3 och inversmatrisen i Exempel 4.

Svar: Högerledets norm: $\|\mathbf{b}\|_\infty = 11.5$. Lösningsvektorns norm: $\|\mathbf{x}\|_\infty = 2.1$. Matrisens och inversens norm: $\|\mathbf{A}\|_\infty = 9$ och $\|\mathbf{A}^{-1}\|_\infty = 651$.

1.8 Experimentell störningsräkning vid ekvationssystem

I många tillämpningar består högerledet i ekvationssystemet av mätdata medan systemmatriselementen är konstanta storheter, ofta heltal. Exempel 3 visar ett sådant system där högerledsvektorn är given med en korrekt decimal. Vi vill veta hur tillförlitlig den erhållna lösningen är.

En god idé är att göra numeriska experiment med några störda högerledsvektorer och studera känsligheten i resultatet. Eftersom indata har en korrekt decimal kan osäkerheten i varje högerledskomponent uppgå till ± 0.05. De valda högerleden i MATLAB-satserna uppfyller detta.

```
dia=[1 5 5 5 5]'; sup=[2 2 2 2]';  sub=sup;
b  =[5.7  11.5  1.6  6.8  9.4]';   x = tridia(dia,sup,sub,b)
bs1=[5.75 11.45 1.65 6.75 9.45]';  xs1=tridia(dia,sup,sub,bs1)
bs2=[5.65 11.55 1.55 6.75 9.45]';  xs2=tridia(dia,sup,sub,bs2)
```

Lösningen i det ostörda fallet är `x = 1.50, 2.10, -1.00, 1.20, 1.40`. Det första experimentet ger `xs1 = 34.05,-14.15, 7.05,-2.65, 2.95`, och det andra experimentet ger `xs2 = -25.45, 15.55,-7.65, 4.35, 0.15`.
Båda resultaten ligger långt från lösningsvektorn vid ostört högerled — inte en siffra är rätt!

Vi tittar på normerna för störningarna i indata (dvs högerledets givna mätdata) och utdata (lösningsvektorn). För båda störningsexperimenten gäller att störningsvektorns maxnorm $\|\mathbf{b}_s - \mathbf{b}\|_\infty$ är 0.05.
För första experimentets utdata får vi $\|\mathbf{x}_s - \mathbf{x}\|_\infty = 32.55$ och för det andra fallet: $\|\mathbf{x}_s - \mathbf{x}\|_\infty = |-25.45 - 1.50| = 26.95$.

Ekvationssystemet är mycket *illa konditionerat*, eftersom små störningar i indata leder till så kraftiga variationer i lösningen. Som mått på konditionen använder man förhållandet mellan relativfelet i utdata och relativfelet i indata. Ett stort konditionstal är i allmänhet en dålig egenskap — små fel in orsakar stora fel ut!

Relativfelet i indata blir räknat i maxnorm: $R_{in} = \frac{\|\mathbf{b}_s - \mathbf{b}\|}{\|\mathbf{b}\|} = \frac{0.05}{11.5} = 0.0043$.

Relativfelet i utdata blir i första experimentet $R_{ut} = \frac{\|\mathbf{x}_s - \mathbf{x}\|}{\|\mathbf{x}\|} = \frac{32.55}{2.1} = 15.5$.

Kvoten $R_{ut}/R_{in} = 15.5/0.0043 \approx 3600 \approx 4 \cdot 10^3$ är alltså det experimentellt erhållna konditionstalet. Ett konditionstal på dryga tusen innebär att man riskerar en noggrannhetsförlust på mer än tre siffror från indata till utdata.

Det teoretiska konditionstalet definieras av $k_{teor} = \|\mathbf{A}\| \cdot \|\mathbf{A}^{-1}\|$. I vårt exempel känner vi $\|\mathbf{A}\|_\infty = 9$ och $\|\mathbf{A}^{-1}\|_\infty = 651$ och beräknar alltså lätt konditionstalet i maximumnorm: $9 \cdot 651 = 5859 \approx 6 \cdot 10^3$. Det är som synes större än vårt experimentellt uträknade konditionstal — om det inte var så, skulle något av dem vara felräknat. Det teoretiska konditionstalet är nämligen ett mått på den allra värsta felförstoringen för alla tänkbara högerledsvektorer $\mathbf{b}$ med alla tänkbara störningsriktningar $\delta\mathbf{b}$.

För konditionstalet i euklidisk norm har MATLAB funktionen `cond(A)`.
Konditionstalet i maxnorm erhålls med `norm(A,inf)*norm(inv(A),inf)`.

Lämpliga övningar är **Ex 3.4, 3.5** *i exempelsamlingen.*

2 MINSTAKVADRATMETODEN

2.1 Överbestämda ekvationssystem

Ett linjärt ekvationssystem är överbestämt om antalet ekvationer är fler än antalet obekanta. I allmänhet finns ingen lösning som satisfierar alla ekvationerna, utan man får försöka finna en så god approximation som möjligt.

Exempel 1: Längs en hundrameters löparbana vill man ha markeringar vid 60 meter och 80 meter. Pelle, som anser sig ha enmeterskliv, stegar upp 60 steg och markerar platsen x_1. Från denna plats stegar han 20 steg till platsen x_2. Men därifrån till hundrametersstrecket blir det bara 17 steg. Bäst att kontrollera, tycker Pelle och stegar tillbaka till x_1. Efter 38 steg är han där. Fyra samband finns nu: $x_1 \approx 60$; $x_2 - x_1 \approx 20$; $100 - x_2 \approx 17$; $100 - x_1 \approx 38$. Kan man genom dessa mätningar få säkrare värden på Pelles avståndsmarkeringar?

Det blir ett överbestämt system som i matrisform kan skrivas $\mathbf{Ax} \approx \mathbf{y}$. Vi kommer att behöva beräkna avvikelsen mellan högerled och vänsterled, alltså $\mathbf{y} - \mathbf{Ax}$. Denna residualvektor betecknas $\mathbf{r}$ och har komponenterna r_1, r_2, r_3, r_4,

$$\begin{pmatrix} 1 & 0 \\ -1 & 1 \\ 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \approx \begin{pmatrix} 60 \\ 20 \\ 83 \\ 62 \end{pmatrix}$$

Systemet består av fyra ekvationer men bara två okända storheter, x_1 och x_2.

För att kunna lösa ut två obekanta krävs två ekvationer. Det gäller att på listigt sätt kombinera ekvationerna så att informationen behålls men antalet ekvationer reduceras till två. Ingen lösning finns som satisfierar alla ekvationerna, i stället försöker man finna en "bästa" lösning som minimerar felkvadratsumman $\sum r_i^2$.

Genom multiplicering av båda leden i $\mathbf{Ax} \approx \mathbf{y}$ med en matris med *två rader och fyra kolumner* överförs systemet till ett system med två ekvationer och två obekanta. En matris som uppfyller dimensionskravet finns nära till hands, nämligen $\mathbf{A}^T$ — vi vet dock inte om den är lämplig för övrigt. Men det är värt att pröva! Bilda alltså $\mathbf{A}^T\mathbf{A}$ och $\mathbf{A}^T\mathbf{y}$ och lös därefter $\mathbf{A}^T\mathbf{Ax} = \mathbf{A}^T\mathbf{y}$.

$$\mathbf{A}^T\mathbf{A} = \begin{pmatrix} 1 & -1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ -1 & 1 \\ 0 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 3 & -1 \\ -1 & 2 \end{pmatrix}$$

$$\mathbf{A}^T\mathbf{y} = \begin{pmatrix} 1 & -1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} 60 \\ 20 \\ 83 \\ 62 \end{pmatrix} = \begin{pmatrix} 102 \\ 103 \end{pmatrix}$$

Det överbestämda systemet $\mathbf{Ax} \approx \mathbf{y}$ ersätts av $\begin{pmatrix} 3 & -1 \\ -1 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 102 \\ 103 \end{pmatrix}$

med lösningen $x_1 = 61.4$ och $x_2 = 82.2$ för Pelles avståndspositioner. Med dessa värden insatta får residualvektorn $\mathbf{r}$ komponenterna $-1.4, -0.8, 0.8, 0.6$. Residualerna är små och har varierande tecken. Vi har faktiskt funnit den så kallade minstakvadratlösningen $\mathbf{x}$ som minimerar felkvadratsumman $\sum r_i^2$.

2.2 Minstakvadratlösning med normalekvationerna

Med minstakvadratmetoden löser man överbestämda linjära ekvationssystem $\mathbf{Ax} \approx \mathbf{y}$. Minstakvadratlösningen $\mathbf{x}$ erhålls som lösning till systemet av *normalekvationer* $\mathbf{A}^T\mathbf{Ax} = \mathbf{A}^T\mathbf{y}$. Lösningen har den goda egenskapen att minimera felkvadratsumman $\sum r_i^2$, som kan skrivas $\mathbf{r}^T\mathbf{r}$ vilket i sin tur är identiskt med euklidiska normen i kvadrat, alltså $\|\mathbf{r}\|_2^2$ eller $\|\mathbf{y} - \mathbf{Ax}\|_2^2$. Annorlunda uttryckt: Lösningen i minstakvadratmetodens mening till ett överbestämt system är den lösning som minimerar residualvektorns euklidiska norm.

I MATLAB får man minstakvadratlösningen till det överbestämda systemet $\mathbf{Ax} \approx \mathbf{y}$ med satsen `x=A\y`, alltså samma som för ett vanligt linjärt ekvationssystem $\mathbf{Ax} = \mathbf{y}$. MATLAB kontrollerar först om $\mathbf{A}$ är en kvadratisk eller rektangulär matris. Om $\mathbf{A}$ är kvadratisk löses systemet med gausselimination, om $\mathbf{A}$ är rektangulär löses motsvarande minstakvadratproblem. Normalekvationerna behöver alltså aldrig ställas upp i MATLAB.

Exempel 2: Anpassning av rät linje till mätdata

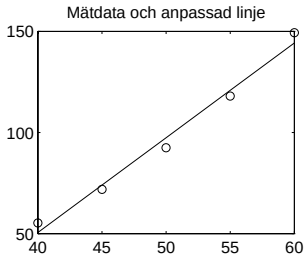
Vi har en tabell över vattenångans partialtryck p i luft uppmätt vid olika temperaturer.

T^oC	40	45	50	55	60
p mm Hg	55.3	71.9	92.5	118.0	149.4

Mätpunkterna ligger approximativt på en rät linje, och som matematisk modell för kurvanpassningen lämpar sig förstagradspolynomet $p(T) = c_1 + c_2T$. Vilka värden ska c_1 och c_2 ha? Vi får fem ekvationer som ska vara approximativt uppfyllda: $c_1 + 40c_2 \approx 55.3$, $c_1 + 45c_2 \approx 71.9$, $c_1 + 50c_2 \approx 92.5$, $c_1 + 55c_2 \approx 118.0$, $c_1 + 60c_2 \approx 149.4$.

Systemet kan skrivas på matrisform $\mathbf{Ac} \approx \mathbf{y}$:

$$\begin{pmatrix} 1 & 40 \\ 1 & 45 \\ 1 & 50 \\ 1 & 55 \\ 1 & 60 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} \approx \begin{pmatrix} 55.3 \\ 71.9 \\ 92.5 \\ 118.0 \\ 149.4 \end{pmatrix}$$



För handräkning av problemet behöver vi ställa upp och lösa normalekvationerna $\mathbf{A}^T\mathbf{Ac} = \mathbf{A}^T\mathbf{y}$:

$$\begin{pmatrix} 5 & 250 \\ 250 & 12750 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} = \begin{pmatrix} 487.1 \\ 25526.5 \end{pmatrix}$$

Minstakvadratlösningen blir $c_1 = -136.9$ och $c_2 = 4.686$. Den bäst anpassande räta linjen lyder alltså $p(T) = -136.9 + 4.686T$. Residualerna ligger mellan -5 och 5 och felkvadratsumman är 85.38. I `matlab` löses problemet av följande satser:

```
T=(40:5:60)';  y=[55.3 71.9 92.5 118 149.4]';
A=[ones(5,1) T];
c=A\y,  yanp=A*c;  r=y-yanp,  felkvsum=r'*r
plot(T,y,'o',T,yanp)
```

2.3 Linjär modellanpassning som minstakvadratproblem

Givet är en uppsättning data $(t_i,\ y_i)$, $i = 1, 2, ..., m$. Givet är också en linjär modellfunktion $F(t) = c_1\phi_1(t) + c_2\phi_2(t) + \ldots + c_n\phi_n(t)$, som är en linjärkombination av på förhand valda basfunktioner $\phi_k(t)$, $k = 1, 2, ..., n,\ n < m$. Problemet är att bestämma parametrarna $c_1,\ c_2, ..., c_n$ så att summan av kvadraterna på avvikelserna $y_i - F(t_i)$ minimeras.

Modellfunktionen kan vara ett förstagradspolynom $F(t) = c_1 + c_2 t$ med basfunktionerna $\phi_1(t) = 1$ och $\phi_2(t) = t$. I periodiska problem kan modellfunktionen vara ett trigonometriskt polynom $F(t) = c_1 + c_2 \cos t + c_3 \sin t$. I ett annat sammanhang kan modellen utgöras av ett blandat uttryck, t ex $F(t) = c_1 t + c_2 e^{-t} + c_3 \cos \pi t$ med basfunktionerna $\phi_1(t) = t,\ \phi_2(t) = e^{-t}$ och $\phi_3(t) = \cos \pi t$.

För att minstakvadratmetoden ska kunna tillämpas krävs att parametrarna ingår på det beskrivna linjära sättet. Om parametrarna inte gör det, är det ibland möjligt att transformera problemet till linjär form. I annat fall krävs en metod för ickelinjär modellanpassning som behandlas senare.

Geometrisk härledning av normalekvationerna

Vi ska med en geometrisk betraktelse härleda normalekvationerna för fallet $F(t) = c_1\phi_1(t) + c_2\phi_2(t)$. Med m mätdata får vi det överbestämda systemet $F(t_i) \approx y_i,\ i = 1, 2, ..., m$. I vektorform kan det skrivas $\mathbf{F} \approx \mathbf{y}$, där $\mathbf{F} = c_1\boldsymbol{\phi}_1 + c_2\boldsymbol{\phi}_2$, eller med utskrivna komponenter

$$\mathbf{F} = c_1 \begin{pmatrix} \phi_1(t_1) \\ \phi_1(t_2) \\ \vdots \\ \phi_1(t_m) \end{pmatrix} + c_2 \begin{pmatrix} \phi_2(t_1) \\ \phi_2(t_2) \\ \vdots \\ \phi_2(t_m) \end{pmatrix} = \begin{pmatrix} \phi_1(t_1) & \phi_2(t_1) \\ \phi_1(t_2) & \phi_2(t_2) \\ \vdots & \vdots \\ \phi_1(t_m) & \phi_2(t_m) \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} = \mathbf{A}\mathbf{c}$$

Önskemålet är att finna c_1 och c_2 så att $\|\mathbf{y} - \mathbf{F}\|_2$ minimeras. Euklidisk norm kan tolkas som längden av en vektor, alltså gäller det att finna $\mathbf{F}$ sådan att längden av vektorn $\mathbf{y} - \mathbf{F}$ blir så liten som möjligt.

Resonera så här: $\mathbf{F}$ är en linjärkombination av $\boldsymbol{\phi}_1$ och $\boldsymbol{\phi}_2$. Mängden av alla linjärkombinationer av $\boldsymbol{\phi}_1$ och $\boldsymbol{\phi}_2$ spänner upp ett plan. Vektorn $\mathbf{y}$ tillhör inte planet; $\mathbf{y}$ kan ju inte skrivas som en linjärkombination av $\boldsymbol{\phi}_1$ och $\boldsymbol{\phi}_2$. Däremot finns den sökta vektorn $\mathbf{F}$ i planet. Problemet är nu att i detta plan finna den vektor som har det kortaste avståndet till den givna vektorn $\mathbf{y}$. Lösningen får man av det välkända förhållandet att det vinkelräta avståndet från en punkt till ett plan är det kortaste.

Vektorn $\mathbf{y} - \mathbf{F}$ ska därför vara ortogonal mot planet. I planet ligger ju vektorerna $\boldsymbol{\phi}_1$ och $\boldsymbol{\phi}_2$. Alltså gäller att $\mathbf{y} - \mathbf{F}$ ska vara ortogonal mot såväl $\boldsymbol{\phi}_1$ som $\boldsymbol{\phi}_2$. Det innebär att $\boldsymbol{\phi}_1^T(\mathbf{y} - \mathbf{F}) = 0$ och $\boldsymbol{\phi}_2^T(\mathbf{y} - \mathbf{F}) = 0$, vilket kan sammanföras i

$$\begin{pmatrix} \boldsymbol{\phi}_1^T \\ \boldsymbol{\phi}_2^T \end{pmatrix} (\mathbf{y} - \mathbf{F}) = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \implies \mathbf{A}^T(\mathbf{y} - \mathbf{F}) = \mathbf{0} \implies \mathbf{A}^T(\mathbf{y} - \mathbf{A}\mathbf{c}) = \mathbf{0}.$$

Det sista sambandet kan också skrivas $\mathbf{A}^T\mathbf{A}\mathbf{c} = \mathbf{A}^T\mathbf{y}$, alltså normalekvationerna. Ortogonalitetsegenskapen $\mathbf{A}^T(\mathbf{y} - \mathbf{A}\mathbf{c}) = \mathbf{0}$ eller kortare skrivet, $\mathbf{A}^T\mathbf{r} = \mathbf{0}$, är viktig. Den kan t ex användas som kontroll av en beräknad minstakvadratlösning.

Recept för minstakvadratproblemlösning

- Formulera först det överbestämda systemet för den linjära modellen.
- Lägg givna data i vektorer. Transformera data om så behövs.
- Generera systemmatrisen i det överbestämda ekvationssystemet.
- Beräkna minstakvadratlösningen.
- Beräkna den linjära modellens värden i mätpunkterna och bilda residualvektorn.
- Beräkna modellens värden med mindre steg för att få en jämn kurva. Återtransformera modellens värden om så behövs.
- Rita kurvresultatet och markera givna mätvärden; rita residualkurvan.

Exempel 3: Radioaktivt sönderfall

Ett radioaktivt ämne sönderfaller i ickeradioaktiva produkter. Strålningsvärdet x (becquerel) har uppmätts vid ökande tider (sekunder). Anpassa mätdata nedan till den matematiska modellen $x(t) = x_0 e^{-kt}$.

t	0	500	1000	1500	2000	2500	3000
x	0.100	0.0892	0.0776	0.0705	0.0603	0.0542	0.0471

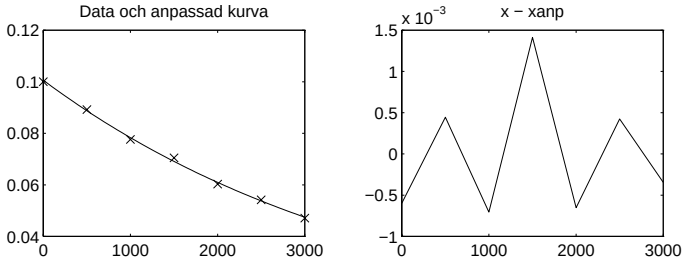
Skriv om uttrycket på linjär form genom att logaritmera bägge leden:

$$y = \ln x = \ln x_0 - k\,t = c_1 + c_2\,t \;\text{ med }\; x_0 = e^{c_1} \;\text{ och }\; k = -c_2.$$

En rät linje kan anpassas till de *logaritmerade* x-värdena. Det överbestämda ekvationssystemet består av ekvationerna $c_1 + c_2 t_i \approx \ln x_i,\ i = 1, 2, ..., 7$.
Receptet tillämpas och leder till MATLAB-satserna:

```
t=(0:500:3000)';  x=[0.1000 0.0892 0.0776 0.0705 0.0603 0.0542 0.0471]';
y=log(x);
A=[ones(size(t)) t]; c=A\y,  yanp=A*c;  res=y-yanp
x0=exp(c(1)), k=-c(2)
xanp=x0*exp(-k*t);  resx=x-xanp;
T=(0:50:3000)'; X=x0*exp(-k*T);
subplot(2,2,1), plot(t,x,'x', T,X)
subplot(2,2,2), plot(t,resx)
```

Resultatet blir $x_0 = 0.1006$ och $k = 2.5052 \cdot 10^{-4}$. Resultatet efter återtransformering till exponentialkurva visas i vänstra figuren. Högra diagrammet visar avvikelsen mellan uppmätta strålningsvärden och den anpassade modellen.



2.4 Val av basfunktioner vid polynomanpassning

Normalekvationerna utgör ett ekvationssystem med några trevliga egenskaper: systemmatrisen $\mathbf{A}^T\mathbf{A}$ är liten jämfört med ursprungliga matrisen, och den är symmetrisk. Tyvärr finns också en sämre egenskap: det är lätt hänt att $\mathbf{A}^T\mathbf{A}$ får ett mycket stort konditionstal. Det innebär att små avrundningsfel i matrisen eller små beräkningsfel som införs under lösningens gång fortplantas kraftigt och kan ha förödande effekt på de beräknade parametrarna. Vid polynomanpassning kan en enkel *centrering* förbättra konditionen avsevärt.

Exempel: Anpassa ett andragradspolynom till fem punkter med x-koordinaterna 15, 16, 17, 18, 19 och y-koordinaterna 15, 8, 5, 3, 3. Vi prövar dels den vanliga naiva formen $F(x) = a_1 + a_2 x + a_3 x^2$, och dels en centrerad ansats då man i stället för potenser av x väljer basfunktioner som är potenser av $x - x_{medel}$. I exemplet är $x_{medel} = 17$, därför skriver vi $F(x) = c_1 + c_2(x - 17) + c_3(x - 17)^2$.

I första fallet byggs det överbestämda systemet upp av en matris med ettor i första kolumnen, x-värdena i den andra och x-värdenas kvadrater i tredje kolumnen. Normalekvationerna blir

$$\begin{pmatrix} 5 & 85 & 1455 \\ 85 & 1455 & 25075 \\ 1455 & 25075 & 434979 \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} 34 \\ 549 \\ 8923 \end{pmatrix}$$

Matrisen får alltså mycket stora element. Konditionstalet är $\approx 3 \cdot 10^9$. Lösningen till normalekvationerna är $a_1 = 363.6$, $a_2 = -39.33$, $a_3 = 1.0714$, men konditionstalets storlek gör att resultatet mycket lätt avviker från detta.

Om till exempel elementet 434979 avrundas till 435000 (en relativ störning på $21/434979 = 5 \cdot 10^{-5}$) och inga andra fel görs blir lösningen $a_1 = 179.1$, $a_2 = -17.47$ och $a_3 = 0.4286$, inte en siffra är rätt!

Den centrerade polynomansatsen $F(x) = c_1 + c_2(x-17) + c_3(x-17)^2$ ger en mycket trevligare matris $\mathbf{A}$ med ettor i första kolumnen, talen -2, -1, 0, 1, 2 i andra kolumnen och talen 4, 1, 0, 1, 4 i tredje kolumnen. Normalekvationerna blir

$$\begin{pmatrix} 5 & 0 & 10 \\ 0 & 10 & 0 \\ 10 & 0 & 34 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} 34 \\ -29 \\ 83 \end{pmatrix}$$

med lösningen $c_1 = 4.657$, $c_2 = -2.900$, $c_3 = 1.0714$. Polynomet blir alltså $F(x) = 4.657 - 2.900(x-17) + 1.0714(x-17)^2$. Matrisen $\mathbf{A}^T\mathbf{A}$ får så lågt konditionstal som 28. Centrering ger dubbel utdelning, dels förbättras konditionen, dels blir räkningarna mycket enklare eftersom matriselementen får behaglig storlek. Genom att välja medelvärdet av x-värdena vid centreringen uppnår vi viss ortogonalitet – en hel del element i normalekvationsmatrisen blir noll.

2.5 Residualanalys

Vid modellanpassning erhålls förutom minstakvadratlösningen även en residualvektor $\mathbf{r}$ som kan ritas som en residualkurva . Genom att studera den lite närmare kan man få uppslag till lämplig modifiering av modellfunktionen. Vi ska visa det i ett exempel. Här är en tabell över vattenångans tryck över ett större temperaturområde:

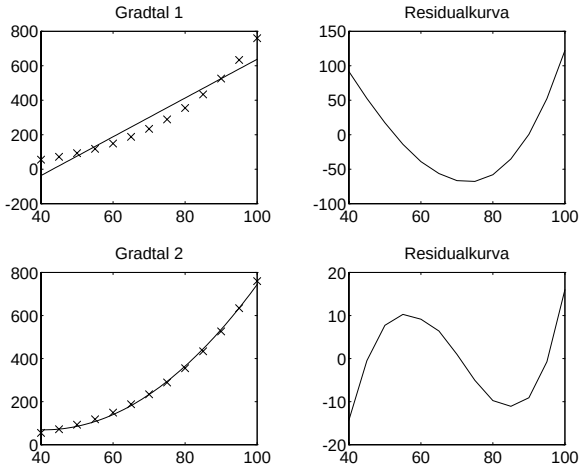
T	40	45	50	55	60	65	70	75	80	85	90	95	100
p	55	72	93	118	149	188	234	289	355	434	526	634	760

Anpassning med rät linje $p(T) = c_1 + c_2 T$ till dessa data ger resultatet överst i nästa figur. Som synes är en rät linje en ganska dålig modell. Till höger visas residualkurvan och den ger idé om hur modellen kan förbättras. Residualkur-

van liknar nämligen en parabel, det vill säga ett andragradspolynom. Korrigera därför ursprungsmodellen med andragradspolynomet $q_1 + q_2T + q_3T^2$.

$$p(T) = c_1 + c_2T + q_1 + q_2T + q_3T^2 = (c_1 + q_1) + (c_2 + q_2)T + q_3T^2$$

som kan ju kan skrivas $p(T) = a_1 + a_2T + a_3T^2$, där a_1, a_2 och a_3 är okända parametrar. När denna modell anpassas till mätdata erhålls resultatet i de undre figurerna.



Parabeln ger mycket bättre anpassning till mätvärdena; felkvadratsumman har nu sjunkit från 48000 till 1100. Eftersom residualkurvan liknar ett tredjegradspolynom kan man pröva att öka gradtalet till tre, men det lämnas som övning.

```
T=(40:5:100)';  p=[55 72 93 118 149 188 234 289 355 434 526 634 760]';
A=[ones(13,1) T ]; c=A\p
r=p-A*c;  fkvsum1=r'*r
X=(40:100)';  S=[ones(size(X)) X];  P=S*c;
subplot(2,2,1), plot(T,p,'x', X,P), title('Gradtal 1')
subplot(2,2,2), plot(T,r), title('Residualkurva')
A=[A T.^2];  c=A\p
r=p-A*c;  fkvsum2=r'*r
S=[S X.^2];  P=S*c;
subplot(2,2,3), plot(T,p,'x', X,P), title('Gradtal 2')
subplot(2,2,4), plot(T,r), title('Residualkurva')
```

I exempelsamlingen finns många övningar på minstakvadratproblem av olika karaktär, alla med fullständiga lösningar.
Ex 4.1–4.19 *rekommenderas.*

2.6 Experimentell störningsräkning

I de flesta praktiska problem bygger indata på uppmätta värden behäftade med viss osäkerhet. Det kan vara onoggrannhet i mätinstrumenten, olika störningar som påverkar mätningarna etc. Naturligtvis orsakar detta att utdata (såsom beräknade värden på modellparametrar) blir behäftade med osäkerhet, som man helst vill ha full kontroll över.

Ett vanligt sätt att behandla mätfel är att anta att de är normalfördelade med medelvärde noll och en viss spridning σ. I detta avsnitt ska vi undersöka störningarnas inverkan genom att generera *simulerade* mätfel med MATLABs slumptalsgenerator. Genom att köra samma minstakvadratproblem med olika störningar kan vi få en uppfattning om hur störningskänslig minstakvadratlösningen är, både när det gäller parametrarna och den resulterande kurvan.

När vi anpassar en viss modell till data är ju önskemålet att få både noggranna (= ej störningskänsliga) parametervärden och små residualer (= god anpassning). Tyvärr går dessa två önskemål inte alltid att uppfylla samtidigt — vill vi ha liten residual så blir parametrarna ofta störningskänsliga; vill vi ha noggranna parametrar så får vi nöja oss med större residualer.

Exempel 4: Störningskänslighet vid polynomanpassning

Låt oss återvända till tabellen i Exempel 2 där T-värdena betraktas som exakta storheter, medan de fem uppmätta p-värdena antas ha en osäkerhet på ± 0.05. Vi vill pröva anpassning med ett förstagradspolynom $c_1 + c_2 T$ och studera hur parametervärdena påverkas av störningar i mätdata. Därefter gör vi om förfarandet då modellen är ett andragradspolynom på den naiva formen $c_1 + c_2 T + c_3 T^2$.

Följande program simulerar 100 störda indatatabeller där störningarna får utgöras av slumpgenererade tal mellan -0.05 och 0.05. Bestäm maxbeloppet av avvikelsen i varje parameter, vilket ger en skattning av absoluta felgränsen för parametervärdet.

```
T=(40:5:60)';  y=[55.3 71.9 92.5 118.0 149.4]';
grad=1, A=[ones(size(T)) T];  c=A\y;
r=y-A*c;  fkvsum=r'*r
c1=c(1);  c2=c(2);                          % lösning, ostörda fallet
c1d=[]; c2d=[];
for i=1:100                                 % 100 förstagradspolynom
  ys=y+0.05*(2*rand(size(T))-1); cs=A\ys;   % simulera störda data och lös
  c1d=[c1d cs(1)-c1]; c2d=[c2d cs(2)-c2];   % lagra avvikelserna
end
c1err=[c1 max(abs(c1d))]
c2err=[c2 max(abs(c2d))]
kondA=cond(A)
```

```
% Öka gradtalet
  grad=2, A=[A T.^2];  c=A\y;
  r=y-A*c;  fkvsum=r'*r
  c1=c(1);  c2=c(2); c3=c(3);
  c1d=[]; c2d=[]; c3d=[];
  for i=1:100                                        % 100 andragradspolynom
    ys=y+0.05*(2*rand(size(T))-1); cs=A\ys;
    c1d=[c1d cs(1)-c1]; c2d=[c2d cs(2)-c2]; c3d=[c3d cs(3)-c3];
  end
  c1err=[c1 max(abs(c1d))],
  c2err=[c2 max(abs(c2d))]
  c3err=[c3 max(abs(c3d))],
  kondA=cond(A)
```

För förstagradspolynomet blir felkvadratsumman 85.4 och parametrarna med skattad osäkerhet blir $c_1 = -136.9 \pm 0.2$ och $c_2 = 4.686 \pm 0.004$. Konditionstal: 360. För det anpassande andragradspolynomet blir felkvadratsumman så liten som 0.36, men parametrarna får nu större osäkerhet: $c_1 = 104.6 \pm 2.2$, $c_2 = -5.2 \pm 0.1$ och $c_3 = 0.099 \pm 0.001$. Det beror på att problemet blivit mera illakonditionerat, matrisens konditionstal är drygt 100000. (Med centrering skulle i detta fall en stor konditionsförbättring nås.)

Studera **Ex 4.21**, *kemisk reaktionsmodell enligt Arrhenius.*

2.7 Praktisk statistik vid minstakvadratanpassning

År 1805 publicerade Legendre en kometbaneberäkning, där för första gången minstakvadratmetoden användes. Han hade många observationer av kometpositioner och till dem skulle han anpassa en kurva av den allmänna form som Newton härlett, alltså precis den typ av problem som vi löst. Legendre gav ingen annan motivering till att han minimerade just felkvadratsumman än att metoden gav enkla räkningar. Gauss kommenterade då att han sedan länge använde metoden, eftersom den gav den *mest effektiva statistiska uppskattningen av parametervärdena.*

Vi ska se vad han menade med det. Låt $\mathbf{c}^*$ vara den verkliga (okända) parametervektorn och $\mathbf{A}\mathbf{c}^*$ de exakta mätvärden (m stycken) som vi skulle ha om inga mätfel funnes. I verkligheten finns det slumpmässiga störningar $d_1, d_2, \ldots, d_m$ i mätvärdena som gör att mätdatavektorn kan skrivas $\mathbf{y} = \mathbf{A}\mathbf{c}^* + \mathbf{d}$. Om vi använder minstakvadratmetoden på problemet $\mathbf{A}\mathbf{c} \approx \mathbf{y}$ gäller enligt normalekvationerna $\mathbf{A}^T\mathbf{A}\mathbf{c} = \mathbf{A}^T\mathbf{y}$, alltså $\mathbf{c} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{y}$.

Sätt in $\mathbf{y} = \mathbf{A}\mathbf{c}^* + \mathbf{d}$ så får vi ett samband mellan parametervektorn och mätfelen: $\mathbf{c} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T(\mathbf{A}\mathbf{c}^* + \mathbf{d}) = \mathbf{c}^* + (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{d}$.

Ju större variansen är i mätdata, desto större blir tydligen variansen i parametrarna $\mathbf{c}$, men medelvärdet är i alla fall $\mathbf{c}^*$ (som det ska vara).

Gauss visade att minstakvadratmetoden är den anpassningsmetod som ger minsta möjliga varians i $\mathbf{c}$, och han gav också en uppskattning av osäkerheten i varje komponent av parametervektorn $\mathbf{c}$:

Om $\mathbf{c}$ (med n komponenter) och residualvektorn $\mathbf{r}$ (med m komponenter) har beräknats med minstakvadratmetoden och om man förutsätter att *modellen är perfekt*, så beror residualerna helt på *slumpfel i mätvärdena*. Variansen i detta slumpfel kan uppskattas med $s^2 = \mathbf{r}^T\mathbf{r}/(m-n)$.

För beräkning av variansen i de beräknade parametrarna behövs matrisen $W = (\mathbf{A}^T\mathbf{A})^{-1}$, egentligen endast dess diagonalelement w_{ii}. Variansen i komponenten c_i kan nämligen skattas med

$$\sigma^2(c_i) \approx w_{ii}\, s^2 = w_{ii}\, \mathbf{r}^T\mathbf{r}/(m-n)$$

och standardavvikelsen alltså $\sigma(c_i) \approx \sqrt{w_{ii}\, \mathbf{r}^T\mathbf{r}/(m-n)}$.

Ofta anger man felgränsen $\pm 2\sigma$. För normalfördelning har man då 95% konfidens att felet ligger inom felgränsen.

Exempel 5: Radioaktivt sönderfall, osäkerhet i parametrarna

Använd skattningarna ovan för att undersöka noggrannheten i de beräknade parametervärdena i Exempel 3.

```
m=7; n=2; t=(0:500:3000)';
x=[0.1000 0.0892 0.0776 0.0705 0.0603 0.0542 0.0471]'; y=log(x);
A=[ones(m,1) t];  c=A\y;
yanp=A*c;  r=y-yanp;
s2=r'*r/(m-n), W=inv(A'*A); sigma2c=diag(W)*s2;
standfel=sqrt(sigma2c);                    % standardavvikelser i c
c1=c(1), c1fel95=2*standfel(1)             % 95% konfidensintervall för c1
c2=c(2), c2fel95=2*standfel(2)             % 95% konfidensintervall för c2
x0=exp(c1);
x0min=exp(c1-c1fel95);
x0max=exp(c1+c1fel95);
x0intval=[x0min  x0  x0max]                % konfidensintervall för x0 (95%)
k=-c2;  kmin=-c2-c2fel95; kmax=-c2+c2fel95;
k_intval=[kmin k  kmax]                    % konfidensintervall för k (95%)
```

Det numeriska resultatet blir $c_1 = -2.2966 \pm 0.0171$ (felgränsen är värdet av variabeln `c1fel95`) och $c_2 = -0.0002505 \pm 0.0000095$. Utskriften av `x0intval` är de tre talen 0.0989, 0.1006, 0.1023.

Ur resultaten kan vi sammanfatta parametervärdena x_0 och k med felgränser: $x_0 = 0.101 \pm 0.002$ och $k = 0.00025 \pm 0.00001$.

3 INTERPOLATION

3.1 Interpolationspolynom som modell till rutschbana

En rutschbana ska konstrueras genom sex punkter med de givna x-värdena $x = 0.5,\ 1.5,\ 2.5,\ 3,\ 4,\ 5$ och tillhörande höjdvärden $y = 3,\ 1.5,\ 1.5,\ 1,\ 1,\ 0$ (mått i meter). Man kan interpolera genom några punkter i taget eller utnyttja alla sex punkterna på en gång. Låt oss pröva några olika modeller och betrakta den interpolerande rutschbanekurvan för varje fall.

Enklast är att dra räta linjer mellan punkterna, *linjär interpolation* . Om vi vill beräkna ett visst värde på den räta linjen som sammanbinder första och andra punkten utnyttjar vi formeln för linjär interpolation

$$P(x) = y_1 + \frac{y_2 - y_1}{x_2 - x_1}(x - x_1). \tag{5}$$

Vid *kvadratisk interpolation* lägger man ett andragradspolynom genom tre givna punkter. Allmänt gäller att genom n givna punkter finns ett entydigt bestämt interpolerande polynom av gradtal $n-1$.
Polynomet kan ansättas på den **naiva formen**

$$P(x) = c_1 + c_2 x + c_3 x^2 + \cdots + c_n x^{n-1} \tag{6}$$

eller skrivas med en ansats som föreslogs av Newton

$$P(x) = c_1 + c_2(x-x_1) + c_3(x-x_1)(x-x_2) + c_4(x-x_1)(x-x_2)(x-x_3) + \cdots \tag{7}$$

och som kallas för **Newtons interpolationsformel**. Koefficienterna c_i här kommer att anta helt andra värden än koefficienterna i den naiva ansatsen.

Vilken av formlerna man än väljer, så gäller att de okända koefficienterna bestäms ur de n sambanden $P(x_i) = y_i, \quad i = 1, 2, \ldots, n,$ som bildar ett linjärt ekvationssystem med n ekvationer och n obekanta.

Newtons ansats (7) är den allra bästa ur handräkningssynpunkt eftersom systemmatrisen blir triangulär och elementen oftast blir lätthanterligt små. Koefficienterna erhålls med så kallad framåtsubstitution.

Vid datorräkning kan man i regel utnyttja den naiva ansatsen (6) som är enklare att programmera. Systemmatrisens första kolumn kommer nu att innehålla n stycken ettor, andra kolumnen består av x-värdena, tredje kolumnen av x-värdena i kvadrat och så vidare. Matrisen kallas vandermondematris efter 1700-talsmatematikern Vandermonde.

Lagrangeinterpolation är en alternativ metod för bestämning av interpolationspolynomet $P(x)$. Det skrivs då

$$P(x) = y_1 L_1(x) + y_2 L_2(x) + \cdot + y_n L_n(x), \quad \text{där } L_j(x) = \frac{\prod_{k=1,k\neq j}^{n}(x - x_k)}{\prod_{k=1,k\neq j}^{n}(x_j - x_k)}$$

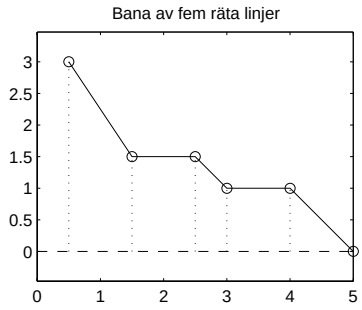
Det är en explicit formel för polynomet av gradtal $n-1$, men tyvärr med ganska otrevligt utseende. Vi skriver ut formeln för fallet med ett interpolerande andragradspolynom genom tre punkter.

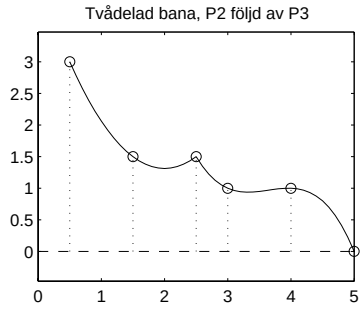
$$P(x) = y_1 \frac{(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)} + y_2 \frac{(x-x_1)(x-x_3)}{(x_2-x_1)(x_2-x_3)} + y_3 \frac{(x-x_1)(x-x_2)}{(x_3-x_1)(x_3-x_2)}$$

Rutschbana av två sammanfogade kurvor:

När det gäller rutschbaneproblemet är sex punkter givna; vi kan till exempel välja ett andragradspolynom genom de tre första punkterna och därefter ett tredjegradspolynom genom punkterna tre till och med sex. Vi delar alltså intervallet i två delar med brytpunkt vid x_3. Det blir följande satser i MATLAB:

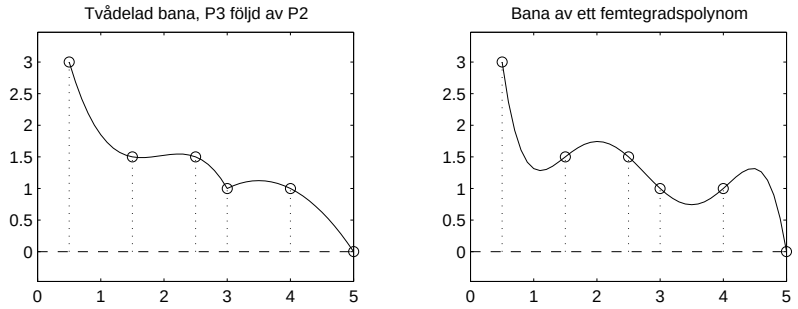
```
xx=[0.5 1.5  2.5  3  4  5]';
yy=[ 3  1.5  1.5  1  1  0]';
subplot(2,2,1), stem(xx,yy,':'), hold on
plot([0 5],[0 0],'g--')              % grönstreckad mark
plot(xx,yy), axis equal              % räta linjer mellan mätpunkterna

subplot(2,2,2), stem(xx,yy,':'), hold on
plot([0 5],[0 0],'g--')
x=xx(1:3);  y=yy(1:3);           % lägg andragradare genom pkt nr 1,2,3
A=[ones(3,1)  x  x.^2],  c=A\y
X2=0.5:0.1:2.5;  P2=c(1)+c(2)*X2+c(3)*X2.^2;
x=xx(3:6);  y=yy(3:6);           % lägg tredjegradare genom pkt 3,4,5,6
A=[ones(4,1)  x  x.^2  x.^3],  c=A\y
X3=2.5:0.1:5;  P3=c(1)+c(2)*X3+c(3)*X3.^2+c(4)*X3.^3;
plot(X2,P2, X3,P3),  axis equal
```





Det blir en rutschbana med en tvär knyck vid brytpunkten x_3. Kan det möjligen bli bättre om brytpunkten läggs vid x_4, så att interpolationen först görs med tredjegradspolynom och andragradskurvan kommer på slutet?

Rutschbanan blir annorlunda med en spets nedåt, men knappast acceptabel nu heller.



3.2 Polynom av hög grad genom alla punkter

Genom att utnyttja alla sex givna punkterna kan vi interpolera med ett femtegradspolynom.

```
% bana5, Femtegradspolynom genom alla punkterna
  x=[0.5 1.5 2.5  3  4  5]';  y=[3 1.5 1.5  1  1  0]';
  A=[ones(6,1) x  x.^2  x.^3  x.^4  x.^5],  c=A\y
  X=x(1):0.1:x(6);
  P=c(1)+c(2)*X+c(3)*X.^2+c(4)*X.^3+c(5)*X.^4+c(6)*X.^5;
  subplot(2,2,4), stem(x,y,':'), hold on
  plot([0 5],[0 0],'g--', X,P), axis equal
```

När gradtalet på polynomet växer, blir satsen som beräknar polynomvärdena allt längre. Det kan då vara idé att ta till *Horners algoritm* som är en enkel och effektiv algoritm för uträkning av polynomvärden. Multiplikationerna och additionerna görs i den ordning som anges med parenteserna här:

$$P(x) = ((((c_6 x + c_5)\, x + c_4)\, x + c_3)\, x + c_2)\, x + c_1$$

Uttrycket inom varje parentes är på formen $p\, x + c_k$. Det gör att hela beräkningen av polynomvärdena kan skrivas:

```
P=0; for k=n:-1:1, P=P.*X+c(k); end
```

Modellen med ett femtegradspolynom leder till en slät och fin kurva överallt men den får ganska kraftiga svängar mellan interpolationspunkterna (så kallat *Runges fenomen*), vilket vi kan se i den högra rutschbanefiguren. Som

rutschbana är det ingen idealisk modell; vid en åktur fastnar man troligen redan i den första dalen strax efter $x = 1$. Femtegradspolynomet blir

$$P(x) = 10.1429 - 23.2190x + 22.1667x^2 - 9.4905x^3 + 1.8476x^4 - 0.1333x^5.$$

Studera systemmatrisen (vandermondematrisen) lite närmare, den är helfylld med elementen:

```
1     0.5      0.25      0.125      0.0625      0.03125
1     1.5      2.25      3.375      5.0625      7.59375
1     2.5      6.25     15.625     39.0625     97.65625
1     3        9         27         81         243
1     4       16         64        256        1024
1     5       25        125        625        3125
```

Konditionstalet är drygt 10^5 vilket är ganska stort men inte katastrofalt ändå, eftersom beräkningarna utförs med femtonsiffrig precision. Vi har ju råd att förlora fem siffror och få kvar tio siffrors noggrannhet i polynomets koefficienter. I avsnitt 3.4 kommer vi att råka ut för ett interpolationsproblem där illakonditioneringen är mer förödande för resultatet.

Rutschbanans femtegradspolynom med Newtons formel

Newtons ansats (7) är räddningen om den naiva ansatsens matris blir alltför illakonditionerad. Det kan vara bra att ha en MATLAB-mall för algoritmen, därför visar vi hur rutschbanefallets entydigt bestämda femtegradspolynom erhålls med MATLAB:

```
% bana5n,  femtegradspolynom med Newtons ansats
  x=[0.5 1.5 2.5 3 4 5]';  y=[3 1.5 1.5 1 1 0]';
  ak=ones(6,1);  A=ak;                % bygg upp A kolumn för kolumn
  for kol=2:6, ak=ak.*(x-x(kol-1)); A=[A ak]; end
  c=A\y
  X=x(1):0.1:x(6);                    % beräkna P(X) med Horners algoritm
  P=0;  for k=6:-1:1, P=P.*(X-x(k))+c(k); end
  plot(x,y,'o', X,P), axis equal
```

Den triangulära matrisen och de beräknade koefficienterna blir i detta fall

```
A =   1     0      0        0         0         0
      1     1      0        0         0         0
      1     2      2        0         0         0
      1     2.5    3.75     1.875     0         0
      1     3.5    8.75    13.125    13.1250    0
      1     4.5   15.75    39.375    78.7500   78.75

c =   3.0000   -1.5000    0.7500   -0.5667    0.3143   -0.1333
```

Femtegradspolynomet skrivs nu

$$P(x) = 3.0000 - 1.5000(x-0.5) + 0.7500(x-0.5)(x-1.5) - \\ -0.5667(x-0.5)(x-1.5)(x-2.5) + 0.3143(x-0.5)(x-1.5)(x-2.5)(x-3) - \\ -0.1333(x-0.5)(x-1.5)(x-2.5)(x-3)(x-4)$$

och det är identiskt med det tidigare bestämda polynomet.

Vi kan konstatera att vi ännu inte funnit någon lämplig modell för rutschbanan. De båda första modellerna med brytpunkt vid x_3 eller x_4 duger ju inte på grund av kantigheten. Det interpolerande femtegradspolynomet har visserligen inga knyckar någonstans men kurvans yviga svängar mellan de givna punkterna (Runges fenomen) gör att den inte fungerar bra som rutschbana. Vi är inte nöjda än utan får söka efter en bättre modell!

Exempelsamlingens **Ex 5.1–5.6, 5.7a,b,c** *passar bra att öva på.*

3.3 Styckvis interpolation

Vi prövar så kallad styckvis interpolation där varje stycke utgörs av partiet mellan två på varandra följande interpolationspunkter. Enklast är *styckvis linjär interpolation* som vi inledde kapitlet med. Men vi skriver nu om formel (5) sedan vi först infört beteckningar för differenserna i x-led och y-led: $h_i = x_{i+1} - x_i$ och $\Delta y_i = y_{i+1} - y_i$.

$$P(x) = P(x_i + t \cdot h_i) = y_i + t \cdot \Delta y_i. \tag{8}$$

Storheten t är en lokal variabel som uppfyller $0 \le t \le 1$ för varje stycke mellan två interpolationspunkter. Sambandet mellan t och x är $t = (x - x_i)/h_i$. Resultatet av styckvis linjär interpolation är en kontinuerlig kurva, men den är kantig (diskontinuerlig derivata) vid interpolationspunkterna.

3.3.1 Hermiteinterpolation

Vid *hermiteinterpolation* utnyttjas förutom punkterna även information om lutningarna k_i i interpolationspunkterna (x_i, y_i).

Det styckvisa tredjegradspolynomet $P(x)$ som bestäms av villkoren

$$P(x_i) = y_i, \quad P(x_{i+1}) = y_{i+1}, \quad P'(x_i) = k_i, \quad P'(x_{i+1}) = k_{i+1}$$

kan skrivas med Hermites interpolationsformel. Inga okända koefficienter behöver bestämmas utan det är en explicit algoritm som för stycket mellan x_i

och x_{i+1} ser ut så här:

$$\begin{array}{l} P(x_i + t\cdot h_i) = y_i + t\cdot\Delta y_i + t(1-t)\, g_i + t^2(1-t)\, c_i \\ \text{där}\ \ g_i = h_i k_i - \Delta y_i \ \text{och}\ \ c_i = 2\Delta y_i - h_i(k_i + k_{i+1}). \end{array} \tag{9}$$

De två första termerna i uttrycket för P är samma som i formel (8), sedan tillkommer en andragrads- och en tredjegradsterm som båda ger bidraget noll då $t=0$ och $t=1$, dvs i de omgivande interpolationspunkterna. Därmed inses att villkoren $P(x_i) = y_i,\ \ P(x_{i+1}) = y_{i+1}$ är uppfyllda.

Genom derivering av $P(x) = P(x_i + t\cdot h_i)$ och insättning av $t=0$ och $t=1$ ska vi visa att de båda derivatavillkoren $P'(x_i) = k_i,\ \ P'(x_{i+1}) = k_{i+1}$ satisfieras. För stycket mellan x_i och x_{i+1} gäller

$$P'(x) = \frac{1}{h_i}(\Delta y_i + (1-2t)g_i + (2t-3t^2)c_i).$$

Sätt in $t=0$: $P'(x_i) = \frac{1}{h_i}(\Delta y_i + (h_i k_i - \Delta y_i)) = k_i$.
Sätt in $t=1$: $P'(x_{i+1}) = \frac{1}{h_i}(\Delta y_i - (h_i k_i - \Delta y_i) - (\, 2\Delta y_i - h_i(k_i + k_{i+1})) = k_{i+1}$.
Hermiteinterpolationsformeln (9) uppfyller alltså de fyra villkoren.

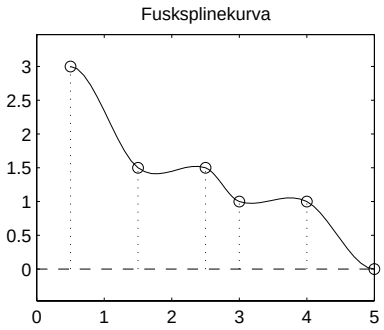
Differenskvot för lutningarna – fusksplinekurva

Ibland känner man till hur kurvan ska luta i början och i slutet, dvs k_1 och k_n är givna, men inget är känt om kurvans lutning vid de inre interpolationspunkterna. En god idé kan då vara att bestämma lutningen k_i som centraldifferenskvoten

$$k_i = \frac{y_{i+1} - y_{i-1}}{x_{i+1} - x_{i-1}}, \quad i = 2,\, 3, \ldots, n-1. \tag{10}$$

Geometriskt sett är det lutningen för linjen som går genom vänstra och högra grannpunkten. I MATLAB kan uttrycket (10) skrivas kompakt (se koden).

Vi prövar den här fusksplinemodellen för vår rutschbana och låter banan vara horisontell i början och slutet, alltså $k_1 = 0$, $k_6 = 0$. Det är lätt att beräkna $k_2, \ldots, k_5$ för hand med differenskvoten (10): $k_2 = -\frac{3}{4},\ \ k_3 = -\frac{1}{3},\ \ k_4 = -\frac{1}{3},\ \ k_5 = -\frac{1}{2}$.
Det blir en fullt acceptabel rutschbana utan tvära knyckar och med lagom stora svängar mellan interpolationspunkterna.



```
% Hermiteinterpolation, fallet fusksplines
% Rutschbanedata och stolpritning med stem som tidigare
  k=[0; (y(3:6)-y(1:4))./(x(3:6)-x(1:4)); 0];
  h=diff(x);  dy=diff(y);
  g=h.*k(1:5)-dy;  c=2*dy-h.*(k(1:5)+k(2:6));
  t=0:0.1:1;
  for i=1:5
    xt=x(i)+t*h(i);
    yt=y(i)+t*dy(i)+t.*(1-t)*g(i)+t.^2.*(1-t)*c(i);
    plot(xt,yt)
  end
  axis equal, title('Fusksplinekurva')
```

3.3.2 Interpolation med kubiska splines

Vi har nu följande djärva problem: Givet ett antal punkter (x_i, y_i) och ingen ytterligare information. Är det möjligt att göra styckvis interpolation och åstadkomma att både första- och andraderivatan är kontinuerliga vid polynombytena?

Omedvetna om numerikernas problem på detta område har de praktiskt arbetande konstruktörerna (skeppsbyggare m fl) sedan långa tider tillbaka grafiskt löst problemet. De har helt enkelt lagt en elastisk linjal (på engelska *spline*, på svenska *ri*) genom mätpunkterna som de markerat med stift på ritbrädet. Det blir en prydlig grafisk lösning på problemet.

Med elasticitetsteorin kan man beräkna ekvationen för den kurva som linjalen beskriver. Denna visar sig med god noggrannhet vara sammansatt av styckvisa tredjegradspolynom. Det blir byte av polynom vid varje interpolationspunkt, men med så mjuk övergång att både förstaderivatan och andraderivatan är kontinuerliga där. Kurvan kallas för en *kubisk splinekurva* och hela förfarandet utgör interpolation med kubiska splines.

Vi använder Hermites formel (9) som ansats för de styckvisa tredjegradspolynomen och skriver upp uttrycken för $P(x), P'(x)$ och $P''(x)$ för det i-te intervallet:

$$\begin{aligned} P(x_i + t\cdot h_i) &= \; y_i + t\cdot\Delta y_i + t(1-t)\, g_i + t^2(1-t)\, c_i \\ P'(x_i + t\cdot h_i) &= \; \tfrac{1}{h_i}(\Delta y_i + (1-2t)g_i + (2t-3t^2)c_i) \\ P''(x_i + t\cdot h_i) &= \; \tfrac{1}{h_i^2}(-2g_i + (2-6t)c_i) \\ \text{där}\;\; g_i = h_i k_i - \Delta y_i &\quad \text{och} \quad c_i = 2\Delta y_i - h_i(k_i + k_{i+1}). \end{aligned}$$

Önskemålet är att andraderivatan ska vara kontinuerlig vid det styckvisa polynombytet. Vid $x = x_i$ innebär detta: $P''(x_i + 0\cdot h_i) = P''(x_{i-1} + 1\cdot h_{i-1})$.

Lite formelmanipulerande behövs för vänsterledet och högerledet:

$$\begin{aligned} P''(x_i + 0 \cdot h_i) &= \tfrac{1}{h_i^2}(-2(h_i k_i - \Delta y_i) + 2(2\Delta y_i - h_i(k_i + k_{i+1}))) = \\ &= \tfrac{2}{h_i}(3\tfrac{\Delta y_i}{h_i} - 2k_i - k_{i+1}) \\ P''(x_{i-1} + 1 \cdot h_{i-1}) &= \tfrac{2}{h_{i-1}}(-3\tfrac{\Delta y_{i-1}}{h_{i-1}} + k_{i-1} + 2k_i) \end{aligned}$$

Sätter vi dessa uttryck lika erhålls

$$\frac{2}{h_i}(3\frac{\Delta y_i}{h_i} - 2k_i - k_{i+1}) = \frac{2}{h_{i-1}}(-3\frac{\Delta y_{i-1}}{h_{i-1}} + k_{i-1} + 2k_i)$$

som efter lite omformning ger följande samband för lutningarna i interpolationspunkterna:

$$h_i k_{i-1} + 2(h_i + h_{i-1})k_i + h_{i-1}k_{i+1} = 3\,(\frac{h_{i-1}}{h_i}\Delta y_i + \frac{h_i}{h_{i-1}}\Delta y_{i-1}). \qquad (11)$$

Totalt innebär det $n-2$ ekvationer för $i = 2,\, 3, \ldots, n-1$.

Naturlig splinekurva

Eftersom det behövs n ekvationer för entydig bestämning av de n obekanta k_i-värdena fordras två villkor till, ett vid $x = x_1$ och ett vid $x = x_n$. De kan väljas med hänsyn till tillämpningarna. Om vi går tillbaka till ursprungssituationen med den elastiska linjalen så är den ju rak utanför $x_1 \leq x \leq x_n$, det vill säga andraderivatan är noll för $x \leq x_1$ och för $x \geq x_n$.

Andraderivatan vid $x = x_1$ lyder

$$P''(x_1) = \frac{2}{h_1}(3\,\frac{\Delta y_1}{h_1} - 2k_1 - k_2).$$

När den sätts till noll får vi sambandet: $\quad 2h_1k_1 + h_1k_2 = 3\Delta y_1$.
Vid $x = x_n$ fås på motsvarande sätt: $\quad h_{n-1}k_{n-1} + 2h_{n-1}k_n = 3\Delta y_{n-1}$.
De här två villkoren kallas naturliga villkor och kurvan som byggs upp på det här viset med elastiska linjalen som förebild kallas för naturlig splinekurva.

Sambanden (11) och de båda extravillkoren leder till ett tridiagonalt ekvationssystem för bestämning av derivatorna k_i. Raderna två till och med den näst sista gäller alltid, medan första och sista raden är beroende av villkoren i ändpunkterna.

Modellen med naturliga splines tillämpad på rutschbanan ger

$$\begin{pmatrix} 2h_1 & h_1 & 0 & 0 & 0 & 0 \\ h_2 & 2(h_2+h_1) & h_1 & 0 & 0 & 0 \\ 0 & h_3 & 2(h_3+h_2) & h_2 & 0 & 0 \\ 0 & 0 & h_4 & 2(h_4+h_3) & h_3 & 0 \\ 0 & 0 & 0 & h_5 & 2(h_5+h_4) & h_4 \\ 0 & 0 & 0 & 0 & h_5 & 2h_5 \end{pmatrix} \begin{pmatrix} k_1 \\ k_2 \\ k_3 \\ k_4 \\ k_5 \\ k_6 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \end{pmatrix} \tag{12}$$

där $b_i = 3\left(\frac{h_{i-1}}{h_i}\Delta y_i + \frac{h_i}{h_{i-1}}\Delta y_{i-1}\right)$ för $i = 2,\ 3,\ 4,\ 5$.

Första och sista högerledskomponenterna blir vid naturliga splines:

$b_1 = 3\Delta y_1$ och $b_6 = 3\Delta y_5$.

Diagonalen i splinematrisen och de omgivande super- och subdiagonalerna kan skrivas så här

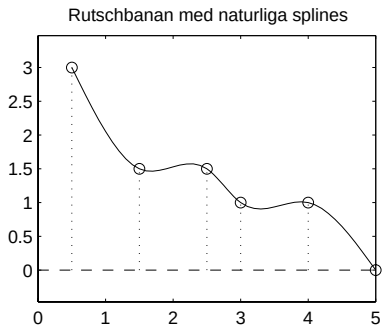
```
dm=2*(h(2:5)+h(1:4)); dia=[2*h(1); dm; 2*h(5)];
supd=[h(1); h(1:4)]; subd=[h(2:5); h(5)];
```

Högerledet i systemet ovan får följande uttryck

```
bm=3*(h(1:4).*dy(2:5)./h(2:5) + h(2:5).*dy(1:4)./h(1:4));
b=[3*dy(1); bm; 3*dy(5)];
```

För att lösa ekvationssystemet är det lämpligt att använda funktionen **tridia** som utnyttjar den tridiagonala strukturen. Med beteckningarna ovan erhålls lösningsvektorn till systemet (12) med anropet `k=tridia(dia,supd,subd,b);`

När k-vektorn är känd återstår ren hermiteinterpolation för att beräkna och rita en rutschbana av naturliga kubiska splines, som vi ser i figuren här.



Programmet som åstadkommer detta erhåller vi smidigast genom att i hermiteinterpolationskoden ersätta differenskvotberäkningen för `k` med satserna som beräknar lutningarna `k` för naturliga splines.

Värt att notera är att ekvationssystemet (12) gäller för det allmänna fallet med varierande avstånd h_i mellan interpolationspunkterna. I det ekvidistanta fallet med konstant avstånd h mellan punkterna får systemet för naturliga

kubiska splines utseendet:

$$\begin{pmatrix} 2 & 1 & 0 & 0 & \cdots & 0 \\ 1 & 4 & 1 & 0 & \cdots & 0 \\ 0 & 1 & 4 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & 4 & 1 \\ 0 & 0 & \cdots & 0 & 1 & 2 \end{pmatrix} \begin{pmatrix} k_1 \\ k_2 \\ k_3 \\ \vdots \\ k_{n-1} \\ k_n \end{pmatrix} = \frac{3}{h} \begin{pmatrix} \Delta y_1 \\ \Delta y_1 + \Delta y_2 \\ \Delta y_2 + \Delta y_3 \\ \cdots \\ \Delta y_{n-2} + \Delta y_{n-1} \\ \Delta y_{n-1} \end{pmatrix}$$

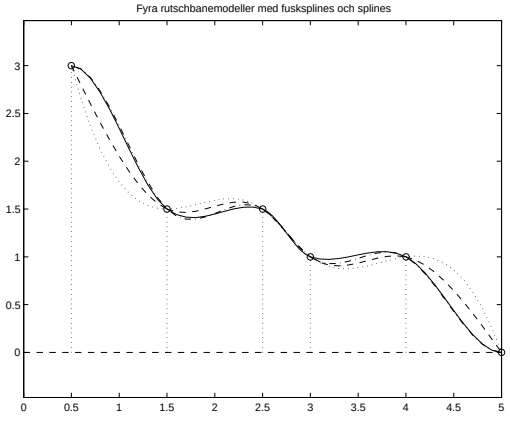
Kubisk splinekurva med givna ändlutningar

En rutschbana som börjar och slutar med horisontell lutning är barnvänligare än den bana som naturliga splinekurvan ger. För att åstadkomma en kurva av kubiska splines med ändvillkoren $k_1 = k_6 = 0$ behövs bara modifiering av första och sista raden i systemet (12).

Första raden kommer nu att bestå av ekvationen $k_1 = 0$ vilket innebär att diagonalelementet blir en etta och resten av matrisraden blir nollor; högerledskomponenten b_1 är 0. Motsvarande gäller för sista raden. Följande MATLAB-uppdatering ordnar till en modifierad rutschbanekurva av splines med derivatan noll i ändarna:

```
dia=[1; dm; 1]; supd=[0; h(1:4)]; subd=[h(2:5); 0];  b=[0; bm; 0];
```

Vi ritar in de styckvis beräknade rutschbanorna i samma figur för jämförelsens skull och konstaterar att de båda som har horisontell början och slut överensstämmer så väl att man med blotta ögat knappt ser någon skillnad. Det gäller modifierade splinekurvan (streckprickad) samt fusksplinekurvan som utnyttjar differenskvot för k_i (heldragen). Detta visar att centraldifferenskvot (10) kan vara ett enkelt och bra alternativ till den äkta splinefunktionens något mer komplicerade k-beräkning via ett ekvationssystem.



Rutschbanan konstruerad av naturliga kubiska splines är streckad i figuren.

Den prickmarkerade banan med stora lutningar i ändarna har erhållits med MATLABs inbyggda splinefunktion som anropas med tre parametrar så här:

```
x=[0.5 1.5 2.5 3 4 5]';  y=[3 1.5 1.5 1 1 0]';
X=x(1):0.1:x(6);  Pm=spline(x,y,X);  plot(X,Pm,':')
```

MATLAB-splines är nära släkt med naturliga kubiska splines, men skapas med en algoritm som utnyttjar andra villkor vid ändarna. Största avvikelsen mellan kurvorna finns därför i de yttersta intervallen, där MATLAB-splines svänger ut mer än naturliga splines.

> *I exempelsamlingen finns fem övningar på styckvis interpolation:* **Ex 5.7–5.11**, *alla rekommenderas.*

3.4 Hackigheter i kurvan, symptom på illakonditionering

Tabellvärden som utgör indata till interpolationsproblem eller andra kurvanpassningsproblem är sällan så tillrättalagda som i rutschbaneproblemet, där indata ligger väl fördelade mellan noll och fem.

Indata med många siffror
Vi vill studera om algoritmerna fungerar lika bra då interpolationspunkterna är förskjutna till $x = 1200.5, 1201.5, 1202.5, 1203, 1204, 1205$ (y-värdena oförändrade). Frågan är alltså hur denna förflyttning påverkar interpolationsmodellernas rutschbanekurvor, som nu beräknas för $1200 \leq x \leq 1205$.

Låt oss börja med fallet **interpolation med femtegradspolynom**. Här har vi två algoritmer att välja mellan: den naiva formen (6) och Newtons formel (7), beskrivna i programmen `bana5` och `bana5n`. Den enda ändringen är att talet 1200 ska adderas till x-vektorn.

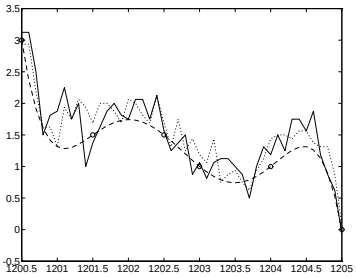
Bra algoritm:
Newtons ansats för polynomet ger i detta fall exakt samma triangulära matris och samma koefficienter som tidigare, eftersom algoritmen bygger på differenser mellan x-värdena, och dessa differenser inte har förändrats vid translationen. Femtegradspolynomet skrivs nu

$$P(x) = 3.0000 - 1.5000(x - 1200.5) + 0.7500(x - 1200.5)(x - 1201.5) - \ldots$$
$$-0.1333(x - 1200.5)(x - 1201.5)(x - 1202.5)(x - 1203)(x - 1204)$$

och det är lätt att räkna ut polynomvärden i området $1200 \leq x \leq 1205$. Rutschbanan blir identisk med den som beräknades före förskjutningen, vilket naturligtvis var vad vi hoppades på.

Dålig algoritm:
Vad händer om man råkar välja den naiva formen för femtegradspolynomet?

```
% Rutschbana med x-värdena förskjutna sträckan 1200
  x=1200+[0.5 1.5 2.5 3 4 5]';  y=[3 1.5 1.5 1 1 0]';
  A=[ones(6,1) x  x.^2  x.^3  x.^4  x.^5]
  c=A\y
  X=x(1):0.1:x(6);
  Pa=c(1)+c(2)*X+c(3)*X.^2+c(4)*X.^3+c(5)*X.^4+c(6)*X.^5;
  plot(x,y,'o', X,Pa)
```



Resultatet blir en mycket hackig rutschbanekonstruktion jämfört med den streckade rutschbanan som beräknats med Newtons interpolationsformel.
Vi vill utröna vad det oväntat hackiga resultatet beror på. Programmet innehåller tre faser som kräver granskning; det är i ordningsföljd: uppställning av matrisen;

beräkning av koefficienterna genom lösning av systemet $\mathbf{Ac} = \mathbf{y}$;

beräkning av polynomvärden.

Vandermondematrisens kolumner består av successivt ökande potenser av x-värdena. Storleken på elementen ökar från ett i första kolumnen till drygt 10^{15} i den sista. Matrisen blir mycket illakonditionerad vilket påverkar nästa beräkningsfas som i MATLAB är `c=A\y`. Inte ens femtonsiffrig datorprecision räcker till för att ge mer än några få siffrors noggrannhet i koefficienterna c_i. Vid körningen ger MATLAB en varning om att matrisen är nära singulär och den varningen ska tas på allvar.

En experimentell störningsanalys krävs för att vi ska få en uppfattning om antalet tillförlitliga siffror i c-värdena. En sådan ska vi göra i nästa avsnitt. Här skriver vi ut koefficienterna med tre siffror för att se att de är av så olika storleksordningar:
$c_1 = 3.24 \cdot 10^{14}$, $c_2 = -1.35 \cdot 10^{12}$, $c_3 = 2.24 \cdot 10^{9}$, $c_4 = -1.86 \cdot 10^{6}$, $c_5 = 774$, $c_6 = -0.129$.

Mysteriet med den hackiga femtegradskurvan

Rutschbanekurvan som utgörs av ett femtegradspolynom kan inte bli hackig av den enda anledningen att c-värdena avviker från de korrekta. En femtegradskurva beter sig snällt och har aldrig mer än två maxima och två minima. Den egenskapen borde bestå även hos polynomet med något avvikande koefficientvärden.

Teoretiskt gäller detta men i praktiken sker som synes värre saker. Något måste alltså gå på tok i själva polynomvärdesuträkningen:

```
Pa=c(1)+c(2)*X+c(3)*X.^2+c(4)*X.^3+c(5)*X.^4+c(6)*X.^5;
```

eftersom de interpolerade värdena (46 stycken med steget 0.1 i x-led) inte hamnar snällt på en femtegradskurva. Låt oss pröva om alternativet Horners algoritm för polynomberäkningen ger ett riktigare resultat:

```
Ph=0; for k=6:-1:1, Ph=Ph.*X+c(k); end, plot(X,Ph,':')
```

Nu erhålls den prickade kurvan som ser helt annorlunda ut men är precis lika mystiskt hackig!

Egentligen borde vi syna alla 46 polynomberäkningarna, men genom att fingranska alla beräkningssteg för *ett* utvalt x-värde kan vi kanske komma mysteriets förklaring lite närmare. Vi väljer $x = 1204$ som ju är en given interpolationspunkt med höjden $y = 1$. Resultatet vid interpolation borde alltså bli $P(1204) = 1$.

För polynomberäkningen prövas tre olika algoritmer:

```
% 1 Beräkning från lägsta till högsta grad
    p=c(1),  for k=1:5, term=c(k+1)*1204^k, p=p+term, end, p1=p
% 2 Beräkning från högsta till lägsta grad
    p=0;  for k=5:-1:0, term=c(k+1)*1204^k, p=p+term, end, p2=p
% 3 Beräkning med Horners algoritm
    p=0,  for k=6:-1:1, v=p*1204, ck=c(k), p=v+ck, end, p3=p
```

Den första algoritmen börjar med konstanttermen c_1 och adderar successivt termer av högre gradtal $c_2 \cdot 1204$, $c_3 \cdot 1204^2$, osv. Det är så som satsen `Pa= c(1)+c(2)*X+...` utförs, men genom att nu skriva den med en for-slinga kan vi skriva ut alla mellanresultat, se nedan.

I algoritm 2 görs räkningarna i omvänd ordning. Börja med $c_6 \cdot 1204^5$, addera $c_5 \cdot 1204^4$, $c_4 \cdot 1204^3$, osv. Algoritm 3 är Horners algoritm.

Resultatet som i samtliga fall borde bli 1.0000 (det korrekta värdet för $P(1204)$) blir i de tre fallen: `p1 = 1.1875, p2 = 1.2500, p3 = 1.4375.` och vi konstaterar att inget har mer än en korrekt siffra. Låt oss därför gå in i den första algoritmen och följa räkningarna steg för steg:

```
Mellanresultat i algoritm 1: Beräkning från lägsta till högsta grad
      p =  3.239972675639072e+14   term = -1.621642524744978e+15
      p = -1.297645257181071e+15   term =  3.246599555673318e+15
      p =  1.948954298492247e+15   term = -3.249916193767807e+15
      p = -1.300961895275560e+15   term =  1.626617481675020e+15
      p =  3.256555863994600e+14   term = -3.256555863994588e+14
      p =  1.187500000000000e+00
```

Koefficienternas och x-potensernas storlek gör att alla mellanresultat blir av storleksordningen 10^{15}. Det är inte förrän i sista beräkningssteget som termerna i stort sett tar ut varandra så att det slutliga värdet kommer ner till storleksordningen ett (där det hör hemma). Men det innebär ju en så kallad katastrofal *kancellation*, endast några få siffror återstår vid subtraktionen.

Om mellanresultaten skrivs ut i de båda övriga algoritmerna upptäcker man samma fenomen. Fram till och med näst sista steget är mellanresultaten mycket stora (cirka 10^{15}), för att i sista steget avslutas med en subtraktion av två nästan lika stora tal. Effekten blir att värdet abrupt kommer ner till rätt storleksordning men med en stor förlust av signifikanta siffror — en oundviklig kancellation.

I vårt fall tycks algoritmerna ge cirka en siffras noggrannhet av de numeriska experimenten att döma (tillförlitligheten i värdena `p1, p2, p3` ovan). Samtliga våra 46 beräknade polynomvärden kan ha en sådan avvikelse uppåt eller nedåt, och det förklarar kurvans hackighet.

Styckvis interpolation — slät eller hackig bana?
Styckvis interpolation för rutschbanan studerades i avsnitt 3.3. Hur påverkas de av en stor förskjutning av punkterna i x-led? Svaret är att alla algoritmer så som de beskrivs av formlerna (8) – (12) har samma ypperliga egenskap som Newtons ansats; de bygger enbart på differenserna h_i mellan x-värdena och på den lokala variabeln t. Att x-värdena finns borta vid 1200 istället för vid noll påverkar inte beräkningen av rutschbanekurvan. (Visa det!)

Visst finns det naiva, dåliga ansatser för styckvisa kubiska polynom också som innebär illakonditionering med tillhörande hackigheter i kurvan. Men varför välja en sådan när allt blir beräkningsmässigt enkelt och välkonditionerat med Hermites interpolationsformel (9).

3.5 Experimentell störningsanalys

3.5.1 Tillförlitlighet i polynomkoefficienterna

Vi vill försöka skatta hur många siffror som kan vara tillförlitliga i koefficienterna som erhålls som lösning till systemet $\mathbf{Ac} = \mathbf{y}$ i den naiva formen för femtegradspolynomet i förra avsnittet. Osäkerheten i resultatet beror på att matriselementen i datorn avrundas till maskinnoggrannhet och på att alla beräkningar utförs med samma begränsade datorprecision. En teoretisk störningsanalys kan bli mycket omfattande, medan en experimentell störningsräkning är lätt att utföra.

Stör ett matriselement (t ex `A(4,6)`) med ett litet relativfel av samma storlek som maskinnoggrannheten, i MATLAB finns `eps`$= 2.2{\cdot}10^{-16}$. Låt $\mathbf{c}_s$ vara lösning till systemet vid störd matris. Jämför $\mathbf{c}_s$ med den förut beräknade lösningen $\mathbf{c}$ till det ursprungliga systemet och beräkna relativfelet i utdata $\|\mathbf{c}_s - \mathbf{c}\|_\infty / \|\mathbf{c}\|_\infty$. En skattning av konditionstalet erhålls som kvoten mellan relativfelet i utdata och relativfelet $2.2 \cdot 10^{-16}$ i indata.

```
% Tillägg till bana5, nu med x=[1200.5  ...]
   ...; As=A;  As(4,6)=A(4,6)*(1+eps); cs=As\y, cdif=cs-c
   relfelc=norm(cdif,inf)/norm(c,inf),  kondtal=relfelc/eps
```

Den pyttelilla matrisstörningen på några enheter i sextonde siffran visar sig få stora effekter på femtegradspolynomets koefficienter, vars relativfel experimentellt skattas till 0.11, alltså elva procent. Det experimentella konditionstalet för koefficientberäkningen ligger så högt som $0.11/(2.2{\cdot}10^{-16}) = 5{\cdot}10^{14}$.

3.5.2 Tillförlitlighet i de beräknade polynomvärdena

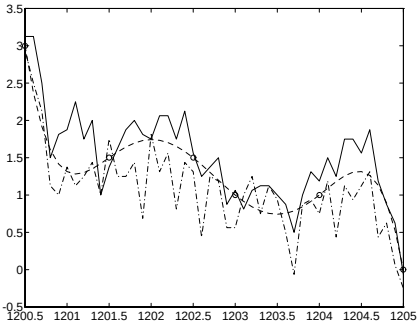
Polynomkoefficienterna är nästan aldrig slutmålet, beräkningen av dem är ju bara ett oundvikligt mellanled på vägen till $P(x)$, dvs polynomvärdet för ett visst x. Av betydligt större intresse än koefficienternas kondition är därför svaret på frågan: Hur känsligt är polynomvärdet för en liten matrisstörning? Lite tydligare frågeställning: Vad blir relativfelet i utdata $P(x)$ om indatastörningen består av en epsilonstörning i ett matriselement?

Låt oss för enkelhets skull studera polynomvärdet för samma $x = 1204$ som i förra avsnittet, med det korrekta värdet 1.0000. Vi fortsätter MATLAB-satserna med några rader:

```
u=1204;  pu=1;
ps=cs(1)+cs(2)*u+cs(3)*u^2+cs(4)*u^3+cs(5)*u^4+cs(6)*u^5   % naiva ansatsen
relfelp=abs(ps-pu)/pu,  kondtalp=relfelp/eps
Ps=cs(1)+cs(2)*X+cs(3)*X.^2+cs(4)*X.^3+cs(5)*X.^4+cs(6)*X.^5;
plot(X,Ps,'-.')                                          % kurvan streckprickad
```

Det numeriska experimentet ger polynomvärdet $p_s = 0.75$ i stället för det verkliga värdet 1.00, det innebär ett relativfel på 0.25. Konditionstalet hamnar på 10^{15} och noggrannhetsförlusten rör sig om femton siffror.

Att naiva ansatsens polynomvärden inte får mer än knappt en siffra korrekt visas mycket tydligt av rutschbanekurvorna i figuren.
Den streckprickade hackiga kurvan är resultatet efter den påtvingade matrisstörningen. Den heldragna är vår gamla hackiga kurva från föregående figur — även den är som vi vet påverkad av den ändliga datorprecisionen som lett till avrundningar av matriselementen.
Liksom i förra figuren har den streckade kurvan beräknats med Newtons välfungerande interpolationsformel och utgör korrekt jämförelsekurva.



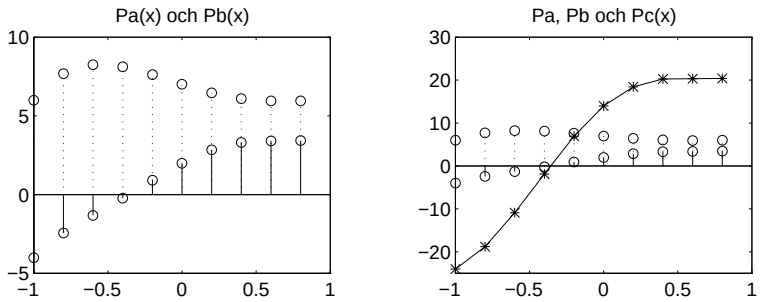
3.6 Tillämpning: Polynommultiplikation på smart sätt

Givet är koefficienterna i polynomen $P_a(x) = a_1 + a_2x + a_3x^2 + \ldots + a_{n+1}x^n$ och $P_b(x) = b_1 + b_2x + b_3x^2 + \ldots + b_{m+1}x^m$.
Beräkna koefficienterna i polynomet $P_c(x) = P_a(x) \cdot P_b(x)$ som blir ett polynom av gradtal $n+m$ med $N=n+m+1$ koefficienter.
Använd interpolationstekniken!

- Plocka ut $N=n+m+1$ stycken x-värden x_1, $x_2, \ldots,$ x_N
- Beräkna $P_a(x_i)$ och $P_b(x_i)$ med Horners algoritm
- Beräkna $P_c(x_i) = P_a(x_i) \cdot P_b(x_i)$
- Genom $N=n+m+1$ kända punkter finns ett entydigt bestämt polynom av gradtal $n+m$. Interpolation med naiva ansatsen ger koefficienterna.

```
% polymult
  clear, clf
  n=5;  a=[2 5 -3 -4 1 3];                  % Pa(x)=2+5x-3x^2-4x^3+x^4+3x^5
  m=4;  b=[7 -3 1 3 -2];                    % Pb(x)=7-3x+x^2+3x^3-2x^4
  N=n+m+1; x=0.2*(-n:m)';                   % välj ut N interpolationspunkter
  pa=0; for i=n+1:-1:1, pa=pa.*x+a(i); end
  pb=0; for i=m+1:-1:1, pb=pb.*x+b(i); end
  subplot(2,2,1), stem(x,pa), hold on       % rita Pa(x) i de N punkterna
  stem(x,pb,':'), title('Pa(x) och Pb(x)')  % rita Pb(x) i de N punkterna
  pc=pa.*pb;                                % beräkna Pc(x) i dessa punkter
  subplot(2,2,2), stem(x,pa), hold on
  stem(x,pb,':'), plot(x,pc,'*', x,pc)
  title('Pa, Pb och Pc(x)')
  Ak=ones(N,1);  A=Ak;                      % bygg upp matrisen A
  for kol=2:N, Ak=Ak.*x; A=[A Ak]; end
  c=A\pc                                    % koeff. i polynomet Pc(x)
```

Komponenterna i vektorn `c` blir 14, 29, −34, −8, 27, −5, −14, 14, 7, −6, alltså $P_c(x) = 14 + 29x - 34x^2 - 8x^3 + 27x^4 - 5x^5 - 14x^6 + 14x^7 + 7x^8 - 6x^9$.

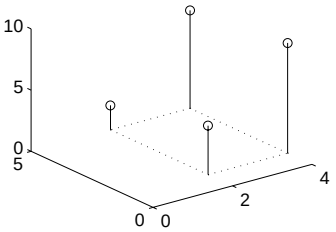


3.7 Flerdimensionell interpolation

Interpolationsmetoderna för envariabelfallet kan generaliseras till flera variabler. Vi ska nöja oss med interpolation i två variabler, när man i ett antal gitterpunkter i xy-planet har givna höjdvärden $z(x_i, y_j)$. Med lämplig form av interpolation vill vi beräkna höjdvärden för mellanliggande x- och y-värden. Problemet kan sägas vara att finna ett interpolerande tak över det aktuella området i xy-planet.

3.7.1 Bilinjär interpolation

Den enklaste interpolationsmodellen är så kallad bilinjär interpolation genom fyra punkter, t ex följande givna höjddata: $z(x_1, y_1) = 4$, $z(x_2, y_1) = 9$, $z(x_1, y_2) = 2$ och $z(x_2, y_2) = 8$. Höjderna antas vara givna för $x_1 = 2$, $x_2 = 4$, $y_1 = 1$, $y_2 = 5$.
Sökt är z-värdet när $x = 2.3$, $y = 3.6$.



Bilinjär interpolation betyder att linjär interpolation utförs i både x- och y-led. Rent praktiskt kan man interpolera i en variabel i taget.
Linjär interpolation i x-led vid $y = y_1 = 1$ ger

$$z(2.3, y_1) = z(x_1, y_1) + \frac{z(x_2, y_1) - z(x_1, y_1)}{x_2 - x_1}(x - x_1) = 4 + \frac{9-4}{2}0.3 = 4.75$$

och linjär interpolation i x-led vid $y = y_2 = 5$ ger

$$z(2.3, y_2) = z(x_1, y_2) + \frac{z(x_2, y_2) - z(x_1, y_2)}{x_2 - x_1}(x - x_1) = 2 + \frac{8-2}{2}0.3 = 2.9$$

Nu återstår att interpolera i y-led och det görs med

$$z(2.3, 3.6) = z(2.3, y_1) + \frac{z(2.3, y_2) - z(2.3, y_1)}{y_2 - y_1}(y - y_1) = 4.75 + \frac{2.9-4.75}{4}2.6 = 3.5475$$

Bilinjär interpolation kan också skrivas med ansatsen

$$P(x, y) = c_1 + c_2 x + c_3 y + c_4 xy$$

där de fyra koefficienterna bestäms ur sambandet $P(x_i, y_j) = z(x_i, y_j)$, dvs

$c_1 + c_2 x_1 + c_3 y_1 + c_4 x_1 y_1 = 4$, $\quad c_1 + c_2 x_2 + c_3 y_1 + c_4 x_2 y_1 = 9$,
$c_1 + c_2 x_1 + c_3 y_2 + c_4 x_1 y_2 = 2$, $\quad c_1 + c_2 x_2 + c_3 y_2 + c_4 x_2 y_2 = 8$,

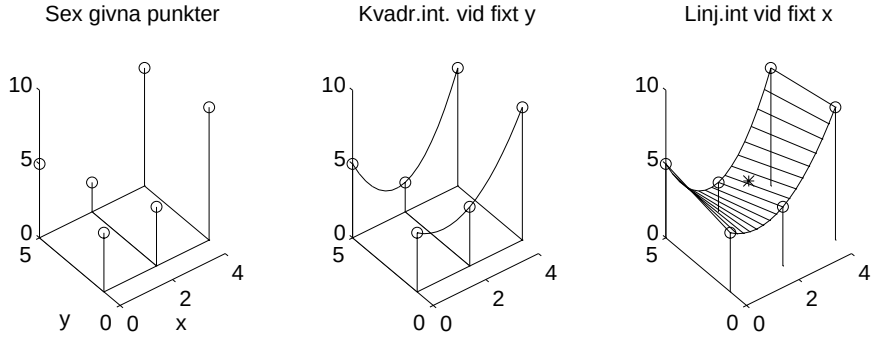
som med insatta x- och y-värden leder till ekvationssystemet

$$\begin{pmatrix} 1 & 2 & 1 & 2 \\ 1 & 4 & 1 & 4 \\ 1 & 2 & 5 & 10 \\ 1 & 4 & 5 & 20 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{pmatrix} = \begin{pmatrix} 4 \\ 9 \\ 2 \\ 8 \end{pmatrix}$$

Koefficienterna blir $c_1 = -0.25$, $c_2 = 2.375$, $c_3 = -0.75$, $c_4 = 0.125$. Det interpolerade värdet $P(x, y)$ vid $x = 2.3$, $y = 3.6$ blir 3.5475, vilket vi ju också fick förut. Eftersom man med linjär interpolation inte kan räkna med att få särskilt god noggrannhet är det nog lämpligt att avrunda värdet till 3.55. Men egentligen behöver vi tillgång till flera datapunkter (och mer information om exaktheten i de givna höjdvärdena) för att vi ska kunna bedöma resultatets tillförlitlighet.

3.7.2 Kvadratisk–linjär interpolation genom sex punkter

Till de fyra tidigare givna punkterna låter vi två nya höjdvärden tillkomma vid $x = 0$: $z(0, 1) = 4$, $z(0, 5) = 5$. Se vänstra figurens stolpar.



Genom sex givna höjdvärden kan man lägga en rymdyta genom att först göra kvadratisk interpolation vid fixt y, dvs interpolera med andragradspolynom i x-led dels vid $y = y_1$, dels vid $y = y_2$ — parabelkurvorna i mittersta figuren. Därefter utförs linjär interpolation tvärs över. Då hålls x fixt och räta linjer sammanbinder de båda parablerna med varandra.

Även i detta fall vill vi beräkna z-värdet vid $x = 2.3$, $y = 3.6$. Andragradspolynomet genom de tre punkterna som finns vid $y = y_1 = 1$ antar värdet 4.431 när $x = 2.3$. Andragradspolynomet genom de tre punkterna vid $y = y_2 = 5$ har för $x = 2.3$ värdet 2.326. Den linjära interpolationen i y-led ger därefter för $y = 3.6$ höjdvärdet 3.063 (asteriskmarkerat) som kan jämföras med det bilinjärinterpolerade värdet 3.55.

Beräkning och uppritning av den interpolerande rymdytan utförs av följande program.

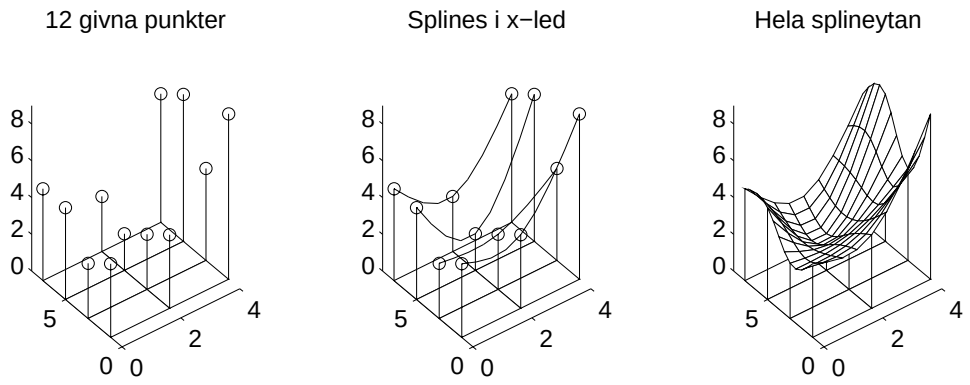
```
% linkvadyta, interpolera fram en yta genom sex punkter
  clear, clf
  x=[0 2 4]';  y=[1 5];                    % givna x- och y-värden
  z=[4  5                                  % höjder vid x=x1
     4  2                                  % höjder vid x=x2
     9  8];                                % höjder vid x=x3
  stem3([x x],[y;y;y],z), hold on          % rita stolpar
  A=[ones(3,1) x x.^2];                    % matrisen vid kvadr.interpol.
  c=A\z(:,1), d=A\z(:,2)                   % koeff i parabeln vid y1 resp y2
  xp=2.3; yp=3.6;
  z1=c(1)+c(2)*xp+c(3)*xp^2;               % parabelvärdet vid (xp,y1)
  z2=d(1)+d(2)*xp+d(3)*xp^2;               % parabelvärdet vid (xp,y2)
  zp=z1+(z2-z1)/(y(2)-y(1))*(yp-y(1))      % linj.interpol. i y-led
  plot3(xp,yp,zp,'*', 'Markersize',10)     % markera beräknade punkten

  X=0:0.1:4;                               % tät tabellering i x-led
  Z1=c(1)+c(2)*X+c(3)*X.^2;                % z-värden vid y=y1
  Z2=d(1)+d(2)*X+d(3)*X.^2;                % z-värden vid y=y2
  ett=ones(size(X));
  Y1=y(1)*ett; Y2=y(2)*ett;
  plot3(X,Y1,Z1,'r', X,Y2,Z2,'r')          % rita kantparablerna
  for j=1:length(X)
  plot3([X(j) X(j)],y,[Z1(j) Z2(j)]),end   % rita tvärribborna
```

3.7.3 Bikubisk interpolation — splineyta

Styckvis interpolation med tredjegradspolynom är ofta lämplig interpolationsform när antalet gitterpunkter i x- och y-led är större än i exemplet ovan. Låt oss ta ett exempel med tre x-värden, $x = 0, 2, 4$ och fyra y-värden, $y = 1, 3, 5, 7,$ dvs totalt tolv interpolationspunkter. Höjdvärdena i gitterpunkterna (stolparna i vänstra figuren) ligger i en matris med elementen $z_{ij} = z(x_i, y_j)$.

Vi vill beräkna och rita en rymdyta som interpolerar mjukt genom alla z-värdena. Det kan åstadkommas med kubiska splines eller med hermiteinterpolation med differenskvot för lutningarna (fusksplines). Man går tillväga på samma sätt som i förra exemplet och arbetar med en variabel i taget. Splinesinterpolationen görs alltså först i x-riktningen för konstant $y = y_j$.

Det innebär att höjdvärdena i matriskolumn nr j tillsammans med x-värdena är indata till splines- och hermiteinterpolationsalgoritmen, se avsnitt 3.3 formel (9), med lagom litet steg δt för den lokala variabeln t som går från 0 till 1.



Mittersta figuren visar kurvresultatet då naturliga splines har tillämpats i de fyra planen $y = y_j$ för höjdvärden givna av matrisen $\mathbf{z}$ nedan. Med fyra delintervall mellan successiva gitterpunkter i x-led gäller $\delta t{=}0.25$. Det betyder att x-vektorn förtätas till $x_{spl} = 0,\ 0.5,\ 1,\ 1.5,\ 2,\ 2.5,\ 3,\ 3.5,\ 4$.

Vi får nu en matris $\mathbf{Z}$ med splinesförtätade kolumner – nio element i var och en av de fyra kolumnerna.

$$\mathbf{z} = \begin{pmatrix} 4 & 2 & 5 & 5 \\ 4 & 3 & 2 & 3 \\ 9 & 5 & 8 & 7 \end{pmatrix} \qquad \mathbf{Z} = \begin{pmatrix} 4 & 2 & 5 & 5 \\ 3.7070 & 2.1914 & 3.7227 & 4.1484 \\ 3.5312 & 2.4062 & 2.6562 & 3.4375 \\ 3.5898 & 2.6680 & 2.0117 & 3.0078 \\ 4 & 3 & 2 & 3 \\ 4.8398 & 3.4180 & 2.7617 & 3.5078 \\ 6.0312 & 3.9062 & 4.1562 & 4.4375 \\ 7.4570 & 4.4414 & 5.9727 & 5.6484 \\ 9 & 5 & 8 & 7 \end{pmatrix}$$

Därefter utförs splineinterpolation i y-led på varje rad av matrisen $\mathbf{Z}$, nu tillsammans med den givna y-vektorn. Med samma diskretisering av den lokala variabeln ($\delta t = 0.25$) förtätas y-vektorn till $y_{spl} = 1,\ 1.5,\ 2, \ldots, 7$ (tretton värden) och den slutliga matrisen blir en nio gånger tretton matris. Den mjukt interpolerande rymdytan visas i högra figuren.

Det spelar ingen roll om man väljer att utföra splinesinterpolationen i omvänd ordning, det vill säga att ta y-led först och x-led efteråt. Resultatet blir exakt detsamma (vilket är lätt att konstatera genom ett numeriskt experiment, men är lite knepigare att bevisa).

3.8 Richardsonextrapolation

Differenskvot som derivataapproximation behandlades i avsnitt 1.3. Där visades att framåtdifferenskvoten

$$\Delta_h = \frac{f(a+h) - f(a)}{h}$$

har ett trunkeringsfel som är ungefärligen proportionellt mot steget h.

Om man väljer centraldifferenskvoten

$$D_h = \frac{f(a+h) - f(a-h)}{2h}$$

får man en bättre approximation till derivatan $f'(a)$ med ett trunkeringsfel proportionellt mot h^2. Man får dock inte låta h vara för litet i formlerna för då dominerar avrundningsfelet (på grund av kancellation).

Den information som taylorutvecklingen ger oss om felet är tillräcklig för att vi ska kunna trolla fram en noggrannare derivataapproximation. Vi tittar på framåtdifferenskvotens felformel dels med steget h, dels med steget $h/10$:

$$\begin{aligned} \Delta_h - f'(a) &\approx c \cdot h \\ \Delta_{h/10} - f'(a) &\approx c \cdot h/10 \end{aligned}$$

Multiplicera det undre sambandet med tio och subtrahera det övre, så erhålls $10\Delta_{h/10} - \Delta_h - 9f'(a) \approx 0$. Vi löser ut $f'(a)$ och får på så sätt en ny derivataskattning:

$$f'(a) \approx \Delta_{h/10} + \frac{\Delta_{h/10} - \Delta_h}{9}$$

Vi tillämpar denna formel på värdena för $h = 0.1$ och 0.01 i första tabellen i avsnitt 1.3 och får $f'(1) \approx 2.7319190 + \frac{2.7319190 - 2.8588418}{9} = 2.717816$. Jämförelse med det korrekta derivatavärdet 2.718282 visar att skattningen har fyra korrekta siffror.

Tillämpas formeln på värdena för $h = 0.01$ och 0.001 blir resultatet $f'(1) \approx 2.7196400 + \frac{2.7196400 - 2.7319190}{9} = 2.718276$ och ännu fler siffror är korrekta.

Det här är ett exempel på en teknik som kallas richardsonextrapolation. Samma teknik kan användas på centraldifferenskvotvärdena. Där är felet ungefärligen proportionellt mot h^2 och det leder till

$$D_h - f'(a) \approx c \cdot h^2, \quad D_{h/10} - f'(a) \approx c \cdot (h/10)^2$$

Multiplicera det andra sambandet med hundra och subtrahera det första, så erhålls $100D_{h/10} - D_h - 99f'(a) \approx 0$. Vi löser ut $f'(a)$ och den förbättrade derivataskattningen blir:

$$f'(a) \approx D_{h/10} + \frac{D_{h/10} - D_h}{99}$$

Liksom förut består uttrycket av $D_{h/10}$ plus en korrektionsterm vars täljare är differensen mellan det bättre och det sämre tabellvärdet. Korrektionstermens nämnare är $10^p - 1$, där p beror av trunkeringsfelets h^p-term, alltså $p = 2$ för centraldifferenskvoten. Vi tillämpar den här formeln på värdena för $h = 0.1$ och 0.01 i den andra tabellen i avsnitt 1.3 och får $f'(1) \approx 2.7183250 + (2.7183250 - 2.7228145)/99 = 2.718280$, som bara avviker två enheter i sjätte decimalen från det korrekta värdet.

Richardsonextrapolation — resultatförbättring i alla stegmetoder
I våra exempel här har steglängderna h och $h/10$ utnyttjats, men vanligare är att man successivt halverar steget, så att faktorn är två istället för tio. För att göra det mer allmänt låter vi faktorn vara Q.

Richardsonextrapolation är tillämpbar som efterbehandling för förbättring – eller som redskap för felskattning – vid alla *stegmetoder*, dvs numeriska metoder som diskretiserar med en viss steglängd h, och som (i teorin åtminstone) ger allt bättre approximation ju mindre steget h väljs.

Om metodens trunkeringsfelutveckling är $c_1h^{p_1} + c_2h^{p_2} + \cdots$ och om metoden gett värdet F_1 med steget h och värdet F_2 med steget h/Q kan man extrapolera ett bättre värde

$$F^* = F_2 + \frac{F_2 - F_1}{Q^{p_1} - 1}.$$

Extrapolationsförfarandet innebär elimination av trunkeringsfelets ledande term. Vid *upprepad richardsonextrapolation* används ett räkneschema av följande typ där $N_j = Q^{p_j} - 1$.

F	Δ/N_1	$F+\frac{\Delta}{N_1}$	Δ/N_2	$F^*+\frac{\Delta}{N_2}$	Δ/N_3	$F^{**}+\frac{\Delta}{N_3}$
F_1						
F_2	$(F_2-F_1)/N_1$	F_1^*				
F_3	$(F_3-F_2)/N_1$	F_2^*	$(F_2^*-F_1^*)/N_2$	F_1^{**}		
F_4	$(F_4-F_3)/N_1$	F_3^*	$(F_3^*-F_2^*)/N_2$	F_2^{**}	$(F_2^{**}-F_1^{**})/N_3$	F_1^{***}

4 BÉZIERKURVOR

4.1 Parameterkurvor

Många kurvor beskrivs enklast i parameterform, $\mathbf{r} = \mathbf{r}(t)$, där $\mathbf{r}(t)$ är en vektor som i det tvådimensionella fallet har komponenterna $x(t)$ och $y(t)$. Vi ska betrakta interpolerande parameterkurvor mellan de givna interpolationspunkterna $\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n$. Då man går längs kurvan från en punkt $\mathbf{p}_i$ till nästa punkt $\mathbf{p}_{i+1}$ löper den lokala parametern t från 0 till 1.
Linjär interpolation mellan $\mathbf{p}_1$ och $\mathbf{p}_2$ uttrycks med formeln

$$\mathbf{r}(t) = (1-t)\,\mathbf{p}_1 + t\,\mathbf{p}_2, \quad 0 \le t \le 1 \tag{13}$$

som i komponentform blir

$$\begin{cases} x(t) &= (1-t)\,x_1 + t\,x_2 = x_1 + t\,(x_2 - x_1) \\ y(t) &= (1-t)\,y_1 + t\,y_2 = y_1 + t\,(y_2 - y_1). \end{cases}$$

Både $x(t)$ och $y(t)$ är förstagradspolynom i t. En fördel med att ange kurvor i vektorform är att samma formler även kan tillämpas på rymdkurvor, med tillägget att kurvvektorn får en tredje komponent: $\mathbf{r} = (x(t),\, y(t),\, z(t))$.

Med ett parameterpolynom av andra graden menas att kurvvektorn $\mathbf{r}(t)$ är ett andragradspolynom i parametern t, vilket i sin tur innebär att både $x(t)$ och $y(t)$ är andragradspolynom. För ett vanligt ickeparametriskt polynom $y(x)$ av n-te graden gäller ju att förändringen i x-led alltid sker linjärt samtidigt som y-koordinaten förändras enligt n-tegradsuttrycket.

4.2 Bézierkurvor

Under de senaste tio åren har bézierkurvor blivit ett populärt verktyg för kurvkonstruktion och finns nu i de flesta CAD-system. Bézierkurvor är parameterpolynom uppbyggda av två interpolationspunkter och en eller flera så kallade styrpunkter.

4.2.1 Kvadratiska bézierkurvor

En *kvadratisk bézierkurva* mellan $\mathbf{p}_1$ och $\mathbf{p}_2$ konstrueras av interpolationspunkterna $\mathbf{p}_1$ och $\mathbf{p}_2$ samt ytterligare en punkt $\mathbf{b}$ enligt uttrycket

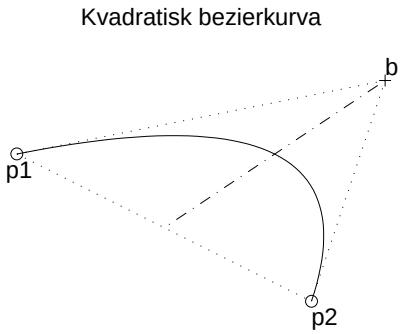
$$\mathbf{r}(t) = (1-t)^2\,\mathbf{p}_1 + 2t(1-t)\,\mathbf{b} + t^2\,\mathbf{p}_2, \quad 0 \le t \le 1. \tag{14}$$

Insättning av $t=0$ resp $t=1$ ger $\mathbf{r}(0) = \mathbf{p}_1$ och $\mathbf{r}(1) = \mathbf{p}_2$. Man bestämmer själv placeringen för styrpunkten $\mathbf{b}$, och den avgör hur kurvan kommer att löpa mellan de båda punkterna. Följande enkla konstruktion visar några viktiga egenskaper hos kurvan.

Dra triangeln $\mathbf{p}_1\,\mathbf{b}\,\mathbf{p}_2$! Kurvans startriktning vid $\mathbf{p}_1$ utgörs av riktningen för linjen $\mathbf{p}_1\,\mathbf{b}$, och kurvriktningen vid $\mathbf{p}_2$ överensstämmer med linjen $\mathbf{b}\,\mathbf{p}_2$. Ur formel (14) erhålls ett enkelt uttryck för punkten $\mathbf{r}(\frac{1}{2})$,

$$\mathbf{r}(\frac{1}{2}) = \frac{1}{2}\left(\frac{\mathbf{p}_1+\mathbf{p}_2}{2} + \mathbf{b}\right) \qquad (15)$$

som gör att punkten lätt kan konstrueras: Dra linjen från $\mathbf{b}$ till mittpunkten på $\mathbf{p}_1\,\mathbf{p}_2$.



Kurvan skär denna linje mitt itu, och just där har parametern t värdet $\frac{1}{2}$. Riktningen bestäms av att $\mathbf{r}'(\frac{1}{2}) = \mathbf{p}_2 - \mathbf{p}_1$ (derivera formel (14) och sätt in $t = \frac{1}{2}$). Kurvtangenten i denna punkt är alltså parallell med linjen $\mathbf{p}_1\,\mathbf{p}_2$.

4.2.2 Kubiska bézierkurvor

En *kubisk bézierkurva* mellan $\mathbf{p}_1$ och $\mathbf{p}_2$ definieras av fyra punkter – dels interpolationspunkterna $\mathbf{p}_1$ och $\mathbf{p}_2$, dels två styrpunkter $\mathbf{b}$ och $\mathbf{c}$. Det är en parametrisk tredjegradskurva och formeln lyder

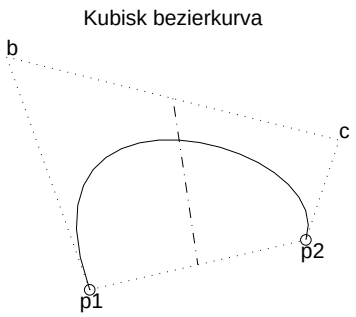
$$\mathbf{r}(t) = (1-t)^3\,\mathbf{p}_1 + 3t(1-t)^2\,\mathbf{b} + 3t^2(1-t)\,\mathbf{c} + t^3\,\mathbf{p}_2, \quad 0 \le t \le 1. \qquad (16)$$

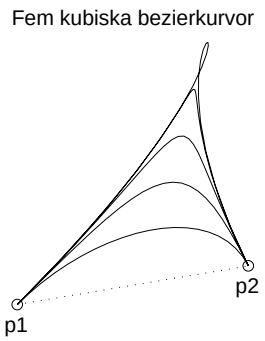
Kurvans startriktning vid $\mathbf{p}_1$ bestäms av linjen $\mathbf{p}_1\,\mathbf{b}$, och slutriktningen vid $\mathbf{p}_2$ bestäms av linjen $\mathbf{c}\,\mathbf{p}_2$. Även i det kubiska fallet har $\mathbf{r}(\frac{1}{2})$ och $\mathbf{r}'(\frac{1}{2})$ en enkel geometrisk innebörd,

$$\mathbf{r}(\frac{1}{2}) = \frac{1}{4}\,\frac{\mathbf{p}_1+\mathbf{p}_2}{2} + \frac{3}{4}\,\frac{\mathbf{b}+\mathbf{c}}{2}, \quad \mathbf{r}'(\frac{1}{2}) = \frac{3}{2}\left(\frac{\mathbf{p}_2+\mathbf{c}}{2} - \frac{\mathbf{p}_1+\mathbf{b}}{2}\right). \qquad (17)$$

Dra en linje som sammanbinder mittpunkterna på $\mathbf{p}_1\,\mathbf{p}_2$ och $\mathbf{b}\,\mathbf{c}$, se nästa figur. Bézierkurvan skär denna linje i proportionen $3:1$ enligt formel (17). Riktningen vid $t = \frac{1}{2}$ bestäms av riktningen på linjen som sammanbinder mittpunkterna på $\mathbf{p}_1\,\mathbf{b}$ och $\mathbf{c}\,\mathbf{p}_2$.

Om styrpunkterna ligger nära $\mathbf{p}_1$ och $\mathbf{p}_2$ erhålls en flack kurva. Ju längre bort de placeras ju buktigare blir kurvan. Man kan till och med åstadkomma en kurva med ögla om linjen $\mathbf{p}_1\,\mathbf{b}$ korsar linjen $\mathbf{c}\,\mathbf{p}_2$ och styrpunkterna ligger tillräckligt långt bortom skärningspunkten.





4.2.3 Bézierkurvor med givna ändriktningar

Givet två interpolationspunkter $\mathbf{p}_1$ och $\mathbf{p}_2$ samt kurvriktningar $\mathbf{k}_1$ och $\mathbf{k}_2$ i dessa punkter. Vi förutsätter att riktningsvektorerna är normerade och har längden ett. Det innebär att $\mathbf{k}_1^T\mathbf{k}_1 = 1$ och $\mathbf{k}_2^T\mathbf{k}_2 = 1$.

För att bézieruttrycket (16) ska kunna användas måste vi uttrycka styrpunkterna med hjälp av den kända informationen som utgörs av $\mathbf{p}_1$, $\mathbf{p}_2$, $\mathbf{k}_1$ och $\mathbf{k}_2$. I fallet kubisk bézierkurva är detta mycket enkelt. Enda kraven på styrpunkterna är att de uppfyller

$$\mathbf{b} = \mathbf{p}_1 + a_1\mathbf{k}_1 \text{ och } \mathbf{c} = \mathbf{p}_2 - a_2\mathbf{k}_2 \tag{18}$$

där a_1 och a_2 är positiva tal. Rent geometriskt är a_1-värdet avståndet från $\mathbf{p}_1$ till styrpunkten $\mathbf{b}$ och a_2-värdet avståndet från $\mathbf{p}_2$ till styrpunkten $\mathbf{c}$.

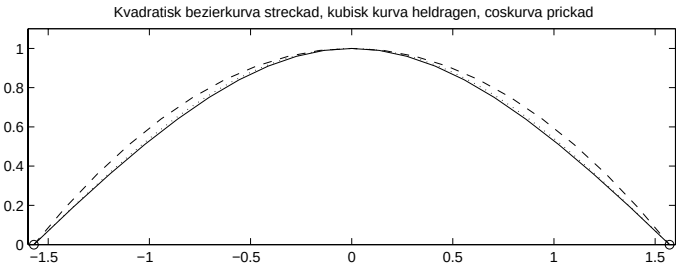
Det finns alltså två frihetsgrader kvar, vi kan välja styrpunktsavstånden a_1 och a_2 fritt. Sätt in uttrycken (18) i formeln (17) för $\mathbf{r}(\frac{1}{2})$ så erhålls

$$\mathbf{r}(\frac{1}{2}) = \frac{\mathbf{p}_1 + \mathbf{p}_2}{2} + \frac{3}{8}(a_1\mathbf{k}_1 - a_2\mathbf{k}_2). \tag{19}$$

Man har ibland kravet att kurvan för t ex $t = \frac{1}{2}$ ska passera genom en viss punkt; detta villkor är ofta tillräckligt för att bestämma styrpunktsavstånden a_1 och a_2. En tumregel annars är att välja a_1 och a_2 som $0.4 \cdot \| \mathbf{p}_2 - \mathbf{p}_1 \|_2$ för då brukar kurvan få en lagom buktad form.

4.2.4 Exempel: Cosinuskurva approximerad av bézierkurva

Vi vill undersöka hur väl en kvadratisk bézierkurva approximerar $y = \cos x$ i intervallet $-\frac{\pi}{2} \leq x \leq \frac{\pi}{2}$. Givet är $\mathbf{p}_1 = (-\frac{\pi}{2},\ 0)$ och $\mathbf{p}_2 = (\frac{\pi}{2},\ 0)$. Utnyttja egenskapen att bézierkurvans mittpunkt finns på halva avståndet upp till styrpunkten. Eftersom $\cos 0 = 1$ får vi direkt $\mathbf{b} = (0,\ 2)$. Bézierkurvan



erhålls genom formel (14): $\mathbf{r}(t) = (1-t)^2\,\mathbf{p}_1 + 2t(1-t)\,\mathbf{b} + t^2\,\mathbf{p}_2$, som för vardera $x(t)$ och $y(t)$ ger

$x(t) = (1-t)^2(-\frac{\pi}{2}) + t^2\frac{\pi}{2} = \pi\,(t-0.5), \quad y(t) = 4t(1-t).$

I MATLAB kan formel (14) skrivas i matris-vektorform så här (förutsatt att `t` är en kolumnvektor med värden från 0 till 1):

```
F2=[(1-t).^2  2*t.*(1-t)  t.^2];  r=F2*[p1; b; p2];
```

Pröva nu att approximera cosinuskurvan med en kubisk bézierkurva i samma intervall. Utnyttja symmetrin och också kunskapen om lutningarna i ändpunkterna: $y'(-\pi/2) = 1$ och $y'(\pi/2) = -1$.

Användbar egenskap för kubisk bézierkurva är att $\mathbf{r}(1/2)$ finns på 3/4 av avståndet upp till punkten $(\mathbf{b}+\mathbf{c})/2$ som alltså måste ha y-koordinaten 4/3. Av symmetriskäl ligger $\mathbf{b}$ och $\mathbf{c}$ på samma höjd. Eftersom lutningen i ändpunkterna är känd är styrpunkternas x-koordinater lätta att bestämma.

Vi får $\mathbf{b} = (-\pi/2 + 4/3,\ 4/3)$ och $\mathbf{c} = (\pi/2 - 4/3,\ 4/3)$. Sätt in i formel (16) och den kubiska bézierkurvan är klar. Överensstämmelsen med cosinuskurvan är så god att man med blotta ögat knappt ser någon avvikelse.

```
% cosbez,  coskurva approximerad av kvadratisk och kubisk bezierkurva
  v=-pi/2:pi/60:pi/2;  plot(v,cos(v),':'), hold on
  t=(0:0.05:1)';
  F2=[(1-t).^2  2*t.*(1-t)  t.^2];                      % kvadr. bezier
  F3=[(1-t).^3  3*t.*(1-t).^2  3*(1-t).*t.^2  t.^3];   % kubisk bezier

  p1=[-pi/2 0];  p2=[pi/2 0];  b=[0 2];
  r=F2*[p1; b; p2]; plot(r(:,1),r(:,2),'--')

% Kubisk bezierkurva
  b=[-pi/2+4/3  4/3];  c=[pi/2-4/3  4/3];
  r=F3*[p1; b; c; p2];  plot(r(:,1),r(:,2))
```

Matematiska uttryck för $x(t)$ och $y(t)$ behövs sällan i designsammanhang, kurvorna säger tillräckligt. Men formel (16) med insatta data och lite räknande ger: $x(t) = \frac{\pi}{2}(-1 + 6t^2 - 4t^3) + 4t(1 - 3t + 2t^2), \quad y(t) = 4t(1-t).$

4.2.5 Areaberäkning för kvadratisk bézierkurva

För kvadratiska bézierkurvor som visat sig ha enkla geometriska egenskaper kan ytterligare en trevlig egenskap fogas: Arean mellan linjen $\mathbf{p}_1\,\mathbf{p}_2$ och den kvadratiska bézierkurvan är två tredjedelar av arean av styrpunktstriangeln $\mathbf{p}_1\mathbf{b}\,\mathbf{p}_2$.

Bevis: Lägg koordinatsystemet med $\mathbf{p}_1$ i origo och $\mathbf{p}_1\,\mathbf{p}_2$ längs x-axeln. Låt $\mathbf{p}_2$ ha koordinaterna $(a,\,0)$ och $\mathbf{b} = (x_b,\,y_b)$. Arean av triangeln $\mathbf{p}_1\mathbf{b}\,\mathbf{p}_2$ är då $ay_b/2$.
Arean mellan bézierkurvan och x-axeln är värdet av integralen $|\int_0^1 y\dot{x}\,dt|$. Bézierkurvan blir i parameterform:

$$x(t) = 2t(1-t)x_b + a\,t^2 \;\text{ och }\; y(t) = 2t(1-t)y_b$$

och derivatan dx/dt blir $\dot{x} = 2(1-2t)x_b + 2at = 2(x_b + t(a-2x_b))$.

$$\text{Arean} = \int_0^1 y\dot{x}\,dt = 4y_b \int_0^1 t(1-t)(x_b + t(a-2x_b))\,dt =$$

$$= 4y_b\left[(\frac{t^2}{2} - \frac{t^3}{3})x_b + (\frac{t^3}{3} - \frac{t^4}{4})(a-2x_b)\right]_0^1 = 4y_b(\frac{1}{6}x_b + \frac{1}{12}(a-2x_b)) = \frac{1}{3}ay_b$$

vilket utgör precis 2/3 av triangelarean.

Area för kubisk bézierkurva: I integrationskapitlets avsnitt 5.2.5 visas en numerisk algoritm för arean mellan $\mathbf{p}_1\,\mathbf{p}_2$ och den kubiska bézierkurvan med $\mathbf{p}_1$ och $\mathbf{p}_2$ som start- och slutpunkt.

4.3 Styckvis interpolation med bézierkurvor

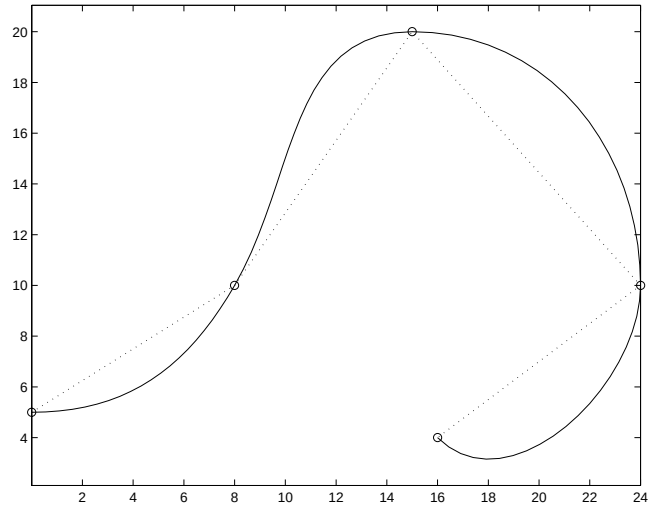
4.3.1 Styckvisa bézierkurvor med kontinuerliga tangentriktningar

Konstruktionsproblemet att finna en kurva som passerar genom många punkter $\mathbf{p}_1, \mathbf{p}_2, ..., \mathbf{p}_n$ och i dessa har givna kurvriktningar $\mathbf{k}_1, \mathbf{k}_2, ..., \mathbf{k}_n$ löser man genom att behandla ett kurvstycke i taget. I både kvadratiska och kubiska fallet får kurvorna kontinuerliga tangentriktningar vid kurvbytena.

Med styckvisa kubiska bézierkurvor går det att få ännu mjukare övergångar genom att välja styrpunktsavstånden lämpligt. En god tumregel är att vid bytet mellan (i–1)-a och i-te kurvan, dvs vid punkten $\mathbf{p}_i$, låta följande styrpunktsförhållande gälla:

$$\frac{a_{1,i}}{a_{2,i-1}} = \frac{\|\,\mathbf{p}_{i+1} - \mathbf{p}_i\,\|_2}{\|\,\mathbf{p}_i - \mathbf{p}_{i-1}\,\|_2}.$$

Om vi dessutom inom varje enskild bézierkurva låter de båda styrpunktsavstånden vara lika, alltså $a_{1,i} = a_{2,i}$, så leder detta styrpunktsförhållande till följande enkla formel: $a_{1,i} = a_{2,i} = \mu \cdot \| \mathbf{p}_{i+1} - \mathbf{p}_i \|_2$ med konstant buktfaktor μ (förslagsvis 0.4 enligt tumregeln tidigare).



Kurvan i figuren består av fyra styckvisa kubiska bézierkurvor genom fem givna punkter med fem givna riktningar, buktfaktorn har valts till 0.4.

```
% bez3fler, Styckvis kubiska bezierkurvor mellan fem punkter
  dt=0.05;  t=(dt:dt:1)';
  F=[(1-t).^3  3*t.*(1-t).^2  3*t.^2.*(1-t)  t.^3];
  p=[0 5; 8 10; 15 20; 24 10; 16 4];          % givna punkter
  plot(p(:,1),p(:,2),'o', p(:,1),p(:,2),':'), hold on
  k=[   1         0                           % normerade riktningar
    cos(pi/3)   sin(pi/3)
        1         0
        0        -1
    cos(3*pi/4) sin(3*pi/4)];
  my=0.4;                                     % buktfaktor
  rr=p(1,:);                                  % första punkten
  for i=1:4
    a1=my*norm(p(i+1,:)-p(i,:)),  a2=a1;      % styrpunktsavstånd
    b=p(i,:)+a1*k(i,:);
    c=p(i+1,:)-a2*k(i+1,:);
    r=F*[p(i,:); b; c; p(i+1,:)];             % i:te kurvan
    rr=[rr; r];                               % bygg på totalkurvan
  end
  plot(rr(:,1),rr(:,2)), axis equal
```

Om endast första och sista kurvriktningen är kända kan centraldifferensformeln $\mathbf{k}_i = (\mathbf{p}_{i+1} - \mathbf{p}_{i-1}) / \parallel \mathbf{p}_{i+1} - \mathbf{p}_{i-1} \parallel_2$ användas för de övriga riktningarna. Den enda ändringen som vi behöver göra i programmet är att ersätta uttrycket för $\mathbf{k}$ med

```
k=[1 0];                                          % känd riktning
for i=2:4
  ki=p(i+1,:)-p(i-1,:); k=[k; ki/norm(ki)];
end
k5=[cos(3*pi/4) sin(3*pi/4)];  k=[k; k5];         % känd riktning
```

Bézierkurveövningar finns i exempelsamlingen i **Ex 5.12–5.15**.

4.3.2 Styckvisa bézierkurvor med kontinuerliga derivator

Det går naturligtvis att räkna fram styrpunktsavstånden som ger mjukast möjliga övergång vid kurvbytena, och vi ska kortfattat beröra hur man går tillväga. Bézierkurvans parameter t är en *lokal parameter* som för varje enskild kurva löper mellan 0 och 1.

För att åstadkomma kontinuerlig derivata (kanske till och med kontinuerlig andraderivata) över kurvgränserna måste den sammansatta kurvan ha en *global* parameter — analogt med sambandet mellan lokala variabeln t och globala variabeln x vid vanlig ickeparametrisk styckvis interpolation (se avsnitt 3.3).

Vid parameterkurvor i vektorform innebär kontinuerlig derivata att inte bara kurvans tangentriktning utan även storleken på derivatavektorn – betraktad som funktion av den globala parametern – ska överensstämma då man byter kurva.

Låt oss i bézierfallet införa en global parameter u. Teoretiskt sett skulle båglängden vara bästa valet, med $u=0$ i första punkten och sedan u monotont växande fram till sista punkten. Men båglängden är alltför komplicerad beräkningsmässigt för att fungera bra, därför brukar man som global parameter u välja det monotont ökande avståndet längs den styckvis linjärinterpolerande polygonen som sammanbinder punkterna $\mathbf{p}_1, \mathbf{p}_2, ..., \mathbf{p}_n$. Det ger följande samband mellan global och lokal parameter i det i-te intervallet:

$$u = u_i + t \parallel \mathbf{p}_{i+1} - \mathbf{p}_i \parallel_2 \quad \text{där} \quad u_i = \sum_{j=1}^{i-1} \parallel \mathbf{p}_{j+1} - \mathbf{p}_j \parallel_2 .$$

Om man deriverar bézierkurveuttrycket med avseende på u, och sätter derivatan i $(i-1)$-a intervallets slutpunkt lika med derivatan i startpunkten av det i-te intervallet, så får man fram det styrpunktsförhållande för mjuka övergångar som föreslogs tidigare:

$$\frac{a_{1,i}}{a_{2,i-1}} = \frac{\| \mathbf{p}_{i+1} - \mathbf{p}_i \|_2}{\| \mathbf{p}_i - \mathbf{p}_{i-1} \|_2}.$$

4.3.3 Parametriska kubiska splinekurvor

Interpolerande parametriska kubiska splines kan byggas upp av verktyget kubiska bézierkurvor. Den givna informationen består här enbart av interpolationspunkterna $\mathbf{p}_1, ..., \mathbf{p}_n$. Riktningarna är inte kända i förväg utan $\mathbf{k}_1, ..., \mathbf{k}_n$ räknas fram genom villkoret att både första- och andraderivatorna $\mathbf{r}'(u)$ och $\mathbf{r}''(u)$ i den globala parametriseringen ska vara kontinuerliga i övergångarna, dvs vid varje interpolationspunkt $\mathbf{p}_i$. Detta leder till ett ekvationssystem för bestämning av styrpunktsavstånden (i varje intervall måste sambandet mellan u och den lokala parametern t beaktas).

Vid interpolation med parametriska kubiska splines går det bra att klara sig utan den vektorbaserade bézierkurverepresentationen. Då den globala parametern u införts, kan man helt enkelt använda den vanliga algoritmen för splinesinterpolation (se avsnitt 3.3.2), dels för x som funktion av u och dels för y som funktion av u. Resultatet blir en mjuk parameterkurva som representeras av $(x(u),\, y(u))$.

4.4 Icke-interpolerande B-splines

Vi ska studera en annan typ av styckvisa parametriska tredjegradspolynom, som är grafiskt verktyg i många CAD-system. Det är så kallade B-splines som formelmässigt påminner mycket om styckvisa kubiska bézierkurvor, men där interpolationskravet på alla inre punkter, $\mathbf{p}_2, \ldots, \mathbf{p}_{n-1}$, har tagits bort.

Man interpolerar bara i allra första punkten $\mathbf{p}_1$ och allra sista punkten $\mathbf{p}_n$, de övriga betraktas som ett slags styrpunkter. Formeln för ett kurvstycke av den mjukt buktande B-splinekurvan lyder

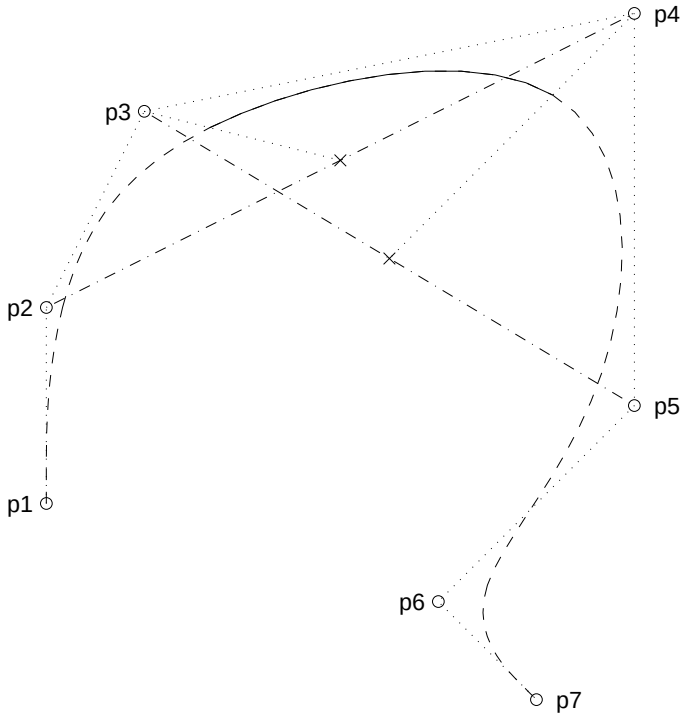
$$\mathbf{r}_i(t) = \frac{1}{6}\left((1-t)^3\,\mathbf{p}_{i-1} + (3t^3-6t^2+4)\,\mathbf{p}_i + (-3t^3+3t^2+3t+1)\,\mathbf{p}_{i+1} + t^3\,\mathbf{p}_{i+2}\right) \tag{20}$$

där t är lokal variabel som inom varje stycke går från noll till ett.

Vi ska med lite geometriska betraktelser få lite grepp om hur B-splines beter sig. I figuren är konstruktionslinjer gjorda för fallet $i=3$ som utnyttjar de fyra punkterna $\mathbf{p}_2$, $\mathbf{p}_3$, $\mathbf{p}_4$, $\mathbf{p}_5$. Insättning av $t = 0$ i formeln ovan ger

$$\mathbf{r}_i(0) = \frac{1}{6}(\mathbf{p}_{i-1} + 4\mathbf{p}_i + \mathbf{p}_{i+1}) = \frac{1}{3}\,\frac{\mathbf{p}_{i-1} + \mathbf{p}_{i+1}}{2} + \frac{2}{3}\,\mathbf{p}_i \tag{21}$$

Uttrycket $\frac{\mathbf{p}_{i-1}+\mathbf{p}_{i+1}}{2}$ är mittpunkten på linjen från $\mathbf{p}_{i-1}$ till $\mathbf{p}_{i+1}$. I figuren finns linjen $\mathbf{p}_2\,\mathbf{p}_4$ streckpunktmarkerad med ett kryss vid mittpunkten.



I uttrycket för $\mathbf{r}_i(0)$ har $\frac{\mathbf{p}_{i-1}+\mathbf{p}_{i+1}}{2}$ vikten $1/3$ och punkten $\mathbf{p}_i$ (dvs $\mathbf{p}_3$ i figuren) vikten $2/3$; det innebär att kurvpunkten $\mathbf{r}_3(0)$ finns på sammanbindningslinjen från mittpunkten av $\mathbf{p}_2\,\mathbf{p}_4$ till punkten $\mathbf{p}_3$ på två tredjedelar av sträckan, alltså där figurens heldragna kurvparti börjar.

Insättning av $t = 1$ i B-splinesformeln (20) ger

$$\mathbf{r}_i(1) = \frac{1}{6}(\mathbf{p}_i + 4\mathbf{p}_{i+1} + \mathbf{p}_{i+2}) = \frac{1}{3}\,\frac{\mathbf{p}_i + \mathbf{p}_{i+2}}{2} + \frac{2}{3}\,\mathbf{p}_{i+1} \tag{22}$$

som är samma uttryck som för $\mathbf{r}_i(0)$ men med index ökat med ett. Konstruktionslinjen $\mathbf{p}_3\,\mathbf{p}_5$ dras, mittpunkten sammanbinds med $\mathbf{p}_4$, och på två tredje-

delar av sträckan finns slutpunkten $\mathbf{r}_3(1)$ för detta stycke av B-splinekurvan (heldragna kurvstycket i figuren).

När vi går vidare till nästa kurvstycke bestäms $\mathbf{r}_{i+1}(0)$ av formel (21) men med index fyra i stället för tre. Det är identiskt med formel (22) och vi konstaterar alltså att nya kurvpartiet börjar där förra slutar — B-splinekurvan är kontinuerlig.

Hur är det med lutningen? Överensstämmer riktningsvektorn $\mathbf{r}_i'(1)$ i slutet av ett stycke med riktningsvektorn $\mathbf{r}_{i+1}'(0)$ i början av nästa?

Derivering av (20) och insättning av $t = 1$ ger $\mathbf{r}_i'(1) = 0.5(\mathbf{p}_{i+2} - \mathbf{p}_i)$. För stycket med $i = 3$ blir kurvriktningen i slutpunkten $0.5\,(\mathbf{p}_5 - \mathbf{p}_3)$, den är alltså parallell med linjen från $\mathbf{p}_3$ till $\mathbf{p}_5$ (kolla i figuren att det stämmer).

Med $t = 0$ insatt i B-splinederivataformeln fås $\mathbf{r}_i'(0) = 0.5\,(\mathbf{p}_{i+1} - \mathbf{p}_{i-1})$. Sätter vi in nästa kurvindex $i = 4$ här, så ser vi att derivatavärdena överensstämmer. Kurvriktningen är alltså kontinuerlig vid övergången från ett stycke till närmast följande.

Ytterligare en derivering ger oss $\mathbf{r}_i''(t)$ som är $\mathbf{p}_i - 2\mathbf{p}_{i+1} + \mathbf{p}_{i+2}$ för $t = 0$ och $\mathbf{p}_{i+1} - 2\mathbf{p}_{i+2} + \mathbf{p}_{i+3}$ för $t = 1$. Det blir vinstlott igen — även andraderivatan är kontinuerlig vid passagen mellan kurvstyckena.

Men hur fixar man till kurvsnuttarna i början och slutet för en B-spline? Genom den första punkten $\mathbf{p}_1$ ska B-splinekurvan interpolera, dvs $\mathbf{r}(0) = \mathbf{p}_1$ ska vara uppfyllt. Det åstadkommer man genom att skapa två kopior $\mathbf{p}_{-1}$ och $\mathbf{p}_0$ av punkten $\mathbf{p}_1$, så att de tre första punkterna i formeluttrycken (20), (21) och (22) alla är $\mathbf{p}_1$. Vi kontrollerar om det stämmer för formel (21): $\mathbf{r}_0(0) = \frac{1}{6}(\mathbf{p}_{-1} + 4\mathbf{p}_0 + \mathbf{p}_1) = \mathbf{p}_1$ som önskat.

Det allra första lilla kurvstycket – som vi låter ha index noll – börjar i punkten $\mathbf{p}_1$ och kommer enligt formel (22) att sluta i punkten $\mathbf{r}_0(1) = \frac{1}{6}(\mathbf{p}_0 + 4\mathbf{p}_1 + \mathbf{p}_2) = \frac{5}{6}\mathbf{p}_1 + \frac{1}{6}\mathbf{p}_2$.

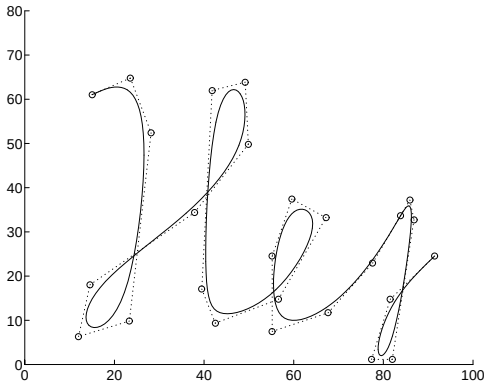
På motsvarande sätt måste man avsluta hela B-splinealgoritmen med att skapa två kopior av sista punkten $\mathbf{p}_n$. De utnyttjas i formel (20) för att åstadkomma att kurvan landar i punkten $\mathbf{p}_n$ (se MATLAB-programmet).

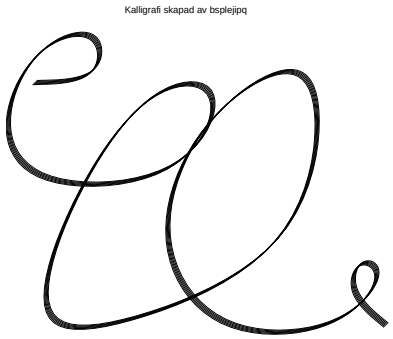
Vi kan konstatera att bézierkurvorna med sina trevliga geometriska egenskaper har hård konkurrens av B-splines. Det är ju faktiskt ganska enkelt att geometriskt konstruera läget och riktningen för B-splinekurvan vid varje nytt kurvstycke.

Pröva följande MATLAB-program (kurskatalogfilen `bsplejip.m`) där punkt efter punkt klickas in och B-splinekurvan snällt kommer smygande efter.

```
% bsplejip, B-splines med ginput
  clear, clf
  t=(0:0.1:1)';
  F=[(1-t).^3  3*t.^3-6*t.^2+4  -3*t.^3+3*t.^2+3*t+1  t.^3]/6;
  axis([0 100 0 80]), hold on
  set(gcf,'DoubleBuffer','on')
  disp(['Klicka punkter (bryt med musens mittknapp)'])
  [x,y]=ginput(1); plot(x,y,'o'), p1=[x y];
  [x,y]=ginput(1); plot(x,y,'o', [p1(1) x],[p1(2) y],'m:')
  p=[x y];
  P=[p1; p1; p1; p];                  % två kopior av första punkten behövs
  r=F*P; plot(r(:,1),r(:,2))
  for i=1:48                          % maxantal punkter är 50
    [x,y,button]=ginput(1);
    if button==2, break
    else plot(x,y,'o', [p(1) x],[p(2) y],'m:')
      p=[x y];  P=[P(2:4,:); p];      % utnyttja de fyra senaste punkterna
      r=F*P; plot(r(:,1),r(:,2))
    end
  end
  for i=1:2                           % avsluta med två kopior av slutpunkten
    P=[P(2:4,:); p];
    r=F*P; plot(r(:,1),r(:,2))
  end
```

Det modifierade programmet `bsplejipq` snyggar till och lägger på lite kalligrafieffekt.





5 NUMERISK INTEGRATION

5.1 Allmänt

Alla formler för numerisk beräkning av integralvärdet $I = \int_a^b f(x)\,dx$ kan skrivas som en summa av N stycken viktade funktionsvärden där vikterna w_i och x-värdena x_i är kända, alltså

$$I = \int_a^b f(x)\,dx \approx \sum_{i=1}^{N} w_i f(x_i). \tag{23}$$

Hit hör Newton-Cotes integrationsformler som bl a innefattar trapetsregeln, Simpsons formel och MATLABs `quad`-funktion. Även gausskvadraturformlerna kan skrivas på den allmänna formen (23), alla med positiva vikter w_i.

5.2 Trapetsregeln med släktingar

5.2.1 Trapetsregeln och Simpsons formel

Trapetsregeln för beräkning av $I = \int_a^b f(x)dx$ lyder

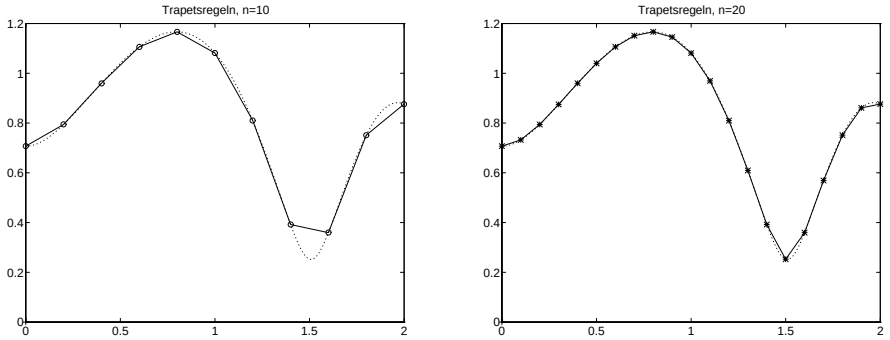
$$I \approx T(h) = h\left(\tfrac{f(a)}{2} + f(a+h) + f(a+2h) + \ldots + f(a+(n-1)h) + \tfrac{f(b)}{2}\right)$$

där $h = (b-a)/n$, och n är antalet delintervall. Det är en $(n+1)$-punktsformel med konstant avstånd h mellan x-värdena. Trapetsregeln på formen (23) uppfyller $N = n+1$ och alla vikter w_i är h utom första och sista som är $h/2$. Med funktionsvärdena skrivna i en vektor med komponenterna $f_1, f_2, \ldots, f_{n+1}$ blir trapetsregeln

$$T(h) = h\left(\sum_{i=1}^{n+1} f_i - \frac{f_1 + f_{n+1}}{2}\right). \tag{24}$$

I MATLAB skrivs formeln `Th=h*(sum(f)-(f(1)+f(n+1))/2)`.
För att få ett bättre resultat fördubblar vi antalet intervall och beräknar ett nytt trapetsregelvärde, nu med halverat steg, dvs h ersätts av $h/2$.

Trapetsregeln approximerar arean mellan x-axeln och integrandkurvan med en summa av parallelltrapetsareor. I ett allmänt fall med varierande steg h_i i x-led blir uttrycket för trapetsregeln $T = \sum_{i=1}^{n} \frac{h_i}{2}(f_i + f_{i+1})$.



Exempel: Beräkna med trapetsregeln en approximation till integralvärdet

$$I = \int_0^2 \sqrt{0.5 + 2e^{-x}\sin 2x^2}\, dx$$

Först skriver vi en funktion i MATLAB för integranden. Filen får heta **fq.m**:

```
function f=fq(x)
   f=sqrt(0.5+2*exp(-x).*sin(2*x.^2));
```

Rita upp integranden och utnyttja trapetsregeln två gånger, dels med tio intervall, alltså $n=10$, dels med $n=20$.

```
t=0:0.02:2;  ft=fq(t);
n=10, h=2/n;  x=h*(0:n); f=fq(x);  T10=h*(sum(f)-(f(1)+f(n+1))/2)
subplot(2,2,1), plot(0,0,'+', t,ft,':', x,f,'o', x,f)

n=2*n, h=h/2; x=h*(0:n); f=fq(x);  T20=h*(sum(f)-(f(1)+f(n+1))/2)
subplot(2,2,2), plot(0,0,'+', t,ft,':', x,f,'*', x,f)
```

Tio intervall ger resultatet 1.6426, och efter intervallfördubbling erhålls värdet 1.6418. Av resultaten att döma har integralvärdet beräknats med ungefär fyra siffrors noggrannhet, och vi kan svara $I = 1.642$.

Men noggrannheten ska ofta vara betydligt bättre än så. Låt oss kräva att integralvärdet ska beräknas med sju korrekta decimaler. Vi ska undersöka några olika numeriska metoder för att uppnå detta.

Först prövar vi trapetsregeln med successiv fördubbling av antalet intervall — vilket innebär steghalvering från h till $h/2$; vi benämner trapetsresultaten $T(h)$ respektive $T(h/2)$.

Trunkeringsfelet, alltså avvikelsen mellan trapetsregelvärdet $T(h)$ och det exakta integralvärdet I, kan skrivas som en utveckling

$$T(h) - I = c_1 h^2 + c_2 h^4 + \ldots$$

(se vidare avsnitt 5.2.3). Det innebär att richardsonextrapolation kan förbättra resultatet, metoden kallas då Rombergs metod. Låt oss börja med att dela integrationsintervallet i 40 delar.

```
n=40,  h=2/n; x=h*(0:n); f=fq(x); T =h*(sum(f)-(f(1)+f(n+1))/2);
n=2*n, h=h/2; x=h*(0:n); f=fq(x); Tt=h*(sum(f)-(f(1)+f(n+1))/2);
S=Tt+(Tt-T)/3;                              % extrapolera till simpsonvärde
n=2*n, h=h/2; x=h*(0:n); f=fq(x); Ttt=h*(sum(f)-(f(1)+f(n+1))/2);
St=Ttt+(Ttt-Tt)/3;                          % extrapolera fram nytt S
d=St-S;  B=St+d/15;                         % extrapolera på S-värdena
format long
richschema=[T    0    0
            Tt   S    0
            Ttt  St   B]
```

Utskriften följer schemat för richardsonextrapolation i avsnitt 3.8.

```
         T             S             B
   1.642000905         0             0
   1.642053159   1.642070577         0
   1.642066167   1.642070503   1.642070498
```

Integralvärdet kan anges $I = 1.6420705$ med en osäkerhet skattad till en enhet i sjunde decimalen (sista differensen i S-kolumnen).

I första kolumnen finns trapetsregelvärdena för $n = 40, 80, 160$. Därefter följer i andra och tredje kolumnen de extrapolerade värdena

$$S = T(h/2) + \frac{T(h/2) - T(h)}{2^2 - 1} \quad \text{och} \quad B = S(h/2) + \frac{S(h/2) - S(h)}{2^4 - 1}.$$

Variabelnamnet S står för *Simpson*, medan B står för *bäst* eller för *Boole* (logiker–matematiker som fick både Boolesk algebra och Booles integrationsregel uppkallad efter sig).

Simpsonvärdet $S(h)$ är en god approximation till integralen I och kan erhållas på två sätt, antingen via trapetsregeln med en extrapolation, eller som följande explicita formel som kallas Simpsons formel:

$$S(h) = \frac{h}{3}\left(f(a)+4f(a+h)+2f(a+2h)+4f(a+3h)+2f(a+4h)+\ldots+4f(b-h)+f(b)\right).$$

Antalet intervall n måste vara jämnt. Simpsons formel betraktad på den allmänna formen (23) har som synes de yttersta vikterna $h/3$ och de inre varannan gång $4h/3$ och varannan $2h/3$.

Simpsons formel har fjärde ordningens noggrannhet, det vill säga för trunkeringsfelet gäller $S(h) - I = O(h^4)$. Det motiverar varför extrapolationsuttrycket för B ovan har en fyra i nämnarens exponent.

Experimentell felskattning
Någon tillämpbarbar teoretisk formel för erhållande av ett numeriskt värde på absolutfelet i en integralapproximation finns inte. I praktiken arbetar man med successiv steghalvering i den valda algoritmen och gör en enkel experimentell felskattning (ofta lite för pessimistisk): Felet skattas som beloppet av differensen mellan de två sist beräknade värdena.

Fördelen med trapetsregeln med steglängdshalveringar är att algoritmen (24) är så enkel. För de flesta integralberäkningar som uppstår i tillämpningarna fungerar metoden tillräckligt bra. Men om integrandkurvan har snabba variationer i integrationsintervallet kan det krävas mycket små steg för att önskad noggrannhet ska uppnås. Man kan pröva med att dela upp integrationsintervallet $[a, b]$ i några delintervall och använda trapetsregeln med steghalveringar på varje del. Det finns dock andra numeriska integrationsmetoder som kan vara bättre att ta till i sådana fall.

5.2.2 Matlabs quad-funktioner

I MATLAB finns funktionerna `quad` (som bygger på Simpsons formel) och `quadl` för beräkning av enkelintegraler över ett ändligt integrationsintervall. Båda är så kallade adaptiva metoder; med adaptiv menas att metoden har inbyggd steglängdskontroll och minskar steget där integrandkurvan har kraftig krökning. Gör `help quad` och `help quadl` för mer information.

Toleransen i `quad`-algoritmerna avser absolutfel. Standardtoleransen är satt till 10^{-6}, vilket innebär att ungefär sex korrekta decimaler erhålls. För vårt exempel med intervallgränserna 0 och 2 blir anropet helt enkelt:

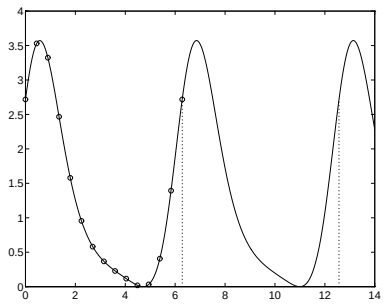
```
I=quad(@fq,0,2)
```

(Skriv `format long` först så erhålls utskriften med många decimaler.)
Vi kan själva ange felgräns för absolutfelet genom att ha med en fjärde parameter. Med följande anrop bör resultatet få sju korrekta decimaler:

```
I=quad(@fq,0,2,0.5e-7)
```

Om integrandfunktionen inte är mer komplicerad än att den kan skrivas på en rad är redskapet `inline` utmärkt (gör `help inline`), då behövs ingen m-fil för integranden.

```
I=quad(inline('sqrt(0.5+2*exp(-x).*sin(2*x.^2))'), 0,2,0.5e-7)
```



5.2.3 Periodiska integrander – lös med trapetsregeln

Betrakta $I = \int_0^{2\pi} (1+\sin x)e^{\cos x}dx$. Den har en periodisk integrand och integrationsintervallet är en hel period. För problem av den här typen, alltså integration över en hel period av en äkta periodisk funktion, är trapetsregeln (24) med successiv steghalvering — men *utan extrapolation* — den allra bästa lösningsmetoden. I detta fall uppstår nämligen inte den felutveckling efter potenser av h^2 som annars skulle motivera richardsonextrapolation!

Detta förklaras av *Euler–Maclaurins formel* för sambandet mellan integralvärdet I och trapetssumman $T(h)$:

$$I = T(h) - \frac{f'(b)-f'(a)}{12}h^2 + \frac{f'''(b)-f'''(a)}{720}h^4 - \frac{f^{(5)}(b)-f^{(5)}(a)}{30240}h^6 + O(h^8). \quad (25)$$

Periodicitet med periodlängd $b-a$ innebär ju att funktionen upprepar sig, dvs $f^{(k)}(b) = f^{(k)}(a)$ för alla k. Detta innebär i sin tur att konstanterna framför h-potenserna försvinner. Formel (25) är en så kallad asymptotisk formel som är giltig för små h i förhållande till periodlängden $b-a$. Felet $I-T(h)$ närmar sig noll snabbare än varje potens av h. (Det tycks som om felet alltid borde bli exakt noll, men högerledet konvergerar inte alltid mot vänsterledet i en oändlig asymptotisk serie. Något enkelt uttryckt: summan av oändligt många nollor blir inte säkert noll.)

Med trapetsregeln erhålls ofta full datornoggrannhet med färre än femtio delintervall. (En riktigt knölig integrandkurva kan kräva ett större antal intervall, dvs mindre h.) För att förvissa sig om att noggrannheten är uppnådd gör man en eller flera steglängdshalveringar och jämför trapetsresultaten.

För integralen ovan ger trapetsregeln med 14 intervall värdet 7.95492652101285. Exakt samma resultat erhålls med $n=28$, datorprecision har nåtts.

Öva numerisk integration i exempelsamlingens **Ex 6.1–6.8**.

5.2.4 Integration av hermiteinterpolationspolynom och splines

När integrandfunktionen $f(x)$ är uppbyggd av styckvisa tredjegradspolynom (hermiteinterpolationspolynom – fusksplines, splines) kan integralvärdet erhållas som ett enkelt uttryck i funktionsvärdena y_i och de kända lutningarna k_i. Det gäller nämligen att integralvärdet I_i för det i-te intervallet blir $I_i = \frac{h_i}{2}(y_i + y_{i+1}) + \frac{h_i^2}{12}(k_i - k_{i+1})$. Hela integralen från x_1 till x_N (vi har n intervall och $N = n+1$ punkter) kan då skrivas

$$\int_{x_1}^{x_N} f(x)\,dx = \sum_{i=1}^{n} I_i = \sum_{i=1}^{n} \left(\frac{h_i}{2}(y_i + y_{i+1}) + \frac{h_i^2}{12}(k_i - k_{i+1}) \right).$$

Härledning: Integrera hermiteinterpolationspolynomet (formel (9) i avsnitt 3.3.1) från x_i till x_{i+1} vilket med substitution till den lokala variabeln t med sambandet $x = x_i + t \cdot h_i$ och $dx = h_i\,dt$ leder till:

$$I_i = \int_{x_i}^{x_{i+1}} f(x)\,dx = \int_0^1 f(x_i + t\,h_i)\,h_i\,dt = h_i \int_0^1 (y_i + t\Delta y_i + (t - t^2)g_i + (t^2 - t^3)c_i)\,dt =$$

$$= h_i \left[t \cdot y_i + \frac{t^2}{2}\Delta y_i + (\frac{t^2}{2} - \frac{t^3}{3})g_i + (\frac{t^3}{3} - \frac{t^4}{4})c_i \right]_0^1 = h_i(y_i + \frac{1}{2}\Delta y_i + \frac{1}{6}g_i + \frac{1}{12}c_i)$$

Sätter vi här in uttrycken för g_i och c_i så erhålls

$$I_i = h_i \left(y_i + \frac{1}{2}\Delta y_i + \frac{1}{6}(h_i k_i - \Delta y_i) + \frac{1}{12}(2\Delta y_i - h_i k_i - h_i k_{i+1}) \right) \implies$$

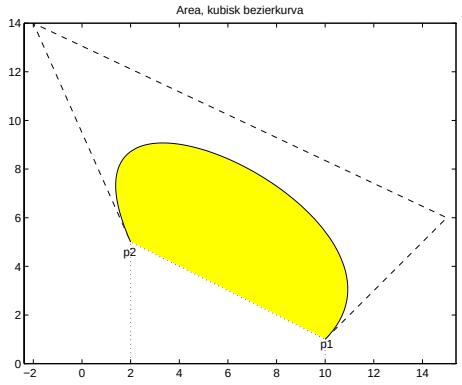
$$I_i = \frac{h_i}{2}(y_i + y_{i+1}) + \frac{h_i^2}{12}(k_i - k_{i+1}).$$

Den första delen är trapetsregelns uttryck och den sista är bidrag från de kända lutningarna i punkterna.

Särskilt trevlig blir integrationsformeln för fallet med ekvidistant styckvis interpolation när alla h_i-värden är lika, för då tar alla k_i ut varann utom det första och sista värdet. Integralen från x_1 till x_N blir

$$\int_{x_1}^{x_N} f(x)\,dx = \frac{h}{2}\sum_{i=1}^{n}(y_i + y_{i+1}) + \frac{h^2}{12}\sum_{i=1}^{n}(k_i - k_{i+1}) = T(h) - \frac{h^2}{12}(k_N - k_1).$$

som överensstämmer helt med de två första termerna i Euler-Maclaurins formel (25).



5.2.5 Integration av kubiska bézierkurvor

I avsnittet 4.2.5 i bézierkurvekapitlet visades att arean mellan en kvadratisk bézierkurva och linjen $\mathbf{p}_1\,\mathbf{p}_2$ är två tredjedelar av styrpunktstriangelns area. Man skulle vilja se ett lika enkelt samband i det kubiska fallet — alltså mellan arean av styrpunktspolygonen och arean som innesluts av den kubiska bézierkurvan och linjen $\mathbf{p}_1\,\mathbf{p}_2$. Eftersom det leder till hiskligt formelräknande att komma fram till eventuellt samband så avstår vi från det och använder numerisk integration för att bestämma den sökta inneslutna arean.

Bézierkurvan och dess derivata bestäms i parameterform av

$$\begin{aligned}(x(t),\,y(t)) = &\quad (1-t)^3(x_1,y_1) + 3t(1-t)^2(x_b,y_b) + 3t^2(1-t)(x_c,y_c) + t^3(x_2,y_2)\\ (\dot{x}(t),\,\dot{y}(t)) = &\quad -3(1-t)^2(x_1,y_1) + 3(1-4t+3t^2)(x_b,y_b) +\\ &\quad +3(2t-3t^2)(x_c,y_c) + 3t^2(x_2,y_2))\end{aligned}$$

För att göra det hela till en sluten kurva passar vi på att skriva den räta linjen från $\mathbf{p}_2$ till $\mathbf{p}_1$ i parameterform också (jämför formel (13) i avsnitt 4.1):

$$(x_L(u),\,y_L(u)) = (1-u)(x_2,y_2) + u\,(x_1,y_1),\;\; 0 \le u \le 1.$$

Arean inom den slutna kurvan från $\mathbf{p}_1$ i positiv led längs bézierkurvan till $\mathbf{p}_2$ och rätlinjigt tillbaka till $\mathbf{p}_1$ bestäms av formeln

$$\text{Area} \;\; = -\left(\int_0^1 y(t)\dot{x}(t)\,dt + \int_0^1 y_L(u)\dot{x}_L(u)\,du\right)$$

I det första uttrycket ska ett femtegradspolynom integreras och det gör vi av bekvämlighetsskäl med numerisk integration (trapetsregeln med halverat steg utnyttjas).

Det andra uttrycket är integralen av ett förstagradspolynom i u. Det kan lätt integreras exakt (gör det!) och blir $(x_1 - x_2)(y_1 + y_2)/2$, dvs parallelltrapetsarean under den räta linjen.

```
p1=[10 1]; p2=[2 5]; b=[15 6]; c=[-2 14];  pbcp=[p1; b; c; p2];
n=40; I=[];
for nr=1:2                                          % trapetsregeln 2 ggr
  dt=1/n; t=(0:dt:1)';
  F=[(1-t).^3   3*t.*(1-t).^2  3*t.^2.*(1-t) t.^3]; r=F*pbcp;
  Fp=3*[-(1-t).^2  1-4*t+3*t.^2  2*t-3*t.^2  t.^2]; rp=Fp*pbcp;
  x=r(:,1); y=r(:,2);  xp=rp(:,1);
  f=y.*xp;                                          % integrand
  T=dt*(sum(f)-0.5*(f(1)+f(n+1))); I=[I T]; n=2*n;
end
Ibez=I(2)+(I(2)-I(1))/3;                            % simpsonvärde

Ilinje=(p1(1)-p2(1))*(p1(2)+p2(2))/2;
bezarea=-(Ibez+Ilinje)                      % arean inom slutna kurvan

% Polygonarean kan t ex beräknas som två triangelareor:
  polygonarea=0.5*abs(det([b-p1;c-p1]))+0.5*abs(det([c-p1;p2-p1]))
```

Arean av det skuggade området blir 44.55 med fyra korrekta siffror. Arean inom hela styrpunktspolygonen är här 90.50, alltså drygt dubbla värdet.

5.3 Gausskvadratur – väl valda punkter

Gausskvadratur är integrationsformler på formen (23)

$$I = \int_{-1}^{1} f(x)\,dx = G_N = \sum_{i=1}^{N} w_i f(x_i),$$

med kända vikter och av Gauss väl valda x-värden (även kallade *noder*) som inte är ekvidistanta. De är så kallade öppna formler vilket betyder att intervallets ändpunkter inte ingår. Formlerna där N är litet, såsom tvåpunktsformeln G_2, trepunktsformeln G_3, etc upp till G_6, utgör oftast alltför grova approximationer till integralen för att vara praktiskt användbara.

Sjupunktsformeln G_7 har nu kommit att bli den viktigaste tack vare Alexander Kronrod som 1964 utvecklade Gauss-Kronrods algoritm.

Varje gausskvadraturformel har alltså en känd uppsättning vikter och x-värden som lätt kan lagras och användas vid datorberäkningar. De gäller för *standardintervallet* $[-1,\ 1]$ det vill säga integralen $\int_{-1}^{1} f(x)dx$. För härledning av vikter och noder, se Appendix.

Tvåpunktsformeln lyder $G_2 = f(-\frac{1}{\sqrt{3}}) + f(\frac{1}{\sqrt{3}})$ och har alltså vikterna $w_1 = w_2 = 1$ och noderna $\pm\frac{1}{\sqrt{3}}$.

Med endast tre medtagna siffror i w- och x-värdena ser sjupunktsformeln ut så här:

$$\int_{-1}^{1} f(x)dx \;\approx\; G_7 \;=\; 0.418\,f(0) \;+\; 0.382\,(f(-0.406)+f(0.406)) \;+$$
$$+\,0.280\,(f(-0.742)+f(0.742)) \;+\; 0.129\,(f(-0.949)+f(0.949)).$$

I funktionen `g7k15` som beskrivs senare är sjupunktsformelns vikter och x-värden givna med femton siffrors noggrannhet.

Att w_i och x_i är beräknade för standardintervallet $[-1,\ 1]$ innebär ingen inskränkning, eftersom varje integral på intervallet $[a,\ b]$ genom substitutionen $t = \frac{b-a}{2}x + \frac{a+b}{2}, \quad dt = \frac{b-a}{2}dx$ kan överföras till intervallet $[-1,\ 1]$:

$$\int_a^b F(t)dt = \frac{b-a}{2}\int_{-1}^{1} F(\frac{b-a}{2}x + \frac{a+b}{2})dx \approx \frac{b-a}{2}\sum_{i=1}^{N} w_i\,F(\frac{b-a}{2}x_i + \frac{a+b}{2})$$

eller enklare uttryckt:

$$\int_a^b F(t)dt \approx \frac{b-a}{2}\sum_{i=1}^{N} w_i F(t_i) \quad \text{med} \quad t_i = \frac{b-a}{2}x_i + \frac{a+b}{2} \tag{26}$$

w_i och x_i är N-punktsformelns parametrar för intervallet $[-1,\ 1]$.

Integration med Gauss N-punktsformel ger exakt resultat för polynom av grad $2N-1$. För sjupunktsformeln gäller alltså att felet blir noll om vi integrerar polynom av trettonde graden eller lägre. Vi vill kunna skatta felet då vi behandlar en allmännare funktion $f(x)$. Någon användbar teoretisk felformel finns inte; vi behöver alltså en experimentell felskattningsregel. Kronrods forskning på detta område har lett fram till följande metod.

5.4 Gauss-Kronrods metod – 15 smart valda punkter

Vid trapetsregeln används steghalvering på ett effektivt sätt för resultatförbättring och experimentell felskattning. Gausskvadraturformlerna har inte samma trevliga egenskap att behålla gamla x-värden när antalet intervall fördubblas. Då N ändras förändras alla x_i och inga tidigare beräknade funktionsvärden $f(x_i)$ är till någon glädje.

Den ryske matematikern Kronrod lyckades år 1965 härleda en effektiv algoritm som bygger på gausskvadratur men också liknar steghalveringsalgoritmen vid trapetsregeln. Kronrods idé var att starta med Gauss sjupunktsformel G_7 och finna en ny formel som använder dessa punkter samt

lämpligt valda mellanliggande punkter, framräknade så att integrationsresultatet förbättras och en god felskattning erhålls. Med åtta nya punkter ξ_i konstruerade han en femtonpunktsformel K_{15} som ger exakt resultat för polynom upp till och med tjugoandra graden.

Algoritm: Beräkna först $f_i = f(x_i)$ för sjupunktsformelns kända x_i-värden (vi betraktar alltså integration över standardintervallet $[-1,\ 1]$). Räkna därefter ut värdena G_7 och K_{15} enligt:

$$G_7 = \sum_{i=1}^{7} w_i f_i, \quad K_{15} = \sum_{i=1}^{7} u_i f_i + \sum_{i=1}^{8} v_i f(\xi_i).$$

Här är w_i sjupunktsformelns vikter, medan den högra formelns vikter u_i och v_i samt ξ_i-värdena har angetts av Kronrod. I funktionen `g7k15` betecknas gausspunkterna `xg` och kronrodpunkterna `xk`.

För integraler över intervallet $[a,\ b]$, alltså $\int_a^b F(t)dt$, tillämpas algoritmen tillsammans med variabelsubstitutionen enligt formel (26) och vi får följande Gauss-Kronrod-algoritm:

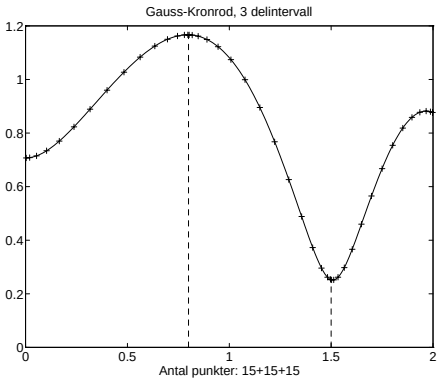
$$\begin{gathered} t_i = \tfrac{b-a}{2}x_i + \tfrac{a+b}{2},\ \ F_i = F(t_i),\ i = 1, 2, \ldots, 7, \quad G_7 = \tfrac{b-a}{2}\sum_{i=1}^{7} w_i F_i \\ \tau_i = \tfrac{b-a}{2}\xi_i + \tfrac{a+b}{2},\ i = 1, 2, \ldots, 8, \quad K_{15} = \tfrac{b-a}{2}\left(\sum_{i=1}^{7} u_i F_i + \sum_{i=1}^{8} v_i F(\tau_i)\right) \end{gathered} \tag{27}$$

Värdet K_{15} accepteras som integralvärde, medan G_7-värdet enbart används för bedömning av noggrannheten i resultatet.

En experimentell felskattning som för de flesta integrander är mycket tillförlitlig ges av $d = |K_{15} - G_7|$, $\quad \text{fel} < 10d\sqrt{d/|K_{15}|}$.

Hela algoritmen beskrivs i MATLAB i funktionsfilen **g7k15.m** som finns i kurskatalogen och kan kopieras därifrån.

```
function [Igk,absfel]=g7k15(F,a,b);
%Gauss-Kronrods 15-punktsmetod, integrand F(x), intervall [a,b]
 xg=[0.405845151377397 0.741531185599394 0.949107912342758];
 xg=[-fliplr(xg)  0  xg];                                  % gausspunkter
 w=[0.381830050505119 0.279705391489277 0.129484966168870];
 w=[fliplr(w)  0.417959183673469  w];                 % gaussvikter
%Kronrodpunkter och nya vikter
 xk=[0.207784955007898 0.586087235467691 0.864864423359769 0.991455371120813];
 xk=[-fliplr(xk)  xk];
 u=[0.190350578064785 0.140653259715525 0.063092092629979];
 u=[fliplr(u)  0.209482141084728  u];
 v=[0.204432940075298 0.169004726639267 0.104790010322250 0.022935322010529];
 v=[fliplr(v)  v];
```




```
%Variabelsubstitution till standardintervallet [-1,1]:
 hh=(b-a)/2;  abm=(a+b)/2;
 tg=hh*xg+abm;  fg=feval(F,tg);
 tk=hh*xk+abm;  fk=feval(F,tk);
 G7=hh*sum(w.*fg);  K15=hh*(sum(u.*fg)+sum(v.*fk));
 d=abs(K15-G7);  fel=10*d*sqrt(d/abs(K15));
 Igk=K15;                                          % integralapproximation
 absfel=max([eps*abs(K15) fel]);                   % felskattning
```

För integraler där integranden har kraftiga variationer i integrationsintervallet är inte femton punkter – hur väl valda de än är – tillräckligt för att få hyfsad approximation till integralvärdet. Då får man dela in intervallet i flera delintervall och tillämpa Gauss-Kronrods formel (27) på varje del.

En god idé är att rita upp integrandkurvan och se hur den beter sig i integrationsintervallet. Lägg delintervallens gränser där krökningen är stor.

Exempel: Skriv ett program som med Gauss-Kronrods metod beräknar integralen $I = \int_0^2 \sqrt{0.5 + 2e^{-x} \sin 2x^2}\, dx$ med minst sju korrekta decimaler. Uppritning av integranden visar att kurvans krökning är störst vid $x \approx 0.8$ och vid $x \approx 1.5$. Vi prövar därför med en indelning i tre intervall, från 0 till 0.8, från 0.8 till 1.5 och slutligen från 1.5 till 2.

```
[I1,fel1]=g7k15(@fq,0, a1); fel1
[I2,fel2]=g7k15(@fq,a1,a2); fel2
[I3,fel3]=g7k15(@fq,a2, 2); fel3
Ig7k15=I1+I2+I3,
absfel=fel1+fel2+fel3
```

Integralvärdet blir 1.6420705 med ett fel skattat till högst en enhet i sjunde decimalen.

5.5 Integraler som fordrar förbehandling

Integraler över ett oändligt integrationsintervall och integraler med singulariteter i integranden kräver förbehandling innan en numerisk metod kan tillämpas.

5.5.1 Knepig integral

Bestäm med tre korrekta decimaler värdet av följande integral

$$I = \int_0^\infty \frac{dx}{\sqrt{x}\,(x^2 + \sqrt{1+x^3})}$$

som har båda egenskaperna ovan. Genom en lämplig substitution, $t = \sqrt{x}$, blir man av med singulariteten vid $x = 0$.

Numerisk integration kan bara göras på ett ändligt intervall. Vi bör därför dela intervallet i två delar, $0 \le t \le T$ och $T \le t \le \infty$, och välja T-värdet så stort att den sista delintegralen (svansen) blir försumbar, i detta fall av storleksordningen 10^{-4}.

Den första delintegralen beräknas med lämplig numerisk metod med en tolerans som garanterar att resultatet totalt sett får begärd noggrannhet.

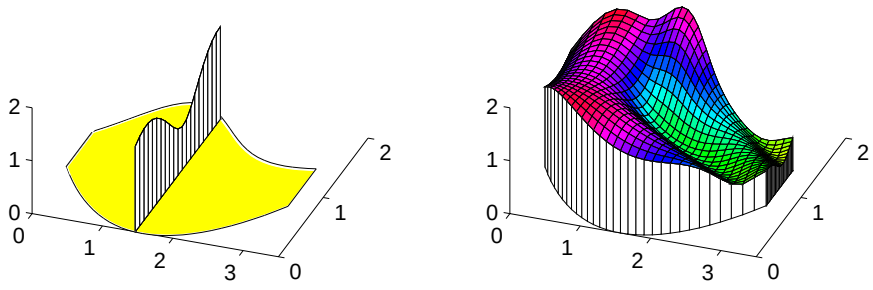
Lösning: Substituera $t = \sqrt{x}$, dvs $t^2 = x$, $2t\,dt = dx$ så blir integraluttrycket $I = \int_0^\infty 2/(t^4 + \sqrt{1+t^6})\,dt$. Svansintegranden kan överskattas med uttrycket $2/(t^4 + t^3) < 2/t^4$. Integrering från T till ∞ ger att svansen är mindre än $\frac{2}{3}T^{-3}$. Med $T = 15$ blir värdet 0.0002 vilket gör svansen försumbar.

Nu återstår beräkningen av integralen mellan 0 och 15. Med MATLABs `quad` erhålls värdet med sex korrekta decimaler (standardtolerans).

```
I1=quad(inline('2./(t.^4+sqrt(1+t.^6))'),0,15)
```

Integralvärdet blir 2.0507 med ett totalfel mindre än 0.0002.

Lös övningarna **Ex 6.4, 6.9, 6.10** *i exempelsamlingen.*



5.6 Dubbelintegraler

Låt oss betrakta ett område D i xy-planet begränsat av de räta linjerna $x = a$ och $x = b$ och av undre och övre gränskurvorna $y = c(x)$ och $y = d(x)$. En dubbelintegral över området kan skrivas

$$I = \int\int_D f(x,y)\,dx\,dy = \int_a^b \int_{c(x)}^{d(x)} [f(x,y)\,dy]\;dx = \int_a^b g(x)\,dx$$

där $f(x,y)$ är en given integrandfunktion och $g(x) = \int_{c(x)}^{d(x)} f(x,y)\,dy$.

I den numeriska algoritmen för dubbelintegraler spjälkar man upp problemet i två endimensionella integrationsproblem. Börja med att dela in x-intervallet i n_x delintervall med steget $\delta x = (b-a)/n_x$. För vart och ett av de n_x+1 stycken x-värdena $x_i = a + (i-1)\,\delta x$ beräknas integralen $g(x_i)$. Det illustreras av den vänstra figuren där integralvärdet $g(x_i)$ utgör arean av den randade väggen med den y-beroende höjden $f(x_i, y)$.

Den numeriska integrationen brukar man oftast utföra med trapetsregeln (24), men `quad` eller Gauss-Kronrods metod (27) kan vara goda alternativ. Om integrandfunktionen $f(x,y)$ har byggts upp av styckvis interpolation med fusksplines eller splines är det lämpligt att i stället för trapetsregeln utnyttja den förbättrade integrationsformeln som presenterades i avsnitt 5.2.4. När alla n_x+1 värdena g_i är beräknade, återstår det att beräkna det slutliga integralvärdet $I = \int_a^b g(x)\,dx$. Här ligger trapetsregeln närmast till hands med den enkla formeln (24): $I \approx \delta x\,(\sum_{i=1}^{n_x+1} g_i - (g_1 + g_{n_x+1})/2)$.

Noggrannheten i resultatet beror liksom i det endimensionella fallet på hur fin indelning som valts. En praktisk tillförlitlighetskontroll av antalet korrekta siffror kan vi åstadkomma genom att räkna om med halvering av steget, vilket i dubbelintegralfallet bör ske både i x- och y-led.

Exempel: Beräkna värdet av integralen

$$I = \int\int_D e^{-(x/\pi)^2}\,\sqrt{3+\sin 3xy+\cos 3x}\,dx\,dy$$

över området D (se figuren) begränsat av $0 \le x \le \pi$ och av kurvorna

$$c(x) = x^2/\pi - x + \pi/4, \quad d(x) = x/\pi + e^{-x/\pi} + e^{-(x-1)^2}.$$

Integranden $f(x,y)$ skrivs som en MATLAB-funktion

```
function f=fdub(x,y)
   f=sqrt(3+sin(3*x*y)+cos(3*y)).*exp(-(x/pi).^2);
```

Vi använder trapetsregeln (24) för alla integralberäkningar. I första exekveringen utnyttjar vi 100 delintervall i x-led och 50 i y-riktningen. Vid varje x_i bestäms steget i y-led av $\delta y = (d(x_i) - c(x_i))/n_y$.

```
  a=0; b=pi;
  nx=100; dx=(b-a)/nx; x=(a:dx:b)';
  c=x.^2/pi-x+pi/4;
  d=x/pi+exp(-x/pi)+exp(-(x-1).^2);
  ny=50;
  Y=[]; F=[]; G=[];
  for i=1:nx+1
    dy=(d(i)-c(i))/ny;  y=c(i):dy:d(i);
    f=fdub(x(i),y);
    g=dy*(sum(f)-0.5*(f(1)+f(ny+1)));        % trapetsregeln vid fixt x
    Y=[Y; y];  F=[F; f];  G=[G; g];
  end
  I=dx*(sum(G)-0.5*(G(1)+G(nx+1)))           % trapetsregeln på G-vektorn

% 3D-ritning (gör help meshz och help surf)
  X=x*ones(1,ny+1);
  meshz(X,Y,F), hold on, surf(X,Y,F)
  axis([0 3.5 0 2 0 2]), view(20,50)
```

Integralvärdet som också är värdet av volymen mellan xy-planet och rymdytan blir $I = 5.7642$ med osäkerhet på en enhet i sista siffran. Detta konstateras eftersom $n_x = 100$, $n_y = 50$ ger 5.7638 och $n_x = 200$, $n_y = 100$ ger värdet 5.7641, och ännu finare indelning med $n_x = 400$, $n_y = 200$ ger 5.7642.

I MATLAB finns en funktion `dblquad` som klarar dubbelintegraler på ett rektangulärt område. Så om vi istället hade de konstanta funktionerna $c(x) = 0$ och $d(x) = 2$ i intervallet $0 \le x \le \pi$ skulle MATLAB-funktionen kunna anropas med:

```
Irekt=dblquad(@fdub,0,pi,0,2)
```

5.7 APPENDIX, Parametrarna i gausskvadratur

Parametrarna är bestämda så att formeln ger exakt resultat för polynom av så hög grad som möjligt. N-punktsformeln G_N innehåller $2N$ parametrar och för att kunna beräkna dem behövs $2N$ samband. Dessa erhålls genom önskemålet att formeln ska stämma för $f(x) =$ konstant, $f(x) =$ förstagradspolynom, $f(x) =$ andragradspolynom, upp till gradtal $2N-1$.

För att ekvationerna ska bli så enkla som möjligt väljer man polynomen $1,\ x,\ x^2, \ldots, x^{2N-1}$, dvs $x^p,\ p = 0, 1, 2, \ldots, 2N–1$. Integration över intervallet $[-1,\ 1]$ ger värdet noll för udda p och $2/(p+1)$ för jämna p.

$$w_1 x_1^p + w_2 x_2^p + \ldots + w_N x_N^p = \begin{cases} 2/(p+1) & p = 0, 2, \ldots, 2N-2 \\ 0 & p = 1, 3, \ldots, 2N-1. \end{cases}$$

Första ekvationen (då $p=0$) blir alltså $w_1 + w_2 + \ldots + w_N = 2$.
Av symmetriskäl (ur nollsambanden ovan) kommer *x-värdena att ligga symmetriskt kring* $x = 0$, dvs vi får $x_i = -x_{N+1-i}$, och tillhörande vikter blir lika, alltså $w_i = w_{N+1-i}$.
Om antalet punkter är udda ligger mittersta punkten alltid i $x = 0$.

Övning: Visa att sambanden för $p = 1, 3, \ldots, 2N-1$ är satisfierade.

Antalet okända parametrar har nu reducerats från $2N$ till N, och genom de återstående ekvationerna kan dessa parametrar bestämmas. För $N \geq 3$ blir det ett ickelinjärt system som löses med till exempel Newtons metod.

Tvåpunktsformeln kan nu skrivas $G_2 = w_1 \left(f(x_1) + f(-x_1)\right)$ och man erhåller direkt $w_1 = 1$ och $x_1 = \pm 1/\sqrt{3}$, alltså $G_2 = f(-1/\sqrt{3}) + f(1/\sqrt{3})$. Formeln är exakt för alla tredjegradspolynom.

Beräkning av sjupunktsformelns vikter och x-värden är en lämplig uppgift på ickelinjära ekvationssystem, Ex3.16 i exempelsamlingen.

Anmärkning 1: Gausskvadratur har interpolerande egenskaper fast det inte syns explicit i formlerna. Det innebär att om vi till sjupunktsformelns x_i-värden bildar $f(x_i)$, interpolerar med ett sjättegradspolynom genom punkterna och integrerar detta, så får vi identiskt samma resultat som om vi tillämpar sjupunktsformeln! Runges fenomen uppstår inte eftersom gausspunkterna ligger tätare utåt kanterna.

Anmärkning 2: Det gäller att x-värdena till gausskvadraturformeln G_N är nollställen till *legendrepolynomet* $P_N(x)$ av gradtal N (se tabell i Betahandboken), till exempel gäller: $P_7(x) = (429x^7 - 693x^5 + 315x^3 - 35x)/16$.
I MATLAB kan vi erhålla alla noderna till G_7 med satsen

```
xg=sort(roots([429 0 -693 0 315 0 -35 0]))
```

6 EKVATIONER OCH EKVATIONSSYSTEM

6.1 Allmänt

Ekvationslösning innebär att vi söker en eller flera rötter till ekvationen $f(x) = 0$. När ekvationen skrivs på standardformen ovan kan problemet också formuleras som "sök nollställena till funktionen $f(x)$". Om funktionsuttrycket inte är alltför komplicerat kan funktionskurvan ritas upp, och ur figuren kan vi avläsa ungefärlig position för nollställena.

Risken är förstås att det finns nollställen utanför det valda ritningsintervallet. Om det gäller att hitta samtliga rötter till en given ekvation måste man göra en grovanalys av $f(x)$, så att man lyckas fånga in alla intervall där funktionen byter tecken. I de praktiska tillämpningarnas ekvationslösningsproblem finns ofta någon känd begränsning för den obekanta roten och då har man ett naturligt intervall där roten kan sökas.

Att funktionen byter tecken vid nollställespassagen gäller inte vid en dubbelrot. Om man vet i förväg att sökningen gäller en dubbelrot är det bättre att lösa ekvationen $f'(x) = 0$.

Metoderna i avsnitten 6.3 – 6.5 gäller enkelrötter. Newton-Raphsons metod kan dock tillämpas vid dubbelrot men med långsam konvergens, se avsnitt 6.4.2, där också ett uppsnabbningsknep visas.

6.2 Polynomekvationer

Ekvationen $P(x) = 0$ där $P(x)$ är ett polynom med givna koefficienter löser man enklast med MATLAB-kommandot `roots(c)`, där `c` är en vektor innehållande polynomets koefficienter från högsta grad ned till konstanttermen. Alla rötter skrivs ut, både reella och komplexa!

Exempel: Bestäm nollställena till legendrepolynomet av sjunde graden (se t ex Betahandboken): $P_7(x) = (-35x + 315x^3 - 693x^5 + 429x^7)/16$.

```
c=[429 0 -693 0 315 0 -35 0]; xg=roots(c)
```

Dessa nollställen utgör de sju x-värdena (noderna) i gausskvadraturformeln G_7 (se kap 5.4, `g7k15`-funktionens femtonsiffriga `xg`-värden).

6.3 Intervallhalveringsmetoden

Intervallhalveringsmetoden är en enkel metod för ekvationslösning, då man vet att roten är instängd i ett intervall $a \leq x \leq b$. Låt oss anta att $f(a) < 0$ och $f(b) > 0$. Beräkna intervallets mittpunkt $m = (a + b)/2$ och räkna ut

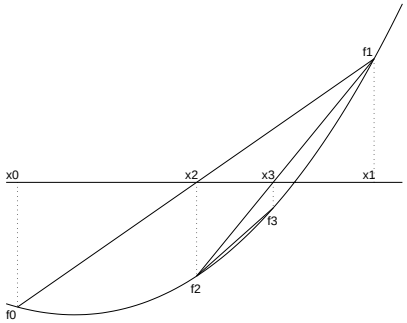
funktionsvärdet $f(m)$. Om funktionsvärdet är negativt förkastas hela intervallet mellan a och m, och a flyttas till mittpunktspositionen. Om $f(m) > 0$ förkastas istället intervallet till höger om mittpunkten, och det blir b som får glida åt vänster och anta m-värdet.

Detta upprepas, och i varje iterationssteg reduceras intervallets längd med en faktor två. Efter n iterationer har den sökta roten stängts in i ett intervall som är 2^{-n} av ursprungsintervallet $b - a$.

Intervallhalveringsmetoden är en långsamt konvergerande metod, som bara bryr sig om tecknet hos det i varje steg beräknade funktionsvärdet. Betydligt effektivare och mer användbara ekvationslösningsmetoder är Newton-Raphsons metod och sekantmetoden.

6.4 Sekantmetoden

I sekantmetoden behöver man två startgissningar x_0 och x_1 till roten. Beräkna funktionsvärdena $f(x_0)$ och $f(x_1)$, dra den räta linjen, sekanten, mellan de två punkterna på kurvan, undersök var linjen skär x-axeln och låt detta x-värde vara nästa approximation till roten. Upprepa förfarandet tills önskad noggrannhet uppnås.



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

Algoritmen för sekantmetoden kan skrivas:

- Utgå från två startgissningar x_0 och x_1; beräkna $f_0 = f(x_0)$
- (*) Beräkna $f_1 = f(x_1)$
- Bilda $\delta x = -f_1 \cdot (x_1 - x_0)/(f_1 - f_0)$
- Uppdatera: $x_0 = x_1, \quad f_0 = f_1, \quad x_1 = x_1 + \delta x$
- Upprepa från (*) ända tills korrektionstermen δx blivit så liten till sitt belopp att den gott och väl hamnar inom den begärda felgränsen.

Fördelen med sekantmetoden framför Newton-Raphsons metod är att funktionen inte behöver deriveras (den behöver inte ens vara deriverbar). Nackdelen är att två startvärden krävs och att konvergensen är något långsammare än för Newton-Raphsons metod, vilket innebär att det krävs fler iterationer för att nå begärd noggrannhet.

6.5 Newton-Raphsons metod

Newton-Raphsons metod $x_{n+1} = x_n - f(x_n)/f'(x_n)$ är en mycket effektiv ekvationslösningsmetod som dock kräver att $f'(x)$ har ett analytiskt uttryck. Geometriskt sett är sekantmetoden och Newton-Raphsons metod mycket lika. Ersätt sekanten med tangenten till kurvan i punkten $(x_0,\ f(x_0)$ (låt alltså sekantmetodens båda startvärden glida ihop till en punkt). Undersök var tangenten skär x-axeln och låt detta vara nästa approximation till roten. Upprepa tills korrektionstermen $\delta x = -f(x)/f'(x)$ är så liten till beloppet att den hamnar inom begärd felgräns.

Gott råd: En given ekvation kan i sin ursprungsform vara onödigt komplicerad. Skriv om den på något smart sätt innan du tillämpar Newton-Raphsons metod. Genom förlängning kan obehagliga nämnare undvikas, genom kvadrering som i exemplet nedan undviks svårderiverade rotuttryck.

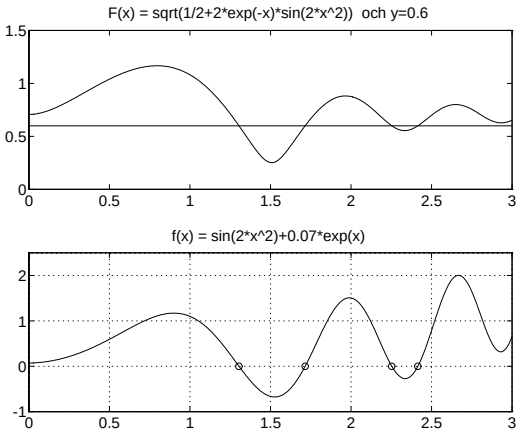
6.5.1 Exempel: Skärning mellan funktionskurva och rät linje

Bestäm alla skärningspunkter mellan kurvan $F(x) = \sqrt{0.5 + 2e^{-x}\sin 2x^2}$ och linjen $y = 0.6$ i intervallet $0 \le x \le 3$. I övre figuren visas kurvan och linjen, och det är lätt att se att det finns fyra skärningspunkter i intervallet. Ekvationen som ska lösas blir

$$\sqrt{0.5 + 2e^{-x}\sin 2x^2} = 0.6$$

men den kan få en mycket enklare form. Börja med att kvadrera båda sidorna: $0.5 + 2e^{-x}\sin 2x^2 = 0.36$ och förenkla ytterligare till $\sin 2x^2 + 0.07e^x = 0$.

Nu är ekvationen på formen $f(x) = 0$ med den enkelt deriverbara funktionen $f(x) = \sin 2x^2 + 0.07e^x$ med derivatan $f'(x) = 4x\cos 2x^2 + 0.07e^x$.



Ur funktionskurvan för $f(x)$ kan vi avläsa att nollställena ligger ungefär vid $x = 1.3$, $x = 1.7$, $x = 2.2$ och $x = 2.4$.
(Har funktionen överhuvudtaget några flera positiva nollställen? Analysera!)

Avbrottskriteriet i programkoden är valt så att rötterna bestäms med tretton siffrors noggrannhet.

```
% Newton-Raphsons metod, fyra rötter
  format long, xstart=[1.3  1.7  2.2  2.4];
  for nr=1:4
    x=xstart(nr);  dx=x;  iter=0;
    while abs(dx/x)>1e-14 & iter<10
      f=sin(2*x^2)+0.07*exp(x);
      fp=4*x*cos(2*x^2)+0.07*exp(x);
      dx=-f/fp;  disp([x dx]),  x=x+dx; iter=iter+1;
    end
  end
```

Resultaten visar att fyra iterationer är tillräckligt för att få den önskade noggrannheten med så goda startvärden som vi hade här. Studera den högra kolumnen nedan som innehåller de successiva korrektionerna. De avtar mycket snabbt!

```
Rot 1:          x                   dx
          1.30000000000000    0.00431508522162
          1.30431508522162    0.00000578996265
          1.30432087518427    0.00000000001197
          1.30432087519624    0.00000000000000

Rot 2:          x                   dx
          1.70000000000000    0.01562099486746
          1.71562099486746   -0.00045244935124
          1.71516854551623   -0.00000034028265
          1.71516820523358   -0.00000000000019
          1.71516820523338    0.00000000000000
```

Tredje och fjärde roten blir (avrundade till sju korrekta siffror) 2.253219 respektive 2.415182.

Linjär och kvadratisk konvergens

En iterativ ekvationslösningsmetod sägs ha *linjär konvergens* mot roten α om $|x_{n+1} - \alpha| \approx C\,|x_n - \alpha|$ där C är en positiv konstant mindre än ett.
Kvadratisk konvergens mot en rot α innebär att $|x_{n+1} - \alpha| \approx K\,|x_n - \alpha|^2$ där K är en positiv konstant.

Newton-Raphsons metod har kvadratisk konvergens. Kan man utläsa det ur korrektionstermernas avtagande i exemplet?

6.5.2 Härledning av Newton-Raphsons kvadratiska konvergens

Låt roten finnas vid $x = \alpha$, då gäller: $f(\alpha) = 0$. Taylorutveckla kring x_n:

$$f(\alpha) = f(x_n) + (\alpha - x_n)f'(x_n) + \frac{(\alpha - x_n)^2}{2}f''(x_n) + \frac{(\alpha - x_n)^3}{3!}f'''(x_n) + \dots$$

Dividera med $f'(x_n)$ (med förutsättning att $f'(x_n) \neq 0$) och utnyttja att vänsterledet har värdet noll:

$$0 = \frac{f(x_n)}{f'(x_n)} + \alpha - x_n + (\alpha - x_n)^2 \frac{f''(x_n)}{2f'(x_n)} + \dots$$

Möblera om lite:

$$x_n - \frac{f(x_n)}{f'(x_n)} - \alpha = \frac{f''(x_n)}{2f'(x_n)}(\alpha - x_n)^2 + \dots \tag{28}$$

Hittills har vi inte utnyttjat att det hela gäller Newton-Raphsons metod. Nu är det dags, de två första termerna ovan är just x_{n+1} i Newton-Raphson vilket ger

$$x_{n+1} - \alpha = \frac{f''(x_n)}{2f'(x_n)}(\alpha - x_n)^2 + \dots \tag{29}$$

$$|x_{n+1} - \alpha| \approx \left|\frac{f''(x_n)}{2f'(x_n)}\right| |x_n - \alpha|^2 \approx K|x_n - \alpha|^2.$$

Detta bevisar att Newton-Raphsons metod för enkelrötter har kvadratisk konvergens, och att för konvergenskonstanten K gäller

$$K = \lim_{n\to\infty} \left|\frac{f''(x_n)}{2f'(x_n)}\right| = \left|\frac{f''(\alpha)}{2f'(\alpha)}\right|.$$

Vad händer om α råkar vara **dubbelrot**?
Förutom $f(\alpha) = 0$ gäller nu också $f'(\alpha) = 0$. Taylorutveckla derivatafunktionen:

$$f'(\alpha) = f'(x_n) + (\alpha - x_n)f''(x_n) + \dots \quad \Longrightarrow 0 = f'(x_n) + (\alpha - x_n)f''(x_n) + \dots$$

Uttrycket $f''(x_n)/f'(x_n)$ approximeras alltså av $-1/(\alpha - x_n)$. Insättning i (29) ger

$$x_{n+1} - \alpha = \frac{-1}{2(\alpha - x_n)}(\alpha - x_n)^2 + \dots, \quad \text{dvs } |x_{n+1} - \alpha| \approx \frac{|x_n - \alpha|}{2}.$$

Slutsatsen blir att vid dubbelrot erhålls linjär konvergens med konvergensfaktorn 1/2; ett nytt iterationssteg i Newton-Raphsons metod ger bara en ungefärlig halvering av felet.

Det finns ett enkelt sätt att modifiera Newton-Raphsons metod så att den ger kvadratisk konvergens för multipelrötter: Om roten har multipliciteten p läggs faktorn p framför korrektionstermen. För dubbelrotsberäkning lyder formeln alltså $x_{n+1} = x_n - 2f(x_n)/f'(x_n)$.

Övning: Visa med taylorutveckling att formeln ovan ger kvadratisk konvergens vid dubbelrötter.

6.6 Fixpunktsiteration

För en ekvation som är skriven på formen $x = G(x)$ kan det vara naturligt att pröva den enkla iterationsformeln $x_{n+1} = G(x_n)$ som kallas fixpunktsiteration.

Exempel: Lös ekvationen $x = 0.5 + \sin(x/2)$ med fixpunktsiterationsformeln $x_{n+1} = 0.5 + \sin(x_n/2)$ med startvärde $x_0 = 1$.

```
x=1; korr=1;
while abs(korr)>5e-4
  xold=x; x=0.5+sin(x/2), korr=x-xold;
end
```

Följande värden skrivs ut: 0.9794, 0.9704, 0.9664, 0.9646, 0.9638, 0.9635.
Det blir alltså konvergens, och roten är $\alpha = 0.9635 \pm 0.0005$.

Men metoden konvergerar inte alltid! Det finns ett konvergenskrav som lyder

$$|G'(x)| < 1$$

i en omgivning av roten, och om detta är uppfyllt har fixpunktsmetoden linjär konvergens med faktorn $C = |G'(\alpha)|$.

Fixpunktsiteration har teoretisk betydelse men används sällan för praktisk ekvationslösning, eftersom den dels kräver lämplig omskrivning för att överhuvudtaget konvergera, dels konvergerar långsamt jämfört med sekantmetoden och Newton-Raphsons metod.

Exempelsamlingens **Ex 2.1–2.20** *rekommenderas!*

6.7 Iterativ lösning av glesa linjära ekvationssystem

Ett linjärt ekvationssystem $\mathbf{Ax} = \mathbf{b}$ kan lösas med iterativ metod om systemet har en gles och diagonaltung systemmatris. Då skriver man om ekvationssystemet så att enbart diagonaltermerna finns på vänstra sidan och flyttar alltså alla övriga termer till höger sida.

Dividera varje rad med diagonalelementet så att endast x_i finns kvar på den i-te radens vänstra sida.

Systemet har nu formen $\mathbf{x} = \mathbf{Mx} + \mathbf{c}$, och fixpunktsiteration kan tillämpas. Man börjar med att sätta in startvärdena $x_1 = x_2 = ... = x_n = 0$ på höger sida och erhåller en ny vektor i vänsterledet; sätter in den i högerledet och räknar fram en förbättrad vektor enligt $\mathbf{x}^{(k+1)} = \mathbf{Mx}^{(k)} + \mathbf{c}$.

Metoden heter Jacobis metod. Konvergens erhålls om $\|\mathbf{M}\| < 1$. Detta krav är uppfyllt om den ursprungliga matrisen $\mathbf{A}$ är diagonaltung.

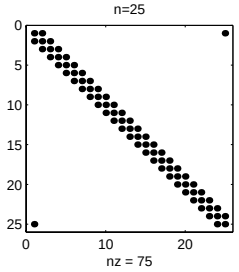
Jacobis metod har teoretiskt intresse men för praktiskt bruk överträffas den av *Gauss-Seidels metod* som är en förbättrad variant av Jacobis metod. I stället för att utföra matris-vektormultiplikationen i högerledet (som skulle kunna åstadkommas av en enda MATLAB-sats) gör man en for-slinga som löper igenom raderna med successiv uppdatering av vektorkomponenterna x_i inom varje iterationssteg.

Det gäller också att utnyttja matrisens glesa egenskap; man bör undvika att lagra hela matrisen och man bör avstå från onödiga multiplikationer med nollelement.

Vi ska demonstrera Gauss-Seidels metod i ett exempel, där vi för jämförelsens skull också gör tidsstudier för gausseliminationsslösning med helfylld matris samt ett litet smakprov på lösning med MATLABs lagrings- och beräkningseffektiva `sparse`-kommandon för glesa matriser. (Gör `help speye` och `help spy`.)

Exempel: Låt $\mathbf{A}$ vara matrisen som uppstår vid ekvidistanta periodiska splines:

$$\begin{pmatrix} 4 & 1 & 0 & 0 & \cdots & 1 \\ 1 & 4 & 1 & 0 & \cdots & 0 \\ 0 & 1 & 4 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & 4 & 1 \\ 1 & 0 & \cdots & 0 & 1 & 4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_{n-1} \\ b_n \end{pmatrix}$$



Strukturen till höger (fallet $n=25$) erhålls med `spy(Asp)` när matrisen lagrats som sparse-matris, se koden.

Ett högerled behövs, låt oss ta $b_i = e^{\cos(2\pi i/n)}$, $i = 1, 2, ..., n$.

I Gauss-Seidels metod itererar vi tills max-avvikelsen mellan två på varandra följande lösningsvektorer är mindre än en enhet i femte decimalen, dvs tills $\|\mathbf{x} - \mathbf{x}_{old}\|_\infty \leq 10^{-5}$. Tidsåtgången noteras för de tre metoderna vid 1000 och 2000 obekanta.

```
  n=1000; b=exp(cos(2*pi*(1:n)'/n));
% Gauss-Seidels metod
  tic,  x=zeros(n,1);  dxnorm=1;
  while dxnorm>1e-5
    xold=x;
    x(1)=(b(1)-x(2)-x(n))/4;
    for i=2:n-1
      x(i)=(b(i)-x(i-1)-x(i+1))/4;     % successiv uppdatering av x(i)
    end
    x(n)=(b(n)-x(n-1)-x(1))/4;
    dxnorm=norm(x-xold,inf);
  end
  toc

% Lösning med helfylld matris, n*n element lagras
  tic, A=4*eye(n);
  for i=1:n-1, A(i,i+1)=1; A(i+1,i)=1; end
  A(1,n)=1; A(n,1)=1;
  xf=A\b;                             % gausseliminationslösning
  toc

% Lösning med sparsematris, endast nollskilda element (3*n) lagras
  tic, Asp=4*speye(n);
  for i=1:n-1, Asp(i,i+1)=1; Asp(i+1,i)=1; end
  Asp(1,n)=1; Asp(n,1)=1;
  xsp=Asp\b;                          % effektiv svartlådelösning
  toc
  spy(Asp)
```

Vid exekvering med 1000 obekanta tar metoderna i tur och ordning 0.3 sekunder, 3 s, 0.1 s. Motsvarande för 2000 obekanta är 0.6, 25, 0.5 sekunder. Dra lämplig slutsats av detta!

Glesa system uppkommer vid behandling av randvärdesproblem för differentialekvationer. Mer om sparsematriser kommer därför i avsnitt 8.7.

6.8 Lösning av ickelinjära ekvationssystem

Vi börjar med ett litet exempel på ett ickelinjärt ekvationssystem med två ekvationer och två obekanta:

$$\begin{aligned} 2x_1 + x_2 &= 2 + x_1 x_2/2 \\ x_1 + 2x_2 &= 1.5 + \cos x_2/2 \end{aligned}$$

Man vet att $x_1 \approx 1$, $x_2 \approx 0.5$. Bestäm x_1 och x_2 med fyra decimaler. Vi ska visa tre metoder för att lösa problemet — två bygger på fixpunktsiteration och den tredje på Newton-Raphsons metod. Den första metoden är Gauss-Seidels metod för ickelinjära system. Då skriver vi om systemet så här:

$$\begin{aligned} x_1 &= (2 + x_1 x_2/2 - x_2)/2 \\ x_2 &= (1.5 + \cos x_2/2 - x_1)/2 \end{aligned}$$

Stoppa in startvärdena för x_1 och x_2 i första högerledet, beräkna nytt x_1, uppdatera x_2 genom insättning i andra högerledet. Upprepa beräkningarna av x_1 och x_2 tills önskad noggrannhet uppnåtts — eller avbryt om ingen konvergens erhålls. I detta fall konvergerar metoden och efter sju iterationer har vektorkomponenterna fyra decimalers noggrannhet: $x_1 = 0.8431$ och $x_2 = 0.5426$.

Metoden som ju är en form av fixpunktsiteration i flera variabler rekommenderas inte för praktiskt bruk av samma skäl som tidigare gavs om fixpunktsiteration i envariabelfallet. Konvergenskravet är nu att normen för jacobianmatrisen för högerledet måste vara mindre än ett.

Vi ska pröva en annan metod som ligger nära till hands när ekvationssystemet är givet på formen ”linjära termer till vänster och resten till höger”. Vårt ekvationssystem kan skrivas på formen $\mathbf{Ax} = \mathbf{b}(\mathbf{x})$:

$$\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 2 + x_1 x_2/2 \\ 1.5 + \cos x_2/2 \end{pmatrix}$$

Algoritm: Sätt in startvärdena x_1 och x_2 i högerledsvektorn $\mathbf{b}$, beräkna förbättrad vektor genom att lösa det linjära systemet $\mathbf{Ax} = \mathbf{b}$ sätt därefter in x_1 och x_2 i högerledet, lös det linjära systemet igen, osv.

Metoden som kallas *picarditeration* är ett specialfall av fixpunktsiteration med hårda konvergenskrav, $\|\mathbf{A}^{-1}\mathbf{J}\| < 1$ där $\mathbf{J}$ är jacobianmatrisen för $\mathbf{b}(\mathbf{x})$. I vårt lilla exempel får vi dock konvergens och det visar sig gå åt sex iterationer för att få lösningen med begärd noggrannhet.

Metoderna ovan har en mycket begränsad användning på grund av de svåruppnåeliga konvergenskraven. Den tredje metoden som vi ska tillämpa på vårt exempel är Newtons metod. Det är den säkraste och bästa metoden för ickelinjära ekvationssystem.

6.9 Newtons metod för ickelinjära ekvationssystem

Ett ickelinjärt ekvationssystem som ska lösas med Newtons metod måste ha standardformen $\mathbf{f}(\mathbf{x}) = 0$. I vårt exempel blir det

$$\begin{aligned} 2x_1 + x_2 - x_1x_2/2 - 2 &= 0 \\ x_1 + 2x_2 - \cos x_2/2 - 1.5 &= 0 \end{aligned}$$

I envariabelfallet beräknas i varje iteration korrektionen $\delta x = -f(x)/f'(x)$ som följs av uppdateringen $x = x + \delta x$. När algoritmen ska generaliseras till ett system med flera obekanta, kommer derivatan $f'(x)$ att ersättas av en matris med partiella derivator, den så kallade jacobianmatrisen $\mathbf{J}$ med elementen $J_{ij} = \partial f_i/\partial x_j$.

Om uttrycket för δx i stället skrivs som sambandet $f'(x)\,\delta x = -f(x)$, övergår det i flervariabelfallet till $\mathbf{J}(\mathbf{x})\,d\mathbf{x} = -\mathbf{f}(\mathbf{x})$. För en bestämd vektor $\mathbf{x}$ är både $\mathbf{f}$ och $\mathbf{J}$ kända numeriskt och korrektionsvektorn $d\mathbf{x}$ erhålls genom lösning av det linjära ekvationssystemet $\mathbf{J}\,d\mathbf{x} = -\mathbf{f}$.

Därefter uppdateras vektorn på vanligt sätt, $\mathbf{x} = \mathbf{x} + d\mathbf{x}$. Förfarandet upprepas tills önskad noggrannhet nås. För vårt exempel blir jacobianmatrisen

$$\mathbf{J} = \begin{pmatrix} 2 - x_2/2 & 1 - x_1/2 \\ 1 & 2 + \sin x_2/2 \end{pmatrix}$$

MATLAB-programmet ser ut på följande sätt.

```
% Litet ickelinjärt system med Newtons metod
  x=[1 0.5]', iter=0;  dxnorm=1;
  while dxnorm>0.5e-4 & iter<10
    f=[2*x(1)+x(2)-x(1)*x(2)/2-2
       x(1)+2*x(2)-cos(x(2))/2-1.5];
    J=[2-x(2)/2   1-x(1)/2
         1    2+sin(x(2))/2];
    dx=-J\f;
    x=x+dx;
    dxnorm=norm(dx,inf), iter=iter+1;
  end
  x, iter
```

Lösningen $x_1 = 0.8431$ och $x_2 = 0.5426$ erhålls redan efter tre iterationer. Den goda egenskapen *kvadratisk konvergens* som finns hos Newton-Raphsons metod i envariabelfallet försvinner inte när antalet obekanta ökar. Vi kan i exemplet ovan notera hur normen för korrektionsvektorn successivt avtar: `dxnorm = 0.1548, 0.0021, 5.4e-07`.

6.9.1 Algoritm för Newtons metod för ett ickelinjärt system

Det ickelinjära systemet med n obekanta ska stå på standardformen $\mathbf{f}(\mathbf{x}) = 0$.

Algoritm för Newtons metod:

- Bestäm de analytiska uttrycken för elementen i jacobianmatrisen $\mathbf{J}(\mathbf{x})$ genom partiella deriveringar
- Utgå från så goda startgissningar som möjligt till de obekanta x_i.
- **(*)** Beräkna $\mathbf{f}$ och $\mathbf{J}$ med insatta x_i-värden
- Lös det linjära systemet $\mathbf{J}\,\delta\mathbf{x} = -\mathbf{f}$ (i MATLAB med `dx= -J\f`)
- Uppdatera: $\mathbf{x} = \mathbf{x} + \delta\mathbf{x}$
- Upprepa från **(*)** tills önskad noggrannhet erhålls.

Metoden konvergerar alltid om startvärdena valts tillräckligt nära roten, och då med kvadratisk konvergens.

6.9.2 Exempel: Interpolerande sinuskurva

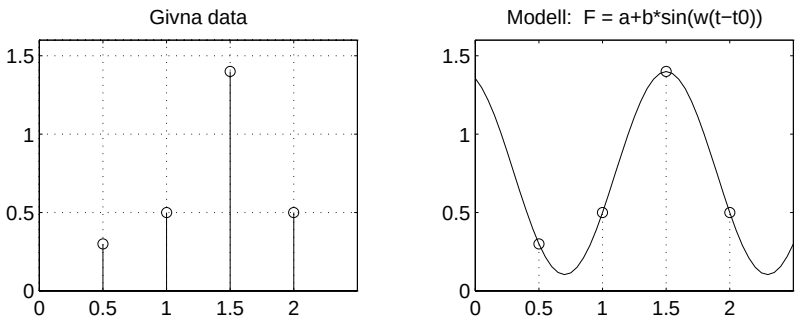
Den trigonometriska modellen $F(t) = a + b\,\sin\omega\,(t - t_0)$ har fyra okända parametrar a, b, ω och t_0. Bestäm parametrarna så att sinuskurvan vid tiderna $t = 0.5,\ 1.0,\ 1.5,\ 2.0$ antar värdena $y = 0.3,\ 0.5,\ 1.4,\ 0.5$.
Det blir ett ickelinjärt system med fyra obekanta och fyra ekvationer:

$$\left.\begin{aligned} a + b\,\sin\omega\,(0.5 - t_0) &= 0.3 \\ a + b\,\sin\omega\,(1.0 - t_0) &= 0.5 \\ a + b\,\sin\omega\,(1.5 - t_0) &= 1.4 \\ a + b\,\sin\omega\,(2.0 - t_0) &= 0.5 \end{aligned}\right\}, \quad \text{dvs} \quad \left.\begin{aligned} a + b\,\sin\omega\,(0.5 - t_0) - 0.3 &= 0 \\ a + b\,\sin\omega\,(1.0 - t_0) - 0.5 &= 0 \\ a + b\,\sin\omega\,(1.5 - t_0) - 1.4 &= 0 \\ a + b\,\sin\omega\,(2.0 - t_0) - 0.5 &= 0 \end{aligned}\right\}$$

på standardform $\mathbf{f}(\mathbf{c}) = 0$, där $\mathbf{c}$ är vektorn med komponenterna a, b, ω, t_0.
Jacobianmatrisens element blir för $i = 1, 2, 3, 4$:

$\partial f_i/\partial a = 1$, $\partial f_i/\partial b = \sin\omega\,(t_i - t_0)$, $\partial f_i/\partial\omega = b\,(t_i - t_0)\cos\omega\,(t_i - t_0)$, $\partial f_i/\partial t_0 = -b\,\omega\cos\omega\,(t_i - t_0)$.

Lämpliga startgissningar till parametrarna skaffar man ur punkternas placeringar tillsammans med kända egenskaper hos sinusfunktionen.

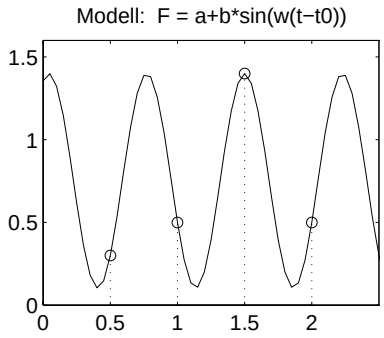


Parametrarna a och b betyder medellinje respektive amplitud. Punkternas lägen antyder att medellinjen hamnar på $a \approx 0.7$ och att amplituden kan vara $b \approx 0.7$. Halva våglängden tycks bli ungefär $t_4 - t_2$, dvs hela våglängden T är cirka 2, vilket ger $\omega = 2\pi/T \approx \pi$. Vid $t = t_0$ gäller att $F(t) = a$, och det ser ut att ske vid $t_0 \approx 1.2$. Iterera tills felet i parametrarna blir mindre än $0.5 \cdot 10^{-5}$.

```
% olinjsin,  ickelinjär modellfunktion, Newtons metod
  t=[0.5  1  1.5  2]';  y=[0.3 0.5 1.4 0.5]';
  subplot(1,2,1), stem(t,y), axis([0 3 0 1.6])
  a=0.7,  b=0.7,  w=pi,  t0=1.2,  c=[a b w t0]';
  iter=0;   dcnorm=1;
  while dcnorm>0.5e-5 & iter<10
    u=w*(t-t0);  f=a+b*sin(u) - y;
    J=[ones(4,1) sin(u) b*(t-t0).*cos(u) -w*b*cos(u)];
    dc=-J\f;  dcnorm=norm(dc,inf),  c=c+dc;  iter=iter+1;
    a=c(1);  b=c(2);  w=c(3);  t0=c(4);
  end, c, iter
  tt=(0:0.05:3)';  Ft=a+b*sin(w*(tt-t0));
  subplot(1,2,2), plot(t,y,'o',tt,Ft), axis([0 3 0 1.6])
```

Parametrarna blir $a = 0.7520$, $b = 0.6480$, $\omega = 3.9404$ och $t_0 = 1.1014$, och det krävs fem iterationer för att få lösningen med önskad precision.

Det finns mer högfrekventa lösningar till problemet. Vi låter $a_{start} = a$ och $b_{start} = b$ men låter startvärdet för ω vara dubbla frekvensen $w_{start} = 2 \cdot 3.94$ och startgissningen för t_0 vara halva värdet ovan, $1.1/2$.
Nu erhålls parametrarna $a = 0.7520$, $b = 0.6480$, $\omega = 8.6259$ och $t_0 = 0.5895$ efter fem iterationer med Newtons metod.



6.9.3 Approximation till jacobianen

Om det är knepigt eller rent av omöjligt att göra analytisk derivering för att få jacobianelementen, kan man approximera de partiella derivatorna med differenskvoter. I sinuskurvproblemet kan till exempel de besvärliga derivatorna $\partial f_i/\partial\omega$ och $\partial f_i/\partial t_0$ ersättas av differensapproximationerna

$$\frac{\partial f_i}{\partial \omega} \approx \frac{f_i(a,b,\omega+\delta\omega,t_0)-f_i(a,b,\omega,t_0)}{\delta\omega}, \quad \frac{\partial f_i}{\partial t_0} \approx \frac{f_i(a,b,\omega,t_0+\delta t_0)-f_i(a,b,\omega,t_0)}{\delta t_0}$$

där $\delta\omega$ och δt_0 väljs lagom litet, till exempel någon procent (eller kanske en promille) av värdet på ω respektive t_0. Man bör experimentera sig fram till lämpligt val i varje enskilt problem.

Med $\delta\omega = 0.01$ och $\delta t_0 = 0.01$ modifieras MATLAB-koden enbart i jacobianberäkningssatsen som ersätts av följande satser:

```
we=w+0.01;  fw=a+b*sin(we*(t-t0))-y;  dfw=(fw-f)/0.01;
te=t0+0.01; ft=a+b*sin(w*(t-te))-y;   dft=(ft-f)/0.01;
J=[ones(4,1)  sin(u)  dfw  dft];
```

Den önskade precisionen i resultatet erhålls efter fem iterationer även i detta fall med numeriskt approximerad jacobian.

> *Exempelsamlingens* **Ex 3.6–3.16** *rekommenderas.*

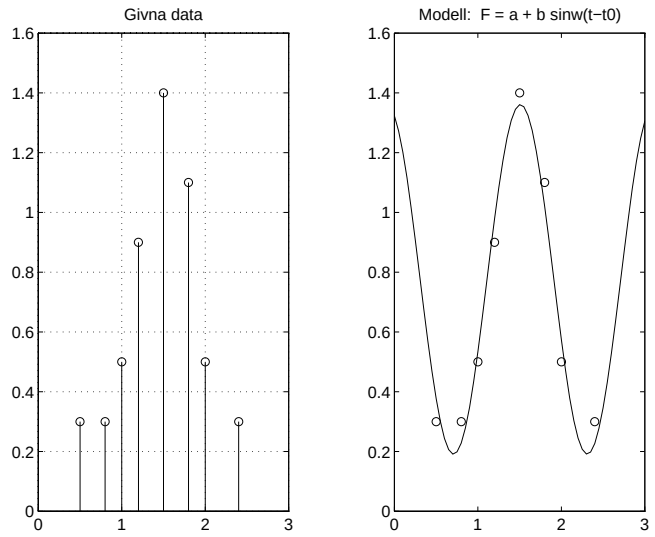
6.10 Ickelinjär modellanpassning

6.10.1 Exempel: Anpassande sinuskurva

Låt oss göra en modifiering av exemplet med sinuskurvan i föregående avsnitt. Den trigonometriska modellen $F(t) = a + b\sin\omega(t-t_0)$ har fyra okända parametrar a, b, ω och t_0. Bestäm parametrarna så att sinuskurvan anpassar sig så väl som möjligt till tabellvärdena

t	0.5	0.8	1.0	1.2	1.5	1.8	2.0	2.4
y	0.3	0.3	0.5	0.9	1.4	1.1	0.5	0.3

Vi har fler mätpunkter än okända parametrar. Det blir ett ickelinjärt ekvationssystem med åtta ekvationer och fyra obekanta:



$$\left.\begin{array}{l} a+b\,\sin\omega\,(0.5-t_0)\approx 0.3 \\ a+b\,\sin\omega\,(0.8-t_0)\approx 0.3 \\ a+b\,\sin\omega\,(1.0-t_0)\approx 0.5 \\ a+b\,\sin\omega\,(1.2-t_0)\approx 0.9 \\ a+b\,\sin\omega\,(1.5-t_0)\approx 1.4 \\ a+b\,\sin\omega\,(1.8-t_0)\approx 1.1 \\ a+b\,\sin\omega\,(2.0-t_0)\approx 0.5 \\ a+b\,\sin\omega\,(2.4-t_0)\approx 0.3 \end{array}\right\}, \quad \text{dvs} \quad \left.\begin{array}{l} a+b\,\sin\omega\,(0.5-t_0)-0.3\approx 0 \\ a+b\,\sin\omega\,(0.8-t_0)-0.3\approx 0 \\ a+b\,\sin\omega\,(1.0-t_0)-0.5\approx 0 \\ a+b\,\sin\omega\,(1.2-t_0)-0.9\approx 0 \\ a+b\,\sin\omega\,(1.5-t_0)-1.4\approx 0 \\ a+b\,\sin\omega\,(1.8-t_0)-1.1\approx 0 \\ a+b\,\sin\omega\,(2.0-t_0)-0.5\approx 0 \\ a+b\,\sin\omega\,(2.4-t_0)-0.3\approx 0 \end{array}\right\}$$

Om ω och t_0 vore kända, skulle den vänstra formen ovan passa bäst. Parametrarna a och b förekommer linjärt och minstakvadratmetoden skulle kunna tillämpas direkt på det linjära överbestämda systemet $\mathbf{Ac} \approx \mathbf{y}$, där $\mathbf{A}$ har ettor i första kolumnen och värdena $\sin\omega\,(t_i - t_0)$ i andra kolumnen. Vektorn $\mathbf{c}$ har komponenterna $c_1 = a$ och $c_2 = b$.

Från kapitel 2 vet vi att minstakvadratlösningen i MATLAB erhålls med `c=A\b`, och att den har egenskapen att minimera felkvadratsumman $\mathbf{r}^T\mathbf{r} = \|\mathbf{r}\|_2^2 = \|\mathbf{y} - \mathbf{Ac}\|_2^2$.

Men ω och t_0 är okända i vårt exempel, så systemet kan inte skrivas på den trevliga matrisformen. Vi väljer det högra skrivsättet ovan, där y-värdena flyttats över till vänster sida. Det överbestämda ickelinjära systemet står nu på formen $\mathbf{f}(\mathbf{c}) \approx 0$, och vektorn $\mathbf{c}$ har komponenterna a, b, ω, t_0 precis som i den interpolerande sinuskurvan i avsnitt 6.9.2.

Det går inte att åstadkomma en sinuskurva som interpolerar genom alla

åtta punkterna, utan vi får nöja oss med att bestämma parametrarna så att avvikelsen mellan mätvärdena och modellkurvan blir så liten som möjligt i en väl vald norm. Den norm som lämpar sig bäst är liksom vid linjär modellanpassning den euklidiska normen.
Önskemålet är att minimera $\|\mathbf{y}-\mathbf{F}\|_2$ där vektorn $\mathbf{F}$ betyder modellkurvans värden i de åtta t_i-värdena för en viss parameteruppsättning a, b, ω, t_0.

Men $\mathbf{F}-\mathbf{y}$ är detsamma som vänsterledet $\mathbf{f}$ i det överbestämda systemet $\mathbf{f}(\mathbf{c}) \approx 0$. Vi vill alltså hitta en algoritm att bestämma parametrarna så att $\|\mathbf{f}\|_2$ minimeras (annorlunda uttryckt: felkvadratsumman $\mathbf{f}^T\mathbf{f}$ minimeras).

När det ickelinjära systemet har lika många ekvationer som obekanta — såsom i exemplet med den interpolerande sinuskurvan – utnyttjas Newtons metod för att lösa systemet. Då får vi en lösning som i teorin uppfyller $\|\mathbf{f}\|_2 = 0$. I praktiken avbryts iterationerna när parametrarna bestämts med viss önskad noggrannhet, vilket innebär att $\|\mathbf{f}\|_2$ inte blir exakt noll — men i princip hur nära som helst med datorprecision.

Vid modellanpassningsproblem där antalet ekvationer överstiger antalet obekanta går det däremot inte att finna en lösning som satisfierar alla ekvationerna. Därför kan vi inte, hur gärna vi än vill, åstadkomma att avvikelsenormen blir noll. Med en dåligt vald modellfunktion blir avvikelserna avsevärda, trots minimering av $\|\mathbf{y}-\mathbf{F}\|_2$.

6.10.2 Gauss-Newtons metod

Den mest använda metoden för lösning av överbestämda ickelinjära system är Gauss-Newtons metod som kan beskrivas som en lätt modifiering av Newtons metod i kombination med minstakvadratmetoden. Algoritmen börjar precis som Newtons metod med att man beräknar uttrycken för jacobianmatrisens element. Eftersom systemet i vårt exempel har åtta ekvationer blir $\mathbf{J}$ en 8×4 matris med partiella derivator enligt uttrycken i exemplet i 6.9.2: $\partial f_i/\partial a = 1$, $\partial f_i/\partial b = \sin\omega\,(t_i - t_0)$, $\partial f_i/\partial \omega = b\,(t_i - t_0)\cos\omega\,(t_i - t_0)$, $\partial f_i/\partial t_0 = -b\,\omega\cos\omega\,(t_i - t_0)$.
Beräkna först vektorn $\mathbf{f}$ och matrisen $\mathbf{J}$ med insatta startgissningar på parametrarna.

Nästa steg i Newtons metod är att lösa systemet $\mathbf{J}\,\delta\mathbf{c} = -\mathbf{f}$. Det blir nu fråga om ett överbestämt linjärt ekvationssystem, där likhetstecknet måste ersättas av approximativ likhet, alltså $\mathbf{J}\,\delta\mathbf{c} \approx -\mathbf{f}$, och minstakvadratmetoden tillämpas för bestämma korrektionsvektorn $\delta\mathbf{c}$. I MATLAB görs det med `dc=-J\f`, men vid handräkning hade man fått lösa normalekvationerna $\mathbf{J}^T\mathbf{J}\,\delta\mathbf{c} = -\mathbf{J}^T\mathbf{f}$.

Uppdatera $\mathbf{c}$ och upprepa förfarandet som alltid vid en iterativ process. När ska man avbryta? Ja, önskemålet är ju att finna den lösning som minimerar $\|\mathbf{f}\|_2$ (eller identiskt: finna den lösning som minimerar felkvadratsumman $\|\mathbf{f}\|_2^2 = \mathbf{f}^T\mathbf{f}$).

Ickelinjär modellanpassning är alltså i grunden ett minimeringsproblem. Därför bör vi i lösningsalgoritmen skriva ut och studera värdet av `norm(f)` (eller felkvadratsumman `f'*f`) i varje iteration. Ett enkelt förfarande är att iterera till exempel fem gånger i den första exekveringen, kolla om normen stabiliserats vid sitt minimum, annars köra om programmet med några extra iterationssteg.

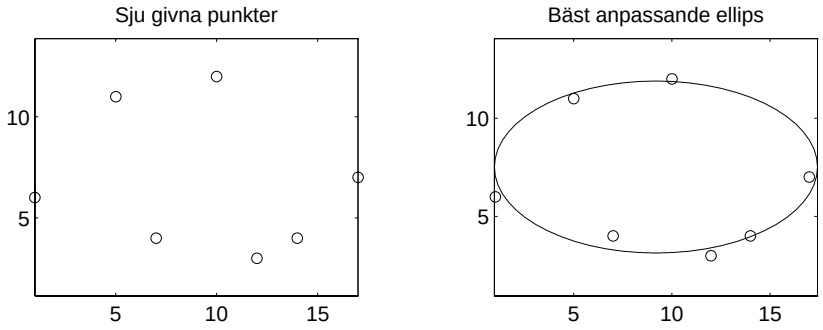
Det är viktigt att hitta goda startgissningar till de okända parametrarna. I vårt exempel väljer vi samma startvärden som i sinuskurvexemplet tidigare (eftersom fyra av de åtta mätpunkterna här är samma som där). Med dåliga startgissningar på parametrarna riskerar man förstås divergens.

```
% olinj8sin, ickelinjär modellanpassning
  n=8,
  t=[0.5  0.8  1  1.2  1.5 1.8  2  2.4]';
  y=[0.3  0.3 0.5 0.9  1.4 1.1 0.5 0.3]';
  subplot(1,2,1), stem(t,y), axis([0 3 0 1.6])
  a=0.7;  b=0.7;  w=pi;  t0=1.2;
  c=[a b w t0]';
  disp('    norm(f)')
  for iter=1:5                    % pröva om fem iterationer räcker
    u=w*(t-t0);
    f=a+b*sin(u) - y;
    disp(norm(f))                 % kolla att norm(f) avtar mot ett minimum
    J=[ones(n,1) sin(u) b*(t-t0).*cos(u) -w*b*cos(u)];
    dc=-J\f;  c=c+dc;
    a=c(1);  b=c(2);  w=c(3);  t0=c(4);
  end
  c, iter
  tt=(0:0.05:3)'; Ft=a+b*sin(w*(tt-t0));
  subplot(1,2,2), plot(t,y,'o',tt,Ft), axis([0 3 0 1.6])
```

Den bäst anpassande sinuskurvan till de åtta punkterna är inritad i figuren. Parametrar: $a = 0.7761$, $b = 0.5850$, $\omega = 3.9225$, $t_0 = 1.1092$. Successiva värden på `fnorm`: `0.8034, 0.3688, 0.2117, 0.1928, 0.1928`.

6.10.3 Exempel: Ellipsanpassning till givna punkter

Vi har ett antal punkter givna approximativt kring periferin på en ellips som har axlarna parallella med x- och y-axeln. Problemet är att hitta den ellips som anpassar sig bäst till de sju punkterna i vänstra figuren. Ellipsens



ekvation lyder $(x-x_c)^2/a^2 + (y-y_c)^2/b^2 = 1$, där x_c, y_c är mittpunktskoordinaterna, a och b är ellipsens halvaxlar. Det är alltså fyra okända parametrar att bestämma, `p=[xc yc a b]`. Det blir ett ickelinjärt överbestämt ekvationssystem som ska ha standardformen $\mathbf{f}(\mathbf{p}) \approx 0$ för att vi ska kunna tillämpa Gauss-Newtons metod på det.

Genom flyttning av högerledets etta till vänsterledet erhålls önskad form. Vi behöver startgissningar till parametrarna, men alla har enkel geometrisk innebörd så det är inte svårt att hitta goda startvärden genom en blick på vänstra figuren.

```
% ellipsmod,  ellipsanpassning
  x=[1 7 10 17  5 12 14]';  y=[6 4 12  7 11  3  4]';
  xc=10;  yc=8;  a=8;  b=3;  p=[xc yc a b]';
  for iter=1:5
    xd=x-xc;  yd=y-yc;
    f=xd.^2/a^2+yd.^2/b^2-1;
    disp(norm(f))                                      % skriv ut norm(f)
    J=-2*[xd/a^2  yd/b^2  xd.^2/a^3  yd.^2/b^3];       % jacobianmatris
    dp=-J\f;  p=p+dp;
    xc=p(1);  yc=p(2); a=p(3); b=p(4);
  end, p
  v=0:2*pi/60:2*pi;
  plot(x,y,'o', xc+a*cos(v),yc+b*sin(v)), axis equal
```

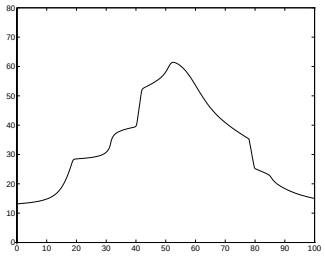
Parametrarna blir $x_c = 9.1879$, $y_c = 7.5159$, $a = 8.2298$, $b = 4.3817$.
$\|\mathbf{f}\|_2$ avtar enligt: 2.5619, 0.8640, 0.4087, 0.3848, 0.3848.
Fem iterationer är tillräckligt. Den bäst anpassande ellipsen till de sju punkterna är inritad i högra figuren.

Ex 4.22–4.26 *i exempelsamlingen är lämpliga övningar.*

7 MAXIMERING OCH MINIMERING

7.1 Maximering/minimering av unimodala funktioner

En funktion $F(x)$ är unimodal i ett intervall, om den har en enda extrempunkt i intervallet. Funktionen behöver inte vara deriverbar — inte ens kontinuerlig. Vid sökning efter en maximipunkt gäller det att ha ett startintervall $a \leq x \leq b$ där funktionskurvan har exakt *en* topp någonstans i det inre.

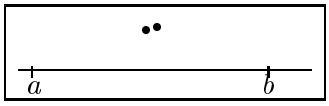


Vid minimeringsproblem för en unimodal funktion ska startintervallet vara så snävt tilltaget att bara ett mimimum $(x^*, F(x^*))$ finns där, dvs funktionen ska vara avtagande för $a \leq x \leq x^*$ och växande för $x^* \leq x \leq b$.

Algoritmen för lösning av den här typen av optimeringsproblem får inte innehålla derivator, eftersom funktionen inte säkert är deriverbar.

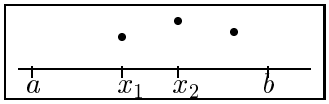
7.1.1 Gyllenesnittetsökning

Anta att vi söker ett maximum mellan a och b.



Det är lockande att som i intervallhalvering börja med att beräkna funktionsvärdet mitt emellan. Men sedan vet vi fortfarande inte alls på vilken sida maximum finns. Man måste beräkna *två* funktionsvärden i mitten för att bestämma rätt halva (andra halvan förkastas alltså), sedan två funktionsvärden mitt i denna halva etc. Kan man inte komma ifrån dessa dubbla funktionsberäkningar?
Jo, med gyllenesnittetsökning!

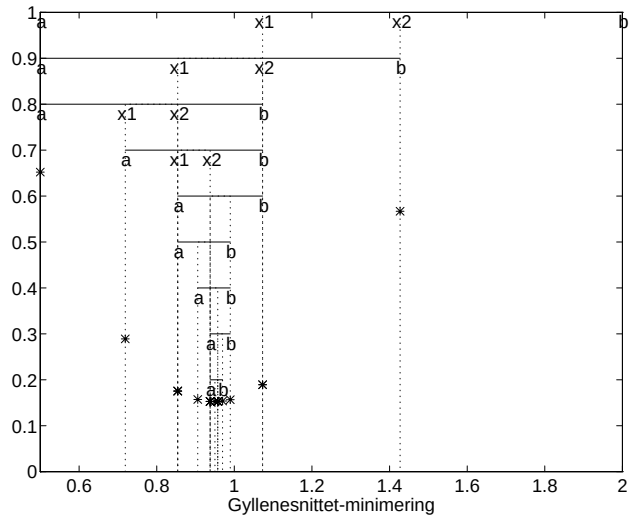
Man glesar ut de båda utnyttjade x-värdena i



mitten symmetriskt kring intervallets mittpunkt just så mycket att det ena kan återanvändas som inre punkt i det nya sökintervallet. Proportionerna ska då vara

$$\frac{b - x_1}{b - a} = \frac{b - x_2}{b - x_1}$$

Inför beteckningen r_g för förhållandet $(b-x_1)/(b-a)$; då gäller av symmetriskäl att $(x_2-a)/(b-a) = r_g$, vilket i sin tur medför att $(b-x_2)/(b-a) = 1-r_g$.



Vänsterledet i sambandet ovan är alltså r_g, medan högerledet $(b-x_2)/(b-x_1)$ kan uttryckas som $(1-r_g)/r_g$. Det ger:

$$r_g = \frac{1-r_g}{r_g} \implies r_g^2 + r_g - 1 = 0 \implies r_g = \frac{\sqrt{5}-1}{2} = 0.6180\ldots$$

Denna proportion har av konstnärer i alla epoker utnämnts till den skönaste, därav namnet *gyllene snittet*, men dessutom är den alltså effektivast.

Metoden fungerar lika bra för minimering som för maximering av unimodala funktioner. Algoritmen för minimering med gyllenesnittetsökning lyder:

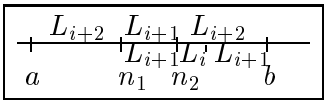
- Bilda $r_g = (\sqrt{5}-1)/2$ och $q_g = 1 - r_g$
- Utgå från ett intervall $a \le x \le b$ där $F(x)$ är unimodal
- Bilda $x_1 = a + q_g\,(b-a),\ F_1 = F(x_1),\ x_2 = a + r_g\,(b-a),\ F_2 = F(x_2)$
- (*) Om $F_1 < F_2$ utförs följande:
  $b = x_2,\ x_2 = x_1,\ F_2 = F_1,\ x_1 = a + q_g\,(b-a),\ F_1 = F(x_1)$
  annars utförs:
  $a = x_1,\ x_1 = x_2,\ F_1 = F_2,\ x_2 = a + r_g\,(b-a),\ F_2 = F(x_2)$
- Upprepa från (*) tills intervallet $[a,\, b]$ ligger inom tillåten felgräns.

Vid maximering blir enda ändringen att villkoret $F_1 < F_2$ byts mot $F_1 > F_2$.

7.1.2 Fibonaccisökning

Gyllenesnittetsökning förutsätter att den unimodala funktionen kan beräknas för alla x i startintervallet $a \leq x \leq b$. Ibland är emellertid F känd bara i vissa punkter, och sökpunkterna måste hålla sej till detta punktgitter. Vi betraktar fallet att funktionen är definierad för heltalen $n = 0, 1, 2, \ldots$, och att $F(n)$ är unimodal med ett maximum i intervallet $0 < n < N$. Enklast är att lägga upp F som en vektor och i MATLAB skriva satsen `[Fmax,index]=max(F)`.

För mer komplicerade funktionsuttryck $F(n)$ behövs en effektiv algoritm som inte kräver att alla vektorelement beräknas. Liksom i gyllenesnittetsökning förkastar man i varje iterationssteg det intervall där maximum inte kan finnas — antingen ska a flyttas åt höger eller b åt vänster.



Nu måste sökintervallängderna vara *heltal*, $L_1, L_2, L_3, \ldots$, där vi låter $L_1 = 1$ vara det minsta (sist utnyttjade) intervallet, som ju måste gå mellan två på varandra följande heltal just där maximum finns. Av den lilla figuren ovan (som visar ett iterationssteg) framgår att $L_{i+2} = L_i + L_{i+1}$, den kända rekursionsformeln för *fibonaccitalen*

$$1,\ 1,\ 2,\ 3,\ 5,\ 8,\ 13,\ 21,\ 34,\ 55,\ 89,\ 144,\ 233, \ldots$$

Man börjar med ett intervall vars längd är ett stort fibonaccital, t ex 144, placerar n_2 vid 89 och n_1 vid 55. Undersök vilket som är störst av $F(n_1)$ och $F(n_2)$, förkasta intervallet mellan 0 och 55 om $F(89)$ är störst. Kvar finns intervallet mellan 55 och 144 med känt funktionsvärde vid 89. Avståndet mellan 55 och 89 är fibonaccitalet 34; det avståndet ska också finnas symmetriskt från andra hållet: $144 - 34 = 110$ som blir nytt n_2.

Fortsätt att reducera intervallet med hjälp av successiva mindre fibonaccital som sökintervallängder. Se utskriften efter följande MATLAB-program som med fibonaccisökning finner maximum av en funktion som är unimodal för n-värden mellan 0 och 144.

```
a=0; b=144;  n2=89;  F2=Ff(n2);  n1=a+b-n2;  F1=Ff(n1);
disp('     a    n1    n2    b '), disp([a  n1  n2  b])
while b-a>1
  if F1>F2, b=n2; n2=n1; F2=F1; n1=a+b-n2; F1=Ff(n1);
     else   a=n1; n1=n2; F1=F2; n2=a+b-n1; F2=Ff(n2);
  end
  disp([a  n1  n2  b])
end
maxF=[F1 F2]
```

Utskriften blir i detta fall:

```
   a      n1      n2      b
   0      55      89    144
  55      89     110    144
  89     110     123    144
  89     102     110    123
  89      97     102    110
  97     102     105    110
  97     100     102    105
 100     102     103    105
 100     101     102    103
 101     102     102    103
 102     102     103    103
maxF = 4.6630   4.6629
```

Maximum är instängt mellan $n_1 = 102$ och $n_2 = 103$. De två kandidaterna till maximumvärdet visas i `maxF`.
(Funktionen i demonstrationsexemplet ovan är $F(n) = e^{\sqrt{n}/10} - \cos(n/30+19)$.)

När gyllenesnittetsökning och fibonaccisökning båda kan användas är de likvärda. Det gäller nämligen att $L_i/L_{i+1} \approx (\sqrt{5}-1)/2$, med allt bättre överensstämmelse för ökande i-värden, till exempel är $34/55 = 0.6182$ och $55/89 = 0.6180$.

7.2 Maximering av funktioner i flera variabler

7.2.1 Maximering med användning av Newtons metod

Ofta kan man finna maximipunkter till en funktion genom att derivera den och sätta derivatorna lika med noll. För en funktion av flera variabler, $Q(x_1, x_2, \ldots, x_n)$, leder det till det ickelinjära ekvationssystemet

$$\operatorname{grad} Q = 0.$$

För att lösa det finns Newtons snabbt konvergerande iterationsmetod som dock kräver goda startvärden till de sökta koordinaterna.

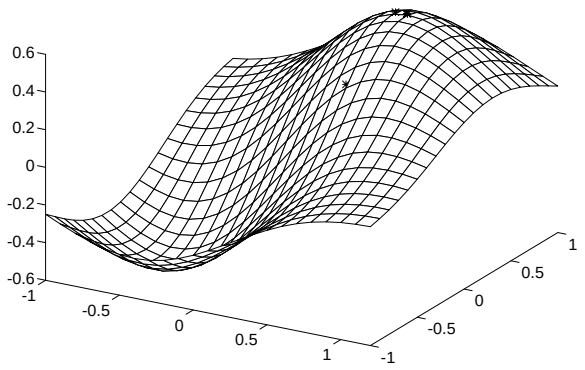
Exempel: Finn maximum till $Q(x,y) = (x + \sin y)e^{-x^2-y^2}$

Bestäm gradienten, dvs derivera med avseende på x och på y:

$$\begin{array}{l} \partial Q/\partial x = (1 - 2x(x+\sin y))e^{-x^2-y^2} \\ \partial Q/\partial y = (\cos y - 2y(x+\sin y))e^{-x^2-y^2} \end{array} \quad \Longrightarrow \quad \left\{ \begin{array}{l} 2x^2 + 2x\sin y - 1 = 0 \\ 2xy + 2y\sin y - \cos y = 0 \end{array} \right.$$

vilket är ett ickelinjärt system av typen $\mathbf{f}(\mathbf{x}) = \mathbf{0}$. Vid lösning med Newtons metod behövs jacobianmatrisen $\mathbf{J}$ som i detta fallet blir

$$\mathbf{J} = \begin{pmatrix} 4x + 2\sin y & 2x\cos y \\ 2y & 2x + 3\sin y + 2y\cos y \end{pmatrix}$$



Figuren visar rymdytan $Q(x,y)$ för $-1 \leq x \leq 1.2$ och $-1 \leq y \leq 1$, och vi ser att maximipunkten finns i trakten av $x = 0.5,\ y = 0.5$.
Ett MATLAB-program för maximeringsproblemet följer här. Det ickelinjära ekvationssystemet grad $Q = 0$ löses med Newtons metod. Som startvärden väljs $x = 0.4$ och $y = 0$ (medvetet halvbra val). I varje iterationssteg skrivs värdet $Q(x,y)$ ut. Om det ökar successivt har vi kontroll på att algoritmen verkligen leder till önskat maximum.

```
% Maximum till  Q(x,y) = (x+sin(y))*exp(-x^2-y^2)  med
% Newtons metod på grad Q = 0
  x=0.4, y=0
  Q=(x+sin(y))*exp(-x^2-y^2)
  p=[x y]';  dpnorm=1;  iter=0;
  while dpnorm>1e-5
    s=x+sin(y);
    f=[1-2*x*s                              % f=grad(Q) utom exp-delen
       cos(y)-2*y*s];
    J=[-4*x-2*sin(y)  -2*x*cos(y)           % jacobianen till f
        -2*y  -2*x-3*sin(y)-2*y*cos(y)];
    dp=-J\f; p=p+dp;
    dpnorm=norm(dp,inf), iter=iter+1;
    x=p(1); y=p(2);
    Q=(x+sin(y))*exp(-x^2-y^2)              % skriv ut Q i varje iteration
  end
  xy_maxkoord=p',
  Qmax=Q, iter
```

Efter sex iterationer erhålls koordinatvärdena $x = 0.5181$ och $y = 0.4634$. Maximumvärdet är $Q_{max} = 0.5953$.

Ex 4.27, 4.28 *och* **Ex 3.14** *i exempelsamlingen är övningar på optimeringsproblem.*

7.2.2 Maximering utan beräkning av jacobianmatris

I vårt exempel var det ett överkomligt problem att beräkna jacobianmatrisen till det ickelinjära ekvationssystemets vänsterled. I många maximerings- och minimeringsproblem blir det så jobbigt att man hellre vill utnyttja en algoritm utan jacobianberäkning.

Sökning efter ett maximum kan då gå till på följande sätt. Man gissar en startpunkt $\mathbf{x}_0$ och väljer en sökriktning, vanligen genom att beräkna gradienten, eftersom gradientvektorn anger den riktning i vilken Q växer snabbast.

När riktningen är fixerad gör man endimensionell gyllenesnittetsökning från $\mathbf{x}_0$ längs denna sökriktning och får fram en punkt $\mathbf{x}_1$ som ger största Q-värde just i den riktningen. Beräkna gradienten i punkten $\mathbf{x}_1$ som ger ny sökriktning. Gyllenesnittetmaximera igen, osv.

Om det gäller minimering ska sökriktningen i stället väljas som negativa gradienten. Metoden kallas sedan gammalt **steepestdescentmetoden**. Den och andra minimeringsmetoder behandlas i Nadas fortsättningskurser *Tillämpade numeriska metoder 1* (4p) och *Tillämpade numeriska metoder 2* (6p) men också i matematikinstitutionens kurser i optimeringslära.

8 DIFFERENTIALEKVATIONER

8.1 Allmänt om ODE

Vi ska studera numerisk behandling av ordinära differentialekvationer , ODE, alltså ekvationer som innehåller derivator av den okända funktionen $y(t)$. Ett begynnelsevärdesproblem för en första ordningens differentialekvation definieras av

$$dy/dt = f(t, y), \quad y(a) = y_0. \text{ Sökt: } y(t) \text{ för } t > a$$

Startpunkten $(a,\ y_0)$ är given, och f är ett känt funktionsuttryck i t och y.

Alla numeriska metoder som vi ska betrakta är så kallade stegmetoder. Man stegar sig fram i t-led med ett visst litet steg h, och för dessa t_i-värden, $t_0,\ t_1,\ t_2, \ldots,$ med $t_{i+1} = t_i + h,$ beräknas en approximation y_i till $y(t_i)$.

8.2 Eulers metod och Runge-Kuttas metod RK4

Först ut är **Eulers metod** som för en första ordningens differentialekvation $y' = f(t, y)$ har den enkla formen:

$$y_{i+1} = y_i + hf(t_i, y_i)$$

där y_{i+1} är en approximation till lösningen då variabeln t flyttats steget h, alltså vid $t_{i+1} = t_i + h$. Härledning kan göras genom taylorutveckling:

$$y(t_i + h) = y(t_i) + h\,y'(t_i) + h^2 y''(t_i)/2 + \cdots =$$

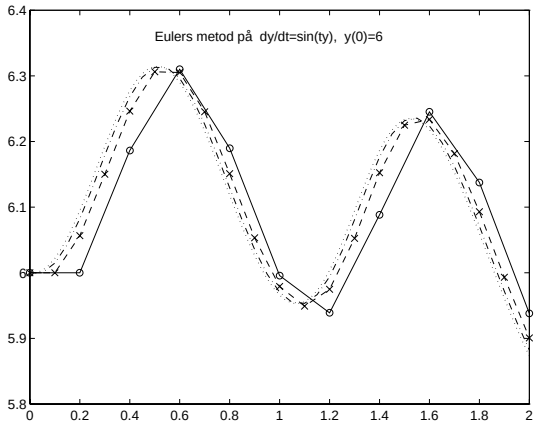
$$y(t_i) + hf(t_i, y(t_i)) + h^2 y''(t_i)/2 + \cdots = y_i + hf(t_i, y_i) + h^2 y''(t_i)/2 + \cdots .$$

Genom att avbryta före h^2-termen får man Eulers metod. Ett annat sätt att nå fram till denna enkla metod är via approximation av derivatan med en framåtdifferenskvot, $y'(t_i) = f(t_i, y_i) \approx (y(t_i + h) - y(t_i))/h$.

Ommöblering av termerna ger Eulers metod som kallas för en enstegsmetod, eftersom värdet vid t_{i+1} bara utnyttjar information från närmast föregående t-värde. Geometriskt sett beräknar man lutningen i punkten (t_i, y_i) och låter den vara konstant ett steg h framåt.

Låt oss ta ett litet exempel, differentialekvationen $y' = \sin(t\,y)$ med begynnelsevärdet $y(0) = 6$. Problemet är att beräkna lösningskurvan $y(t)$ i intervallet $0 \le t \le 2$. Intressant är också att titta på resultatet vid slutpunkten $t = 2$ och studera hur tillförlitligheten beror av metodvalet och steglängden. Vilket steg som är lagom kan vara svårt att veta i förväg. Vi prövar $h = 0.2$, vilket innebär tio steg fram till $t_{slut} = 2$.

```
% Eulers metod på y'=sin(ty), y(0)=6
  tslut=2;
  t=0; y=6;  T=t; Y=y;
  h=0.2; n=tslut/h;
  for i=1:n
    f=sin(t*y); y=y+h*f; t=t+h; T=[T; t]; Y=[Y; y];
  end, y
  plot(T,Y, T,Y,'o')
```



De tio beräknade värdena är markerade med en ring och däremellan är den kantiga kurvan heldragen.

Eulers metod är ett verktyg som yxar till resultatet ganska grovt. För att få vetskap om hur grovt (eller hur tillförlitligt) det blivit, är det säkrast att göra om hela beräkningen med samma enkla metod men med halverad steglängd några gånger. Med $h = 0.1$ är kurvan streckad, med $h = 0.05$ streckprickad och med $h = 0.025$ prickad. Resultatet vid $t_{slut} = 2$ blir med de successivt halverade steglängderna: 5.9379, 5.9007, 5.8823, 5.8731.

Härledningen av Eulers metod via två termer i taylorutvecklingen ger ett trunkeringsfel som är proportionellt mot h^2. Det kallas för lokalt fel. I varje steg uppkommer ett nytt lokalt fel. Det innebär att i y_{slut}-värdet har det accumulerats $n = t_{slut}/h$ stycken lokala fel, vilket leder till att det så kallade globala felet i Eulers metod blir proportionelt mot h.

Man säger att Eulers metod har första ordningens noggrannhet och den (globala) felutvecklingen är av typen $c_1 h + c_2 h^2 + \cdots$.

Richardsonextrapolation kan användas för att ge bättre noggrannhet. Det sista värdet ovan, 5.8731, bör vi kunna förbättra genom att till värdet addera korrektionstermen $\Delta/(2^p - 1) = \frac{5.8731-5.8823}{2-1} = -0.0092$ (p har värdet ett eftersom detta är första h-potensen i felutvecklingen). Alltså, en bättre approximation till $y(2)$ är det extrapolerade värdet $5.8731 - 0.0092 = 5.8639$.

Runge-Kutta–metoder gör flera funktionsberäkningar i varje steg och får därigenom högre noggrannhet. Den enklaste av dem är RK2, en andra ordningens rungekuttametod som även benämns Heuns metod.

I RK2-algoritmen utnyttjas dels lutningen i punkten (t_i, y_i) – precis som i Euler – dels ett approximativt lutningsvärde i nästa punkt, alltså vid $t_{i+1} = t_i + h$. Medelvärdet av de båda lutningarna leder till följande beräkningsformel:

$$y_{i+1} = y_i + \frac{h}{2}(f_1 + f_2), \quad \text{där} \quad \begin{cases} f_1 = f(t_i,\, y_i) \\ f_2 = f(t_i{+}h,\, y_i{+}hf_1) \end{cases}$$

Felet i $y(t_{slut})$ är proportionellt mot steget i kvadrat, alltså $O(h^2)$, i stället för $O(h)$ som gäller för Eulers metod.

RK2 och Eulers metod är båda teoretiskt intressanta — de visar grundstenarna för numerisk lösning av differentialekvationer, men de har inte så stor praktisk användning. I verkliga tillämpningar väljer man hellre en högre ordningens metod såsom RK4, vars algoritm bara är aningen mer omfattande men som är betydligt noggrannare.

Mest känd är därför RK4, en fjärde ordningens rungekuttametod, som härleddes omkring år 1900 av de tyska matematikerna Carl Runge och Wilhelm Kutta – oberoende av varandra sägs det. Liksom Eulers metod och RK2 är RK4 en enstegsmetod. Den använder den sig av ett viktat medelvärde av fyra lutningsberäkningar — förutom i ändpunkterna t_i och $t_i + h$ även i mitten vid $t_i{+}h/2$ — allt enligt formeln:

$$y_{i+1} = y_i + \frac{h}{6}(f_1 + 2f_2 + 2f_3 + f_4), \quad \text{där} \quad \begin{cases} f_1 = f(t_i,\, y_i) \\ f_2 = f(t_i{+}h/2,\, y_i{+}hf_1/2) \\ f_3 = f(t_i{+}h/2,\, y_i{+}hf_2/2) \\ f_4 = f(t_i{+}h,\, y_i{+}hf_3) \end{cases}$$

Såsom namnet RK4 antyder gäller att det globala felet är proportionellt mot steget h höjt till fyra. Runge och Kutta påvisade att metodens felutveckling kan skrivas $c_1h^4 + c_2h^5 + \ldots$

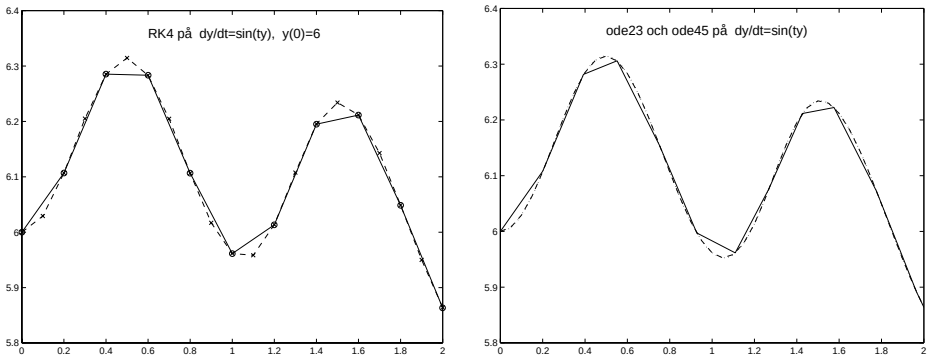
Vi prövar RK4 med steget $h = 0.2$ på vårt differentialekvationsproblem $y' = \sin(t\,y)$ med begynnelsevärdet $y(0) = 6$. Det är lämpligt att skriva en MATLAB-funktion för differentialekvationens högerled:

```
function f=fsin(t,y)
   f=sin(t*y);
```

Programmet som beräknar och ritar lösningen med RK4 blir:

```
% RK4 på y'=sin(ty)
  tslut=2;  t=0; y=6;  T=t; Y=y;
  h=0.2;
  while t<tslut-h/2
    f1=fsin(t, y);
    f2=fsin(t+h/2, y+h*f1/2);
    f3=fsin(t+h/2, y+h*f2/2);
    f4=fsin(t+h, y+h*f3);
    y=y+h/6*(f1+2*f2+2*f3+f4); t=t+h;  T=[T; t]; Y=[Y; y];
  end, y
  plot(T,Y, T,Y,'o')
```

Resultatet vid $t=2$ med Runge-Kuttas metod och $h=0.2$ blir 5.8635. Om vi exekverar programmet igen med ändring av steget till $h=0.1$ erhålls slutvärdet 5.8639. Värdena avviker endast 0.0004; en extrapolation här innebär att korrektionstermen $0.0004/(2^4-1)$ ska adderas till värdet 5.8639. Det påverkar inte sista decimalen, vilket innebär att $y(2)=5.8639$ med fem siffrors noggrannhet. Kurvan med $h=0.1$ är streckad i vänstra figuren.



Nu har vi betraktat några metoder där *konstant steglängd* utnyttjats i hela intervallet. I MATLAB finns flera differentialekvationslösare, bland annat `ode23` och `ode45` som är varianter av rungekuttametoder men försedda med *automatisk steglängdsreglering*. I standardanropet gäller en relativ noggrannhet på 10^{-3}, men man kan via en `odeset`-sats sätta en annan tolerans. Mer information ges av `help ode23`, `help ode45` och `help odeset`.

```
  tslut=2;  y0=6;
  [T,Y]=ode23(@fsin,[0 tslut],y0); ys=Y(end), plot(T,Y), hold on

  [T,Y]=ode45(@fsin,[0 tslut],y0); ys=Y(end), plot(T,Y,'--')

  tol=odeset('RelTol',1e-8);
  [T,Y]=ode45(@fsin,[0 tslut],y0,tol); ys=Y(end), plot(T,Y,'c:'),
```

```
% Med ode23:              ys = 5.8640     (13 Y-värden beräknas)
%     ode45:              ys = 5.8639     (41 Y-värden beräknas)
% ode45 med reltol 1e-8:  ys = 5.8639379  (69 Y-värden beräknas)
```

8.3 Rävar och kaniner – ett differentialekvationssystem

Följande system av differentialekvationer är en modell för växelspelet mellan rävar och kaniner i naturen:[1]

$$\begin{cases} dk/dt = 2k - 0.01\,k\,r \\ dr/dt = -r + 0.01\,k\,r \end{cases}$$

Här betecknar t tiden, $k = k(t)$ antalet kaniner och $r = r(t)$ antalet rävar. Låt det finnas 300 kaniner och 150 rävar vid beräkningens början. Undersök hur beståndet av kaniner och rävar ändras med tiden.

Eulers metod, RK2, RK4, `ode23`, `ode45` kan allesammans användas på system av första ordningens differentialekvationer. Det gäller bara att först skriva ode-systemet i vektorform.

Inför en vektor $\mathbf{y}$ med komponenterna $y_1 = k$ och $y_2 = r$. Nu kan systemet av kaniner och rävar skrivas $\mathbf{y}' = \mathbf{f}(t, \mathbf{y})$ där vektorn $\mathbf{f}$ har komponenterna $f_1 = 2y_1 - 0.01y_1y_2$ och $f_2 = -y_2 + 0.01y_1y_2$.

Differentialekvationen är *autonom* (självstyrande), det betyder att tiden inte finns explicit i högerledet. Ett autonomt differentialekvationssystem kan alltså skrivas $\mathbf{y}' = \mathbf{f}(\mathbf{y})$. Vår MATLAB-funktion kan då bli så här:

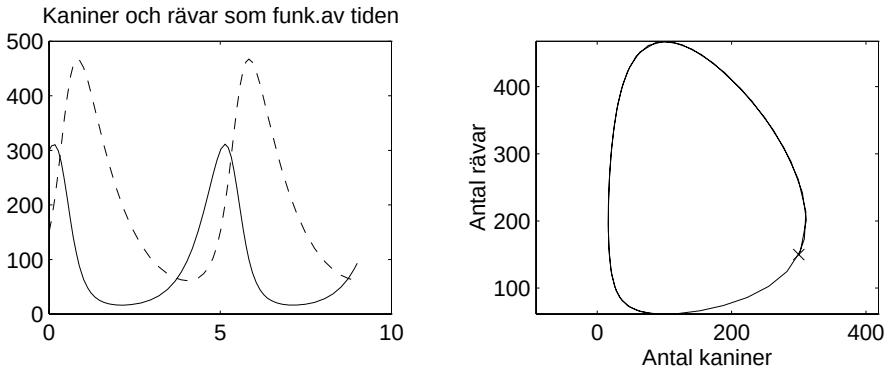
```
function f=fkr(y)
   f=[2*y(1)-0.01*y(1)*y(2)  -y(2)+0.01*y(1)*y(2)];
```

I en egen RK4-lösning kan vi utnyttja funktionen skriven så (se programkoden längre fram). En lösning med MATLABs `ode45` kräver två parametrar i funktionen; t-parametern måste finnas med. Ytterligare ett krav är att `f` ska vara kolumnvektor. Därför skriver vi en sådan funktion:

```
function f=fdjur(t,y)
   f=[2*y(1)-0.01*y(1)*y(2)  -y(2)+0.01*y(1)*y(2)]';
```

Vi demonstrerar lösning både med Runge-Kuttas metod och `ode45` (tidssteget $h = 0.1$ i RK4 är lagom visar testkörningar). Vi undersöker om nio tidsenheter är tillräckligt för att se växelspelet mellan kaniner och rävar.

[1] Hämtat ur Kahaner, Moler, Nash: Num.Meth. and Software, uppgift P8-5




```
% Egen rungekutta RK4
  t=0; y=[300 150]; T=t; Y=y; tslut=9; h=0.1; n=90;
  for i=1:n
    f1=fkr(y); f2=fkr(y+h*f1/2); f3=fkr(y+h*f2/2); f4=fkr(y+h*f3);
    y=y+h/6*(f1+2*f2+2*f3+f4); t=t+h;  T=[T; t]; Y=[Y; y];
  end

% Alternativet med ode45
  [T,Y]=ode45(@fdjur,[0 9],[300 150]');
  kanin=Y(:,1); raev=Y(:,2);
  subplot(2,2,1), plot(T,kanin, T,raev,'--')
  subplot(2,2,2), plot(300,150,'x', kanin,raev), axis equal
```

Kaninantalet ökar inte mycket över startvärdet 300 (maxvärde 311) och minskar sedan snabbt då rävstammen växer. Som minst finns 16 kaniner ungefär vid $t = 2$, sedan ökar de till 311 igen efter cirka fem tidsenheter. Antalet rävar stiger från 150 till nära 470, men när tillgången på kaniner börjar bli dålig minskar rävantalet (minvärde 61) för att sedan öka igen. Vänstra figuren visar att det blir ett periodiskt förlopp med en periodtid på ungefär fem tidsenheter.

Högra figuren visar det så kallade *fasplanet* eller fasporträttet, som i ett periodiskt fall utgörs av en sluten kurva. Den kryssmärkta punkten har koordinaterna (300, 150) dvs $(k(0),\, r(0))$. Kurvan följer punkterna med koordinater $(k(t),\, r(t))$ — men i vilken riktning i detta fall?

8.4 Högre ordningens differentialekvationer

En andra ordningens differentialekvation $u'' = F(t, u, u')$ kan skrivas om till ett system av två första ordningens differentialekvationer genom substitutionen $y_1 = u,\ \ y_2 = u'$ som leder till systemet $y_1' = y_2,\ \ y_2' = F(t, y_1, y_2)$.
På vektorform blir det $\mathbf{y}' = \mathbf{f}(t,\, \mathbf{y})$ där $\mathbf{f} = (y_2,\ F(t, y_1, y_2))$.

En n-te ordningens differentialekvation, $u^{(n)} = f(t, u, u', \ldots, u^{(n-1)})$, kan på motsvarande sätt alltid skrivas som ett system av n stycken första ordningens differentialekvationer.
Som exempel tar vi följande tredje ordningens differentialekvation:

$$u''' = \sin(tu) + u''\sqrt{1 + (u')^2}, \quad u(0) = 0, \;\; u'(0) = 1, \;\; u''(0) = 0.$$

Inför $y_1 = u, \; y_2 = u', \; y_3 = u''$. Derivering leder till:

$$y_1' = y_2, \;\; y_2' = y_3, \;\; y_3' = \sin(ty_1) + y_3\sqrt{1 + y_2^2}.$$

De tre vänsterleden bildar komponenterna i vektorn $\mathbf{y}'$ och högerleden bildar vektorn $\mathbf{f} = (y_2, \; y_3, \; \sin(ty_1) + y_3\sqrt{1 + y_2^2})$, som i MATLAB skrivs:

```
function f=fordn3(t,y)
   f=[y(2)  y(3)  sin(t*y(1))+y(3)*sqrt(1+y(2)^2)]';
```

Startvektorn $\mathbf{y}(0)$ är lätt att ange, den är uppbyggd av startvärdena för u, u' och u'' och blir $\mathbf{y}(0) = (0, \, 1, \, 0)$. Vi väljer här `ode23` för att beräkna den numeriska lösningen för t-värden mellan 0 och 2. Kurvan över $u(t)$ ritas av `plot(T,Y(:,1))`.

```
ystart=[0 1 0]'; [T,Y]=ode23(@fordn3,[0 2],ystart); plot(T,Y(:,1))
```

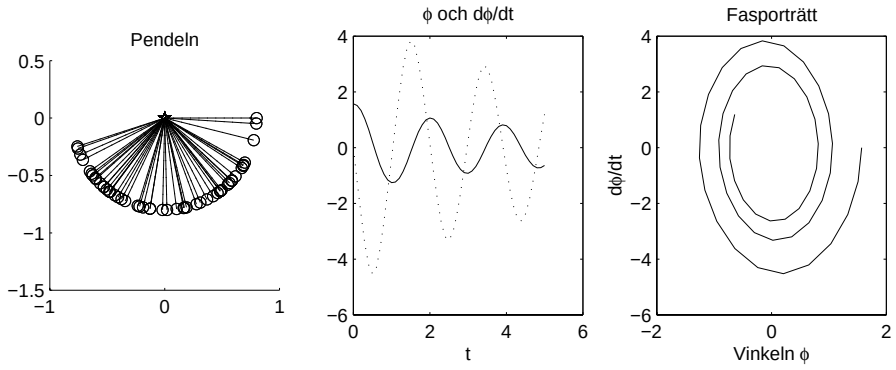
8.5 Pendel med stort utslag

En pendel med längden L släpps från horisontalläge, alltså med utslagsvinkeln nittio grader räknat från lodlinjen. Pendeln utför en plan dämpad svängningsrörelse. Differentialekvationen som beskriver pendelrörelsen lyder:

$$\frac{d^2\varphi}{dt^2} + c\,\frac{d\varphi}{dt}\left|\frac{d\varphi}{dt}\right| + \frac{g}{L}\sin\varphi = 0 \quad \text{med startvärdena } \varphi(0) = \pi/2, \;\; \varphi'(0) = 0.$$

Det är en autonom differentialekvation. Skriv om den genom substitutionen $y_1 = \varphi, \;\; y_2 = \varphi'$, alltså $y_1' = y_2, \;\; y_2' = -\frac{g}{L}\sin y_1 - c\, y_2|y_2|$. På vektorform blir det $\mathbf{y}' = \mathbf{f}(\mathbf{y})$ där $\mathbf{f} = (y_2, \; -\frac{g}{L}\sin y_1 - c\, y_2|y_2|)$.

Vi vill beräkna svängningsrörelsen för en pendel med $L = 0.8$ och dämpningskoefficienten $c = 0.1$. Tyngdaccelerationen sätts till $g = 9.81$. Här vill vi rita pendelläget varje tiondels sekund, det är alltså lämpligast med en numerisk metod som utnyttjar konstant tidssteg. Vi väljer Runge-Kuttas metod RK4 och skriver en funktion som definierar differentialekvationen:




```
function fp=fpend(y)
   global L c
   fp=[y(2)  -9.81/L*sin(y(1))-c*y(2)*abs(y(2))];
```

Programmet stegar framåt 0.1 sekund i taget i fem sekunder och ritar i varje steg upp ögonblicksbilden över pendeln i vänstra figuren (vid körningen suddas pendeln och ritas om igen). Exekvera `rkpendel` så kan du följa förloppet.

```
% rkpendel,  RK4 på en pendel
  clear, clf
  global L c
  L=0.8;  c=0.1;
  subplot(2,3,1), axis([-1 1 -1.5 0.5]), axis square, hold on
  plot([0 L],[0 0], L,0,'o'), title('Pendeln'), pause(0.1)
  set(gcf,'DoubleBuffer','on')
  tslut=5;   dt=0.1;
  t=0;  y=[pi/2 0];  Y=y;  T=t;
  xL=L;  yL=0;
  while t<tslut-dt/2
    plot(0,0,'p')
    f1=fpend(y);
    f2=fpend(y+dt*f1/2);
    f3=fpend(y+dt*f2/2);
    f4=fpend(y+dt*f3);
    y=y+dt*(f1+2*f2+2*f3+f4)/6;  t=t+dt;
    Y=[Y; y];   T=[T; t];
    plot([0 xL],[0 yL],'w', xL,yL,'wo')          % ritar över med vitt
    xL=L*sin(y(1));  yL=-L*cos(y(1));
    plot([0 xL],[0 yL], xL,yL,'ro'), pause(0.1)  % ritar ny pendel
  end, t, y
  fi=Y(:,1);  fiprim=Y(:,2);
  subplot(2,3,2), plot(T,fi, T,fiprim,'r:')
  title('\phi och d\phi/dt'), xlabel('t')
  subplot(2,3,3), plot(fi,fiprim), title('Fasporträtt')
  xlabel('Vinkeln \phi'), ylabel('d\phi/dt')
```

Fasplanets inåtgående spiral är typisk för en dämpad svängningsrörelse. En kontroll av tillförlitligheten i lösningen ges av jämförelsen mellan vinkelvärdena efter fem sekunders svängning med $dt=0.1$ och med $dt=0.05$. Vinklarna blir $\varphi = -0.6412$ respektive $\varphi = -0.6408$. tre decimaler verkar säkra.

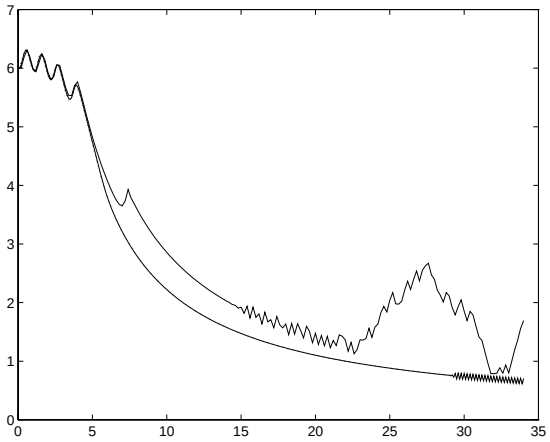
Exempelsamlingens **Ex 7.1–7.13** *är lämpliga övningar på numerisk behandling av ODE.*

8.6 Varnande exempel

Vi går tillbaka till exemplet i avsnitt 8.2: $dy/dt = \sin(ty)$ med begynnelsevärdet $y(0) = 6$. Då löstes problemet fram till $t = 2$ (se figurerna i avsnitt 8.2). Nu är vi nyfikna på lösningens utseende vid större t-värden. Låt oss välja slutpunkten $t_{slut} = 34$.

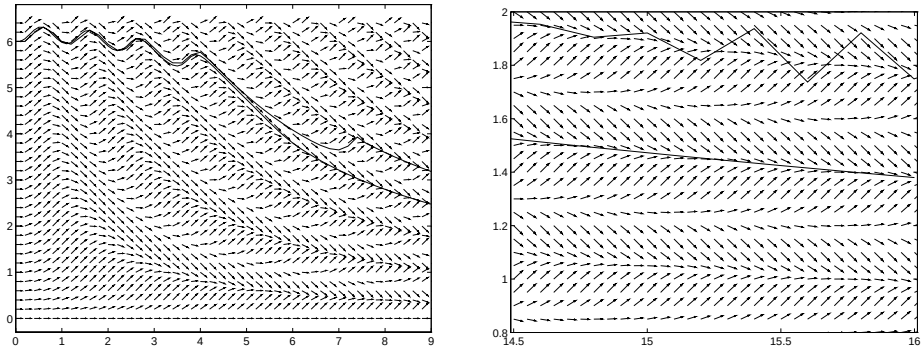
Vi prövar Eulers metod med intervallet delat i 170 steg, det innebär att steglängden blir $h = (t_{slut} - t_0)/170 = 34/170 = 0.2$. Kurvan ritas ut heldragen och i figuren avslöjas att det fina sinusformade beteendet som fanns fram till $t = 2$ försvinner ungefär vid $t = 5$. Därefter sjunker kurvan men får en intressant topp ungefär vid $t = 7$. Efter $t = 15$ uppstår små hackigheter och en kulle.

Ja, så ser alltså lösningen ut till differentialekvationen, och man vill gärna tro att det datorn ritar ut är sant. Men som redan erfarna diffekvationslösare vet vi att man alltid ska kontrollera resultatet, antingen med samma metod och halverad steglängd eller med en annan tillförlitlig metod. Med Eulers metod och steglängden 0.1 blir kurvan ganska annorlunda.



Det är fortfarande sant att sinusbeteendet slutar vid $t = 5$, men den fina toppen i trakten av $t = 7$ har nu försvunnit. Hackigheterna och kullen har också försvunnit. Kurvan är slät och fin ända fram till $t = 28$ men här börjar en tendens till oscillationer.

En inzoomning av förloppet fram till $t = 9$ visas i vänstra figuren nedan. Nu är också riktningsfältet inritat, bestående av riktningarna $\sin(ty)$ i varje punkt $(t,\ y)$. Den sanna diffekvationslösningen ska surfa på vågtoppen så som den undre kurvan gör. En liten störning orsakar att man halkar av vågtoppen och hamnar vid sidan av, vilket vi råkar ut för i övre kurvan som stegas fram med lite för stort steg.



I högra bilden har det sicksackiga partiet mellan t-värdena 14.5 och 16 zoomats in. Nu kan vi tydligt se att det steg som vi använt är ett jättekliv jämfört med de snabba variationerna som riktningsfältet visar upp.

Oscillationsfenomenet kallas *instabilitet*. Eulers metod är för vissa differentialekvationer en instabil metod om steglängden h är för stor. Om vi i exemplet väljer Eulers metod med betydligt mindre steglängd (eller löser med ode45 som har automatisk steglängdsreglering) blir kurvan slät ända fram till $t_{slut} = 34$.

8.6.1 Explicita och implicita metoder, instabilitet

Man skiljer på explicita och implicita metoder för numerisk lösning av ODE. Hittills har vi enbart sysslat med explicita metoder — Eulers metod, RK2 och RK4 — alla med egenskapen att värdet y_{i+1} vid nästa steg finns explicit i vänsterledet i beräkningsformeln.

Samtliga har tyvärr nackdelen att behöva orimligt liten steglängd i vissa typer av differentialekvationer (så kallade styva problem) för att inte instabilitet med kraftigt ökande oscillationer ska uppstå.

I sådana besvärliga ode-problem kan man i stället utnyttja en implicit

metod, till exempel *bakåteulermetoden* med formeln $y_{i+1} = y_i + h\, f(t_{i+1}, y_{i+1})$ eller *trapetsmetoden* med formeln $y_{i+1} = y_i + \frac{h}{2}\,(f(t_i, y_i) + f(t_{i+1}, y_{i+1}))$.

Som synes finns det sökta värdet y_{i+1} även i högerledet och ekvationslösning krävs för att räkna fram det. Implicita metoder blir därför krångligare ur beräkningssynpunkt, men både bakåteulermetoden och trapetsmetoden ger garanterat alltid stabilitet. Steglängden kan väljas lagom stor för att uppfylla noggrannhetskraven. Bakåteulermetoden har liksom Eulers metod noggrannhetsordning ett, felet är $O(h)$. Trapetsmetoden påminner mycket om RK2 och har noggrannhetsordningen två, felet är $O(h^2)$.

I fortsättningskurserna *Tillämpade numeriska metoder 1 och 2* (2D1220 TILNUM1 och 2D1250 TILNUM2) och i *Numerisk behandling av differentialekvationer 1 och 2* (2D1225, 2D1255) behandlas stabililitet/instabilitet, styva differentialekvationer och implicita metoder mer ingående.

Det bör dock framhållas att många differentialekvationer är av så snäll natur att vi kan tillämpa våra vanliga explicita metoder utan att det blir någon risk för obehagliga oscillationer.

8.7 Randvärdesproblem

All numerisk behandling av differentialekvationer har hittills gällt begynnelsevärdesproblem, som för en andra ordningens ODE lyder

$$y'' = f(t, y, y'), \quad y(a) = y_0, \;\; y'(a) = \gamma,$$

där $f(t, y, y')$ är ett känt uttryck och y_0 och γ har kända numeriska värden. Sökt är lösningen y för t-värden från a till något slutvärde t_{slut}.

För att en andra ordningens differentialekvation ska få entydig lösning behövs två villkor, men de behöver inte båda ligga vid starten. Om de finns i vardera änden av ett intervall $a \le t \le b$ blir problemet i stället ett *randvärdesproblem*:

$$y'' = f(t, y, y'), \quad y(a) = y_0, \;\; y(b) = y_{slut},$$

där y_0 och y_{slut} har givna numeriska värden. Sökt är lösningen y för t-värden mellan a och b. Randvillkoren kan även vara av typen känd startderivata $y'(a) = \gamma$ och känt slutvärde $y(b) = y_{slut}$ (eller till och med en given kombination $y(b) + \alpha y'(b) = \beta$).

8.7.1 Inskjutningsmetoden

Inskjutningsmetoden för randvärdesproblem innebär kortfattat att man tillfälligt gör problemet till ett begynnelsevärdesproblem genom att man så bra

som möjligt gissar startderivatan om den är okänd (alltså gör ett provskott) eller gissar y_0-värdet om startderivatan är given.

Så löser man differentialekvationen med bästa tänkbara metod t ex RK4 eller ode45, ser vad man får för slutvärde vid $t = b$; gissar nytt startvärde, löser differentialekvationen, ser på slutvärdet etc.

Sekantmetoden används för effektiv bestämning av startderivatan (respektive y_0-värdet), så att avvikelsen mellan det erhållna slutvärdet och det önskade slutvärdet y_{slut} ligger inom en i förväg bestämd felgräns.

8.7.2 Finitadifferensmetoden, FDM

Finitadifferensmetoden innebär ett helt annat angreppssätt för randvärdesproblemet och går till på följande sätt:
Dela intervallet $a \le t \le b$ i N delintervall med steget $h = (b-a)/N$.
Inför beteckningen t_i för de kända t-värdena $t_i = a + i \cdot h$ och y_i för de n okända y-värdena $y(t_i)$. Hur många okända finns det, vilket värde har n?
Svaret är $n = N-1$ i ett randvärdesproblem med känt start- och slutvärde, däremot gäller $n = N$ om det ena randvillkoret råkar innehålla ett derivatavillkor (för då är ju y-värdet okänt i den randpunkten).

Man approximerar differentialekvationens första- och andraderivata med differenskvoterna (2) och (4):

$$y_i' \approx \frac{y_{i+1} - y_{i-1}}{2h}, \quad y_i'' \approx \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2},$$

båda med ett trunkeringsfel $O(h^2)$.

Insättning i uttrycket $y_i'' = f(t_i, y_i, y_i')$ för $i = 1, 2, ..., n$, ger en uppsättning av n differensekvationer som leder till ett ekvationssystem med tridiagonal bandmatris[2] för bestämning av de obekanta y_i-värdena. Hur det går till ska åskådliggöras i några exempel.

8.7.3 Linjärt randvärdesproblem med en parameter

Bestäm lösningen $y(x)$ till randvärdesproblemet

$$y'' + a\sqrt{x}\, y' = (6 - y)x, \quad y(0) = 1, \;\; y(4) = -1, \;\; 0 \le x \le 4,$$

för a-värdena $-3, -2, -1,\ 0,\ 1,\ 2$.
(Vi låter x vara oberoende variabel i stället för t, för det känns naturligare med rumsberoende än tidsberoende i detta problem.)

[2] Finitadifferensmetoden kallades förr bandmatrismetoden.

Dela intervallet $0 \le x \le 4$ i N delintervall. Approximera derivatorna $y'(x_i)$ och $y''(x_i)$ med

$$y_i' \approx \frac{y_{i+1} - y_{i-1}}{2h}, \quad y_i'' \approx \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2},$$

och sätt in i differentialekvationsuttrycket $y_i'' + a\sqrt{x_i}\, y_i' = (6 - y_i)x_i$:

$$\frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} + a\sqrt{x_i}\,\frac{y_{i+1} - y_{i-1}}{2h} = (6 - y_i)x_i \quad \text{för } i = 1,\, 2,\, ...,\, n.$$

Steglängden är $h = 4/N$ och antalet obekanta är $n = N - 1$ eftersom randvärdena är bekanta. Vi multiplicerar med h^2 och sorterar om termerna och får nu

$$(1 - \frac{ha\sqrt{x_i}}{2})y_{i-1} - (2 - h^2 x_i)y_i + (1 + \frac{ha\sqrt{x_i}}{2})y_{i+1} = 6h^2 x_i.$$

För $i = 1$ gäller $y_{i-1} = y_0 = y(0) = 1$. Flytta första termen till högersidan:

$$-(2 - h^2 x_1)y_1 + (1 + \frac{ha\sqrt{x_1}}{2})y_2 = 6h^2 x_1 - (1 - \frac{ha\sqrt{x_1}}{2})$$

För $i = n$ gäller $y_{n+1} = y(4) = -1$, Flytta denna term till högersidan:

$$(1 - \frac{ha\sqrt{x_n}}{2})y_{n-1} - (2 - h^2 x_n)y_n = 6h^2 x_n - (1 + \frac{ha\sqrt{x_n}}{2})(-1).$$

För övriga i-värden, alltså $i = 2,\, ,3, ..., n$–1, ingår tre obekanta, $y_{i-1},\, y_i,\, y_{i+1}$, i vänsterledet. Allt kan ställas samman i ett ekvationssystem för de obekanta $y_1,\, y_2,\, ...,\, y_n$ med en tridiagonal systemmatris:

$$\begin{pmatrix}
-(2-h^2x_1) & 1+\frac{ha}{2}\sqrt{x_1} & 0 & 0 & \cdots & 0 \\
1-\frac{ha}{2}\sqrt{x_2} & -(2-h^2x_2) & 1+\frac{ha}{2}\sqrt{x_2} & 0 & \cdots & 0 \\
0 & 1-\frac{ha}{2}\sqrt{x_3} & -(2-h^2x_3) & & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & 0 & 1-\frac{ha}{2}\sqrt{x_{n-1}} & -(2-h^2x_{n-1}) & 1+\frac{ha}{2}\sqrt{x_{n-1}} \\
0 & 0 & 0 & 0 & 1-\frac{ha}{2}\sqrt{x_n} & -(2-h^2x_n)
\end{pmatrix}$$

och högerledet:

$$\begin{pmatrix}
6h^2x_1 - (1-\frac{ha}{2}\sqrt{x_1}) \\
6h^2x_2 \\
6h^2x_3 \\
\vdots \\
6h^2x_{n-1} \\
6h^2x_n - (1+\frac{ha}{2}\sqrt{x_n})(-1)
\end{pmatrix}$$

Vid lösning av det glesa systemet bör man utnyttja den tridiagonala strukturen och lösa systemet med vår egen `tridia` (se avsnitt 1.3) eller med MATLABs `sparse`-finesser.

Låt oss pröva en indelning av intervallet i 100 delintervall, alltså $N = 100$, $h = 4/N = 0.04$. Beräkna och rita lösningskurvorna för de sex a-värdena från -3 till 2. Man bör testa tillförlitligheten genom att räkna om allt med dubbla antalet delintervall och studera överensstämmelsen i de erhållna resultaten. (I algoritmen nedan noteras lösningens min- och maxvärde, som kan kontrolleras mot motsvarande värden när N fördubblats till 200.)

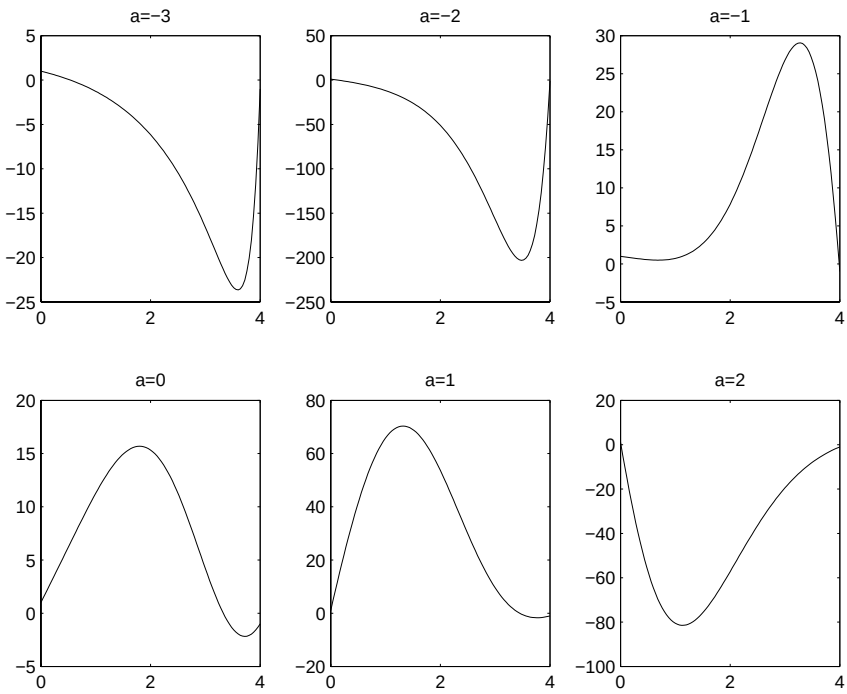
```
% randvs, FDM på y"+ay'sqrt(x)=(6-y)x;  y(0)=1; y(4)=-1
% med tridia (eller spdiags)
  clear, clf
  y0=1; yslut=-1;
  N=100;
  h=4/N;  n=N-1; x=h*(1:n)'; X=[0;x;4];
  dia=-(2-h^2*x);
  disp('     a       ymin       ymax ')
  for a=-3:2
    sup=1+h*a/2*sqrt(x(1:n-1));
    sub=1-h*a/2*sqrt(x(2:n));
    b=6*h^2*x;
    b(1)=b(1)-(1-h*a/2*sqrt(x(1)))*y0;
    b(n)=b(n)-(1+h*a/2*sqrt(x(n)))*yslut;
    y=tridia(dia,sup,sub,b);
    Y=[y0; y; yslut];
    subplot(2,3,a+4), plot(X,Y), title(['a=' num2str(a)])
    disp([a  min(y)  max(y)])
  end
```

Programkoden behöver ändras på några ställen om glesmatrisfinesser ska utnyttjas. Byt raderna för `dia, sup, sub` mot

```
  A=sparse(n);                          % initierar kompaktlagring
  for i=1:n, A(i,i)=-(2-h^2*x(i)); end  % diagonalelementen
  for i=1:n-1
    A(i,i+1)=1+h*a/2*sqrt(x(i));        % superdiagonalelementen
    A(i+1,i)=1-h*a/2*sqrt(x(i+1));      % subdiagonalelementen
  end
```

Ersätt tridia-anropet med `y=A\b` som löser det glesa systemet effektivt.

När matrisen har bandstruktur finns alternativet att bygga upp den med `spdiags`. (Gör `help spdiags` för att förstå innebördena av parametrarna). Vektorerna `dia, sup, sub` behålls och den kompaktlagrade matrisen erhålls med: `A=spdiags([[sub; 0] dia [0; sup]], -1:1,n,n);`



Här är randvärdesproblemets lösningskurvor för de önskade a-värdena. Skalorna i y-led är som synes mycket olika. Kurvan för $a = -3$ och $a = -2$ har en djup minimipunkt, medan den för a mellan -1 och 1 har vänt till ett maximum, sedan vänder den igen. (Kritiska värden för a är i närheten av -1.9 och 1.5.)

Titta närmare på kurvan för fallet $a = 0$, för den är av intresse som startapproximation i följande ickelinjära problem. Vi noterar att kurvan har en maxpunkt och att maxvärdet är ungefär 15.

Randvärdesproblem övas i **Ex 7.14–7.16** *i exempelsamlingen.*

8.7.4 Ickelinjärt randvärdesproblem med FDM

För en linjär differentialekvation som i ovanstående exempel leder finitadifferensmetoden till ett linjärt ekvationssystem. När vi har löst systemet är också hela randvärdesproblemet löst.

Finitadifferensmetoden kan tillämpas på ickelinjära randvärdesproblem också men nu uppkommer i stället ett ickelinjärt ekvationssystem för bestämning av de okända y_i-värdena.

Låt oss pröva FDM på det ickelinjära randvärdesproblemet

$$y'' + c\,y^2 = (6-y)x, \quad y(0) = 1, \;\; y(4) = -1, \;\; 0 \le x \le 4,$$

där parametern c har värdet 0.1.

Med samma intervallindelning som tidigare och med differensapproximationen (4) för $y''(x_i)$ erhålls:

$$\frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} + c\,y_i^2 = (6 - y_i)x_i.$$

Vi multiplicerar med h^2 och samlar alla y_i-termer i $G(y_i)$:

$$y_{i-1} + G(y_i) + y_{i+1} - 6h^2 x_i = 0 \;\text{ där }\; G(y_i) = -(2 - h^2 x_i)y_i + ch^2 y_i^2.$$

Derivatan av G behövs nedan, den är $G'(y_i) = -(2 - h^2 x_i) + 2ch^2 y_i$.

På grund av differentialekvationens y^2-term som i FDM-hanteringen hamnat i uttrycket för G, bildar sambanden för de obekanta y_1, y_2, ..., y_n ett ickelinjärt ekvationssystem:

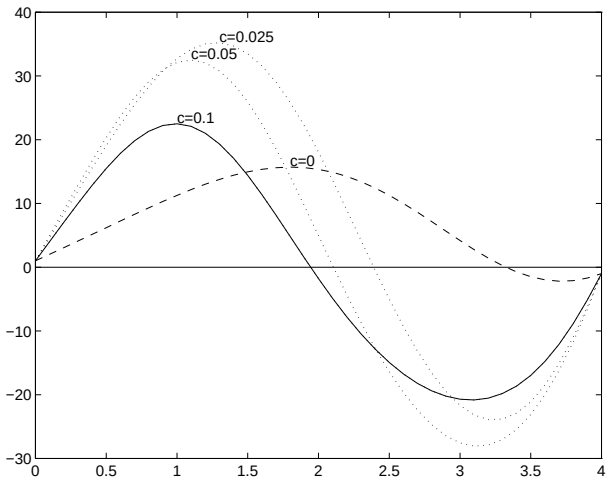
$$\left.\begin{array}{rcl} 1+ & G(y_1) + y_2 - 6h^2 x_1 & = 0 \\ y_1+ & G(y_2) + y_3 - 6h^2 x_2 & = 0 \\ \cdots & \cdots & .. \\ y_{n-2}+ & G(y_{n-1}) + y_n - 6h^2 x_{n-1} & = 0 \\ y_{n-1}+ & G(y_n) - 1 - 6h^2 x_n & = 0 \end{array}\right\}$$

Vi löser det ickelinjära systemet med Newtons metod (algoritm i avsnitt 6.9.1) och behöver bilda jacobianen som blir en tridiagonal matris:

$$\mathbf{J} = \begin{pmatrix} G'(y_1) & 1 & 0 & \cdots & 0 \\ 1 & G'(y_2) & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 1 & G'(y_{n-1}) & 1 \\ 0 & 0 & 0 & 1 & G'(y_n) \end{pmatrix}$$

Startvektorn $\mathbf{y}$ i Newtons metod måste vara en god gissning till lösningsvektorn för att konvergens ska erhållas.

Hur åstadkommer vi en god startapproximation? Jo, lösningen för fallet $c = 0$ vet vi redan, den erhölls i förra exemplet med parametern $a = 0$. Den ger förresten också en upplysning om det ickelinjära bidragets storlek i den aktuella differentialekvationen:



Eftersom $y_{max} \approx 15$ kan den ickelinjära termen cy^2 uppskattas till $0.1 \cdot 15^2 \approx 20$, inte så litet!

Algoritmen börjar med att det linjära problemet för $c = 0$ löses, så att vi har tillgång till en startapproximation för fallet med c skilt från noll. Dessvärre visar ett körningsförsök att den streckade startgissningen inte alls duger för att finna lösningen till problemet då c=0.1; det blir ingen konvergens!

Men om vi gör experimentet med ett c som är fjärdedelen så stort, dvs $c = 0.025$ i stället för 0.1, så visar det sig att Newtons metod med vår startgissning konvergerar. Alltid något, använd då lösningen för $c = 0.025$ som startapproximation för att komma vidare till lösningen av randvärdesproblemet för ett något större c-värde.

Här måste man ofta arbeta interaktivt, alltså med experiment avgöra hur stor ökning av c som tillåts utan att metoden spårar ur. I vårt exempel visar det sig bli tre etapper (c-värdena 0.025, 0.05, 0.1) för att nå fram till den slutliga lösningskurvan för $c = 0.1$ (heldragen i figuren).

```
% randolinj, FDM på y"+c*y^2=(6-y)x;  y(0)=1; y(4)=-1, fallet c=0.1
  clear, clf
  y0=1;  yslut=-1;
  plot([0 4],[0 0]), hold on
  N=100;  h=4/N;  n=N-1;
  x=h*(1:n)';

% Lös det linjära problemet (c=0)
% Tridiagonalt system, lös med sparsematrishantering
  A=sparse(n);
  for i=1:n, A(i,i)=-(2-h^2*x(i)); end
```

```
  for i=1:n-1, A(i,i+1)=1; A(i+1,i)=1; end
  b=6*h^2*x;
  b(1)=b(1)-y0;  b(n)=b(n)-yslut;
  y=A\b;
  X=[0; x; 4];
  Y=[y0; y; yslut]; plot(X,Y,'--')

% Arbeta stegvis med ökat c
% Ickelinjärt system, lös med Newtons metod
  J=sparse(n);                                % initiera sparsematris J
  for i=1:n-1, J(i,i+1)=1; J(i+1,i)=1; end   % element i J som inte ändras

% Tre etapper behövs för att nå slutlig lösning
  cvec=[0.025 0.05 0.1];                      % experimentellt erhållna
  for nr=1:3
    c=cvec(nr), dynorm=1;  iter=0;
    while dynorm>1e-5 & iter<15
      G=-(2-h^2*x).*y+c*h^2*y.^2;
      Gprim=-(2-h^2*x)+2*c*h^2*y;
      F=[y0; y(1:n-1)]+G+[y(2:n); yslut]-6*h^2*x;
      for i=1:n, J(i,i)=Gprim(i); end        % diagonalen i J
      dy=-J\F;
      dynorm=norm(dy,inf)
      y=y+dy; iter=iter+1;
    end
    if iter==15, disp('Ingen konvergens'), break, end
    Y=[y0; y; yslut]; plot(X,Y,':')
  end
  plot(X,Y)                                   % rita slutkurvan heldragen
```

Sakregister

överbestämda system, 14, 85

absolutfel, 2
adaptiv metod, 60
anpassande sinuskurva, 84
area, 50, 63, 69
Asplunds metod, 10
autonom differentialekvation, 99
avrundningsfel, 2

B-splines, 53
    icke-interpolerande, 53
bézierkurvor, 46
    kubiska, 47
      integration av, 63
    kvadratiska, 46, 50
    styckvisa kubiska, 50
bakåtsubstitution, 8
bandmatris, 9
bandmatrismetoden, 106
basfunktioner, 16
begynnelsevärdesproblem, 95
bilinjär interpolation, 40

centraldifferenskvot, 5, 44, 106
centrering, 18

dblquad, 70
diagonaltung matris, 8, 78
differenskvot, 4
    för andraderivatan, 6, 106
differentialekvation
    autonom, 99
differentialekvationer, 95
differentialekvationssystem, 99
dubbelintegraler, 69
dubbelrot, 72, 76

ekvationslösning, 72
ekvationssystem
    överbestämda linjära, 14, 85
    överbestämt olinjärt, 85
    ickelinjära, 79
      Newtons metod för, 81
    linjära, 6
euklidisk norm, 12, 86
Euler-Maclaurins formel, 61
Eulers metod, 95, 103
experimentell felskattning vid integration, 60
experimentell störningsräkning, 3, 12, 21, 37
explicit metod, 104

fasplan, 100
fasporträtt, 100
felfortplantning, 2
felfortplantningsformeln, 3
felgräns, 2
felkällor, 2
felkvadratsumma, 14, 22, 86
fibonaccisökning, 91
finitadifferensmetoden, 106
fixpunktsiteration, 77
framåtdifferenskvot, 4, 44
framåtsubstitution, 8, 24
fusksplines, 29, 33, 42

g7k15, 66
Gauss-Kronrods metod, 65
Gauss-Newtons metod, 86
Gauss-Seidels metod, 78
gausselimination, 6
gausskvadratur, 64
    härledning av parametrarna, 71

gles matris, 8, 78, 108
global parameter, 52
globalt fel, 96
gradient, 92
gyllenesnittetsökning, 89

hermiteinterpolation, 28
hermiteinterpolationspolynom
    integration av, 62
Heuns metod, 97
Horners algoritm, 26, 36, 39

ickelinjära ekvationssystem, 79, 109
illakonditionering, 13, 27, 34
implicit metod, 104
inline, 60
inskjutningsmetoden, 105
instabilitet, 104
integral
    över oändligt intervall, 68
    med singularitet, 68
integration, 57
interpolation, 24
    bikubisk, 42
    bilinjär, 40
    flerdimensionell, 40
    kvadratisk, 24
    linjär, 24
    styckvis, 28
interpolationspolynom, 24
interpolerande sinuskurva, 82
intervallhalveringsmetoden, 72
inversmatris, 10

jacobianmatris, 81, 110
    approximation till, 84
Jacobis metod, 78

kancellation, 3, 36
konditionstal, 3, 13, 18, 27
konvergens
    kvadratisk, 75, 81
    linjär, 75, 77
korrekta siffror, 2
kvadratisk interpolation, 24
kvadratisk konvergens, 75, 82

lagrangeinterpolation, 25
legendrepolynom, 71
linjär konvergens, 75, 77
linjär modellanpassning, 16
lokal parameter, 52
lokal variabel, 28, 52, 53
lokalt fel, 96

matris
    diagonaltung, 8, 78
    gles, 8, 78
    triangulär, 8, 27
    tridiagonal, 9, 31, 107
matrisnorm, 12
maximering, 89
maxnorm, 12
minimering, 89
minstakvadratlösning, 15, 85
minstakvadratmetoden, 14, 15, 85
modellanpassning, 16
    ickelinjär, 84
modellfunktion, 16

naiva formen, 24
naturlig splinekurva, 31
naturliga splines, 31
Newton-Cotes integrationsformler, 57
Newton-Raphsons metod, 74
Newtons ansats, 24
Newtons interpolationsformel, 24
Newtons metod för ickelinjära ekvationssystem, 81

noggrannhetsordning, 105
norm, 12
normalekvationer, 15
numerisk integration, 57

oändligt integrationsintervall, 68
ode23 och ode45, 98
odeset, 98

parameterbestämning, 16
parameterkurva, 46, 53
periodisk integrand, 61
picarditeration, 80
polynomekvation, 72
polynommultiplikation, 39
praktisk statistik, 22

quad och quadl, 60

randvärdesproblem, 105
relativfel, 2
residualanalys, 19
residualkurva, 19
richardsonextrapolation, 44, 59, 96
    upprepad, 45
riktningsfält, 104
RK2 (Heuns metod), 97
RK4, 97
Rombergs metod, 59
Runge-Kutta-metoder, 97
Runges fenomen, 26, 28
rutschbaneproblemet, 24

sekantmetoden, 73
Simpsons formel, 59
simulerade mätfel, 21
sparse, 9, 78, 108
spdiags, 108
splines
    MATLAB-, 34
    integration av, 62
    naturliga, 31
    parametriska kubiska, 53
spy, 78
stabilitet, 105
steepestdescentmetoden, 94
stegmetod, 45, 95
styrpunkt, 47
styrpunktsavstånd, 48

taylorutveckling, 5, 76, 95
trapetsregeln, 57, 61
triangulär matris, 8, 27
tridiagonal matris, 9, 31, 107
trunkeringsfel, 2, 6, 44, 58, 60, 96, 106

unimodal, 89

vandermondematris, 24, 27, 35
volym, 70